Høgskulen
på Vestlandet

BACHELOROPPGAVE

Publiseringsløsning

Publishing solution

Christian Grainger Føleide
Kristoffer Davidsen

Bachelor i Dataingeniør/Informasjonsteknologi
Fakultet for ingeniør- og naturvitskap
Institutt for datateknologi, elektroteknologi og realfag
Veileder: Bjarte Kileng
4. juni 2021

TITTELSIDE FOR HOVEDPROSJEKT

Rapportens tittel: Publiseringsløsning Publishing solution	*Dato:* 04.06.2021
Forfatter(e): Christian Grainger Føleide, Kristoffer Davidsen	*Antall sider u/vedlegg: 33* *Antall sider vedlegg: 4*
Studieretning: Dataingeniør/Informasjonsteknologi	*Antall disketter/CD-er:* 0
Kontaktperson ved studieretning: Bjarte Kileng	*Gradering:* Ingen
Merknader:	

Oppdragsgiver: Styreportalen AS	*Oppdragsgivers referanse:*
Oppdragsgivers kontaktperson: Stian Sømoen	*Telefon:* *48438080*

Sammendrag:

Målet med dette prosjektet er å utvikle en prototype for en publiseringsløsning, som lar Styreportalens kunder sette opp og redigere deres egen nettside. Prototypen er en webapplikasjon, med fokus på brukervennlighet.

This project's goal was to develop a prototype for a publishing solution, which lets customers of Styreportalen set up and edit their own web page. The prototype is a web application, focusing on ease of use.

Stikkord:

Webapplikasjon	Design	SSG
Content Management System	Redigering	SSR
Brukergrensesnitt	Statisk HTML	

FORORD

Denne rapporten dokumenterer arbeidet som er utført på prosjektet "Publiseringsløsning" for Styreportalen AS. Arbeidet ble utført ved Høgskulen på Vestlandet, Campus Bergen, våren 2021. Prosjektet ble utført av Christian Grainger Føleide og Kristoffer Davidsen.

Vi ønsker å takke prosjekteier og ansatte i Styreportalen for god dialog, veiledning og innspill gjennom hele prosjektforløpet. Vi ønsker også å takke Bjarte Kileng for god veiledning og støtte.

TABLE OF CONTENTS

1 INNLEDNING

Styreportalen AS er en bedrift som leverer digitale tjenester til lag og foreninger. Tjenestene består av administrering av organisasjonen i form av medlemsregistre, økonomistyring, kalendere for redigering av arrangementer, kommunikasjonsløsninger for medlemmer internt i organisasjonen, og verktøy for daglig drift. Styreportalen inngikk i 2020 en avtale med Norges Korpsforbund, og forventer en økt kundebase som følge av dette.

1.1 Motivasjon og mål

Prosjekteieren, Styreportalen AS, har i dag flere løsninger for å opprette og vedlikeholde sine kunders nettsider. Dagens nettsider bærer preg av eldre teknologi og utdatert design som er lite brukervennlig, og bedriften bruker mer tid enn ønskelig på å vedlikeholde applikasjonen. Bedriften ønsket å øke sitt konkurransefortrinn overfor andre bedrifter ved å lage en mer brukervennlig applikasjon for å sette opp og redigere kundenes nettsider, med håp om at den også vil gjøre det lettere å vedlikeholde disse nettsidene. Målet for prosjektet ble derfor å utvikle en prototype for en publiseringsapplikasjon, med fokus på at prototypen skal være brukervennlig og gjøre det lettere for bedriften å vedlikeholde kundenes nettsider. Prosjektets hovedfokus var å utvikle et intuitivt og brukervennlig brukergrensesnitt som var lett å forstå, samt å lage statiske komponenter til bruk på websidene.

1.2 Kontekst

Bedriften driver i dag et sted mellom 50 og 100 nettsteder for sine kunder. Bedriften venter en økt kundebase i løpet av 2021, og forventer at antall nettsteder i drift kan stige til 500.
Hvert nettsted redigeres i dag ved bruk av verktøyet Joomla (Joomla), som er et *Content Management System* (CMS) for å styre og redigere innhold. I tillegg har hvert nettsted sitt eget domene og database. Joomla har et stort utvalg av muligheter for design, men det er også mange begrensninger i verktøyet, og kan være komplekst for kunden å bruke i praksis.

For å oppnå bedriftens ønske, må man først definere hva det innebærer at applikasjonen skal være brukervennlig. Et av de viktigste momentene innen brukervennlighet er å definere kundesegmentene for applikasjonen (Pawel, 2019), og gjøre de tilpasninger som behøves deretter. Et kundesegment er en del av et marked, som kan deles inn i flere segmenter basert på kjønn, alder eller andre egenskaper (Toft Sundbye, 2018). Er brukerne unge voksne med teknologiske ferdigheter, eller består brukerbasen mer av den eldre garde som har lite erfaring med datamaskiner? Dette er spørsmål som er viktig for gruppen å besvare før prosjektarbeidet starter. Andre momenter som er med på å definere brukervennlighet kan være at applikasjonen er stabil og pålitelig, at den er forståelig og lett å komme i gang med m.m.

Definisjonen av hva som gjør en applikasjon lettere å vedlikeholde kan trekkes ut i flere retninger. Det første man kan gjøre er å se på dagens løsning og finne årsaker til hvorfor bedriften synes man bør finne forbedringer. Det kan handle om en gammel kodebase som er utdatert, ha sikkerhetshull eller andre feil som kan gjøre det risikabelt å videreføre bruken.
Hvor lett den er å vedlikeholde avhenger også av at utviklere er kjent med teknologien og/eller programmeringsspråk. En nyere løsning vil være lettere å vedlikeholde dersom *kodebasen* er skrevet i et programmeringsspråk som flere kan. Dette vil og gjøre det lettere for nyansatte i bedriften å jobbe med løsningen, noe også bedriften tjener på.

1.3 Avgrensninger

Begrenset tid ble den største avgrensningen for prosjektarbeidet. På grunn av dette bestemte gruppen at hovedfokuset for arbeidet var å utvikle frontend/brukergrensesnittet i applikasjonen. Dersom gruppen kom i mål med dette arbeidet før tiden var ute, hadde vi også muligheter for å jobbe videre med *backend*-delen av prosjektet.

En annen faktor som måtte tas hensyn til, var at gruppen hadde begrenset kunnskap om teknologien som skulle brukes i applikasjonen. Applikasjonen bygger på Next.js og React, som gruppen hadde lite kjennskap til før prosjektarbeidet startet. Det ble derfor satt av tid til å bli bedre kjent med, og lære om disse teknologiene.

1.4 Ressurser

Den viktigste ressursen for gruppens arbeid med prosjektet var bedriften og dens ansatte.
De ansatte hadde god erfaring med teknologier som ble benyttet, og bidro med innspill og veiledning underveis i prosjektet. Bedriftens ansatte hadde i tillegg erfaring med lignende prosjektarbeid som hadde vært utført på et tidligere tidspunkt.

Internett var en annen viktig ressurs ved prosjektarbeidet. Gruppen fant mye nødvendig dokumentasjon på diverse nettsteder som var behov for underveis, samt eksempler og inspirasjon til bruk når gruppen skrev sin egen implementasjon.

1.5 Oppbygging av rapporten

- Kapittel 1:
    - Presentere prosjektet, samt motivasjonen og målet bak prosjektet. Kapittelet tar også for seg bakgrunnen for at prosjektet ble til. Til slutt drøftes det kort rundt begrensninger knyttet til prosjektet, samt hvilke ressurser som ble benyttet i forbindelse med arbeidet.
- Kapittel 2:

    - Beskrivelse av prosjektet og hvem som er eier av prosjektet. Videre presenteres initielle krav som prosjektet må møte, samt bedriftens initielle forslag til løsning.
- Kapittel 3:
    - Presentasjon av prosjektdesign og mulige fremgangsmåter for prosjektet. Det presenteres mulige løsninger til prosjektet, og det drøftes hvorfor gruppen valgte en spesifikk løsning fremfor andre som ble presentert.
- Kapittel 4:
    - Detaljert løsning av prosjektet og grundige beskrivelser av arkitekturen i alle lag av applikasjonen.
- Kapittel 5:
    - Evalueringsmetoder og -resultat.
- Kapittel 6:
    - Resultat av prosjektarbeidet.
- Kapittel 7:
    - Diskusjon rundt valgene som ble gjort både initielt og underveis, hvilke konsekvenser de fikk og evt. alternative løsninger.
- Kapittel 8:
    - Konklusjon.
- Kapittel 9:
    - Referanser.
- Kapittel 10:
    - Appendix.

2 PROSJEKTBESKRIVELSE

2.1 Praktisk bakgrunn

2.1.1 Prosjekteier

Eier av prosjektet er Styreportalen AS ved daglig leder Stian Sømoen. Styreportalen tilbyr ulike løsninger for styret i lag og organisasjoner som medlemsregistre, prosjektstyring, digitalt notearkiv m.m. Arbeidet med løsningene som Styreportalen tilbyr startet i 2014, da som en del av et annet selskap. I 2018 ble løsningene skilt ut i eget aksjeselskap. I mars 2020 signerte Styreportalen en samarbeidsavtale med Norges Musikkorps Forbund (NMF) for å bygge bedre løsninger for alle forbundets medlemmer (Mogstad, 2020). Formålet med denne avtalen er å få et tettere samarbeid mellom partene, slik at NMF har muligheter til å komme med innspill om hvilken funksjonalitet som bør prioriteres. Styreportalen får med denne avtalen muligheter til å utvide kundebasen sin betraktelig, ettersom NMF har over 1600 medlemsorganisasjoner.

2.1.2 Tidligere arbeid

Arbeidet med prosjektet har i noen grad basert seg på tidligere arbeid fra bedriften, i den forstand at man har videreført noen konsepter fra tidligere løsninger. Bedriften har hatt lignende oppgaver i forbindelse med tidligere bacheloroppgaver, og har fått mange erfaringer derfra. Gruppen har også utforsket andre lignende løsninger i markedet for å finne ut hvilke løsninger som har fungert bra, og hvilke løsninger som ikke fungerte.

2.1.3 Initielle krav

Bedriftens krav til prosjektet dreide seg i stor grad om hvilke funksjoner og komponenter som måtte implementeres for at resultatet ville være ansett som tilfredsstillende:

- Kalender
    - Bruker må kunne sette opp en kalender i applikasjonen som henter og viser data om aktuelle hendelser. Modifisering av dataene som vises blir håndtert av en tredjepartsapplikasjon.
- Nyheter
    - Bruker må ha mulighet til å vise nyheter om foreningen i applikasjonen. Oppretting og redigering av nyheter gjøres i en tredjepartsapplikasjon.
- Andre komponenter:
    - Kontaktskjema
    - Informasjon om forening
- Design, farger, font
    - Bruker må kunne bestemme farger, skriftstørrelse, font ol. i applikasjonen.
- Lagring av endringer

    - Bruker må kunne lagre endringer som er gjort, som deretter blir lagret på en server.
- Form og funksjon
    - Applikasjonen skal være brukervennlig og lett å forstå.
    - Applikasjonens funksjoner og moduler skal ha et profesjonelt design.

Bedriften har også satt krav om at websidene skal kunne genereres statisk så langt det lar seg gjøre, det betyr at man må generere HTML, CSS og JavaScript på forhånd. Dersom dette i noen tilfeller ikke lar seg gjøre, skal applikasjonen generere websider på tradisjonelt vis, når hver forespørsel kommer inn til tjener.
I tillegg til det nevnte skal applikasjonen være responsiv; innholdet i applikasjonen må skalere fint for både mobil, nettbrett og PC.

2.1.4 Initiell løsnings-idé

Bedriftens initielle idé til løsning var å benytte verktøyet *TinaCMS* (TinaCMS, 2019-2021) sammen med rammeverket *Next.js* i applikasjonen. *TinaCMS* leverer fundamentet for å lage et *content management system*. Deres argumentasjon for å benytte dette verktøyet var at gruppen hadde mulighet til å bruke mer tid på å implementere redigering av innholdet, fremfor å bygge nevnte fundament fra bunnen. *Next.js* har muligheten til både å statisk generere websider på forhånd eller å generere websider når forespørsel skjer.
Bedriften har også satt opp en demonstrasjon av hvordan de ser for seg at nettsiden til kunden skal se ut når prosjektet er ferdig. Gruppen kunne bruke denne demoen til inspirasjon, men har hatt frihet til å velge bort eller legge til funksjoner.

