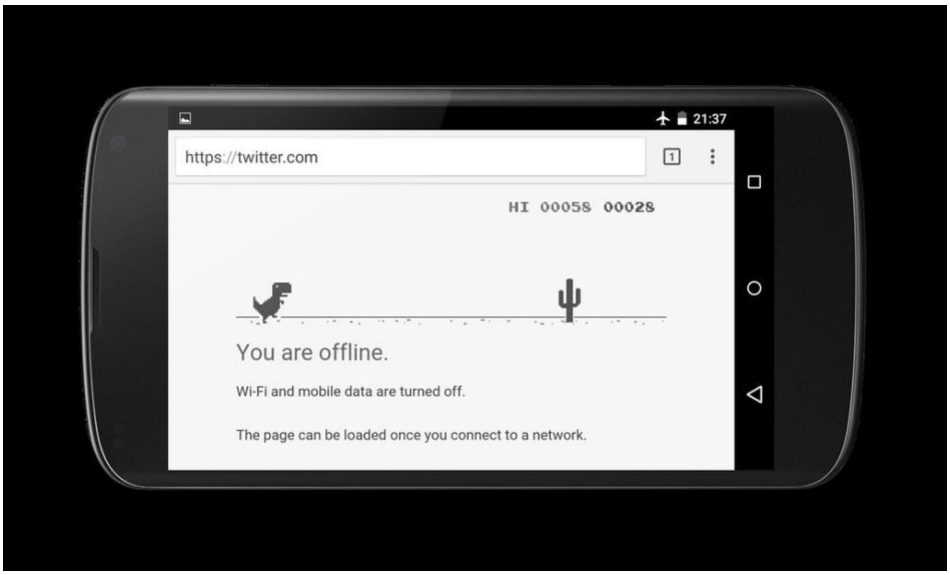
Astrid Amalie Storflor Hegge

Knut Yngve Barstad Gjelle

Offline Webapp; kun et trendord eller banebrytende teknologi?



Bilde 1 Illustrasjon av offline modus i Chrome (Agarwal, 2019).

Veileder: Atle Nes
Oppgavenummer: 66
Mai 2019
Bacheloroppgave i Digital Forretningsutvikling

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU
Kunnskap for en bedre verden

Sammendrag

Denne rapporten belyser teknologier som gir brukeren av mobile plattformer en bedre opplevelse uten dekning. Store deler av verden har mangelfull og ustabil mobildekning (Downs, 2017). For at webapper skal kunne konkurrere med native mobilapper må de gi brukeren en god opplevelse også når man mangler dekning. Rapporten er utarbeidet etter et ønske fra IT Data AS.

Rapporten er utarbeidet med bruk av metoden litteraturstudie og med innslag fra designvitenskap. Som en del av arbeidet med ulike teknologier har vi jobbet med veiledninger og guider om offline funksjonalitet. Denne blandingen i tilnærmelser har gitt oss god innsikt i utviklingen av offline webapper.

Vi fant to hovedutfordringer ved å kunne få en webapp til å virke offline. For det første må man lagre de individuelle websidene og systemressursene de trenger for å fungere. I tillegg må også informasjon brukeren lagrer eller manipulerer, tas vare på. Den første utfordringen kan møtes ved å installere en service worker. Dette JavaScriptet jobber i bakgrunnen av nettstedet, separat fra resten av nettsiden. Lagring av informasjon blir løst ved å lagre denne i en lokal NoSQL database. Vi har evaluert flere databaser av denne typen. Webstandarden IndexedDB, IndexedDB med Dexie wrapper (innpakning), PouchDB og Cloudant for å nevne noen. For å se hvordan de forskjellige teknologiene samspiller konstruerte vi også et konseptbevis, en Offline Web App (POOWA). POOWA simulerer en handleliste og lar deg synkronisere varer mellom ulike nettlesere og enheter.

Vi var på utkikk etter å finne teknologier som gjør at webapper kan konkurrere mot mobilapper på steder med dårlig dekning. Vi har funnet ut at service worker API har en essensiell rolle i dette. Vår POOWA (Proof of Offline Web App) viser en mulig løsning og bekrefter at man kan lage en webapp som gir samme brukeropplevelse som en mobilapp.

Abstract

This rapport aims to illuminate different technologies that improve user experience in areas with poor data coverage. Large parts of the world have inadequate data coverage (Downs, 2017). For web apps to compete with their native counterparts, they need to consider what happens when a user has bad data coverage. The rapport originates from a request made by IT Data AS.

The rapport was developed using a combination of a literature search and guidelines from design science. To further explore different web technologies that give us offline capabilities we have followed tutorials to test the technologies feasibility. Together, this mix of a theoretic and hands on approach has given us a unique insight in development of offline web apps.

We have found that you need to consider two things when making sure your web app works offline. First, you need to save the individual web pages and system resources they need to operate. Furthermore, any data the user is viewing or manipulating needs to be stored in a way that makes them easily accessible. The first part is solved by employing a service worker, a JavaScript that works in the background, separate from the rest of the web app. Storing information on the other hand, is solved by putting the information in a local NoSQL database of some kind. We have evaluated several such databases; the web standard solution IndexedDB, the IndexedDB wrapper Dexie, Pouch and Cloudant to name a few. In order to see how the different technologies intertwined we also constructed a proof of concept, of an Offline Web App (POOWA). The POOWA simulates a shopping list and allows you to synchronize shopping items across different browsers and units.

We set out to find technologies that enable web apps to compete with native apps in areas with bad coverage. We found that the service worker API is a key part of this. Our POOWA shows one approach and proves that you can make a web app that feels like a native app.

1 INNHOLDSFORTEGNELSE

Forsidebilde

Figuroversikt

Tabelloversikt

Oversikt over kodesnutter

Oversikt over skjermbilder

2 BEGREPSFORKLARING

API	En programkode som utfører bestemte hendelser og henter informasjon fra andre kilder uten at nettleseren må lastes på nytt (Store norske leksikon, 2018b).
Applikasjon	Et program som utfører et bestemt sett med oppgaver (Merriam-Webster, 2019b).
Cross Site Scripting (XSS)	En type injeksjonsangrep hvor skadelig kode blir utført på en ellers trygg side (OWASP, 2018).
Dokumentorientert database	En spesiell type database som organiserer informasjon fra dokumenter. Databasen kan tolke informasjon fra ulike dokumenter og organisere den etter ulike nøkler (Techopedia, U.Å.-b).
Hybrid Bridged Apps	En hybrid webapp som benytter brukergrensesnitt fra et mobilt operativsystem (Nunkesser, 2018).
Hybrid Web App	En webapp som bruker et skall utviklet til et mobilt OS. Innhold hentes fra internett og har samme funksjonalitet som en vanlig nettside (Nunkesser, 2018).
JSON	JSON er et tekstsyntak som legger til rette for strukturert dataoverføring mellom ulike programmeringsspråk (Ecma, 2017). Benyttes i Web App manifest og ofte i dokumentorienterte databaser.
Minimalisert	En prosess hvor man fjerner mellomrom og forenkler navn på variabler for å gjøre kode så kort som mulig (Techopedia, U.Å.-a).

Mobilapp	En applikasjon designet for bruk med smarttelefoner. Mobilapper er også kjent som en native app (Merriam-Webster, 2019a).
NoSQL	Er en betegnelse på databaser som ikke følger regelsettet for å kunne benytte tradisjonell SQL. Ofte omtalt som "Not Only SQL" (Techopedia, U.Å.-f).
Offline First	Er en fremgangsmåte for å designe apper først (first) for offline bruk. Deretter tilføye teknikker slik appen fungerer bedre og bedre ved tilgang til større systemressurser (Holt, 2017).
Offline Web Apps (OWA)	Vår betegnelse på webapper som er laget for å fungere uten internettilkobling.
OS	Også omtalt som operativsystem. Programvare som kontrollerer hvordan et datasystem håndterer programmer og systemressurser (Store norske leksikon, 2018a).
Konseptbevis	Et konseptbevis (eng: Proof of Concept) er en demonstrasjon for å vise at ulike konsepter eller teorier har potensiale for videre utvikling. Utformet for å bestemme gjennomførbarhet, og ikke for kommersiell distribusjon (Techopedia, U.Å.-e).
Sandbox (sandkasse)	Sandbox er type programvaretestingsmiljø for utføre selvstendig tester på programmet. Kan være en egen testserver (Techopedia, U.Å.-c).
Scope	Området eller omfanget hvor noe er relevant. For service worker avgjøres scope av sw.js filens

	plassering (Oxford University Press, U.Å.) (mozilla Contributors, 2019a).
System Language Apps	Mobil app skrevet i C eller C++, virker på både Android og iOS (Nunkesser, 2018).
Veiledning	Betegnelsen blir brukt istedenfor tutorials.
Web App	Definisjonen blir også omtalt som Progressive Web Apps. Skrevet med HTML, CSS og Javascript. Gjerne utviklet med tanke på bruk fra mobile enheter (Nunkesser, 2018). I denne rapporten vil vi benytte til begrepet webapper.
Working draft	En teknisk rapport publisert for historisk dokumentasjon og innsamling av tilbakemeldinger. Working drafts er et tidlig skritt på veien til å bli en webstandard (Rooney *et al.*, 2019).
Wrapper (innpakning)	En wrapper er et program eller kode som omslutter kompleksiteten til et annet og gjør det enklere å jobbe med det underliggende programmet eller koden (Techopedia, U.Å.-d).

3 INNLEDNING

I store deler av verden kan man i dag oppleve å ha for ustabil eller manglende internettilkobling, dette gjelder særlig i Norge på grunn av vår særegne geografi med trange daler og høye fjell og lange tunneler. For brukerne skaper dette et irritasjonsmoment. Dette forårsaker alt fra mistet informasjon til tapte arbeidstimer. Dette kan løses med native mobilapper, men disse er kostbare å utvikle og vedlikeholde. En mulighet for å løse disse utfordringene knyttet til utvikling og vedlikehold kan være webapper, men de trenger en aktiv internettilkobling. For at moderne webapper skal kunne konkurrere med native mobilapper må de ta hensyn til at brukeren ikke alltid er koblet til internett. Denne rapporten undersøker ulike teknologier som gjør dette mulig.

Rapporten besvarer oppgaven til oppdragsgiver IT Data AS. Bedriften utfører salg- og konsulenttjenester innen informasjonsteknologi, samt alt i tilknytting til den type virksomhet (se vedlegg 2 for oppgavetekst) (Brønnøysundregistrene, 2017). IT Data AS er en del av Adcom og har en egen utviklingsavdeling innen IT (Adcom, U.Å). Bedriften ønsker belysing på problematikken rundt dårlig dekning på grunnlag av brukerbehov hos sine kunder. Kunder av bedriften har opplevd at de må benytte papir og blyant når de har manglet nettilgang på grunn av utfordrende geografi. Vi valgte oppgaven fordi dens tema bygger direkte videre på emner vi har tatt i løpet av studieløpet og at tematikken er spennende. Løsningen på utfordringen tilknyttet mangelfull dekning er et dagsaktuelt tema (Sheppard, 2017). Gjennom samtale med bedriften har vi kommet frem til denne problemstillingen:

"Hvilke teknologier kan tilrettelegge for bedre funksjonalitet i offline webapplikasjoner?"

Avgrensning og presisering

All teknologi som muliggjør offline funksjonalitet i webapper samsvarer med problemstillingen. Vår studie avgrenses fra native mobilapper og teknologier tilknyttet til dem. Dette inkluderer en avgrensning fra hybride løsninger som er tilpasset ulike operativsystemer. I vår betraktning er vi ikke ute etter å finne en kommersiell løsning, så vi har ikke inkludert finansielle perspektiv. For å avklare problemstillingen er det nødvendig å definere hvilken offline funksjonalitet som eksisterer i dag. Vårt utgangspunkt for studien er at webapper har tilgang til noen teknologier som kan gi de offline kapasitet. Enkel varsling som "ingen internettilkobling" vil være en forbedring for de fleste av dagens webapper. Flere webapper blir endret eller utviklet for å tilpasses utfordringene ved manglende internettilgang. Hovedvekten av webapper er ikke tilpasset med tilstrekkelig offline funksjonalitet. Mer presist vil rapporten besvare hvilke muligheter som finnes innenfor offline webapper og teknologiene som er tilknyttet for å oppnå dette. Sikkerhet er et viktig område som ikke er direkte knyttet til problemstilling, og derfor inkluderer vi kun et delkapittel om temaet.

Oppgavens hensikt

I vår rapport er vi først å fremst ute etter å finne frem til de teknologiene som muliggjør offline funksjonalitet i en webapp. Vi har valgt å lage et for å demonstrere et utvalgt offline webteknologier i samspill.

Oppgavens oppbygging

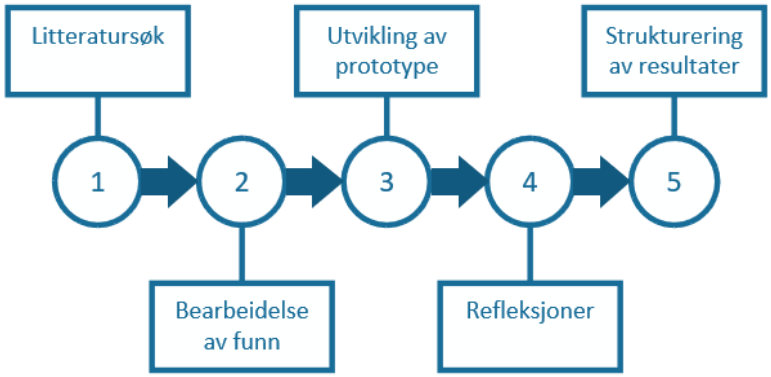
I oppgavebesvarelsen har vi tatt utgangspunkt i en litteraturstudie med innslag fra designvitenskap. De ulike funnene våre blir presentert på et teoretisk grunnlag før vi går nærmere inn på deres virkemåte og vår bruk i henhold til designvitenskapen. Avslutningsvis konkluderes funnene våre for å gi et helhetlig svar på problemstillingen og komme med alternative løsninger på den.

4 METODE OG FORSKNINGSTILNÆRMING

" Å bruke en metode, av det greske ordet *methodos*, betyr å følge en bestemt vei mot et mål". (Johannessen, Tufte og Christoffersen, 2010, s. 25). I denne studien har vi hovedsakelig benytte oss av en kvalitativ litteraturstudie med innslag fra designvitenskap.

Metodelæren blir delt inn i kvantitative og kvalitative metoder. Kvantitativ metode er opptatt av å telle antall fenomener (Ibid.). Eksempel på kvantitativ tilnærming vil være å telle antall programmeringsspråk som kan brukes. Kvalitativ metode sier noe om kvaliteten eller spesielle kjennetegn eller egenskaper ved det valgte fenomenet som skal undersøkes (Ibid.). Eksempel på en kvalitativ undersøkelse er å studere bruksmønster til en programvare.

Vår studie har to mål, der hovedmålet er å få innsikt i hvilke teknologier som muliggjør bruk av OWA. Dette har søkelys på eksisterende kunnskap og på å beskrive nevnt kunnskap. Det andre målet er å demonstrere teknologien ved hjelp av en eller flere. For å belyse det første målet vil vi naturlig gjøre oss kjent med ulike webteknologier og sammenligne disse med hverandre.



Figur 1 Tidslinje av gjennomførelse

4.1 LITTERATURSØK OG SØKESTRATEGI

Et systematisk litteratursøk er planmessig, begrunnet, dokumentert og etterprøvbart (Haraldstad og Christophersen, 2004). I vår studie gjennomfører vi et litteratursøk som er semi-systematisk, og uten å følge de samme reglene for dokumentasjon av funnene som et systematisk litteratursøk. Rapporten vil være en oversiktsartikkel med våre funn fra litteratursøket. En oversiktsartikkel som tar opp og kritisk vurderer forskning som allerede er presentert i vitenskapelige tidsskrifter (Dalland, 2007). Vi valgte litteratursøk fordi metoden er hensiktsmessig for å tilegne seg kunnskap om vår problemstilling.

Litteratursøkene våre vil i hovedsak omhandle webteknologier, med undertemaer HTML5, databaser, internettdekning og mobil-funksjonalitet. Vi vil ikke benytte noen bestemte søkeord, dette er fordi søkene våre vil føre til at man finner nye søkeord. Deretter benyttes de nye søkeordene sammen med eksisterende søkeord. Med tanke på utvelgelsen av informasjon som beskriver teknologier eller tar for seg bruk av disse vil vi primært se etter kilder som er fra 2010 eller senere. Der det er snakk om eldre teknologier vil vi se etter den siste oppdaterte informasjon som er tilgjengelig.

Vi vil primært benytte databasene til Google Scholar, Oria, IEEE og Web of Science til litteratursøket. Sekundært har vi brukt andre databaser slik som Google og tilsvarende søkemotorer.

4.1.1 Kildevurdering

"Kildevurdering er et samlebegrep for metoder brukt for å skille verifiserte opplysninger fra spekulasjoner" (Søk&Skriv, 2018). Vi har benyttet følgende for å bestemme relevans av kilden; den må omhandle temaene i problemstillingen. Vurdering av faglig bakgrunn til forfatterne, antall siteringer, kildebruk, målgruppe, sjanger og publiseringstidspunkt vil avgjøre kildens kvalitet (ibid.).

Innenfor webteknologi er det stadig endringer med nye teknologier og utvidet funksjonalitet for eksisterende teknologier. Denne bakgrunnen gjør det vanskelig å finne kun forskningsbaserte kilder og artikler. For å forsikre oss om at vi likevel finner den informasjonen vi er ute etter, velger vi hovedsakelig å benytte kilder som er skrevet av webutviklere tilknyttet større webutviklingsmiljø, for eksempel Google og Mozilla. Dette er en av konsekvensene ved å benytte litteratursøk. En annen konsekvens er at vi ikke får mulighet til å etterprøve all informasjon vi finner. Det har vært utfordringer knyttet til å finne akademiske kilder. Vi kan derfor ikke utelukke at det forekommer feilinformasjon i enkelte kilder. Fagfeltets raske utvikling gjør dette nesten uunngåelig. På grunn av den raske utviklingen kan vi heller ikke være sikker på at vi har oppdaget alle mulighetene, eller at den beste løsningen er beskrevet i rapporten vår.

4.2 DESIGNVITENSKAP

I planleggingsfasen av uttesting av teknologier bestemte vi oss for å benytte Hevner et al. (2004) sine publikasjoner knyttet til designvitenskap. Der beskrives fremgangsmåter man kan få til en forskningsbasert systemutvikling på (Hevner *et al.*, 2004). Som en del av den forskningsbaserte

tilnærmingen benytter vi noen av Hevners (Hevner *et al.*, 2004) retningslinjer for vitenskapelig tilnærming til systemutvikling.

Rapporten vår er en teknologibasert løsning på et konkret problem som kan bidra til å løse relevante forretningsmessige utfordringer. Dette er i henhold til Hevner sin retningslinje nummer to. Vi har i tillegg benyttet ulike teknologibaserte konseptbevis for å løse problematikken til IT Data AS sine kunder. Som en del av en iterativ tilnærming til systemutviklingen har vi kontinuerlig evaluert konseptbevis ved å teste funksjonaliteten tidlig i utviklingsprosessen, samt senere ved å simulere reelle bruk uten tilkobling. Dette er i tråd med retningslinje tre i henhold til kvalitetssikring av uttestingen.

Hevner (Hevner *et al.*, 2004) argumenterer for at letingen etter en løsning som virker er viktigere enn å finne den beste løsningen. Dette gjelder for tilfeller hvor en nøyaktig beskrivelse kan være umulig å oppnå. Ved vår uttesting av teknologier er dette ikke aktuelt, da vi ønsket å finne de beste løsningene innenfor offline webteknologi. Hevner sine resterende retningslinjer er ikke aktuell for vår studie fordi disse handler om forretningsmessig hensyn.

4.2.1 Metodekvalitet

I vår studie vil vi ta hensyn til pålitelighet, gyldighet og overførbarhet for å sikre en tilfredsstillende kvalitet. “Pålitelighet er knyttet til målekvalitet” (Busch, 2013, s. 62). Ved bruk av litteratursøk vil bruken av god kildekritikk være nøkkelen til om vi kan stole på dataene, samt gjennomførelsen av flere konseptbevis for å teste løsningen. Gyldighet er knyttet til i hvor stor grad vi måler det vi tror vi måler (ibid. s. 62). Vår studie vil fokusere på søk etter og testing av webteknologier, ved holde oss innenfor temaet og opprettholde fokusert blikk for å finne svar på problemstillingen. “Overførbarhet er knyttet til om våre resultater kan overføres til andre populasjoner eller andre situasjoner” (Ibid.). Ved korrekt og nøyaktig kildebruk, vil det være enkelt å finne tilbake til de samme kildene vi har benyttet i vår studie. Å gjenskape et konseptbevis skal være problemfritt ved bruk av kildekode. Der vi har benyttet veiledninger for ulike teknologier ligger den tilhørende referansen i en egen fil ved navn referanser i tilhørende mappe. Dermed kan vår studie lett gjenskapes i andre situasjoner og bruken av webapplikasjoner er relevant for verdenssamfunnet.

5 TEORI

I dette kapittelet skal vi gi en presentasjon av relevant informasjon for å avklare oppgavens elementer. På grunn av den stadige utviklingen innenfor det fagfeltet vi studerer vil denne teoridelen bidra til forståelse av mulighetene som finnes på nåværende tidspunkt.

Vi begynner med bakgrunnen innenfor webapper og forklarer de sentrale begrepene webapp, Progressive Web Apps (PWA) og mobilapper. Tekniske og praktiske funksjoner til webapp og mobilapper blir videre omtalt og vi ser på noen sikkerhetsaspekter rundt dette. Videre går vi dypere i programmeringsspråket JavaScript og tilhørende biblioteker som muliggjør OWA funksjonalitet. Service worker blir beskrevet i den sammenheng og den er sentral for vår problemstilling. Videre beskriver vi ulike muligheter for koblinger til servere og lagring i nettlesere.

5.1 BAKGRUNN

I slutten av 2015 overgikk datatrafikken fra mobiler datamaskiner for første gang, og siden den tid har mobilmarkedet fortsatt å øke (McLeod, 2018). Mer enn 80 % av tiden blir brukt på mobilapper sammenlignet med websider på mobilen i 2018 (Chaffey, 2018). Viktige fremskritt som vil endre den siste statistikken er kontinuerlig forbedring av nye og gamle teknologier, samt nye oppfinnelser innenfor fagfeltet.

Et annet viktig fremskritt innenfor området webapper og offline funksjonalitet er service worker API (SW). Ved bruk av SW kan man oppnå funksjoner som tidligere ville kreve en mobilapp, og før var det kun AppCache API som var benyttet for å gi offline brukeropplevelse. Service worker er utviklet for å unngå feilene som oppstå ved bruk av AppCache (Gaunt, 2019).

5.2 WEBSTANDARD

Organisasjonen World Wide Web Consortium (W3C) publiserer dokumenter som omhandler ulike web-teknologier. Teknologiene gjennomgår en prosess som skal sikre at de opprettholder et tilstrekkelig nivå med tanke på verdiene til internett om åpenhet og rettferdighet. Når denne prosessen er ferdig, blir teknologien publisert som en anbefaling av W3C og er da å anse som en webstandard (World Wide Web Consortium, 2012).

Rapporten beskriver teknologi som er på vei til å bli godkjent som standard som for eksempel Web App Manifest og teknologi som forsøker å bli standard som for eksempel Background Fetch ((W3C, 2019b)). I tillegg beskriver rapporten standardisert teknologi som IndexedDB (IDB) ((W3C, 2019a)).

Generelt vil det at en teknologi er blitt tatt opp som en standard, øke sannsynligheten for at flere nettlesere begynner å støtte bruken av den, men dette er opp til de respektive utviklerne av nettleserne.

