Høgskulen på Vestlandet

BACHELOROPPGAVE

Implementasjon av microfrontend-komponenter for web- og mobilapplikasjoner

Implementation of microfrontend components for web and mobile applications

Sølve Steinar Haugen, Thomas Brekke Fyllingen og Teodor Alveberg

Bacheloroppgave data/informasjonsteknologi

Fakultet for ingeniør- og naturvitskap

Institutt for datateknologi, elektroteknologi og realfag

Veileder: Remy Andre Monsen

Innleveringsdato: 03.06.2021

Vi bekrefter at arbeidet er selvstendig utarbeidet, og at referanser/kildehenvisninger til alle kilder som er brukt i arbeidet er oppgitt, *jf. Forskrift om studium og eksamen ved Høgskulen på Vestlandet, § 12-1.*

Høgskulen på Vestlandet

Fakultet for ingeniør- og naturvitskap

Institutt for datateknologi, elektroteknologi og realfag

TITTELSIDE FOR HOVEDPROSJEKT

Rapportens tittel: Implementasjon av microfrontend-komponenter for web- og mobilapplikasjoner	*Dato:* *03.06.2021*
Forfatter(e): Sølve Steinar Haugen, Teodor Alveberg, Thomas Brekke Fyllingen	*Antall sider u/vedlegg: 40* *Antall sider vedlegg: 18*
Studieretning: Dataingeniør/Informasjonsteknologi	*Antall disketter/CD-er:* 0
Kontaktperson ved studieretning: Remy Andre Monsen	*Gradering:* Ingen
Merknader: Ingen	

Oppdragsgiver: DNB	*Oppdragsgivers referanse:*
Oppdragsgivers kontaktperson: Anders Finseråas Graneng, Adrian Kristoffer Borgund, Jens-Åge Ellingsrud	*Telefon:* +47 958 59 872

Sammendrag:

Prosjektet omhandler implementering av microfrontend komponenter, og det blir drøftet hvordan komponenten kan virke som en PWA og i web- og mobilapplikasjoner. Oppgaven tar for seg teorien om dette konseptet, og diskuterer et valgt rammeverk. Rammeverket valgt for løsning av denne oppgaven er single-SPA. Løsningen i denne oppgaven vil derfor basere seg på dette rammeverket.

The project addresses the implementation of microfrontend components, and it is discussed how the component can work as a PWA and in web and mobile applications. The thesis addresses the theory of this concept and discusses a chosen framework. The chosen framework for solving this task is single-SPA. The solution discussed in this task will therefore be based on this framework.

Stikkord:

Microfrontend	Web- og mobilapplikasjoner	Single-SPA

Høgskulen på Vestlandet, Fakultet for ingeniør- og naturvitskap

Postadresse: Postboks 7030, 5020 BERGEN Besøksadresse: Inndalsveien 28, Bergen

Tlf. 55 58 75 00 Fax 55 58 77 90 E-post: post@hvl.no Hjemmeside: http://www.hvl.no

FORORD

Rapporten forklarer prosessen og løsningene på problemstillingen «implementasjon av gjenbrukbare microfrontend-komponenter i web- og mobilapplikasjoner». Prosjektet er utført av Thomas Brekke Fyllingen, Sølve Steinar Haugen og Teodor Alveberg, i samarbeid med DNB, representert av Jens-Åge Ellingsund, Adrian Kristoffer Borgund og Anders Finserås Graneng. Vi vil gjerne takke DNB og deres representanter for veiledning og tilrettelegging for best mulig løsning av oppgaven. Videre vil vi takke veileder ved HVL, Remy Andre Monsen, for god oppfølging og gode tilbakemeldinger underveis i prosessen.

Korona-pandemien har gitt en ekstra utfordring underveis i prosjektet, da deler av både undervisning, gruppesamarbeid og møter med oppdragsgiver og veileder har foregått digitalt.

INNHOLDSFORTEGNELSE

FORORD II

1 INNLEDNING 1

1.1 MOTIVASJON OG MÅL 1

1.2 KONTEKST 1

1.3 RESSURSER 3

1.3.1 Tekniske veiledere 3

1.3.2 Akademisk veileder 3

1.4 OPPBYGGING AV RAPPORTEN 3

2 PROSJEKTBESKRIVELSE 5

2.1 PRAKTISK BAKGRUNN 5

2.1.1 Prosjekteier 5

2.1.2 Tidligere arbeid 5

2.1.3 Initielle krav 6

2.1.4 Initiell løsningsidé 6

2.1.4.1 Milepæl 1 delmål 6

2.1.4.2 Milepæl 2 delmål 7

2.1.4.3 Milepæl 3 delmål 7

2.1.4.4 Milepæl 4 delmål 7

2.2 LITTERATUR OM PROBLEMSTILLINGEN 8

3 DESIGN AV PROSJEKTET 9

3.1 FORSLAG TIL LØSNING 9

3.1.1 Alternativ løsning 1: Podium 9

3.1.2 Alternativ løsning 2: Single-SPA 9

3.1.3 Alternativ løsning 3: BIT 9

3.1.4 Diskusjon av alternativene 10

3.2 VALGT LØSNING 10

3.3 VALG AV VERKTØY OG PROGRAMMERINGSSPRÅK 10

3.3.1 Visual Studio Code 10

3.3.2 React 10

3.3.3 Eufemia 11

3.3.4 GitHub 11

3.3.5 Android Studio 11

3.3.6 Xcode 11

3.3.7 Amazon Web Service, Simple Storage Service 12

3.3.8 JavaScript Object Notation [JSON] 12

3.3.9 JSON Server 12

3.3.10 Travis CI 12

3.3.11 Lighthouse 13

3.4 PROSJEKTMETODIKK 13

3.4.1 Utviklingsmetodikk 13

3.4.2 Prosjektplan 15

3.4.2.1 Revidering og endringer i planer 15

3.4.2.2 Tidlig fase 16

3.4.3 Risikovurdering 16

3.5 EVALUERINGSPLAN 18

4 DETALJERT DESIGN 19

4.1 MILEPÆL 1: ENKEL WEBSIDE 19

4.1.1 Arkitektur 19

4.2 MILEPÆL 2: WEBSIDE SOM EN MICROFRONTEND 20

4.2.1 Arkitektur 20

4.2.2 Fremgangsmåte 20

4.2.3 Utfordringer 21

4.3 MILEPÆL 3: PWA SOM EN MICROFRONTEND 22

4.3.1 PWA 22

4.3.2 Arkitektur 23

4.3.3 Arbeid så langt 23

4.3.4 Utfordringer 26

4.4 MILEPÆL 4: MOBILAPPLIKASJON SOM EN MICROFRONTEND 26

4.4.1 Native-applikasjoner sammenlignet med hybrid applikasjoner 26

4.4.2 Fremgangsmåte 27

4.4.3 Arkitektur 27

4.4.4 Arbeid 27

4.4.4.1 Android 27

4.4.4.2 iOS 29

4.4.5 Utfordringer 29

4.4.6 Løsning 30

5 EVALUERING 31

5.1 EVALUERINGSMETODE 31

5.1.1 Heuristisk evaluering 31

5.1.1.1 Milepæl 1 31

5.1.1.2 Milepæl 2 31

5.1.1.3 Milepæl 3 32

5.1.1.4 Milepæl 4 32

5.1.2 Enhetstester 32

5.1.3 Ende-til-endetesting 33

5.1.4 Evaluering fra oppdragsgiver 33

5.2 EVALUERINGSRESULTAT 33

5.2.1 Heuristisk evaluering av milepæler 33

5.2.2 Evaluering fra oppdragsgiver 34

6 DISKUSJON 35

6.1 KONSEKVENSER AV VALGT LØSNING 35

6.2 UTFORDRINGER 35

6.3 KONSEKVENSER AV DE OPPNÅDDE RESULTATENE 36

6.3.1 Single-SPA 36

6.3.2 Direkte import 36

6.3.3 Arbeidsmetoder 36

6.4 FORBEDRINGER VED NY START 37

6.4.1 Anvendelse av teori 37

6.4.2 Delmål 37

7 KONKLUSJON OG VIDERE ARBEID 38

7.1 PROSJEKTETS MÅL 38

7.2 MÅLOPPNÅELSE 38

7.3 NYTTEVERDI 39

7.4 VIDERE ARBEID 39

7.4.1 Videre arbeid med single-SPA 39

7.4.2 Valg av nytt rammeverk 40

8 REFERANSER 41

9 APPENDIX 46

9.1 RISIKOLISTE 46

9.2 GANTT-SKJEMA 47

9.3 BRUKERMANUALER 48

9.3.1 Fremgangsmåte PWA 48

9.3.2 Fremgangsmåte Single-SPA 56

1 INNLEDNING

Det finnes mange ulike løsninger på et problem innenfor teknologien. Teknologien er i stadig utvikling og har kommet langt de siste 30 årene. Nye teknikker, ulike, smarte og forbedrede løsninger er et resultat av teknologiens utvikling. Istedenfor å holde seg til en løsning på et problem, vil det alltid utforskes om det finnes andre muligheter for å løse det samme problemet og se om produktet kan forbedres. Innenfor utvikling er det mange ulike måter å jobbe på og komme frem til en løsning av et produkt. Eksempler på dette kan være at to komponenter implementerer en løsning, en individuell komponent som har de samme funksjonalitetene eller kommer på en ny løsning som ikke er tatt i bruk. En eksisterende løsning som blir stadig mer populær, er microfrontend. Oppgaven utforsker microfrontend-konseptet og ser på muligheten for implementasjon av et microfrontend-rammeverk. En slik implementasjon gjør at forskjellige utviklingsteam kan ha ansvar for deler av en webside, uten å tenke på andre komponenter.

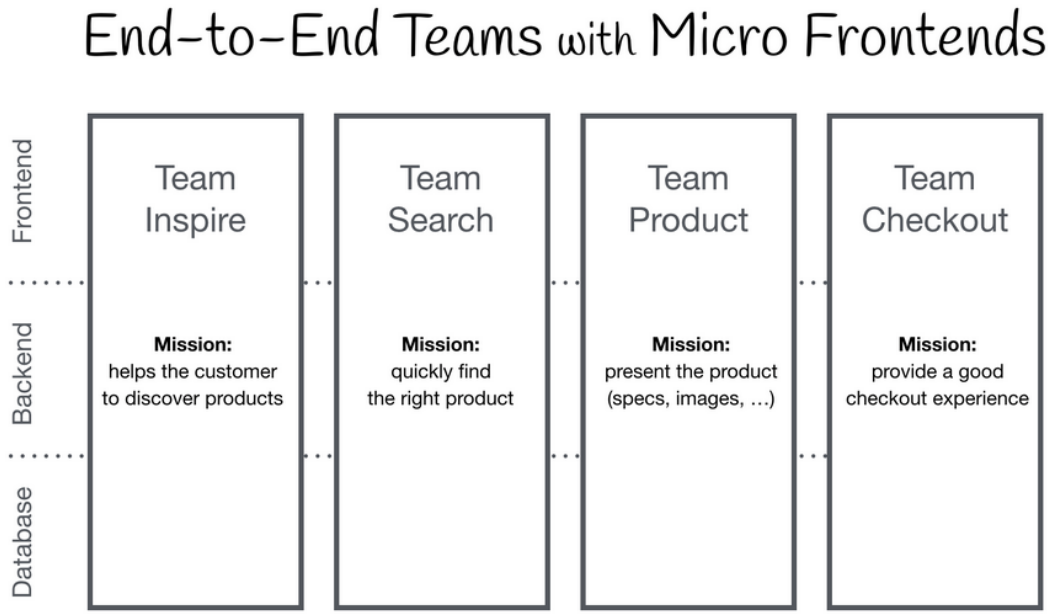
1.1 Motivasjon og mål

Store bedrifter ønsker å være best mulig til enhver tid. Særlig i teknologien skjer det store forandringer i løpet av få år. DNB er intet unntak, og ønsker å hevde seg i toppen. Dermed er det ønsket å utforske ulike løsninger på et problem. Prosjektets mål er å implementere og integrere en komponent som uavhengig av funksjon, kan gjenbrukes av utviklere, istedenfor å manuelt lage nye komponenter hver gang. Komponenten skal være responsiv, og brukerne av nettsider og applikasjon skal ha samme opplevelse uavhengig av plattform. Det skal være en microfrontend-komponent, som kan gjenbrukes av utviklere for web- og mobilapplikasjoner. Bruken av microfrontend, og fordelene med dette er utdypet i punkt 1.2. For å nå målet, er prosjektet delt inn i fire delmål. Disse vil bli presentert og drøftet i punkt 2.1.4.

1.2 Kontekst

Microfrontend har blitt en kjent teknikk de siste årene. Teknikken ble først tatt i bruk av selskapet ThoughtWork i 2016, og siden den gang er det flere selskaper som har tatt i bruk denne teknikken, for eksempel Spotify og DAZN. Microfrontend er en idé om at en webside er bygget opp av ulike, selvstendige komponenter. Hver komponent er utarbeidet av et team, og har kun

ansvar for komponentens funksjoner. En illustrasjon av hvordan microfrontend fungerer er vist i Figur 1 (Geers, 2021).



Figur 1: Illustrasjon av hvordan microfrontend fungerer (Geers, 2021)

Fordeler med en slik arkitektur er at komponenter på nettsiden er uavhengig av hverandre. Som vist på Figur 1 kan hvert team utvikle sin del av applikasjonen uten at andre komponenter påvirker deres arbeid. Dette gjør at vedlikehold blir lettere i en større applikasjon ved at frontend-siden blir delt inn i flere og mindre deler. I tillegg kan hvert team velge hvilken teknologi som skal brukes. Dette effektiviserer utviklingsprosessen ved at det blir brukt teknologi som hver enkelt er kjent med, og kan bruke dette for videre arbeid (Myers, 2020).

DNB ønsker å utforske mulighetene for implementering og integrering av web-baserte komponenter som er mulig å gjenbruke. Dette vil de gjøre ved å lage en microfrontend-komponent som er uavhengig av andre komponenter, og kan brukes til både mobil- og webapplikasjoner. DNB bruker dermed studenter for å utforske om en slik løsning er mulig. Rapporten legger til rette for innføring i microfrontend-prinsippet og DNB kan bygge videre på konklusjonen i denne rapporten.

Prosjektets oppgave er å implementere gjenbrukbare microfrontend-komponenter. Av den grunn vil prosjektets omfang være begrenset til enkeltkomponenter og ikke miljøet i sin helhet. Infrastrukturen finnes allerede og skal ikke endres for annet enn implementering av en eventuell løsning.

1.3 Ressurser

1.3.1 Tekniske veiledere

Oppdragsgiver stiller med to tekniske veiledere. Dette er en viktig ressurs med tanke på deres kompetanse, krav, støtte og spørsmål knyttet rundt produktet. Det er viktig å opprettholde en kontinuerlig dialog slik at produktet blir som ønsket. Dette blir gjort ved å holde ukentlige møter med oppdragsgiver. Hjelp fra de tekniske veilederne er en viktig ressurs underveis i prosjektets varighet, spesielt ved tilfeller der det oppstår problemer. Ved å benytte seg av de tekniske veiledernes kompetanse, vil dette sørge for at slike utfordringer løses raskere, og dermed fremgang i prosjektet.

1.3.2 Akademisk veileder

Veileder fra HVL hjelper til med rapportens innhold og oppbygging. Dette er en viktig ressurs som forsikrer at rapporten tilfredsstiller HVL's krav til akademisk skriving og struktur. Innholdet i rapporten og prosjektets fremgang bygger på det tekniske produktet. Det er viktig å opprettholde en god dialog for en god rapport. Veileder overvåker prosjektets fremgang via obligatoriske innleveringer, bloggoppdateringer og statusmøter. Utover dette svarer veileder på oppdukkende spørsmål via mail eller fysiske/digitale møter. Oppstår det problemer internt i gruppen eller angående oppgaven, skal veilederen kontaktes så raskt som mulig.

1.4 Oppbygging av rapporten

Kapittel 1 handler om motivasjon og mål for prosjektet, hva som er konteksten av oppgaven, hvilke avgrensninger som har blitt støtt på og de ulike ressurser som er brukt for å komme frem til en løsning.

Kapittel 2 beskriver prosjektet i mer detaljert form ved å ta for seg prosjektbeskrivelser. Dette er beskrivelser som omhandler produkteier, viktigheten av produktet, tidligere arbeid, initielle krav, løsningsidé og litteratur rundt problemstillingen.

Kapittel 3 handler om prosjektets design. Ulike løsninger er diskutert, der designet av prosjektet er basert på valgt løsning, og hva slags verktøy og programmeringsspråk som blir tatt i bruk. Videre i kapittelet blir prosjektmetodikk, evalueringsplan og risikohåndtering diskutert og drøftet.

Kapittel 4 tar for seg produktets design og de grunnleggende arkitekturene som er brukt for å produsere produktet, samt utfordringer i hver milepæl.

Kapittel 5 tar for seg hvilke evalueringsmetoder som er tatt i bruk, og evalueringsresultat som er kommet frem i prosjektets varighet.

Kapittel 6 diskuterer konsekvensene av valgte fremgangsmåter. Det drøftes hvordan disse fremgangsmåtene har hatt en innvirkning på resultatet, hvilke utfordringer som har dukket opp, og hvordan disse er håndtert. Videre presenteres det hva som kunne blitt gjort annerledes ved ny start på prosjektet.

Kapittel 7 beskriver prosjektets mål, og hvorvidt målene for de ulike milepælene er oppnådd eller ikke. I tillegg tar kapittelet for seg nytteverdien av arbeidet, og et forslag til hva som kan gjøres for fremtidig arbeid.

2 PROSJEKTBESKRIVELSE

2.1 Praktisk bakgrunn

DNB er Norges største finanskonsern som tilbyr ulike varianter av finansielle tjenester for bedrifts- og privatkunder (DNB, 2021a). Selskapet har et stort fokus på nyutvikling der teknologi er et sentralt tema. DNB ønsker derfor å se nærmere på konseptet om microfrontend som kan brukes på nett og i mobilapplikasjoner. Webapplikasjonen baserer seg på et rammeverk som implementerer microfrontend-konseptet, Single Single Page Application [single-SPA]. Informasjon om dette rammeverket finnes i punkt 2.2. Mobilapplikasjonen skal utforske muligheter for hvordan komponentene kan tas i bruk fra denne webapplikasjonen. Detaljert design for oppgaven er utdypet i punkt 4.

2.1.1 Prosjekteier

DNB forbeholder seg eierskap til sluttproduktet, samt mulighet til å benytte deler eller hele resultatet i sine kommersielle løsninger.

2.1.2 Tidligere arbeid

Store selskaper som har utnyttet microfrontend-prinsippet er selskaper som Spotify og DAZN. Teknikken endrer strukturen i hvordan selskapet leverer moderne brukeropplevelse, uten at brukeren merker noe forskjell. Microfrontend blir mer vanlig og bedriftsledere utnytter disse fordelene med microfrontend for å gi den beste kundeopplevelsen og opprettholde konkurransen (Entando, 2020).