3 DESIGN AV PROSJEKTET

3.1 Forslag til løsning

Når forslag til løsningen ble drøftet, delte man løsningen i to deler. Den første delen handlet om hvordan gruppen planla å implementere *fundamentet* i applikasjonen. Fundamentet i dette tilfelle består av mange komponenter, som for eksempel menyer og vinduer, som kreves for å implementere resten av applikasjonen. Den andre delen handlet om hvordan man implementerte funksjonene som ble brukt til å sette inn og redigere innholdet på brukerens nettside. For å lettere forstå oppdelingen, kan vi ta applikasjonen Microsoft Word som eksempel. Fundamentet i Word er selve visualiseringen av applikasjonen; Knapper og menyer som viser funksjoner, arket som viser hvor man kan legge inn innhold ol. Den andre delen kan sammenlignes med funksjonene i Word som endrer tekststørrelse, farge, font ol.

3.1.1 Alternativ løsning 1

Det første alternativet til prosjektløsning, var å lage alle komponenter i applikasjonen på egen hånd. Det betyr at alle menyer og vinduer som benyttes for å redigere innholdet blir produsert av gruppen, fremfor å bruke et bibliotek som allerede eksisterer. I tillegg vil alle funksjoner som brukes til redigering og innsetting av innhold bli produsert selv. Fordelen med denne løsningen var at gruppen hadde mulighet til å bestemme egenskapene ved komponentene i større grad selv. Egenskaper i dette tilfelle kan være form og farge på meny ol. Den andre fordelen ved å ha produsert alt selv var at man slapp å tenke på lisenser og rettigheter ved bruk av andres materiale. Ulempen ved denne løsningen var at det hadde blitt svært tidkrevende å produsere alle komponenter på egen hånd.

3.1.2 Alternativ løsning 2

Det andre alternativet for hvordan prosjektet kunne løses, var å benytte eksisterende verktøy som tar seg av fundamentet for applikasjonen. I bedriftens idé for løsning foreslo de å benytte *TinaCMS* (Tina CMS). Et alternativt verktøy som leverer et lignende fundament er *KeystoneJS* (KeystoneJS).
Funksjoner for innsetting og redigering av innhold kunne også implementeres ved bruk av eksisterende komponenter som finnes på nettet.
Fordelen med denne løsningen var at gruppen hadde spart mye tid i ved at man kunne ta i bruk eksisterende komponenter fremfor å produsere selv. En ulempe ved denne løsningen var at gruppen hadde brukt mer tid på å gjennomgå rettigheter og lisenser ved bruk av eksisterende komponenter. En annen ulempe er at man måtte ha satt av en del tid til å studere hvilke komponenter som egner seg til prosjektets formål, og som har egenskapene det behøver.

3.1.3 Alternativ løsning 3

Det tredje alternativet hadde vært å kombinert de to første alternativene. Løsningen ville da vært basert på en blanding av eksisterende verktøy og egenproduserte komponenter og funksjoner. De nevnte fordeler og ulemper ved alternativene ble jevnet ut, alt etter hvilket alternativ man ønsket å benytte seg av i størst grad. Ulempen med denne løsningen var det ble vanskelig å finne balansepunktet mellom alternativene, ettersom gruppen måtte drøfte dette for hver komponent som skal brukes i applikasjonen.

3.1.4 Diskusjon av alternativene

Når gruppen tok avgjørelsen om hvilket alternativ som var best for prosjektet, ble avgjørelsen tatt basert på avgrensningene som er presentert i kapittel 1.3 og de initielle kravene i kapittel 2.1.3. Gruppen ble enige om at alternativ 1 var det mest tidkrevende alternativet, og som krevde mest kunnskap fra gruppen. Å lage fundamentet og alle komponenter fra bunnen av ville krevd intensivt arbeid fra gruppen gjennom hele perioden for å komme i mål, og det var heller ikke mulig å garantere at resultatet ville vært tilstrekkelig. I tillegg måtte gruppen ha skrevet alle komponentene selv istedenfor å gjenbruke eksisterende kode, og det ville gjort det vanskeligere for bedriften å vedlikeholde prosjektet. Fordelen ved å ha valgt dette alternativet var at gruppen hadde tilegnet seg mye kunnskap om hvordan alle komponentene i applikasjonen fungerte, som hadde vært gunstig når prosjektet ble overlevert til bedriften.
Alternativ 2 hadde vært mulig å gjennomføre med atskillig mindre arbeidsmengder enn ved alternativ 1. Gruppen hadde derimot ikke sittet igjen med med samme nivået av kunnskap som ved gjennomføring av alternativ 1, ettersom man hadde benyttet eksisterende verktøy og komponenter. Fordelen med denne løsningen var at man hadde fulgt prinsippet om gjenbruk av kode, slik at det potensielt hadde vært enklere for bedriften å forstå kodebasen ved overtagelse, ettersom det finnes mer dokumentasjon og eksempler for eksisterende komponenter.
Basert på avgrensningene ga alternativ 3 en god balanse mellom tid og begrenset kunnskap. Gruppen hadde fått mulighet til å velge hva som ble implementert på egen hånd, og hvilke eksisterende verktøy som ble benyttet. Alternativet krevde imidlertid mer planlegging rundt disse valgene, og kunne potensielt “stjålet” tid som hadde blitt brukt til andre formål.

3.2 Valgt løsning

Gruppen valgte alternativ 3 som løsning for prosjektet. Tidsavgrensningen gjorde det veldig vanskelig å gjennomføre alternativ 1 med et tilfredsstillende resultat. Alternativ 2 ga gruppen for lite kontroll over egenskapene ved komponentene, og for lite kunnskap om hvordan de fungerte. Alternativ 3 ble dermed det beste alternativet, til tross for nevnte ulemper.

Prosjektet ble gjennomført ved at den første delen, fundamentet, ble bygget basert på verktøyet TinaCMS (Tina CMS). Dette er et verktøy som bedriften anbefalte, og som sikret at vi kunne bruke minimalt med tid på denne delen. Videre benyttet gruppen en blanding av egenproduserte og

eksisterende komponenter i den andre delen, og det ble gjort en vurdering i hvert enkelt tilfelle for hva som var mest optimalt.

3.3 Valg av verktøy

3.3.1 TinaCMS

Prosjektet benyttet *TinaCMS* (Tina CMS), som er et verktøy som brukes for å lage et eget *content management system*.

3.3.2 React.js

React (Facebook Inc., 2021) er et deklarativt Javascript bibliotek. Dette betyr at det ikke er nødvendig å fortelle React *hvordan* innholdet skal vises, men kun *hva* som skal vises. Det er også komponentbasert, som betyr at man kan samle innhold, og logikk på innholdet i en gjenbrukbar modul. Strukturen disse komponentene får minner veldig om en vanlig DOM-struktur, hvor man nøster komponentene i hverandre, og kan dermed få flere *underkomponenter* til å endre innhold basert på en annen komponent lengre oppe i strukturen. Disse forholdene kalles gjerne *parent-child* forhold.

Mer om hvordan gruppen har brukt React beskrives i kapittel 4.3.2.

3.3.3 Next.js

Gruppen benyttet *Next.js* (Vercel, 2021), som er et produksjonsrammeverk for å bygge applikasjonen. Next.js er et Javascript rammeverk, eller mer spesifikt et *React* rammeverk.

3.3.4 Tailwind CSS

Gruppen benyttet Tailwind CSS for å skrive css. Tailwind er et rammeverk som gjør det enklere å skrive css direkte på React-komponenter, slik at man ikke behøver å lage eksterne css-filer. Tailwind gjør dette ved å tilby forhåndsdefinerte css-klasser.

3.3.5 Visual Studio Code

Gruppen brukte egne datamaskiner, og benyttet tekstprogrammet *Visual Studio Code* (Microsoft, 2021). Årsaken til at gruppen valgte å bruke dette programmet, er fordi det tilbyr mange *utvidelser*, små programmer, som gjorde det lettere for gruppen å skrive kode. Et eksempel er *Tailwind CSS*-plugin (Cornes, 2021), som ved bruk gjorde det lettere for gruppen å skrive css-kode ved å tilby generering av forslag og beskrivelse av hver css-klasse. I Visual Studio Code sin integrerte terminal benyttet vi npm og git, som forklares i de neste kapitlene.

3.3.6 npm

Gruppen benyttet *npm* (npm, Inc. 2021), som er et pakkehåndteringssystem. Dette ble benyttet for å administrere og laste ned eksterne verktøy og komponenter som ble brukt i applikasjonen. Pakkene som brukes må ha åpne, og helst gratis lisenser (Licenses | Open Source Initiative, 2021). Hvilke pakker som er brukt kan sees i appendix D.

3.3.7 Git

Git (git, 2021) ble brukt for deling og versjonskontrollering av kildekoden, som ble publisert på et privat oppbevaringssted på *GitHub* (GitHub, 2021).

3.3.8 Vercel

Vercel (Vercel) ble brukt som midlertidig vert for applikasjonen, og ble integrert mot Github slik at kodeendringer eller implementasjoner kan rulles ut og publiseres umiddelbart. Vercel gjør dette ved å lytte på endringer i det private oppbevaringsstedet, stoppe sin egen tjener, hente endringene, for så å bygge og publisere på nytt. Dette gjorde at Styreportalen fikk mulighet til å se fremgang og teste internt underveis.

3.3.9 Firebase

Firebase er en Backend-as-a-service (BaaS) som tilbyr ulike tjenester til brukere, blant annet databaser, kommunikasjon mellom database og applikasjon m.m.

3.4 Prosjektmetodikk

3.4.1 Utviklingsmetodikk

Prosjektets utviklingsmetodikk baserte seg på prototypemodellering. Denne metodikken er en iterativ, prøv-og-feil prosess som tillater at gruppen har kunnet utvikle, teste jevnlig og tilpasse endringer etterhvert (Lewis, 2019). Kravspesifikasjonen nevnte mye om hvilke funksjoner den ferdige applikasjonen behøvde, men lite om hvordan disse skulle implementeres, og gruppen hadde dermed mye spillerom til å bestemme dette selv. Prototypemetodikken passet bra for denne type prosjekt, da det ga gruppen mulighet til å teste forskjellige implementasjoner, samt gi bedriften mulighet til å komme med tilleggskrav underveis i prosjektet.

Prosjektet hadde en iterativ tilnærming i utviklingsprosessen. Gruppen hadde ukentlige møter med bedriften der arbeidet som ble gjort i inneværende iterasjon ble diskutert, og hva målsetningen for neste iterasjon var. Dersom inneværende iterasjon ikke ble fullført eller av andre grunner hadde mangler, diskuterte man også hvorfor dette skjedde, og eventuelle løsninger på problemet.

3.4.2 Prosjektplan

Prosjektplanen ble delt opp i en rekke aktiviteter, og visualiseres i et Gantt-diagram (Appendix C). Aktivitetene er som følger:

- Kartlegging av kundesegmenter:
    - For å kunne tilrettelegge implementasjonen på en brukervennlig måte var det viktig å kartlegge hvilke segmenter som fantes i kundegruppen. Kundegruppen besto av både gamle og unge, med stor variasjon innen datakunnskaper.
- Dokumentasjon/rammeverk:
    - Gruppen satt av mye tid til å studere dokumentasjon og rammeverk som skal benyttes i prosjektet. Mye var ikke nok, og gruppen endte opp med å bruke mer tid enn planlagt på denne aktiviteten. Årsaken til dette var at vi brukte mer tid til å forstå hvordan TinaCMS og Next.js fungerte i praksis.
- Kravspesifisering:
    - Gruppen utviklet i samråd med bedriften en initiell kravspesifisering som la grunnlaget for arbeidet med prosjektet.
- Skrive blogg:
    - Gruppen satt av tid til å skrive blogg ukentlig, der det ble redegjort for fullført arbeid i uken som gikk, og hvilket arbeid som var planlagt for neste uke.
- Skrive rapport:
    - For å få en god flyt i forholdet mellom arbeid med prosjektet og rapporten, satt gruppen av faste tidspunkter til skriving.
- Fase 1 - fundament:
    - Den første fasen handlet om å få på plass fundamentet for applikasjonen, slik at det ble mulig å starte på fase 2. Denne fasen ble gjennomført i tråd med planlagt tid.
- Fase 2 - funksjonelle krav:
    - I denne fasen ble det arbeidet med komponenter som skulle brukes i applikasjonen, og andre funksjonelle krav for at applikasjonen skulle fungere som optimalt. Gruppen brukte mer tid enn planlagt på denne aktiviteten, da det viste seg å være mer arbeid enn forutsatt.
- Fase 3 - lagring og backend:
    - I denne aktiviteten implementerte gruppen lagring for applikasjon på tjenersiden. Denne aktiviteten viste seg å være enklere enn antatt, og det ble brukt mindre tid enn prosjektert. Tiden til overs ble benyttet til fase 2.
- Testfase 1 og 2:
    - Denne aktiviteten ble endret på underveis, ettersom begge aktivitetene ble gjennomført på samme tidspunkt. Det ble lagt opp til ukentlig testing og evaluering sammen med bedriften, som også ble gjennomført. På slutten av prosjektarbeidet gjennomførte gruppen diverse ytelsestester ved bruk av programvare, samt kundetesting.
- Ferdiggjøring av rapport:

- Aktiviteten handlet om å skrive ferdig rapporten, rette opp i eventuelle feil etc., og ble gjennomført etter planen.