5.3 WEBAPPER, PROGRESSIVE WEB APPS OG MOBILAPPER

Begrepene webapper og mobilapper har røtter tilbake til utgivelsen av den første iPhone i 2007 (Nunkesser, 2018). På IEEEs konferanse om Mobile Software Engineering and Systems i 2018 ble det foreslått en ny terminologi for utviklingen av mobilapper. I følge Nunkesser er ikke begrepene web, native og hybrid dekkende utenfor utvikling av apper til iOS plattformen. Han foreslår å bruke ulike betegnelser avhengig av teknologien som ligger bak utviklingen av den respektive appen.

Nunkesser foreslår å skille mobilappbegrepet i tre overordnede kategorier; ecdemic, endemic og pandemic apper. Begrepene kommer fra gresk og gjenspeiler de grunnleggende prinsippene i utviklingen av de ulike typene. Ecdemic og Endemic apper er kategorier av rene mobilapper, som skilles fra hverandre ut ifra språkene de er skrevet på. Nunkesser deler Pandemic apps i fire underkategorier, Web Apps, Hybrid Web Apps, Hybrid Bridged Apps, og System Language Apps. Dette er apper som er utviklet i språk som støttes av alle større mobile OS (ibid.). Web apps, også omtalt som Progressive Web Apps er applikasjoner skrevet ved hjelp av HTML, CSS og JavaScript. Til forskjell er hybride web apps en webapper som har et skall rundt seg. Det vil si at applikasjonen er utviklet spesielt for en mobil plattform, f.eks. Android eller iOS, der innholdet hentes fra internett i form av nettsider med HTML, CSS og JavaScript. Hybrid Bridged Apps er en videre utvidelse av hybride webapper. De tar i bruk elementer av brukergrensesnittet på den mobile plattformen, noe hybride webapper ikke gjør. System Language Apps er mobilapper som er skrevet i C eller C++, og gjelder for eksempel spill utviklet med rammeverk som Unreal Engine og Unity.

Native applikasjoner er utviklet for et bestemt operativsystem. Det vil si at de benytter ulike programmeringsspråk alt etter hva operativsystemet krever. Dette gjør at utviklingen av denne typen apper blir dyrere basert på antall plattformer den aktuelle applikasjonen skal distribueres til. Samtidig gir det utviklerne muligheten til å utnytte ulike styrker operativsystemene gir (Jog *et al.*, 2017). Foruten utvikling til ulike plattformer har også eierne av plattformene ulike krav til publisering av native apper. Android og iOS, som markedsledende mobile operativsystem (Statcounter, 2019) stiller også ulike krav til utviklere med tanke på godkjenning, publisering og oppdateringer av native apper (se avsnitt 5.4.3 Distribusjon og godkjenningsprosess)

Webapper er applikasjoner som ligger på webservere og leses av brukernes nettleser. Webapper er i utgangspunktet tilgjengelig uavhengig av plattformen brukeren benytter og brukeropplevelsen kan være like god på stasjonære og bærbare pc-er som nettbrett og smarttelefoner. Webappene benytter teknologier, som HTML, CSS og JavaScript, som blir tolket av nettleseren slik at brukeren kan få en god opplevelse uavhengig av hvilken plattform vedkommende benytter (Jog *et al.*, 2017). Vi vil heretter benytte begrepene OWA eller webapper.

Progressive web apps (PWA) ligger et sted mellom native apps og web apps. De er i utgangspunktet webapper som fra bunnen av er utviklet med et mål om å være pålitelige, raske og engasjerende (Basques, 2019a) (Sheppard, 2017). Påliteligheten kommer av at PWA forsøker å levere den samme brukeropplevelsen selv om man er i et område med mangelfull dekning. For nettsteder er også

hastigheten brukeren får tilgang til informasjon en svært viktig faktor (Alabbas og Bell, 2018). Ifølge An øker sjansen med 90% for at en bruker forlater nettstedet dersom tiden det tar å laste siden øker fra ett til fem sekunder (An, 2018). En annen studie om hvor lenge brukere blir på et nettsted slår fast at brukeropplevelsen de første ti sekundene er avgjørende for hvor lenge brukeren blir på nettstedet (Nielsen, 2011). Et annet kjennetegn på Progressive Web Apps er at de kan installeres på brukerens enhet og tillater fullskjerm med blant annet pushvarsler som minner om funksjonaliteten i en tradisjonell mobilapp (Basques, 2019a). PWA-ene bruker en høyere level kodespråk i forhold til mobilapper, som gjør at mobilen må jobbe hardere for å tolke koden (Strutt, 2017).

5.4 TILGANG TIL FUNKSJONER PÅ MOBILEN

Etter nedlasting av en mobilapp kommer det vanligvis forespørsler om tilgang til ulike funksjoner i mobilens operativsystem. Uten tilstrekkelig tilgang vil ikke mobilappen fungere optimalt, det samme gjelder for webapper (Jog *et al.*, 2017).

Bar (2019) har utviklet en oversikt over hvilke funksjoner som en webapp kan få tilgang opp mot ulike nettlesere. For kunne få tilgang til disse funksjonene er det ulike kodesnutter og API-er som må legges ved webappen. Den kan da oppnå tilgang til blant annet kamera, Bluetooth, push meldinger, filsystemet og geolocation. Det gjenstår fortsatt tilgang til noen funksjoner som kontaktliste og SMS (Bar, 2019).

5.4.1 Feilsøking

Remote Debugging gir muligheten til å inspisere webapper som kjører på en Android-enhet fra en utviklingsmaskin. Denne feilsøkningsmetoden behøver i tillegg en USB-kabel og Google Devtools for å fungere (Basques, 2019a). Å rette opp feil er en normal del av utviklingsprosessen og tar mellom 5-10% av det totale tiden brukt på prosjektet (Viktoria K. og V., U.Å). Ved bruk av samme tekniske utstyr som Remote Debugging, kan Android-enheten få tilgang til Localhost på utviklermaskinen (Kayce Basques, 2019).

Etter at et Web App Manifest er satt opp kan det inspisere filen i DevTools. Herfra kan man sjekke “start url”, simulere en “Add to Home screen” hendelse og få en mer brukervennlig fremstilling av koden. Service worker sin fane i DevTools er hovedplassen for å utføre feilsøking på service worker. Får tilgang til funksjonene offline, update on reload, statuslinje, feilmeldinger med mer. I Cache Storage vises en liste over elementene som har blitt lagt i cache ved bruk av Cache API som er en del av livssyklusen til service worker.(Mozilla Contributors, 2019f) og (Basques, 2019b)

5.4.2 Oppdatering

Mobilapper krever at brukeren hele tiden må laste ned oppdateringer for å forbedre tilgjengelighet og brukeropplevelse. For webapper skjer hele oppdateringsprosessen uten involvering av brukeren (Dua, 2019). Innholdet i webapper er tilgjengelig øyeblikkelig, samt vil være tilgjengelig for alle enheter som har støtte for nettleser.

I webapper er det lik kildekode til ulike plattformer og dermed vil utvikleren kun behøve å lage et sett med oppdateringer (Poetker, 2019). Hvis webappen er koblet til tredjeparts tjenester som Twitter og Facebook, og denne tjenesten endrer på API-ene sine må webappen oppdateres med den nye service API-en for at webappen skal fortsette å fungere. Slik oppdateringer er så hyppige, at de populære tjenestene har valgt å tillate støtte for gamle og nye API-versjoner. Oppdaterte apper er tryggere, mer effektiv og har den nyeste funksjonaliteten. Ved å oppdatere webappen får bruker ikke oversikten over tidligere feil og endringer (Viktoria K. og V., U.Å).

5.4.3 Distribusjon og godkjenningsprosess

Webapper må ikke godkjennes i App Store og Google Play, dette gjør at webappene kan lanseres raskere i forhold til mobilapper (Poetker, 2019).

Apple

iOS apper må gjennom en tidkrevende godkjenningsprosess i App Store, og hver oppdatering må gå igjennom denne samme prosessen (Naylor, 2019). I 2015 var gjennomsnittstiden på 15 dager for nylansering og oppdateringer for iOS App Store, i 2016 har det gått ned til to dager. Et utviklingsteam går grundig igjennom innsendte apper i leting etter feil før godkjennelse (Kesavan, 2017). Ved avslag vil denne prosessen vare lengre, og ved godkjennelse krever plattformen rundt 30 % av inntektene for iOS apper (Serrano, Hernantes og Gallardo, 2013).

Android

Godkjenningsprosessen for Android apper har brukt å ta en par timer og det har vært stor frihet til valg av innhold i appene (Kesavan, 2017). Det har tidligere vært enklere sammenlignet med App Store, men nå vil Google gjennomføre strengere krav og grundigere gjennomgang. Det gjelder for utviklere som ikke er registrert tidligere for å unngå ondsinnede utviklere (Now, 2019). Google Play Store åpent for PWA 31. januar 2019 ved bruk av Trusted Web Activity (TWA). Appen blir lagt til på mobilen ved hjelp av manifest og åpnes i frittstående modus i Chrome (Firtman, 2019).

Mobilapper er enkle å distribuere til brukerne, men kan være vanskelig å finne utenfor de ulike distribusjonsplattformene for digitale applikasjoner. På grunn av at mengden av apper på distribusjonsplattformer kontinuerlig øker, kreves det mer innsats for å oppnå synlighet (Serrano, Hernantes og Gallardo, 2013). For at webappen skal bli distribuert tilstrekkelig må den være tilgjengelig i Google sin database og benytte Search Engine Optimization (SEO) som hjelper for å oppnå synlighet på et Google-søk (Quality, 2019).

Webapper kan bli distribuert gjennom Chrome Web Store og blir dermed tilgjengelig for millioner av brukere. Her er det en rekke krav for å bli godkjent slik som bruk av bestemte funksjoner (Developers, 2019a).

5.5 NETTLESERSTØTTE

Tabell 1 Tabellen viser kompatibilitet mellom nettlesere og teknologi.

Dataene er bearbeidet for å skape et bedre sammenligningsgrunnlag. Teknologiene i tabellen er Offline.js (HubSpot, 2017)), Web App Man Manifest (Deveria, 2019d)*, Hoodie* (Hoodie, 2017b)*, Bootstrap* (W3schools.com, U.Å)*, Background Sync* (Deveria, 2019e)*, UpUp* (Ater, 2018)*, FileSystem* (Deveria, 2019f)*, Web Storage* (Deveria, 2019g)*, LocalForage* (MacPherson, 2015)*, Cache API* (Copes, 2018)*, WebSQL* (Deveria, 2019i)*, IndededDB* (Deveria, 2019c)*, Promises* (Archibald, 2019a)*, Dexie* (Fahlander, 2017d)*, CouchBase* (Couchbase, U.Å)*, Pouch DB* (Pouchdb)*, IBM Cloudant* (IBM, 2019)*,* (Deveria, 2019b)*, service worker* (Deveria, 2019h) *og CSP* (Deveria, 2019a)*.*

	IE (11)	Edge (17-)	Firefox (65-)	Chrome (71-)	Safari (12.-)	iOS Safari (11.-)	Opera (58)	Chrome for Android (73)	Samsung internett (8.-)
Offline.js	Kompatibel	Mangler data	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Inkompatibel	Kompatibel	Mangler data
Web App Manifest	Inkompatibel	Inkompatibel	Inkompatibel	Kompatibel	Inkompatibel	Delvis støttet	Inkompatibel	Kompatibel	Delvis støttet
Hoodie	Kompatibel	Inkompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Mangler data
Bootstrap	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel
Background Sync	Inkompatibel	Inkompatibel	Inkompatibel	Kompatibel	Inkompatibel	Inkompatibel	Kompatibel	Kompatibel	Kompatibel
UpUp	Inkompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel
File system	Inkompatibel	Inkompatibel	Inkompatibel	Kompatibel	Inkompatibel	Inkompatibel	Kompatibel	Kompatibel	Kompatibel
Web Storage	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel
LocalForage	Kompatibel	Inkompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Inkompatibel
Cache API	Inkompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Mangler data
WebSQL	Inkompatibel	Inkompatibel	Inkompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel
IndexedDB	Delvis støttet	Delvis støttet	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel
Promises	Mangler data	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Mangler data	Kompatibel	Mangler data	Mangler data
Dexie	Kompatibel	Mangler data	Kompatibel	Kompatibel	Teoretisk kompatibel	Teoretisk kompatibel	Kompatibel	Teoretisk kompatibel	Mangler data
CouchBase	Kompatibel	Mangler data	Kompatibel	Kompatibel	Kompatibel	Mangler data	Mangler data	Mangler data	Mangler data
PouchDB	Kompatibel	Mangler data	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Mangler data
IBM Cloudant	Kompatibel med CORS	Kompatibel med CORS	Kompatibel med CORS	Kompatibel med CORS	Kompatibel med CORS	Kompatibel med CORS	Kompatibel med CORS	Kompatibel med CORS	Kompatibel med CORS
Service Worker	Inkompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel
CSP	Inkompatibel	Delvis støttet	Delvis støttet	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Inkompatibel	Kompatibel
Server-sent events	Inkompatibel	Inkompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel	Kompatibel

Nettsider og webapper kan bli åpnet fra ulike nettlesere på ulike plattformer. For at innholdet og elementene skal fungere korrekt, må det være støtte hos nettleseren (European Commission, 2016). Tabell 1 viser alle teknologiene som omtales i rapporten og hvilke nettlesere som er kompatible. Background Fetch er kun tilgjengelig i Chrome 71, med "Experimental Web Platform features" aktivert (Archibald, 2019c). Teknologier som CouchDB er bygget for servere (Apache CouchDB). Informasjonskapsler er støttet hos det fleste nettleserne, men brukerne kan velge å slå av denne teknologien. For å unngå feilaktig informasjon er ikke Informasjonskapsler nevnt i tabellen (European Commission, 2018). Fordi Node.js er en server-side teknologi vil det ikke være hensiktsmessig å finne den i nettleserstøtte (Tutorialspoint, 2019).

5.6 SIKKERHET

I dette delkapitlet ser vi nærmere på sikkerhetstrusler webutviklere bør ta hensyn til og tiltak mot disse. Vi er spesielt interesserte i trusler forbindelse med vår problemstilling. OWASP (Open Web Application Security Project) publiserte i 2017 en rapport over de ti største sikkerhetsrisikoene for webapper (OWASP, 2017).

I denne oppgaven utforskes det hvordan man kan løse utfordringer knyttet til mangelfull dekning og om webapper kan nærme seg funksjonaliteten til en native app. Med det som bakgrunn er trusler som omhandler injeksjon og XSS, tema vi går nærmere inn på.

Tabell 2 De ti største sikkerhetsrisikoene for webapper.

Tabellen er oversatt til norsk (OWASP, 2017).

Trussel	**Beskrivelse**
Injeksjon	Angriper sender kommandoer til en database der det er forventet informasjon. Databasen utfører kommandoene.
Feil i autentisering	Svakheter i autentisering og forvaltning av sessions kan føre til at bruker-id og passord kommer på avveie.
Eksponering av sensitive data	Sensitive data som ikke er beskyttet, åpner for at angripere kan endre eller stjele data.
XML External Entities (XXE)	Risiko ved bruk av XML-filer. Angriper laster opp XML-filer som inneholder kode. Eldre utgaver av XML tillater at denne koden blir utført.
Tilgangskontroll som ikke virker	Svakheter knyttet til brukerrettigheter, kan gi brukere tilgang til andre brukerkontoer og filer, samt tilgang til å endre data, tilgangsrettigheter o.l.
Feil i sikkerhetsinnstillinger	HTTP headers, feilmeldinger med sensitiv informasjon, ad hoc løsninger og programvare som ikke er oppdatert, kan alle gi en angriper tilgang til systemet.
Cross-site scripting XSS	Script fra usikre kilder åpner blant annet for at bruker-id kommer på avveie eller brukere kan bli sendt til ondsinnede nettsider.
Usikret deserialisering	Programobjekter som inneholder bruker-id og som blir sendt sammen med nettverksforespørsler, kan i noen tilfeller bli deserialisert og gi en angriper tilgang til
Bruk av komponenter med kjente	Programvare fra tredjeparts forhandlere hvor det blir oppdaget en sårbarhet, gjør data sårbar inntil programvaren er oppdatert.
Loggføring og overvåking	Mangelfulle logger og dårlige rutiner for overvåking gir angripere god tid om de kommer seg forbi sikkerheten.

5.6.1 Injeksjon

Dette er en svakhet ved bruk av SQL-databaser, der angriperen gjennom f.eks. en funksjon for å logge på en webapp sender en spørring istedenfor brukernavn eller passord. Ved å skrive `"brukernavn' – "` vil spørringen som blir sendt til databasen tolke det som er skrevet i passordfeltet som en kommentar. Om brukernavnet da finnes i databasen, får man tilgang til sidene bak passordbeskyttelsen uten å vite passordet (Horgen, 2014).

Det er mulig å beskytte seg mot SQL-injeksjoner ved å benytte seg av prepared statements. Dette skiller logikken og dataene som blir brukt i spørringen fra hverandre. I programkoden blir dataene opprettet som variabler som blir inkludert spørringen. Ved at logikk og data er skilt fra hverandre forstår programmeringsspråket at dataene ikke skal tolkes som en del av selve spørringen {OWASP, 2019 #152}.

5.6.2 XSS (Cross-site scripting)

En av de største sikkerhetsrisikoene ved bruk av internett i dag er Cross-site scripting (OWASP, 2017) hvor en angriper kjører JavaScript som ikke er en del av nettstedet originalt. I beste fall handler dette bare om å få et nettsted til å oppføre seg uvanlig (Wagenseil, 2014), i verste fall om datatyveri. XSS kan deles i tre hovedtyper avhengig av hvor det uønskede scriptet kommer fra (OWASP, 2018).

Presistent XSS er JavaScript-kode som kommer fra nettstedet sine databaser. Dette skjer typisk via kode skrevet i et kommentarfelt eller annet HTML-skjema der svaret så blir hentet og vist et annet sted på nettstedet. Dersom det ikke er tatt forhåndsregler mot dette, er dette en relativt enkel måte å få nettstedet til å kjøre script som ikke hører hjemme der (ibid.). Reflected XSS er script som kommer fra brukerens egen henvendelse til nettstedet. Den vanligste måten dette skjer på, er gjennom en lenke som inkluderer scriptet. Det finnes tjenester som gjør det mulig å korte ned på en lenke slik at man kan gjemme JavaScript i den (ibid.). Dom-based XSS skiller seg fra de andre angrepstypene ved at alt skjer i nettleseren til brukeren. Den som blir utsatt for angrepet blir lurt til å åpne en link som inkluderer JavaScript-kode som manipulerer `document.innerHTML` direkte i brukerens nettleser (ibid.).

XSS angrep kan forhindres ved å ta i bruk en Content Security Policy nærmere forklart i avsnitt 5.6.3, og ved å programmere et eget JavaScript etter anbefalte retningslinjer for sikkerhet. Der man bruker input fra brukeren er det viktig at man tar høyde for at inputen kan inneholde uønsket kode. Denne må hindres fra å kjøre eller enkelte tegn må byttes ut slik at koden ikke vil virke lengre (Kallin og Valbuena, 2013).

5.6.3 CSP (Content Security Policy)

CSP er en måte å fortelle nettleseren fra hvilke sider det er greit å hente ressurser. Det fungerer ved at man legger inn kode i head-taggen på nettstedet. Denne sikkerhetspolicyen kan skreddersys til å dekke en mengde ulike HTML-elementer og er en god måte å beskytte mot XSS-angrep som kommer fra brukerens kode (ibid.).

CSP-en ligger inne i head-taggen i HTML-filen og leses først når nettstedet lastes. CSP-en hvitelister de kildene for script man tillater. Alle script som man ikke har direkte kontroll over selv, vil være en kilde til sikkerhetsrisikoer. Dette inkluderer JavaScript biblioteker som Bootstrap, JQuery og Ajax (ibid.). En CSP som er satt opp riktig vil beskytte mot XSS ved at den ikke lar nettleseren kjøre andre tredjepartskilder enn de som er tillatt.

CSP er kommet til sin andre iterasjon, CSP 2, som i dag støttes av de mest brukte nettleserne, unntatt Internet Explorer (Deveria, 2019a). En fordel med CSP er at dersom en brukers nettleser ikke støtter det, får det ingen effekt for nettstedet. Siden ser lik ut for en bruker, uavhengig om CSP er støttet. Naturligvis kan ikke policyen beskytte brukere som benytter nettlesere der det ikke støttes, men om den er satt opp riktig, kan den rapportere forsøk på XSS og gjøre administrator eller nettverksansvarlig oppmerksom på at et XSS angrep har skjedd (Kallin og Valbuena, 2013).

5.7 JAVASCRIPT

I dette delkapittelet ser vi nærmere på JavaScript og mulighetene det gir utvikling av moderne webapper. JavaScript er et programmeringsspråk som blir mye brukt for å gjøre nettsteder mer dynamiske ved at det tillater nettsiden å respondere på brukeraktivitet. Dette gjør nettsiden mer levende. Språket blir regulert av Ecma (Ecma Internatinal - European Association of Standardizing Information and Commnication Systems (Ecma, U.Å), og er i dag i sin 9. utgave (Ecma, 2018). Programmeringsspråket er objektorientert og har en syntaks som minner om Java og C++. Dette er et bevisst valg utviklerne har tatt for å gjøre språket enklere å lære

(Mozilla Contributors, 2019g). JavaScript er i dag et svært utbredt og blir benyttet i ulik grad av omtrent 95% av alle nettsider (W3Techs.com, 2019).

Et JavaScript-bibliotek er en samling JavaScript-kode som er skrevet for å gjøre et sett med funksjoner. Ved bruk av denne typen bibliotek kan utviklere spare tid ved at de slipper å skrive gjentagende kode (Wozniewicz, 2019). Ved bruk av bibliotek er det opp til utvikleren hvor og hvordan biblioteket blir brukt.

JavaScript finnes også som rammeverk, og det er også kode ment for gjenbruk. Rammeverk fungerer som oppskrifter for å oppnå et gitt resultat. Når man benytter et rammeverk følger man oppskriften og koden i rammeverket gir deg resultatet du er ute etter. Det er ikke lenger opp til utvikleren hvor og hvordan koden blir brukt, og man har mindre påvirkningskraft enn ved bruk av biblioteker (ibid.).

5.7.1 Offline.js

Dette er et enkelt JavaScript, med et tilhørende bibliotek, som oppdager og varsler brukeren dersom man mister nettverkstilkoblingen. Skriptet fungerer ved at det overvåker henvendelser til en ekstern ressurs. Ved manglende nettilgang tar skriptet vare på Ajax henvendelser og sender de når tilkoblingen er tilbake (HubSpot, 2017).

5.7.2 Bootstrap

Bootstrap er et rammeverk for utvikling med HTML, CSS og JavaScript. Ved bruk av verktøysettet blir det enkelt å bygge responsiv og mobilvennlig prosjekter (Bootstrap, U.Å).

5.7.3 Hoodie

Hoodie er en rask og enkel programvare som er utviklet for webapper. Hoodie er en noBackend teknologi for å gjøre utviklingen enklere, og apper utviklet med Hoodie vil være Offline First. Programvaren er basert på teknologiene CouchDB og Node.js, og skrevet i JavaScript. API til Hoodie benytter programmeringsspråket Dreamcode. Dreamcode er laget for å være lett å forstå for front-end utviklere. Det er mulig å legge data til Cloudant for sikkerhetskopiering og koble til Localhost med Node.js. Hoodie er utviklet til at ved å koble kildekoden opp mot Hoodies API, vil appen være klar til bruk. (Hoodie, 2018).

5.7.4 UpUp

UpUp er et JavaScript-bibliotek som er en innpakning rundt service worker. Skriptet fungerer ved at det gir utvikleren muligheten til å kontrollere arbeidet SW utfører i bakgrunnen, uten at utvikleren direkte trenger å konfigurere den. UpUp trenger bare å få vite hva man vil ha tilgang til når man mangler internettilkobling, så sørger skriptet for at man får tilgang til lagret versjon av nettstedet (Ater, 2018).

