Bacheloroppgave

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

Imran, Zaim
Schau, Max Torre

Utvikling av et internt HR-system med utforsking av no-code rammeverk

Bacheloroppgave i Dataingeniør
Veileder: Nils Tesdal

Mai 2021

NTNU
Norwegian University of
Science and Technology

Imran, Zaim
Schau, Max Torre

Utvikling av et internt HR-system med utforsking av no-code rammeverk

Bacheloroppgave i Dataingeniør
Veileder: Nils Tesdal
Mai 2021

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

NTNU

Kunnskap for en bedre verden

Forord

Denne rapporten er skrevet for Norges teknisk-naturvitenskapelige universitet i forbindelse med bacheloroppgaven for dataingeniør. Oppgaven bygger på prosjektet "Lettvekts HR-system" gitt av Blank. Ved videre lesing vil man få en oversikt over selve oppgaven, hva som er blitt utviklet, problemstillingen og tilknyttet teori. I tillegg vil problemstillingen bli besvart.

Teamet fant denne oppgaven spennende av flere årsaker:

- Det var et reelt problem for bedriften, og en god implementasjon av dette ville gi god verdi til hele firmaet
- Oppgaven utfordret teamet til å prøve ut ny teknologi
- Løsningen kunne bli implementert av bedriften og kunne være med på å skape et bedre miljø hos de ansatte
- Teamet så verdien av å teste ut arbeidslivet ved å jobbe for et konsulentselskap

Under denne perioden var verden fortsatt rammet av covid-19-pandemien. Dette førte til at prosjektet ble stort sett gjennomført digitalt med noen få unntak hvor teamet kunne møtes fysisk. Det har vært utfordrende, men samtidig svært lærerikt å gjennomføre bacheloroppgaven heldigitalt.

Teamet ønsker å uttrykke en spesiell takk til følgende personer for hjelp under bacheloroppgaven:

- Nils Tesdal, universitetslektor fra Institutt for datateknologi og informatikk ved NTNU, for rollen som veileder og all tid han har brukt for å veilede oss
- Magne Davidsen, Even Kallevik og Erlend Åmdal fra Blank, for deres rolle som kunde og veiledere i løpet av prosjektet
- Alle som har stilt opp for å gjennomføre brukertester av systemet
- Våre medstudenter for å ha vært sterke bidragsytere til tre flotte og lærerike år

Trondheim, 20. mai 2021 — Sted, Dato

Zaim Imran — Imran, Zaim

Trondheim, 20. mai 2021 — Sted, Dato

Max T. Schau — Schau, Max Torre

Oppgavetekst

Hensikten er å utforske styrker og svakheter ved et no-code rammeverk sammenlignet med tradisjonell programvareutvikling gjennom å implementere et lettvekts HR-system. Bruksområder, arkitektur, utviklingshastighet og integrasjonsmuligheter er spesielt interessant å utforske.

Blank er et konsulentselskap bestående av ca 50 utviklere og designere, som lager digitale skreddersømsløsninger for store og små kunder. Oppgaven går ut på å lage et enkelt HR-system for å støtte de i Blank som har personalansvar med å følge opp de ansatte. I dag løses dette ved hjelp av maler for huskelister.

Systemet må kunne:

- ha oversikt over alle ansatte (legge til og fjerne ansatte)
- holde oversikt på hvem som har personalansvar for hvem
- støtte forskjellige prosesser: (Onboarding, Løpende oppfølging, Følge progresjon innenfor fagutvikling og tilpassing, Offboarding)

Systemet bør kunne:

- sende påminnelser om oppgaver til de som har personalansvar
- la de ansatte velge personalansvarlig
- booke medarbeidersamtale
- integrere med Blanks internsystemer

Forandringer på oppgaveteksten

Oppgaveteksten har endret seg etter planlegging og diskusjoner med oppgavestiller. Blant annet skal ikke systemet ha mulighet til å legge til og fjerne ansatte fordi systemet skal integreres direkte med Blanks internsystemer. Vi gikk også vekk fra planen om å kunne booke medarbeidersamtaler og la en ansatt velge sin egen personalansvarlig. Teamet og oppgavestiller ble i stedet enige om å fokusere på oppgaveoversikt og automatisk opprettelse av oppgaver.

For nærmere beskrivelse av krav til oppgaven se Vedlegg A Visjonsdokument kapittel 5 Produktets funksjonelle egenskaper og Vedlegg B Kravdokumentasjon kapittel 2 User Stories.

Sammendrag

Blank er et konsulentselskap som holder felleskap og de ansattes tilfredshet høyt. For å sikre dette må alle ansatte bli jevnlig fulgt opp av sin personalansvarlig. Dette er grunnen til at vi i denne bacheloroppgaven har utviklet systemet TRAK.

TRAK er en web-applikasjon som er utviklet med React-rammeverket Next.js. Systemet tilbyr en intuitiv oversikt over oppgaver knyttet til de ansatte. I tillegg til dette gir TRAK en oversikt over alle ansatte, og statusen på ansatte i de ulike prosessene (onboarding, løpende og offboarding).

Ikke minst minimaliserer TRAK arbeidet de personalansvarlige må gjøre, ettersom det automatisk blir opprettet oppgaver ved ansettelse og oppsigelse. Dette gjøres ved å lytte til firmaets internsystemer. Personalansvarlig behøver heller ikke opprette løpende oppgaver ettersom systemet tar seg av dette.

I løpet av bachelor-perioden har teamet utforsket fordeler og ulemper med et no-code rammeverk sammenlignet med tradisjonell programvareutvikling. Dette ble gjort ved å utvikle et tilsvarende system med no-code rammeverket Bubble, hvor teamet så på ulike aspekter ved utviklingsprosessen og sluttproduktet.

Resultatene tilsier at man kan, visuelt sett, oppnå identiske løsninger, men at det er flere begrensninger knyttet til det funksjonelle. Deriblant er det utfordrende å utvikle funksjonalitet som involverer kompleks logikk, men for enkle, mindre komplekse nettsider fungerer no-code rammeverk godt.

Videre viser resultatene at det kan være svært effektivt å utvikle et system med et no-code rammeverk sammenlignet med tradisjonell programvareutvikling.

Bubble tilbyr sikkerhet ut av boksen, slik at systemet dekker viktige sikkerhetskrav. På den andre siden må man med tradisjonell utvikling selv avgjøre hvordan dette skal implementeres. Dette er tidkrevende, og kan føre til sikkerhetshull ved feil implementering.

No-code rammeverket tilbyr ikke tilstrekkelig tilgjengelighet som en standard. I tillegg er det heller ikke mulig å tilpasse komponentene for å øke tilgjengeligheten. Det innebærer at man ikke nødvendigvis kan utvikle løsninger som innfrir kravene til universell utforming med et no-code rammeverk. Dette er i kontrast til tradisjonell utvikling hvor man ikke har disse begrensningene.

Innhold

Figurer

Akronymer

API Application Programming Interface

AWS Amazon Web Services

HTTP Hypertext Transfer Protocol

ORM Object Relation Mapping

OWASP Open Web Application Security Project

REST Representational state transfer

SSL Secure sockets layer

SSR Server Side Rendering

WCAG Web Content Accessibility Guidelines

Ordliste

I denne rapporten blir det benyttet en del begreper som teamet sammen med oppgavestiller har blitt enige om. Disse blir brukt uten nærmere forklaring i hovedrapporten, og vil derfor bli forklart i dette kapittelet.

Prosess

En prosess beskriver hva slags stadiet den ansatte er i, og kan være enten onboarding, offboarding eller løpende.

Fase

Enhver prosess vil bestå av en eller flere faser. Fasene har en fast rekkefølge slik at man alltid går gjennom den samme rekken faser for å fullføre en prosess.

Onboarding

Dette er prosessen som en ansatt er i når hen skal starte i jobben.

Offboarding

Prosessen som en ansatt er i når hen skal slutte i jobben.

Løpende

Dette er en prosess som skjer årlig, og som gjerne deles opp kvartalsvis.

Prosessmal

Prosessmalen er en mal som består av alle fasene med tilhørende oppgaver. Malen fungerer som et utgangspunkt for alle ansatte slik at alle oppgaver som opprettes tar utgangspunkt i denne malen.

Revisjonshistorikk

Dato	**Versjon**	**Beskrivelse**	**Forfatter**
11/01/21	0.1	Oppsett av kapitler	Max T. Schau og Zaim Imran
06/05/21	1.0	Skrevet første utkast	Max T. Schau og Zaim Imran
12/05/21	1.1	Gått over skrivefeil	Max T. Schau og Zaim Imran
18/05/21	2.0	Gjort klar for innlevering	Max T. Schau og Zaim Imran

Tabell 1: Revisjonshistorikk

1 Introduksjon

1.1 Bakgrunn for oppgaven

Blank er et konsulentselskap som holder til i Oslo. Selskapet har ca. 50 ansatte, og består av både utviklere og designere. For å sikre at alle ansatte opplever å ha det bra og føler seg ivaretatt er det nødvendig å jevnlig følge opp hver enkelt ansatt. Dette gjelder den perioden de starter eller slutter i jobben, og på et årlig basis.

Tidligere har det blitt benyttet Trello for å holde oversikt over oppgavene som må gjøres. Dette førte til at Blank måtte manuelt opprette alle oppgavene for hver ansatt, og systemet hadde heller ingen mekanisme som passet på at oppgavene ble gjort innen rimelig tid. Det hele ble veldig manuelt og tidkrevende, og sannsynligheten for at oppgaver ble glemt bort var stor. Firmaet følte derfor et sterkt behov for å automatisere opprettelsen av disse oppgavene. Videre ønsket de å ha et system som var skreddersydd for å sømløst passe på at oppgaver blir gjort ved å kunne minne personalansvarlige på deres ansvar.

1.2 Problemstilling

Systemutvikling har de siste tiårene gjennomgått store forandringer. Nye rammeverk til det allerede eksisterende HTML-, CSS- og JavaScript-økosystemet har blitt utviklet, og i 2021 eksisterer det mange muligheter for å kunne utvikle webløsninger.

John Rymer, hovedanalytiker ved Forrester Research uttalte i et intervju med Betty Blocks[1] at mange av hans klienter gjennomgår en stor forandring i måten de gjør forretninger på. Blant annet står automatisering mer sentralt enn det har gjort tidligere. For å få til dette er det nødvendig at firmaene har programvare av høy kvalitet [16, 00:42].