3.4.3 Risikovurdering

Risikoanalysen tar for seg:

- **Sannsynlighet (S)** (skala fra én til fem, der fem betyr høy sannsynlighet)
- **Konsekvens (K)** (skala fra én til fem, der fem betyr stor konsekvens)
- **Risikofaktor (RF)** (sannsynlighet multiplisert med konsekvens, hvor høy faktor er dårlig)

Den første risikoen for prosjektet var risikoen for at applikasjonen endte opp med dårlig design, se appendix A. Dårlig design kan bety at utseende ikke ser profesjonelt ut, at komponentene ikke er intuitive eller enkel å bruke etc. Gruppens medlemmer hadde lite tidligere erfaring med visuell design av komponenter, og sannsynligheten var til stede for at resultatet kunne blitt dårlig design. Gruppen ønsket å redusere denne risikoen ved å ha tett dialog med bedriftens ansatte, ettersom de hadde masse erfaring med denne delen fra tidligere arbeid.

Den andre risikoen for prosjektet var eventuelle tidsforsinkelser som kunne oppstå underveis, se appendix A. Eksempler på tidsforsinkelser som gruppen så for seg kunne oppstå var dårlige prioriteringer rundt arbeidsoppgavene, tekniske problemer, feilkommunikasjon mellom gruppen og bedriften e.l. For å minimere denne risikoen planla gruppen å sette opp ukentlige mål for arbeidet der arbeidsoppgavene ble nummerert etter viktighetsgrad. Videre var planen å diskutere disse målene med bedriften for å sikre at begge parter var enige. For å minimere eventuelle tidsforsinkelser rundt tekniske problemer, planla gruppen å lagre alle rapporter og kildekoder i diverse skytjenester, slik at ikke arbeid gikk tapt.

Problemstillingen rundt bruk av lisensiert programvare var noe gruppen også anså som en potensiell risiko for utvikling av applikasjonen, se appendix A. Dersom gruppen ønsket å ta i bruk eksisterende programvare var det en risiko for at lisensen ikke tillot kommersiell bruk uten at man måtte betale for lisens. Da måtte gruppen isteden utviklet en egen løsning som en erstatning for denne programvaren.

3.5 Evalueringsplan

For å evaluere løsningen, ble det testet både internt i Styreportalen tidlig i prosjektforløpet, og mot en av Styreportalens kunder mot slutten. Den interne testingen ble utført på den måten at funksjonalitet, design og hvor responsiv den er ble testet av prosjekteier og ansatte. Dette ble gjort på ukentlig basis, og ble diskutert på møtene. Mangler eller feil ble notert av gruppen, og ble jobbet med parallelt med andre krav frem til neste ukentlige møte.
Testing mot kunder ble gjennomført mot slutten av prosjektforløpet. Det ble da utført *end-to-end* testing for å teste applikasjonen i sin helhet.

Gruppen benyttet seg også av *Google Lighthouse* (Lighthouse | Google Developers) for å teste ytelse, tilgjengelighet, SEO, og elementer fra beste praksis.
På slutten av prosjektet ble det også gjort en sluttevaluering av produktet sammen med prosjekteier.

4 Detaljert design

Dette kapittelet beskriver i detalj hvordan applikasjonen og arkitekturen i hvert lag er bygget opp. Merk at i dette kapittelet blir det referert til *brukere* og *besøkende*. Brukere er de som representerer organisasjonen, og som bruker løsningen for å publisere og redigere innhold og design på organisasjonens nettside. Besøkende har ikke disse rettighetene, og vil kun se *resultatet* av det brukerne har gjort, i form av en tilsynelatende helt vanlig nettside.

4.1 Databasearkitektur

Applikasjonen benyttet Firebase som tjenersideløsning. Firebase er en samling av tjenester, som for eksempel lagring av filer, hosting av nettsteder m.m. Styreportalen benytter Firebase i sine eksisterende løsninger og det er derfor naturlig at applikasjonen ble integrert i deres database.

4.1.1 Lagring av brukernes innhold

Cloud Firestore er databasen som leveres gjennom *Firebase*. Databasen er av typen noSQL, det betyr at applikasjonen lagrer data i form av filer og dokumenter istedenfor i relasjonstabeller. Når brukeren gjør endringer i applikasjonen, vil den modifiserte dataen skrives i et JSON-dokument som deretter lagres i databasen. *Cloud Firestore* støtter de aller fleste typer filer, slik at brukeren også kan laste opp bilder, tekstfiler og andre filer som kan brukes i applikasjonen .

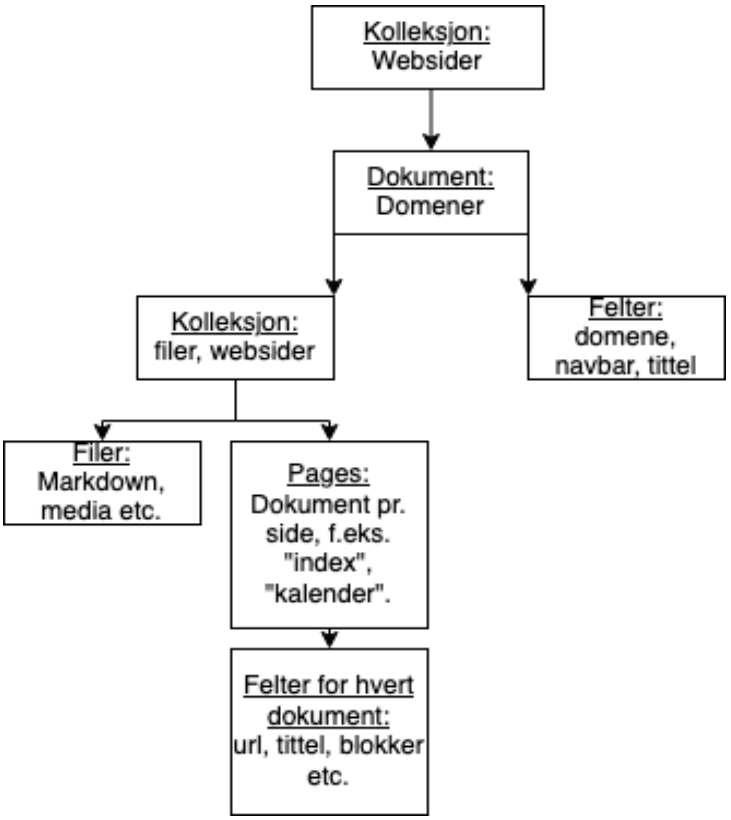


Fig 4.1: Databasestruktur for applikasjonen.

Databasen lagrer data i en “kolleksjon”, som er et hierarki av dokumenter. Det er ingen begrensning på hvor mange nivåer som er tillatt i hver kolleksjon, slik at det er mulig for gruppen å definere hierarkiet på dataen selv. Den øverste kolleksjonen holder dokumenter for alle nettsteder som Styreportalen administrerer på sine løsninger, hvor hver organisasjon har ett dokument. Den enkelte bruker får tilgang til å redigere ett av dokumentene i kolleksjonen basert på organisasjonen brukeren tilhører. I hvert dokument finnes det to nye kolleksjoner, “files” og “pages”. Files-kolleksjonen inneholder forskjellig data som brukes på nettstedet, som for eksempel bilder, tekstfiler, markdownfiler ol. Pages-kolleksjonen inneholder dokumenter om hver “page”, hver nettside for nettstedet. Hvert page-dokument holder felter som definerer strukturen til siden, for eksempel hvilke moduler siden har, tema, m.m. I tillegg finnes det metadata for dokumentet, som en unik identifikator og opprettelsesdatoen. Alle dokumentene leses og skrives fra en *firebaseklient*, et dataobjekt som opprettes på klientsiden og kobler seg til databasen.

4.2 Tjenersidearkitektur

På tjenersiden benytter gruppen rammeverket *Next.js* som ble introdusert i kapittel 3. *Hvordan* og *hvorfor* gruppen har brukt dette rammeverket blir beskrevet i detalj i dette kapittelet.

4.2.1 Statisk generering av HTML

Next.js er rammeverket som ble benyttet på tjenersiden for applikasjonen. Formålet med å bruke dette rammeverket er muligheten for å statisk generere websider for hver forespørsel på forhånd.

Det betyr at alle websider genererer HTML, CSS og JavaScript ved oppstart, slik at når en klient forespør en side, går det mye raskere å returnere en respons. Denne måten å servere websider til brukere kalles *static site generation* (Hawksworth, 2020). Dersom dette ikke er mulig har man også mulighet til å generere websider når forespørselen fra bruker mottas. Dette kalles *server-side rendering* (Ramos, 2016). Grunnen til at man helst ønsker å forhåndsgenerere sider er fordi man oppnår mye raskere lastetid, som igjen øker det helhetlige inntrykket av løsningen.

Statiske websider består av HTML-, CSS- og JavaScript-kode lagret i filer, som sendes på forespørsel fra klient (Ramos, 2016). Hver webside inneholder statisk innhold som lastes inn fra en funksjon kalt *getStaticProps* (*Basic features: Data Fetching*…, 2021). Dette er en funksjon som eksporteres til Next.js, som laster inn data ved oppstart og deretter bygger websiden(e) ved bruk av denne dataen. Funksjonsnavnet er viktig fordi Next.js gjør en bestemt handling basert på navnet. Dersom dataene som hentes fra funksjonen endres etter oppstart, kan Next.js oppdatere den statiske siden i bakgrunnen og deretter sende den oppdaterte siden ved nye forespørsler. Dette konseptet kalles *incremental static regeneration* (*Basic features: Data Fetching*…, 2021), og står sentralt i applikasjonens måte å bygge websider på. Kundene kan dermed redigere sin webside samtidig som andre besøker den, uten at man behøver å ta nettstedet ned for vedlikehold.
I tillegg vil noen websider benytte seg av dynamisk innhold. Blant annet vil kalenderkomponenten laste inn ekstern data for kalenderhendelser til den enkelte forening på klientsiden. Ved klientsidelasting kan man legge inn intervaller for henting av kalenderdata, slik at den holder seg oppdatert uten å måtte laste siden på nytt. *Mediastore* er en annen komponent i applikasjonen som henter ressurser fra databasen på klientsiden. Årsaken til at klienten henter data for Mediastore, er at dataen er spesifikk for hver enkelt organisasjon, slik at man ikke har mulighet til å forhåndsgenerere dette.

Brukeren kan også sette opp egne *sider* på nettstedet, dersom det foretrekkes å ha enkelte moduler separert fra resten. For eksempel kan brukeren lage en egen side for kalender og terminlister, slik at man kan skille dette innholdet fra resten. Denne siden vil da ligge på www.<korpsnavn>.no/kalender, dersom kunden ønsker å navngi siden “kalender”.

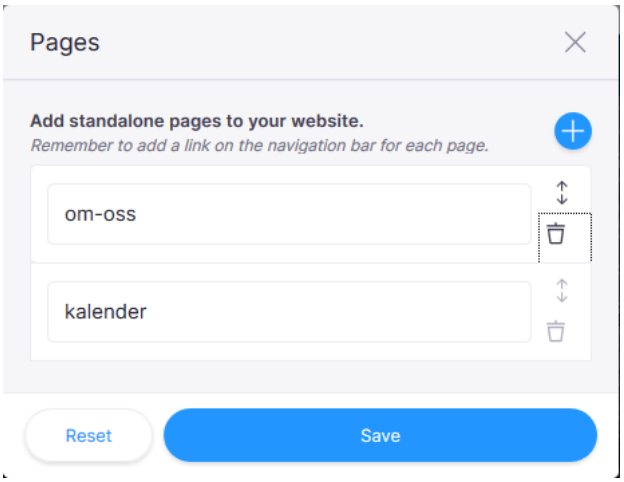


Fig. 4.2: Innstillingsmodal fra *TinaCMS* som endrer sider for nettstedet.