5.7.5 Web App Manifest

Består av en JSON-fil som forteller nettleseren om webappen og hvordan den skal oppføre seg på brukeren sin mobile enhet eller datamaskin. Filen gjør det mulig å legge en snarvei til webappen på startskjermen og skrivebordet (Gaunt og Kinlan, 2019). Web App Mainfest forteller om webappens navn, ikon, hvilken URL som skal åpnes først, hvilken visningsmodus og hvilke linker som kan åpnes i den. Felles for disse metadataene er at de bidrar til at webappen fremstår som en vanlig mobilapp (Caceres *et al.*, 2018).

5.7.6 Web Background Sync

Background Sync utvider service worker med muligheten for å legge til en synkroniseringshendelse. Det er ikke å regne som en webstandard per i dag (Josh Karlin og Kruisselbrink, 2018). Service worker tar vare på informasjonen brukeren prøver å sende til dekningen er god nok (Archibald, 2019c).

Background Sync er fullt støttet i Chrome og under utvikling til Firefox og Edge. Selve synkroniseringshendelsen vil i utgangspunktet samle opp alle forsøkene på å sende informasjon. Dersom en bruker forsøker å sende tre e-poster med utilstrekkelig internettilkobling vil synkroniseringshendelsen utløses for hver av de tre. Første gang vil den prøve å sende den første e-posten, og tar så vare på den når det ikke fungerer. Når e-post nummer to blir sendt forsøker synkroniseringshendelsen å sende både den første og den andre e-posten. Dette vil fortsette helt til internettilkoblingen er tilbake og alle meldingene er blitt sendt. Med API-en er det mulig å konfigurere synkroniseringen til å inkludere en siste sjanse, og dersom denne feiler, vil ikke Background Sync forsøke

å sende dataene flere ganger (Josh Karlin og Kruisselbrink, 2018). Det er verd å merke seg at Background Sync er avhengig av at brukeren aktivt ser på siden data blir sendt fra. Den kan ikke konfigureres til å 'spamme serveren' med sendinger hvert 5. sekund eller lignende (Archibald, 2019c).

5.7.7 Background Fetch

Dette er en API som bygger videre på grunnarbeidet rundt Background Sync, men på en større skala og den kan brukes både til å sende og motta data (Archibald og Beverloo, 2019) (Archibald, 2019b). Den største forbedringen kontra Background Sync er at Background Fetch virker selv om både nettleseren og service worker blir slått av. Nedlastingen vil fortsette neste gang brukeren er tilkoblet internett igjen (Salnikov, 2018).

Nedlastninger av større filer er ikke en nyvinning i seg selv. Det som skiller Background fetch fra vanlige nedlastinger er at nedlastingene foregår i en webapp, ikke i operativsystemet på datamaskinen eller mobiltelefonen (Salnikov, 2018). Dette har vært mulig ved bruk av service worker tidligere, men den får problemer ved større filer og avbrudd. Background Fetch er utviklet for å unngå disse problemene.

5.8 KOBLINGEN TIL SERVERNE

I dagens samfunn bruker vi mye tid på internett, både hjemme, på jobb og på tur (Statcounter, 2016). Norge er et land med god mobildekning, men det er fortsatt områder der geografien gjør at dekningen er utilstrekkelig (Olsen *et al.*, 2017). I dette delkapitlet ser vi nærmere på teknologi som omhandler kommunikasjon mellom klienten og webtjeneren, samt teknologi som hører hjemme på webtjeneren.

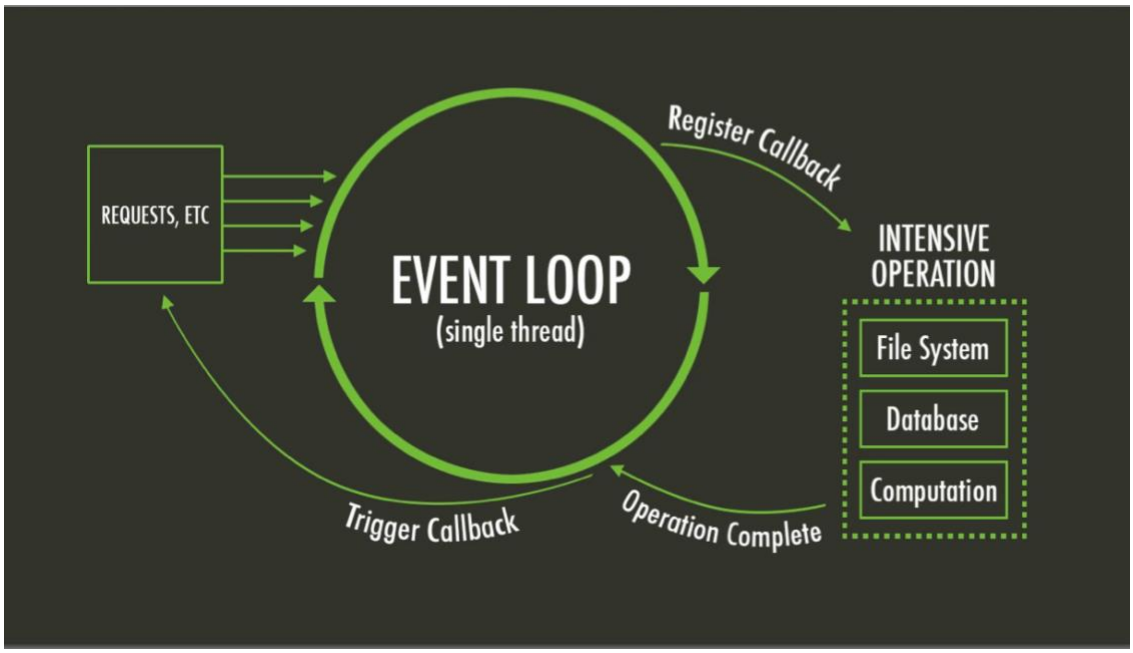
5.8.1 Node.js

Nettsider består av en fremside (frontend) og en bakside (backend). Fremsiden er det en bruker ser når vedkommende åpner en nettside i nettleseren. Baksiden av nettstedet, de delene brukeren ikke ser, gjør at nettstedet er tilgjengelig. Dette inkluderer ofte en server, en applikasjon og en database. Fremsiden av en webside er bygget på HTML, CSS og JavaScript. Baksiden derimot, er vanligvis bygget med programmeringsspråk som PHP, Ruby og Python (Long, 2012).

Node.js tar JavaScript-språket til webserverne og gjør at man kan benytte den samme syntaksen i hele utviklingen av et nettsted. Node er åpen kildekode og uavhengig av plattform, og er designet for å være lett å bruke gjennom et stort bibliotek av JavaScript-moduler (Tutorialspoint, 2019). Dette biblioteket gjør det enklere å bygge nye nettsted da man kan gjenbruke kode andre har skrevet.

Den største forskjellen mellom Node.js og Apache- server (PHP) er hvordan serveren behandler forespørsler fra klienter. En Apache-server behandler forespørsler fortløpende ettersom de kommer inn. Hver forespørsel blir tildelt en tråd som er opptatt til forespørselen er ferdig behandlet. Når alle trådene serveren har tilgjengelig er opptatte, må den neste henvendelsen vente til en oppgave blir gjort ferdig. En server som kjører Node.js derimot, benytter seg av en enkelt tråd til å motta forespørsler og sende responser. Hver ny forespørsel blir lagt i en event-loop styrt av en enkelt tråd. Når serveren blir ferdig

med behandling av oppgaven, sendes responsen tilbake til loopen. Den sørger for at responsen blir returnerer til den som sendte forespørselen.



Figur 2 Node.js behandling av forespørsler (Ofoegbu, 2018).

I følge (Chaniotis, Kyriakou og Tselikas, 2015) fungerer Node.js bedre enn PHP/Apache i de fleste tilfeller. Chaniotis har testet tre kjente serverteknologier mot hverandre, PHP/Apache, Nginx og Node.js for å komme med en anbefaling basert på testresultatet. Testene så nærmere på antall forespørsler, bruk av CPU og minne og stabiliteten på tilkoblingene. Som en viktig målestokk for webservere så testen også på hvordan systemene behandlet større kvantum av samtidige tilkoblinger, også opp mot systemressursene.

5.8.2 Server-sendte hendelser

Server-sendte hendelser gjør det mulig for webtjenere å varsle en klient når noe blir endret på tjeneren. Serveren kan fortløpende sende varsel som består av en hendelse og data (Mozilla Contributors, 2019k). Tradisjonelt har dette bare vært mulig ved at klienten spør tjerner om det er kommet noen oppdateringer (W3schools.com). Denne teknologien kan være av spesiell interesse ved bruk av mobile enheter ved at de slipper å bruke unødvendig batteritid og systemressurser på å finne ut om det er kommet en oppdatering webappen (WHATWG, 2019b).

5.8.3 API

En Application Programming Interface (API) er programkode som tar informasjon fra klienten til en webserver og returnerer et svar. Dette gjør at man via en webside kan hente ny informasjon uten å laste nettsiden på nytt. For eksempel når man henter informasjon fra en database ved å sende en ny spørring. Den programkoden som henter informasjonen fra databasen og viser den via for eksempel DOM

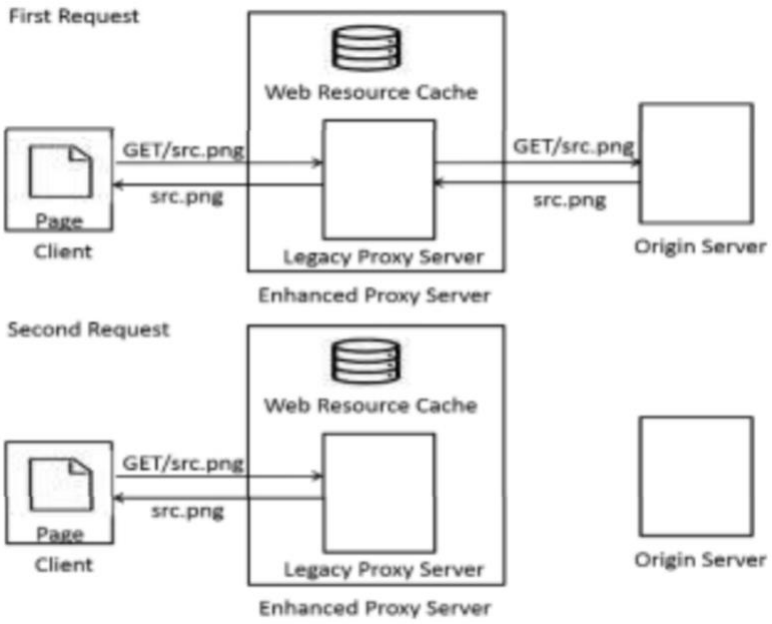
manipulasjon er en API. På samme måte krever teknologier som IndexedDB at man benytter funksjoner som transaction og callback. Et annet eksempel på en API er service worker som ligger mellom klienten og webserveren.

5.8.4 Service worker API

"Service workers are a fundamental technology in the future web platform. " (Basques, 2019b).

Service worker (SW) er en web worker, nærmere bestemt et JavaScript som kjører i bakgrunnen, adskilt fra resten nettsiden. Dette gir muligheter for funksjoner som ikke trenger et nettsted eller interaksjon fra brukeren. SW har allerede støtte for push varsler og Background Sync. SW er basert på Promise (se avsnitt 5.10.3), et moderne JavaScript attributt som gjør bruken lettere (Gaunt, 2019).

SW fanger opp ulike nettverksforespørsler og svarer dem, samt mellomlagre og mottar ressurser fra cache (Google Developers, 2019) (Se figur 3). Service worker krever HTTPS for å fungere, men kan testes med Localhost (Gardner, 2016).



Figur 3 Service worker med to forespørsler (Pande et al., 2018).

I følge Gaunt vil service worker kunne ha mulige funksjoner som periodisk synkronisering eller geofencing. Tidligere har det vært brukt en annen API for offline opplevelse AppCache, SW er utviklet for å unngå problemene med AppCache (Gaunt, 2019).

Service worker er avhengig av to API for å få webapper til å fungere offline, Fetch og Cache. Fetch den er vanlig måte for å motta innhold fra nettverket, og cache er å lagre lokalt (Google Developers, 2019). SW kan blant annet returnere et dokument fra cache istedenfor å gå via nettverket, som muliggjør offline brukeropplevelse (Aderinokun, 2018).

5.9 LAGRING I NETTLESERE

Nettleserne gir oss tilgang til ulike måter å ta vare på data midlertidig. Hva de ulike teknologiene kan lagre og hvordan de behandler dataene, varierer noe mellom de ulike formene. Før HTML5 var informasjonskapsler det eneste alternativet for å lagre data midlertidig. HTML5 har gitt oss tilgang til WebStorage (local- og session-storage), Filesystem API, og Cache API. I tillegg gir nettleserne oss tilgang til to ulike Databaseformater, WebSQL og IndexedDB (Kitamura, 2014).

Under følger en oversikt som tar for seg disse teknologiene med tanke på nettleserstøtte, om de støtter transaksjoner og hvorvidt de er synkrone eller asynkrone. Transaksjoner er en form for overføring av data tilknyttet databaser. De sørger for at data som hører sammen ikke kan bli delvis lagret.

Med andre ord vil transaksjonen enten mislykkes og ingen data blir lagret i databasen. Hvis transaksjonen lykkes blir hele datamengden lagret (Cohen, 2019). Synkron/Asynkron bestemmer om teknologien kan behandle data i bakgrunnen, eller om man må vente til behandlingen er ferdig før man kan gjøre noe mer (ibid.).

Tabell 3 Oversikt over lagring i nettlesere.

Tabellen er oversatt fra (Cohen, 2019) (se vedlegg 1). Nettleserstøtte tallene kommer fra: FileSystem (Deveria, 2019h), Webstorage (Deveria, 2019i), Cache API (Copes, 2018), IndexedDB (Deveria, 2019g) og WebSQL (Deveria, 2019g)*, Cache API* (Copes, 2018)*, IndexedDB* (Deveria, 2019c) *og WebSQL* (Deveria, 2019i)*. Informasjonskapsler er fullt støttet ifølge den originale tabellen* (Cohen, 2019)*, men brukere kan selv velge å avvise de i nettleseren (European Commission, 2018).*

API	Nettleserstøtte	Transaksjoner	Synkron/Asynkron
Filesystem API	68 %	Nei	Asynkron
Webstorage (session og local)	97 %	Nei	Synkron
Informasjonskapsler	100 %	Nei	Synkron
Cache API	90 %	Nei	Asynkron
IndexedDB	90 %	Ja	Asynkron
WebSQL	87 %	Ja	Asynkron

Mengden data som kan lagres varierer med bakgrunn i hvilken teknologi, nettleser og plattform man befinner seg på. Hvordan systemet behandler data når det opplever at harddisken nærmer seg full varierer også. Som eksempel vil Mozilla Firefox finne og slette data som ble sist brukt når man kommer over den globale grensen nettleseren bruker for diskplass (Mozilla Contributors, 2019d).

5.9.1 Filesystem API

En JavaScript-basert API som lar nettleseren simulere en virtuell harddisk. Denne virtuelle harddisken støtter dra og slipp funksjoner for filer, lik det som er mulig i operativsystemet. Denne virtuelle harddisken får ikke tilgang til andre ressurser ved at den er plassert i en sandbox som isolerer API-en fra andre systemressurser. Denne begrensningen kan omgås ved for eksempel å benytte egne utvidelser i nettlesere som Chrome (Mozilla Contributors, 2019c).

API-en er utviklet av Web Incubator Community Group (WICG) og er ment som et utkast på noe som kan bli en webstandard dersom det får nok oppslutning (ibid. og (WICG, 2015). File System API-en er per april, 2019 ikke på vei til å bli det, og den har begrenset støtte av ulike nettlesere (Bell, 2018).

5.9.2 Webstorage

Webstorage inkluderer to former for lagring i nettleseren, local- og session-storage. Dette er to former for lagringsmedier som enkelt kan aksesseres ved hjelp av JavaScript. Det er eksempler på teknologier som er synkrone på den måten at de deler systemressurser med resten av nettstedet. Det vil si at når data i forbindelse med local- eller session-storage blir behandlet må man vente til operasjonen er ferdig før websiden kan gjøre noe mer. Ved behandling av større datamengder vil dette gi utslag i dårligere ytelse for brukeren. Nettstedet må vente på at lagringsmediet gjør seg ferdig før man kan gjøre noe annet (Degges, 2018). Som en følge av dette virker ikke local- og session-storage sammen med SW (Google Developers, 2019).

Det som skiller local- og session-storage fra hverandre er hvor lenge dataene blir lagret. Data lagret med localstorage varer til man eksplisitt sletter de enten ved at nettstedet er programmert til å gjøre det eller ved at man manuelt ber nettleseren om å slette dataene (Mozilla Contributors, 2019b). Sessionstorage på den andre siden blir lagret til økten er ferdig, altså at man lukker nettleservinduet (Mozilla Contributors, 2019i). En videre begrensning ved local og session-storage er mulighetene for hva man kan lagre. Disse lagringsmediene støtter bare muligheten til å lagre tekststrenger, så dersom man har behov utover det kan ikke disse lagringsmediene løse det (Degges, 2018).

5.9.3 Informasjonskapsler

Informasjonskapsler ble tidligere brukt for å lagre data lokalt i nettleseren, men i dag anbefales det å heller velge mer moderne API-er som dekker denne funksjonaliteten bedre. I dag brukes de hovedsakelig til å ta vare på brukerinnstillinger som språkinnstillinger og handlekurver, og lagring av brukeroppførsel (Mozilla Contributors, 2019e).

Informasjonskapsler er ikke designet for bruk sammen service worker, og har begrensninger på hvor mye data de kan lagre (Degges, 2018). Dessuten blir de sendt på nytt hver gang nettleseren sender en henvendelse til serveren, så en informasjonskapsel som inneholder mye informasjon kan potensielt gjør at hele nettstedet oppleves tregt (Mozilla Contributors, 2019e).

5.9.4 Cache API

Cache API er i dag et system for lagring av henvendelser og responsene fra en webserver. API- en fungerer som en del av service worker og lar utviklere bestemme hvilke ressurser som skal være tilgjengelige fra nettleserens lagringsmedium Cache. Dette gjør at man kan styre hvilken funksjonalitet som er tilgjengelig offline, og bygge opp et nettsted til å fungere i områder uten dekning. Cache er godt egnet til å ta vare på statisk innhold (Archibald, 2019d). API-en støttes i Chrome, Firefox og Opera, det per Februar 2019 jobbes aktivt med å implementere støtte for dette i Edge og Safari (Scales, 2019). Cache API er til forveksling lik en API som ikke lengre blir jobbet med og som var en tidlig måte å bestemme hvilke filer (av nettsiden) som skal lagres lokalt (Mehta, 2011).

5.9.5 LocalForage

LocalForage er et JavaScript-bibliotek som gjør det enklere å lagre data for bruk i en offline setting. Det er utviklet av Mozilla og er designet rundt bruken av flere ulike API-er, localstorage, IndexedDB og WebSQL. LocalForage velger hvilken av disse API-ene som skal benyttes ut ifra hvilke teknologier som støttes av nettleseren (Matthew MacPherson, Robert Nyman og Fabbro, 2014). Utviklere som benytter LocalForage har også muligheten til å overstyre dette valget og står fritt til å spesifisere hvilke API de vil benytte for sitt prosjekt. LocalForage sitt førstevalg er IndexedDB. I eldre versjoner av Safari ble WebSQL benyttet istedenfor, frem til Safari fikk støtte for IndexedDB i versjon 10.1. fra 2017 (Apple Inc, 2017), (Greasidis og MacPherson, 2018b). LocalForage kan benytte både JavaScript Callbacks og Promises og er på den måten fleksibel etter hva utviklere liker å arbeide med (Greasidis og MacPherson, 2018b).

5.10 LOKALE DATABASER I NETTLESEREN

5.10.1 WebSQL

Er en database i nettleseren som benytter SQL til å behandle dataene som blir lagret i den. WebSQL er JavaScript-basert og støttes av de fleste nettleserne. WebSQL er basert på SQLite som ifølge utviklerne er en av de mest utbredte databaseteknologiene i verden (SQLite). Selv om WebSQL bygger på en teknologi som er svært utbredt er det ikke en webstandard, og det jobbes ikke med å videreutvikle produktet. Dette henger sammen med at WebSQL er bygget på SQLite. Dette diskvalifiserer WebSQL fra å kunne bli en standard fordi alt arbeidet med databaseteknologien kommer fra den samme kilden man ikke har kontroll over (Hickson, 2010). Eventuelle fremtidige endringer til SQLite vil da kunne påvirke WebSQL på uforutsette måter (Ranganathan, 2010).

5.10.2 IndexedDB

IndexedDB er en W3C standard database som egner seg godt til lagring av informasjon lokalt i nettleseren (Alabbas og Bell, 2018). Strukturen i en IndexedDB-database minner om strukturen i SQL databaser, der objects i IndexedDB minner om tabeller fra SQL (Google Developers, 2018). Teknologien er støttet av de aller fleste nettlesere (se tabell 1), (Deveria, 2019c), men mengden data en IndexedDB kan lagre varierer mellom de forskjellige nettleserne (Kitamura, 2014).

Selv om strukturen minner om en SQL-database har ikke IndexedDB et SQL-lignende språk. På grunn av dette må man selv programmere funksjoner for å manipulere databasen. Dette foregår gjennom transaksjoner og gjennom bruken av disse sørger man for at integriteten til databasen blir ivaretatt. Transaksjonen kan bestå av en eller flere manipulasjoner av databasen. Dersom en av disse feiler blir alle stoppet og databasen blir returnert til det stadiet den var i før transaksjonen begynte. IndexedDB har en kompleks grunnstruktur, og tillater bare søk etter eksakte likheter (Fahlander, 2017a).

5.10.3 IndexedDB med Promises

Det finnes flere JavaScript-biblioteker som forenkler arbeidet med IndexedDB. Denne typen bibliotek ligger som et mellomledd mellom koden utvikleren skriver og databasen. IndexedDB med promises er et av disse. Skriptet er lite og legger i hovedsak til rette for å bruke promises istedenfor IndexedDB-Requests (IDBRequests), som er en del av den originale API-en (Archibald, 2019a). IDBRequests er betegnelsen på events som leser eller skriver data til databasen (Mozilla Contributors, 2019h). Dette biblioteket er ideelt for å lære IndexedDB, og Google bruker det aktivt ved opplæringer til API-en (Google Developers, 2018).

5.10.4 Dexie

Dexie er et JavaScript-bibliotek som har som hovedmål å løse tre utfordringer med IndexedDB, nemlig feilbehandling, spørringer og kompleksitet (Fahlander, 2017a). Dexie er skrevet slik at hver forespørsel til databasen bare trenger en catch funksjon på slutten av koden for fange opp feilbeskjeder (Fahlander, 2018). Dexie lar brukerne utføre spørringer mot databasen, uten at de er følsomme for store og små bokstaver. Dexie åpner også for spørringer med logiske 'eller' operatører og sammenligner av datasett (Fahlander, 2017a).

Både den enklere feilbehandlingen og de forbedrede spørringene, sammen med andre forbedringer gjør at Dexie kan gjør mer mot IndexedDB med færre linjer kode enn teknologien kan uten biblioteket (ibid.).

5.10.5 Couchbase

Couchbase er en NoSQL, åpen kildekode og dokumentertorientert database utviklet av Couchbase Inc. Databasen er utviklet for bruk i interaktive webapper og mobilapper. I likhet med Cloudant lagrer databasen dokumentene som JSON-filer. Databasen har integrert caching, andre effektive funksjoner og tilbyr lineær skalering (Chopade og Dhavase, 2017). NoSQL databaser fungere meget bra til skalering på grunn av at de er vellaget til å ha en fleksibel struktur på data (Camp *et al.*, 2018).