DAZN er en strømmetjeneste for sport, og er mye brukt rundt om i verden. DAZN startet i Europa, men er tilgjengelig i ni andre land i verden (DAZN, u.å.). Luca Mezzalira er sjefsarkitekt for selskapet og en talsmann for microfrontend-prinsippet. Han har laget mange bøker der han forklarer hvordan denne teknikken implementeres og hvilke fordeler den gir (Entando, 2020). DAZN bruker bootstrap for å implementere microfrontend. Bootstrap setter kontekst for applikasjonen og laster inn riktig microfrontend-komponent (Mezzalira, 2019b).

Spotify er en populær musikkstrømmetjeneste som er kjent verden over. Selskapet bruker microfrontend for sine skrivebordapplikasjoner. Teknikken de tar i bruk er Iframes. Kommunikasjonen mellom Iframes skjer via en hendelsesadministrator (event bus) som frakobler

de ulike delene av applikasjonen. Dette gjør at komponentene kan kommunisere uten å vite hvem som skal høre på meldingen eller hendelsene (Mezzalira, 2019a).

DNB ønsker å utforske mulighetene ved implementasjon av microfrontend. Oppgaven gir frihet til valg av implementeringsmetode og valg av komponent. Hovedmålet med oppgaven er å utforske muligheten for at komponenten kan bli implementert som en gjenbrukbar microfrontend. Videre er hensikten at komponentene gir best mulig brukervennlighet, uten at dette går utover allerede implementerte funksjoner i systemet. DNB har ikke tidligere undersøkt muligheten for å ta i bruk microfrontend konseptet og har ingen interne løsninger oppgaven vil bygge på.

2.1.3 Initielle krav

Oppgavens krav bygges opp av milepæler fremlagt i oppgaveteksten. Disse kommer til å fungere som delmål underveis i utviklingen. Hver milepæl og dens delmål blir beskrevet i punkt 2.1.4. Milepælene er som følger:

1. En responsiv frontend-komponent som implementerer ønsket funksjonalitet på en enkel side.
2. En responsiv microfrontend bygget på milepæl 1.
3. En responsiv microfrontend implementert i/som en Progressive Web Application [PWA].
4. En responsiv microfrontend implementert i en webløsning og hentet i et webview iOS/Android.

2.1.4 Initiell løsningsidé

Basert på de ulike milepælene er det laget et løsningsforslag med delmål for hver milepæl. Ettersom milepælene bygger på hverandre, vil også løsningsforslagene gjøre det samme.

2.1.4.1 Milepæl 1 delmål

1. Oppsett av en enkel webside ved hjelp av React med grafisk fremstilling av statiske data.
2. Responsiv UI komponent.
3. En komponent som henter data fra en JSON server.

4. Oppdatere komponentens utseende i henhold til Eufemia.

Ved oppfylt milepæl 1, har brukeren mulighet til å gå inn på en webside som gir en grafisk fremstilling av data. Brukeren er i dette tilfellet en bankkunde, og dataen som blir vist skal simulere bankkundens saldo. Komponenten skal være responsiv, som vil si at den skalerer i forhold til webside-vinduet. Det finnes utallige enheter med forskjellige skjermstørrelser, og det er ønskelig å kunne tilpasse innholdet til disse. For eksempel vil en telefon vise en forenklet visning av innholdet sammenlignet med et nettbrett eller en PC-skjerm (Andrew & LePage, 2021). Komponenten henter JSON-data (3.3.8) fra en JSON server (3.3.9), som simulerer brukerens konto. Websiden og grafkomponenten er stylet ved hjelp av DNB's Eufemia (DNB, 2021b).

2.1.4.2 Milepæl 2 delmål

1. Det bygges videre på løsningsforslaget fra milepæl 1, men websiden og komponenten skal utvikles til å ta i bruk microfrontend-prinsipper.
2. Applikasjonen skal lastes opp i en skybasert løsning.

Ved oppfylt milepæl 2 vil websiden ta i bruk resultatet fra milepæl 1 og implementere dette som en microfrontend. Komponentene skal være uavhengige av hverandre og bli hentet inn på samme side. Videre skal det være mulig at andre skal ha tilgang til applikasjonen på nett.

2.1.4.3 Milepæl 3 delmål

1. Bygge opp siden som en PWA, der komponenten fra milepæl 1/2 gjenbrukes.

Ved oppfylt Milepæl 3 skal PWA prinsippet være implementert som en del av microfrontend-miljøet. PWA prinsippet går ut på at det blir laget en applikasjon som kan kjøre på web og mobil. Brukeropplevelsen skal være like god uavhengig av plattform.

2.1.4.4 Milepæl 4 delmål

1. En responsiv microfrontend implementeres i en webløsning og hentet i et webview i Android og iOS.
2. Android- og en iOS-applikasjon der det kan legge til komponenten fra tidligere milepæler.

Ved oppfylt milepæl 4 skal mobilopplevelsen til brukeren være tilfredsstilt ved at det skal være mulig å hente inn en Android- og en iOS-applikasjon som kjører som en mobilapplikasjon.

Brukeren av applikasjonen skal ha samme brukeropplevelse som på nett. Mobilapplikasjonen skal være implementert som en microfrontend.

Oppgaven ser nærmere om dette er mulig å gjøre på en god og brukervennlig måte. Brukervennligheten vil gå ut på at det er lett å ta i bruk, fungerer på web og mobil og enkelt å legge til nye komponenter. Brukerne vil bestå av utviklere som skal kunne gjenbruke koden på ulike prosjekter. I hovedsak vil det være viktig å dokumentere fremgangsmåten og lage dokumentasjon på konseptet. Dette vil være til hjelp for DNB om de ønsker å ta i bruk dette konseptet i fremtiden. For utdypende informasjon om dokumentasjon, se punkt 9.3.

2.2 Litteratur om problemstillingen

Problemstillingen baserer seg på følgende teknologier:

React: JavaScript-bibliotek for utvikling av brukergrensesnitt. React er ment for å lage websider som kan laste og forandre data uten å måtte oppdatere hele websiden (Pandit, 2021).

Progressive Web App: Type applikasjonsprogramvare for å minske forskjellene på brukeropplevelsen i nettbasert og native applikasjoner (LePage & Richard, 2020).

Microfrontend: Bygger på konseptet om «micro-services» der hvert team skal kunne ha ansvar for sin egen del av koden og fremstillingen på websiden. Ulike komponentene fungerer uavhengig av hverandre og vil ikke bli påvirket av andre komponenter (Geers, 2021).

Service-workers: Baserer seg på skript som kjører i nettleseren bakgrunn. Hovedfunksjonen til en service worker er at den kan ha kontroll på nettverksforespørsler. Det muliggjør for bruk uten internettilkobling (Gaunt, 2021).

Single-SPA: Single-SPA er et rammeverk som brukes for å bringe flere JavaScript-microfrontend-komponenter til en frontend-applikasjon. Arkitekturen åpner for bruk av flere rammeverk på samme webside, uavhengig utrulling av microfrontendene og implementasjon av kode i hvilket som helst rammeverk uten å måtte skrive eksisterende applikasjon på nytt (Single-SPA, u.å.b).

3 DESIGN AV PROSJEKTET

3.1 Forslag til løsning

Valg av microfrontend-rammeverk har vært den mest sentrale delen for videre utvikling av prosjektet. Det har blitt lagt inn mye tid i å prøve å finne ut hvilket rammeverk som kan bidra med mest mulig gjenbrukbarhet og enklest implementasjon til videre utvikling. Ettersom de ulike milepælene bygger på samme grafkomponent fra milepæl 1, vil det være ønskelig at rammeverket støttes både for PWA- og mobilapplikasjonsutvikling for å oppnå flest mulig suksesskriterier. Det finnes mange varianter av microfrontend-rammeverk og de alternative løsningene bygger på dette.

3.1.1 Alternativ løsning 1: Podium

Podium er et microfrontend-rammeverk som er utviklet av Finn.no. Dette rammeverket tar i bruk JavaScript, og har sin egen syntaks med Podlets som komponenter (Finn, 2021). Dette muliggjør microfrontend ettersom en nettside kan brytes ned til individuelle komponenter som eies av egne team. Fordeler med dette rammeverket er at det er utviklet og brukt av Finn, samtidig som det er godt dokumentert.

3.1.2 Alternativ løsning 2: Single-SPA

Single-SPA er et annet rammeverk for microfrontend. Dette rammeverket gjør det mulig å bruke ulike typer JavaScript-rammeverk som Ember, React og Angular. Single-SPA tillater bruk av forskjellige programmeringsspråk. Microfrontend-komponentene kan publiseres uavhengig av hverandre (Single-SPA, u.å.b). Deretter kan komponentene tas inn i en container som bygger selve nettsiden for enklest mulig vedlikehold av kode. Dette fører til at hvert team har stor frihet til å utvikle selvstendig og det kan jobbes i «vertikale søyler», som vist i Figur 1. Single-SPA-prinsippet er blitt tatt i bruk av et stort internasjonalt miljø, noe som gjør at det finnes god dokumentasjon på dette.

3.1.3 Alternativ løsning 3: BIT

En annen populær metode for microfrontend er BIT. BIT er en komponent-samarbeidsplattform som gjør at hver enkelt komponent kan importeres i prosjektet. Hver komponent, for eksempel

en knapp, kan være frittstående. BIT legger til semantiske lag som kartlegger filene. Dette gjør at det er stor mulighet for gjenbruk på tvers av prosjekter (BIT.dev, 2021).

3.1.4 Diskusjon av alternativene

De forskjellige alternative løsningene har sine fordeler og ulemper. Som nevnt tidligere er gjenbrukbarhet og enkel implementering i fokus. Ettersom microfrontend er et konsept som ikke har vært utforsket, er det viktig at det finnes god dokumentasjon på det som skal gjøres. Podium har ikke egne kanaler for diskusjon og spørsmål, i motsetning til BIT og single-SPA. BIT har mer fokus på hver enkelt komponent, der det blir laget en egen microfrontend for den gjeldende komponenten. Single-SPA har mer fokus på større logiske komponenter og hvordan helheten til siden blir satt opp.

3.2 Valgt løsning

Ved valg av løsning var det BIT og single-SPA som var de mest aktuelle. Grunnet god dokumentasjon og kommunikasjonsplattformer, gjorde at disse var mer interessant en Podium. Valget falt til slutt på single-SPA ettersom helheten til rammeverket virket mest lovende.

3.3 Valg av verktøy og programmeringsspråk

3.3.1 Visual Studio Code

Visual Studio Code er en kodeeditor som er tilgjengelig for både macOS, Windows og Linux. (Microsoft, 2021b). Kodeeditoren har innebygget støtte for JavaScript, et programmeringsspråk som står sentralt i prosjektet. Visual Studio Code er en kodeeditor som har blitt benyttet i tidligere prosjekter. Grunnlaget for valg av verktøy faller dermed på denne kodeeditoren. Verktøyet har blitt brukt gjennom hele den tekniske biten av prosjektet.

3.3.2 React

React er et populært JavaScript bibliotek utviklet av Facebook for å bygge web applikasjoner. Visual Studio Code støtter React.js IntelliSense og navigasjon for kode. IntelliSense er en generell betegnelse som blir brukt for å hjelpe utvikleren med koden. Dette kan være i form av automatisk kodeutfylling og info om funksjoner (Microsoft, 2021a). DNB bruker React i sine systemer, og på bakgrunn av dette blir det brukt React som programmeringsspråk istedenfor

Vue.js og Angular. Som en del av prosjektets oppstartsfase, ble det satt av tid til å gjøre seg kjent med biblioteket. Dette fordi React står sentralt i utviklingen av produktet, og dermed for prosjektets fremgang.

3.3.3 Eufemia

Eufemia er DNB's eget designsystem. Her finnes ressurser for å opprettholde DNB's standard design (DNB, 2021b). Eufemia har forhåndsdefinerte ressurser, og sparer tid med tanke på produktdesign. Siden det skal utvikles et produkt for DNB, vil det være naturlig å bruke Eufemia for å integrere komponenter fra DNB's designsystem. Designsystemet brukes hver gang det blir implementert for eksempel knapper, tekstbokser og menyer for å nevne noen.

3.3.4 GitHub

GitHub er en plattform som lar brukere oppbevare og dele kode via internett. Her er det ikke nødvendig å være fysisk til stede for å samarbeide (GitHub, 2021). Koden blir delt via internett slik at alle er oppdatert med den riktige versjonen av prosjektet. Bruken av GitHub er rask og enkel, og passer for mindre prosjekter. Grunnen til at valget falt på GitHub er at verktøyet er godt kjent fra før. Hver gang det er fremgang i produktet vil en oppdatert kode bli lastet opp til GitHub.

3.3.5 Android Studio

Android Studio er den offisielle IDE'en til Android. IDE'en skal sørge for raskere utvikling og hjelpe med å bygge et høykvalitetsprodukt for Android-enheter (Google Developers, 2021a). Android Studio tillater både utvikling, kompilering og testing i samme programvare, noe som gjør prosessen mer smidig. Verktøyet blir brukt ved siste milepæl i prosjektet, som tidligere er beskrevet i punkt 2.1.4.4.

3.3.6 Xcode

Xcode er et verktøy for utvikling av applikasjoner for Apple plattformer. Dette verktøyet brukes for å styre hele flyten i utviklingsprosessen i iOS – lage applikasjonen, testing, optimalisering og publisering av applikasjon på App Store (Apple Inc., 2021). Mobilapplikasjonen skal være tilgjengelig for både Android og iOS. Fordi Xcode er hovedplattformen for utvikling av iOS

applikasjoner, blir dette verktøyet brukt. Xcode blir brukt ved siste milepæl i prosjektet, som tidligere beskrevet i punkt 2.1.4.4.

3.3.7 Amazon Web Service, Simple Storage Service

Amazon Web Service [AWS] Simple Storage Service [S3] er en nettjeneste som brukes til å lagre og hente ut data via internett. Målet med denne tjenesten er å maksimere fordelene ved skalering for utviklere (Amazon, u.å.). Denne har blitt brukt til å lagre resultatet fra bygginger av applikasjonen i skyen. Dette muliggjør tilgang til nettsiden via en URL.

3.3.8 JavaScript Object Notation [JSON]

JSON er et dataformat som er i klartekst og er enkelt for mennesker og maskiner å lese. Fordi JSON bruker konvensjoner av blant annet C-familien, Java, JavaScript, Pearl, Python og mange flere, er JSON et ideelt språk for datautveksling (JSON, u.å.). Formatet inneholder en samling av objekter, som igjen inneholder ulike attributter med verdier. JSON muliggjør oversetting fra JavaScript til tekst, og fra tekst til JavaScript. Dette gjør at det kan jobbes med data som JavaScript-objekter, uten å måtte håndtere oversetting ved sending til server, eller henting fra nettleser (W3 Schools, u.å.).

3.3.9 JSON Server

En JSON server er en node-modul som kan brukes for å simulere et lignende rest-API. Dette blir brukt for henting av data fra et API (Kumar, 2018). En JSON server har forenklet arbeidet i prosjektet, i den forstand at det ikke har vært nødvendig å sette seg inn i DNB's systemer. Dette har blitt gjort siden en JSON server bekrefter om konseptet fungerer på lik måte. Frem til JSON filen ble lastet opp i AWS S3, ble serveren brukt.

3.3.10 Travis CI

Travis CI er en continious intergration [CI] plattform. CI er et verktøy for å automatisere integrasjonen av kodeendringer fra flere bidragsytere i et og samme prosjekt (Rehkopf, u.å). Denne plattformen støtter utviklingen ved å automatisk bygge og teste endringer i koden, og gir umiddelbar tilbakemelding på byggingen. Travis CI kan også automatisere andre deler av utviklingsprosessen ved å håndtere utrullinger og notifikasjoner (Travis CI, 2021). Travis CI har

blitt brukt til å laste opp endringer i koden til S3, og dermed håndtere selvstendighet blant microfrontend-komponentene, samt å oppdatere eksternt import-map.

3.3.11 Lighthouse

Lighthouse er et automatisk, åpent kildeverktøy for å forbedre kvaliteten på nettsider. Dette er en utvidelse for Google Chrome og verktøyet kan kjøres på hvilken som helst nettside. Lighthouse gir en rapport på ulike egenskaper ved en nettside, deriblant ytelse og PWA (Google Developers, 2021b). Ved utvikling av en PWA i milepæl 3 ble dette verktøyet brukt som en sjekkliste på om applikasjonen oppfylte kravene til en PWA. Verktøyet er enkelt å bruke, og gav konkrete svar på hva slags krav som stilles i en PWA.

3.4 Prosjektmetodikk

3.4.1 Utviklingsmetodikk

Det blir tatt i bruk tidligere kjente fremgangsmåter samt utviklingsmetodikk.

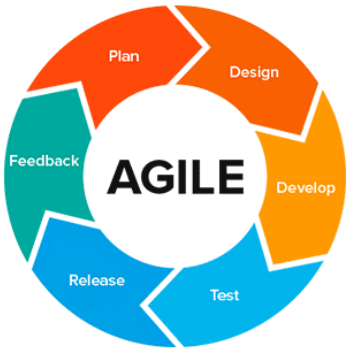
Generelle webutviklingsprinsipper: Ethical Web beskriver de fire prinsipper for utvikling av websider:

1. «Websiden skal fungere for alle»
2. «Websiden skal fungere overalt»
3. «Websiden skal respektere brukerens personvern og sikkerhet»
4. «Webutviklere skal ta hensyn til nestemann»

(Ethical Web, 2021)

Ved å ta i bruk disse prinsippene i løpet av utviklingsfasene forbedres utgangspunktet for videre utvikling.

Utviklingen vil ta i bruk en smidig fremgangsmåte. Denne type utviklingsmetodikk bygger på iterasjoner underveis i utviklingsprosessen, som vist i Figur 2. Ved å bryte ned større milepæler i mindre delmål, kan tester og evaluering skje med større hyppighet.