Figur 4.2 er et eksempel på hvordan innstillingsmodalen for sider i *TinaCMS* ser ut i applikasjonen. Det er mulig å legge til eller slette sider, og det fins ingen begrensning på antall sider brukeren kan ha. Navnet kan redigeres, og tilsvarer identifikatoren som brukes i URL.

For å kunne opprette sider under kjøretid, har applikasjonen en side kalt [page].js som tar seg av alle ruter og sjekker om de er lovlige. Dette kalles for dynamiske ruter, og identifiseres av Next.js ved å bruke hakeparenteser rundt filnavnet. Eksempelvis dersom brukeren har opprettet siden kalt kalender, så vil [page].js ta seg av forespørselen med URL www.<korpsnavn>.no/kalender. Derimot vil eksempelvis en forespørsel mot URL www.<korpsnavn>.no/notater være ulovlig, fordi brukeren ikke har opprettet denne siden. [page].js vil håndtere dette ved å omdirigere til en 404-side.
Måten [page].js sjekker om forespørselen er lovlig eller ei, er ved å bruke et Router-objekt, som er integrert i Next.js. [page] vil fungere som en parameter, og det er dermed mulig å hente denne ut med Router-objektet og sjekke ruten opp mot listen av sider som er opprettet.

En av applikasjonens viktigste funksjoner er muligheten for å vise nyheter om kundens forening eller organisasjon. For å få til dette, benyttes dynamiske ruter. URL for én enkel nyhetssak vil se slik ut: www.<korpsnavn>.no/news/[slug]. En *slug* er en unik identifikator for en webadresse (*Slug - MDN Web Docs*, 2021), på samme måte som man identifiserer [page]. Forskjellen mellom visning av nyheter og dynamisk opprettede sider er at applikasjonen har informasjon om hvilke *slugs* som finnes på forhånd. Det betyr at Next.js har mulighet til å bygge sider for alle *slugs* som finnes under oppstart av applikasjonen fremfor under kjøring. Listen med slugs hentes ved oppstart gjennom funksjonen *getStaticPaths* (*Basic features: Data Fetching | getStaticPaths*, 2021). Dataen om ruter hentes fra databasen i *getStaticPaths*, eksporteres deretter til Next.js og resultatet sendes som parameter til *getStaticProps*. Next.js vil i denne metoden hente data for alle *slugs*, og rendre sider for hver og én av dem.

4.3 Klientsidearkitektur

På klientside bruker vi en kombinasjon av rammeverk og verktøy. Hva disse verktøyene er, og hvordan de blir brukt for å løse problemstillingen, blir beskrevet mer detaljert i dette kapittelet.

4.3.1 Kommunikasjon mellom klient og database

Som nevnt i kap. 4.1.1, bruker gruppen Cloud Firestore, som er en tjeneste på Firebase plattformen. For å kommunisere med databasen, brukes et Javascript-objekt som representerer en klient. Firebase-klienten opprettes ved oppstart av applikasjonen, og kan brukes både på tjener- og klientside. Dette objektet er viktig for applikasjonen fordi det fungerer som en bro mellom tjener- og klientside, og databasen. Alle endringer som brukeren gjør, mates inn som en parameter i en objektfunksjon, som deretter sørger for å lagre dette på riktig måte i databasen.

Objektet har to hovedfunksjoner, *savePageData* og *fetchPageData*. *savePageData* lagrer data som er endret på klientsiden når brukeren forespør det, og applikasjonen viser deretter en dialog til bruker med resultatet fra forespørselen. *fetchPageData* henter data på tjenersiden, før websiden bygges og sendes til klienten.

4.3.2 Løsningens hovedkomponenter

Som nevnt i kapittel 3.3.2, benyttes *React*-komponenter for både å vise og å endre innhold. Hver komponent holder på sin egen tilstand, og kan få sine underkomponenter til å reagere på denne nye tilstanden når komponentens tilstand endres. Underkomponentene vil da lastes på nytt, med ny data fra forelder komponenten.
Et eksempel på en React-komponent som holder på tilstand, kan sees i kodesnutten i figur 4.3.

```
import { useState } from 'react'

const Nyhet = () => {

  const [value, setValue] = useState(0)

  return (
    <div>
      <button onClick={() => setValue(value+1)}>Trykk her</button>
      <p>Verdien er: {value}</p>
    </div>
  )
}

export default Nyhet
```

Fig. 4.3: En komponent som holder en tilstand kalt *value*, og returnerer JSX basert på verdien til *value*. Her blir *hooken useState* brukt for å holde på komponentens tilstand.

Dette er en helt vanlig Javascript-funksjon, som returnerer det som kalles JSX (Introducing JSX | React docs, 2021). JSX er en syntaktisk utvidelse som produserer React elementer. Uten å gå mer i detalj på hva JSX er, kan man se på det som en kombinasjon av HTML og Javascript. Funksjonene må nødvendigvis ikke returnere JSX, men det er anbefalt ettersom det kombinerer både det visuelle og det logiske.
Denne komponenten returnerer et React element som inneholder en helt vanlig HTML <button>, og en HTML <p> som viser tekst.

useState-funksjonen tar en initiell verdi som grunnlag for tilstanden. I eksempelet over setter settes variabelen *value* til 0 som utgangspunkt. Knappen har en *onClick* lytter som kaller funksjonen den er gitt når man trykker på den. Denne funksjonen bruker funksjonen *setValue* returnert av useState-hooken, som i eksempelet inkrementerer verdien til *value*.
Eksempelet i kodesnutten over er veldig enkelt, men tilstander til komponenter kan imidlertid være relativt komplekse datastrukturer.

Ofte lager man komponenter som kan gjenbrukes, og som kan ta informasjon fra en komponent lengre opp i komponent-hierarkiet. Da kan *forelder*-komponenten gi under-komponentene sine det som kalles *props (Components and Props | React docs, 2021)*. Props er en forkortelse for *properties*, som oversatt til norsk betyr egenskaper. Eksempelet under illustrerer hvordan dette kan gjøres.

```
const OnsdagsNyheter = () => {
  return (
    <div>
      <Nyhet dag={"onsdag"} />
      <Nyhet dag={"onsdag"} />
    </div>
  )
}

const Nyhet = (props) => {
  const [value, setValue] = useState(0)

  return (
    <div>
      <button onClick={() => setValue(value+1)}>Trykk her</button>
      <p>Verdien er: {value}</p>
      <p>Denne artikkelen ble skrevet på ukedag {props.dag}</p>
    </div>
  )
}
```

Fig. 4.4: En komponent som viser nyheter for ukedagen onsdag. Komponenten gjenbruker Nyhets-komponenter, og gir dem *props* i form av teksten "dag", som forklarer hvilken dag den ble skrevet på. Merk at nyhets-komponenten nå tar inn *props* i parameteren til funksjonen.

Dette er en veldig vanlig måte å jobbe med komponent-baserte rammeverk på. Typisk om man har en liste eller en tabell av elementer som kan visualiseres i nettleseren, lages det en komponent som representerer listen eller tabellen, og som videre holder på alle elementene i form av rader. Hvert element, eller rad, er én underkomponent enkeltvis. Dersom noen av

elementene har forskjellig innhold, eksempelvis tekst, bilder eller datoer, gis disse av komponenten lengre opp i hierarkiet gjennom *props* som ble beskrevet over.

Oppbygningen av komponentene til løsningen følger samme mønster som i eksemplene over. De holder på tilstand, kan reagere på tilstand fra komponenter lengre opp i komponent-hierarkiet, og endrer HTML strukturen sin dersom det har skjedd endringer i en forelder-komponents tilstand. Fra figur 4.3 kan det sees at tilstanden til en komponent settes først med en initiell verdi. De initielle verdiene for tilstanden til komponentene gruppen har laget hentes fra Firebase, og endres ved brukerinput. Endringen i tilstanden vil så føre til en ny HTML struktur som reflekteres på nettsiden.

Løsningen har syv hovedkomponenter, og blir beskrevet under.

4.3.2.1 Medlemsliste

Medlemsliste-komponenten lar bruker legge til visning av medlemmer i organisasjonen. Visningen presenteres i form av små kort, hvor brukeren har valg om å legge til bilde, navn, stilling og/eller arbeidsoppgavene til medlemmet. For at denne komponenten skal være responsiv, altså at den skalerer fint på mobil, nettbrett og datamaskin, er antall kort begrenset til maksimalt tre per medlemsliste. Bruker har imidlertid ingen begrensninger på hvor mange medlemslister som skal vises.
Denne komponenten har to under-komponenter; én komponent for opplastning og visning av bilde, og én komponent for redigering av tekst direkte på kortet, som er gjenbrukt tre ganger for navn, arbeidsoppgaver, og tilleggsinformasjon.

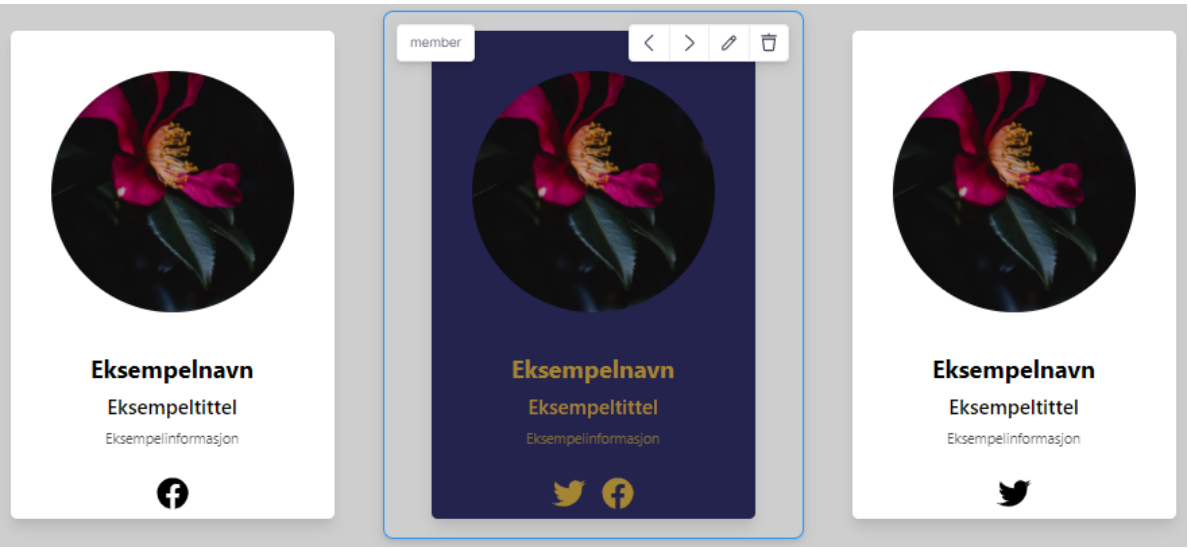


Fig. 4.5: Medlemslisten som inneholder tre medlemskort. Det merkede kortet representerer én enkelt underkomponent av listen. Disse er i dette tilfellet gjenbrukt tre ganger under selve liste-komponenten.

4.3.2.2 Kontaktform

Kontakt-komponenten er en komponent som viser en kontaktform, som besøkende bruker for å kontakte organisasjonen. Bruker bestemmer selv hvilken informasjon besøkende må oppgi for å sende en melding. Dette kan være fullt navn, e-postadresse og telefonnummer. Bruker kan også redigere e-postadressen som mottar meldingene, teksten på varselet eller dialogboksen besøkende ser når meldingen blir sendt, samt farge og tekst på send-knappen.

4.3.2.3 Introduksjon

I introduksjons-komponenten kan bruker legge til bilder og tekst, slik at organisasjonen får en oversiktlig introduksjon for besøkende. I denne komponenten er det mulighet for å redigere tekst direkte i nettleseren, samt å redigere hvordan bilde og tekst er plassert i forhold til hverandre. Introduksjons-komponenten er en samling av to under-komponenter, henholdsvis én komponent for redigering av tekst, og én for opplastning og valg av bilde.