5.10.6 Apache CouchDB

CouchDB er en åpen kildekode dokumentertorientert database som er skrevet i Erlang, og benytter NoSQL. Databasen er designet for lokal kopiering og skalering horisontalt til en rekke ulike enheter. I likhet med PouchDB har CouchDB støtte for de andre databasene CouchBase og Cloudant (Apache CouchDB, U.Å)

5.10.7 PouchDB

PouchDB er en åpen kildekode JavaScript-database som er inspirert av Apache CouchDB. Databasen opererer i nettleseren og lagrer dataene lokalt. Dette gjør at brukerne kan benytte funksjonene når de er offline. Dataene blir synkronisert mellom klientene, slik at dataene blir oppdatert på de ulike enhetene (Pouchdb, U.Å.-b)Spørringene utføres raskt fordi de ikke må utføres over nettverket (J Justin og Jude, 2017).

Databasen blir støttet av alle de moderne nettlesere, og bruker IndexedDB som standard i bakgrunnen (se avsnitt 5.10.2). Dermed vil PouchDB ha tilsvarende nettleserstøtte som IndexedDB (avsnitt 5.5). Hvis nettleseren ikke har støttet for IndexedDB, bruker PouchDB WebSQL(Pouchdb, U.Å.-a). PouchDB kan benyttes som en direkte interface til CouchDB-kompatible servere, siden de begge kan kjøre i Node.js. I Node.Js benytter PouchDB seg av LevelDB i bakgrunnen. PouchDB kan også kjøre sin egen CouchDB-kompatible server ved bruk av PouchDB Server, men også CouchDB, Cloudant og Couchbase (Pouchdb, U.Å.-a).

5.10.8 IBM Cloudant

Cloudant er en NoSQL-database som lagrer JSON-dokumenter og fungere godt med web- og mobil-apper (Appel *et al.*, 2016). Den tilbyr individuell server-løs skalering med kapasitet og lagring mot ulik betaling. Databasen tilbyr en gratis løsning med begrensinger av mengder med data lagret. Alle data blir krypterte når de er inaktive og når de blir synkroniserte mellom databasene (IBM, 2019).

6 DISKUSJON OG FUNN

I denne delen av rapporten vil vi drøfte vår problemstilling ut ifra fremlagt teori. Vi har fulgt den samme rekkefølgen som i teoridelen, men i denne delen beskriver vi hvordan teknologiene fungerer og hvordan vi har benyttet de. Vi vil anvende kodesnutter og skjermbilder for å vise ulike konseptbevis av teknologiene som muliggjør vår problemstilling. Vi ønsker å vise alternativer til disse teknologiene, og vi argumentere for hvilke teknologier som enten alene eller i samspill med andre vil være den beste løsningen.

Det er et større fokus på utviklingen av apper til mobilen, dette er på grunn av at 80% av tiden brukes på mobilapper sammenlignet med websider i 2018 (Chaffey, 2018). Viktige fremskritt som har bidratt til å øke fokuset på webapper er service worker (se avsnitt 5.8.4). SW gjør at man kan oppnå tilgang til funksjoner som tidligere kun var mulig via mobilapper. Service worker er utviklet for å unngå feilene til forgjengeren AppCache (Gaunt, 2019), og er en viktig del av den nye generasjonen av webapper.

Nye fremskritt innenfor webapp teknologien vil fortsette å oppstå i tiden fremover, og vi har prøvd å ta med teknologier som ikke er standarder for å vise frem alle mulighetene. Vi har også med teknologier som er standarder, og noen der det er samspill med de nyeste teknologiene. Dagens utvikler benytter seg av både nye og gamle teknologier som er tilgjengelig innen webteknologi. Nyvinningene gir muligheten til å gi brukerne en følelse av å bruke en mobilapp ved bruk en moderne webapp. Webappene har muligheten til å være raskere og offline, samt er lettere å utvikle. Dermed kan webappene gjøre mobilappene mer eller mindre overflødig over tid. Sluttsummen av dette kan endre mobilapp-markedet for alltid.

6.1.1 Webstandard

Tabell 4 Webstandard.

Tabellens kilder: CSP, (West et al., 2016), Server-sent events (WHATWG, 2019b),Web storage (WHATWG, 2019a), IndexedDB (Alabbas og Bell, 2018), Dexie (Fahlander, 2017a) Promises (Archibald, 2019a), Hoodie (Hoodie, 2017b), Localforage (Greasidis og MacPherson, 2018a), PouchDB (Pouchdb), File system (Mozilla Contributors, 2019c), Web App manifest (Caceres et al., 2019), Service worker (Alex Russel, 2019), Background sync (Josh Karlin og Kruisselbrink, 2018), Cache API (Mozilla Contributors, 2019f), UpUp (Ater, 2018), Offline.js (HubSpot, 2017), WebSQL (Hickson, 2010), CouchDB og Couchbase (Pouchdb), IBM Cloundant (Pouchdb).

Teknologi	**Status**
CSP	Standard
Server-sent events	Standard
Web Storage	Standard
IndexedDB	Standard
Dexie	Basert på standard
Promises	Basert på standard
Hoodie	Basert på standard
LocalForage	Basert på standard
PouchDB	Basert på standard
File System	Eksperimentell
Web App Manifest	Eksperimentell
Service worker	Eksperimentell
Background Sync	Eksperimentell
Cache API	Eksperimentell
UpUp	Eksperimentell
Offline.js	Utdatert
WebSQL	Utdatert
Bootstrap	Mangler data
CouchDB	Mangler data
CouchBase	Mangler data
IBM Cloudant	Mangler data

I tabellen er teknologier som hører sammen gruppert sammen, Dexie og Promises er gruppert under IndexedDB som de bygger på. Det samme gjelder Hoodie og PouchDB. LocalForage er spesiell, i og med at det er et JavaScript-bibliotek som velger IndexedDB, WebSQL eller localstorage avhengig av støtte i nettleseren. Vi har valgt å plassere det i tilknytning til IndexedDB fordi dette er førstevalget for LocalForage og har bred støtte i ulike nettlesere (se tabell 1).

Av standardene jobbes det aktivt med nye utgaver, som er på det eksperimentelle stadiet både for CSP og IndexedDB (Alabbas og Bell, 2019) (West, 2016). Web Storage og Server-sent events er deler av HTML standarden (WHATWG, 2019b, 2019a). Av de eksperimentelle teknologiene jobbes det aktivt med utvikling av service worker og Web App Manifest (Alex Russel, 2019; WHATWG, 2019a). W3C vil gi de nevnte eksperimentelle teknologiene status som webstandard når teknologiene er utviklet tilstrekkelig.

Background Sync, Cache API og UpUp er alle basert på service worker API-en. Hvorvidt de to første blir å regne som standard når service worker er ferdig utviklet er fortsatt uvisst.

Felles for alle teknologiene som mangler data er at det er snakk om databaseteknologier som benytter NoSQL. Unntaket fra dette er Bootstrap som er et rent JavaScript-rammeverk. Cloudant, PouchDB og CouchBase er basert på eller har store likheter med CouchDB. Node.js er ikke inkludert på tabellen fordi det er en teknologi som utelukkende befinner seg på web-servere og ikke i nettleseren.

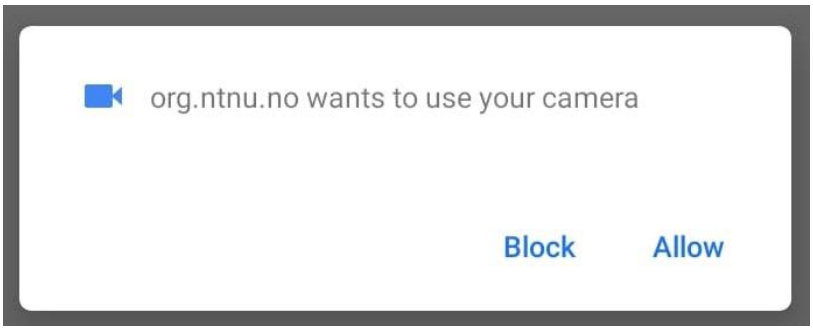
Det kan være gunstig å benytte webstandard, men det kan være utfordrende på grunn av tidskrevende godkjenningsprosessen. En alternativ tilnærming kan være å basere seg på de teknologiene som bygger på standarder eller som det jobbes aktivt med for å gjør til webstandard.

6.1.2 Webapper, Progressive Web apps og mobilapper

Av begrepene Nunkesser (2018) foreslår er "Web App" og Progressive Web Apps" (PWA) av interesse for oss. For utvikleren er det kostnadsbesparende å velge en PWA fremfor native app, så lenge man kan dekke behovene til brukeren ved hjelp av nettleseren (smArtapps, 2018). For brukeren sin del vil en PWA bruke små systemressurser, den kan være lett å dele og den får tak i innholdet raskt (ibid.) En ulempe er at PWA er skrevet i høy level kode som PHP og JavaScript. Dette fører til at mobilen bruker mer tid på tolking av koden, og dermed kan PWA bruke mer batteri sammenlignet med en mobilapp (Strutt, 2017). Som vi har vært inne på tidligere er flere av teknologiene som man benytter for å få en webapp til å bli en PWA, under utvikling. Dette medfører at enkelte funksjoner ikke nødvendigvis trenger å være til stede i alle nettleserne, noe som kan påvirke brukeropplevelsen. Samtidig kan det være andre funksjoner en native app får tilgang til som PWA må klare seg uten (smArtapps, 2018).

6.1.3 Tilgang til funksjoner på mobilen.

Tilgang til funksjoner i mobilens operativsystem er en essensiell del for å kunne få full utnyttelse av webapper (Jog *et al.*, 2017). Ved å bruke forskjellige kodesnutter og API-er er det mulig å få tilgang til en rekke funksjoner i operativsystemet. Det er for eksempel vanskelig å utvikle en kamera-webapp uten tilgang til kamera. Skjermbilde 1 nedenfor viser en webapp som spør etter tilgang til kameraet (se vedlegg 7). Det er på grunn av sikkerhet i nettleserne at forespørsel om tilgang kommer. Det kan sammenlignes med at det kommer opp flere forskjellige forespørsler om tilgang ved installeringen av en mobilapp. Tilgangen til enkelte funksjoner på mobilen er noe som skiller webapper fra mobilapper, videre utvikling på området vil påvirke konkurransesituasjonen.

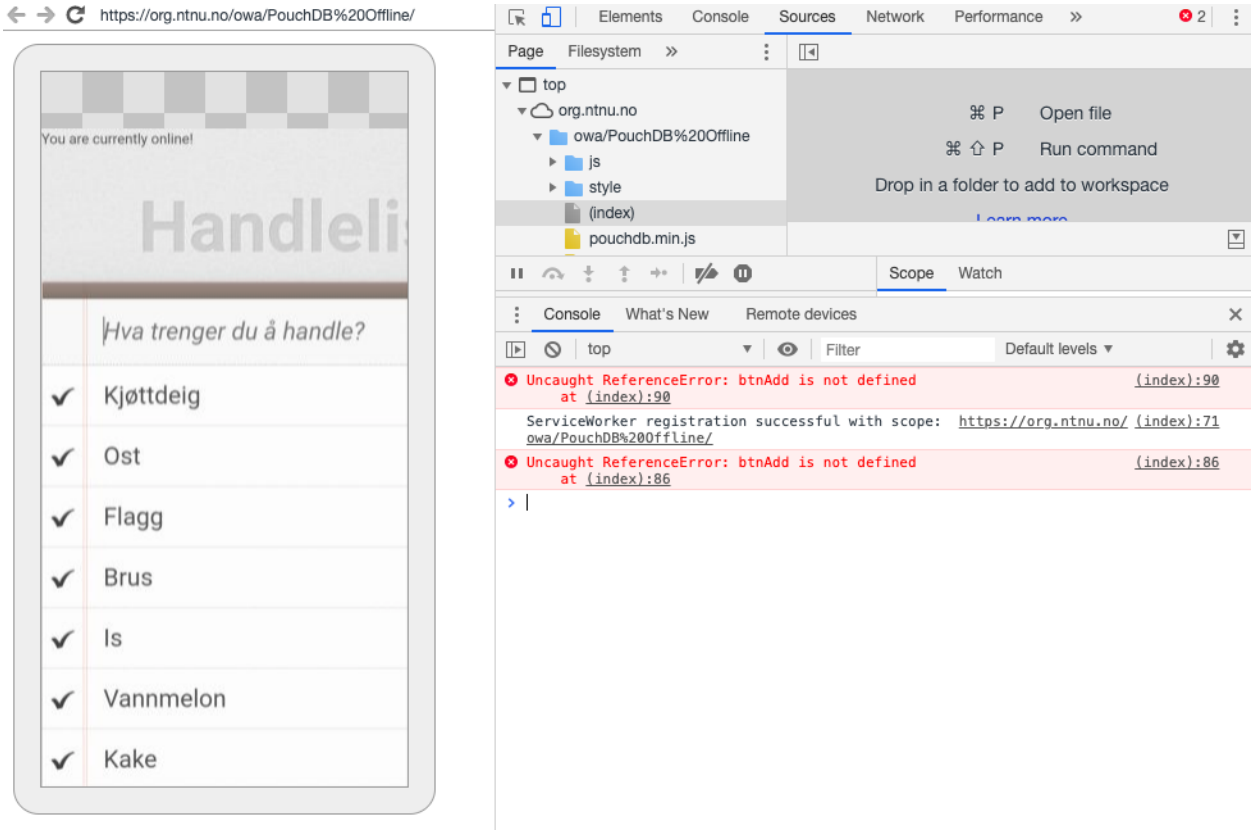


Skjermbilde 1 Webapp ønsker tilgang til kamera (Benjamin, 2018).

6.1.4 Feilsøking

Remote Debugging er en metode for å gjennomføre feilsøking på en kjørende webapp. For å kunne gjennomføre trenger man en datamaskin (Windows, Mac eller Linux), en USB-kabel, og Chrome DevTools. Denne metoden gir mulighet til finne nye feil og forbedringsområder på webapper. Skjermbilde 2 viser mulig bruk av Remote Debugging, her med feilmeldinger om en manglende knapp.

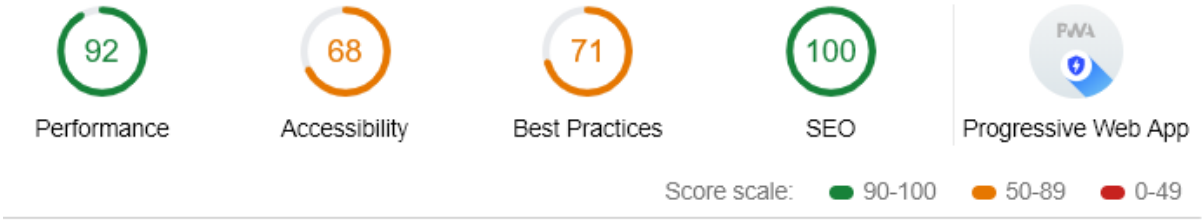
Feilsøking i Chrome DevTools kan benyttes for noen av elementene i webapper som f.eks. manifest, service worker og service worker cache. Web App Manifest kan inspiseres i DevTools og gir en mer visuell fremvisning av koden. Hovedplassen for inspisering og feilsøking av service worker er i DevTools. Dette er på grunn av tilgangen til beskrivelse om scope, status og flere andre funksjoner tilknyttet SW. Verktøyet har egne feilmeldinger tilhørende til SW, og under Cache Storage vises en liste med hvilke ressurser som har blir cashet ved bruk av service worker. Clear Storage er en veldig nyttig funksjon for utviklingen av webapper, funksjonen kan slette service worker, fjerne all cache og lagring i nettleseren (Basques, 2019b).



Skjermbilde 2 Bruk av Remote Debugging på en kjørende webapp.

Lighthouse ligger som en av DevTools og gir mulighet til å teste ytelse, brukervennlighet, beste praksis, SEO og PWA som skjermbilde 3 viser. Det er mulig å øke poengscoren innenfor de ulike grupperingene ved å etterfølge forslagene som testen viser (Developers, 2019b). Skjermbildet 3 viser at webappen som ble testet ikke scoret perfekt, i dette tilfellet er det blant annet på grunn av bruk av manglende mobiltilpasning og ineffektiv cache policy.

DevTools gir fordeler fordi det er et velkjent verktøy for en webutvikler og er lett tilgjengelig via nettleseren. Den gir ikke muligheten til å teste hvordan webapp vil fungere 100 % lik som på mobilen, men det er mulig å benytte Remote Debugging til dette.



Skjermbilde 3 Resultat etter Lighthouse brukt på en webapp.

Vi har i all hovedsak benyttet oss av DevTools ved utviklingen av ulike konseptbevis for å feilsøking. Vi har i tillegg brukt redigeringsprogrammer og forumsider som Stack Overflow. Alle konseptbevis har i tillegg blitt testet på mobil for å utelukke feil, vi har benyttet Remote Debugging kun for å teste funksjonalitet.

6.1.5 Oppdateringer

Hele oppdateringsprosessen av webapper skjer uten involvering fra bruker og er tilgjengelig øyeblikkelig for alle enheter som har støtte for nettleser (Dua, 2019). Dette gjør at brukeren sparer tid og at utviklingsteamet sparer både tid og penger ved utvikling av webapper. Dette er med tanke på å at utvikleren trenger kun et sett med oppdateringer på grunn av tilsvarende lik kildekode på de ulike plattformene (Poetker, 2019). Det er flere sikkerhetsfordeler ved å oppdatere appen ofte, den blir i tillegg mer effektiv og får tilgang til den nyeste funksjonaliteten. Ved webapper blir det lettere å holde appen oppdatert, men man får ikke tilgang til endringsloggen. Under utviklingen har det vært en ressurs at vi kunne enkelt ha testet det nyeste innholdet på ulike plattformer øyeblikkelig.

6.1.6 Distribusjon og godkjenningsprosess

En viktig forskjell mellom webapper og mobilapper er lanseringstiden, for iOS tok det i gjennomsnitt to dager i 2016 (Kesavan, 2017). Til tross for utviklingen innenfor webapper og distribusjonsplattformer er det fortsatt viktig å få tilstrekkelig med synlighet.

En måte å få mer synlighet på er ved å benytte SEO og dermed få bedre treff ved et Google-søk (Quality, 2019). SEO er en gratis metode for å markedsføre webapper, men det er også mulig å betale for en SEO ekspert for å oppnå mer synlighet (Quality, 2019). Det å få en app godkjent hos Apple krever tid og penger, men hos Android sin Play Store tar det kortere tid. Hos Play Store og App Store er det lett å bli en av mengden, der har webapper fordel med bruk av SEO. Google blir ofte brukt når brukere leter etter noe, men ikke i like stor grad når brukeren vil laste ned en app. Fordelene ved webapper er at de ikke må betale til distribusjonsplattformen og de blir lansert øyeblikkelig.

6.1.7 NETTLESERSTØTTE

Nettleserstøtte er essensielt for om webappene vil fungerer optimalt, tabell 1 viser en oversikt over hvilke teknologier som er kompatibel hos nettleserne. De teknologien som har støtte hos nesten alle nettleserne ut ifra tabellen er service worker, IndexedDB, UpUp, Hoodie, Bootstrap, Web Storage og Cloudant (ved bruk av CORS).

I vår oppgave har vi fokuset på at teknologien skal fungere i moderne Chrome (fra 71) og Chrome for Android. Tabell 1 viser at alle teknologiene i rapportene har full nettleserstøtte hos Chrome, og nesten full støtte i Chrome for Android.

6.2 SIKKERHET

Som vi så fra tabellen (se tabell 2) i teoridelen er eksponering av sensitive data også blant de største sikkerhetstruslene i webapper. Ettersom sikkerhet ikke et fokus for denne oppgaven, går vi ikke dypt inn i måter å ta vare på sikkerheten. I dette delkapittelet ser vi nærmere på måter man kan beskytte brukeren mot injeksjoner og XSS angrep. Vi inkluderer også en generell del om sikring av informasjon.

Uten at vi tar spesielle hensyn, vil data lagret ved hjelp JavaScript være sårbare, som for eksempel i IndexedDB og localstorage (Metcalf, 2016; Degges, 2018). Dersom webappen behandler sensitiv informasjon må man ta hensyn til dette. En løsning kan være å kryptere informasjonen som blir lagret. Kryptering er allerede en del av HTTPS-protokollen, som omhandler informasjonen man sender og mottar (Symantec Corporation, U.Å.).

Fordi service worker kan brukes til å stoppe trafikk til og fra nettleseren, krever denne at man benytter HTTPS for å kunne brukes (Gaunt, 2019). IBM Cloudant-databasen har innebygde funksjoner for kryptering av data (IBM, 2019). Dette viser at sikkerhet er elementer i teknologiene vi har benyttet.

6.2.1 Injeksjon

Som en beskyttelse mot SQL-injeksjoner er det anbefalt at man benytter seg av prepared statements {OWASP, 2019 #152}.

```
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email)
VALUES (?, ?, ?)");
$stmt->bind_param("sss", $firstname, $lastname, $email);
$firstname = "John";
$lastname = "Doe";
$email = "john@example.com";
$stmt->execute();
```

Kodesnutt 1 Prepared statements i PHP

Eksemplet over er hentet fra W3schools og viser hvordan man benytter prepared statements i PHP (W3schools.com, U.Å.-a). Ved bruk av Node.js sin MySQL-modul er ikke prepared statements tilgjengelig per mai 2019 (Wilson og Sidorov, 2019). Her må vi benytte oss av en teknikk som kalles Escape-string-spørringer. Et eksempel på dette er vist i kodesnutt 2.

```
var userId = 'some user provided value';
var sql    = 'SELECT * FROM users WHERE id = ' +
connection.escape(userId);
connection.query(sql, function (error, results, fields) {
  if (error) throw error;
  // ...
});
```

Kodesnutt 2 Escape-string kommando med MySQL modul til Node.js

Denne teknikken gjør at det blir satt inn skråstrek (/) foran potensielt skadelige tegn, noe som gjør at potensielle kommandoer til databasen blir ufarliggjort.

6.2.2 XSS og CSP

Vi testet CSP ved å legge til en CSP-header på en av de tidlige IndexedDB-demoene vi lagde (se vedlegg 4). Den gjør at man ikke kan kjøre skript på siden fra andre enn kildene som er lagt til i CSP-en. Ettersom vi bruker flere eksterne JavaScript-biblioteker og rammeverk, fikk vi også testet og bekreftet at disse ble blokkert når de ikke var inkludert i CSP-en.

```
<meta http-equiv="Content-Security-Policy" content="default-src 'self';
 style-src 'self'
 https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css ;
 font-src 'self'
 https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/fonts/ ;
 script-src 'self'
 https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js
 https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js
  ;">
```

Kodesnutt 3 Enkel CSP

Som nevnt i teorien er CSP-en en måte å beskytte seg på mot XSS-angrep. Ettersom sikkerhet ikke er et hovedfokus for vår oppgave, gikk vi ikke lengre enn å sette opp en enkel CSP og bekrefte at den virker.

Med tanke på å være et forsvar mot XSS-angrep er det vert å nevne at dersom JavaScript fra en kilde man stoler på blir kompromittert, så vil ikke CSP-en beskytte lengre. For vårt eksempel vil det si at dersom Google sitt jQuery-bibliotek kunne ha inneholdt kode for å stjele informasjon fra vår IndexedDB uten at vi hadde merket noe.

Som et alternativ og supplement til CSP bør man også benytte seg av metoder for å desinfisere inndata. Dette gjelder både når vi lagrer informasjon lokalt i nettleseren og når vi sender informasjon til databaser på serveren. Vi kan oppnå denne ekstra beskyttelsen ved å benytte JavaScript bibliotek som DOMPurify (bos, 2016).

6.3 JAVASCRIPT

Det finnes alternativer til å programmere i JavaScript. Kharchenko (2018), lister åtte alternativer programmeringsspråk for JavaScript. Felles for de fleste av alternativene er at de har metoder som oversetter sin egen kode til JavaScript. Det vil si at de i praksis bare er alternativ for JavaScript for utvikleren og brukeropplevelsen vil være tilnærmet den samme.

Det er flere grunner til at JavaScript er det naturlige førstevalget for arbeid med webutvikling. Viktigst av disse er den brede støttet språket har i nettlesere og utbredelsen av det, 95% av alle nettsider benytter programmeringsspråket i større eller mindre grad (W3Techs.com, 2019). I en undersøkelse blant aktive bidragsytere på Stack Overflow svarte nærmere 70% av brukerne at de bruker JavaScript, noe som gjorde det til det mest utbredte programmeringsspråket blant Stack Overflow sine brukere for sju år på rad (Stack Overflow, 2019). Ifølge Kharchenko (2018) er JavaScript også det språket, av alternativene hun nevner, som har flest innebygde funksjoner.