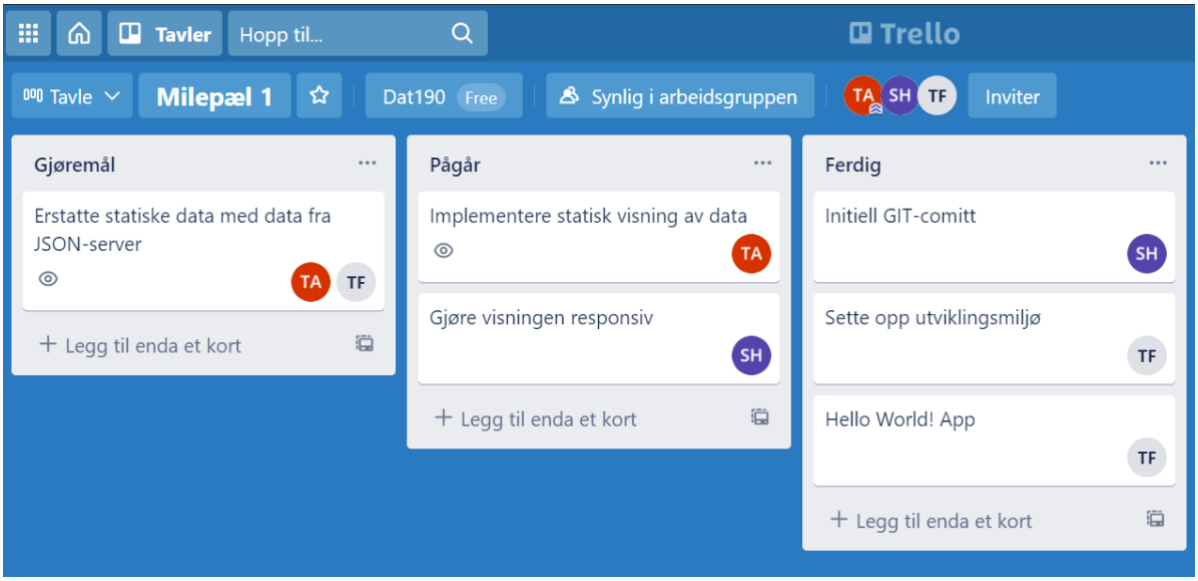


Figur 2: Smidig utviklingsmetodikk (Iddo, 2019)

Det vil være lettere å få tilbakemeldinger av oppdragsgiver ved å bruke denne type utviklingsmetodikk. Dette gjør at sluttproduktet stemmer overens med problemstillingen. (Atlassian, 2021).

Resultatene fra hver iterasjon vil vises frem til prosjekteier på en ukentlig basis. Hensikten med dette er å tidligst mulig å oppdage uoverensstemmelser mellom prosjektets resultater og fremgang, og eierens faktiske ønsker og krav, utover det som er spesifisert i milepælene. For denne metodikken vil milepælene stå som de sentrale utviklingsmålene, med under/delmål som til sammen dekker behovene beskrevet i milepælene.

Delmålene blir definert og planlagt i en tidlig fase, og blir typisk delegert i starten av en iterasjon. Selve delegeringen foregår ved bruk av Trello-tavle, der det settes opp oppgaver som skal gjøres, pågår og er ferdige. Dette er blitt gjort for å synliggjøre hvilke oppgaver som kan tas tak i ved ledig tid/fullførte oppgaver. Videre er dette med på å gi en god kort- og langsiktig oversikt for progresjonen i for eksempel en milepæl, se Figur 3.



Figur 3: Utklipp fra Trello-tavle (www.trello.com)

3.4.2 Prosjektplan

For planlegging og strukturering av prosjektet har det blitt brukt et GANTT-skjema (se kapittel 9.2). Skjemaet gir oversikt over langsiktige og kortsiktige mål og tidsfrister; Det planlegges langsiktig med store milepæler og obligatoriske oppgaver som utgangspunkt, for å så bryte disse ned i mindre, mer angripelige og konkrete arbeidsoppgaver.

Skjemaet er delt inn i to hoveddeler. Den ene delen er obligatoriske øvinger gitt av HVL i forbindelse med rapportskriving. Den andre delen omhandler tekniske mål, som er blitt til i løpet av forprosjektfasen, og fra milepæler gitt fra oppdragsgiver.

Som beskrevet innledningsvis i dette punktet, er smidig prosjektmetodikk blitt brukt for utvikling. GANTT-skjemaet har lagt rammene for iterasjonene og vært et viktig verktøy for planlegging og strukturering av prosjektet i sin helhet.

GANTT-skjemaet illustrerer et parallelt løp, der det arbeides med milepæler og rapportskriving. Dette gjøres fordi resultatene fra milepælene er det som drøftes og beskrives i sluttrapporten. Uten fremgang i utviklingen, vil det være liten eller ingen fremgang i utarbeidelsen av rapporten.

3.4.2.1 Revidering og endringer i planer

Revidering av milepælene ble gjort i en tidlig fase. Milepælene ble brutt ned i flere små delmål, for å konkretisere suksesskriterier og arbeidsoppgaver. Dette for å kunne fullføre de ulike milepælene i tråd med planlagte tidsfrister.

Tidlig i prosjektet, ble det brukt minimal pie chart (NPM, u.å.a), for grafisk fremstilling av dataene. Fordi dette biblioteket kun har kakediagram, ble det klart at dette biblioteket ikke dekket behovene for grafisk fremstilling. Derfor ble grafbiblioteket endret til chart.js (Chart.js, 2021), der det kan implementeres blant annet kake-, linje, og søylediagrammer. Det er også flere muligheter for modifisering, og implementasjonen av dette biblioteket gjør det lettere å tilpasse designet.

I prosjektets innledende fase ble det arbeidet mye samlet for å bygge en felles forståelse for nye konsepter og ny teknologi. Etter hvert som prosjektet hadde flere oppgaver som forholdt seg til spesifikke milepæler, ble det besluttet at arbeidet i større grad skulle foregå parallelt, men med ukentlige oppdateringer til resterende gruppemedlemmer. Dette ble gjort for å utnytte ressurser i større grad, og dermed også progresjonen.

Fordi prosjektet er todelt, der den ene delen handler om rapportskriving og den andre om å levere et produkt, har det blitt prioritert litt annerledes enn først anslått i GANTT-skjemaet. Obligatoriske frister for blant annet forprosjekt- og statusrapport, har ført til noe forskyvning av arbeidet med selve produktet, noe som har blitt tatt igjen i siste del av prosjektet. Endringene fra opprinnelig plan, er merket gult og kan sees i Tabell 2.

3.4.2.2 Tidlig fase

Fra januar til midten av mars, har det vært en periode der prosjektet har gått parallelt med andre fag. Her har det blitt fokusert på å sette seg inn i oppgaven, dens omfang og mer konkrete teknologier som skal bli brukt. Tiden har blitt brukt individuelt, med unntak av et ukentlig møte med oppdragsgiver.

3.4.3 Risikovurdering

Risikoplanen er utarbeidet ved å identifisere potensielle risikoer, og gradere hver risikos sannsynlighet og konsekvens på en skala fra en til fem. Risikolisten finnes i Tabell 1. Risikofaktoren (RF) er deretter regnet ut ved å multiplisere risikoens sannsynlighet (S) med konsekvensen (K). Bragelien, Skjølsvik og Voldsund beskriver gradering av sannsynlighet som følger:

1. *Det er ytterst usannsynlig at det vil inntreffe*
2. *Der er usannsynlig, men det er en liten sjanse for at det vil skje.*
3. *Det er mulig det kan inntreffe, men like mulig at det ikke gjør det*
4. *Det er sannsynlig at det vil inntreffe, og det er en liten sjanse for at det ikke gjør det.*
5. *Det er nesten sikkert at det vil inntreffe*

(Bragelien, Skjølsvik & Voldsund, 2020, s. 315).

Konsekvensens gradering beskrives som følger:

1. *ubetydelig – nesten ingen innflytelse*
2. *mindre – liten innflytelse som med letthet kan ordnes*
3. *moderat – medium innflytelse som kan bli ordnet med noe innsats*
4. *betydelig – seriøs innflytelse som vil være vanskelig å ordne*
5. *katastrofal – skjebnesvanger innflytelse som vil være nesten umulig å ordne*

(Bragelien, Skjølsvik & Voldsund, 2020, s. 316).

Risikofaktoren gir et bilde på kombinasjonen av sannsynlighet og konsekvens, og gir oversikt over hvilke risikoer som kan betegnes som kritiske, betydelige og neglisjerbare (Bragelien, Skjølsvik & Voldsund, 2020, s. 316). Basert på risikohåndteringsplanen kan det utarbeides tiltak for å begrense sannsynligheten og konsekvensen for at en risiko inntreffer, og dermed også minimere risikofaktoren. Det er viktig å merke seg at det er umulig å klare å forutse og ta høyde for alle risikoer som kan oppstå, og at dette i seg selv er en risiko. De neste avsnittene vil ta for seg de fire risikoene som har blitt identifisert med høyest risikofaktor. Alle disse ligger mellom 10 og 20, som kan betegnes som betydelige risikoer.

Manglende teknisk kunnskap

Når et prosjekt med krav til teknisk kompetanse skal gjennomføres, vil det alltid være teknologi som i større eller mindre grad er ukjent. Ved prosjektets start var det tydelig at dette er en risiko som med høy sannsynlighet ville inntreffe, og konsekvensen av dette ville kreve noe ekstra arbeid. Det er derfor blitt satt av tid til å sette seg inn i teknologier knyttet til prosjektet som en del av den overordnede planen.

Undervurdering av milepælers omfang og kompleksitet

Undervurdering av milepælers omfang og kompleksitet er noe som kan føre til forskyvning av tidsplanen, og dermed økt arbeidsmengde. Ferdigstilling av milepælene er avgjørende for prosjektets fremgang, både for produktet og rapporten, og er derfor et viktig suksesskriterium for levering av endelig rapport.

Pandemi

Skulle et av gruppemedlemmene pådra seg Korona, vil dette føre til økt arbeidsmengde hos de resterende, friske medlemmene. Dette kan føre til forskjøvet tidsplan, som igjen kan gjøre at milepæler ikke blir oppnådd. En proaktiv tilnærming til dette har vært å ha ukentlige oppdateringer internt der kunnskapen fra den enkelte har tilegnet seg, har blitt delt med resterende gruppemedlemmer.

Risikoer det ikke er tatt høyde for

Det er svært sannsynlig at det eksisterer risikoer som ikke er identifisert, og dermed ikke er tatt høyde for i risikoanalysen. Konsekvensen av slike risikoer er svært varierende, men kan i større eller mindre grad føre til forsinkelser og endringer i opprinnelig plan.

Denne risikoen kan begrenses i noen grad ved å utføre en risikoanalyse, og utarbeide en risikohåndteringsplan. Risikohåndteringsplanen er utarbeidet basert på identifiserte og analyserte risikoer. Planen er utarbeidet for å forebygge ulike risikoer knyttet til prosjektet. Dette er en risiko som må aksepteres. Endring i planene er en del av prosessen, da opprinnelig plan sjeldent kan følges til punkt og prikke.

3.5 Evalueringsplan

Det holdes ukentlige møter med oppdragsgiver der det har blitt vist frem midlertidig resultat for evaluering, som en del av den smidige utviklingsmetodikken. Resultatet av disse evalueringene har blitt brukt til å forstå hvorvidt milepælene er oppfylt. Det vil også bli skrevet enhetstester underveis som tester logikken i microfrontend-komponenten.

Det har vært en pågående dialog om brukertesting av prosjektet. Fordi komponentene ikke er implementert i oppdragsgivers miljø, vil det ikke være tid til brukertesting i løpet av prosjektets varighet. Ved fullstendig implementasjon er brukertesting noe oppdragsgiver vil ta tak i, og da for utviklere i organisasjonen. Det er disse som er brukerne av komponentene som skal utvikles. Dette fordi «bruken» av disse komponentene er å implementere nye eller eksisterende funksjoner som microfrontend.

Det er derfor besluttet å gjennomføre en heuristisk evaluering av de opprinnelige milepælene. Netinbag beskriver en slik evaluering som «en prosess der en ekspert evaluerer et brukergrensesnitt eller lignende system ved hjelp av en liste over retningslinjer» (Netinbag, u.å.). Denne evalueringsmetoden er i utgangspunktet basert på brukergrensesnitt, men har i dette tilfellet blitt modifisert og tilpasset milepælene gitt fra oppdragsgiver. Preece, Rogers & Sharp beskriver denne som en evaluering som kan gi gode tilbakemeldinger når det ikke er mulig å utføre brukertester (Preece, Rogers & Sharp, 2019, s. 550).

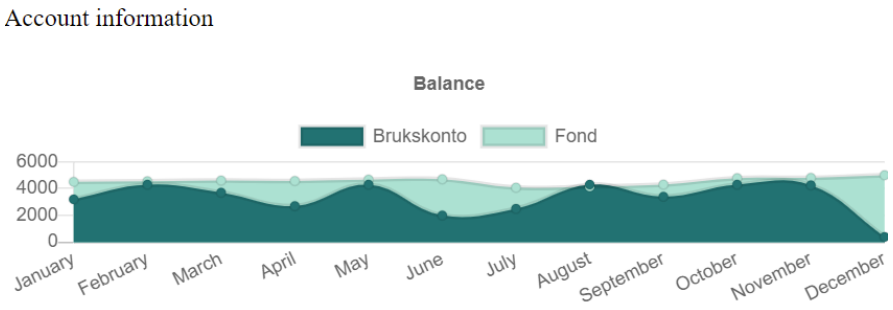
4 DETALJERT DESIGN

4.1 Milepæl 1: Enkel Webside

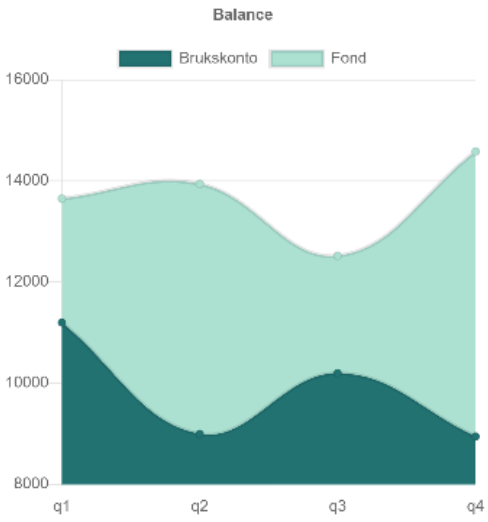
4.1.1 Arkitektur

I første milepæl ble det utviklet en helt enkel webside som består av en React-komponent. Denne har grafisk fremstilling av data hentet fra en JSON server som er driftet lokalt på maskinen.

Grafen ble også gjort responsiv, og viser alle månedene på større skjermer, mens på mindre skjermer vises saldo per kvartal. Idéen med grafen er at den skal gjenspeile en kundes konto(er), og gi ut saldoen månedsvis. Grunnen til at designet og implementasjonen av grafene er enkle, er at oppgaven har et mer konseptuelt fokus. En illustrasjon av grafene vises i Figur 4 og Figur 5 nedenfor.



Figur 4: Grafen vist i maksimert vindu i nettleser



Figur 5: Grafen simulert på vindu på størrelse med iPhone X, i stående retning

4.2 Milepæl 2: Webside som en microfrontend

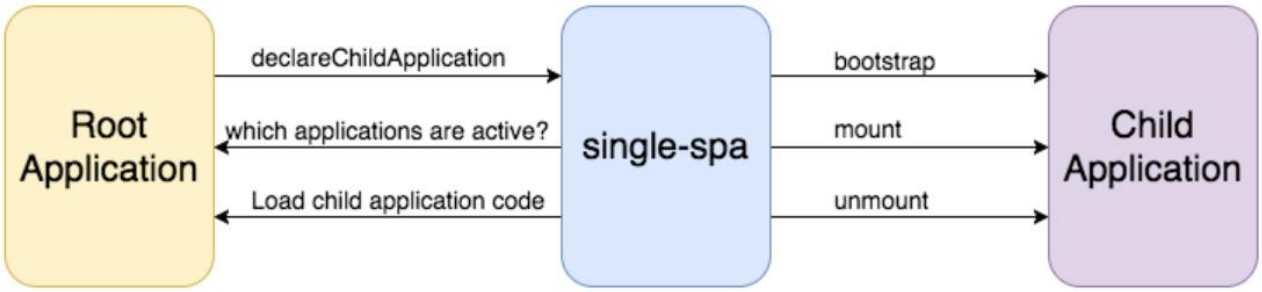
Metode valgt for implementasjon av microfrontend-konseptet er drøftet og gjort rede for i punkt 3.1.4. Konseptet single-SPA er beskrevet i punkt 2.2.

4.2.1 Arkitektur

Single-SPA tar inspirasjon fra moderne rammeverk-komponent livssykluser ved å abstrahere livssyklusene for hele applikasjonen. Single-SPA er et bibliotek som muliggjør en frontend microservice-arkitektur, også kjent som microfrontend. En Single-SPA-applikasjon består av minimum to komponenter: en rot-applikasjon og en «barne»-applikasjon (Single-SPA, u.å.a). Single-SPA er ansvarlig for å håndtere barne-applikasjonen basert på rot-applikasjonens meldinger, å gi rot-applikasjonen informasjon om hvilke applikasjoner som er aktive og å laste inn kode fra barneapplikasjonene (se Figur 6).

Rot-applikasjonen er komponenten som gjengir HTML-siden og registrerer applikasjonene. Hver applikasjon er registrert med et navn, en funksjon for å laste inn applikasjonens kode og en funksjon som bestemmer om/når applikasjoner er aktive (Single-SPA, u.å.a).

Applikasjoner kan sees på som enkeltsideapplikasjoner som er pakket inn i moduler. Hver applikasjon må kunne bootstrappe, feste og løsne seg selv fra sidens Document Object Model [DOM]. Den store forskjellen fra «vanlig» SPA og single-SPA er at i single-SPA, må applikasjonene kunne sameksistere med andre applikasjoner, fordi hver enkelt applikasjon ikke har egen HTML-side (Single-SPA, u.å.a).



Figur 6: beskrivelse av single-SPA (Denning, 2020b)

4.2.2 Fremgangsmåte

Fremgangsmåten for å gjøre en React-App til en microfrontend-applikasjon har blitt gjort ved å følge single-SPA's egen dokumentasjon (Single-SPA, u.å.d). Dette gjorde at arbeidet fra milepæl 1

kunne gjenbrukes og implementeres i milepæl 2. Det har også blitt opprettet single-SPA-applikasjoner fra begynnelsen, for å gjøre seg kjent med begge fremgangsmåter. Disse fremgangsmåtene er beskrevet i 9.3.2.

For bygging av prosjektet ble verktøyet Travis CI brukt. Verktøyet lager en samling av Javascript-objekter for hver microfrontend-applikasjon. Denne lastes opp i AWS S3. Videre har egne filer i hvert prosjekt blitt knyttet opp mot et eksternt import-map i S3. Dette sørger for at en oppdatering i en applikasjon kun krever oppdatering av denne applikasjonen og eksternt import-map. Dette er en av de store fordelene med microfrontend; endringer kan rulles ut til en applikasjon uten å måtte bygge hele prosjektet på nytt. Skulle det oppstå en feil i oppdatert applikasjon, vil resten av komponentene forbli som før oppdateringen ble rullet ut.