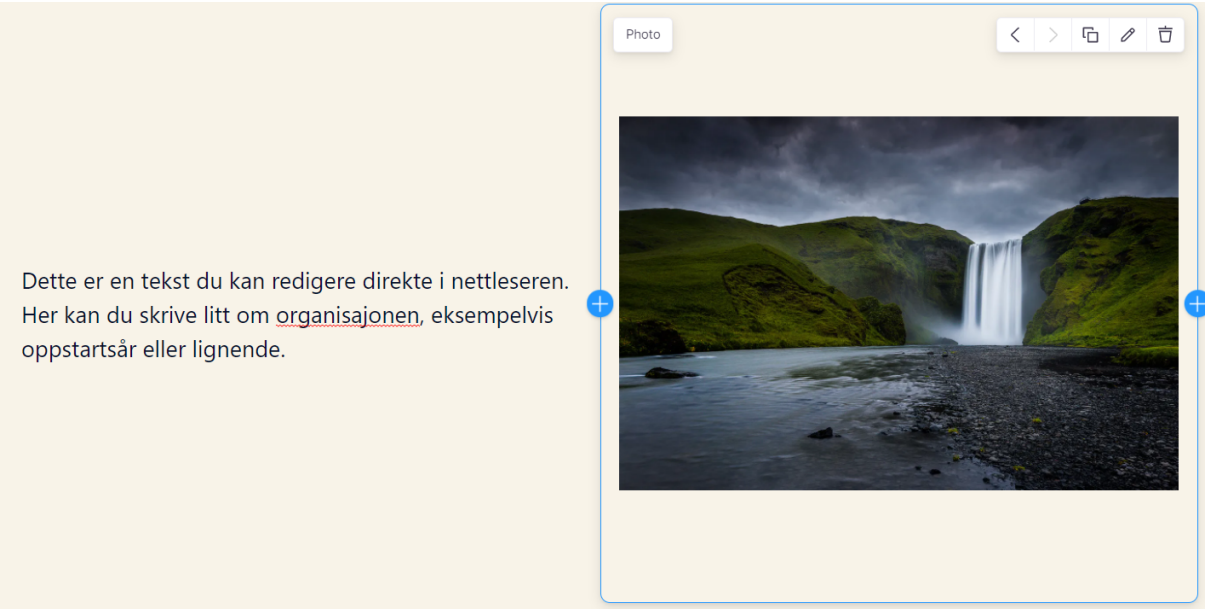


Fig. 4.6: En introduksjons-komponent som inneholder to underkomponenter. Den merkede komponenten (blått omriss) er for opplastning og endring av bilde, og er én av disse under komponentene.

4.3.2.4 Nyhetsvisning

Organisasjonen ønsker gjerne at besøkende kan holde seg oppdatert på siste nyheter. Denne komponenten henter ut nyhetsartikler fra Styreportalens interne tjenere basert på domenet, altså hvilken organisasjon det gjelder, og viser dem i nettleseren. Disse nyhetsartiklene er kortet ned, slik at kun bilde og introduksjon til artikkelen vises. Besøkende kan da navigere seg videre til artikkelen i sin helhet ved å trykke på *les mer*.
Bruker er begrenset til å vise maksimalt tre av de siste nyhetsartiklene, men bestemmer selv antallet opp til den begrensningen.

Denne komponenten har én under-komponent som gjenbrukes maksimalt tre ganger. Disse komponentene er selve nyhetsartiklene på kort format.

4.3.2.5 Forside

Når besøkende bruker nettsider, møtes de av et heldekkende bilde, gjerne av organisasjonen, med en tekst som ligger over bildet. Bruker kan trykke på dette bakgrunnsbilde direkte i nettleseren for å endre det.
Det ligger også en tekst over bildet, som bruker kan redigere i nettleseren, men kan velge å la det stå blankt dersom de ikke ønsker det.
Komponenten har to under-komponenter, én for å laste opp å redigere bilde, og én for å redigere teksten som flyter over bildet.

4.3.2.6 Toppmeny

Toppmenyen er en komponent som reflekterer seksjoner på nettsiden. Denne gir besøkende mulighet til å navigere seg gjennom innholdet. Navnet på hvert menypunkt kan redigeres, og bruker kan også legge til lenker til sosiale medier, med støtte for ikoner for kjente nettsteder som Facebook, Instagram og Twitter.

4.3.2.7 Kalendervisning

For at besøkende skal kunne se hendelser, kanskje i form av arrangementer i regi av organisasjonen, er det laget en komponent som viser disse i en kalender. Denne komponenten, i likhet med nyhets-komponenten, henter hendelsene hos Styreportalens interne tjenere basert på domenet.
Komponenten har én under-komponent, som er hentet fra FullCalendar (FullCalendar | FullCalendar, 2021). Dette er en ekstern komponent som Styreportalen selv bruker i sine løsninger i dag.

4.3.2.8 Overskrifter

Komponentene som er presentert til nå, må gjerne “kategoriseres”, slik at besøkende forstår hva innholdet i hver komponent betyr i kontekst. Det er derfor laget en overskrifts-komponent, som i utgangspunktet er ment for å introdusere innholdet i neste komponent, eller komponenten som vises i neste seksjon på nettsiden.
I denne komponenten kan bruker velge font, font-størrelse, kursiv eller fet tekst, store eller små bokstaver, og om overskriften skal plasseres til venstre eller i midten.
Denne komponenten har én under-komponent, som står for selve overskriften.

Fig. 4.7: Komponent for overskrifter, i dette tilfellet brukt for å presentere neste komponent på nettsiden, som er en medlemsliste.

I alle komponentene beskrevet over er det også mulighet for å endre bakgrunnsfarge, altså farge rundt selve innholdet i hver komponent.

Hvordan bruker bestemmer hvilke komponenter som vises for besøkende, blir beskrevet mer detaljert i neste kapittel.

4.3.3 Redigering av websider med TinaCMS

TinaCMS er en samling med verktøy som lar bruker legge til og redigere innhold på websider. Eksempler på innhold kan være tekst, bilder, farger og annet design som gir et personlig preg for organisasjonen eller foreningen. Det finnes to måter å redigere innhold på, enten via en *sidebar* eller *inline*. En *sidebar* er en egen komponent som kan åpnes og lukkes, og som rendrer alt innholdet på den aktuelle websiden som kan redigeres. *Inline*redigering derimot, betyr at man kan endre innholdet rett i websiden, uten behov for en ekstern komponent som rendrer redigeringskomponentene. Da vil TinaCMS supplere en verktøylinje som ligger øverst på websiden. Denne verktøylinjen gir tilgang til funksjoner som lagring, tilbakestilling og diverse *plugins*. Applikasjonen bruker *inline*redigering fordi det gir en god oversikt over innholdet, og brukeren får et bedre overblikk over helheten på websiden.

4.3.3.1 Former

Hver *sidekomponent*, en komponent som rendrer en webside, er tilknyttet en *form*. En *form* er en byggeblokk for applikasjonen som inneholder alt innhold og innstillinger knyttet til den aktuelle sidekomponenten (*Forms | TinaCMS Docs*, 2020). Når applikasjonen startes, blir former for alle sider hentet fra databasen og lastet inn i sidekomponentene, slik at brukeren kan fortsette å redigere på den siste versjonen.

```
const indexForm = {
    id: '../data/data.json',
    label: 'Hovedside',
    initialValues: indexData ?? data,
    onSubmit: async (formData) => {
        try {
            await savePageData(formData, 'index')
            saveSuccessCallback()
        } catch {
            saveFailureCallback()
        }
    },
}
```

Fig. 4.8: Form for indekssiden i applikasjonen.

Over er et eksempel på hvordan et formobjekt kan se ut i JavaScript. Objektet har variabler som beskriver formen til siden, og de viktigste er *initialValues* og *onSubmit*. *initialValues* forteller at *indexData* skal lastes inn i formen når applikasjonen starter opp. Denne dataen hentes fra databasen i funksjonen *getStaticProps* via Next.js, ref. kap. 4.2.1, og sendes til formen. Dersom noe skulle gå galt ved henting av dataen, laster formen inn *data*, som er en standardmal den kan falle tilbake på. *onSubmit* definerer funksjonen som skal utføres når bruker trykker "Lagre"-knappen på verktøylinjen. Funksjonen sender en forespørsel med data som skal lagres, og venter på responsen. Basert på resultatet av responsen returnerer den en visuell bekreftelse eller feilmelding til brukeren.

4.3.3.2 Blokker

Innholdet i en form består av *blokker*. En blokk er en datastruktur som inneholder flere ulike React-komponenter (DJ Walker, 2019). Alt innholdet i formen vil ligge inne i en blokk, som deretter kan inneholde flere blokker med komponenter. Dermed vil innholdet få en trestruktur, hvor det er mulig å nøste i flere nivåer.

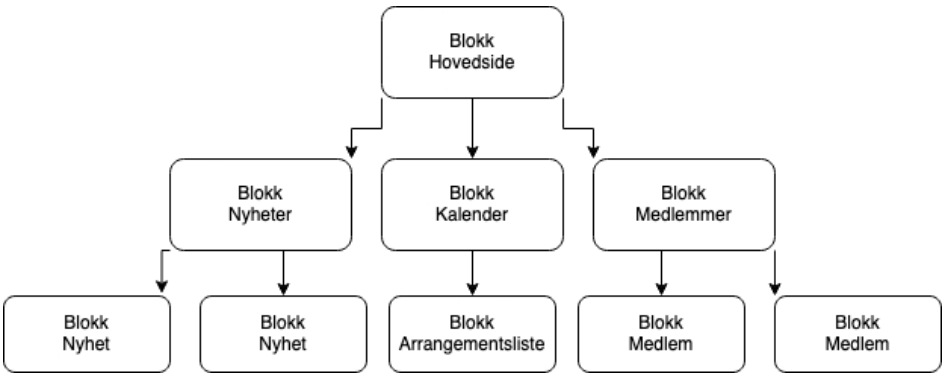


Fig. 4.9: Diagram for blokkstruktur

Blokkstrukturen kan se ut som i diagrammet over. Den øverste blokken er strukturen for hele siden, og inneholder blokker av hovedkomponenter, som for eksempel

medlemslistekomponentem, ref. kap. 4.3.2. Hver blokk av medlemslistekomponenten kan videre inneholde blokker av medlemskomponenter, slik at man til slutt ender opp med mange lag av blokker.

Applikasjonen bruker *inline blocks*, som betyr at man kan redigere blokker og endre rekkefølgen rett på websiden. For å få *inline blocks* til å fungere, må hver React-komponent pakkes inn i et eget blokk-objekt.

```
export const MembersBlock = {
  Component: ({ index, data }) => (
    <BlocksControls
      index={index}
      focusRing={{ offset: { x: -10, y: 0 } }}
      insetControls
    >
      <InlineBlocks
        blocks={{ member: MemberBlock }}
        name='members'
        max={3}
        direction='horizontal'
      />
    </BlocksControls>
  ),
}
```

Fig. 4.10: Blokk for medlemskomponenten

Over er et forenklet eksempel på hvordan medlemslistekomponenten er pakket inn i en blokk. *BlocksControls*-komponenten rendrer knapper for å blant annet justere blokkens plassering og instillinger. Innenfor *BlocksControls* ligger en *InlineBlocks*-komponent, som forteller hvilke blokker som kan opprettes i den nåværende blokken. I dette tilfelle kan vi kun rendre én type blokk, medlemsblokken. Hver blokk har muligheter for å redigeres av bruker, og TinaCMS har en enkel måte å løse dette på; *felter*.

Fig. 4.11: *BlocksControls*

4.3.3.3 Felter

Et *felt* er en komponent som tilhører en blokk, og gir brukeren mulighet til å redigere verdier for den aktuelle blokken. TinaCMS kommer med mange forhåndsdefinerte komponenter som kan brukes, og eksempler på disse kan være tekst-, farge- eller bildefelter. I medlemsblokken benytter man både tekst- og bildefelter som *inline*felter, mens innstillinger som er vanskelig å redigere

inline dukker opp i en egen *modal* ved å trykke på blyantknappen som rendres av *BlocksControls*. De samme feltene kan brukes uavhengig av om de skal redigeres inline eller i en modal.

```
export const italic = {
  name: 'italic_text',
  component: 'toggle',
  label: 'Italic text',
  description: 'Skru av og på kursiv tekst',
  toggleLabels: { true: 'On', false: 'Off' },
}
```

Fig. 4.12: *Eksempel på bruk av et felt. Dette feltet lar bruker velge om teksten skal være kursiv eller ikke. Komponenten som brukes er en 'toggle', og er en forhånds-definert TinaCMS-komponent, men her kan man også bruke egendefinerte komponenter om ønskelig.*

Som nevnt over tilhører feltene en blokk. Disse importeres i en Javascript-fil hvor man definerer blokken som et Javascript-objekt, og settes på *fields*, som er en tabell av felter. Eksempelet under viser hvordan overskrifts-komponenten tar bruk av flere felter som redigerer selve fremstillingen av overskriften, og bruker i dette tilfellet **ikke** inline-redigering.