6.3.1 Offline.js

Offline.js varsler besøkende på en nettside som benytter scriptet om aktiv internettilkobling og gjør det mulig å aktivere JavaScript events når dette endre seg. Foruten at scriptet overvåker Ajax-hendelser for å vite om brukeren er online, er det mulig å konfigurere scriptet for hvordan det skal lytte etter internettilkobling. For å benytte Offline.js trenger man bare å inkludere JavaScript filen, samt filer for utseende og språk. Alle er tilgjengelige fra scriptets hjemmeside (HubSpot, 2017).

Offline.js virket i utgangspunktet interessant, men arbeid på prosjektet stoppet i 2017. Vi vurderer dette som et inaktivt script og valgte å ikke gå videre med testing av det.

6.3.2 Bootstrap

Bootstrap har bidratt til å gjøre webappen leselig på mobile enheter, ved å gjøre webappen responsiv (Bootstrap, U.Å). En minimaliserte utgave av biblioteket er brukt som en del av veiledningen til UpUp og blir der lastet for at webapp skal holde seg responsiv når den er offline. Vi har benyttet Bootstrap i vår utvikling for at webappene skulle være leselig på mobilen uten å måtte gjennomføre ytterlige tilpasninger. Slik at vi har kunnet fokusere på offline funksjonalitet.

6.3.3 Hoodie API

Hoodie benytter noBackend-teknologi, dette betyr at man trenger ikke å forholde seg til servere (Hoodie, 2018). Hoodie har også støtte for tredjeparts løsninger for tilleggsfunksjoner for servere (Peyser, 2017). Kodesnutt 4 nedenfor viser hvordan man registrere en ny bruker med kun en linje av kode. Informasjon om den nye brukeren lagres på Hoodie Server. Det er på grunn av hoodie.account API som er inkludert som en del av Hoodie (Hoodie, 2017a). Det er flere API-er som er inkludert ved nedlastning, og som alle for det meste har en lettfattelig kode (Peyser, 2017).

```
hoodie.account.signUp(account.Properties)
```

Kodesnutt 4 Registrering av ny bruker i Hoodie. (Hoodie, 2017a)

Et annet fokus for Hoodie er å fungere Offline First som betyr at brukerne sine data vil bli lagret lokalt som standard. Dette gjør at apper basert på Hoodie er tilgjengelig når tilkoblingen er ustabil (Hoodie, 2018). Vi anbefaler å sette opp egen server for mer avanserte apper med tredjeparts løsninger for å få fleksibilitet og skalerbarhet.

Hoodie er basert på teknologiene CouchDB og Node.js, og kan kobles opp mot databaser slik som Cloudant. Dette er teknologier vi har testet i andre sammenhenger og valgte derfor å ikke teste Hoodie.

6.3.4 UpUp

UpUp sin programvare gjør det lettere for utvikleren å kontrollere service worker for å oppnå Offline First. Service worker legger de nødvendige filene for offline funksjonalitet i Storage Cache som vist på skjermbilde 4. Den største ulempen med UpUp er også den største fordelen, nemlig hvor enkel den er. Dette gjør service worker vanskeligere å modifisere ved bruk av andre teknologier, og feilsøking blir vanskelig siden man ikke har programmert service worker koden selv. Som kodesnutt 5 viser skal det lite

koding til for å implementere UpUp. I tillegg til innholdet i kodesnutten må tilhørende filer til UpUp legges ved. For uerfarne webutviklere vil UpUp være en fin start for lære seg å utvikle offline webapper, men for erfarne webutviklere og bedrifter anbefaler vi å benytte egenutviklet service worker for lagring i Cache, siden dette gir større muligheter for å gjøre tilpasninger.

```
<script src="upup.min.js"></script>
  <script>
    UpUp.start({
      'content-url': 'offline.html',
      'assets': ['css/bootstrap.min.css', 'css/offline.css']
    });
</script>
```

Kodesnutt 5 Hovedkoden bak UpUp

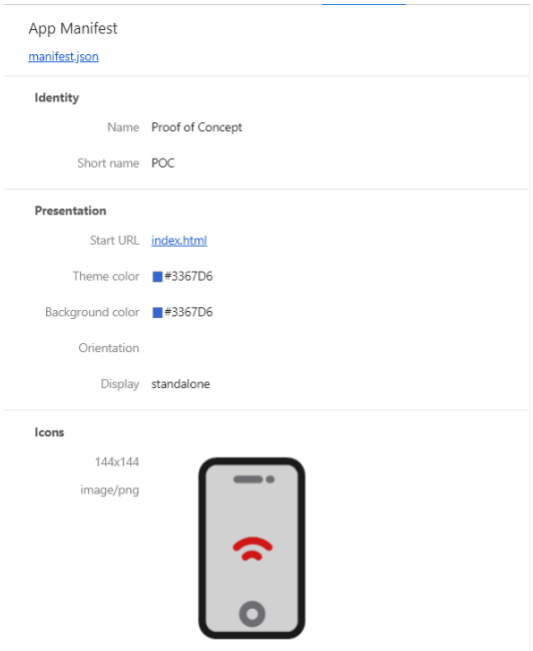
Path	Response-Type	Content-Type	Content-Length	Time Cached
/owa/UpUp%2020.03/css/bootstrap.min.css	basic	text/css	19,252	10.5.2019, 13:...
/owa/UpUp%2020.03/css/offline.css	basic	text/css	191	10.5.2019, 13:...
/owa/UpUp%2020.03/sw-offline-content	basic	text/html	513	10.5.2019, 13:...

Skjermbilde 4 Filer i Cache Storage.

Vi har gjenskapt veiledningen fra nettsiden til UpUp for å få en webapp til å fungere offline (se vedlegg 16). Den største forskjellen mellom offline og online versjonen er at førstnevnte ikke viser bildene. Det er fullt mulig å legge til bilder og andre filer, men det er ikke anbefalt å legge for mange elementer i cache på grunn av lagringsplass og nedlastingshastigheten.

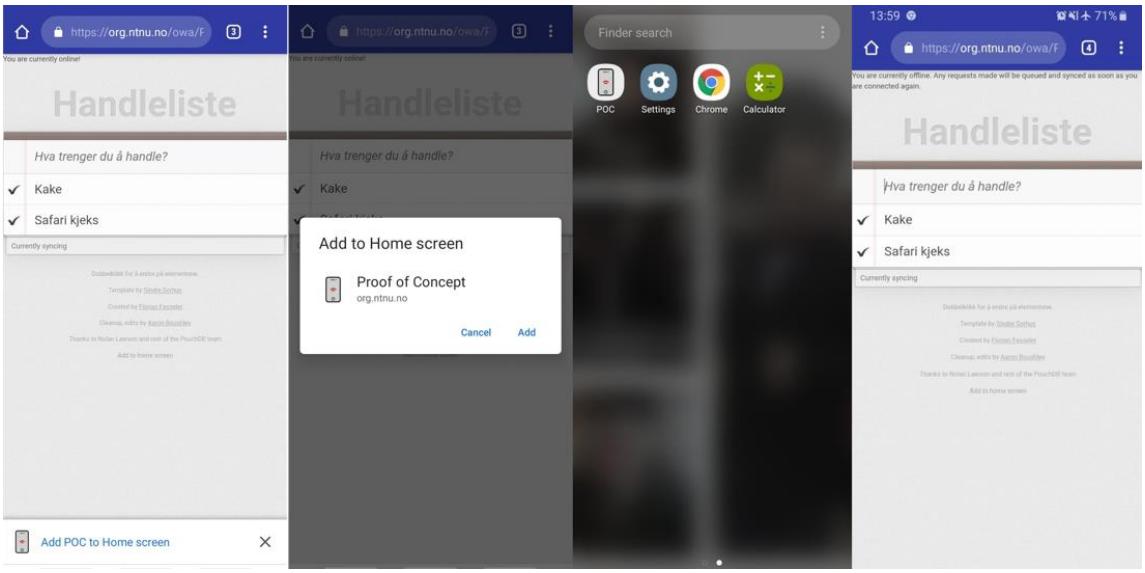
6.3.5 Web App Manifest

Ved å benytte manifest vil webappen fremstå som en vanlig mobilapp og være like tilgjengelig sammenlignet med andre mobilapper på mobilen (Caceres *et al.*, 2018). Bruk av Web App Manifest gjør overgangen til webapper svært liten for brukeren, og det vil gjøre det lettere å flytte brukermassen. Skjermbilde 5 viser hvordan manifest blir fremstilt på en mer grafisk metode ved bruk av DevTools.



Skjermbilde 5 Chrome DevTools sin fremstilling av manifest.

Det er en rekke krav fra Chrome for at en bruker skal kunne installere en Progressive Web App. Appen kan ikke være installert fra før, oppnå krav om, har et fullverdig manifest, benytter HTTPS og har en registeret service worker med "fetch" hendelseshåndtering (LePage, 2019). Når kravene ved å være en PWA er møtt, kan Chrome be brukerne om å legge PWA til deres Home screen slik som på skjermbilde 5.



Skjermbilde 6 Fremstilling av nedlasting til åpnet webapp som fungerer offline.

Vi har benyttet manifest i samspill med service worker (se vedlegg 9). Under testingen har vi benyttet en logo-generator for å lage et applikende ikon. I starten ville ikke Chrome godkjenne webappen vår, og den ble derfor lagret på samme måte som et bokmerke. For å ta hensyn til manglende nettleserstøtte er løsningen å legge til flere meta-linker, webappen ble åpnet etter beskrivelsen fra disse meta-linkene. Etter tilpassingene godkjente Chrome webappen vår, og nå fungere den som vist i skjermbilde 6.

6.3.6 Web Background Sync

Background sync gir service worker en ny event som blir utløst av at brukeren forsøker å sende data over internett. Ettersom service worker jobber i bakgrunnen av nettsiden, trenger ikke brukeren å ta hensyn til at den forsøker å sende data. For å bruke Background Sync er vi avhengig av at vi har en funksjon som utløser hendelsen i service worker.

```
function sendChatMessage(message) {
  return addChatMessageToOutbox(message).then(() => {
    // Wait for the scoped service worker registration to get a
    // service worker with an active state
    return navigator.serviceWorker.ready;
  }).then(reg => {
    return reg.sync.register('send-chats');
  }).then(() => {
    console.log('Sync registered!');
  }).catch(() => {
    console.log('Sync registration failed :(');
  });
}
```

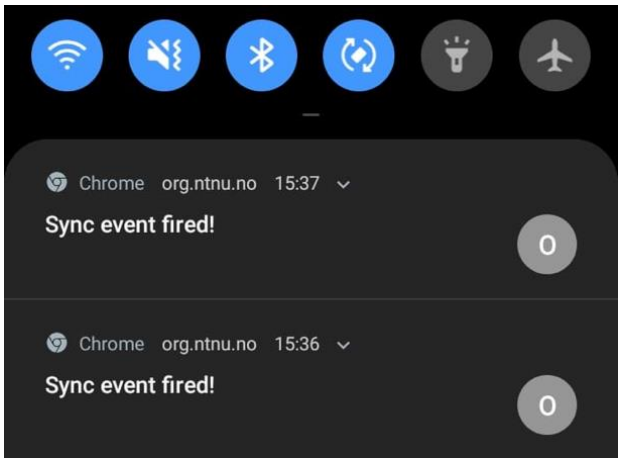
Kodesnutt 6 JavaScript for å aktivere service worker og starte en Background Sync

Kodesnutt 6 (vedlegg 3) er et eksempel på hvordan funksjonen for å utløse en Background Sync event kan se ut. I denne koden får vi først beskjed om at service worker er aktiv før synkroniseringshendelsen blir utløst. Som vi ser av bruken av `.then` benytter dette JavaScript promises. Når hendelsen er blitt registrert, får vi enten beskjed om at den er registrert eller at dette feilet. Dette forteller oss ikke om synkroniseringen er vellykket eller ikke, men hvorvidt service worker har mottatt arbeidsoppgaven riktig (Josh Karlin og Kruisselbrink, 2018).

```
self.addEventListener('sync', function(event) {
  self.registration.showNotification("Sync event fired!");
});
```

Kodesnutt 7 Background Sync i service worker

Vi har benyttet (Archibald, 2019b), sin demo som viser hvordan en sync hendelse viser et varsel på mobilen eller desktop. Kodesnutt 7 viser koden som utløser varslet. Vi trenger ikke å inkludere kode som sier noe om når vi skal synkronisere, det tar service worker seg av automatisk (Archibald, 2019b). Skjermbilde 7 viser hvordan det ser ut for brukeren på mobil. Demoen er veldig enkel og må utvikles videre for å bli en fullverdig webapp. En enkelt hendelse med Background Sync krever ikke tillatelse, men brukeren må tillate varsler for å vise at hendelsen har skjedd.



Skjermbilde 7 Varsler ved to Background Sync hendelser på mobil.

Vi har også testet Background Sync sammen med service worker med å følge guiden til Hume (2016). Skjermbilde 8 viser hva som skjer når knappen på webappen blir aktivert og "doge.png" blir sendt i bakgrunnen ved hjelp av SW.

```
sync registered                                                    (index):37
firing: sync                                                       sw.js:12
sync event fired                                                   sw.js:14
firing: doSomeStuff()                                              sw.js:21
Request successful                                                 sw.js:27
▸ Response {type: "basic", url: "https://org.ntnu.no/owa/BackgroundSync/demo/doge.png", redire
  cted: false, status: 200, ok: true, …}
```

Skjermbilde 8 Konsollogg ved bruk av Background Sync og service worker for å hente et bilde.

Ved å benytte seg av Background Sync kan man sørge for at en bruker som befinner seg i et område med dårlig dekning likevel får til å sende informasjon til webtjeneren. Dette skjer uten at brukeren trenger å gjøre mer enn å trykke send en gang, så lenge han eller hun ikke lukker nettleseren. Dette er en eksperimentell teknologi under utvikling. Per i mai 2019 har Background Sync begrenset støtte i de ulike nettleserne (se tabell 2).

6.3.7 Web Background Fetch

Denne API-en virker på nesten samme måten som Background Sync, men er beregnet for å håndtere større filer. Dessuten fortsetter den å virke i bakgrunnen selv om man går ut av webappen eller lukker nettleserfanen. Som nevnt i teorien er denne API-en ment å ta seg av større handlinger enn Background Sync, og den skal være mer synlig for brukeren ved å vise progresjon på nedlastingen (WICG, 2019). Archibalds demo spør også om tillatelse om man prøver å laste ned mer enn en fil av gangen (Archibald, 2019c).

```
navigator.serviceWorker.ready.then(async (swReg) => {
  const bgFetch = await swReg.backgroundFetch.fetch('my-fetch', ['/ep-
5.mp3', 'ep-5-artwork.jpg'], {
    title: 'Episode 5: Interesting things.',
    icons: [{
      sizes: '300x300',
      src: '/ep-5-icon.png',
      type: 'image/png',
  }],
    downloadTotal: 60 * 1024 * 1024,
 });
});});
```

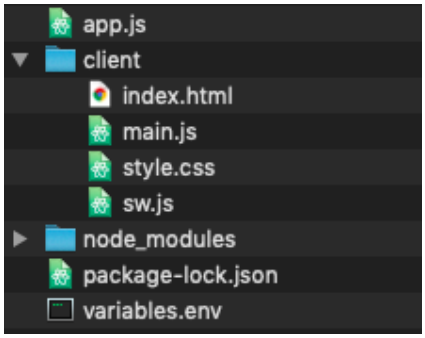
Kodesnutt 8 Eksempel på hvordan Background Fetch kan se ut.

Med unntak av koden som beskriver det som skal lastes ned, i dette tilfellet en MP3-fil, ser vi at det er med en downloadTotal: etterfulgt av tall. Dette er en valgfri parameter som gjør at Background Fetch kan fortelle brukeren hvor langt nedlastingen har kommet. Se kodesnutt 8 for eksempelkode (Archibald og Beverloo, 2019). Salnikov, en utvikler hos Google, ser på Background Fetch API som det neste store skrittet for Progressive Web Apps (Salnikov, 2018) . Background fetch er per dagens dato (mai 2019) kun støttet i nettleseren Chrome 71 og nyere (Archibald, 2019c).

6.4 KOBLINGEN TIL SERVERNE

6.4.1 Node.js

For oss er node.js en helt ny måte å bygge web-servere på. Basert på Chaniotis (2015) forskning, er Node.js bedre egnet enn Apache/PHP både når det gjelder serverens systemressurser og behandling av forespørsler. Som nevnt er Node.js bygget rundt muligheten til å inkludere moduler med ulik funksjonalitet. Disse laster man ned ved hjelp av NPM (Node Packet Manager) kommandoer i ledetekst (Windows) eller terminal (macOS) etter at man har installert Node.js (W3schools.com, U.Å.-b).



Skjermbilde 9 Mappestruktur Node-SW-pushvarsel

Vår testing av Node kom ikke lengre enn å følge en guide for å bygge en enkel side for å legge inn informasjon i en MYSQL-database (Prince, 2018) (se vedlegg 10) og en enkel app som viser et push varsel (Isaiah, 2019)(se vedlegg 11).

Skjermbildet 9 viser mappestrukturen for en Node server. Avhengig av denne strukturen på nettstedet trenger man ikke å ta spesielle hensyn for å få service worker til å fungere sammen med node.js. Service worker må ligge i den samme mappen som index-filen til nettstedet for å kunne fungere optimalt, noe vi kan se i skjermbildet. Fra skjermbildet kan vi se at serveren har filen app.js, som må initieres fra kommandolinjen via funksjonen "Node app.js".

Node.js gjør det mulig å benytte det samme programmeringsspråket for hele utviklingen av et nettsted. Dette sammen med bedre ytelse enn Apache/PHP-servere gjør Node.js til et godt alternativ til bygging av webapper (Chaniotis, Kyriakou og Tselikas, 2015). Node er dårligere enn konkurrentene til å utføre CPU intensivt arbeid. Selv om biblioteket med ulike moduler er stort er mange av modulene dårlig dokumentert(Chrzanowska, 2017).

6.4.2 Server-sent events

Server-sent events er allerede en del av HTML5 standarden, men er ikke støttet i alle nettleserne (se tabell 1). For å benytte oss av denne teknologien, må vi inkludere en "eventsource" som responderer på en matchende henvendelse fra serveren.

```
var evtSource = new EventSource("ssedemo.php");

evtSource.onmessage = function(e) {
  var newElement = document.createElement("li");
  var eventList = document.getElementById('list');

  newElement.innerHTML = "message: " + e.data;
  eventList.appendChild(newElement);
}
```

Kodesnutt 9 Kode for å motta beskjed sendt fra serveren.

Kodesnutt 9 viser hvordan man kan motta en beskjed fra serveren og skrive denne ut i en liste (Mozilla Contributors, 2019k). Selv om Server-sent events kan bidra til å minske bruken av systemressurser og batteri på mobile enheter (WHATWG, 2019b), valgte vi ikke å gå videre med teknologien.

6.4.3 API

I teorien slo vi fast at API-er er den programkoden som henter informasjon fra webservere slik at man får muligheten til å få tak i ny informasjon uten at man trenger å laste inn nettstedet på nytt. Flere av teknologiene vi har vurdert i denne oppgaven faller under denne kategorien.

Vi benytter oss av API-er for å gjøre jobben som utviklere lettere. For eksempel kan vi sammenligne kodesnutt 12 og 15, IndexedDB og Dexie. Koden skrevet med Dexie-biblioteket er enklere å lese, uten at man mister noe av funksjonaliteten. API-en oversetter våre instruksjoner til databasen og vi kan anvende en enklere kode som ikke har mistet noe funksjonalitet. På samme måte har Cache API-en til

service worker instruksjoner om hvordan den skal lagre informasjonen om nettsiden vår i cache. Vi bruker API-er der vi kan, for å utnytte mulighetene i den underliggende teknologien på best mulig måte {Bettilyon, 2018 #204}.

6.4.4 Service worker API

Service worker gir mulighet til å lage gode offline webapper med Background Sync og push varsler, og danner grunnmuren for disse funksjonene. API-et gir utvikleren full kontroll over brukerens offline opplevelse. Alt fra en enkel feilmelding til fullverdige offline webapper kan utvikles ved bruk av SW. En av hovedfunksjonene er muligheten til å styre nettverksforespørsler. Det er kun tillatt med en service worker per scope og er basert på hvor selve SW-filen ligger. Dette gjør at man ikke kan lage en service worker fil som tolker forespørsler fra andre opprinnelser (Gaunt, 2019).

```
//Filer som er kreves for å gjøre den til den offline app.
 var urls = [
'index.html',
'./' // alias for index.html
];
```

Kodesnutt 10 Bruk av cache ved bruk av service worker.

Vi har benyttet service worker for å oppnå offline funksjonalitet sammen med flere av konseptbevisene. Kodesnutt 10 en viser en del av koden bak den offline funksjonalitet i POOWA, viser hvilke filer eller scope som vil være tilgjengelig offline (se vedlegg 14 for komplett kode).

6.4.5 Informasjonskapsler

Informasjonskapslenes funksjon til å lagre data i nettleseren har blitt overtatt av mer moderne API-er (Mozilla Contributors, 2019e). I tillegg er informasjonskapsler ikke er utviklet for bruk sammen med service worker {Degges, 2018 #42. Dette har ført til at vi ikke har benyttet informasjonskapslene for å utvikle offline webapper.

6.5 LAGRING I NETTLESERE

For å få en nettside til å fungere uten dekning må man lagre filene nettsiden består av slik at de er tilgjengelige også uten internett. I teoridelen introduserte vi fire teknologier for bevaring av data og flere former for databaser. I denne delen ser vi nærmere på hvordan det er å jobbe med noen av disse teknologiene, samt ulike fordeler og ulemper. Der det er hensiktsmessig vil vi sammenligne teknologier med tanke på blant annet kompleksitet, nettleserstøtte og bruksområde.

I hovedsak er det to aspekter ved lagring i nettleseren vi må ta hensyn til når vi skal få en webapp til å fungere uten dekning. For det første må vi ha tilgang til HTML, JS og CSS-filene som bygger den statiske rammen av nettstedet. Når vi først har gjort at nettstedet er tilgjengelig uten internett, har vi et behov for å forvalte informasjon som er relevant for brukeren. Den beste løsningen vil la brukeren redigere og tilføye ny data etter at man har mistet internettilkoblingen, og når tilkoblingen er gjenopprettet oppdateres informasjonen på serveren automatisk.

6.5.1 Filesystem API

API-en lar deg simulere et filsystem med «dra og slipp» funksjonalitet lik den vi finner i Windows eller Mac OS. Dette filsystemet er isolert fra andre ressurser i nettleserens sandkasse (Mozilla Contributors, 2019c). Det jobbes ikke med å gjøre denne API-en til en webstandard (ibid.). Chrome, som den eneste nettleseren, har støtte for en utvidet API for Filsystemet som får tilgang til filer i delte mapper på brukerens enhet (Google, U.Å.). Ifølge Cohen (2019) er File system asynkront, noe som vil gjøre det mulig å bruke sammen med service worker. Samtidig påpeker Cohen og Osmani (Osmani og Cohen, 2019) at filsystem ikke er støttet utenfor Chrome. De lignende File and Directory Entries API ((Bell, 2018) og File API (Kruisselbrink og Ranganathan, 2019) er ikke modne nok (Osmani og Cohen, 2019). Begrenset støtte og et bruksområde som ikke stemmer med det vi er ute etter gjorde at vi tidlig valgte bort Filsystem API-en som et mulig lagringsmedia for en OWA.

6.5.2 Webstorage

Som beskrevet i teoridelen er forskjellen på local- og session-storage hvor lenge lagret data blir tatt vare på. Ettersom sessionstorage blir slettet når man går ut av nettleserfanen, er den ikke ideell for vår problemstilling, og vi har ikke jobbet noe videre med den. Begge formene for webstorage er bredt støttet i ulike nettlesere (se tabell 1). Localstorage støtter kun lagring av tekststrenger.