I samme intervju følger Chris Obdam opp med at "One of the things people tend to say now a days is that software is eating the world. We need a lot of software. And we cannot really rely on the traditional programmers to do that because there are not enough of them." [16, 02:12]

Som Obdam og Rymer nevner er det et stadig behov for mer programvare i verden, men det er mangel på teknologer som skal utvikle disse. De impliserer at no-code rammeverk kan være en løsning på problemet. I lys av disse uttalelsene har teamet valgt å utforske et no-code rammeverk, og har kommet frem til følgende problemstilling:

"Hvilke fordeler og ulemper er det ved å bruke et no-code rammeverk sammenlignet med tradisjonell utvikling"

Det er mange ulike aspekter å se på med problemstillingen, men vi har valgt å konkretisere det ned til følgende forskningsspørsmål:

1. Hvilke begrensninger og muligheter medfører no-code rammeverk for det visuelle og funksjonelle, sammenlignet med tradisjonell utvikling?
2. Vil det være gunstig å benytte et no-code rammeverk med tanke på utviklingshastighet?
3. I hvilken grad vil sikkerheten bli ivaretatt for et no-code rammeverk sammenlignet med tradisjonell utvikling?
4. Kan man med et no-code rammeverk oppnå samme tilgjengelighet som man kan oppnå med tradisjonell utvikling?

[1] Betty Blocks er et no-code rammeverk

1.3 Struktur i hovedrapporten

Denne rapporten er delt opp i 8 kapitler.

Kapittel 1 - Introduksjon tar for seg selve oppgaven, og hvordan oppgaveteksten var formulert med de endringene som har blitt gjort underveis. I tillegg vil også problemstillingen for rapporten bli introdusert.

Kapittel 2 - Teori presenterer litteraturstudiumet knyttet til utviklingen og problemstillingen som blir definert i kapittel 1.

Kapittel 3 - Valg av teknologi og metode handler om de ulike valgene som er blitt tatt underveis i prosjektet. Det omfatter blant annet valg av utviklingsrammeverk og utviklingsmetodikk.

Kapittel 4 - Resultater tar for seg resultatene knyttet til det vitenskapelige, ingeniørfaglige og administrative.

Kapittel 5 - Diskusjon presenterer drøfting og refleksjon rundt resultatene fra kapittel 4.

Kapittel 6 - Konklusjon og videre arbeid vil svare på problemstillingen presentert i kapittel 1, og ta for seg videre arbeid knyttet til utviklingen.

Kapittel 7 - Referanser viser de ulike kildene benyttet.

Kapittel 8 - Vedlegg presenterer vedleggene til rapporten.

2 Teori

Dette kapittelet vil ta for seg det teoretiske arbeidet utført for å kunne svare på problemstillingen fra kapittel 1.2, og nødvendig teori benyttet for å utvikle systemet.

2.1 Systemutvikling

2.1.1 ORM

Relasjonsdatabaser og objekt-oriententert programmering er velkjente termer for studenter innen informatikk. De er begge basert på paradigmer som har flere ulikheter mellom seg [21, s. 1]. Dette problemet kalles gjerne for "object-relational impedance mismatch". Problemet går ut på at det oppstår en rekke utfordringer når man forsøker å koble seg til en relasjonsdatabase ved hjelp av en objekt-orientering tilnærming på tjeneren [10].

På et overordnet nivå tilbyr ORM en løsning på problemet ved å konvertere objekter til relasjonell data for å kunne benytte en objekt-orientert tilnærming med relasjonsdatabaser [21, s. 2]. Dette gjør at utviklingen kan bli mer effektiv.

2.1.2 REST

Ved bruk av interaktiv og dynamisk data på en webklient er det nødvendig å kunne kommunisere med en webtjener. Dette kan gjøres ved hjelp av HTTP-verb som beskriver hensikten med en forespørsel. Denne forespørselen kan enten hente, lage, oppdatere eller slette en ressurs som ligger lagret hos webtjeneren. Ved hjelp av disse begrensningene vet klienten hvordan den skal bearbeide data fra tjeneren og kan utføre oppgaver basert på denne informasjonen [15].

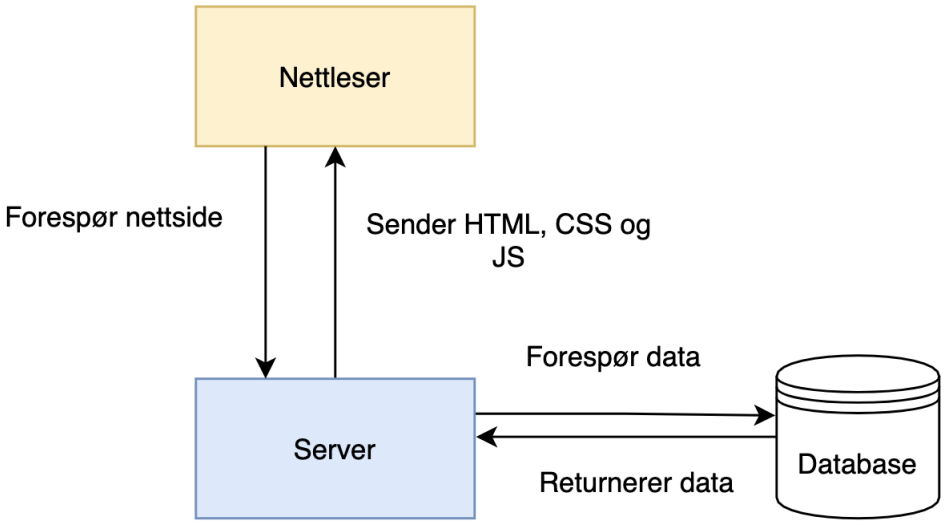
Ifølge Roy Thomas Fielding kan et API kalles for *restful* dersom følgende prinsipper er fulgt [5]:

- Klient-tjener
  - Man skiller mellom klienten og tjeneren som to uavhengige parter slik at endringen av den ene ikke påvirker den andre. Dette er med på å forbedre skalerbarheten
- Tilstandsløshet
  - Enhver forespørsel fra klienten til tjeneren må inneholde all nødvendig informasjon for at tjeneren skal forstå forespørselen
- Cache
  - Enhver forespørsel må inneholde informasjon om den skal være mulig å cache eller ikke. Dersom den er *cacheable* vil klienten ha rettighet til å kunne gjenbruke responsen ved en senere anledning
- Uniformt grensesnitt
  - Man standardiserer grensesnittet mellom komponenter for å simplifisere den overordnete systemarkitekturen
- Lagdelt system
  - Klienten har ikke oversikt over hvem den kommuniserer med. Det kan være sluttjeneren eller andre mellom-tjenester. Årsaken er at tjenere som er plassert mellom klient og sluttjeneren ikke skal ha påvirkning på systemet
- Kode på etterspørsel (valgfritt)

– Med REST kan man utvide klient-funksjonaliteten ved å laste ned og kjøre kode. Dette kan for eksempel være applets eller scripts

2.1.3 SSR

Server Side Rendering omhandler måten en nettside blir bygget opp på. Tjeneren aksesserer data fra databasen, og genererer HTML-siden på forhånd. Deretter blir dette sendt tilbake til klienten. Dette står i kontrast til "client side rendering" hvor all HTML blir generert av klienten [17].



Figur 1: Server Side Rendering

Nettleseren begynner med å sende en forespørsel til tjeneren. Tjeneren henter deretter data fra databasen og generere HTML som blir sendt tilbake til nettleseren.

2.1.4 Kontinuerlig integrasjon

Kontinuerlig integrasjon er, ifølge M. Fowler og M. Foemmel i artikkelen "Continuous Integration", at man automatisk bygger, integrerer, tester, rapporterer og installerer koden. Når man er flere medlemmer på et team vil man ofte jobbe med ulike deler av koden, og vil måtte på et tidspunkt slå sammen sin del av koden med den sentrale koden. Fordelen med kontinuerlig integrasjon er at det automatisk vil bygge prosjektet og kjøre tester når man forsøker å gjøre dette. Dermed vil man også kunne si at den overordnede byggingen er godkjent når testene og bygging av prosjektet er vellykket. Ved feil vil det ikke være mulig å slå sammen koden med den sentrale koden [6, s. 3].

2.1.5 Kontinuerlig utrulling

Kontinuerlig utrulling henger sterkt sammen med *Kontinuerlig integrasjon*. På tilsvarende måte vil man kjøre de automatiserte testene for å forsikre seg om at kodebasen er stabil. Ved suksessfull testing og bygging vil man kunne rulle ut systemet til produksjonsmiljøet [12].

2.1.6 Cron

Cron er et verktøy som utfører gitte oppgaver periodisk ved fastsatte tider, datoer eller intervaller. Cron-jobbene kan utføre operasjoner som å gjøre et API-kall, kjøre script-filer og redigere

tekst-filer. Ved hjelp av cron kan man automatisk vedlikeholde komplekse systemer ved å kjøre enkle tester hver uke, hver dag eller hver time [4].

2.1.7 Hvordan kjører en cron jobb?

En cron-tjeneste bruker et cron-uttrykk til å bestemme når en jobb skal utføres. Cron-uttrykket er en streng med seks variabler skilt med mellomrom [4].

Eksempelvis vil følgende cron-uttrykk bety at jobben skal kjøres 10:15 mandag til fredag:

$$0\ 15\ 10\ ?\ *\ MON-FRI$$

2.2 Sikkerhet

2.2.1 OWASP Top Ten

OWASP Top Ten er en liste over de ti vanligste sikkerhetsrisikoene for nettsider. Sikkerhetseksperter fra hele verden jobber sammen for å lage og oppdatere listen. Denne listen er en veiledning for utviklere som ønsker å utvikle sikrere systemer og beskytte mot de vanligste sikkerhetshullene [11].

2.3 Brukervennlighet

2.3.1 Don Normans 6 designprinsipper

Don Norman er en amerikansk forsker og forfatter som er blitt kjent for sin kunnskap innen brukervennlighet og design. Norman har blant annet utarbeidet seks prinsipper som kan være viktige for å sikre et intuitivt og brukervennlig brukergrensesnitt [23] [18]:

- Synlighet
  - Når en bruker aksesserer et grensesnitt skal man umiddelbart se hvilke muligheter man har. På den måten har man også en tanke om hva man skal gjøre videre
- Tilbakemelding
  - Tilbakemeldingsprinsippet handler om at brukere skal få tilbakemeldinger når en handling har blitt utført. For eksempel når en bruker oppretter en entitet skal vedkommende få en melding om dette gikk bra eller ikke
- Begrensninger
  - Dette handler om å begrense mulighetene til en bruker på visse steder i grensesnittet. Hensikten er å unngå at brukeren blir overveldet av mulighetene som finnes. For eksempel vil det å forhindre en bruker fra å skrive bokstaver i et felt for telefonnummer være en form for begrensning
- Mapping
  - Mapping-prinsippet går ut på at de kontrollene tilknyttet noe må samsvare med den faktiske effekten av kontrollene. For eksempel vil piltastene opp og ned samsvare med at man kontrollerer noe opp eller ned i et grensesnitt
- Konsistens

  - Ifølge prinsippet skal lignende elementer gi lignende resultater. Slik vil brukeren kunne gjenkjenne mønstre, og bruke erfaringen fra tidligere operasjoner til å gjennomføre nye operasjoner på grensesnittet

- Overkommelig
  - Systemet må være intuitivt slik at brukeren umiddelbart ser linken mellom hvordan et element ser ut og hvordan det skal brukes. For eksempel vil en museknapp indikere at man trykker på et element

2.3.2 Universell utforming

Ron Mace fra North Carolina State University sa i 1998 at universell utforming er:

"The design of products and environments to be usable by all people, to the greatest extent possible, without the need for adaptation or specialized design" [3, s. 2]

Slik Mace nevner handler universell utforming om at alle former for produkter og grensesnitt skal fungere uavhengig av hva slags forutsetninger brukeren har. Produktet skal dermed fungere for alle, uten å måtte gjøre konkrete tilpasninger til brukeren. Således skal grensesnittet være det samme, og fungere likt uavhengig om bruker er rød-grønn fargeblind, om brukeren er blind eller om brukeren ikke har noen utfordringer i det hele tatt.