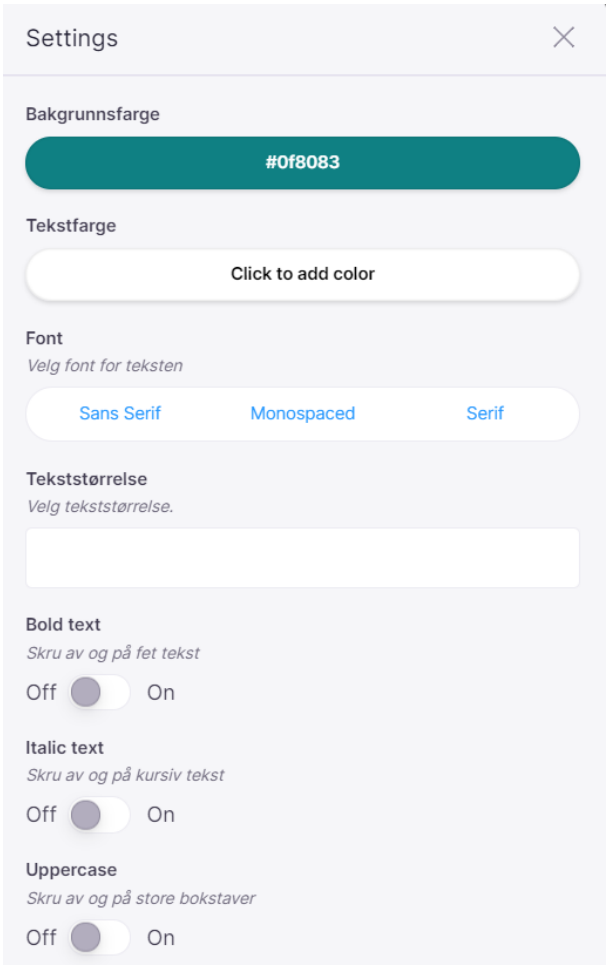


Fig. 4.13: Modal som gir bruker mulighet til å redigere stilen på overskrifter.

Under vises hvordan feltene settes på innstillingsmodalen til blokken for overskrifts-komponenten. Feltene fremstilles i modalen i samme rekkefølge som de settes på *fields*.

```
fields: [
    backgroundColor,
    textColor,
    font,
    textSize,
    bold,
    italic,
    uppercase,
    alignObj,
    customTailwind,
],
```

Fig. 4.14: Hvordan felter er satt på overskrifts-blokken.

4.3.3.4 Plugins

Stort sett alle komponentene redigeres *inline* ved bruk av former, blokker og felter, men det finnes noen unntak. Navigeringslinjen er et eksempel på en komponent som ikke kan redigeres *inline*, men isteden må redigeres gjennom en *modal*. Komponenten er likevel tilknyttet et formobjekt, og må dermed registreres gjennom TinaCMS som en *FormScreenPlugin*. Ved å registrere formobjektet som en plugin, vil brukeren kunne åpne en modal for og redigere komponenten ved bruk av diverse felter, som tidligere. Denne modalen ligger tilgjengelig sammen med andre *plugins* i Tinas verktøylinje. Formobjektet registreres ved bruk av en *React hook* kalt *useFormScreenPlugin*.

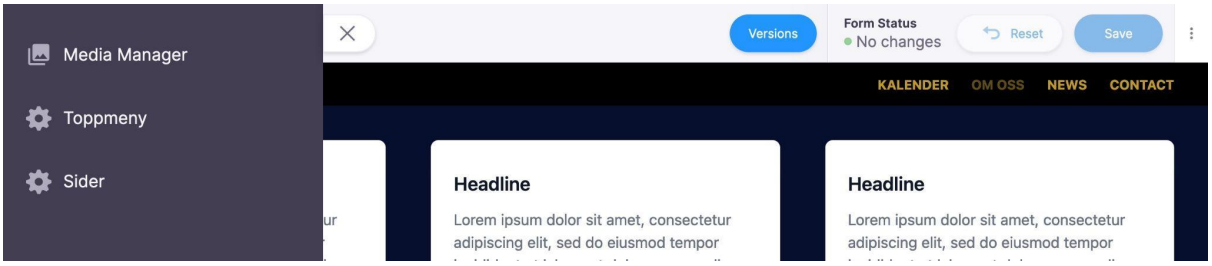


Fig. 4.15: Figuren viser alle *plugins* som er registrert i TinaCMS.

4.3.3.5 Media Manager

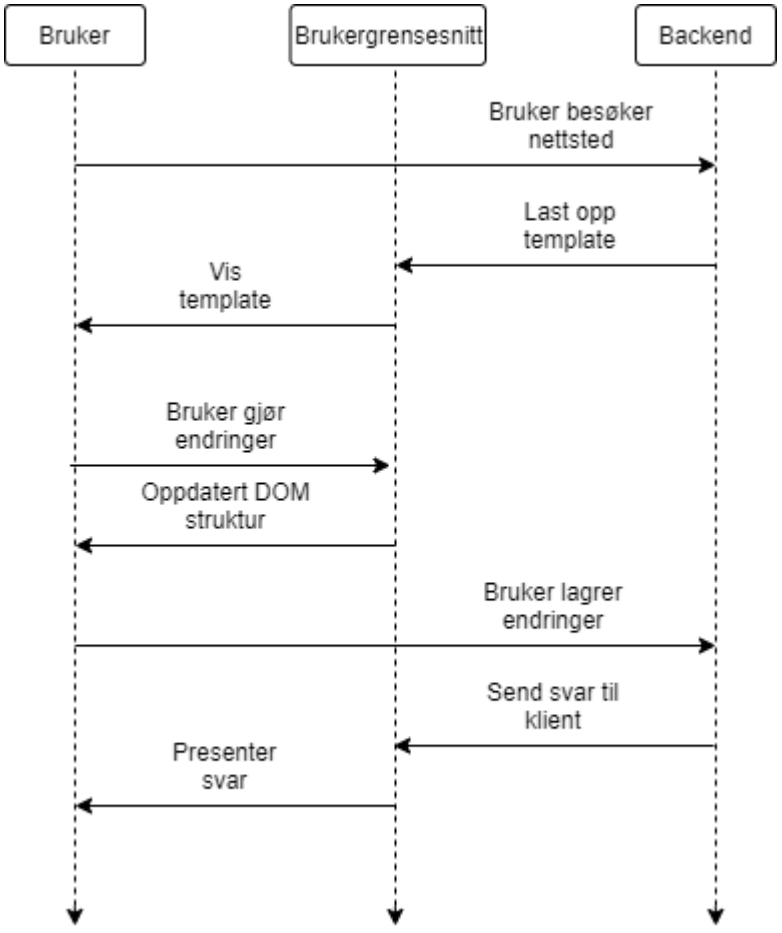
Det finnes også forhåndskonfigurerte plugins som kan benyttes sammen med TinaCMS, og en av disse er *Media Manager*. Media Manager gir brukeren mulighet til å administrere ressurser til

bruk på websiden, eksempelvis bilder eller markdownfiler. Bruker kan også laste opp nye filer som umiddelbart blir tilgjengelig for bruk. Når *Media Manager* klikkes av bruker åpner TinaCMS en *fullscreen sidebar*, en sidebar som dekker nesten hele skjermen, se appendix E.

4.4 Programsyklus

Interaksjonen med løsningen skjer i fire steg:

1. Bruker besøker nettsted, og laster opp siste endringer dersom dette **ikke** er første gang bruker besøker nettstedet. Tjeneren henter en template dersom dette er første besøk.
2. Bruker gjør endringer nettsiden i form av tekst, bilder eller rekkefølge på blokkene.
3. Bruker lagrer endringene, og venter på svar fra tjeneren.
4. Tjeneren svarer med resultatet av lagringen. Dersom lagringen var vellykket vises dette i en dialogboks med grønn tekst, tilsvarende vises feilmelding med rødt dersom det oppsto en feil.



Figur 4.16: Brukerinteraksjon

Figur 4.16 viser hvordan bruker interagerer med løsningen. Merk at endringer bruker gjør på nettsiden ikke lagres før bruker selv ber om det.

5 Evaluering

5.1 Evalueringsmetode

Gruppen benyttet mye intern testing underveis sammen med bedriften i forbindelse med evaluering av arbeidet. Det ble satt opp ukentlige møter med bedriften for å diskutere arbeidet som var gjort siden sist. Der gikk gruppen gjennom komponentene og funksjonene som var implementert, og alle ansatte fra bedriften som deltok hadde muligheten til å komme med mulige forbedringer eller alternative måter å gjøre implementasjonen på. Gruppen noterte ned og utførte forbedringene til neste møte.
Gruppen hadde også en sluttevaluering av arbeidet sammen med bedriften. Formålet med denne sluttevalueringen var å oppsummere arbeidet, diskutere om målene ble oppnådd, oppklare ting som bedriften var usikre på i forbindelse med kildekode. Spesielt viktig for gruppen var det å oppklare om applikasjonen oppnådde delmålet om at den skulle gjøre det "lettere for bedriften å vedlikeholde kundenes nettsider".

Gruppen benyttet brukertesting for å evaluere om applikasjonen var "brukervennlig og enkel å ta i bruk". Brukertestingen ble gjennomført ved at bedriften tok kontakt med en av deres kunder, ga en rask gjennomgang av hvordan applikasjonen brukes, og lot denne kunden prøve ut de forskjellige funksjonene. Kunden ga deretter en tilbakemelding til bedriften hvor det ble påpekt hva som fungerte bra og hvor det kunne gjøres forbedringer.

Gruppen benyttet *Jest.js* for å teste ulike funksjoner som ble brukt i applikasjonen, og det var spesielt fokus på testing av funksjoner som ble brukt til å lese og skrive data mellom applikasjon og database. Figur 5.1 er et eksempel på hvordan en test i *Jest.js* ser ut. *test* har to parametre, en beskrivelse av testen og en funksjon som skal utføres. I dette tilfelle er funksjonen asynkron, fordi den gjør en forespørsel til databasen ved å kalle funksjonen *fetchPageData* med parameter 'test null'. I denne testen forventes det at variabelen *data* skal være *null*, og grunnen til det er fordi man ikke forventer at parameterverdien til *fetchPageData* skal eksistere i databasen.

```
test('Fetching data from server with wrong document name', async () => {
    const data = await fetchPageData('test null')
    expect(data).toBe(null)
})
```

Fig. 5.1: Test for å sjekke om funksjonen *fetchPageData* fungerer som den skal.

For å teste applikasjonens ytelse, tilgjengelighet og *SEO*, benyttet gruppen verktøyet Lighthouse. Lighthouse er et Google Chrome-verktøy som analyserer webapplikasjoner og viser data om ytelse og *best practices* innen utvikling av programvare (Irish, 2016). Gruppen brukte verktøyet jevnlig for å teste om ny funksjonalitet hadde negativ påvirkning på applikasjonens ytelse.

Ytelsens poengscore beregnes ut i fra flere faktorer, blant annet (*Lighthouse Performance scoring*, 2021):

- Speed index: Hvor lang tid det tar å laste inn innhold på websiden.
- Time to interactive: Hvor lang tid det tar før siden er interaktiv.

I tillegg var gruppen interessert i å hente data om *SEO* underveis, for å sjekke at applikasjonen alltid var optimalisert for søkemotorer.