Localstorage er lett å konfigurere. Med utgangspunkt i guiden fra W3school satte vi opp et HTML-form som lagret informasjon i localstorage-objektet (W3schools.com, U. Å.). Dataene i webstorage blir lagret som nøkkel og verdi- par. Det vil si to verdier som hører sammen, noe som gjør det enklere å hente informasjonen fra webstorage senere. Lagringsmediet er synkront og vil derfor ikke virke med service worker (W3schools.com, U. Å.) (mozilla Contributors, 2019a).

Vi avsluttet arbeidet med localstorage etter å ha laget et enkelt script som lagrer innholdet fra et HTML-form.

```
function localStorageTest() {
            var fnavn = document.getElementById("fornavn").value;
            var enavn = document.getElementById("etternavn").value;
            localStorage.setItem("firstname", fnavn);
    localStorage.setItem("lastname", enavn);
    document.getElementById("pt").innerHTML = localStorage.lastname;
            alert("Dataene ble lagret i localStorage.");
            }
```

Kodesnutt 11 Eksempelkode ved bruk av Localstorage

Kodesnutt 11 (se vedlegg 8) viser hvordan man enkelt kan legge tekststrenger i localstorage og skrive ut informasjonen fra et av elementene i HTML-elementet "pt".

Man må kode logikken for å slette data fra localstorage selv. Dette gjelder ikke sessionstorage, som tidligere nevnt slette data når nettleseren blir lukket. Data lagret i webstorage kan også slettes manuelt fra DevTools i nettleseren. Dette er felles for all lagring i nettlesere ved bruk av JavaScript.

6.5.3 Cache API

Denne API-en lar brukeren lagre filene et nettsted består av i nettleserens cache. Cache API-en er en del av service worker API-en (Scales, 2019), men det er mulig å bruke den uten SW (Mozilla Contributors, 2019f). Som tidligere nevnte localstorage, blir ikke cache slettet uten at man går inn og gjør det selv. Dersom operativsystemet har lite ledig plass kan nettleseren slette det som er lagret i cache, uten å varsle brukeren (Mozilla Contributors, 2019f). Dette gjøres per domene, så om det er lagret data fra flere nettsteder i cache kan all dataen fra et av domenene bli slettet, de resterende derimot, forblir urørt.

Som nevnt er denne API-en en del av service worker API-en og dermed asynkron slik at informasjon kan lagres i bakgrunnen, uten at det forstyrrer resten av webappen. Per mai 2019 er ikke dette en webstandard, men har status som et working draft og det jobbes aktivt med å gjøre det til en webstandard i fremtiden (Alex Russel, 2019).

Vi har benyttet cache som en del av service worker. Cache API-en er nevnt her fordi det er lagringsmediet den benytter. For et eksempel på bruk av Cache API-en i samspill med SW, se kodesnutt 10.

6.5.4 LocalForage

Denne teknologien er ikke testet i forbindelse med vår utvikling, men den er svært populær (Potter, 2019). Teknologien velger mellom localstorage, WebSQL og IndexedDB automatisk (Matthew MacPherson, Robert Nyman og Fabbro, 2014). Det gjør at brukeren slipper å tenke på hvilken av de man skal velge som lagringsmedia. Ettersom localstorage er støttet av alle nettlesere, gjør LocalForage at man kan slippe unna med å skrive en kode for lagring av data. Skriptet tar seg av å plassere det på best mulig måte.

LocalForage er et godt valg for utvikling av offline webapper, men blir ikke vurdert videre i vår rapport. Vi valgte å heller se på IndexedDB som har alle de samme funksjonene som LocalForage. Under IndexedDB har vi sett på promise som har vært nyttig for å forstå hvordan service worker fungerer.

6.6 LOKALE DATABASER I NETTLESEREN

6.6.1 WebSQL

Som nevnt i teoridelen er WebSQL en teknologi for lagring av data som tillater bruk av SQL-spørringer. Dette er en teknologi vi tidlig valgte bort, selv om den både er asynkron og støtter transaksjoner av data. Dette skyldes flere faktorer. WebSQL er ikke en webstandard og kommer heller aldri til å bli det. Teknologien har begrenset støtte i nettlesere og det er ingenting som tyder på at flere nettlesere vil støtte WebSQL i fremtiden.

6.6.2 IndexedDB

IndexedDB har flere trekk som minner om WebSQL. Databaseteknologien støtter transaksjoner slik at man unngår problemer med delvis lagrede data. Den er asynkron slik at den vil kunne utføre arbeid uten å forstyrre flyten i webappen som bruker den (Cohen, 2019). Samtidig er det en NoSQL-database som er en webstandard. IDB har bred støtte for ulike formater som strenger, Blob og filer (Mozilla Contributors, 2019j).

Som nevnt i teorien lagres data i ObjectStores, som minner om tabeller i SQL databaser (Google Developers, 2018). Strukturen på innholdet i en ObjectStore trenger ikke defineres på forhånd, utover en definisjon av hva som skal være primærnøkkel. For å søke etter informasjon som er lagret i en IDB oppretter utvikler Indexer som matcher informasjonen i databasen. Indexene gjør at søk etter informasjon blir begrenset til den riktige "kolonnen" i tabellen og på den måten går søkene raskere. En begrensende faktor for søk i en IndexedDB er at databasen skiller mellom store og små bokstaver. Et søk etter "adresse" vil ikke finne "Adresse" (Mozilla Contributors, 2019j).

Selv om IndexedDB er en NoSQL-database og ikke støtter SQL, er det mulig å utføre CRUD-operasjonene (create, read, update, delete) mot data som ligger i databasen. For å gjøre det må man selv skrive programlogikken for det (Alabbas og Bell, 2018). Selve koden bak en IndexedDB er i utgangspunktet basert på JavaScript events og callbacks. I praksis vil det si at man skriver kode som ber databasen om å utføre instrukser, og venter på callback som utløser hendelsen. Funksjonen callback har da informasjonen som inneholder resultatet av instruksen, enten det lykkes eller man får en feilmelding (Mozilla Contributors, 2019j).

```
var request = indexedDB.open("turer");

request.onupgradeneeded = function() {
  var db = request.result;
  var store = db.createObjectStore("turer", {keyPath: "turid"});
  var tittelIndex = store.createIndex("etter_tittel", "tittel", {unique:
true});
  var beskrivelseIndex = store.createIndex("etter_besk", "beskrivelse");
};
request.onsuccess = function() {
  db = request.result;
};
```

Kodesnutt 12 Ren IndexedDB

Kodesnutt 12 viser hvordan man oppretter en database (se vedlegg 5 og 6 for eksempler på IndexedDB). Vi kan se hvordan man benytter callback i request.onupgradeneeded og request.onsuccess. For å teste IDB fulgte vi en veiledning hvor man lagde en enkel liste over gjøremål (West, 2013). Denne webappen fungerte slik at man skrev gjøremål i et tekstfelt og trykket returtast for å lagre. Da ble gjøremålet lagret i IndexedDB og automatisk vist i en liste under tekstfeltet. Ved å trykke på en avkryssingsboks foran gjøremålet ble det fjernet fra databasen og listen.

Etter å ha sett et virkende eksempel på hvor kompleks-koden for bruk av IndexedDB er, valgte vi å jobbe videre med JavaScript-bibliotek som gjør bruk av IndexedDB enklere. Herunder Promise- og Dexie-innpakningene som beskrevet under.

6.6.3 Promise

Som nevnt i teorien ligner IndexedDB med promises på den originale syntaks. Bruken av innpakningen gjør at man kan bruke JavaScript Promises istedenfor Callback og events.

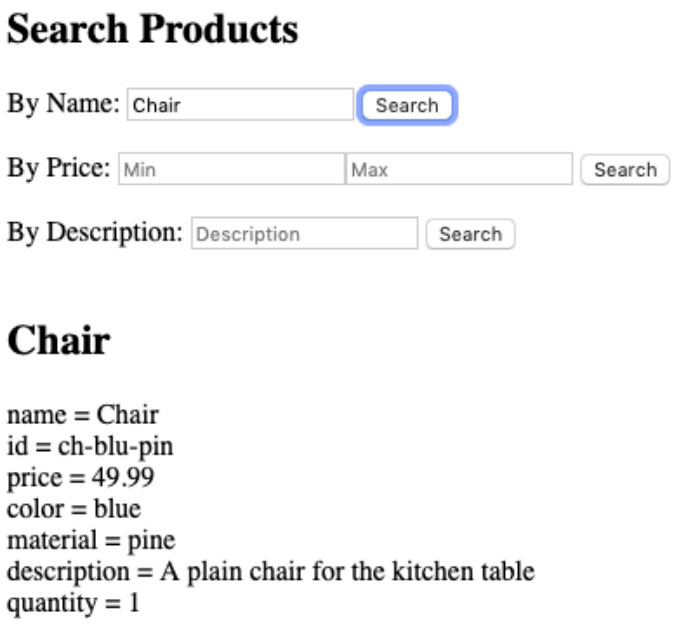
For å bedre vise forskjellene har vi tatt med en tabelloversikt fra Googles veiledning som sammenligner forskjellene mellom IDB og IDB med promises (Google Developers, 2018). Kolonnen for Dexie har vi tatt med for å gjøre sammenligningen mer komplett mot det arbeidet vi har gjort.

__Tabell 4:__ Sammenligning av IndexedDB med innpakninger.

IndexedDB (Google Developers, 2018), med Promises (ibid.) og Dexie (Fahlander, 2017b), (Fahlander, 2017c), (Fahlander, 2018)

Kommando	IndexedDB	med Promises	Dexie
Åpne database	indexedDB.open(navn, versjon)	idb.open(navn, versjon, upgradecallback)	db.open()
Oppgrader til ny versjon av databasen	request.onupgradeneeded	En del av åpningen av databasen	db.version(1) db.version(2) osv.
Suksess	request.onsuccess	.then	.then
Feilmelding	request.onerror	.catch	.catch

Som vi ser av tabell 4 er IndexedDB litt enklere å jobbe med når man benytter Promises biblioteket. Under uttesting av IDB med promises fulgte vi Googles veiledning via Codelabs (Google Developers, U.Å.). I denne veiledningen bygger man en enkel webapp som bruker IndexedDB til å simulere en varebeholdning med informasjon om varer og ordre. Appen inkluderte enkel databehandling av elementene i databasen. Denne webappen inkluderte en søkefunksjon og lot oss bekrefte at man må matche store og små bokstaver for å se resultat (se skjermbilde 10).



Skjermbilde 10 Case sensitivt søk i IDB

Derifra utviklet vi en egen webapp som lagret informasjon fra et HTML-form i nettleseren med IndexedDB med promises. I kodesnutt 13 ser vi hvordan koden for å opprette en database med promise innpakningen er. Som vi ser er forskjellene i kodespråket små sammenlignet med IndexedDB i kodesnutt 12.

```
ar dbHendelse = idb.open('hendelseDb', 1, function(upgradeDb) {
                console.log('Oppretter ObjectStore');
            upgradeDb.createObjectStore('hendelseOS', {keyPath:
'timestamp'});
                console.log('Oppretter Indexer');
            var store = upgradeDb.transaction.objectStore('hendelseOS');
                store.createIndex('timestamp', 'timestamp', {unique:
true});
                store.createIndex('fornavn', 'fornavn');
                store.createIndex('etternavn', 'etternavn');
                store.createIndex('hendelse', 'hendelse');
});
```

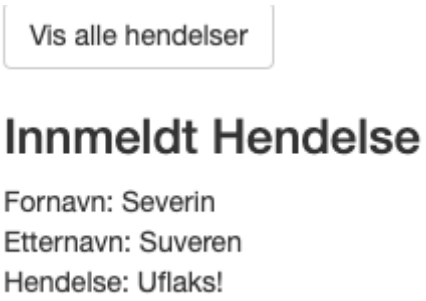
Kodesnutt 13 IndexedDB med Promise oppretter databasen

```
function visAlleHendelser() {
   var tekst = '';
   dbHendelse.then(function(db) {
     var tx = db.transaction('hendelseOS', 'readonly');
     var store = tx.objectStore('hendelseOS');
     return store.openCursor();
   }).then(function showRange(cursor) {
     if (!cursor) {return;}
     tekst += '<h3> Innmeldt Hendelse </h3>';
     tekst += '<p>Fornavn: ' + cursor.value.fornavn + '<br/>';
     tekst += 'Etternavn: ' + cursor.value.etternavn + '<br/>';
     tekst += 'Hendelse: ' + cursor.value.hendelse + '<br/>';
     tekst += '</p>';
     return cursor.continue().then(showRange);
   }).then(function(){
     if (tekst === '') {'<p>Ingen hendelser</p>';}
    document.getElementById('Hendelser').innerHTML = tekst;
   })
```

Kodesnutt 14 Eksempel på bruk av JavaScript med Promises

Webappen ble videreutviklet slik at man kunne finne data fra databasen og endre informasjon tilhørende en brukerprofil og oppdatere dataene i databasen. Funksjonaliteten man får med bruk av JavaScript promises er vist i kodesnutt 13, som viser koden vi brukte for å skrive ut informasjonen vi lagret i nettleseren (se skjermbilde 11).

Kodesnutt 14 viser hvordan promises brukes i praksis sammen med cursor-egenskapen til å gå gjennom innholdet i databasen "dbHendelse" og fremvisning det i nettleseren.



Skjermbilde 11 Resultatet av kodesnutt 14

Den siste iterasjonen av denne webappen blandet inn PHP og hentet informasjon fra en online MySQL-database (se vedlegg 13). Hensikten bak denne kodebiten var å se at teknologiene virket når de ble blandet sammen. Hentingen av informasjon fra MySQL-databasen var ment å simulere at man logget seg på et system og fikk tak i den nødvendige informasjonen for brukeren. Informasjonen ble vist i en tekstboks som man kunne endre på. Vi brukte den samme løsningen for å lagre informasjonen i IndexedDB som i tidligere iterasjoner, det vil si knapper som utløste funksjonene for lagring. Så lenge nettsiden var lastet inn, fungerte det å endre på dataene selv om man ikke hadde nettilkobling som også bekreftet at IndexedDB virker uten at man er koblet til internett.

15. mars 2019 ble IndexedDB med promises-biblioteket oppdatert til versjon 4. Vår kode er fra den 3. utgaven av biblioteket, og det er dermed ikke gitt at kodeeksemplene våre stemmer med den siste utgaven av biblioteket.

6.6.4 Dexie

Som nevnt i teorien er Dexie designet for å gjøre det enklere å bruke IndexedDB, både med tanke på kompleksitet i koden og bedre muligheter for søk mot databasen. Fra hjemmesiden Dexie.org kan man se at innpakningen har en godt utbygd samling av dokumentasjon (Fahlander, 2017d).

Dexie gjør mye av databehandlingen enda enklere sammenlignet med Promises-biblioteket (se tabell 4. For oppgraderinger til en ny versjon av databasen, trenger man i Dexie bare å ta med db.version(2) og føre opp hva som blir endret. Dexie-scriptet tar seg av resten. I både IDB og med Promises krever dette mer kode enn i tabellen hvor vi ser hvordan funksjonen starter (Alabbas og Bell, 2018) (Archibald, 2019a).

Dexie fungerer på samme måte som IndexedDB med promises, men Dexie biblioteket er mer enn fire ganger så stort. For sammenligningen sin del er det fortsatt snakk om små størrelser, og selv om Dexie-biblioteket er større enn Promises er det mindre enn en halv MB. Bibliotekene er ikke spesielt store, 8 kB for Promises og 40 kB for Dexie.

Vi testet ut Dexie biblioteket ved å finne eksempel på en enkel gjøremålsliste-webapp (se vedlegg 12). Denne webappen lar oss skrive inn informasjon som lagres i IndexedDB ved hjelp av Dexie biblioteket. Bruken av biblioteket gjør at koden man trenger å skrive for å behandle dataene blir mye kortere og

enklere å lese. For gjøremålslisten er hele programkoden på bare 45 linjer. Det inkluderer å legge inn, og slette data, samt koden som skal til for å vise innholdet i databasen.

Kodesnutt 14 viser kode for å opprette en database og legge inn data. Vi kan også se hvordan koden benytter seg av JavaScript promises funksjonen `.then`,. I dette tilfellet legger vi inn informasjonen i databasen og så utføres koden bak `.then` og tekstfeltet input tilbakestilles. Etter det starter funksjonen refreshView som lar oss se alle gjøremålene i databasen.

```
function onSubmit(e) {
  e.preventDefault();
  db.todo.put({ text: input.value, _id: String(Date.now()) })
    .then(function() {
      input.value = '';
    })
    .then(refreshView);
}
```

Kodesnutt 15 Oppretting av database med Dexie og lagring av tekststreng.

Foruten simplifisering av kode har Dexie som mål å gjøre det enklere å søke etter informasjon, samt behandle eventuelle feil som oppstår. Dexies søkefunksjon bruker en form for spørringer som minner om SQL, selv om IndexedDB og Dexie er NoSQL databaser. For å søke etter informasjon i Dexie uten å ta hensyn til store og små bokstaver, inkluderer man IgnoreCase i den konstruerte spørringen. Eksempelkoden fra kodesnutt 15 viser dette.

```
await db.friends
    .where("name").equalsIgnoreCase("josephine")
    .each(friend => {
        console.log("Found Josephine", friend);
    });
```

Kodesnutt 16 Eksempel på søkefunksjon som ignorerer store og små bokstaver (Fahlander, 2018).

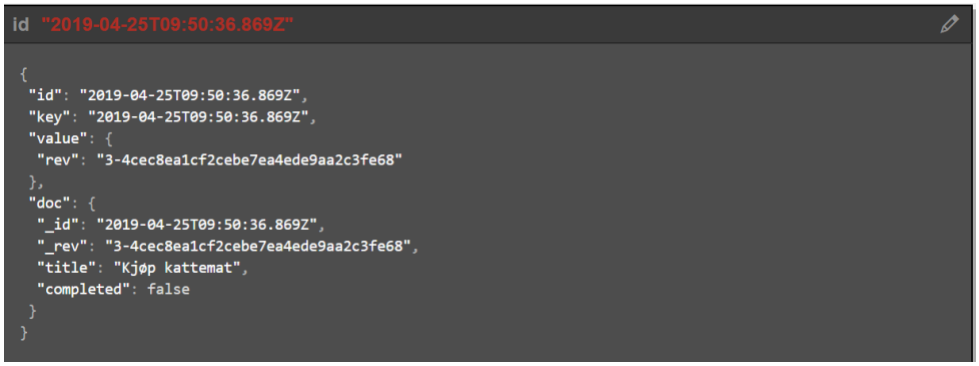
6.6.5 CouchBase

Databasen er spesielt utviklet for bruk av interaktive apper og mobilapper. Den benytter JSON-filer for lagring av dokumenter og NoSQL (Chopade og Dhavase, 2017). Databasen kan ha opptil tre replikasjoner av ulike data, men trenger lokal installasjon på utviklingsmaskinen.

Vi har benyttet CouchBase på samme måte som CouchDB for lokal lagring. Da forskjellen ikke er nevneverdig mellom de to lokale databasene, har vi valgt å fokusere på CouchDB. Sistnevnte vil fungere bedre sammen med PouchDB på grunn av flere fellestrekk.

6.6.6 Apache CouchDB

CouchDB er en NoSQL database designet for lokal datareplikering og skalering. Jason Smith hevder *"CouchDB is bad at everything, except syncing".* Videre forteller Smith at synkronisering er den viktigste funksjon for mange ulike typer program (Pouchdb, 2018). Skjermbilde 12 viser hvordan et element i databasen blir vist frem i JSON, der det også er mulig med tekst i tabell eller som metadata. CouchDB er gunstig når konflikthåndtering er viktig.



Skjermbilde 12 JSON fremstilling av data i en database i CouchDB.

Vi har benyttet CouchDB ved bruk av videreutviklingen av PouchDB sin guide. Etter å få webappen til å fungere med Cloudant, som er en skybasert tjeneste. Vi har konkludert med at vi ønsker å benytte en ekstern database som ikke er lokal videre i utviklingen. Ved bruk av lokale databaser ville ikke brukerne få samme dataene på ulike enheter. Selve koden for å opprette replikasjon er lettforståelige (se kodesnutt 17).

```
//Legg til CouchDB som ekstern db
var remoteDB = new PouchDB('http://127.0.0.1:5984/todos/')

localDB.replicate.to(remoteDB).on('complete', function () {
  // Suksess!
}).on('error', function (err) {
  // Auda!
});
```

Kodesnutt 17 Legge til CouchDB som en ekstern database

6.6.7 PouchDB

PouchDB gjør det mulig å ha offline funksjonalitet på flere ulike enheter som kan synkroniseres ved hjelp av en skybasert database (Pouchdb).

Ved å sette opp en webapp etter veiledningen til PouchDB, kan vi konkludere med at teknologien er hovedsakelig enkel å forstå. Den gir muligheten til å sette opp kopiering av data til en ekstern server, og har muligheten til å velge hvor ofte denne kopieringen skal skje. I vårt tilfelle velger vi å kopiere dataene kontinuerlig, som er hovedfordelen med PouchDB. Kodesnutten nedenfor (kodesnutt 18, se vedlegg 15) er endring vi har gjort ved veiledning, men valgte å bruke et API for å bestemme hvor lenge funksjonen skal vente før den forsøker å opprette tilgang igjen. Funksjonen starter med 1000 millisekunder (1 sekund) og tripler det hver gang det ikke fungere frem til brukeren er online igjen. Ved utvikling måtte vi endre antall eventEmitters i selve PouchDB-filen. Dette førte til at noen av løsningene må ha PouchDB liggende lokalt.

```
//Gjenta replikasjon
localDB.replicate.to(remoteDB, {
  live: true,
  retry: true,
  back_off_function: function (delay) {
    if (delay === 0) {
      return 1000;
  }
    return delay * 3;
 }
});
```

Kodesnutt 18 Kode som bestemmer tidsintervallene for synkronisering mot en ekstern database.

6.6.8 IBM Cloudant

Cloudant er likhet med de andre databasene NoSQL-basert, og det som er felles for disse databasene er at de fungerer meget bra til skalering (Camp *et al.*, 2018). Cloudant er ekstra gunstig på grunn av at den er skybasert og dermed har mulighet til å øke kapasiteten raskt, men det er mot betaling.

```
var remoteDB = new PouchDB('https://85a49bf5-e8c8-47d0-9c69-cc83c43b2a74-
bluemix:33c02e667d8c25acf006562587301b66e16779d3e8e205fa7cded6f6ff34123f@8
5a49bf5-e8c8-47d0-9c69-cc83c43b2a74-
bluemix.cloudantnosqldb.appdomain.cloud/todos');
```

Kodesnutt 19 Hvordan legge til Cloudant som en ekstern database

Under videreutviklingen av veiledningen til PouchDB har vi benyttet Cloudant for å få en database som fungerer på kryss av ulike enheter. For å finne riktig URL vist i kodesnutt 18 måtte vi lage en tjenesteleverandører-legitimasjon og deretter legge til navnet på databasen i URL-en. Under ser man data fra webappen som samspiller med Cloudant, og dataen oppdaterer seg øyeblikkelig (se skjermbilde 13). Ved offline bruk blir dataene opplastet på databasen når brukeren har oppnådd online status igjen.

	_id	completed	title
☐	2019-04-25T12:23:15.675Z	false	Safari kjeks
☐	2019-05-08T09:09:00.548Z	false	Kake
☐	2019-05-12T16:14:19.729Z	false	Mel

Skjermbilde 13 Oversikt over data som ligger på Cloudant.