2.3.2.1 WCAG

Web Content Accessibility Guidelines er en rekke universelle retningslinjer som bidrar til at nettet blir mer tilgjengelig for alle. Retningslinjene dekker ikke alle problemer som brukere kan ha, men hjelper utviklere og designere å løse problemer som kan være relevante.

I nyere tid har WCAG blitt utformet basert på fire prinsipper. Nettsiden skal være intuitiv, uavhengig av brukerens erfaring og kunnskap. Derfor skal det være mulig for alle å forstå hvordan man anvender nettsiden. I tillegg skal systemet være brukbart for alle brukere, uavhengig av hvilke hemninger de måtte ha. Tredje prinsippet understreker at nettsiden skal være forståelig. Det betyr at alle bør kunne gå på nettsiden, og forstå hva som blir forklart. Det impliserer at nettsiden bør forsøke å forenkle komplekse temaer, og bruke enkle og forståelige begreper. Til slutt må nettsiden være robust, slik at den kan brukes med andre verktøy og applikasjoner som skjermlesere, nettlesere og andre teknologier som en bruker kan anvende [22].

2.3.3 7 prinsipper om universell utforming

The Center for Universal Design publiserte i 1997 syv prinsipper som omhandler universell utforming[3, s. 3]:

- Like muligheter for alle
  - Alle brukere skal ha like muligheter på nettsiden. Både når det gjelder personvern, sikkerhet og trygghet på nettsiden

- Fleksibel i bruk
  - Fleksibel i bruk handler om at grensesnittet må være fleksibelt nok til å tilpasse seg brukeren. For eksempel om brukeren er venstre- eller høyrehendt. Dette kan sammenlignes med at på et museum vil det alltid være mulig å både lese og høre informasjon om et objekt

- Enkel og intuitivt i bruk

  - Man skal fjerne unødvendige elementer fra en grensesnitt for å øke intuitiviteten. På et generelt grunnlag skal brukerne klare å benytte seg av systemet, uavhengig av brukerens erfaring og kunnskap

- Forståelig informasjon
  - Grensesnittet skal formidle nødvendig informasjon på en effektiv og enkel måte. I tillegg skal det være et klart skille mellom viktig og uviktig informasjon
- Toleranse for feil
  - Man ønsker å minimalisere faren for feil og farer. I tillegg skal grensesnittet gi advarsler til brukeren, og tilby funksjoner som er feilsikre
- Lav fysisk anstrengelse
  - Prinsippet sier at man skal minimere den fysiske anstrengelsen til brukeren. Det vil si at brukeren skal kunne bruke systemet på en lett og komfortabel måte. Derfor ønsker man å for eksempel minimere repetitive oppgaver
- Størrelse og plass for tilgang og bruk
  - Grensesnittet må tilby nok plass og størrelse slik at brukeren har enkel tilgang og god nok rekkevidde, uavhengig av brukerens fysiske størrelse

2.4 Utviklingsmetodikk

2.4.1 Agil utviklingsmetode

Utvikling vil ofte medføre endringer av krav og forventninger. For å kunne tilpasse seg de konstante endringene kan man anvende smidige utviklingsmetoder. Utvikling av systemet skjer stegvis med hyppig utrullinger. Ved å dele opp prosjektet i mindre deler, kan man omprioritere delene og likevel ha en god flyt i utviklingsprosessen. Eksempler på smidige utviklingsmetodikker er *Scrum*, *Kanban* og *Lean*. [27]

2.4.2 Scrum

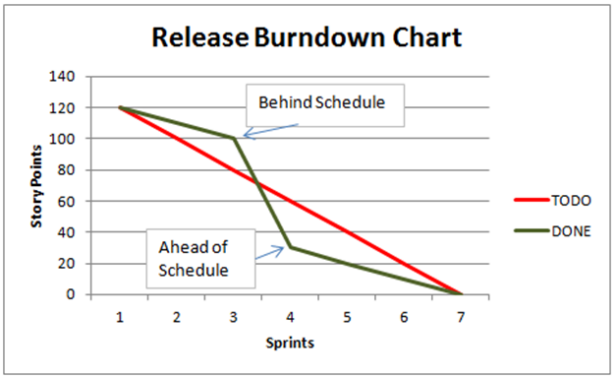
Scrum er en utviklingsmetodikk som hjelper personer, team og organisasjoner til å møte komplekse problemer, samtidig som man øker produktiviteten og leverer produkter av høy kvalitet. Metodikken bygger på "The agile manifesto" som understreker følgende [7]:

- **Personer og samspill** fremfor prosesser og verktøy
- **Programvare som virker** fremfor omfattende dokumentasjon
- **Samarbeid med kunden** fremfor kontraktsforhandlinger
- **Å reagere på endringer** fremfor å følge en plan

I Scrum benytter man seg av sprinter som er selve kjernen i metoden. Disse varer i en til fire uker hvor hensikten er å få utviklet et videre inkrement av produktet. Man definerer et sprintmål som man ønsker å nå i løpet av sprinten. Selve sprinten kan forstås som et mindre prosjekt av det overordnede prosjektet [25].

Videre har Scrum flere artefakter. Eksempelvis *product backlog*, hvilket er en liste over krav som må innfris for å forbedre produktet. Denne er gjerne prioritert slik at man sørger for at teamet er klar over hvilke krav som må prioriteres [24].

I tillegg finnes *sprint backlog*. Denne inneholder elementer fra product backlogen, men er spesifisert mot en sprint. Det vil si at de elementene som blir lagt i sprint backlogen skal fullføres i den aktuelle sprinten. Den siste artefakten er *burndown chart*. Det er et diagram som viser teamets fremgang i sprinten. På den måten gir det en pekepinne på hvorvidt man er i ferd med å bli ferdig med alle oppgavene i sprint backlog.



Figur 2: Burndown chart hentet fra [2]

Videre har man definert ulike roller for de involverte aktørene. En *scrum master* har det overordnede ansvaret for å lede teamet gjennom prosjektet. I tillegg har man en *produkteier* som ofte representeres av en kunde. Det er produkteieren som bestemmer hvilke gjenstander fra *product backlogen* som skal prioriteres i sprintene. Til slutt er det utviklerne som er ansvarlig for å utvikle selve systemet [29].

I forkant av en sprint har man et sprint-planleggingsmøte. Her bestemmer man hvilke elementer fra product backlog man ønsker å få gjennomført i løpet av sprinten [30]. For å holde alle teammedlemmene oppdaterte på hva som gjøres har man stand-up møter. Dette er møter som gjerne skjer en gang i løpet av dagen hvor man informerer kort og presist om hva man har gjort siden sist, hva man ønsker å gjøre og eventuelle hindringer for å nå det man ønsker.

Ved endt sprint vil man gjennomføre en sprint-review. Her deltar produkteier, scrum master, kunde, utviklerteam og andre interessenter for å se på en demonstrasjon av det som har blitt utviklet i løpet av sprinten. Det er under denne hendelsen man avgjør hvorvidt sprinten er vellykket i henhold til sprintmål og hvilke gjenstander som er fullført. Dersom noen av gjenstandene fra sprint backlog ikke er fullført vil de bli videreført til neste sprint.

Til slutt vil man ha en *sprint retrospektiv*. Dette er en seremoni for teamet til å diskutere og evaluere sprinten som var. Hensikten er å finne ut av hva som fungerte, hva som ikke fungerte og eventuelle tiltak for å forbedre teamet-arbeidet. Slik sørger man for å hele veien forbedre samarbeidet for å øke effektiviteten og kvaliteten i teamet [26].

2.5 No-Code

2.5.1 Hva er et no-code rammeverk?

No-code er en type utviklingsplattform som benytter seg av et grafisk brukergrensesnitt for å kunne utvikle applikasjoner til mobil og web. Ved å benytte seg av "drag and drop"-funksjoner kan man enkelt flytte komponenter rundt omkring på nettsiden. Dette muliggjør at personer uten kjennskap til tradisjonell utvikling kan utvikle og teste applikasjoner [8]. De tekniske delene av utviklingen blir nemlig abstrahert bort ved hjelp av det grafiske brukergrensesnittet.

2.5.2 Ulike typer no-code rammeverk

Med no-code rammeverk kan man bygge ulike typer applikasjoner. Det er her man hovedsakelig skiller mellom de ulike typene. Altså, hva er det som kan produseres ved hjelp av rammeverkene. Ifølge no-code rammeverket Betty Blocks kan man lage tre typer systemer ved hjelp av tilsvarende no-code rammeverk [28]:

- Database-applikasjoner

- Web-applikasjoner
- Mobil-applikasjoner

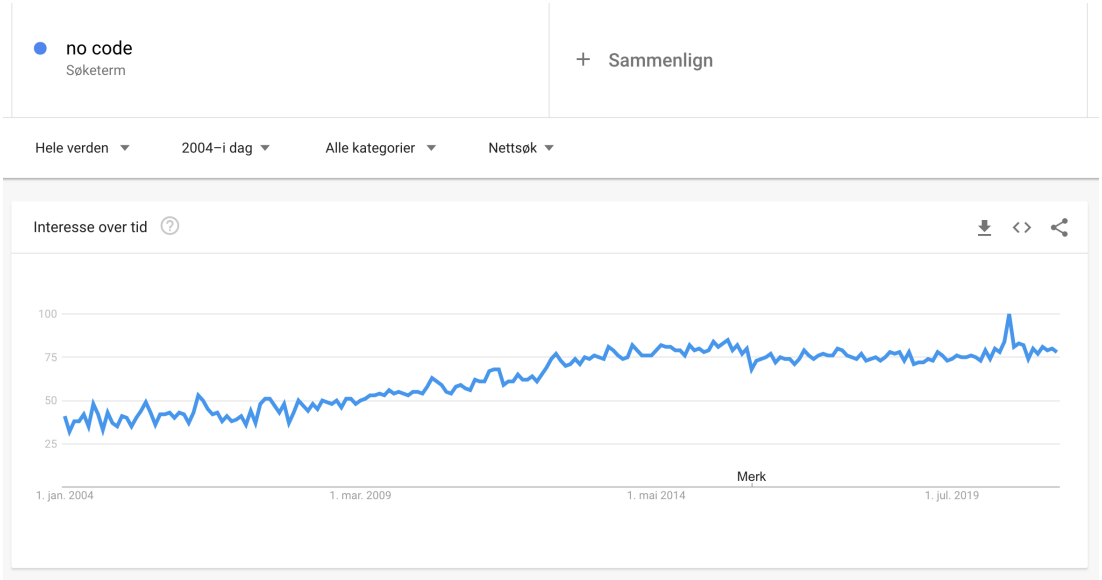
2.5.3 Forskjell på no-code rammeverk og low-code rammeverk

Low-code og no-code begreper som deler flere likhetstrekk. Der no-code abstraherer bort all form for kode, vil low-code rammeverk kreve små mengder med kode som må bli skrevet [20]. Likevel etterstreber de det samme; å muliggjøre utvikling av applikasjoner for alle på en rask og sømløs måte.