5.2 Evalueringsresultat

Resultatet fra brukertestingen ble for det meste positivt. Brukerens førsteinntrykk var at bruk av TinaCMS til å sette opp websiden fungerte veldig godt. Det var oversiktlig, og bruker var fornøyd med at man hadde mulighet til å redigere rett i websiden, slik at endringene vises umiddelbart. De forskjellige komponentene var enkel å ta i bruk, med et par unntak. Brukeren ønsket seg funksjonalitet for å sette opp kolonner med innhold, noe som applikasjonen ikke støttet. En annen ting som bruker synes var forvirrende, var at knappen som trykkes for å opprette nye blokker ikke er synlig som standard. I tillegg var det ikke lett å forstå hva knappen gjorde, ettersom det ikke fantes noen beskrivelse for den. Brukerens samlede inntrykk var at løsningen fungerte veldig godt til formålet, men at det kunne gjøres noen tilpasninger for å få den enda bedre.

Sluttevalueringen som ble gjennomført sammen med bedriften fokuserte på resultatet av prosjektarbeidet og deres oppfatning om måloppnåelse. Bedriftens samlede inntrykk var at gruppen i stor grad hadde oppnådd målet om å lage en prototype for en publiseringsløsning som var brukervennlig og enkel å ta i bruk, samt gjorde det lettere å vedlikeholde kundenes nettsider. Bedriften mente det var avgjørende at gruppen hadde fulgt anbefalingen om å bruke TinaCMS i applikasjonen, og at resultatet ikke hadde blitt det samme uten dette verktøyet. Videre konkluderte bedriften med at applikasjonen oppnådde alle de initielle kravene som ble satt. Basert på prosjektets utviklingsmetodikk, var det også rom for å legge til nye krav underveis. Bedriften mente at en stor del av kravene som ble satt underveis i utviklingen ble oppnådd, men at noe av funksjonaliteten hadde mangler.

Gruppen testet ytelsen med Lighthouse jevnlig gjennom prosjektets levetid, slik at man kunne danne et bilde av trenden. Ytelsen for applikasjonen holdt seg på et jevnt høyt nivå, og lå på mellom 90-97 av 100 mulige poeng i alle testene som ble gjennomført, med en avsluttende poengsum på 93. Gruppen er godt fornøyd med dette resultatet, som viste at applikasjonen hadde god ytelse selv om det ble lagt til flere komponenter og funksjoner gjennom prosjektets levetid.
SEO hadde en avsluttende poengsum på 73, så applikasjonen hadde et forbedringspotensiale på denne delen. Blant annet manglet siden *metabeskrivelser* og enkelte *img*-tagger hadde ikke en *alt*-attributt.

6 Resultat

Ved hjelp av verktøy som ble valgt tidlig i prosjektforløpet mener gruppen selv at de har oppnådd de initielle kravene til løsningen. Disse var å utvikle komponenter for medlemsvisning, kontaktform, nyhetsvisning, kalender, samt å kunne endre design og form på disse. Et av verktøyene gruppen har benyttet seg av, TinaCMS, har vært et godt valg for å komme i mål med de forskjellige komponentene.

Interntestingen, og diskusjonen sammen med Styreportalen på de ukentlige møtene gjorde at gruppen hele tiden fikk pekere på hvordan de forskjellige komponentene, eller initielle kravene, kunne implementeres og bli seende ut gjennom hele prosjektforløpet. Dette gjorde at gruppen holdt seg på riktig spor, og å til slutt kunne presentere en prototype til løsningen i sluttevalueringen.

Brukertesting ble avholdt mot slutten av prosjektforløpet, og det ble derfor ikke tid til å ta ut og bearbeide resultatene av denne for å høyne kvaliteten på løsningen. Den viste imidlertid et samlet positivt inntrykk av løsningen som gruppen har presentert, med noen mangler.

Funksjonaliteten som ble forespurt underveis handlet hovedsakelig om oppretting av flere undersider på samme domene. Gruppen mener selv at funksjonaliteten er på plass, men selve interaksjonen bruker må foreta seg for å oppnå ønsket resultat ble ikke like intuitiv som ønsket. Hva som kunne blitt gjort annerledes diskuteres i kapittel 7.

7 Diskusjon

Dette kapittelet tar for seg problemer eller utfordringer som gruppen støtte på under prosjektarbeidet. I tillegg diskuteres det alternative måter å løse deler av prosjektet på dersom resultatet ikke ble tilfredsstillende.

Underveis i prosjektarbeidet støtte vi på noen problemer i forbindelse med verktøyene som ble valgt til bruk i applikasjonen. Gruppen møtte på stabilitetsproblemer ved bruk av *inline editing* fra TinaCMS, og grunnen til dette er at hele *react-tinacms-inline*-pakken fremdeles er eksperimentell. Dette var også noe gruppen merket seg under utvikling av løsningen. Alternativet til *inline editing* kunne vært å redigere ved bruk av Tinas *sidebar*. Fordelen med å bruke TinaCMS til å redigere på denne måten, er at alle funksjonene er stabile og fungerer som de skal. Ulempen ved å benytte denne formen for redigering er at det blir mindre oversiktlig og brukervennlig, slik at brukeren potensielt hadde fått en dårligere opplevelse.

Inlineimage, som er en komponent eller en blokk for å endre et bilde med ett enkelt tastetrykk, sluttet å virke etter at man navigerte seg til en annen side på samme domene. Gruppen klarte ikke å diagnostisere hvor problemet lå, men antok at det var en feil i kildekoden til *Inline Image*-komponenten. Gruppen hadde ikke tid til å implementere en fiks for dette problemet i det ferdige resultatet, fordi det ikke var prioritert. Grunnen til at man valgte å ikke prioritere denne feilen, var fordi den enkelt kunne omgås ved å laste inn siden på nytt. Dersom gruppen hadde hatt tid til å rette opp i dette, hadde man lagt inn et eget *Image*-felt for blokkene det gjaldt. Dette innebærer ekstra klikk for brukeren, men samtidig så får man en funksjon som er stabil.

En annen problemstilling gruppen møtte på var måten brukeren setter opp sider under domenet, som var en forespørsel av prosjekteier som kom litt senere i prosjektforløpet. I løsningen må dette skje i to steg, ved først å gå til toolbaren og trykke på “sider”, deretter legge til en ny side med navn, navigere seg til innstillinger for toppmeny, for å så registrere navigeringsinnstillinger til den nye siden der. Dette er en litt omstendelig prosess, og ikke særlig intuitiv.
Skulle dette gjøres på nytt hadde gruppen tatt i bruk selvlagde *React hooks* for å dele en tilstand om sider og menypunkter på samme sted. Da kunne alle komponenter, hovedsakelig toppmenyen, brukt denne informasjonen og se endringer på den umiddelbart. Ved endring ville toppmeny-komponenten kunne registrere at brukeren har lagt til en ny side på en helt annen komponent, hente ut informasjonen om den nye siden gjennom den selvlagde hooken, for så å rendre et nytt menypunkt. På denne måten ville brukeren sluppet å ta de ekstra stegene for å nå målet sitt.

Denne funksjonaliteten var vanskelig å implementere sent i prosjektet, i og med at logikken for å opprette menypunkter, vise dem og å redigere dem ble relativt komplekst gjennom prosjektforløpet.

Prototypemodellering var en utviklingsmetodikk som passet gruppen godt under implementasjonen av kravene. Tidlig i prosjektforløpet var det vanskelig å se hvordan hvert krav til løsningen spesifikt skulle implementeres og bli seende ut, og denne metodikken gjorde at gruppen kunne prøve forskjellige fremgangsmåter for å nå målene til løsningen. Det gjorde også at gruppen kunne jobbe i et høyt tempo med høy terskel for å gjøre feil, noe som ga oss mye frihet. Med utgangspunkt i denne metodikken ble implementasjoner diskutert ukentlig sammen med prosjekteier og utviklere fra Styreportalen.
Baksiden med denne metodikken for gruppen ble at det var veldig enkelt å ta på seg flere oppgaver, i form av forespørsler fra prosjekteier. I visse perioder kunne det bli uklart hva som måtte prioriteres, og det kunne gå mye tid til små finjusteringer av krav gruppen så på som ferdig.

Riktig bruk av lisenser var også noe som var nytt for gruppen. Det var derfor viktig å sjekke lisenser på all kildekode som ble brukt. I *FullCalendar* ligger det funksjonalitet som ikke har åpen lisensiering, da spesifikt en tidslinje-visning av hendelser i kalenderen. Gruppen måtte derfor bruke standard-funksjonaliteten på åpen lisens. Ulempen med å ikke ha tilgang til hele kildekoden ble at komponenten ikke ble så detaljert som ønsket.

Testing av funksjoner ved bruk av Jest.js fungerte enkelt og greit, men kunne vært utført bedre. Testing av funksjoner som ble brukt i forbindelse med databasen kunne heller vært utført på *mock data*, slik at man ikke behøvde å teste mot databasen.
I tillegg burde gruppen gjennomført *snapshottesting* av de forskjellige React-komponentene som benyttes i prosjektet. *Snapshottesting* er en måte å teste komponenter på for å sjekke om resultatet som kommer ut fra komponenten stemmer overens med forventet resultat (Kariuki, 2020). Ved å bruke denne metoden for testing kan man sjekke eventuelle *sideeffekter* som oppstår når det skjer en interaksjon med komponenten.

Gjennomføring av brukertestingen gikk som planlagt, men tilbakemeldingen gruppen fikk var litt tynn. Det ble ikke gitt noen særlig detaljerte tilbakemeldinger, og gruppen fikk ikke så mye innsikt i brukerens opplevelse av forskjellig funksjonalitet i applikasjonen. Hadde det vært muligheter for å gjennomføre en ny brukertesting, ville gruppen ha sendt med instruksjoner til bruk av applikasjonen og spørsmål som brukeren måtte svart på. Sjansen for at gruppen satt igjen med mer nyttig informasjon fra resultatet hadde økt betraktelig dersom testen hadde blitt gjennomført på denne måten.

8 Konklusjon

Målet til prosjektet var å utvikle en prototype for en publiseringsløsning for Styreportalens kunder, som er brukervennlig og enkel å ta i bruk, samt å gjøre det lettere å vedlikeholde kundenes nettsider. Gruppens oppfatning er at målet om å utvikle en brukervennlig prototype i stor grad er oppnådd. Valget av TinaCMS som verktøy for brukergrensesnittet viste seg å være svært viktig for oppnåelsen av dette målet, noe prosjekteier også konkluderte med. Mulighetene brukeren får til å redigere og sette inn innhold rett i websiden er unikt for dette verktøyet, og bidrar til å løfte brukeropplevelsen. Resultatet av brukertesten fortalte at applikasjonen hadde mangler som gjorde at den ikke var så enkel å ta i bruk som gruppen hadde ønsket. Gruppen kom derfor ikke helt i mål med dette delmålet. En sannsynlig grunn til dette kan være at brukeren ikke fikk instruksjoner på forhånd om hvordan man bruker applikasjonen.
Delmålet om å lage en applikasjon som gjorde det lettere å vedlikeholde kundenes nettsider ble oppnådd, noe bedriften også konkluderte med. I den nåværende løsningen som blir brukt, har bedriften få muligheter til å gjøre egne tilpasninger. Ved å ta i bruk prototypen, vil bedriften oppnå mer fleksibilitet i forhold til hvilke funksjoner som kan tilbys til kundenes nettsider, samt gjøre det enklere å rette opp i eventuelle feil, uten å måtte gå via en tredjepartsløsning.

Resultatet av prosjektet kan brukes til flere løsninger. Med verktøyene gruppen har benyttet seg av er det mulighet for å lage skreddersydde applikasjoner for flere typer kunder. Det beste eksempelet er kommersielle applikasjoner, hvor man kan la nyoppstartede bedrifter lage eksempelvis nettbutikker hvor de kan styre presentasjon av produkter og tjenester, nyhetsbannere og kampanjer, bare med et par tastetrykk direkte på websiden.
For videre arbeid hadde gruppen fortsatt med implementasjonen av undersider, og hvordan brukere setter opp dette. Dette var funksjonalitet gruppen ikke kom helt i mål med, selv om det fungerer til en viss grad.
Gruppen hadde også bearbeidet resultat av brukertesting for å høyne kvaliteten på løsningen, og prøvd å løfte den ut av prototype-stadiet. Da må det også integreres mot flere av Styreportalens tjenere som eksempelvis autentisering, og til slutt produksjonsdatabaser.

Det er gruppens mening at prototypen som er utviklet har gode forutsetninger for å brukes videre i utviklingen av et fullverdig produkt. Selv om ikke alle målene ble fullt oppnådd, har prototypen bevist at verktøyene som ble benyttet har fungert godt, og at det er mulig å bygge videre på disse. Prosjekteiers planer for løsningen er uansett å bruke vår kodebase, enten som referanse, eller som utgangspunkt for å løfte den ut til sine kunder i løpet av 2021, noe gruppen er veldig fornøyd med.