6.7 SAMSPILL MELLOM DE ULIKE TEKNOLOGIER, POOWA.

Vi har valgt å sammenslå teknologiene PouchDB, Cloudant, manifest og service worker for å vise hvordan teknologiene samspiller (se vedlegg 14). Webappen går under navnet POWA som står for Proof Of Offline Web App. POOWA er utviklet for å fungere offline, uten å forenkle funksjonene. For en profesjonell utviklet webapp anbefales det å lage en forenklet versjon til offline modus. Dette er fordi

SW legger de nødvendige filene for offline-funksjonalitet i cache. Data som blir lagt i huskelisten blir først lagret lokalt i nettleseren ved hjelp av PouchDB. Som benytter IndexedDB under overflaten. For at dataene skal være tilgjengelig til alle brukere av webappen, har vi valgt å overføre dataene til den skybaserte databasen Cloudant. Vi har benyttet manifest i samspill med service worker, samt oppfylt andre krav slik at Chrome tillater at brukerne kan legge til webappen på startsiden.

```
//tilkobling status
function isOnline() {
var connectionStatus = document.getElementById('connectionStatus');
//Melding hvis offline:
if (navigator.onLine) {
connectionStatus.innerHTML = 'Du er online!';
} else {
 // hvis offline:
connectionStatus.innerHTML = 'Du er offline. Alle forespørsler vil lagt i
kø og synkronisert når du er online igjen.';
}}
 window.addEventListener('online', isOnline);
window.addEventListener('offline', isOnline);
 isOnline();
```

Kodesnutt 20 Kode for varsling ved endringer i internettilkobling.

Øverst i webappen har vi valgt å legge til varsler som melder om internettilkoblingen, dette er for å vise manglende internettilgang, siden POOWA ellers oppfører seg vanlig. Se kodesnutten 20 for koden bak varslene.

Grunnen til at vi har valgt PouchDB er at på tross av sin mindre kjente posisjon så er den godt dokumentert. Vi ville teste hvordan databasen fungerte i samspill med andre, siden en webapp ofte vil inneholde mer enn kun webteknologi. I den sammenheng ble det naturlig å bruke Cloudant siden vi allerede hadde testet at skybaserte databaser er ideelle for offline webapper. Vi har hatt flere tester på ulike bruk av IDB, slik at vi visste at det funker godt med andre teknologier.

Bruk av manifest er for å gjøre den mer tilgjengelig og er et viktig element i en PWA. Testing av manifest sammen med andre teknologier vil være med å vise om teknologiene vil fungere som en PWA. Det samme gjelder service worker, men hovedfokuset for SW er å skape en fungerende offline webapp. Som er problemstillingen for denne rapporten.

Vi valgt å ikke benytte teknologier i POOWA som vi har betegnet som ikke egnet for å løse offline funksjonalitet og teknologier med tilsvarende egenskaper. Sistnevnte ville ført til mange forskjellige webapper som gjør tilsvarende det samme.

Av teknologiene vi har sett på samlet sett gir ikke webstorage en bedre brukeropplevelse i forhold til alternativene. Informasjonskapsler og filesystem har utilstrekkelig funksjonalitet for å løse vår problemstilling. Både WebSQL og offline.js kan i teorien brukes, men begge er utdaterte teknologier.

I tillegg til denne komplette løsningen har vi vurdert flere ulike sammensetninger av teknologier som skal gi den samme funksjonaliteten. For lagring av ressurser som trengs for å oppnå offline funksjonalitet, må SW eller Cache API benyttes.

For lagring av brukerinformasjon lokalt har vi funnet flere mulige løsninger. Felles for disse er at de er alle NoSQL databaser som gjør at skaleringen blir enklere. Denne funksjonaliteten dekkes av databaseteknologier som Dexie, CouchDB, Couchbase og PouchDB. For å lagre data eksternt benyttes Cloudant og tilsvarende skybaserte databaser. Ved hjelp av SW sin Background Sync vil det være teoretisk mulig å oppnå den samme funksjonaliteten ved bruk av andre databaseklienter som phpMyAdmin.

Et mulig alternativ til vår POOWA løsning, kan være å benytte Dexie til lagring i nettleseren og en MySQL klient på en ekstern server for synkronisering og permanent lagring av data. SW bidrar i dette eksemplet med offline funksjonaliteten til å få webappen til å fungere uten nettilkobling og Background Sync sørger for at oppdatert informasjon blir sendt til serveren.

7 KONKLUSJON

For kommunikasjon mellom en webapp og en webtjener er man avhengig av å være koblet til internett. Når man som bruker har lastet ned ressursene til en nettside, er brukeren ikke i like stor grad avhengig av tilkoblingen. I dag finnes det en rekke teknologier som legger til rette for bedre funksjonalitet offline. Dette innebærer at man kan designe en webapp slik at man har tilnærmet den samme brukeropplevelsen uten nettilkobling. En slik OWA løser kundebehovet IT Data AS la frem i oppgaveteksten (se vedlegg 2).

I løpet av vår studie har vi konkludert med at det er mange teknologier som tilrettelegger for bedre funksjonalitet i offline webapper. Offline webapper er utviklet for å gi tilnærmet den samme brukeropplevelsen som en mobilapp, men en Progressive Web Apps (PWA) må oppfylle en rekke kriterier for at Chrome skal godkjenne og installere den. Ved økt bruk av webapper har mulighetene til distribusjon av webapper og utvikling av API-er som gir tilgang til funksjoner i operasjonssystemet økt tilsvarende. OWA benytter velkjente feilsøkingsverktøy, har øyeblikkelig oppdateringer og oppdateres så godt som likt på alle plattformer, men bruker har ingen tilgang til endringsloggen. Webapper som benytter avanserte programmeringsspråk bruker også mer strøm enn andre mobilapper fordi det er mer krevende for enheten å tolke språket.

Det å legge til ekstra API-er for å oppnå tilgang til OS krever ekstra arbeid for utvikleren. Samlet sett har OWA flere fordeler enn mobilapper og PWA, så lenge brukerkravene kan dekkes ved bruk av nettleser.

Som vi viser i delkapittelet om sikkerhet er det flere trusler man bør ta høyde for når man bygger en moderne webapp. Spesielt er det viktig å være bevisst på sikring av informasjon dersom man lagrer

denne lokalt i nettleseren. Bruk av gode sikkerhetstiltak er som bruk av forsikring. Det lønner seg når uhellet er ute og kan bli svært kostbart dersom man ikke har det.

Vi ser på bruken av JavaScript som en essensiell del for å oppnå god funksjonalitet for brukeren, uavhengig av om det er snakk om online eller offline. JavaScript benyttes som kodespråk for alt fra SW til lagring av data. JavaScript-bibliotek som UpUp gjør det enkelt for utviklere å tilby offline funksjonalitet uten å sette seg inn i bruken av SW. Ved økt forståelse anbefaler vi å heller benytte service worker uten bruk av UpUp, og særlig ved modifisering og feilsøking av koden.

Background sync er velegnet til å sende små datamengder over internett, og arbeider i bakgrunnen. For større datafiler anbefaler vi bruk av Web Background Fetch. Begge disse teknologiene har per i dag begrenset nettleserstøtte. Manifest bidrar til å gjøre at brukeropplevelsen blir tilsvarende en mobilapp, og er et av kravene til Chrome for å få installert webappen på mobilen. For bruker kan dette oppleves som masing på lik linje med reklame, siden det kan være uønsket. Det finnes innstillinger på hvor ofte elementet skal dukke opp slik at man ikke ødelegger for brukeren.

Vi mener at service worker og utvidelsene som tilhører teknologien er noe av det mest sentrale for å kunne oppnå offline funksjonalitet i en webapp. SW er sentral fordi teknologien kan styre nettverksforespørsler, er asynkron, fungerer med Background Sync og push varsler. Det kan kun være en service worker per scope, og den kan dermed ikke tolke forespørsler fra andre opprinnelser. Ved utvikling har service worker vært sentralt ved uttesting av flere teknologier. En del av SW er Cache API som lagrer brukerdata i nettleseren og er godt egnet for lagring av systemressurser. Bare teknologier som er asynkrone virker sammen med SW.

Avhengig av hvor mye og hva slags trafikk man forventer over webserveren kan Node.js fungere svært bra som ryggraden for en moderne web. En stor fordel ved å benytte Node også til serverside-programmering er at det er bygget på JavaScript som gjør at utvikleren ikke trenger å beherske mange forskjellige programmeringsspråk.

Flere av databaseteknologiene vi har undersøkt i vår studie har IndexedDB i bunnen. IndexedDB er webstandard og støtter flere typer ulike dataformater. Den er dermed godt egnet til å være en grunnmur for de andre databasene. IndexedDB har utfordringer knyttet til feilbehandling, spørringer og kompleksitet. Dette er problemer Dexie og Promises unngår.

Promises legger til funksjonalitet for å bruke de moderne JavaScript-promises til kodingen av databasen, Dexie går lengre og gir oss muligheten til å søke etter informasjon uavhengig av store og små bokstaver. For arbeid mot IndexedDB anbefaler vi at man velger en av disse innpakningene fremfor å jobbe direkte med IDB.

NoSQL databasene CouchBase, Apache CouchDB, PouchDB og IBM Cloudant kan virke tilsynelatende like, men har ulike egenskaper. CouchBase er utviklet for å fungere på apper, men vil kun fungere på en lokal server. Apache CouchDB har et lett oppsett og har en velfungerende versjonslogg, men på samme måte som Couchbase fungere den kun lokalt. PouchDB er det alternativet vi har valgt å benytte i størst

grad, siden den krever minimalt med oppsett, har fokus på sikkerhet og er offline first. PouchDB er mindre kjent enn de nevnte databasene over, men har tilstrekkelig med dokumentasjon ved problemer. Fokus på offline first er en viktig grunn til at vi anbefaler denne lokale databasen. Ved bruk av IBM som er en kjent leverandør, og deres skybasert databaseløsning Cloudant slik at dataene er tilgjengelig på tvers av nettleserne. Vi fikk kombinert kjente og ukjente databaseteknologier for offline webapper, samt testet både lokale og skybaserte løsninger.

Vi har funnet teknologier som gjør det mulig å få tilnærmet samme funksjonalitet med en offline webapp som med en native app. Våre konseptbevis viser også at dette er mulig i praksis, ikke bare i teorien. Vi mener at det i de fleste tilfellene vil være mulig å utvikle en offline webapp som er så vellykket at den kan konkurrere med native og hybride apper.

Minidisk ble omtalt til å være det neste store, men ble erstattet av mp3 spillere. Vi mener at webapper vil ha oppleve en bedre fremtid (Plikk, 2013).

7.1 VIDERE FORSKNING

For videre forskning vil vi anbefale å se nærmere på flere teknologier innenfor offline webapper. Et alternativ er den dokumentorient databasen MongoDB Atlas, som en skybasert database i likhet med IBM Cloudant (mongoDB, U.Å). Videre forskning innenfor samme problemstilling vil inkludere det å teste ut funksjonalitet til MongoDB. Hoodie er også en aktuell database-teknologi som tilsvarer databasene vi benyttet i vår studie. Den kan teoretisk erstatte blant annet PouchDB og Cloudant.

LocalForage er godt alternativ til IndexedDB ved at den velger lagringsmedia automatisk (Matthew MacPherson, Robert Nyman og Fabbro, 2014), forskningen ville ha vært å forsøke å sammenligne de to lagringsmedie-teknologiene. Web Background Fetch er utviklet for å håndtere store datamengder og fungerer asynkront. I vår rapport har vi ikke testet teknologien om dette er mer enn teoretisk mulig, noe som ville ha vært naturlig for videre forskning.

Ved annen videre forskning ville vi ha sett nærmere på konflikthåndtering av data, der dataene er lagret både en lokalt og en ekstern server. I forbindelse med konflikthåndteringen mener vi det vil være naturlig å se på hvordan teknologiene oppfører seg i større skala, med flere samtidige brukere. Vi ville også ha sett nærme på sikkerhet til dataene som ligger lagret lokalt. Er det mulig og hensiktsmessig å sikre data ved kryptering?

Et naturlig neste skritt synes vi vil være å finne ut hvordan man kan kommersialisere en OWA og samtidig ta vare på personvern på en forsvarlig måte. Informasjonen om hvordan man lager en OWA er tilgjengelig, men mangler en felles kanal. En slik kanal kan etablere bånd mellom utviklere av teknologiene, webutviklere og undervisningsinstitusjoner.

Vi ser for oss at i fremtiden kan en kommunikasjonschip erstatte mobil med tanke på kommunikasjon mellom ulike enheter. Dette vil ikke erstatte de mer avanserte oppgavene smarttelefonene i dag utførere som for eksempel å ta opp video.

Hvordan vil Internet of Things (IoT) som inkluderer stadig flere enheter bli påvirket i områder med dårlig dekning? Kan IoT tjene på å være koblet til OWA? Vil bruk av offline webteknologier bidra til å forbedre funksjonalitet til IoT i fremtiden? For eksempel for et kjøleskap som mister internettilkobling og får feil med synkroniseringen av innhold i kjøleskapet.

8 REFERANSELISTE

Adcom (U.Å) *JOHN ERIK JOHNSEN*. Tilgjengelig fra: https://www.adcom.no/ansatte/john-erik-johnsen/ (Hentet: 14.05.2019).

Aderinokun, I. (2018) Web workers vs Service workers vs Worklets, *Bitsofcode*. Tilgjengelig fra: https://bitsofco.de/web-workers-vs-service-workers-vs-worklets/ (Hentet: 28.02.2019).

Alabbas, A. og Bell, J. (2018) Indexed Database API 2.0. Tilgjengelig fra: https://www.w3.org/TR/IndexedDB-2/ (Hentet: 18.01.2019).

Alabbas, A. og Bell, J. (2019) *Indexed Database API 3.0*. Tilgjengelig fra: https://w3c.github.io/IndexedDB/ (Hentet: 13.05.2019).

Alex Russel, J. S., Jake Archibald, Marijn Kruisselbrink (2019) *Service Workers Nightly*. Tilgjengelig fra: https://w3c.github.io/ServiceWorker/ (Hentet: 11.03.2019).

An, D. (2018) *Find out how you stack up to new industry benchmarks for mobile page speed*. Tilgjengelig fra: https://www.thinkwithgoogle.com/marketing-resources/data-measurement/mobile-page-speed-new-industry-benchmarks/ (Hentet: 26.04.2019).

Apache CouchDB (U.Å) *Apache CouchDB*. Tilgjengelig fra: http://couchdb.apache.org/ (Hentet: 29.04.2019).

Appel, A. P. *et al.* (2016) Destiny: a cognitive mobile guide for the olympics, i *Proceedings of the 25th International Conference Companion on World Wide Web*, *29.04.2019*. International World Wide Web Conferences Steering Committee, s. 155-158.

Apple Inc (2017) Om sikkerhetsinnholdet i Safari 10.1.1. doi: https://support.apple.com/no-no/HT207804.

Archibald, J. (2019a) *IndexedDB, but with promises*. Tilgjengelig fra: https://github.com/jakearchibald/idb (Hentet: 30.04.2019).

Archibald, J. (2019b) *Introducing Background Sync*. Tilgjengelig fra: https://developers.google.com/web/updates/2015/12/background-sync (Hentet: 30.04.2019).

Archibald, J. (2019c) *Introducing Background Fetch*. Tilgjengelig fra: https://developers.google.com/web/updates/2018/12/background-fetch (Hentet: 02.05.2019).

Archibald, J. (2019d) *The offline cookbook*. Tilgjengelig fra: https://developers.google.com/web/fundamentals/instant-and-offline/offline-cookbook/#on-install-as-dependency (Hentet: 04.04.2019).

Archibald, J. og Beverloo, P. (2019) *Background Fetch*. Tilgjengelig fra: https://wicg.github.io/background-fetch/ (Hentet: 30.04.2019).

Ater, T. (2018) *UpUp*. Tilgjengelig fra: https://github.com/TalAter/UpUp/blob/master/docs/README.md (Hentet: 25.04.2019).

Bar, A. (2019) *Can I rely on the Web Platform features to build my app?* Tilgjengelig fra: https://whatwebcando.today/ (Hentet: 26.04.2019).

Basques, K. (2019a) Get Started with Remote Debugging Android Devices. Tilgjengelig fra: https://developers.google.com/web/tools/chrome-devtools/remote-debugging/?utm_source=dcc&utm_medium=redirect&utm_campaign=2016q3 (Hentet: 26.04.2019).

Basques, K. (2019b) Debug Progressive Web Apps. Tilgjengelig fra: https://developers.google.com/web/tools/chrome-devtools/progressive-web-apps (Hentet: 29.04.2019).

Bell, J. (2018) *File and Directory Entries API*. Tilgjengelig fra: https://wicg.github.io/entries-api/#api-files-directories (Hentet: 29.04.2019).

Bootstrap (U.Å) *Bootstrap*. Tilgjengelig fra: https://getbootstrap.com/ (Hentet: 30.04.2019).

bos, W. (2016) *How to Sanitize Data with ES6 Template Strings*. Tilgjengelig fra: https://wesbos.com/sanitize-html-es6-template-strings/ (Hentet: 13.05.2019).

Brønnøysundregistrene (2017) Nøkkelopplysninger fra Enhetsregisteret. Tilgjengelig fra: https://w2.brreg.no/enhet/sok/detalj.jsp?orgnr=966946873 (Hentet: 14.05.2019).

Busch, T. (2013) *Akademisk skriving -for bachelor- og masterstudenter*. Oslo: Fagbokforlaget.

Caceres, M. *et al.* (2018) *Web App Manifest*. Tilgjengelig fra: https://www.w3.org/TR/2018/WD-appmanifest-20181212/ (Hentet: 27.02.2018 2019).

Caceres, M. *et al.* (2019) *Web App Manifest*. Tilgjengelig fra: https://www.w3.org/TR/appmanifest/ (Hentet: 13.05.2019).

Camp, B. *et al.* (2018) A new cross-platform architecture for epi-info software suite, *BMC bioinformatics*, 19(11), s. 359. Tilgjengelig fra: https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-018-2334-8 (Hentet: 03.05.2019).

Chaffey, D. (2018) Mobile marketing statistics compilation. Tilgjengelig fra: https://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/ (Hentet: 03.05.2019).

Chaniotis, I. K., Kyriakou, K.-I. D. og Tselikas, N. D. (2015) Is Node. js a viable option for building modern web applications? A performance evaluation study, *Computing*, 97(10), s. 1023-1044. Tilgjengelig fra: https://link.springer.com/content/pdf/10.1007%2Fs00607-014-0394-9.pdf (Hentet: 20.03.2019).

Chopade, M. R. M. og Dhavase, N. S. (2017) Mongodb, couchbase: performance comparison for image dataset, i *2017 2nd International Conference for Convergence in Technology (I2CT)*. IEEE, s. 255-258.

Chrzanowska, N. (2017) Why to Use Node.js: Pros and Cons of Choosing Node.js for Back-end Development. Tilgjengelig fra: https://www.netguru.com/blog/pros-cons-use-node.js-backend (Hentet: 15.04.2019).

Cohen, M. (2019) *Web Storage Overview*. Tilgjengelig fra: https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/ (Hentet: 18.01.2019).

Copes, F. (2018) First Steps with the Cache API. Tilgjengelig fra: https://alligator.io/js/cache-api/ (Hentet: 02.05.2019).

Couchbase (U.Å) Supported Web Browsers. Tilgjengelig fra: https://docs.couchbase.com/server/6.0/install/install-browsers.html (Hentet: 03.02.2019).

Dalland, O. (2007) *Metode og oppgaveskriving for studenter*. Oslo: Gyldendal Akademisk.

Degges, R. (2018) Please stop using local storage *Random Thoughts of a Happy Programmer*. Tilgjengelig fra: https://www.rdegges.com/2018/please-stop-using-local-storage/ (Hentet: 14.02.2019).

Developers, G. (2019a) *Distribute a Web App through the Chrome Web Store*. Tilgjengelig fra: https://developers.google.com/apps-script/guides/distribute-web-app (Hentet: 01.05.2019).
Developers, G. (2019b) Lighthouse. Tilgjengelig fra: https://developers.google.com/web/tools/lighthouse/ (Hentet: 09.05.2019).
Deveria, A. (2019a) *Can I use CSP?* Tilgjengelig fra: https://caniuse.com/#search=CSP (Hentet: 20.02.2019).
Deveria, A. (2019b) *Can I use CORS?* Tilgjengelig fra: https://caniuse.com/#search=cors (Hentet: 02.05.2019).
Deveria, A. (2019c) *Can I use IndexedDB?* Tilgjengelig fra: https://caniuse.com/#search=IndexedDB (Hentet: 26.02.2019).
Deveria, A. (2019d) *Can I use Web App Manifest?* Tilgjengelig fra: https://caniuse.com/#search=manifest (Hentet: 02.05.2019).
Deveria, A. (2019e) *Can I use Background Sync?* Tilgjengelig fra: https://caniuse.com/#search=background%20sync (Hentet: 02.05.2019).
Deveria, A. (2019f) *Can I use Filesystem & FileWriter API*. Tilgjengelig fra: https://caniuse.com/#search=filesystem (02.05.2019).
Deveria, A. (2019g) *Can I use Web Storage?* Tilgjengelig fra: https://caniuse.com/#search=web%20storage (Hentet: 02.05.2019).
Deveria, A. (2019h) *Can I Use Service Worker?* Tilgjengelig fra: https://caniuse.com/#search=service%20worker (Hentet: 02.05.2019).
Deveria, A. (2019i) *Can I use Web SQL?* Tilgjengelig fra: https://caniuse.com/#search=websql (Hentet: 02.01.2019).
Downs, R. (2017) UN: Majority of world's population lacks internet access, *UPI*. Tilgjengelig fra: https://www.upi.com/Top_News/World-News/2017/09/18/UN-Majority-of-worlds-population-lacks-internet-access/6571505782626/ (Hentet: 14.05.2019).
Dua, K. (2019) The Complete Guide to Mobile App Development: Web vs. Native vs. Hybrid. Tilgjengelig fra: https://clearbridgemobile.com/mobile-app-development-native-vs-web-vs-hybrid/#Advantages_of_Native_Apps (Hentet: 29.04.2019).
Ecma (2017) *The JSON Data Interchange Syntax*. Tilgjengelig fra: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf (Hentet: 17.05.2019).
Ecma (2018) Standard ECMA-262. Tilgjengelig fra: http://www.ecma-international.org/publications/standards/Ecma-262.htm (Hentet: 20.04.2019).
Ecma (U.Å) *History of Ecma*. Tilgjengelig fra: https://www.ecma-international.org/memento/history.htm (Hentet: 02.05.2019).
European Commission (2016) Browsers support. Tilgjengelig fra: http://ec.europa.eu/ipg/standards/browsers/index_en.htm (Hentet: 03.05.2019).
European Commission (2018) Cookies. Tilgjengelig fra: http://ec.europa.eu/ipg/basics/legal/cookies/index_en.htm (Hentet: 03.05.2019).
Fahlander, D. (2017a) *The three main limitations of IndexedDB*. Tilgjengelig fra: https://dexie.org/docs/The-Three-Main-Limitations-of-IndexedDB (Hentet: 02.05.2019).
Fahlander, D. (2017b) *Dexie.open()*. Tilgjengelig fra: https://dexie.org/docs/Dexie/Dexie.open() (Hentet: 02.02.2019).
Fahlander, D. (2017c) *Design*. Tilgjengelig fra: https://dexie.org/docs/Tutorial/Design#database-versioning (Hentet: 03.02.2019).
Fahlander, D. (2017d) *Dexie.js*. Tilgjengelig fra: https://dexie.org/docs/Dexie.js.html (02.05.2019).