2.5.4 Historisk utvikling

No-code rammeverk er på ingen måte et nytt fenomen. Det har vært tilstede i mange år, men det er ikke før i senere tid at det har tatt av i programvareindustrien. Allerede i mai 2003 ble Wordpress lansert som et av de første no-code rammeverkene i verden [31].

Siden den gang har det dukket opp stadig nye utviklings-plattformer. Flere har blitt svært populære, og man ser at av alle eksisterende nettsider, er flere utviklet med no-code plattformer. For eksempel var det ifølge en statistikk fra Netcraft fra 2020 rundt 450 000 000 nettsider som var utviklet med Wordpress [1].



Figur 3: Google Trends siden 2004

Grafen fra figur 3 viser søkeinteressen relativ til det høyeste punktet på diagrammet. Det vil si at en verdi på 100 gjenspeiler det tidspunktet hvor søketermen var mest populær. Uansett gir dette en god indikasjon på interessen for no-code er økende, og at det stadig er flere som har interesse for å finne ut hva dette går ut på.

3 Valg av teknologi og metode

3.1 Klient

3.1.1 Next.js

Next.js er et React-rammeverk som ble lansert i 2016. Siden den gang har rammeverket blitt kontinuerlig utviklet, og er benyttet av mange av verdens største selskaper[9].

Teamet valgte å benytte seg av Next.js av flere årsaker. Først og fremst var det ønskelig å prøve ut et nytt rammeverk, og oppdragsstiller hadde fra tidligere gode erfaringer med rammeverket. Det tilbyr flere fordeler. Blant annet er det bygget på React og passer godt inn i Javascript, React og Node-økosystemet. Rammeverket gir også muligheten for å rendre[2] React-apper på tjeneren før den genererte HTML blir sendt til klienten.

Måten nettsiden blir bygget på (omtalt i 2.1.3) gjør at systemet blir svært raskt da all HTML genereres hos tjeneren, og ikke på klient-siden. Dette gjør at systemet blir svært responsivt for brukerne, hvilket kan øke brukertilfredsheten.

3.1.2 Material UI

Material UI er et komponentbibliotek utviklet for React. Biblioteket kommer med mange ferdigdefinerte komponenter som man kan benytte seg direkte av eller som base for egne komponenter. I tillegg er det svært godt dokumentert med flere eksempler og god dokumentasjon av hver enkelt komponent.

Material UI har fokus på universell utforming og WCAG 2.1 ved utvikling av sine komponenter. Det betyr at mange av WCAG-kriteriene allerede er implementert ved å benytte seg av komponentene. Dette gir fordelen av at det blir tidsbesparende å benytte seg av biblioteket.

3.1.3 TypeScript

I utgangspunktet er JavaScript et dynamisk programmeringsspråk. Det vil si at datatypen for variabler ikke blir oppgitt når man deklarerer en variabel. TypeScript fungerer som en utvidelse av JavaScript, og tilbyr utvikleren å typesette variabler. På den måten vil man kunne oppdage feil tidligere, da TypeScript forteller utvikleren allerede før kjøring at en feil har forekommet.

Teamet valgte å bruke TypeScript i prosjektet ettersom dette gir fordelen av at man innfører statisk typesjekking i prosjektet. Videre vil man få en mer strukturert kodebase som dokumenterer seg selv og gjør den mer lesbar. Dette er fordelaktig når man er flere utviklere, ettersom det blir enklere å forstå andres kode.

Dermed konkluderte vi med at TypeScript ville hjelpe teamet å utvikle en bedre og godt dokumentert kodebase, samtidig som det ville være med på å effektivisere utviklingen.

3.2 Tjener

3.2.1 Next.js

I tillegg til å kunne utvikle hele brukergrensesnittet med Next.js er det også mulig å bygge et API. Dette gjør at man ikke behøver å utvikle en dedikert tjener for endepunktene.

[2]Generering av HTML

En fordel ved dette, er at det ikke er nødvendig å *hoste* tjener og klient hver for seg, ettersom Next.js tilbyr både en klient og tjener i samme rammeverk. I tillegg er det lettere for utviklerteamet å jobbe i samme økosystem, nemlig Node med Typescript som allerede blir benyttet på klientsiden. Det vil også være gunstig for kunden, da økosystemet er lett og billig å hoste. Årsaken til dette er at det kun trengs én node-instans for å kjøre systemet.

3.2.2 Prisma

Prisma er et ORM-rammeverk for Node.js og TypeScript. Prisma støtter flere ulike databaser, deriblant MySQL, PostgreSQL og SQLite [14]. Dette ga teamet muligheten til å koble til PostgreSQL-databasen ved hjelp av ORM (2.1.1). ORM er fordelaktig, fordi man kan ha en objektorientering tilnærming til utvikling av databasen. De ulike modellene blir definert i et skjema som Prisma tolker, som videre bygger opp SQL-setninger basert på disse modellene. Dette gjør det svært effektivt å gjøre direkte kall til databasen, da Prisma oversetter spørringene til SQL-setninger. På den måten blir det mer effektivt og oversiktlig å utvikle.

I tillegg til å gi en oversiktlig kodebase, vil Prisma parametrisere spørringene slik at man ikke er sårbare for angrep som *SQL injections*. Dermed bidrar også rammeverket til å øke sikkerheten i systemet.

3.2.3 OAuth 2.0

OAuth er et autentiserings-rammeverk som benytter seg av eksterne tredjeparter for innlogging og registrering. Dette ble benyttet for å verifisere at brukeren har en gyldig Google-bruker og at brukeren eksisterer i databasen. OAuth gir fordelen av at systemet ikke behøver å håndtere passord for brukeren fordi dette allerede er håndtert av en ekstern tredjepart.

3.2.4 JWT

I tillegg til Google OAuth 2 benyttet vi også JWT for å verifisere brukeren etter at brukeren har logget inn. JWT følger den åpne industri-standarden RFC 7519, og dermed anså teamet dette som en sikker måte å kunne autentisere brukere.

3.3 No-code rammeverk

3.3.1 Bubble

No-code rammeverk har blitt mer populært den siste tiden (2.5.4). Av den grunn finnes det også mange virksomheter som tilbyr ulike rammeverk. Bubble har et intuitivt brukergrensesnitt for å utvikle applikasjoner, og har også innebygd database-mulighet. Dermed er det mulig å utvikle et helt system ved hjelp av rammeverket. Videre finnes det flere *plugins* som gjør at man eksempelvis kan koble til eksterne API om det skulle være nødvendig. Ikke minst er det enkelt å designe et responsivt brukergrensesnitt. Av den grunn konkluderte teamet med at Bubble var et rammeverk som tilbydde nødvendig funksjonalitet for å kunne besvare problemstillingen.

3.4 Database

3.4.1 PostgreSQL

PostgreSQL er et gratis, open-source relasjonelt database-system [13]. Dette er et type system som har mange likheter med MySQL som teamet var kjent med tidligere. Oppgavestiller hadde i tillegg svært gode erfaringer med dette. Videre samhandler Blanks allerede eksisterende

internsystemer med en PostgreSQL-database, hvilket gjorde at det var naturlig å velge denne løsningen.

3.5 Prosess

3.5.1 Scrum

Tidlig i oppstarten ble teamet enige om å bruke en agil utviklingsmetode. Slik kunne kunden være kontinuerlig delaktig og være med på å ta avgjørelser underveis. I tillegg ville vi også kunne respondere raskt til endringer da det var svært sannsynlig at kravene til systemet ville forandres underveis.

Av alternativene for agile utviklingsmetoder fant vi blant annet Scrum og Kanban. Til slutt falt valget på en Scrum-inspirert utviklingsmetode. Med tanke på at vi også var et lite team på to stykker gjorde vi enkelte endringer på Scrum-metodikken for å tilpasse teamet bedre. I Vedlegg D Prosjekthåndbok kapittel 1 Prosess kan man lese mer om hvordan prosessen ble ivaretatt gjennom prosjektet.

3.5.2 ZenHub

Teamet bestemte seg for å bruke ZenHub som Scrum-tavle fordi den tilbyr god oversikt over prosessfasene, og har sømløs integrasjon med GitHub sin funksjonalitet som *issues* og *pull requests*. I tillegg har ZenHub verktøy for planlegging av sprinter. Videre kan man ved hjelp av estimater på oppgaver og milepæler følge progresjonen i sprinten.

3.6 Versjonskontroll

3.6.1 GitHub

Vi valgte GitHub hovedsakelig fordi oppdragsgiver ønsket at vi skulle bruke GitHub som distribusjonsplattform. I tillegg til dette så teamet nytteverdien av å benytte seg av flere av GitHubs funksjonaliteter. Blant annet tilbyr plattformen som *GitHub actions* for å implementere kontinuerlig integrasjon og utrulling. Man kan også benytte seg av *pull requests* for å kvalitetsikre teammedlemmenes kode. Til slutt integrerer GitHub godt med Zenhub (3.5.2).

4 Resultater

4.1 Vitenskapelige resultater

Dette kapittelet tar for seg de vitenskapelige resultatene teamet har kommet frem til gjennom dette prosjektet. Resultatene legges til grunn for diskusjonen rundt problemstillingen i kapittel 5.

4.1.1 Utviklet system

For å utforske fordelene og ulempene ved å benytte seg av et no-code rammeverk sammenlignet med tradisjonell utvikling valgte teamet å gjøre en kombinasjon av kvalitative og kvantitative undersøkelser. Først og fremst ble det utviklet et tilsvarende system som selve produktet ved hjelp av no-code rammeverket Bubble. Det har blitt foretatt en rekke justeringer sammenlignet med produktet grunnet både tid og begrensninger på rammeverket. Under følger en skjermutklipp fra systemet utviklet ved hjelp av Bubble, samt en kort forklaring på sidene.