4.2.3 Utfordringer

Per i dag fungerer ikke prosjektets applikasjoner i Internet Explorer [IE]. Ifølge Adrian Kristoffer Borgund, teknisk veileder fra DNB (samtale i forbindelse med ukentlig møte med oppdragsgiver, 29. april 2021) er det fortsatt en liten andel av deres brukere som benytter seg av IE som standard nettleser. Det har derfor blitt sett på muligheter for å komme rundt denne problematikken. En av Single SPA's forfattere, Joel Denning, viser til at dette lar seg løse med polyfilling, som i en vanlig React-app (Denning, 2021). Polyfill er en bit med kode (vanligvis JavaScript i Web) som brukes for å sørge for moderne funksjonalitet på eldre nettlesere som ikke støtter denne funksjonaliteten (Mozilla Developers Network, u.å). Dette ble forsøkt, men ble ikke prioritert å få ferdig til prosjektets frist. Grunnen til dette var pga. tidsmangel og at støtten for IE i alle Microsoft 365-applikasjoner utløper 17. august 2021 (Keizer, 2020). Adrian Kristoffer Borgund, teknisk veileder fra DNB (samtale i forbindelse med ukentlig møte med oppdragsgiver, 26. mai 2021) opplyser om at polyfilling løser konkrete funksjoner som klasser og objekter, men ikke funksjoner som for eksempel pil (=>)-funksjoner.

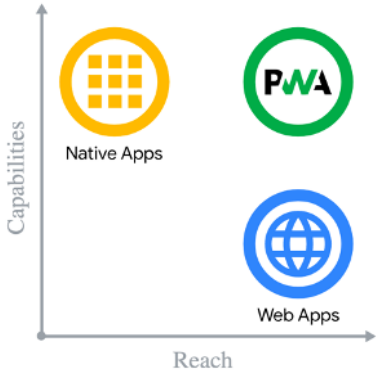
I single-SPA's dokumentasjon beskrives bruken av egne moduler for eksportering av funksjonalitet (utility-modules) som andre microfrontends kan importere. Eksempler på dette er styling, autentiseringshjelpere og API-hjelpere (Single-SPA, u.å.f). Slike moduler ville vært med på å forsterke gjenbrukbarheten av produsert kode. Dette ble ikke utforsket, da disse funksjonene, som nevnt i foregående avsnitt, ble tilsidesatt/nedprioritert. Henting av data og styling skjer derfor i microfrontend-applikasjonene.

4.3 Milepæl 3: PWA som en microfrontend

Ønsket fra oppdragsgiver var å lage en PWA som implementerer en gjenbrukbar microfrontend-komponent for utviklerne. I dette kapittelet blir det gjort rede for hva en PWA er, hvordan en slik løsning er forsøkt løst og utfordringer som har dukket opp.

4.3.1 PWA

En PWA er en applikasjon som kjører via en nettleser, og samtidig beholder de samme egenskapene som en vanlig mobilapplikasjon, som for eksempel kamera og pushvarsler (se Figur 7). Nettstedet er responsiv og kan lastes ned til skrivebordet på enheten. Nettsiden installeres som en applikasjon på enheten, og oppleves som en vanlig mobilapplikasjon for brukeren. Denne kan kjøres både med og uten internettilkobling ved hjelp av en service worker (Ateles, 2019). Hovedkomponentene i en PWA, er en service worker og en manifestfil.



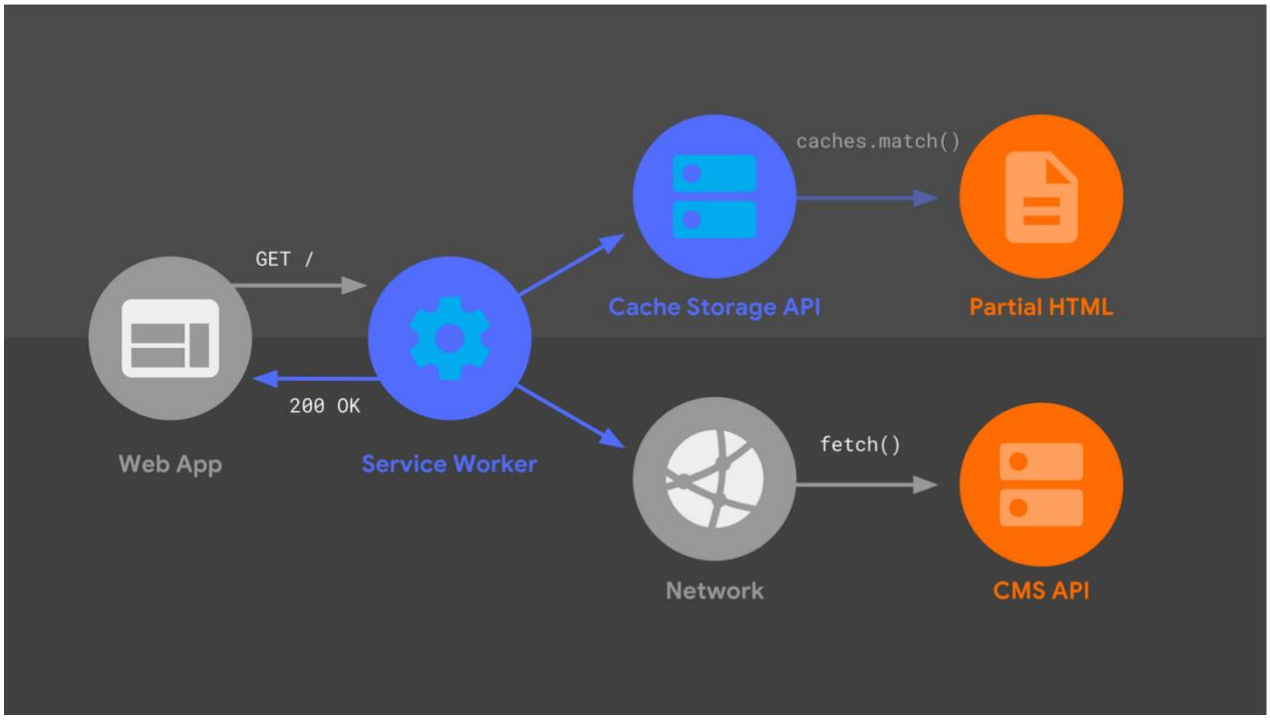
Figur 7: Web applikasjoner, Native applikasjoner og PWA (LePage & Richard, 2020)

En service worker er et skript som blir kjørt i bakgrunnen av nettleseren, og brukes for å kontrollere nettverksforespørsler. Service workeren gjør det mulig å lagre samling av data (cache) alle ressurser som trengs for applikasjonen. Utvikleren bestemmer når PWA skal hente fra datasamlingen og når den skal hente fra internett (Gaunt, 2020). I tillegg brukes en service worker til å få tilgang til enhetens system meldinger, for eksempel kamera og pushvarslinger.

Manifestfilen inneholder informasjon om applikasjonen og gjør det mulig å installere applikasjonen på sin enhet. Manifestfilen forteller hvordan applikasjonen oppfører seg når den er installert på skrivebordet eller mobilen til en bruker. Dette er informasjon som applikasjonens navn, ikon som skal brukes og hvilken URL applikasjonen skal kjøres fra. Filen lages i den overordnede mappen i prosjektet. Hovedattributtene i manifestfilen kan leses mer om i punkt 9.3.1.

4.3.2 Arkitektur

Som tidligere nevnt, består en PWA av to hovedkomponenter: En service worker og en manifestfil. En service worker registreres i applikasjonen. Dersom tilkoblingen til internett er til stede, vil service workeren hente data fra internett. Uten tilkobling til internett, vil service workeren hente data fra datasamlingen. Her ligger alle lagrede ressurser. Når applikasjonen ikke har internett, hentes det frem en egendefinert side som vises til brukeren. En illustrasjon om hvordan dette fungerer, blir vist i Figur 8.



Figur 8: Arkitekturen i PWA (Posnick, 2021)

4.3.3 Arbeid så langt

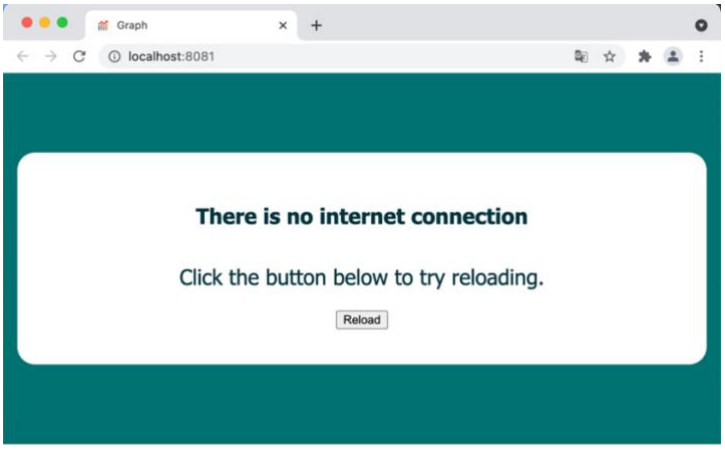
Etter et møte med arbeidsgiver, ble det rådet å lage en vanlig PWA først, og i ettertid integrere microfrontend i PWA. Dette ble gjort og fremgangsmåten på hvordan en PWA lages blir beskrevet trinn for trinn i punkt 9.3.1. En egendefinert service worker er laget og blir registrert i HTML-filen der alle sidene som er med i applikasjonen blir lastet inn. Koden for egendefinert service worker er vist i Figur 9.

```
<script>
  if('serviceWorker' in navigator) {
      window.addEventListener('load', () => {
          navigator.serviceWorker.register('./service-worker.js')
              .then((reg) => console.log('Registration Succeeded: ', reg.scope))
              .catch((err) => console.log('Registration Failed: ', err));
      })
  }
</script>
```

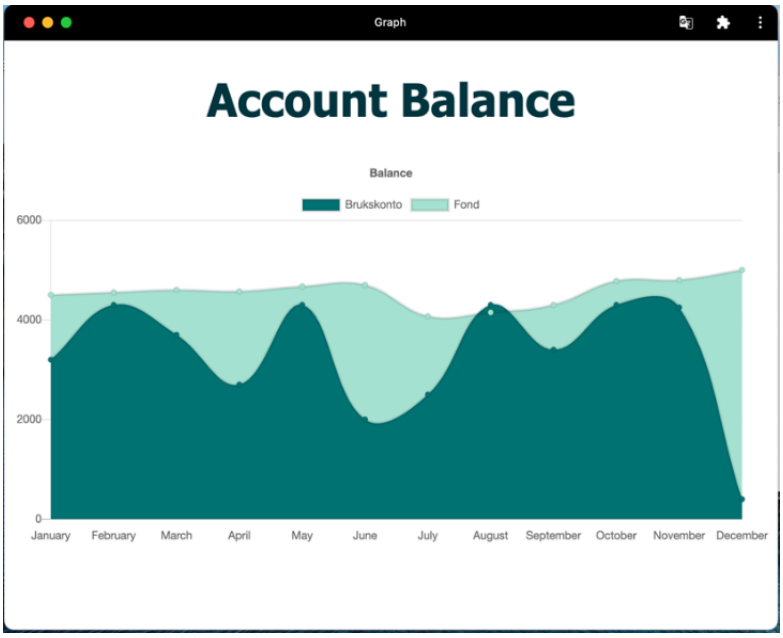
Figur 9: Registrering av egendefinert service worker

Det er blitt laget et skript for å registrere service workeren i applikasjonen. Selve service workeren er laget i en egen fil kalt service-worker.js. Denne filen inneholder tre metoder som beskriver hvordan service workeren fungerer: installer, aktiver og hent. Installer-metoden legger til alle ressurser som skal lagres i datasamlingen. Kilden til disse ressursene blir definert slik at service workeren vet hvor den skal hente data. Ressursene lagres, og gjør at applikasjonen kan brukes uten internettilkobling. Aktiver-metoden oppdaterer datasamlingen dersom det har skjedd endringer i denne, og sletter den gamle. Hent-metoden henter den responsen som blir sendt tilbake av nettleseren. Dersom forespørselen får en respons tilbake, henter den responsen. Dersom forespørselen ikke får noe respons, returnerer den en side som er lagret i datasamlingen.

Det har blitt laget en egen HTML fil som lastes inn når internettilkoblingen er borte. Filen heter offline.html og blir lagret i datasamlingen. Offline.html inneholder informasjon om at brukeren ikke har internett med en knapp som oppdaterer siden. Dersom tilkoblingen har kommet tilbake, får brukeren opp det som faktisk vises i applikasjonen. Ellers blir samme melding vist. Se Figur 10 og Figur 11 for nærmere illustrasjon.



Figur 10: Illustrasjon av offline.html



Figur 11: PWA som frittstående applikasjon

Det har blitt brukt en manifestfil for å gi informasjon til nettleseren om applikasjonen. Applikasjonen har fått navnet Graph og et ikon som fremstiller en graf. Med denne filen blir det mulig å installere applikasjonen på sin egen enhet. Bilder blir knyttet til applikasjonen som et ikon. Eksempler på situasjoner dette ikonet brukes, er i fanen på nettleseren og på brukerens skrivebord. I tillegg blir det definert hvilken URL applikasjonen kjøres fra. I dette tilfellet blir den kjørt lokalt på maskinen. Applikasjonen kjøres som en frittstående applikasjon, som vil si at applikasjonen ikke er del av et noe sett. Annen informasjon som nettleseren trenger er også definert i denne filen. For alle attributtene som er med i manifestfilen, se 9.3.1.

Videre er applikasjonen gjort om fra en PWA til en single-SPA applikasjon. Det første steget var å lage selve applikasjonen til en single-SPA applikasjon. Dette gjøres ved å legge til en ekstra fil som implementerer metodene: bootstrap, feste og løsne fra single-SPA. Videre var å registrere applikasjonen som en single-SPA. Dette ble gjort ved å bruke metoden registrerApplication fra single-SPA, og hente attributten "root" i HTML-dokumentet. Mer ble ikke gjort. I delkapittel 4.3.4 blir det gjort rede for hva slags utfordringer og problemer som dukket opp ved implementasjon av single-SPA i en PWA.

4.3.4 Utfordringer

Ved integrering av microfrontend-miljøet i PWA, startet det å oppstå problemer. Først var det vanskelig å finne dokumentasjon og fremgangsmåte på hvordan integrering av single-SPA fungerer i PWA. Dette førte til at det ble utforsking på egenhånd. Da applikasjonen skulle implementeres som en del av microfrontend-miljøet som beskrevet i punkt 4.2.1, oppstod det problemer. I single-SPA er det en HTML-fil for hele miljøet. Det er denne som har kontroll på hva som vises på siden. I PWA finnes det også en HTML-fil med tilsvarende hensikt. Konflikten oppstår ved at to HTML-filer prøver å kontrollere. En syntaks-feilmelding fra single-SPA oppstår. Pga. tidspress måtte denne delen av prosjektet nedprioriteres, slik at andre deler ble fullført. Det ble gjentatte ganger forsøkt å løse problemene, men etter mye tid og ingen progresjon, måtte milepælen nedprioriteres. Dette pga. manglede kunnskap. Fokuset ble derfor flyttet til milepæl 4. Ifølge Joel Denning, er det mulig at en PWA fungerer med single-SPA, men ikke hvordan dette gjøres (Denning, 2020a).

4.4 Milepæl 4: Mobilapplikasjon som en microfrontend

4.4.1 Native-applikasjoner sammenlignet med hybrid applikasjoner

Native-applikasjoner er applikasjoner som er utviklet for en bestemt enhet, for eksempel Android eller iOS. Native-applikasjoner er brukt for å ha tilgang til funksjonene som Android/iOS enheter har. Dette kan for eksempel være tilgang til kamera og notifikasjoner (Gillis, 2020). Ved å bygge en native-applikasjon, har utvikleren full kontroll på enheten, og applikasjonen vil ha større mulighet til å utnytte mobilens potensiale.

En annen tilnærming for utvikling av mobilapplikasjoner, er bruken av hybride applikasjoner. Når en hybrid applikasjon lages, blir det laget i både Android og iOS applikasjoner basert på den originale koden. En delt kodebase gjør at det kan spares tid og penger på utviklingen.

For å kunne tilrettelegge for senere bruk av enhetsfunksjoner som Android og iOS har tilgjengelig, er applikasjonen blitt laget som en native-applikasjon.

4.4.2 Fremgangsmåte

Det er lagt til rette for utvikling av to mobilapplikasjoner: en for Android og en for iOS. Android tar i bruk Android Studio med Java, mens for iOS brukes Swift og Xcode. Dette har blitt gjort for å kunne ha mest mulig kontroll på hver enkelt applikasjon.

Fokuset har vært å fremstille grafen fra tidligere milepæler som en microfrontend. Ønsket om å kunne bruke den samme grafen baserer seg på de initielle kravene knyttet til gjenbrukbarhet og brukervennlighet. Grafen er først gjenspeilet i et webview og er videre tenkt implementert som en egen komponent i applikasjonen.

4.4.3 Arkitektur

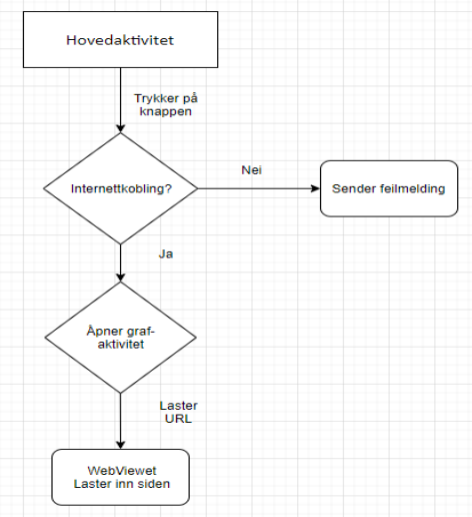
Når det gjelder arkitektur innenfor mobilapplikasjoner, er det viktig å tenke på at mange hendelser kan inntreffe parallelt (Android Developer, 2021a). Et eksempel på en slik hendelse kan være et innkommende anrop. Når anropet avsluttes, ønskes det at applikasjonen tar brukeren tilbake til det samme utgangspunktet som da hendelsen inntraff. Eksempler på andre hendelser som må tas hensyn til, er minimering av applikasjonen, notifikasjoner og vertikal/horisontal modus.

4.4.4 Arbeid

Hovedfokuset har vært å få laget en protype for å sjekke om microfrontend-prinsippet vil fungere i mobilapplikasjoner. Innledningsvis ble det laget en Android-applikasjon for å teste ut konseptet. Deretter ble webview-konseptet testet for iOS-applikasjoner.