9 Referanser

Lewis, S. (2019), *What is the prototyping model?*, Tilgjengelig fra: https://searchcio.techtarget.com/definition/Prototyping-Model (Hentet: 17. april 2021).

Content Management System (CMS), Tilgjengelig fra: https://www.optimizely.com/optimization-glossary/content-management-system/ (Hentet: 18. april 2021).

Cornes, B. (2021) *Tailwind CSS IntelliSense*, Tilgjengelig fra: https://marketplace.visualstudio.com/items?itemName=bradlc.vscode-tailwindcss (Hentet: 10. mai 2021)

git (2021) Tilgjengelig fra: https://git-scm.com/ (Hentet: 29. mars 2021)

npm, Inc. (2021) Tilgjengelig fra: https://www.npmjs.com/ (Hentet: 29. mars 2021)

Microsoft (2021) Tilgjengelig fra: https://code.visualstudio.com/ (Hentet: 29. mars 2021)

GitHub Inc. (2021) Tilgjengelig fra: https://github.com/ (Hentet: 29. mars 2021)

TimaCMS (2019-2021) Tilgjengelig fra: https://tina.io/ (Hentet: 29. mars 2021)

Vercel (2021) Tilgjengelig fra: https://nextjs.org/ , (Hentet: 29. mars 2021)

BrowserStack (2021) Tilgjengelig fra: https://www.browserstack.com/responsive-design , Hentet: 29. mars 2021

https://jestjs.io/ , Hentet: 29. mars 2021

Facebook Inc. (2021) Tilgjengelig fra: https://reactjs.org/ (Hentet: 29. mars 2021)

Pawel (2019) *10 awesome tips for developing user friendly apps,* Tilgjengelig fra: https://itcraftapps.com/blog/10-awesome-tips-to-developing-user-friendly-apps/ (Hentet: 7. mai 2021)

Hawksworth, P. (2020) *What is a static site generator?*, tilgjengelig fra: https://www.netlify.com/blog/2020/04/14/what-is-a-static-site-generator-and-3-ways-to-find-the-best-one/ (Hentet: 11. mai 2021)

Ramos, M. (2016) *SSGs Part 1: A static vs dynamic website*, tilgjengelig fra: https://about.gitlab.com/blog/2016/06/03/ssg-overview-gitlab-pages-part-1-dynamic-x-static/ (Hentet: 11. mai 2021)

Basic features: Data Fetching | Incremental static regeneration, tilgjengelig fra: https://nextjs.org/docs/basic-features/data-fetching#incremental-static-regeneration (Hentet: 11. mai 2021)

Basic features: Data Fetching | getstaticprops-static-generation, tilgjengelig fra: https://nextjs.org/docs/basic-features/data-fetching#getstaticprops-static-generation (Hentet: 11. mai 2021)

Routing: Dynamic routes, tilgjengelig fra: https://nextjs.org/docs/routing/dynamic-routes (Hentet: 18. mai 2021)

Basic features: Data Fetching | getStaticPaths, tilgjengelig fra: https://nextjs.org/docs/basic-features/data-fetching#getstaticpaths-static-generation (Hentet: 18. mai 2021)

Slug - MDN Web Docs Glossary, tilgjengelig fra: https://developer.mozilla.org/en-US/docs/Glossary/Slug (Hentet: 18. mai 2021)

https://www.freecodecamp.org/news/legacy-software-maintenance-challenges/

Hooks | React docs, tilgjengelig fra: https://reactjs.org/docs/hooks-intro.html (Hentet: 20.mai 2021)

Introducing JSX | React Docs, tilgjengelig fra: https://reactjs.org/docs/introducing-jsx.html (Hentet: 20. mai 2021)

Components and Props | React docs, 2021, tilgengelig fra https://reactjs.org/docs/components-and-props.html (Hentet: 20. mai 2021)

FullCalendar | FullCalendar docs, 2021, tilgjengelig fra https://fullcalendar.io/docs (Hentet 21. mai 2021)

Forms | TinaCMS Docs, tilgjengelig fra: https://tina.io/docs/plugins/forms/ (Hentet: 19. mai 2021)

DJ Walker (2019) *What are Blocks? | Tina Blog*, tilgjengelig fra: https://tina.io/blog/what-are-blocks/ (Hentet: 20.mai 2021)

Mogstad, Håkon *NMF + Styreportalen = Sant*, tilgjengelig fra: https://musikkorps.no/nmf-styreportalen-sant/ (Hentet: 28.mai 2021)

Toft Sundbye, Live Marie (2018) *Segmenter og målgrupper*, tilgjengelig fra: https://ndla.no/subject:1:433559e2-5bf4-4ba1-a592-24fa4057ec01/topic:2:183193/resource:1:88177 (Hentet: 31. mai 2021)

Licenses | Open Source Initiative, tilgjengelig fra https://opensource.org/licenses (Hentet: 31. mai 2021)

Irish, Paul (2016) *Lighthouse*, tilgjengelig fra: https://github.com/GoogleChrome/lighthouse/blob/master/readme.md (Hentet: 1. juni 2021)

HTML: HyperText Markup Language| *mozilla web docs, 2021,* tilgjengelig fra: https://developer.mozilla.org/en-US/docs/Web/HTML (hentet 2. juni 2021)

Kariuki, John (2020) *How To Write Snapshot Tests For Components With Jest*, tilgjengelig fra: https://www.digitalocean.com/community/tutorials/how-to-write-snapshot-tests-for-react-components-with-jest (Hentet: 2. juni 2021)

Lighthouse Performance scoring (2021), tilgjengelig fra: https://web.dev/performance-scoring/?utm_source=lighthouse&utm_medium=devtools (Hentet: 2. juni 2021)

Codebase | *Wikipedia* (2021), tilgjengelig fra: https://en.wikipedia.org/wiki/Codebase (Hentet: 3. juni 2021)

Backend | *techterms* (2020), tilgjengelig fra: https://techterms.com/definition/backend (Hentet: 3.juni 2021)

Third party | *techterms* (2009), tilgjengelig fra: https://techterms.com/definition/thirdparty (Hentet: 3.juni 2021)

Komponent | Store norske leksikon (2017), tilgjengelig fra: https://snl.no/komponent (Hentet: 3.juni 2021)

API | Store norske leksikon (2020), tilgjengelig fra: https://snl.no/API (Hentet: 3.juni 2021)

10 Appendix

Appendix A: Risikoliste

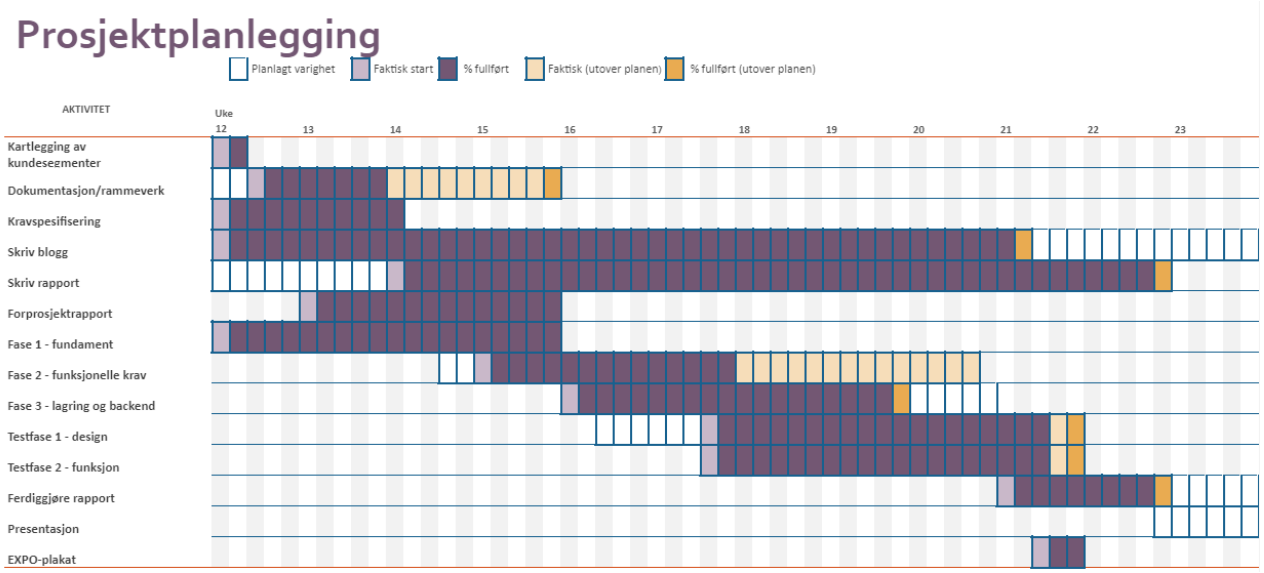
Suksessfaktor	S	K	RF	Tiltak	Interessent	Fase
Dårlig design	3	4	12	Diskutere med Styreportalens egne designere. Samle tilbakemeldinger fra kunder i testfasen.	Kunder, prosjekteier	Implementasjon – fase 3
Tidsforsinkelse	3	5	15	Strukturere og fordele oppgaver bedre. Finne minst akseptable løsning, slik at prosjekteier har et godt utgangspunkt for videreutvikling.	Utviklere, kunder, prosjekteier	Implementasjon – fase 1 til 3
Bruk av kildekode med restriktive lisenser	2	2	4	Diskutere mulig kjøp av lisens med prosjekteier. Alternativt implementere funksjonaliteten selv.	Utviklere, prosjekteier	Implementasjon – fase 1 til 3

Appendix B: Ordliste

Ord	Beskrivelse
CMS	*Content Management System* - styring av innhold.
Kodebase	Samling av kildekode.
Backend	Delen av et datasystem som ikke sees av en klient eller bruker, og som ikke interageres med direkte.
Tredjepartsapplikasjon	Programvare laget av en annen leverandør.

Komponent	Én del eller byggeblokk til en løsning.
API	*Application Programming Interface* - et sett av funksjoner eller prosedyrer for å lage applikasjoner som benytter seg av en annen applikasjon, operativsystem, eller tjeneste.
DOM	*Document Object Model* - API for HTML og XML dokumenter.
Rammeverk	En abstraksjon som tilbyr generisk funksjonalitet.
Versjonskontrollsystem	Et system som håndterer endringer på kildekode.
BaaS	*Backend-as-a-service* - mellomvare for programutviklere og databasetjenester i skyen.
SEO	*Search Engine Optimization* - prosess for å øke kvalitet og kvantitet i nettrafikk.
Host(ing)	Vert for applikasjoner.
JSON	*Javascript Standard Object Notation* - lesbar, åpen standard filformat og kommunikasjonsmedium.
Klient	Et dataprogram som kjører lokalt og som kommuniserer med en applikasjon eller tjeneste over et nettverk.
Modal	Et grafisk kontrollvindu underordnet en applikasjons hovedvindu.
JSX	*Javascript XML* - Syntaktisk utvidelse av Javascript som produserer React elementer.
React hook	React livssyklus-funksjoner for funksjonelle komponenter.
URL	*Uniform Resource Locator* - referanse til en ressurs som spesifiserer lokasjonen til ressursen på en datamaskin på et nettverk.
Template	Et startpunkt eller utgangspunkt for et dokument.
SEO	*Search Engine Optimization* - hvor optimalisert applikasjonen er for søkemotorer.
Best practices	Gode vaner og rutiner for å skrive god kode.
Sideeffekter	Modifisering av data utenfor komponentens lokale miljø.
Mock data	Falsk data som brukes til testing av programvare.

Appendix C: Gantt-skjema



Appendix D: Liste over npm-pakker

Pakke	Beskrivelse	Lisens
fontawesome	Samling av fonts, CSS og ikoner	CC BY 4.0
fullcalendar	Kalender komponent	MIT
tinacms	Redigering av innhold	Apache
tippyjs	Grafiske verktøytips	MIT
tailwindcss	Forhåndsdefinerte CSS-klasser	MIT
firebase	Klient-bibliotek for kommunikasjon mot diverse Firebase tjenester	Apache 2.0
swr	Henting av data på klientside	MIT
react	Komponent-basert webrammeverk	MIT
react-hot-toast	Grafisk notifikasjon	MIT

next	Statisk side-generering med React	MIT

Appendix E: Media Manager