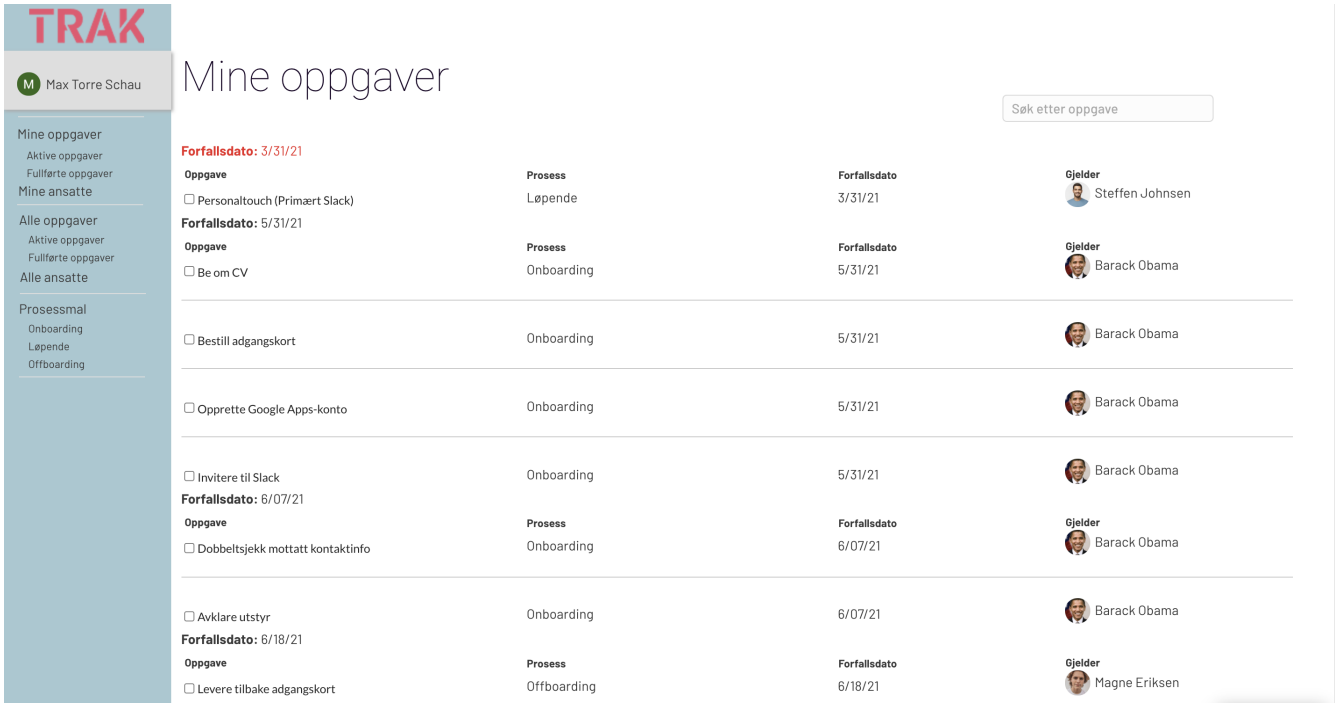
4.1.1.1 Innlogging

Innloggingen skjer ved hjelp av en Google-konto. Ved innlogging opprettes det en bruker i databasen. Så lenge en bruker har en Google-konto har man tilgang til systemet.

Figur 4: Bubble: Logg inn

4.1.1.2 Mine oppgaver

Mine oppgaver viser frem oppgaver som ikke er blitt fullført. Disse er gruppert etter datoen de er forfalt. Dersom en oppgave ikke har blitt gjort innen forfallsdatoen vil det bli merket med rød skrift for å tydeliggjøre at den har forfalt.



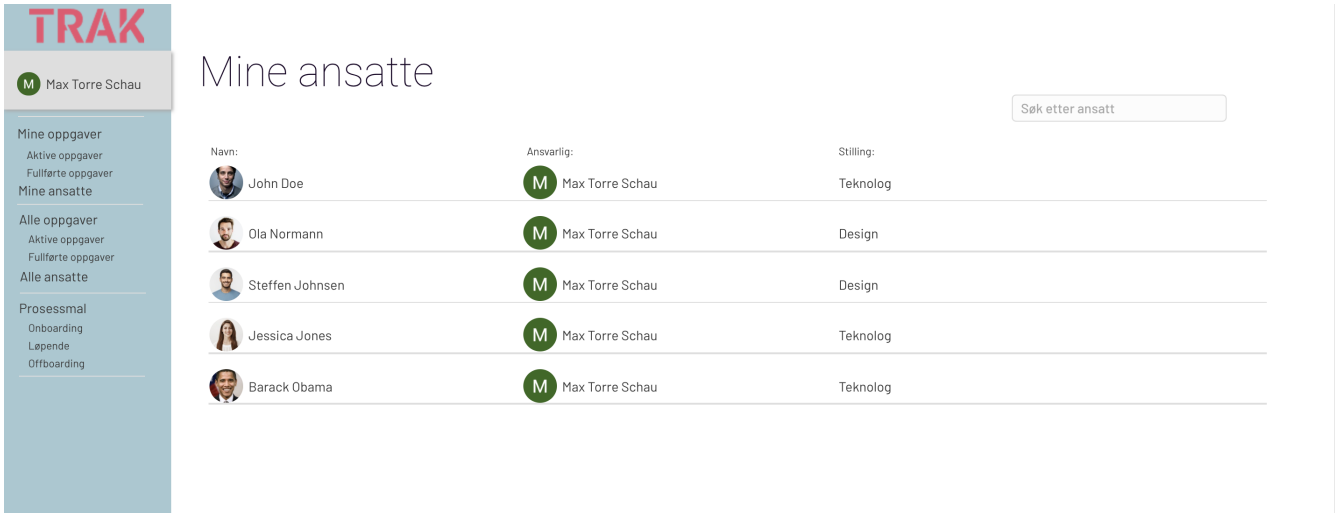
Figur 5: Bubble: Mine oppgaver

Det er også mulig å søke etter oppgaver for å raskt finne frem til spesifikke oppgaver.

Figur 6: Bubble: Søk i mine oppgaver

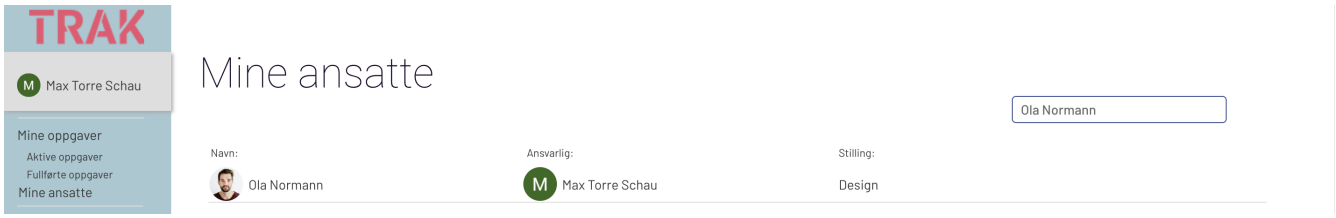
4.1.1.3 Mine ansatte

Mine ansatte viser frem en oversikt over alle ansatte som har den innloggede brukeren som personalansvarlig. Det skilles ikke mellom hvilken prosess den ansatte er i. Dette i kontrast til systemet utviklet med tradisjonell utvikling hvor man viser de ulike ansattene i sin respektive faser.



Figur 7: Bubble: Mine ansatte

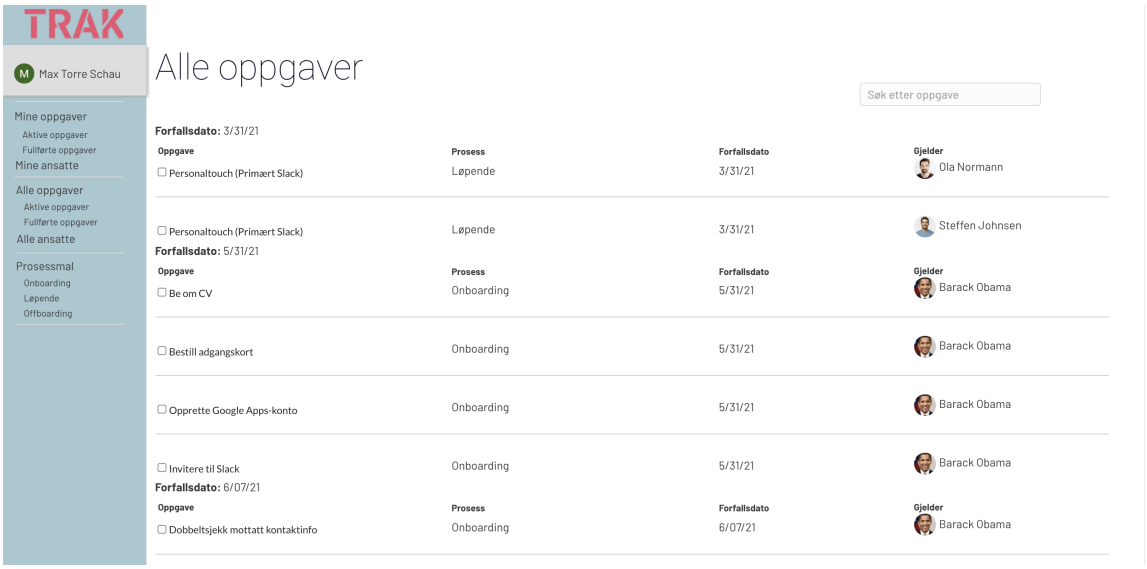
I likhet med mine oppgaver er det også mulig å søke etter en spesifikk ansatt.



Figur 8: Bubble: Søk i mine ansatte

4.1.1.4 Alle oppgaver

Alle oppgaver er tilsvarende som mine oppgaver, men det blir ikke filtrert på hvem som er den ansvarlige. Det vil si at alle aktive oppgaver i systemet vises på denne siden.