4.4.4.1 Android

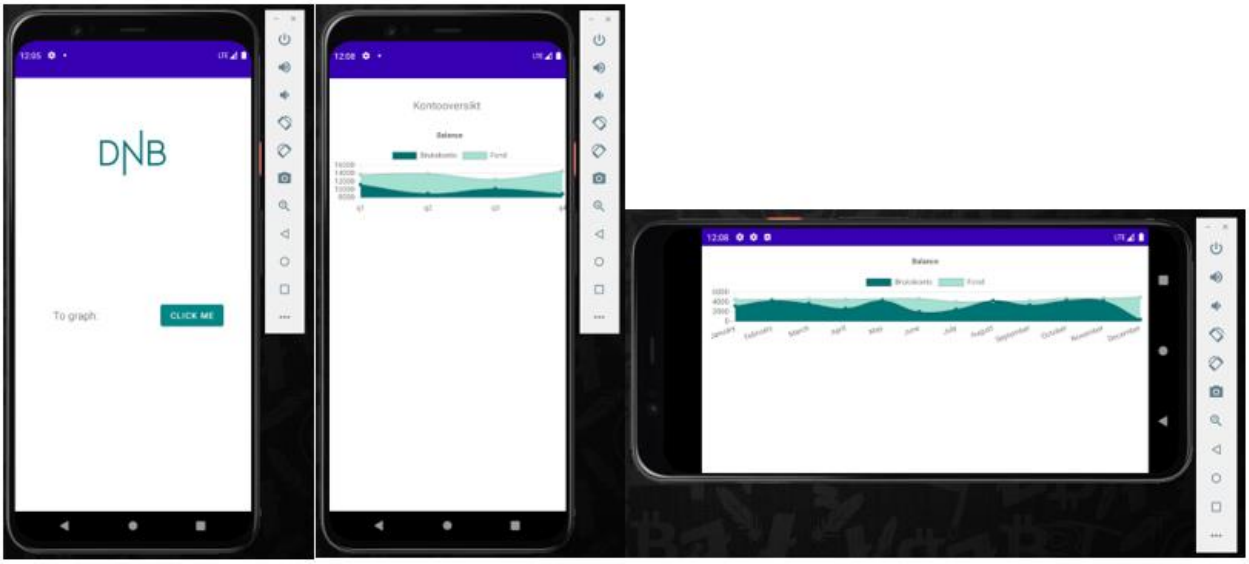
Android-applikasjonen tar utgangspunkt i kvalitetsretningslinjer fra Android Developers nettside (Android Developer, 2021b). Denne listen beskriver kjernekvalitetskriterier for vurdering av applikasjonen. Applikasjonen er laget for å teste konseptet, og det er ikke nødvendigvis tatt hensyn til alle de ulike kriteriene. Kriterier som er blitt tatt hensyn til er visuell kvalitet og funksjonalitet.



Figur 12: Handlingsdiagram applikasjon

Selve applikasjonen er laget i Android Studio, som er beskrevet i punkt 3.3.5. Applikasjonen fungerer som vist i Figur 12 og illustrert i Figur 13.

- Hovedaktivitet: har kontroll på de ulike aktivitetene, og viser en enkel hjemme side der brukeren har valg om å se på grafen. Når brukeren trykker seg videre til grafen vil det først bli sjekket opp i følgende:
    - Har brukeren internettkobling?
        - Hvis «JA», send videre en ny hendelse og start graf-aktivitet
        - Hvis «NEI», send feilmelding til brukeren og forteller at det trengs internettilkobling.
    - Graf-aktivitet består av en enkel side med tekst og et webview. Webviewet laster inn URL'en til grafen. Grafen som er fremstilt i webviewet er laget for å undersøke skjermstørrelsen og deretter skalerer seg i forhold til dette.
        - Tilstand til applikasjonen blir lagret. Hvis brukeren snur mobilen, vil samme data fortsatt bli vist.
        - Det er blitt tatt hensyn til at brukeren kan navigere med den vanlige tilbakeknappen i webviewet.

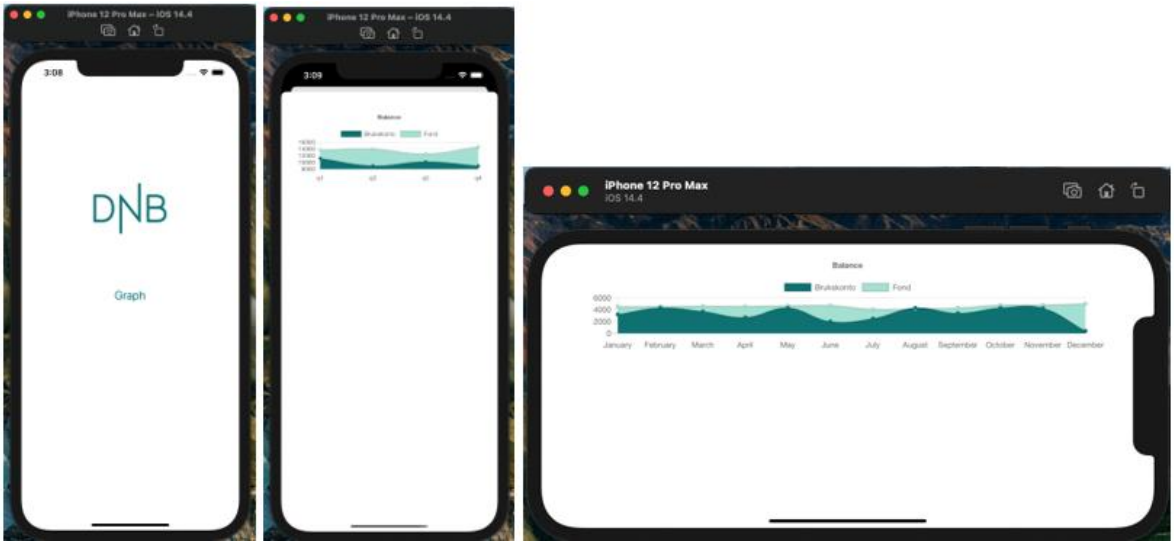


Figur 13: Android applikasjon med webview

4.4.4.2 iOS

iOS-applikasjonen ble utviklet for å teste ut webview-konseptet. Utover testingen av dette konseptet har ikke applikasjonen blitt videreutviklet, grunnet lite nytteverdi. Applikasjonen består av en enkel forside med en knapp. Ved å trykke på denne knappen, vises grafen i et webview. Det er mulig å se microfrontend-komponenten fra milepæl 2, som vist på Figur 14.

Webview konseptet er prøvd ut, og fungerer til å gjenspeile microfrontend-komponenten. Det er ikke tatt høyde for feilmeldinger, rotasjon av enhet og generell oppbygning av applikasjonen. Hvordan dette blir håndtert i en Androidapplikasjon blir videre diskutert i 4.4.6.



Figur 14: iOS-mobilapplikasjon

4.4.5 Utfordringer

Det har vist seg å være vanskelig å klare å fremstille grafen i et webview ved kjøring lokalt på maskinen. Derfor har det vært nødvendig å kjøre applikasjonen på en webserver. Løsningen på dette har vært å ta i bruk en AWS S3, som beskrevet i punkt 3.3.7. Webviewet fungerer som planlagt når den tar i bruk URL'en.

Direkte import av komponenter til mobilapplikasjonen har ikke vært mulig å få til. Dette kom frem i en samtale med Joel Denning, forfatter av single-SPA (single-SPA's Slack kanal 14. mai 2021), der han forteller at single-SPA ikke er ment for mobilapplikasjoner. Det ble prøvd ut teknikker for å løse problemet, men en virkende løsning ble ikke utrettet. Direkte importering var et ønske fra oppdragsgiver, men ikke en del av den opprinnelige oppgaven. Grunnet dette ble det ikke prioritert mer tid til å finne løsninger på problemet. Fremgangsmåten som har vært

brukt her er som følgende: Pakke ned den originale applikasjonen, deretter importere den inn i React Native, for å så kunne ta denne i bruk i Android Studio. Denne måten var ment som en bro mellom React og Android Studio.

Ved nedpakking av den originale applikasjonen, oppstår det problemer. Problemet er at mange av importpakkene til single-SPA må fjernes. Det gjør at microfrontend-prinsippet forsvinner og det vil ikke være noen annen forskjell enn å lage et vanlig bibliotek som kan bli brukt for mobilapplikasjonen.

4.4.6 Løsning

Løsningen i begge mobilapplikasjonene bygger på bruk av webview. Webviewet gjenspeiler grafkomponenten samtidig som Android-applikasjonen tar hensyn til ulike feilmeldinger. Eksempel på en feilmelding er at webviewet ikke lastes inn hvis det ikke er tilgang til internett. Det vil da vises en notifikasjon som forteller brukeren dette. Det er utført visuelle tester for responsivhet av grafen for å sjekke ønsket funksjonalitet i forhold til nettsiden, og at den oppfører seg som ønsket ved rotering av skjerm.

Løsningen svarer på oppgaveteksten og gir et inntrykk for videre muligheter. Det gjenstår å finne ut av alternative måter for optimalisering av applikasjonen, som blir diskutert i punkt 7.4.

5 EVALUERING

Oppgavens evaluering baserer seg på evalueringsplanen nevnt i punkt 3.5. Metodene som er blitt brukt for evaluering er beskrevet i punkt 5.1, og resultatene av disse i punkt 5.2.

5.1 Evalueringsmetode

5.1.1 Heuristisk evaluering

5.1.1.1 Milepæl 1

«En responsiv frontend-komponent som implementerer ønsket funksjonalitet på en enkel side»

Evalueringen av milepæl 1 fokuserer på om hvorvidt komponenten er responsiv, og om den implementerer ønsket funksjonalitet. For å undersøke om siden er responsiv, har det blitt undersøkt hvorvidt innholdet i komponenten endres og tilpasses ulike enheter, som for eksempel iPhone X og nettleser.

Funksjonaliteten i komponenten har blitt evaluert underveis med oppdragsgiver. Midlertidige prototyper har blitt vist frem, og oppdragsgiver har kommet med tilbakemeldinger på disse prototypene, og deres innhold. Eksempler på funksjonalitet i komponenten, er bruk av Eufemia, henting av data, og grafisk fremstilling av disse.

5.1.1.2 Milepæl 2

«En responsiv microfrontend bygget på milepæl 1»

Evalueringen i denne milepælen fokuserer på microfrontend-konseptet, og implementasjonen av dette. Dette har blitt gjort for å evaluere om komponenten har samme design og funksjonalitet fra forrige milepæl, men implementert som en microfrontend. Hovedfokuset i evaluering av denne milepælen har derfor vært enhetstester for å se om komponenten fungerer logisk. Ende-til-endetester er utført for å undersøke funksjonaliteten som en del av helhetlig microfrontend-system. Forenklet kan evalueringen være så enkel som «er komponenten fra milepæl 1 implementert i et fungerende microfrontend-miljø?».

5.1.1.3 Milepæl 3

«En responsiv micro-frontend implementert i/som en PWA»

Evalueringen av denne milepælen baserer seg på om hvorvidt funksjonaliteten fra milepæl 2 er implementert i en PWA, sammen med single-SPA. Evalueringen av denne milepælen vil basere seg på om det er en fungerende PWA med ønsket funksjonalitet og om denne fungerer i single-SPA-miljøet. Videre vil det fokuseres på hvilke momenter som fungerte, og hvilke som ikke gjorde det, for å gjøre kartlegging av veien videre mer konkret; hva er begrenset av rammeverket og hva er begrenset av programmererens kompetanse.

5.1.1.4 Milepæl 4

«En responsiv micro-frontend implementert i en webløsning og hentet i et webview i Android/iOS»

Evalueringen av webløsningen vil basere seg på evalueringen fra milepæl 2, mens evaluering av løsningen som omhandler webview og Android/iOS må sees nærmere på. Det ble undersøkt hvorvidt funksjonaliteten i komponenten, og implementasjonen av microfrontend-konseptet fungerte som en del av en mobilapplikasjon. Det har blitt sett på ulike løsninger som implementasjon av et webview direkte i mobilapplikasjonen.

5.1.2 Enhetstester

Det er gjennomført enhetstester gjennom prosjektets varighet. Testene baserer seg på logikken i grafkomponenten, samt testing av rot-komponenten til grafapplikasjonen.

For testing av grafapplikasjonen er det blitt brukt Jest og testing-library-react. Testene har blitt skrevet ved å opprette en grafkomponent som en , sjekke at denne blir lagt til i dokumentet, for å deretter gjøre en forespørsel via axios som hentet data fra en JSON server. Axios er en løftebasert (promise based) HTTP-klient for nettleseren og node.js (NPM, u.å.b). Denne klienten gjør det mulig å sende en forespørsel til en URL, som returnerer en respons.

For rot-komponenten er det blitt gjort testing på at det faktisk blir opprettet en graf-komponent når denne gjengis.

5.1.3 Ende-til-endetesting

For ende-til-endetesting anbefaler single-SPA-dokumentasjonen en av to tilnærminger: ende-til-endetesting med frittstående modus, eller teste alt samlet (Single-SPA, u.å.c).

Valget i denne oppgaven falt på å teste alt samlet. Denne tilnærmingen speiler en full ende-til-endetest der det kjøres samme kode som i miljøet. I prosjektet ble alle komponentene kjørt lokalt. Det er mulig å overstyre import-map slik at alle komponentene, bortsett fra den som skal testes, driftes på en server (Single-SPA, u.å.c).

Testene ble gjort ved å bruke Cypress, et verktøy som brukes for å teste alt som kjører i en nettleser (Cypress, 2021). Dette muliggjorde simulering av en forespørsel som hentet data fra en JSON server, og at dette fungerte som forventet. Testen simulerte hvordan siden ville se ut før data ble hentet inn, og hvordan den vil se ut når data var hentet og prosessert av graf-komponenten. Som en del av testene ble det også testet implementasjonen av komponenter fra Eufemia og bruken av disse.

5.1.4 Evaluering fra oppdragsgiver

Som nevnt i punkt 3.5, har det blitt holdt ukentlige møter med oppdragsgiver. Oppdragsgiver har hjulpet med å sette rammer og begrensninger for prosjektet, og kommet med konstruktive tilbakemeldinger på allerede gjennomført arbeid, samt veien videre.

5.2 Evalueringsresultat

5.2.1 Heuristisk evaluering av milepæler

Resultatet av evalueringen i milepæl 1, er at komponenten er delvis responsiv, og ikke implementerer ønsket i funksjon i noe særlig grad.

Komponenten har to forskjellige fremstillinger av grafer. Hvilke av disse to som vises, er basert på skjermens bredde i forhold til en fastsatt grense. Figurene som viser de ulike fremstillingene, er vist i Figur 4 og Figur 5. Dette er et av mange virkemidler for å ta i bruk prinsippene om at grafen skal være responsiv, men er alene ikke nok til å tilfredsstille kravene om en responsiv komponent.

Funksjonaliteten til komponenten evalueres til heller begrenset. Den ivaretar det overordnede målet om å hente inn data og viser en graf basert på disse, men tar ikke i bruk DNB's dev-API.

Dette gjør at komponenten ikke har funksjonalitet som ble diskutert tidlig i prosjektets fase, som for eksempel brukerautentisering.

Evalueringen av milepæl 2, er at denne milepælen er fullført, men med etterslep fra milepæl 1's utfordringer. Funksjonaliteten til komponenten fra milepæl 1 er implementert som en microfrontend i en webapplikasjon.

For milepæl 3 evalueres hele milepælen som ikke oppnådd, selv om noe funksjonalitet og en fungerende PWA eksisterer. Dette er fordi at PWA ikke er implementert som en microfrontend. Det er verdt å merke seg at det er teknisk kompetanse som fører til ikke fullført, og ikke rammeverkets begrensninger.

I milepæl 4 evalueres websiden som fullført, men med svakheter rundt design og funksjon, som nevnt i avsnittene over. Websiden fungerer som microfrontend, er delvis responsiv og implementerer en svært grunnleggende funksjonalitet med tanke på henting og fremstilling av data. Mobilapplikasjonen er en fungerende applikasjon, men har utfordringer med innhenting av komponenten implementert som en microfrontend. Fordi denne blir hentet inn som et webview, er det større utfordringer med tanke på funksjonalitet som en del av en mobilapplikasjon.

5.2.2 Evaluering fra oppdragsgiver

Resultatet av tilbakemeldingene har vært avgjørende for det endelige resultatet, da hele prosjektet baserer seg på disse. Et eksempel på dette er at milepæl 3 ble nedprioritert som en direkte konsekvens av oppdragsgivers tilbakemelding på viktigheten av denne milepælen, i forhold til de resterende. Et annet eksempel er bruken av en JSON server istedenfor å benytte seg av DNB's dev-API. Ettersom oppdragsgiver ikke anså dette som direkte knyttet til oppgaven, ble dette skrinlagt for å frigjøre tid til andre oppgaver.

Det er verdt å nevne at om prosjektet skulle ha blitt gjennomført på nytt, ville det vært lagt større vekt på konkrete evalueringer knyttet til for eksempel en prototype per milepæl. Denne kunne sørget for en tydeligere evaluering av prosjektets hel- og delmål, og dermed gikk mer konkrete evalueringsresultater å jobbe med.

6 DISKUSJON

6.1 Konsekvenser av valgt løsning

Valget av microfrontend-rammeverk har hatt en stor konsekvens for videre utvikling. Som tidligere beskrevet i punkt 3.2, falt valget på single-SPA. Single-SPA virket som et lovende rammeverk da det ble valgt, men har i ettertid vist seg å ha begrensninger som ikke ble oppdaget før aktuelle problemstillinger i hver milepæl ble utforsket. Valget ble tatt relativt tidlig i prosjektet og ble derfor påvirket av suksesskriteriene knyttet til milepæl 2, som omhandler webapplikasjoner.

Single-SPA egner seg best for web-løsninger, og ikke for native applikasjoner. Dette ble oppdaget sent i prosjektet, og var derfor ikke tid til å starte på nytt med et annet rammeverk. Vurderingen av hvilket rammeverk som skulle velges, var vanskelig i den tidlige fasen av prosjektet. Kunnskapen om de ulike teknologiene var liten, og etter undersøkelser av rammeverket, virket single-SPA som en god løsning på daværende tidspunkt.

6.2 Utfordringer

Nye teknologier og lite erfaring har vært grunnen til at oppgaven ble mer omfattende enn først antatt. Dette har ført til en forskjøvet tidsplan og dermed lavere måloppnåelse av hver enkelt milepæl. Grunnet dette har det vært et større fokus på hva som må prioriteres.

Fordi microfrontend-konseptet er en relativt ny teknologi, har det blitt investert vesentlig tid for å danne seg en kunnskapsbase om de grunnleggende prinsippene. Denne prosessen har igjen ført til mange underliggende temaer som må utforskes, for å skaffe seg et helhetlig bilde. Av den grunn har det blitt investert vesentlig tid på konsepter som i utgangspunktet virket aktuelle for oppgaven, men som ikke er tatt i bruk.

Underveis har designet og funksjonaliteten i komponenten blitt nedprioritert, da det er konseptet og implementasjonen som har stått i fokus, og ikke hva slags komponent som vises. Design og bruken av DNB's eget dev-API har derfor blitt bortprioritert for å frigjøre ekstra tid til å forstå implementasjonen av microfrontend som en web- og mobilapplikasjon.