Fahlander, D. (2018) *API Reference*. Tilgjengelig fra: https://dexie.org/docs/API-Reference#exception-handling (Hentet: 01.03.2019).
Firtman, M. (2019) Google Play Store now open for Progressive Web Apps Tilgjengelig fra: https://medium.com/@firt/google-play-store-now-open-for-progressive-web-apps-ec6f3c6ff3cc (Hentet: 01.05.2019).
Gardner, L. D. (2016) Making A Service Worker: A Case Study, *SmashingMag*. Tilgjengelig fra: https://www.smashingmagazine.com/2016/02/making-a-service-worker/ (Hentet: 27.02.2019).
Gaunt, M. (2019) Service Workers: an Introduction. Tilgjengelig fra: https://developers.google.com/web/fundamentals/primers/service-workers/ (Hentet: 27.02.2019).
Gaunt, M. og Kinlan, P. (2019) The Web App Manifest. Tilgjengelig fra: https://developers.google.com/web/fundamentals/web-app-manifest/ (Hentet: 15.02.2019).
Google (U.Å.) chrome.fileSystem. Tilgjengelig fra: https://developer.chrome.com/apps/fileSystem (Hentet: 29.04.2019).
Google Developers (2018) *Working with IndexedDB*. Tilgjengelig fra: https://developers.google.com/web/ilt/pwa/working-with-indexeddb (Hentet: 27.02.2019).
Google Developers (2019) *Introduction to Service Worker*. Tilgjengelig fra: https://developers.google.com/web/ilt/pwa/introduction-to-service-worker (Hentet: 28.02.2019).
Google Developers (U.Å.) *Developing Progressive Web Apps 10.0: IndexedDB API*. Tilgjengelig fra: https://codelabs.developers.google.com/codelabs/pwa-indexed-db/index.html?index=..%2F..dev-pwa-training#0 (Hentet: 11.03.2019).
Greasidis, T. og MacPherson, M. R. (2018a) Localforage. Tilgjengelig fra: https://github.com/localForage/localForage (Hentet: 25.04.2019).
Greasidis, T. og MacPherson, M. R. (2018b) *Offline storage, improved. Wraps IndexedDB, WebSQL, or localStorage using a simple but powerful API.* Tilgjengelig fra: https://github.com/localForage/localForage (Hentet: 12.04.2019).
Haraldstad, A.-M. B. og Christophersen, E. (2004) Litteratursøk og personlige referansedatabaser. Oslo: Gyldendal akademisk, 2004, s. s. 115-151.
Hevner, A. R. *et al.* (2004) Design Science in Information Systems Research, *Management Information Systems Quarterly*, 28. Tilgjengelig fra: https://www.researchgate.net/publication/201168946_Design_Science_in_Information_Systems_Research (Hentet: 14.01.19).
Hickson, I. (2010) *Web SQL Database*. Tilgjengelig fra: https://www.w3.org/TR/webdatabase/ (Hentet: 02.05.2019).
Holt, B. (2017) *Offline First: What's in a name?* Tilgjengelig fra: https://medium.com/codait/offline-first-whats-in-a-name-89c410910694 (Hentet: 17.05.2019).
Hoodie (2017a) hoodie.account. Tilgjengelig fra: http://docs.hood.ie/en/latest/api/client/hoodie.account.html (Hentet: 10.05.2019).
Hoodie (2017b) Requirements. Tilgjengelig fra: http://docs.hood.ie/en/latest/about/requirements.html (Hentet: 02.05.2019).
Hoodie (2018) *Welcome to Hoodie!* Tilgjengelig fra: http://hood.ie/intro/ (Hentet: 30.04.2019).
Horgen, S. A. (2014) *Webprogrammering i PHP*. 4. utgave. utg. Gyldendal akademisk.
HubSpot (2017) *Offline*. Tilgjengelig fra: https://github.com/HubSpot/offline (Hentet: 25.04.2019).
Hume, D. (2016) ServiceWorker: A Basic Guide to BackgroundSync, *Ponyfoo.com*. Tilgjengelig fra: https://ponyfoo.com/articles/backgroundsync (Hentet: 01.04.2019).

IBM (2019) *Cloudant*. Tilgjengelig fra: https://cloud.ibm.com/catalog/services/cloudant (Hentet: 29.04.2019).
Isaiah, A. (2019) *Add push notifications to your web app with Node.js and service workers*. Tilgjengelig fra: https://pusher.com/tutorials/push-notifications-node-service-workers (Hentet: 13.05.2019).
J Justin og Jude, J. (2017) Go Offline. *Learn Ionic 2.* Apress, Berkeley, CA. Tilgjengelig fra: https://doi.org/10.1007/978-1-4842-2617-9_8 (Hentet: 26.04.2019).
Jog, Y. *et al.* (2017) Understanding Mobile Apps and Related Permissions for Android Platform, *International Journal of Advanced Science and Technology*, Vol.105, s. 37-55. Tilgjengelig fra: http://article.nadiapub.com/IJAST/vol105/4.pdf (Hentet: 26.04.2019).
Johannessen, A., Tufte, P. A. og Christoffersen, L. (2010) *Introduksjonsjon til samfunnsvitenskapelig metode*.
Josh Karlin og Kruisselbrink, M. (2018) *Web Background Synchronization*. Tilgjengelig fra: https://wicg.github.io/BackgroundSync/spec/#service-worker-registration-extensions (Hentet: 02.05.2019).
Kallin, J. og Valbuena, I. L. (2013) *Excess XSS*. Tilgjengelig fra: https://excess-xss.com/ (Hentet: 03.05.2019).
Kayce Basques, M. K. (2019) Access Local Servers. doi: https://developers.google.com/web/tools/chrome-devtools/remote-debugging/local-server.
Kesavan, M. (2017) The Approval Process: Android vs. iPhone. Tilgjengelig fra: https://dzone.com/articles/the-approval-process-android-vs-iphone (Hentet: 30.04.2019).
Kharchenko, N. (2018) *8 JavaScript Alternatives for Web Developers to Consider*. Tilgjengelig fra: https://codeburst.io/8-javascript-alternatives-for-web-developers-to-consider-22f8d38bdfa9 (Hentet: 10.05.2019).
Kitamura, E. (2014) *Working with quota on mobile browsers*. Tilgjengelig fra: https://www.html5rocks.com/en/tutorials/offline/quota-research/ (Hentet: 26.02.2018).
Kruisselbrink, M. og Ranganathan, A. (2019) *File API*. Tilgjengelig fra: https://w3c.github.io/FileAPI/ (Hentet: 29.04.2019).
LePage, P. (2019) Add to Home Screen. Tilgjengelig fra: https://developers.google.com/web/fundamentals/app-install-banners/ (Hentet: 12.05.2019).
Long, J. (2012) *I Don't Speak Your Language: Frontend vs. Backend*. Tilgjengelig fra: https://blog.teamtreehouse.com/i-dont-speak-your-language-frontend-vs-backend (Hentet: 01.05.2019).
MacPherson, M. R. (2015) Supported Browsers Platforms. Tilgjengelig fra: https://github.com/localForage/localForage/wiki/Supported-Browsers-Platforms (Hentet: 02.05.2019).
Matthew MacPherson, Robert Nyman og Fabbro, A. (2014) *localForage: Offline Storage, Improved*. Tilgjengelig fra: https://hacks.mozilla.org/2014/02/localforage-offline-storage-improved/ (Hentet: 29.04.2019).
McLeod, B. (2018) 75+ MOBILE MARKETING STATISTICS FOR 2019 AND BEYOND. Tilgjengelig fra: https://www.bluecorona.com/blog/mobile-marketing-statistics (Hentet: 03.05.2019).
Mehta, N. (2011) *Programmable HTTP Caching and Serving*. Tilgjengelig fra: https://www.w3.org/TR/DataCache/ (Hentet: 08.04.2019).
Merriam-Webster (2019a) *app*. Tilgjengelig fra: https://www.merriam-webster.com/dictionary/app (17.05.2019).
Merriam-Webster (2019b) *application*. Tilgjengelig fra: https://www.merriam-webster.com/dictionary/application (Hentet: 17.05.2019).

Metcalf, C. (2016) *Security in Offline First Apps*. Tilgjengelig fra: https://medium.com/offline-camp/offline-first-security-59bf4800e82a (Hentet: 13.05.2019).
mongoDB (U.Å) MongoDB Atlas. Tilgjengelig fra: https://www.mongodb.com/cloud/atlas (Hentet: 14.05.2019).
mozilla Contributors (2019a) *Using Service Workers*. Tilgjengelig fra: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers (Hentet: 08.05.2019).
Mozilla Contributors (2019b) *Window.localStorage*. Tilgjengelig fra: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage (Hentet: 22.02.2019).
Mozilla Contributors (2019c) *FileSystem*. Tilgjengelig fra: https://developer.mozilla.org/en-US/docs/Web/API/FileSystem (Hentet: 25.04.2019).
Mozilla Contributors (2019d) *Browser storage limits and eviction criteria*. Tilgjengelig fra: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Browser_storage_limits_and_eviction_criteria (Hentet: 29/04/2019).
Mozilla Contributors (2019e) *HTTP cookies*. Tilgjengelig fra: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies (10.04.2019).
Mozilla Contributors (2019f) Cache. Tilgjengelig fra: https://developer.mozilla.org/en-US/docs/Web/API/Cache (Hentet: 24.04.2019).
Mozilla Contributors (2019g) *What is JavaScript?* Tilgjengelig fra: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript (Hentet: 20.04.2019).
Mozilla Contributors (2019h) *IDBRequest*. Tilgjengelig fra: https://developer.mozilla.org/en-US/docs/Web/API/IDBRequest (Hentet: 30.04.2019).
Mozilla Contributors (2019i) *Window.sessionStorage*. Tilgjengelig fra: https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage (Hentet: 22.02.2019).
Mozilla Contributors (2019j) *Basic concepts*. Tilgjengelig fra: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Basic_Concepts_Behind_IndexedDB (Hentet: 10.05.2019).
Mozilla Contributors (2019k) *Server-sent events*. Tilgjengelig fra: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events (Hentet: 03.05.2019).
Naylor, I. (2019) Are Progressive Web Apps the Future of Mobile? Tilgjengelig fra: https://www.business2community.com/mobile-apps/are-progressive-web-apps-the-future-of-mobile-02193543 (Hentet: 26.04.2019).
Nielsen, J. (2011) *How Long Do Users Stay on Web Pages?* Tilgjengelig fra: https://www.nngroup.com/articles/how-long-do-users-stay-on-web-pages/ (Hentet: 26.04.2019).
Now, T. (2019) Google to introduce stricter approval checks on Android apps, *Irish Examiner*. Tilgjengelig fra: https://www.irishexaminer.com/breakingnews/technow/google-to-introduce-stricter-approval-checks-on-android-apps-918422.html (Hentet: 30.04.2019).
Nunkesser, R. (2018) Beyond Web/Native/Hybrid: A New Taxonomy for Mobile App Development, i *2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, *27 May-3 June 2018*. s. 214-218.
Olsen, S. J. *et al.* (2017) *Den store dekningstesten*. Tilgjengelig fra: https://www.tek.no/artikler/test-den-store-dekningstesten-2017/382426/alle (Hentet: 02.05.2019).
Osmani, A. og Cohen, M. (2019) *Offline Storage for Progressive Web Apps*. Tilgjengelig fra: https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/offline-for-pwa (Hentet: 12.02.2019).

OWASP (2017) *OWASP Top 10 - 2017*. Tilgjengelig fra: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf (Hentet: 02.05.2019).
OWASP (2018) *Cross-site Scripting (XSS)*. Tilgjengelig fra: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS) (Hentet: 03.05.2019).
Oxford University Press (U.Å.) *Definition of scope in English*. Tilgjengelig fra: https://en.oxforddictionaries.com/definition/scope (Hentet: 17.05.2019).
Peyser, J. (2017) Intro to Hoodie and React, *CSS-Tricks*. Tilgjengelig fra: https://css-tricks.com/intro-hoodie-react/ (Hentet: 10.05.2019).
Plikk, N. (2013) Farvel, MiniDisc, *Tek.no*. Tilgjengelig fra: https://www.tek.no/artikler/farvel-minidisc/116519 (Hentet: 18.05.2019).
Poetker, B. (2019) Web Apps vs Mobile Apps: An Explanation for Non-Developers (+Real Examples). Tilgjengelig fra: https://learn.g2crowd.com/web-apps-vs-mobile-apps (Hentet: 29.04.2019).
Potter, J. (2019) *db.js vs dexie vs localforage*. Tilgjengelig fra: https://www.npmtrends.com/db.js-vs-dexie-vs-localforage (Hentet: 02.05.2019).
Pouchdb *About PouchDB*. Tilgjengelig fra: https://pouchdb.com/learn.html (Hentet: 23.04.2019).
Pouchdb *Adapters*. Tilgjengelig fra: https://pouchdb.com/adapters.html (Hentet: 23.04.2019).
Pouchdb *Introduction to PouchDB*. Tilgjengelig fra: https://pouchdb.com/guides/ (Hentet: 23.04.2019).
Pouchdb (2018) *Replication*. Tilgjengelig fra: https://pouchdb.com/guides/replication.html (Hentet: 03.05.2019).
Pouchdb (U.Å.-a) *About PouchDB*. Tilgjengelig fra: https://pouchdb.com/learn.html (Hentet: 23.04.2019).
Pouchdb (U.Å.-b) *Introduction to PouchDB*. Tilgjengelig fra: https://pouchdb.com/guides/ (Hentet: 23.04.2019).
Prince, A. (2018) *Build a simple app using Node JS and MySQL.* Tilgjengelig fra: https://dev.to/achowba/build-a-simple-app-using-node-js-and-mysql-19me (Hentet: 13.05.2019).
Quality, G. S. (2019) Search Engine Optimization (SEO) Starter Guide. Tilgjengelig fra: https://support.google.com/webmasters/answer/7451184?dark=1 (Hentet: 01.05.2019).
Ranganathan, A. (2010) *Beyond HTML5: Database APIs and the Road to IndexedDB*. Tilgjengelig fra: https://hacks.mozilla.org/2010/06/beyond-html5-database-apis-and-the-road-to-indexeddb/ (Hentet: 25.04.2019).
Rooney, N. *et al.* (2019) *World Wide Web Consortium Process Document*. Tilgjengelig fra: https://www.w3.org/2019/Process-20190301/#working-draft (Hentet: 17.05.2019).
Salnikov, M. (2018) *Background Fetch API: Get Ready To Use It!* Tilgjengelig fra: https://medium.com/google-developer-experts/background-fetch-api-get-ready-to-use-it-69cca522cd8f (Hentet: 30.04.2019).
Scales, M. (2019) *Using the Cache API*. Tilgjengelig fra: https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/cache-api (Hentet: 04.04.2019).
Serrano, N., Hernantes, J. og Gallardo, G. (2013) Mobile Web Apps, *IEEE Software*, 30(5), s. 22-27. doi: 10.1109/MS.2013.111.
Sheppard, D. (2017) *Beginning Progressive Web App Development*. Apress, Berkeley, CA. doi: https://doi.org/10.1007/978-1-4842-3090-9_1.

smArtapps (2018) *Native application VS Progressive Web App: which one should you choose?* Tilgjengelig fra: https://medium.com/inside-smartapps/native-application-vs-progressive-web-app-which-one-should-you-choose-5eeaaf6ee92d (Hentet: 01.05.2019).
Søk&Skriv (2018) *Kildevurdering*. Tilgjengelig fra: https://sokogskriv.no/kildebruk-og-referanser/kildevurdering/ (Hentet: 07.02.2019).
SQLite *Most Widely Deployed and Used Database Engine*. Tilgjengelig fra: https://www.sqlite.org/mostdeployed.html (Hentet: 25.04.2019).
Stack Overflow (2019) *Developer Survey Results*. Tilgjengelig fra: https://insights.stackoverflow.com/survey/2019#technology-_-programming-scripting-and-markup-languages (Hentet: 25.04.2019).
Statcounter (2016) *Mobile and tablet internet usage exceeds desktop for first time worldwide*. Tilgjengelig fra: http://gs.statcounter.com/press/mobile-and-tablet-internet-usage-exceeds-desktop-for-first-time-worldwide (Hentet: 02.05.2019).
Statcounter (2019) *Mobile Operating System Market Share Worldwide*. Tilgjengelig fra: http://gs.statcounter.com/os-market-share/mobile/worldwide (Hentet: 02.05.2019).
Store norske leksikon (2018a) *Operativsystem*. Tilgjengelig fra: https://snl.no/operativsystem (Hentet: 17.05.2019).
Store norske leksikon (2018b) *API*. Tilgjengelig fra: https://snl.no/API (Hentet: 17.05.2019).
Strutt, D. (2017) Progressive Web Apps (PWA) Vs True Native apps for Small Business. Tilgjengelig fra: https://medium.com/@danstrutt/progressive-web-apps-pwa-vs-true-native-apps-for-small-business-30fd3a5a7ef6 (Hentet: 13.05.2019).
Symantec Corporation (U.Å.) *Encryption: What it is and why it's important*. Tilgjengelig fra: https://us.norton.com/internetsecurity-privacy-what-is-encryption.html (Hentet: 13.05.2019).
Techopedia (U.Å.-a) *Minification*. Tilgjengelig fra: https://www.techopedia.com/definition/33786/minification (Hentet: 17.05.2019).
Techopedia (U.Å.-b) *Document-Oriented Database*. Tilgjengelig fra: https://www.techopedia.com/definition/30329/document-oriented-database (Hentet: 17.05.2019).
Techopedia (U.Å.-c) *Sandbox*. Tilgjengelig fra: https://www.techopedia.com/definition/27681/sandbox-software-development (Hentet: 17.05.2019).
Techopedia (U.Å.-d) *Wrapper*. Tilgjengelig fra: https://www.techopedia.com/definition/4389/wrapper-software-engineering (Hentet: 17.05.2019).
Techopedia (U.Å.-e) *Proof of Concept (POC)*. Tilgjengelig fra: https://www.techopedia.com/definition/4066/proof-of-concept-poc (Hentet: 17.05.2019).
Techopedia (U.Å.-f) *NoSQL*. Tilgjengelig fra: https://www.techopedia.com/definition/27689/nosql-database (Hentet: 17.05.2019).
Tutorialspoint (2019) *Node.js - Introduction*. Tilgjengelig fra: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm (Hentet: 10.04.2019).
Viktoria K. og V., V. (U.Å) 6 Categories of Web Application Maintenance: From Bugs to Scaling. Tilgjengelig fra: https://rubygarage.org/blog/web-application-maintenance (Hentet: 26.04.2019).
W3C (2019a) Data Storage. Tilgjengelig fra: https://www.w3.org/2019/04/web-roadmaps/mobile/storage.html (Hentet: 03.05.2019).
W3C (2019b) Application Lifecycle. Tilgjengelig fra: https://www.w3.org/2019/04/web-roadmaps/mobile/lifecycle.html (Hentet: 03.05.2019).

W3schools.com *HTML5 Server-Sent Events*. Tilgjengelig fra: https://www.w3schools.com/html/html5_serversentevents.asp (Hentet: 03.05.2019).
W3schools.com (U. Å.) *HTML 5 Web Storage*. Tilgjengelig fra: https://www.w3schools.com/html/html5_webstorage.asp (Hentet: 14.02.2019).
W3schools.com (U.Å) *Bootstrap 4 Get Started*. Tilgjengelig fra: https://www.w3schools.com/bootstrap4/bootstrap_get_started.asp (Hentet: 02.05.2019).
W3schools.com (U.Å.-a) *PHP Prepared Statements*. Tilgjengelig fra: https://www.w3schools.com/php/php_mysql_prepared_statements.asp (Hentet: 12.05.2019).
W3schools.com (U.Å.-b) *Node.js NPM*. Tilgjengelig fra: https://www.w3schools.com/nodejs/nodejs_npm.asp (Hentet: 12.05.2019).
W3Techs.com (2019) *Usage of JavaScript for websites*. Tilgjengelig fra: https://w3techs.com/technologies/details/cp-javascript/all/all (Hentet: 10.04.2019).
Wagenseil, P. (2014) *Hacker's Prank Makes Websites Do Harlem Shake*. Tilgjengelig fra: https://www.tomsguide.com/us/harlem-shake-hack,news-19595.html (Hentet: 03.05.2019).
West, M. (2013) *Create Your Own To-Do App with HTML5 and IndexedDB*. Tilgjengelig fra: https://blog.teamtreehouse.com/create-your-own-to-do-app-with-html5-and-indexeddb (Hentet: 22.02.2019).
West, M. (2016) *Content Security Policy Level 3*. Tilgjengelig fra: https://www.w3.org/TR/CSP3/ (Hentet: 12.03.2019).
WHATWG (2019a) *Web Storage*. Tilgjengelig fra: https://html.spec.whatwg.org/multipage/webstorage.html (Hentet: 13.05.2019).
WHATWG (2019b) *Server-sent events*. Tilgjengelig fra: https://html.spec.whatwg.org/multipage/server-sent-events.html#eventsource-push (Hentet: 03.05.2019).
WICG (2015) *Web Incubator Community Group Charter*. Tilgjengelig fra: http://wicg.github.io/admin/charter.html (Hentet: 29.04.2019).
WICG (2019) *Background-fetch*. Tilgjengelig fra: https://github.com/WICG/background-fetch (Hentet: 30.04.2019).
Wilson, D. og Sidorov, A. (2019) *A pure node.js JavaScript Client implementing the MySQL protocol*. Tilgjengelig fra: https://github.com/mysqljs/mysql#todo (Hentet: 12.05.2019).
World Wide Web Consortium (2012) *Standards FAQ*. Tilgjengelig fra: https://www.w3.org/standards/faq (Hentet: 15.02.2019).
Wozniewicz, B. (2019) *The Difference Between a Framework and a Library*. Tilgjengelig fra: https://medium.freecodecamp.org/the-difference-between-a-framework-and-a-library-bd133054023f (Hentet: 01.05.2019).

9 VEDLEGG 1 WEB STORAGE OVERVIEW

API	Data Model	Persistence	Browser Support	Transactions	Sync/Async
File system	Byte stream	device	52%	No	Async
Local Storage	key/value	device	93%	No	Sync
Session Storage	key/value	session	93%	No	Sync
Cookies	structured	device	100%	No	Sync
WebSQL	structured	device	77%	Yes	Async
Cache	key/value	device	60%	No	Async
IndexedDB	hybrid	device	83%	Yes	Async
cloud storage	byte stream	global	100%	No	Both

Tabelloversikt av ulike lagringsteknologier fra Google Developers (Cohen, 2019).

10 VEDLEGG 2 OPPGAVETEKST NUMMER 66.

Arbeidstittel: Er mobilappenes tid forbi?

Hensikten med oppgaven:

Kundebaserte erfaringer hos IT Data AS forteller om utfordringer ved dårlig mobildekning og løsninger basert på ren webteknologi. Bedriften mener også at godkjenningsprosessen hos ulike distribusjonsplattformer av mobilapper er tidkrevende. Det er også komplikasjoner vedrørende bruk av flere ulike smarttelefon-operativsystemer hos kundene. Ved bruk av webapplikasjoner kan man unngå utfordringene knyttet til distribusjonsplattformene og operativsystemene. Finnes det webteknologier som gjør det mulig at webapplikasjoner som kan kjøres offline kan konkurrere med mobilapper?

Kort beskrivelse av oppgaveforslag:

Hvilke teknologier finnes i dag eller er under utvikling for å lage offline webapplikasjoner? Hvordan kan disse teknologiene implementeres inn i dagens webløsninger? Hvilken funksjonalitet tilbyr disse? Hvordan ser framtidsutsiktene for webapplikasjonene ut?

Oppgaven passer for (kryss av de(t) som passer og skriv evt. en kommentar til oss):

- Prosjektoppgave
- Bacheloroppgave

Kan oppgavestiller stille arbeidsplass med nødvendig utstyr og programvare:

Ja

Begrensninger i tilgjengelighet av opplysninger o.l.:

Ingen

Oppgaven passer best for, antall studenter:

- 1
- 2

Opplysninger om oppgavestiller

Navn på bedrift eller organisasjon:

IT Data AS

Adresse

Fabrikkvegen 13

Postnummer

6415

Poststed

MOLDE

Navn på kontaktperson/veileder:

John Erik Johnsen

Utfyllende kommentarer til hva oppgaven gjelder

Oppgaven gjelder å kartlegge og fordype seg i eventuelle teknologier som gjøre det mulig å lage webapplikasjoner som kjører offline. Aktuelle teknologier har vært under utvikling i mange år, men er nå muligens i ferd med å bli såpass modne at man kan lage mer avanserte webapplikasjoner som kan brukes uten nettilkobling. Disse webapplikasjonene vil være spesielt nyttige i områder med uten mobildekning.