Figur 9: Bubble: Alle oppgaver

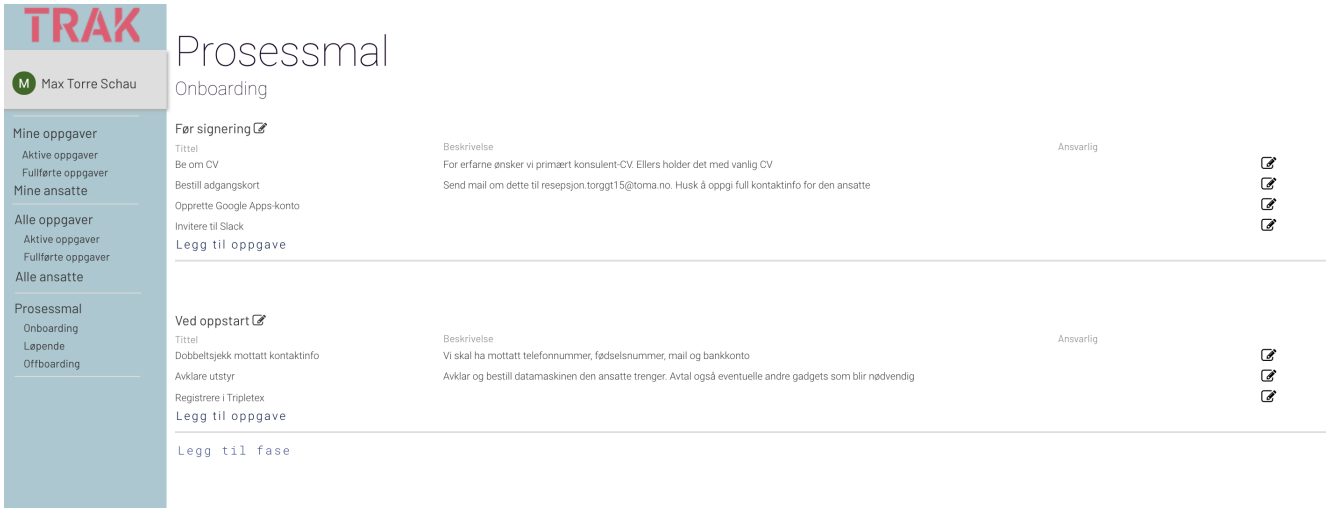
4.1.1.5 Alle ansatte

Tilsvarende som mine ansatte, men siden tar med alle ansatte uavhengig av personalansvarlig.

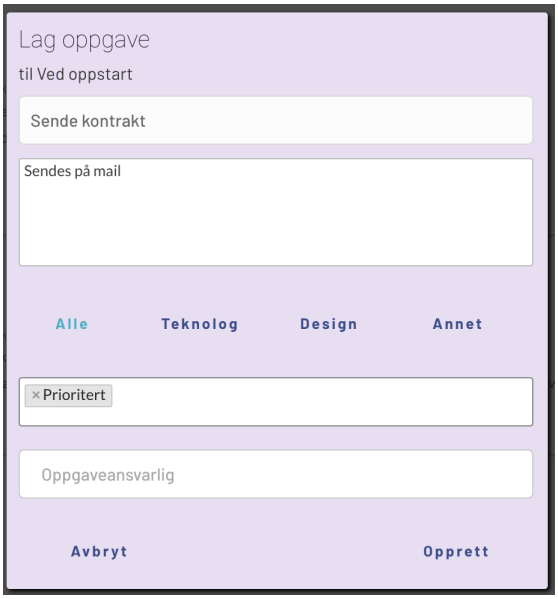
Figur 10: Bubble: Alle ansatte

4.1.1.6 Prosessmal

Prosessmalen inneholder alle oppgaver som blir opprettet for de ansatte. Siden gir en fullstendig oversikt over de ulike fasene med tilhørende oppgaver. Her er det mulig å opprette nye oppgaver og faser, samt endre eksisterende faser og oppgaver.



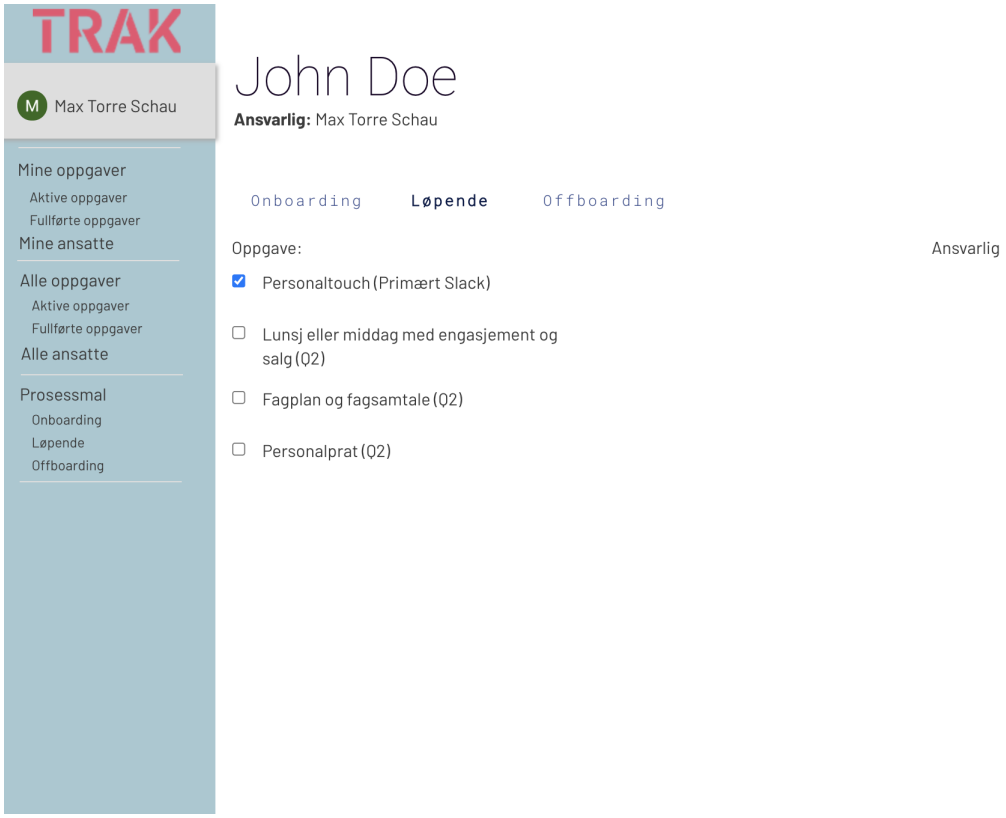
Figur 11: Bubble: Prosessmal onboarding



Figur 12: Bubble: Oppgave modal prosessmal

4.1.1.7 Ansatt

Ansattsiden viser frem de ulike oppgavene knyttet til prosessene for en ansatt. I motsetning til systemet utviklet med tradisjonell utvikling er denne blitt mer begrenset. Man grupperer ikke oppgavene etter hvilken fase de tilhører, men de blir alle vist frem på den aktuelle siden. I tillegg tilbyr ikke systemet historikk over tidligere gjennomførte faser.



Figur 13: Bubble: Ansatt-siden

4.1.2 Utviklingshastighet

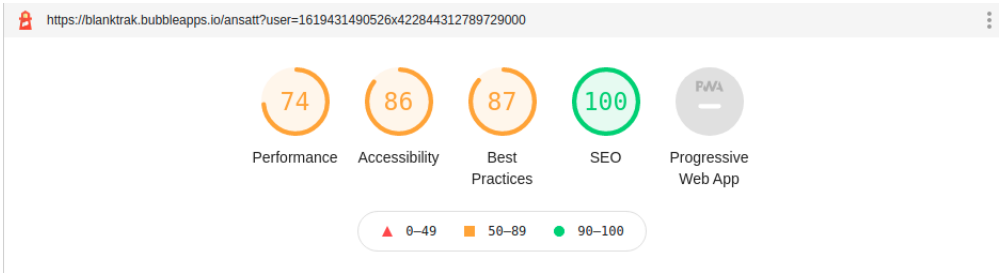
Gjennom prosjektet har teamet loggført hvor lang tid det er blitt benyttet på de ulike aktivitetene for å kunne svare på forskningsspørsmålet som omhandler utviklingshastighet. Tiden det har tatt for de ulike aktivitetene er målt i antall timer.

Aktivitet	No-code	Tradisjonell
Oppsett av prosjekt og deploy av system	0,2	4,00
Innlogging med Google (OAuth)	0,25	16,00
Mine oppgaver *	2,0	15,0
Ansatt-side *	2,5	15,0
Sidenavigering	0,75	2
Prosessmal	3	16
Mine ansatte *	2	35
Innstillinger	0,33	2,0
Søke etter ansatte	2,0	2,0
Brukbarhet (WCAG)	1,0	7,0

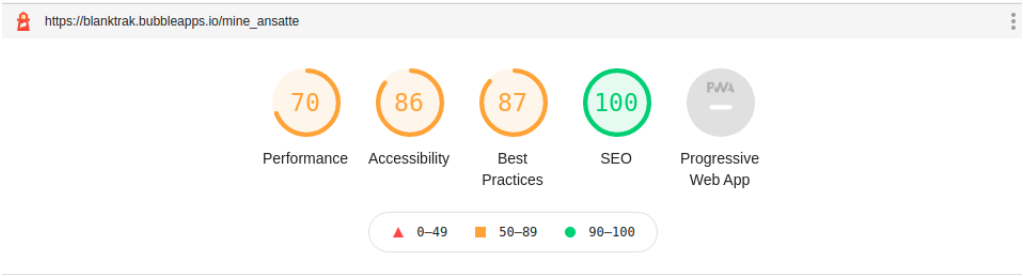
* Indikerer at aktivitetens funksjonalitet er begrenset sammenlignet med den tradisjonelle utviklingsmåten.

4.1.3 Brukbarhet

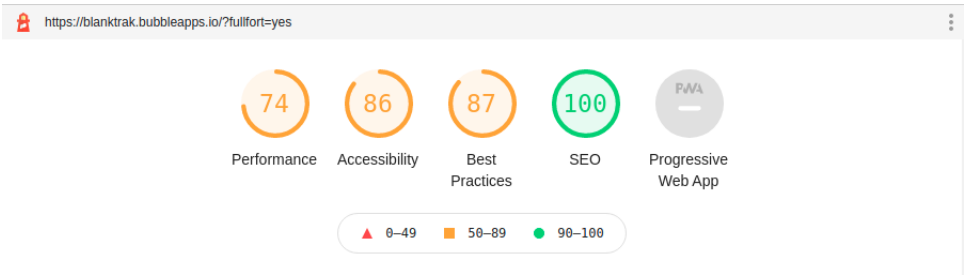
Under følger skjermutklipp fra Google Lighthouse-rapporten. Dette er en rapport som måler blant annet tilgjengeligheten til nettsiden. Det vil si at den tester systemet opp mot diverse kriterier fra WCAG 2.1.



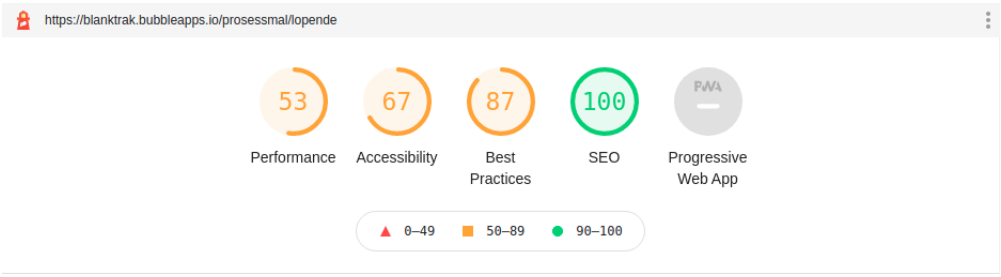
Figur 14: Lighthouse for ansattsiden



Figur 15: Lighthouse for mine ansatte



Figur 16: Lighthouse for mine oppgaver



Figur 17: Lighthouse for prosessmal

4.2 Ingeniørfaglige resultater

Dette kapittelet vil ta for seg de ingeniørfaglige resultatene gjennom prosjektet.