Implementasjon av microfrontend i PWA viste seg å være mer kompleks og krevende oppgave enn først antatt, noe som førte til liten progresjon i prosjektet. Tidspress førte til at milepælen som omhandler PWA, ble nedprioritert i samråd med oppdragsgiver. Dette førte til at hovedfokuset ble flyttet til milepælen som omhandler utvikling av mobilapplikasjon.

6.3 Konsekvenser av de oppnådde resultatene

6.3.1 Single-SPA

Valget av Single-SPA rammeverket har gjort at milepæl 3 og 4 har vært krevende å jobbe med. I begynnelsen virket single-SPA lovende med mye ulik dokumentasjon, men lite til ingen dokumentasjon på hvordan det kan brukes i PWA og i mobilapplikasjoner. Fremgangsmåten som ble brukt for å utforske single-SPA, i henhold til milepælene, var prøving og feiling. Prøving og feiling har vært en tidkrevende prosess og medførte lite fremgang selv med mye arbeid.

6.3.2 Direkte import

Utforskning av direkte importering av komponenter til Android Studio, er beskrevet i punkt 4.4.5. Det har som nevnt ført til at et webview er blitt tatt i bruk for å gjenspeile grafen. Ettersom komponenten ikke har blitt direkte importert, vil det være vanskelig å ha full kontroll på applikasjonen. Dette kan skape problemer for videre arbeid, som for eksempel en innloggingsfunksjon. Det kan oppstå hindringer ved overføring av data fra mobilapplikasjonen til nettsiden, og funksjonaliteten rundt dette.

6.3.3 Arbeidsmetoder

I startfasen av utviklingsdelen, ble det jobbet på samme delmål. Problemene ble en gjenganger, og det ble brukt mye tid på å finne riktige løsninger på problemene. Effektiviteten ble derfor lav. Arbeidsmetoden endret seg fra samlet til individuelt arbeid. Dette førte til at tidsbruken ble mer effektiv, som igjen førte til at utviklingsplanen ble overholdt. Ved en individuell fremgangsmåte, har det vært viktig med daglige møter og oppdatering for å dele kunnskap.

6.4 Forbedringer ved ny start

6.4.1 Anvendelse av teori

Å jobbe med en konseptoppgave har vært en ny erfaring. Tidligere prosjekterfaring har basert seg på utvikling og ferdigstilling av et produkt, og ikke hatt en vitenskapelig tilnærming. Dette har ført til at tilnærmingen til løsningen av oppgaven, har måttet bli endret. Anvendelse av ny kunnskap og feilmeldinger ved utviklingen, har tatt mye tid. Ved å ha mindre fokus på utviklingen og isteden referere til teori, kunne dette gitt de samme svarene. Denne tilnærmingen hadde spart mye tid.

6.4.2 Delmål

De største utfordringene i prosjektet underveis er knyttet til milepæl 3 og 4. Ved å fokusere mer helhetlig på milepælene, ville utgangspunktet for valg av rammeverk vært annerledes. Dette hadde gitt klarere retningslinjer for viktige funksjoner i valg av rammeverk. Ved å bruke disse retningslinjene, hadde det vært lettere å oppdage begrensninger ved de ulike rammeverkene. Det er ikke sikkert et annet rammeverk hadde løst nåværende utfordringer, men begrensningene ved valgt rammeverk hadde blitt oppdaget tidligere.

Et samarbeid med fagmiljøet innenfor microfrontend, ville vært til stor hjelp. Ved å aktivt benytte seg av kommunikasjonskanalene, kunne det i en tidlig fase kartlagt begrensningene ved hvert enkelt rammeverk.

7 KONKLUSJON OG VIDERE ARBEID

Gjennom denne oppgaven har det blitt utforsket muligheten for å lage en microfrontend-komponent som lett kan gjenbrukes for utviklere i både web og mobil. Rammeverket som er brukt for en mulig løsning er Single-SPA, og kapittelet tar for seg hva konklusjonen er, nytteverdien av arbeidet og videre arbeid.

7.1 Prosjektets mål

Prosjektets mål var å utforske om muligheten for en gjenbrukbar microfrontend-komponent lett kunne brukes av utviklere i web- og mobilapplikasjoner. Tanken var at denne komponenten skulle lages en gang, og implementeres som en egen komponent uten endringer for fremtidige prosjekter.

7.2 Måloppnåelse

Ved nedprioritering av design og funksjonalitet i komponenten, ble milepæl 1 og 2 er ferdigstilt. Ved å gjøre dette, var delmålene for disse milepælene oppnåelige, og står igjen med et produkt å vise til.

Når det kommer til milepæl 3 og 4, ble det støtt på større utfordringer. Som tidligere nevnt ble milepæl 3 nedprioritert da tidspresset ble for stort. Det er nevnt at en løsning med single-SPA og PWA er mulig, men det finnes ikke konkrete fremgangsmåter på hvordan dette gjøres (Denning, 2020a). Manglende dokumentasjon og mangel på kunnskap gjorde milepæl 3 svært krevende. Det gjenstår fortsatt arbeid til at en PWA fungerer med en microfrontend-komponent. Måloppnåelsen i milepæl 3 er dermed ikke oppnådd. Ved milepæl 4 er det kommet frem til en delvis løsning. Løsningen som er utarbeidet er en mobilapplikasjon som viser microfrontend-komponenten som et webview. Denne løsningen fungerer i både Android og iOS. Applikasjonene som er utarbeidet er svært enkle, og har lite fokus på design og funksjonalitet. Fokuset er rettet mot at konseptet fungerer for en slik løsning.

7.3 Nytteverdi

Arbeidet som er utredet i dette prosjektet kommer til god nytte for de som ønsker å utforske microfrontend-konseptet. Rapporten er en god start for å bli kjent med microfrontend og implementering av et microfrontend-rammeverk i sine systemer. Dette prosjektet baserer seg på rammeverket single-SPA. Dokumentasjon og begrunnelser på hvorfor de ulike valgene er tatt, hjelper godt for andre som ønsker å utforske problemstillingen. Valg blir begrunnet, og kan gi en pekepinn til andre som vil prøve samme fremgangsmåte, eller prøve på noe nytt. I tillegg gir rapporten et innsyn i hvordan dette miljøet fungerer opp imot en skylagringstjeneste. Selv om arbeidet i prosjektet ikke har funnet en løsning, vil dette være en løsning i seg selv. Det har blitt sett på muligheten ved ett rammeverk, og undersøkte at dette ikke var mulig for både web- og mobilapplikasjoner. Dette er et konkret svar for de som ønsker å utforske dette prinsippet.

7.4 Videre arbeid

I dette kapitlet vil videre arbeid bli beskrevet basert på følgende valg: om det skal undersøkes videre implementasjonen av single-SPA, eller om et nytt rammeverk skal undersøkes. Hvert av disse valgene vil bli beskrevet i kommende underkapitler.

7.4.1 Videre arbeid med single-SPA

Arbeidet blir å finne ut hvordan syntaks-feilmeldingene i PWA løses, og en annen løsning enn et webview i en mobilapplikasjon. I PWA må det utforskes hvordan PWA kan samarbeide med en rot-applikasjon i single-SPA, samtidig som applikasjonen kvalifiseres som en PWA. Rot-applikasjonen leser bare .js filer, slik at dette skaper konflikt mellom HTML-filene i hvert prosjekt.

I en mobilapplikasjon kan det undersøkes nærmere om det er mulig å komme rundt vindu problematikken. Single-SPA baserer seg på ting som kun er tilgjengelig i nettleseren, slik som window.location og window.history. Dette egner seg ikke for mobilapplikasjoner. Et alternativ som kan undersøkes, er hybride applikasjoner.

Ved et videre arbeid må Internet Explorer 11 støttes i nettleseren. I nåværende tilstand støtter ikke microfrontend-komponenten Internet Explorer 11. Her anbefales det å undersøke polyfilling for å komme rundt problemet. Problemet er løselig, men ikke funnet ut av ved prosjektets slutt; det finnes eksempler som fungerer, men dette ble ikke løst i denne oppgaven.

Bruken av moduler for eksportering av funksjonalitet må undersøkes. Hensikten med en slik modul, er at den kan ta seg av henting av data, autentisering og styling. Videre kan den eksportere sin funksjonalitet for at andre komponenter skal kunne importere denne. Ved å ta i bruk disse modulene, vil det forsterke gjenbrukbarheten, og konsistensen mellom applikasjonene. Eksempelvis kan brukerautentisering gjøres av en slik modul, og da være tilgjengelige for applikasjoner som importerer modulens funksjonalitet.

7.4.2 Valg av nytt rammeverk

Ved valg av nytt rammeverk, er det viktig å gjøre gode undersøkelser. Et godt innblikk i hvordan teknologien og rammeverket fungerer er essensielt for helhetlig forståelse av problemet. Ved å ha god kunnskap om teknologien og rammeverket, kan problemet som ikke ble løst i dette prosjektet, utforskes.

Et alternativt rammeverk til single-SPA for å løse problemstillingen, kan være BIT. Om BIT fungerer bedre enn single-SPA, er ikke lett å vite uten videre undersøkelse på rammeverket. Dette er i så fall et alternativ som er verdt å undersøke. BIT har god dokumentasjon og kan bruke hvilket som helst JavaScript-basert språk. I tillegg finnes det forum for å diskutere problemstillinger og kode for bruk av BIT.

8 REFERANSER

Amazon (u.å.), *What is Amazon S3?* Tilgjengelig fra: https://aws.amazon.com/s3/ (Hentet 23.05.21)

Andrew R. & LePage P. (2020) *Responsive web design basics.* Tilgjengelig fra: https://web.dev/responsive-web-design-basics/ (Hentet 25.05.2021)

Android developer (2021a) *Guide to app architecture*. Tilgjengelig fra: https://developer.android.com/jetpack/guide (Hentet 20.05.2021)

Android developer (2021b) *Core app quality*. Tilgjengelig fra: https://developer.android.com/docs/quality-guidelines/core-app-quality#ux (Hentet 20.05.2021)

Apple Inc. (2021) *Xcode.* Tilgjengelig fra: https://developer.apple.com/documentation/xcode/ (Hentet: 11.05.2021)

Atlassian (2021*) What is Agile?* Tilgjengelig fra: https://www.atlassian.com/agile (Hentet 28.03.2021)

Ateles (2019) *Hva er PWA eller Progressive Web Apps?* Tilgjengelig fra: https://no.ehandel.com/artikler/annonse-hva-er-pwa-eller-progressive-web-apps/474081 (Hentet: 19.04.2021)

BIT.dev (2021). *How bit Works?* Tilgjengelig fra: https://docs.bit.dev/docs/how-bit-works (Hentet: 14.05.2021)

Bragelien, J. J., Skjølsvik, T. & Voldsund, K. H. (2020). *Forretningsforståelse*. Oslo: Cappelen Damm Akademisk.

Chart.js (2021), *Creating a Chart*. Tilgjengelig fra: https://www.chartjs.org/docs/latest/ (Hentet 12.04.2021).

Cypress, (2021) *The web has evolved. Finally, testing has too.* Tilgjengelig fra: https://www.cypress.io (Hentet 16.05.2021)

DAZN (u.å.) *Where is DAZN available?* Tilgjengelig fra: https://www.dazn.com/en-US/help/articles/where-is-dazn-available (Hentet: 14.05.2021)

Denning, J. (2020a) *Is it possible to use this framework in a pwa application?* Tilgjengelig fra: https://github.com/single-spa/single-spa/issues/483 (Hentet: 27.05.2021)

Denning, J. (2020b) *Announcing single-spa@5.* Tilgjengelig fra: https://es.single-spa.js.org/blog/#step-five-test-it-out (Hentet: 02.06.2021)

Denning, J. (2021) *Single spa react not loading in IE11.* Tilgjengelig fra: https://github.com/single-spa/single-spa-react/issues/109 (Hentet 27.04.2021)

DNB (2021a) *DNB in brief.* Tilgjengelig fra: https://www.ir.dnb.no/about-dnb/dnb-brief (Hentet 26.03.2021)

DNB (2021b) *About Eufemia.* Tilgjengelig fra: https://eufemia.dnb.no/design-system/about (Hentet 30.03.21)

Entando (2020) *7 Successful Companies Using Micro Frontends.* Tilgjengelig fra: https://www.entando.com/page/en/7_successful_companies_using_micro_frontends_en_1?contentId=BLG2303&modelId=25 (Hentet: 13.05.2021)

Ethical Web (2021) *Ethical Web Development*. Tilgjengelig fra: https://www.ethicalweb.org/ (Hentet 26.03.2021)

Finn (2021). *Conceptual Overview.* Tilgjengelig fra: https://podium-lib.io/docs/podium/conceptual_overview (Hentet: 14.05.2020)

Gaunt, M (2021) *Service Workers: an Introduction.* Tilgjengelig fra: https://developers.google.com/web/fundamentals/primers/service-workers (Hentet 30.03.2021)

Geers, M. (2021*) What are Micro Frontends?* Tilgjengelig fra: https://micro-frontends.org/ (Hentet 30.03.2021)

Gillis, A. (2020) *Native app*. Tilgjengelig fra: https://searchsoftwarequality.techtarget.com/definition/native-application-native-app (Hentet 20.05.2021)

GitHub (2021) *Hello World.* Tilgjengelig fra: https://guides.github.com/activities/hello-world/ (Hentet: 23.03.2021)

Google Developers (2021a) *Everything you need to build on Android.* Tilgjengelig fra: https://developer.android.com/studio/features (Hentet: 23.03.2021)

Google Developers (2021b) *Lighthouse.* Tilgjengelig fra: https://developers.google.com/web/tools/lighthouse (Hentet: 19.05.2021)

Iddo, D. (2019) *Agile Development.* Tilgjengelig fra: https://medium.com/moodah-pos/agile-development-95cad3573abf (Hentet 28.03.2021)

JSON (u.å.) *Introducing JSON.* Tilgjengelig fra: https://www.json.org/json-en.html (Hentet 25.05.21)

Keizer, G. (2020) *Microsoft sets new support deadlines for IE11 and Edge.* Tilgjengelig fra: https://www.computerworld.com/article/3571442/microsoft-sets-new-support-deadlines-for-ie11-and-edge.html (Hentet: 02.06.2021)

Kumar, P. (2018) *Json Server (json-server).* Tilgjengelig fra: https://www.journaldev.com/10660/JSON server (Hentet: 11.05.2021)

LePage, P. & Richard, S. (2020) *What are Progressive web apps?* Tilgjengelig fra: https://web.dev/what-are-pwas/ (Hentet 30.03.21)

Mezzalira, L. (2019a) *Adopting a Micro-Frontend architecture.* Tilgjengelig fra: https://lucamezzalira.com/2019/04/08/adopting-a-micro-frontends-architecture/ (Hentet: 13.05.2021)

Mezzalira, L. (2019b) *Orchestrating micro-frontends.* Tilgjengelig fra: https://medium.com/dazn-tech/orchestrating-micro-frontends-a5d2674cbf33 (Hentet: 13.05.2021)

Microsoft (2021a) *IntelliSense.* Tilgjengelig fra: https://code.visualstudio.com/docs/editor/intellisense (Hentet: 23.01.2021)

Microsoft (2021b) *Using React in Visual Studio Code*. Tilgjengelig fra*:* https://code.visualstudio.com/docs/nodejs/reactjs-tutorial (Hentet: 23.01.2021)

Mozilla Developers Network (u.å) *Polyfill.* Tilgjengelig fra: https://developer.mozilla.org/en-US/docs/Glossary/Polyfill (Hentet 31.05.2021)

Myers, B. (2020) *The Strengths and Benefits of Micro Frontends.* Tilgjengelig fra: https://www.toptal.com/front-end/micro-frontends-strengths-benefits (Hentet: 09.04.2021)

Netinbag (u.å) *Hva er heuristisk evaluering?* Tilgjengelig fra: https://www.netinbag.com/no/internet/what-is-a-heuristic-evaluation.html (Hentet 27.05.2021)

NPM (u.å.a) *React Minimal Pie Chart.* Tilgjengelig fra: https://www.NPMjs.com/package/react-minimal-pie-chart (Hentet 12.04.2021)

NPM (u.å.b) *Axios.* Tilgjengelig fra: https://www.npmjs.com/package/axios (Hentet 31.05.2021)

Pandit, N. (2021) *What and why React.js.* Tilgjengelig fra:

https://www.c-sharpcorner.com/article/what-and-why-reactjs/ (Hentet 30.03.2021)

Posnick J. (2021) *Beyond SPAs: alternative architectures for your PWA.* Tilgjengelig fra: https://developers.google.com/web/updates/2018/05/beyond-spa (Hentet: 14.05.2021)

Preece J, Rogers Y & Sharp H. (2019) *Interaction Design, beyond human-computer interaction*. Indianapolis, Indiana: John Wiley & Sons

Rehkopf, M. (u.å) *What is continious integtation?* Tilgjengelig fra: https://www.atlassian.com/continuous-delivery/continuous-integration (Hentet 01.06.2021)

Single-SPA (u.å.a) *Architectural Overview.* Tilgjengelig fra:

https://single-spa.js.org/docs/getting-started-overview#architectural-overview (Hentet 12.02.2021)

Single-SPA (u.å.b) *Getting Started.* Tilgjengelig fra:

https://single-spa.js.org/docs/getting-started-overview/ (Hentet 12.02.2021)

Single-SPA (u.å.c) *Testing Options.* Tilgjengelig fra:

https://single-spa.js.org/docs/testing/e2e/#testing-options (Hentet 16.05.2021)

Single-SPA (u.å.d) *Migrating an Existing React Project.* Tilgjengelig fra: https://single-spa.js.org/docs/4.x/migrating-react-tutorial/ (Hentet 23.04.2021)

Single-SPA (u.å.f) *Utility modules (styleguides, API, etc).* Tilgjengelig fra: https://single-spa.js.org/docs/recommended-setup/#utility-modules-styleguide-api-etc (Hentet 20.04.2021)

Travis CI (u.å.) *Core concept for beginners.* Tilgjengelig fra: https://docs.travis-ci.com/user/for-beginners/#what-is-continuous-integration-ci (Hentet 19.05.2021)

Trello (2021) *What is Trello?* Tilgjengelig fra: https://help.trello.com/article/708-what-is-trello (Hentet: 19.03.2021).

W3 Schools (u.å.) *JSON – Introduction.* Tilgjengelig fra:
https://www.w3schools.com/js/js_json_intro.asp (Hentet 25.05.2021)

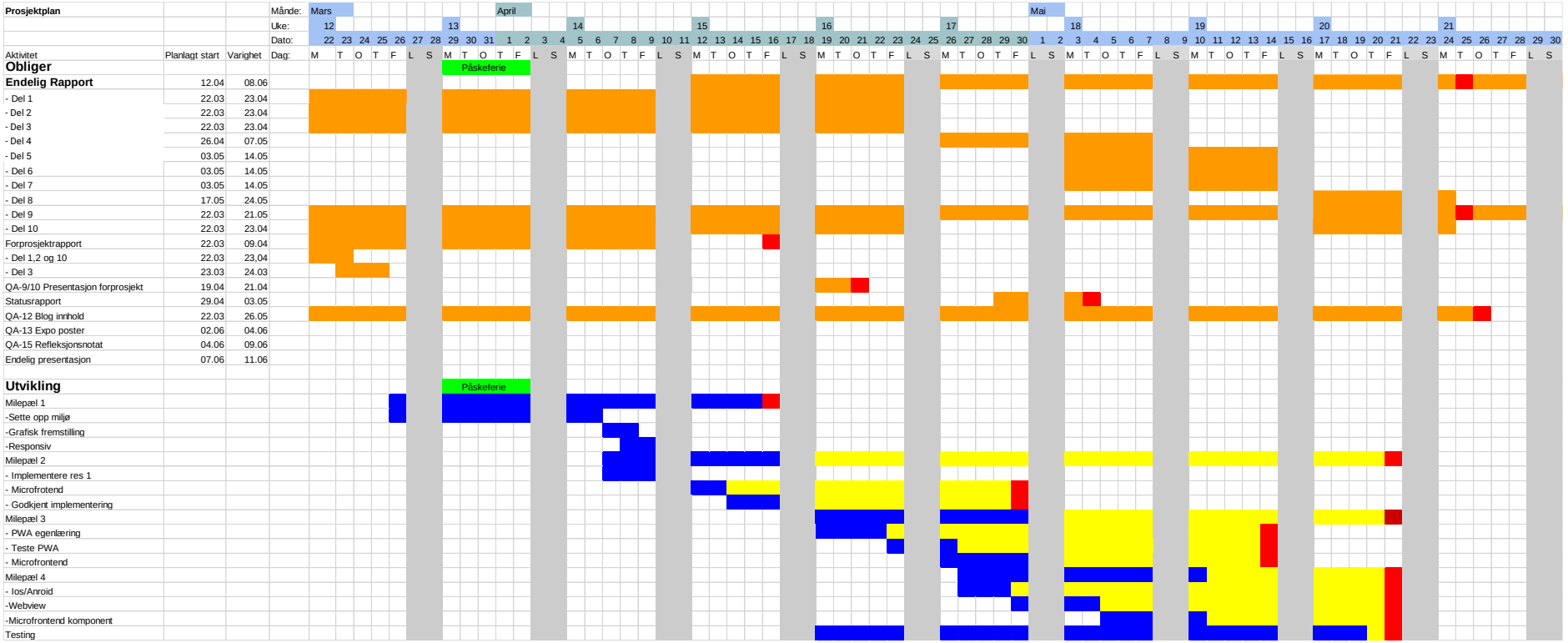
9 APPENDIX

9.1 Risikoliste

Hva	Beskrivelse	Risiko	S	K	RF	Tiltak	Ansvar
Obligatoriske oppgaver	Tidsperspektiv	Tar lengre tid enn antatt	4	2	**8**	Planlegge god tid til besvarelse av obligatoriske oppgaver	Gruppen
	Dårlig arbeid	Må bruke tid på å levere på nytt	2	3	**6**	Sørge for felles gjennomgang av alle obligatoriske øvinger for innlevering	Gruppen
Utvikling	Teknisk kunnskap	Manglende teknisk kunnskap kan føre til lite fremgang i utviklingen	4	3	**12**	Sette av tid til egenstudie av aktuele teknologier gjennom hele prosjektet. Fylle på med kunnskap i tråd med hvilke teknologier som skal implementeres. Gruppen har også fått tekniske veiledere fra oppdragsgiver som kan hjelpe.	Gruppen
	Milepæler	Undervurdering av milepælers omfang og kompleksitet	3	4	**12**	Opparbeide seg faglig kompetanse om aktuell teknologi og problemstillinger	Gruppen
	Tilgang til systemer	Får ikke implementert ønskede funksjoner i oppdragsgivers systemer	2	3	**6**	Dialog med oppdragsgiver for å tidligst mulig få tilgang til deres systemer	Gruppen og oppdrags-giver
	Misforståelse av oppgaven	Oppgaven blir ikke løst i tråd med oppdragsgivers krav og spesifikasjoner	2	3	**6**	Ukentlige møter med oppdragsgiver der man diskuterer fremgang og planen videre.	Gruppen og oppdrags-giver
Rapport	Utarbeiding	Klarer ikke å følge planen for skriving parallelt med det tekniske	2	4	**8**	Sette av tid til kontinuerlig skriving på rapporten for å unngå en lang skriveøkt på slutten av oppgaven.	Gruppen
Andre risikoer	Sykdom/pandemi	Mer arbeid på friske medlemmer og økt arbeidsmengde.	2	4	**8**	Gruppemedlemmer må ha en generell oversikt over pågående og fremtidige oppgaver som blir eller skal bli løst.	Gruppen
	Risikoer det ikke er tatt høyde for	Utsettelse i prosjektplan, uforutsette endringer i tidsestimat	5	3	**15**	Hyppig revurdere konsekvenser av risikoer. Ta høyde for at man ikke kan planlegge og forutse alle risikoer. Felles forståelse i gruppen om at bekymringer/utfordringer tas opp med en gang de oppstår.	Gruppen

Tabell 1: Risikohåndteringsplan

9.2 GANTT-skjema

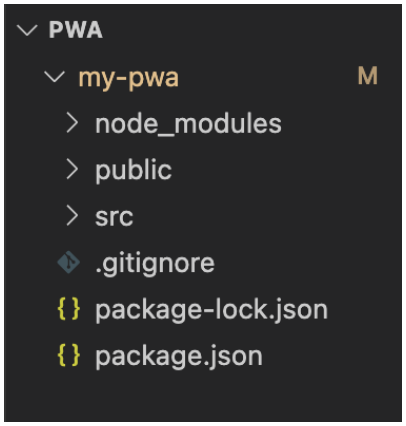


Tabell 2: Oppdatert GANTT-skjema

9.3 Brukermanualer

9.3.1 Fremgangsmåte PWA

1. Start med å lage et react prosjekt. Dette kan gjøres ved å skrive inn denne kommandolinjen i terminalen: "npx create-react-app my-pwa". Dersom det ønskes et ferdiglaget PWA prosjekt, kan man skrive denne kommandolinjen i terminalen: "npx create-react-app my-pwa --template cra-template-pwa". For denne gjennomgangen lages det en PWA applikasjon manuelt.
    a. Npx er et verktøy for å kjøre npm pakker.
    b. create-react-app my-pwa lager et react prosjekt med navnet my-pwa
    c. Strukturen vil se noe slikt ut:

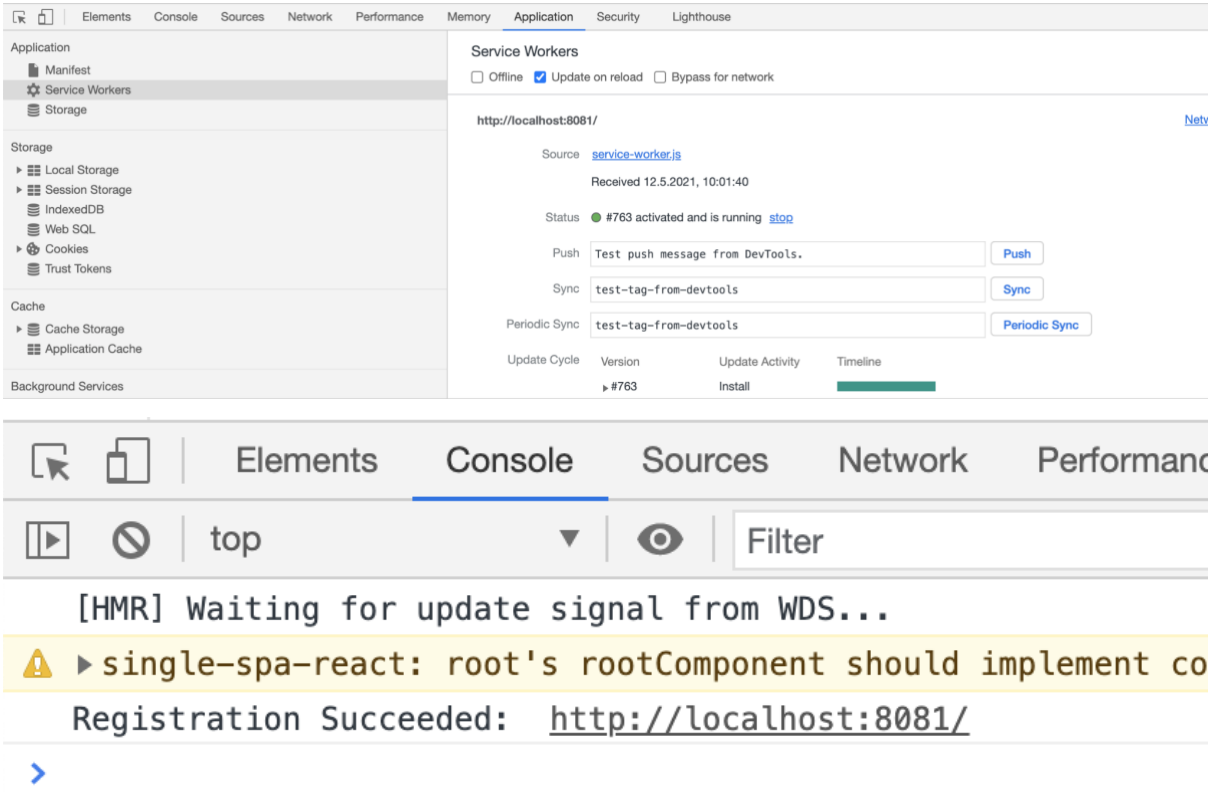


2. Neste steg er å registrere en service worker. I mappen som heter public, ligger det en fil som heter index.html. Naviger til denne filen og legg til dette scriptet i html-filens body:

```
<script>
  if('serviceWorker' in navigator) {
      window.addEventListener('load', () => {
          navigator.serviceWorker.register('./service-worker.js')
              .then((reg) => console.log('Registration Succeeded: ', reg.scope))
              .catch((err) => console.log('Registration Failed: ', err));
      })
  }
</script>
```

Dersom service workeren registreres, vil det i konsollen skrives ut “Registration succeeded”. Dersom registreringa av service workeren feiler, vil konsollen skrive ut “Registration Failed”.

3. For å se om service workeren har blitt registrert i nettleseren, kan applikasjonen startes ved npm start, og inspisere nettsiden i Google Chrome. Klikk videre til fanen som heter "Application". Da vil det synes om service workeren er registrert eller ikke. I tillegg blir det skrevet ut i konsollen om registreringen feilet eller ikke.



4. Service workeren som registreres, finnes i filen som heter service-worker.js. Neste steg er å lage denne filen. Lag en ny js fil i mappen public som heter service-worker.js. I denne filen legges inn all logikk som brukes for å lage en fungerende service worker. I hovedtrekk er det tre metoder som blir brukt i en service worker: installer, aktiver og hent.

a. Installer → Legger til alle ressursene som skal caches.

```
const precacheVersion = 1;
const precacheName = 'precache-v' + precacheVersion;
const precacheFiles = [
  'offline.html',
  '/images/favicon.ico',
  '/images/maskable.png',
  '/images/logo192.png',
  '/images/logo256.png',
  '/images/logo384.png',
  '/images/logo512.png',
  '/images/apple-touch-icon.png',
  '/images/favicon-16x16.png',
  '/images/favicon-32x32.png',
];

self.addEventListener('install', (e) => {
  console.log('[ServiceWorker] Installed')

  self.skipWaiting();

  e.waitUntil(
    caches.open(precacheName).then((cache) => {
      console.log('[ServiceWorker] Precaching files')
      return cache.addAll(precacheFiles);
    }) // end cacheOpen()
  ) // end e.WaitUntil()
});
```

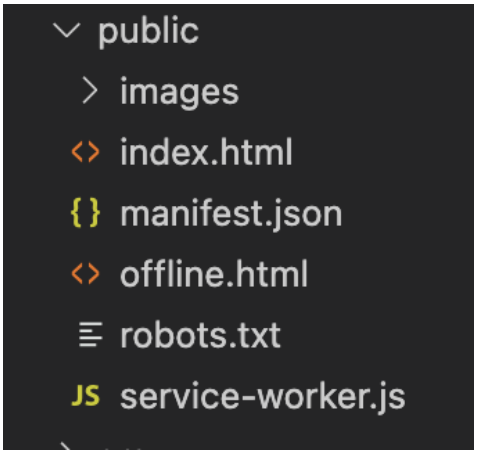
b. Aktiver → Oppdaterer cachen dersom en ny cache har blitt oppdaget og sletter den gamle

```
self.addEventListener('activate', (e) => {
  console.log('[ServiceWorker] Activated')

  e.waitUntil(
    caches.keys().then((cacheNames) => {
      return Promise.all(cacheNames.map((thisCacheName) => {

        if (thisCacheName.includes("precache") && thisCacheName != precacheName) {
          console.log('[ServiceWorker] Removing cached files from old cache - ' + thisCacheName)
          return caches.delete(thisCacheName)
        }
      }))
    }) //end caches.keys()
  ) //end e.waitUntil()
})
```

c. Hent → Henter den responsen som blir sendt tilbake. Dersom forespørselen får en respons tilbake, henter den responsen. Dersom forespørselen ikke får noe respons, vil det da lastes inn en egendefinert offline.html side som er cachet, som fungerer når brukeren ikke har tilgang til internett.

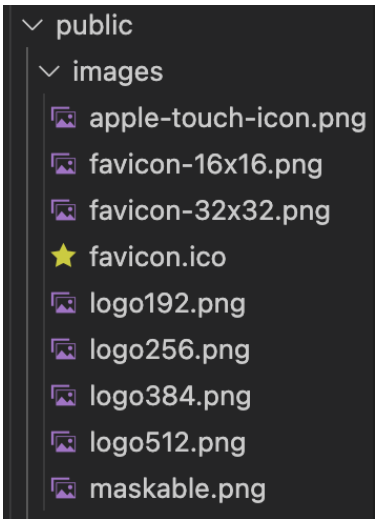
```
self.addEventListener('fetch', (e) => {
  console.log('[ServiceWorker] Fetch event for ' + e.request.url)

  e.respondWith(
    caches.match(e.request).then((cachedResponse) => {

      if(cachedResponse) {
        console.log("Found in cache!")

        return cachedResponse;
      }

      return fetch(e.request)
      .then((fetchResponse) => fetchResponse)
      .catch((err) => {
        const isHTMLPage = e.request.method === "GET" && e.request.headers.get("accept").includes("text/html");

        if (isHTMLPage) return caches.match('offline.html')
      })
    })
  ) //end e.responWith()
})
```

5. Videre konfigureres det en manifest.json fil i mappen public. Denne filen inneholder informasjon om applikasjonen og gjør det mulig å installere nettsiden som en applikasjon. Filen inneholder informasjon om hva applikasjonens navn er, en beskrivelse av applikasjonen, styling, hvilken url appen skal kjøres på og hvilket ikon som blir brukt til applikasjonen. For logo må man laste ned ikon som er lik størrelsen lik manifest.json filen. Opprett en manifest.json fil i mappen public:



```
{
  "short_name": "Graph",
  "name": "Graph",
  "icons": [
    {
      "src": "./images/maskable.png",
      "type": "image/png",
      "sizes": "192x192",
      "purpose": "any maskable"
    },
    {
      "src": "./images/favicon.ico",
      "sizes": "64x64 32x32 24x24 16x16",
      "type": "image/x-icon"
    },
    {
      "src": "./images/logo192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "./images/logo512.png",
      "type": "image/png",
      "sizes": "512x512"
    },
    {
      "src": "./images/logo384.png",
      "type": "image/png",
      "sizes": "384x384"
    },
    {
      "src": "./images/logo256.png",
      "type": "image/png",
      "sizes": "256x256"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#ffffff"
}
```

6. Legg merke til at det finnes en mappe med navn images. Her ligger det bilder som brukes som ikon for applikasjonen. Disse bildene brukes i manifest.json filen. Src attributten sier noe om hvor bildet blir hentet fra. Type beskriver hvilken type bildet er. Size sier noe om størrelsen til det angitte ikonet. Bildene har ulike størrelser ut fra hvor et ikon brukes. For at en applikasjon skal fungere som en PWA, må manifest filen inneholde et maskable bilde. Dette er et bilde som kan forme seg etter bruk av ikon, som et rektangel, en sirkel, oval form eller et kvadrat. Bildene må formateres i riktig størrelse.



7. Når dette er gjort må manifest.json filen legges til i index.html siden. Dette gjøres ved å legge til en <link> tag, og legge til en relasjon der kilden til bilde ligger. I tillegg til manifest filen, må det også legges til en <link> tag til ikonet som skal brukes for applikasjonen og en <link> tag til iOS brukere. Grunnen til at iOS brukere må ha egen referanse til bilder, er fordi brukerne kan manuelt legge til PWA på sine enheter.

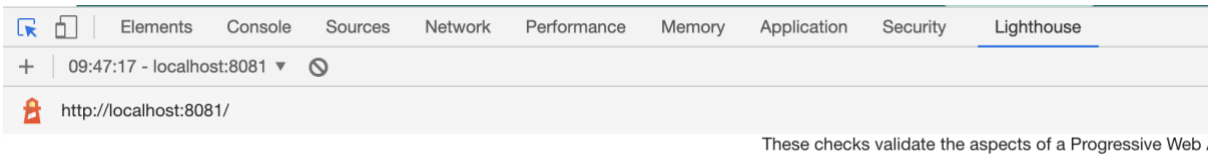
```
<link rel="apple-touch-icon" href="./images/apple-touch-icon.png" />
<link rel="shortcut icon" href="./images/favicon.ico" />
<link rel="manifest" href="./manifest.json" />
```

8. Når dette er gjort trenger man noen <meta> tags for å oppfylle kravene for at en applikasjon skal være en PWA. Dette er blant annet fargen som skal vises i bakgrunnen når applikasjonen kjører som en app via nettleseren og at applikasjonen er responsiv etter hvilken enhet man bruker.