4.2.1 Funksjonelle krav

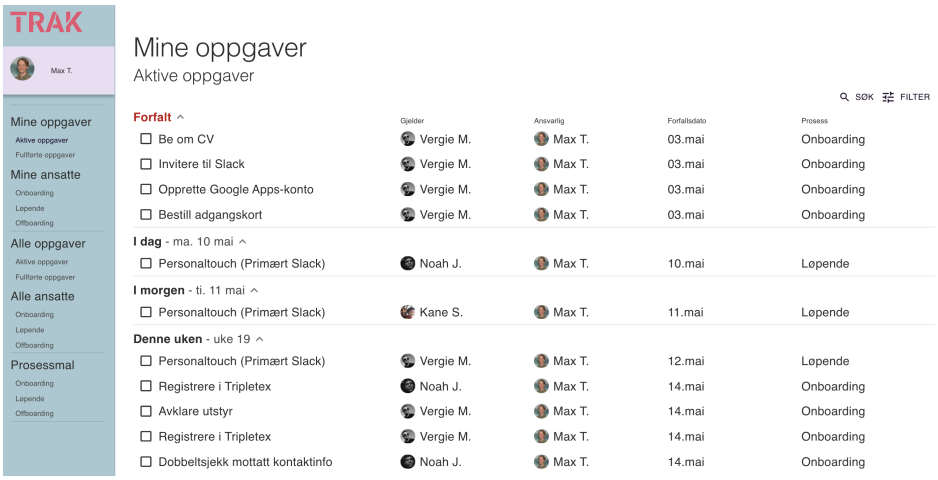
For de funksjonelle kravene blir det tatt utgangspunkt i brukerhistoriene fra vedlegg B Kravdokumentasjon kapittel 2, som igjen baseres på de funksjonelle kravene fra Vedlegg A Visjonsdokument kapittel 5.

4.2.1.1 Oversikt over mine oppgaver

En bruker har oversikt over sine kommende arbeidsoppgaver. Oppgavene er sortert og gruppert etter forfallsdato. Følgende grupperinger vil kunne vises:

- Forfalt
- I dag
- I morgen
- Denne uken
- Denne måneden
- Neste måned

Oversikten gir muligheten til å fullføre en oppgave, se en nærmere beskrivelse av oppgaven, hvem oppgaven gjelder, når oppgaven forfaller og hvilken prosess den tilhører.



Figur 18: Mine Oppgaver

Videre vil det være oppgaver som krever at man går via en ekstern link for å kunne fullføres. Det er derfor lagt inn støtte for hurtiglinker for både nettsteder og epost for de ulike oppgavene. Dette kan være hensiktsmessig dersom en oppgave innebærer å sende en mail eller registrere noe på en ekstern nettside.

Figur 19: Ekstern link for oppgave

4.2.1.2 Oversikt over alle oppgaver

En bruker har også oversikt over alle ansattes arbeidsoppgaver, sortert på forfallsdato. I tillegg til all informasjon som er tilgjengelig på *Mine oppgaver* er det også mulig å se hvem som er satt opp til å utføre oppgaven.

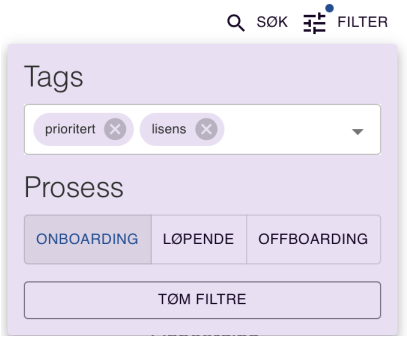
4.2.1.3 Fullførte oppgaver

Det kan være ønskelig å se på oppgaver som har blitt fullført. Det er derfor blitt implementert en oversikt over fullførte oppgaver. Det vil kun vises oppgaver som ikke har forfalt enda ettersom det ble bestemt at forfalte, fullførte oppgaver ikke er av relevans for personalansvarlige.

For enhver oppgave som fullføres blir det lagret hvem som fullførte oppgaven, og når den ble fullført for å gi de ansatte full oversikt.

4.2.1.4 Filtrere oppgaver

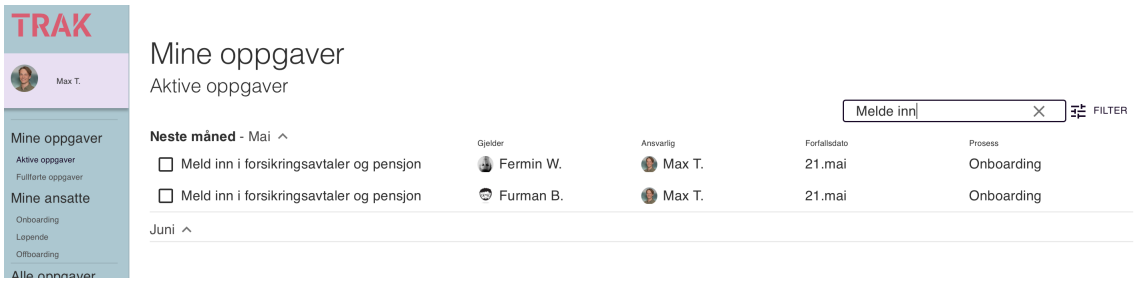
For å kunne filtrere bort oppgaver man ikke behøver å se er det mulig å velge ulike parametere å filtrere på. Ved flere valgte stikkord vil systemet filtrere på de som har minst en av stikkordene.



Figur 20: Filtrering av oppgaver

4.2.1.5 Søke etter oppgaver

I oppgaveoversikten har man også mulighet til å søke etter tittel på oppgaven, hvem den gjelder og hvem som er ansvarlig. Søkefunksjonen bruker såkalt tilnærmet streng-søking (fuzzy-search). Det er en teknikk som sørger for at søkeordet ikke må være korrekt bokstav for bokstav for å få opp riktig søkeresultat.



Figur 21: Søke etter oppgaver

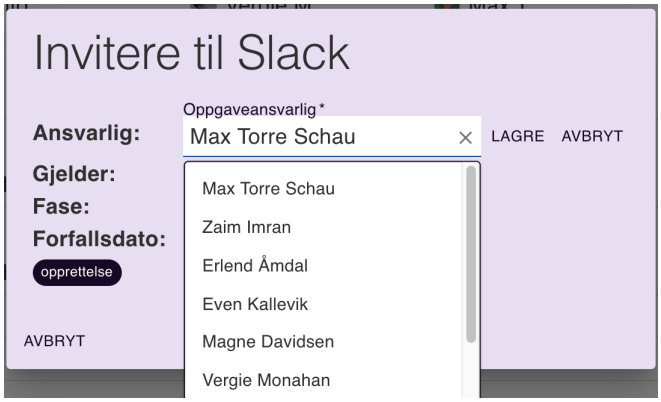
4.2.1.6 Markere oppgaver

En personalansvarlig kan markere oppgaver som fullført ved å trykke på avkrysningsboksen ved en oppgave. Da vil boksen krysses av og oppgaveteksten blir også markert som fullført.

Figur 22: Markere oppgave som fullført

4.2.1.7 Delegere oppgaver

Dersom en personalansvarlig ikke har anledning til å gjøre en oppgave kan vedkommende delegere bort arbeidsoppgaven til andre ansatte. Dette gjøres i informasjonsmodalen for hver oppgave. Ved eventuell delegering blir oppgaven flyttet til oppgaveoversikten til den aktuelle ansatte.



Figur 23: Delegere oppgaver

4.2.1.8 Opprettelse av Onboardings-kort

For å opprette onboardings-oppgaver ved nyansettelse kjøres det en cron-jobb daglig. Denne cron-jobben sjekker alle ansatte og ser om de har fått en ansettelsesdato og om de har eksisterende onboardingsoppgaver. Hvis de ikke har noen onboardings-oppgaver vil cron-jobben opprette oppgavene i onboardingsfasen for den aktuelle ansatte.

Det har blitt foretatt manuelle simuleringer som står nærmere beskrevet i vedlegg E Simulering av opprettelse av oppgaver kapittel 1.3.

4.2.1.9 Opprettelse av Løpende-kort

For løpende oppgaver sjekker cron-jobben den neste fasen i den løpende prosessmalen. Det vil si at hvis vi er i fase 1 vil den sjekke for fase 2. Her vil det sjekkes om enhver ansatt har oppgaver i den kommende fasen. Hvis dette ikke er tilfellet vil det opprettes oppgaver for dette.

Det har blitt foretatt manuelle simuleringer som står nærmere beskrevet i vedlegg E Simulering av opprettelse av oppgaver kapittel 1.2.

4.2.1.10 Opprettelse av Offboarding-kort

Oppgavene blir opprettet på samme måte som onboardings-oppgavene, med unntak av at det baseres på termineringsdato i stedet for ansettelsesdato.

Det har blitt foretatt manuelle simuleringer som står nærmere beskrevet i vedlegg E Simulering av opprettelse av oppgaver kapittel 1.4.

4.2.1.11 Se prosessmaler

En bruker kan se alle prosessmaler i systemet. For hver prosessmal kan man også se hvilke oppgaver som tilhører de ulike fasene. På den måten kan man enkelt få en fullstendig oversikt over de ulike prosessene, og hva som skal gjøres.

4.2.1.12 Oppdatere oppgaver

En bruker kan både opprette, endre og slette oppgaver. Her blir det benyttet en modal med ulike felter som må fylles ut. For felter som behøver mer forklaring har det blitt lagt til en *tooltip* for å gi brukeren mer informasjon om hva dette feltet faktisk er.