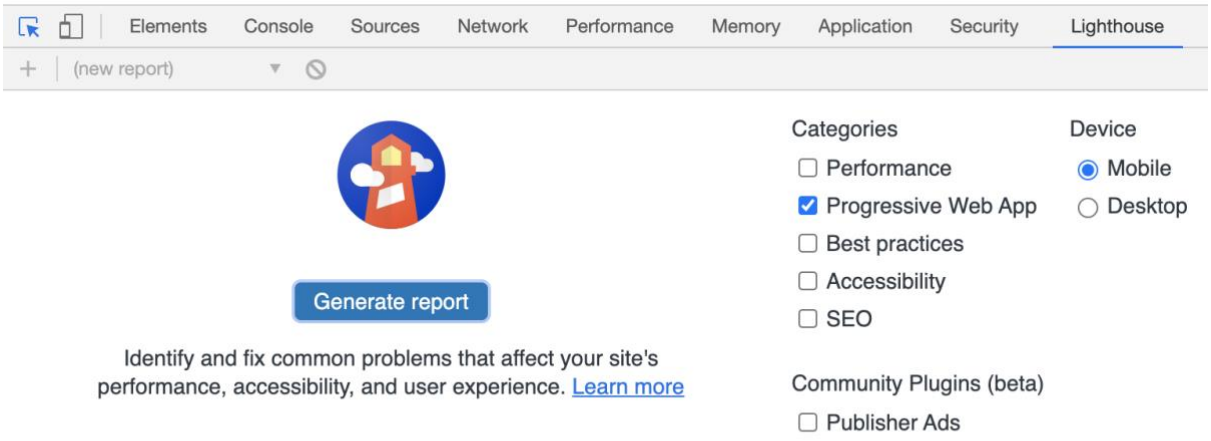
```
<meta name="viewport" content="width=device-width, initial-scale=1" />
<meta name="theme-color" content="#000000" />
<meta name="description" content="Graph React Application" />
```

9. Et verktøy som er veldig behjelpelig, er Google Chrome utvidelsen Lighthouse. Med denne utvidelsen kan man scanne siden sin og få opp en sjekkliste på hva som trengs for å oppfylle kravene for at en applikasjon skal være en PWA. For å installere Lighthouse, søk på “Lighthouse extension” i adressefeltet. Klikk på den første linken og legg denne til i chrome. Når dette er gjort, gå til siden du kjører applikasjonen din på. Iinspiser nettsiden

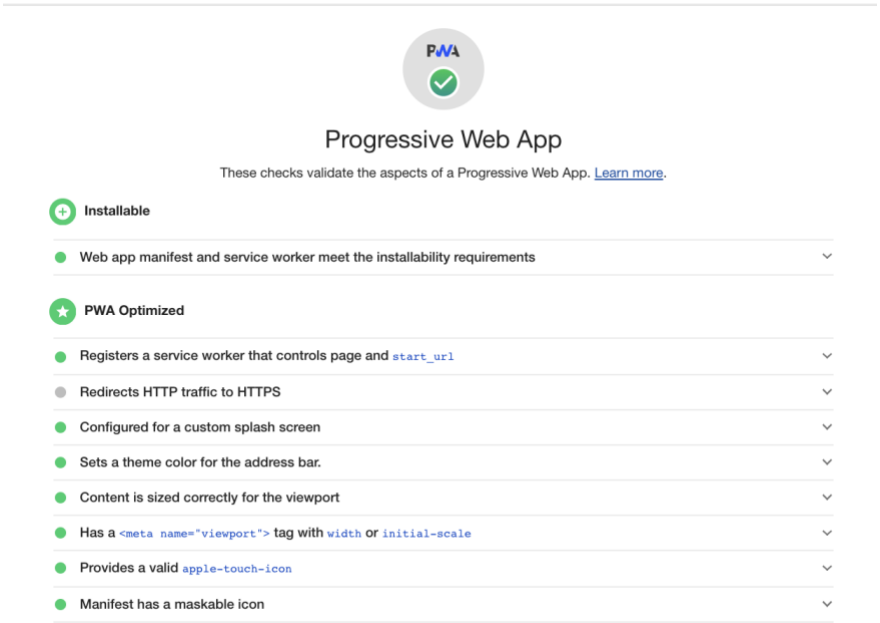
og klikk på fanen som heter Lighthouse.



10. Videre huker man av boksen med “Progressive Web App” og hvilken enhet det skal testes på: mobil eller Desktop. Deretter klikker man på Generate report.



11. Når rapporten er ferdig generert, får man opp en sjekkliste på en PWA.



12. Når dette er oppe å kjører, og applikasjonen er kvalifisert for en PWA, er neste steg å integrere microfrontend-miljøet i PWA'en. Første steg er å gjøre PWA applikasjonen om til en single-spa applikasjon. Dette gjøres ved å legge til

Manifest.Json

Hovedattributtene i en manifest.json fil er:

- **start_url:** Forteller nettleseren hvor applikasjonen skal **kjøres** fra når den blir kjørt.
- **short_name eller name:** Gir et navn på applikasjonen. Short_name blir brukt til å vise navnet på applikasjonen på skrivebordet. Name blir brukt når man installerer applikasjonen.
- **icon:** Definerer hvilket ikon som skal vises på skrivebordet, i nettleser eller ved bytting av applikasjoner. Icon er en array, og må inneholde disse attributtene:
    - **src:** kilden til der ikonet blir hentet
    - **sizes:** størrelsen på ikonet som blir brukt
    - **type:** forteller hvilen type bilde det er (jpg, png osv.)
    - **purpose:** hvilken kontekst bildet skal brukes
- **background_color:** Blir brukt som bakgrunnsfarge i applikasjonen når man først har startet den.
- **display:** hvordan applikasjonen vises. Dette kan være i form av fullskjerm, i nettleseren enten med eller uten adresselinje eller kjøres som eget vindu.
- **theme_color:** setter fargen på verktøylinjen av applikasjonen.
- **description:** En beskrivlsem applikasjonen
- **screenshots:** En array med bilder som viser bruken av applikasjonen. Hvert objekt inneholder src, sizes og type.

9.3.2 Fremgangsmåte Single-SPA

How-to guiden baserer seg på dokumentasjon fra https://single-spa.js.org/docs/create-single-spa. Guiden tar trinnvis for seg opprettingen av en ny applikasjon, migrering av eksisterende prosjekt til single-SPA, import av app og delte dependencies, lokal utvikling, eksterne import-maps, hvordan opprette komponenter og implementasjon av tredjeparts biblioteker.

Opprette rot og app

- For rooten/containeren:
  $ npx create-single-spa --moduleType root-config
    - følg instruksene gitt i terminalen
    - Organisasjonsnavnet må være likt i alle applikasjonene, siden dette blir brukt som namespace
- For applikasjonen/komponenten
  $ npx create-single-spa --moduleType app-parcel
    - NB: organisasjonsnummer må være det samme som i roten

Man står nå igjen med en del automatisk generert kode:

Root

microfontend.layout.html

Root-filen/Containeren er den eneste med egen HTML-fil. Denne er delt mellom alle single-SPA-applikasjoner og har ansvaret for å rendre siden. Applikasjonene legges inn her basert på navn fra import-maps (se avsnitt om delte dependencies og app). Dokumentet inneholder en single-SPA-ruter, og man kan velge å legge applikasjoner på samme side/rute, eller på forskjellige.

Eksempel der en applikasjon er lagt til som en nav-bar, og en er lagt til i ruten/page-1. Path spesifiserer ruten.

```
  <single-spa-router></single-spa-router>
  <nav>
    <application name = "@myOrg/navbar"></application>
  </nav>
  <main>
    <route path="/page-1">
      <application name="@myOrg/my-project"></application>
    </route>
  </main>
</single-spa-router>
```

index.ejs

Her ligger import-map. Dette peker på applikasjonens URL til der den kjøres (for eksempel localhost ved lokal utvikling, eller et eksternt import-map hostet i en AWS S3-bucket). Import-map muliggjør in-browser-moduler ved å kunne bruke alias der man peker på en URL.

test-org-root-config.js

Her registreres applikasjoner fra import-map ved bruk av metoden registerApplications(). Bruker en layout-engine for å gjøre det lettere å blant annet plassere DOM-elementer og ordne applikasjoner.

App

root.component.js

Som app.js i et vanlig react prosjekt. Denne som eksporteres fra test-org-my-app.

test-org-my-app.js

Dette er komponenten som tar seg av livssyklus-funksjonene. Av de obligatoriske finnes bootstrap, mount og unmount. Disse eksporteres i automatisk generert kode fra create-single-spa. Disse funksjonene er mulige å endre på, ved å for eksempel spesifisere et prefiks match for når komponenten skal være festet. Eksempelvis kan dette være når man befinner seg på sider som inkluderer /home. Eksakt spesifisering er også mulig.

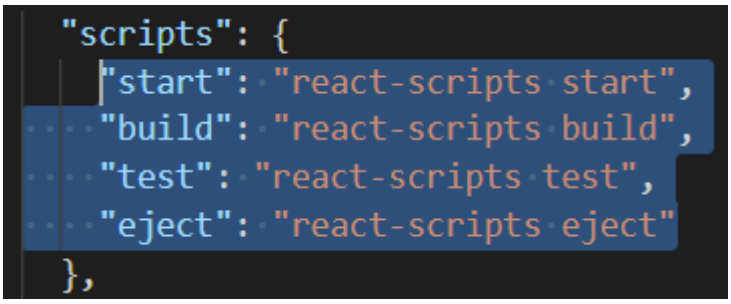
Likheter mellom root-config og app

Begge har egen webpack-config og package.json.

Gjøre om eksisterende React-prosjekt til single-SPA

Utfordringer: html-fil og web-pack-config → virke med SystemJs og Single SPA. Kan ikke endre denne (skjult), derfor bygge på nytt.

1. Slette alle script i package.json:

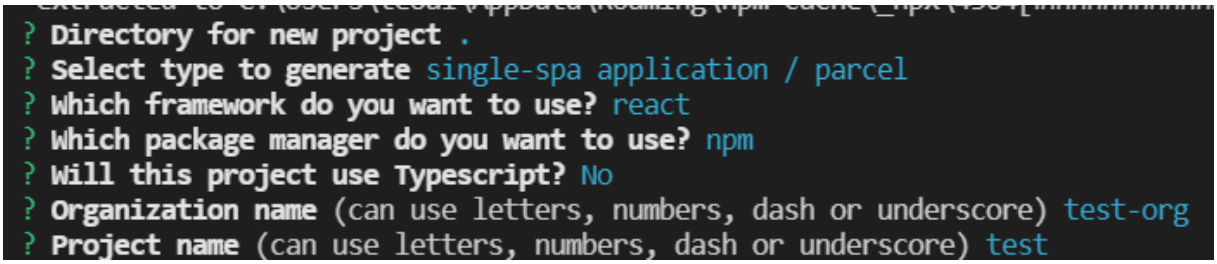


2. Slett Dependencies (trenger kun slette de som kommer med create react-app)

3. Kjør create-single-spa oppå eksisterende prosjekt

```
npx create-single-spa
```

Svar på navn ++:

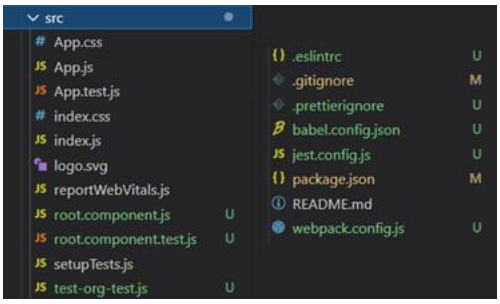


4. Override package.json

```
Initialized git repository

 conflict package.json
? Overwrite package.json? overwrite
    force package.json
   create jest.config.js
   create babel.config.json
   create .eslintrc
 conflict .gitignore
? Overwrite .gitignore? overwrite
    force .gitignore
   create .prettierignore
   create webpack.config.js
   create src\root.component.js
   create src\root.component.test.js
   create src\test-org-test.js
```

- Create-single-spa gjør endringer i package.json (fletter package.json)

5. Slett public-mappa (trenger ikke HTML-fil)
6. Opprettes nye filer nødvendig for single-SPA



7. Sletter index.js (gammelt entry-point), beholder <org-name>-<name>.js som nytt entry-point

- Importerer eventuell css her

8. Root-component erstatter App.js (kan også velge å beholde App.js)
9. Har nå en web-pack-config
    - Må kunne litt om webpack
    - Får en default
        - Merge: se implementering av tredjepartsbiblioteker

Kan også bruke react-app-rewired eller eject, men anbefalte metode er å kjøre create-single-spa som beskrevet over.

Importere app og delte dependencies i roten

Dependencies

Neste steg er å importere appen og delte dependencies i containeren. Single-SPA bruker her noe som kalles import-maps. Denne finnes som en <script>-tag i rotens index.ejs:

<script type="systemjs-importmap">. Her er allerede single-SPA lagt til, men vi må også legge til typiske delte dependencies som React og React DOM.

React og React DOM i container sin index.ejs

```
<script type="systemjs-importmap">
  {
    "imports": {
      "single-spa": "https://cdn.jsdelivr.net/npm/single-spa@5.9.0/lib/system/single-spa.min.js",
      "react": "https://cdn.jsdelivr.net/npm/react@16.13.1/umd/react.production.min.js",
      "react-dom": "https://cdn.jsdelivr.net/npm/react-dom@16.13.1/umd/react-dom.production.min.js"
    }
  }
</script>
```

App

Her bruker man organisasjonsnavn og app navn. Dette finnes i Apps package.json:

```
"name": "@test-org/my-app",
```

Navnet på komponenten vil være uavhengig av om man utvikler lokalt eller via feks en AWS S3-bucket. Forskjellen ligger i hvor man henter import-mappet fra:

I index.ejs finnes det et import-map for lokal, og et for utvikling via feks. git-repo. Ved eksterne import maps, vil det kun ligge en linkreferanse til en importmap. Fordelen med dette er at containeren alltid henter de nyeste versjonene av komponentene, uten at dette trenger å endres manuelt; oppdateres on the fly.

Utvikle lokalt

For lokal utvikling legges alle tre også inn i scriptet med if(local) som argument. Man må også enten spesifisere porten i package.json, eller bruke allerede eksisterende port.

Dependencies og @test-org/my-app lagt inn i lokal import-map:

```
<% if (isLocal) { %>
<script type="systemjs-importmap">
  {
    "imports": {
      "@single-spa/welcome": "https://unpkg.com/single-spa-welcome/dist/single-spa-welcome.js",
      "@test-org/root-config": "//localhost:9000/test-org-root-config.js",
      "react": "https://cdn.jsdelivr.net/npm/react@16.13.1/umd/react.production.min.js",
      "react-dom": "https://cdn.jsdelivr.net/npm/react-dom@16.13.1/umd/react-dom.production.min.js",
      "@test-org/my-app": "//localhost:8080/test-org-my-app.js"
    }
  }
</script>
<% } %>
```

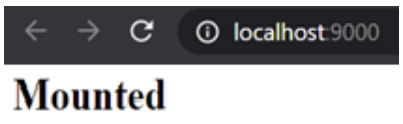
Ved å kjøre NPM start i begge prosjektene, skal du nå kunne hoste appen i containeren. Hva som er output avgjøres i applikasjonens root.component.js:

```
import React from "react";


export default function Root() {
  return (
    <section>
        <h1>Mounted</h1>
    </section>
  );
}
```

Selve utførelsen skjer i test-org-my-app.js, der Root importeres fra nevnte root.component:

```
import React from "react";
import ReactDOM from "react-dom";
import singleSpaReact from "single-spa-react";
import Root from "./root.component";


const lifecycles = singleSpaReact({
  React,
  ReactDOM,
  rootComponent: Root,
});

export const { bootstrap, mount, unmount } = lifecycles;
```

Resultatet i nettleseren etter å ha kjørt NPM start i containeren og applikasjonen:



Eksterne import-maps

Ved å bruke eksterne import-maps, er utformingen av selve import-mappet samme; man bruker et alias, som tidligere, men istedenfor å referere til porter, refererer man til plasseringen til import-map via en URL.

importmap.json's utforming*:*

Eksempel i index.ejs:

```
<script type="systemjs-importmap"
src="https://single-spa-test.s3.us-east-2.amazonaws.com/%40org/importmap.json">
</script>
```

Fordelen med eksterne import-maps er at dette ikke ligger i hvert enkelt GitHub-repo, men som et felles delt import-map. Endringer i en applikasjon vil automatisk bli oppdatert i import-map, og dermed også i de andre applikasjonene.

Opprette komponent(er)

Oppretting av komponenter), skjer som i et hvilket som helst React-prosjekt. Komponenten må importeres i app sin root-component.js, inne i <section>-taggen. Legg merke til at komponentene må har stor forbokstav når de importeres og eksporteres

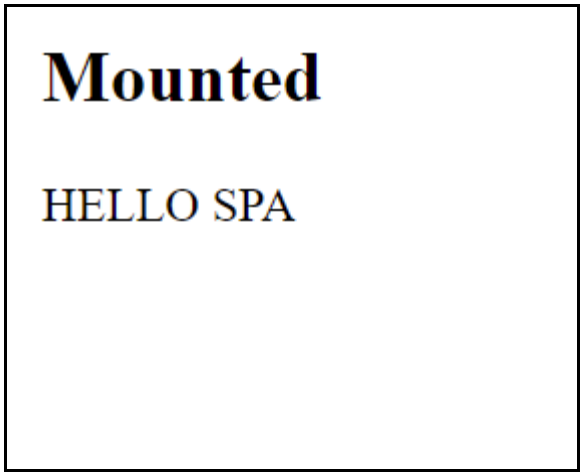
Eksempel:

Oppretter en ny JavaScript-fil: Hello-SPA, og lar denne returnere en med HELLO SPA og henter denne i root.component.js:

```
import React from 'react';

export default function HelloSPA () {
    return (
        <div>
        HELLO SPA
        </div>
    );
}
```

```
import React from "react";
import Hello from "./HelloSPA";


export default function Root() {
  return (
    <section>
        <h1>Mounted</h1>
        <Hello/>
    </section>
  );
}
```

Resultat:



Implementering av tredjeparts biblioteker: Eufemia

Installer file-loader og dnb-ui-lib i My-App:

$ npm install dnb-ui-lib

$npm install file-loader

Legg til webpack loader i MyApp/webpack.config:

Metoden merge lar oss modifisere webpack-config

```
module: {
  rules: [
    {
      test: /\.m?js/,
      resolve: {
        fullySpecified: false
      }
    }, {
      test: /\.(woff(2)?|ttf|eot|svg)(\?v=\d+\.\d+\.\d+)?$/,
      use: [
        {
          loader: 'file-loader',
          options: {
            name: '[name].[ext]',
            outputPath: 'fonts/'
          }
        }
      ]
    }

  ]
}
```

Importere Button og tilhørende styling i My-App/src/Chart.js:

```
import "dnb-ui-lib/style/basis";
import "dnb-ui-lib/style/components";
import "dnb-ui-lib/style/themes/ui";
import { Button } from "dnb-ui-lib";
```

Enkel funksjonalitet for knappen:

```
export default function Hello () {
    return (
        <div>
        <h1>HELLO SPA</h1>
        <Button
            text="Hello"
        />
        </div>
    );
}
```

Knappen fra eufemia med riktig design:

Mounted

HELLO SPA

Hello