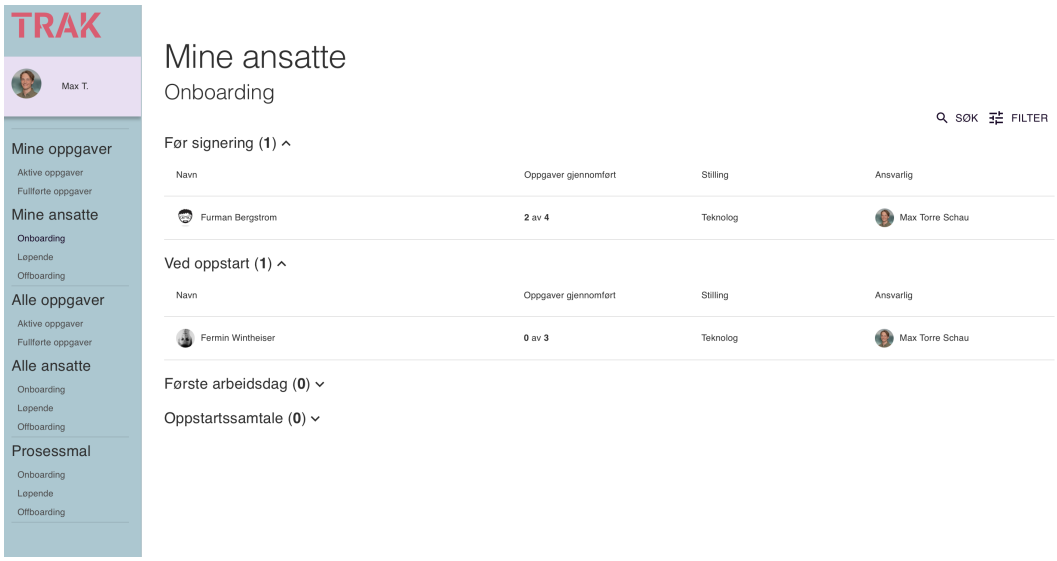
4.2.1.13 Oppdatere faser

Det er mulig å opprette, redigere og slette faser i systemet. Man skiller mellom løpende og offboarding/onboarding da løpende faser må sette en forfallsdato uavhengig av år. Når det gjelder offboarding og onboarding vil man måtte sette en forfallsdato som baserer seg på henholdvis termineringsdato og ansettelsesdato. Årsaken er at forfallsdatoen må være dynamisk ettersom den blir forskjellig fra ansatt til ansatt.

4.2.1.14 Oversikt over mine ansatte

En bruker har mulighet til å se alle sine ansatte. Denne oversikten viser ansatte i de ulike prosessene samt grupperer de i hvilken fase de er i. Her kan man også få en oversikt over hvor mange oppgaver som er fullført i de ulike fasene. En ansatt vil kun være i en fase av gangen slik at man kun vil vises i fasen som er aktuell. Dette gir personalansvarlig en rask og enkel oversikt over status på de ulike prosessene.

Her er det også mulig å filtrere etter ansatte. Det er kun mulig å filtrere etter hva slags rolle den ansatte har i bedriften.



Figur 24: Mine Ansatte

4.2.1.15 Oversikt over alle ansatte

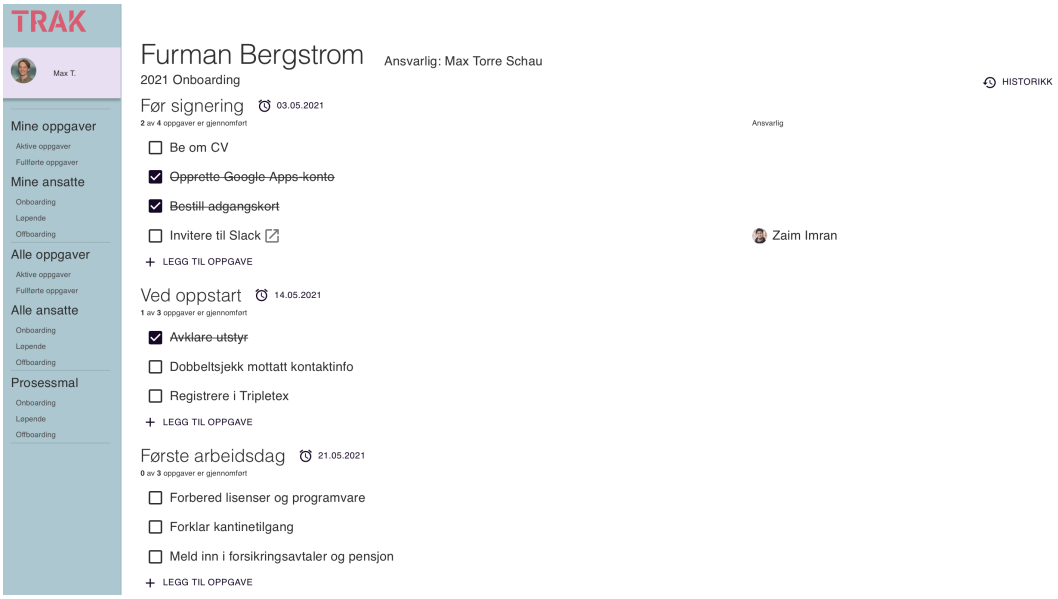
En bruker har mulighet til å se alle ansatte. Denne oversikten er helt lik Oversikt over mine ansatte, men det blir ikke filtrert på hvem den ansatte har som personalansvarlig.

4.2.1.16 Søke etter ansatte

På samme måte som man i oppgave-oversikten kunne søke etter oppgaver kan man også søke etter ansatte. Her benyttes det også tilnærmet streng-søking.

4.2.1.17 Oppgaver tilknyttet ansatt

Gjennom ansatt-siden har man full oversikt over oppgavene tilhørende hver prosess. Dette gir personalansvarlig status på den aktuelle ansatte.



Figur 25: Ansatt side

4.2.1.18 Ansatt-historikk

I tillegg til å se aktuelle prosesser kan man også gå tilbake i tid for å se tidligere gjennomførte prosesser.

4.2.1.19 Legge til individuelle oppgaver

Fra prosessmalen medfølger det et sett med oppgaver som gjelder for alle ansatte. For å gi personalansvarlig mer frihet kan man også legge til individuelle oppgaver som kun gjelder for den enkelte ansatte. Dette gjøres ved å trykke på "Legg til individuelle oppgaver" på ansatt-siden.

4.2.1.20 Endre forfallsdato

Selv om forfallsdatoen blir satt for alle faser vil det også være mulig å endre denne for en enkelt ansatt. Dermed vil alle oppgaver i den aktuelle fasen få endret forfallsdato.

4.2.1.21 Filopplastning / notering

Dette ble aldri implementert.

4.2.1.22 Varslinger

Systemet har en innebygd varslingssystem som gir beskjed til brukeren dersom enkelte hendelser skulle inntreffe. Varslingene kan komme i både selve systemet, men også på Slack dersom brukeren har godkjent dette.

Det vil bli opprettet varslinger daglig for å informere om en ansatt har oppgaver som forfaller i nærmeste fremtid. Se kapittel 4.2.1.23 for fullstendig oversikt.

Figur 26: Varslinger

4.2.1.23 Bestemme hvilke varslinger

En bruker kan også bestemme hvilke typer varslinger vedkommende ønsker å motta, samt om de skal bli sendt på Slack og/eller i systemet. Det er mulig å få varslinger når:

- Man blir delegert en arbeidsoppgave
- En fase utløper
- En fase utløper om en uke
- En ansatt skal slutte, og man får oppgaver i offboarding
- En ansatt skal starte, og man får oppgaver i onboarding

4.2.1.24 Logg inn

Dersom en ansatt ligger lagret i internsystemet kan vedkommende benytte seg av Google-innlogging til å logge inn og bruke systemet. Om man ikke ligger lagret i internsystemet vil man heller ikke ha adgang til siden.

4.2.1.25 Innstillinger

Som nevnt i Varslinger (4.2.1.22) kan man endre hvilke typer notifikasjoner man ønsker å få. I tillegg kan man velge om man vil motta notifikasjonene på Slack.

4.2.1.26 Vanlig ansatt

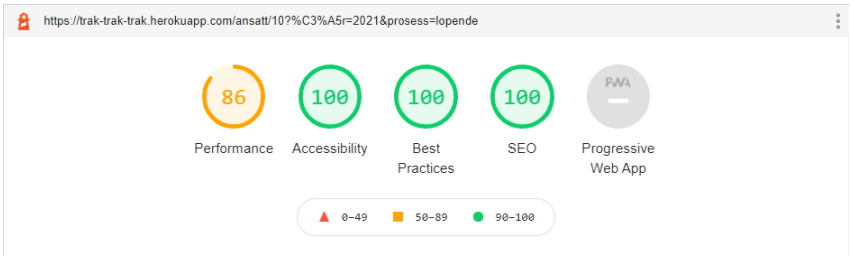
Dette ble aldri implementert.

4.2.2 Ikke-funksjonelle krav

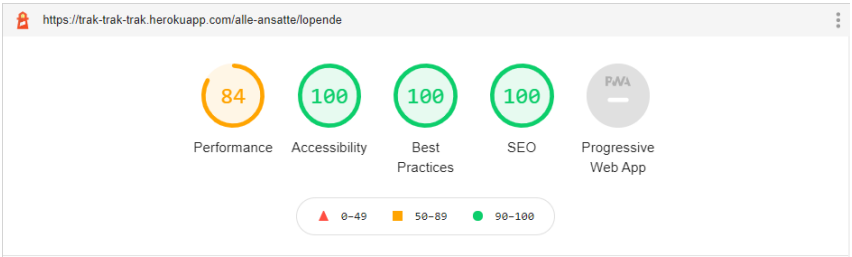
I Vedlegg A Visjonsdokument kapittel 6 ble det definert en rekke ikke-funksjonelle krav som systemet måtte dekke. Resultatene for de gitte kravene vil bli gjennomgått i dette kapittelet.

4.2.2.1 Brukbarhet

I henhold til kapittel 6.1.1 fra Vedlegg A visjonsdokument skulle systemet innfri de viktigste kriteriene fra WCAG 2.1 for å sikre at systemet i størst mulig grad er universelt utformet (2.3.2). Teamet har fokusert på dette ved å gjennomføre manuelle tester gjennom hele utviklingsprosessen. Det vil si at det blant annet har blitt testet for at det er mulig å bruke systemet med kun tastaturet, ved bruk av skjermleser, ha gode nok kontraster osv. I tillegg har rammeverkene WAVE Evaluation Tool og Google Lighthouse blitt benyttet for å bekrefte at systemet ikke inneholder konvensjonelle feil knyttet til WCAG 2.1. Det innebærer at rapportene sjekker at for eksempel alle bilder har en alternativ tekst, at knapper har en aria-label[3] som beskriver knappen osv. Under kan man se skjermbilder fra rapportene generert av Google Lighthouse:

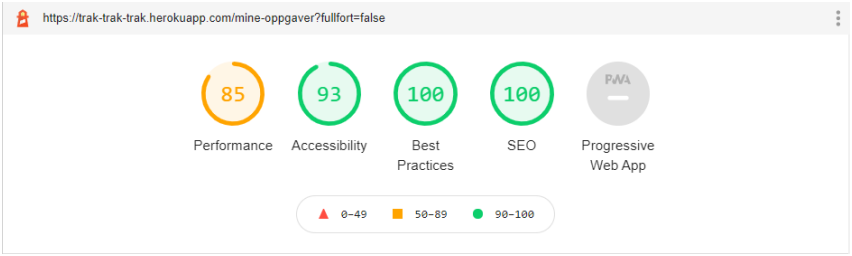


Figur 27: Lighthouse for ansattsiden

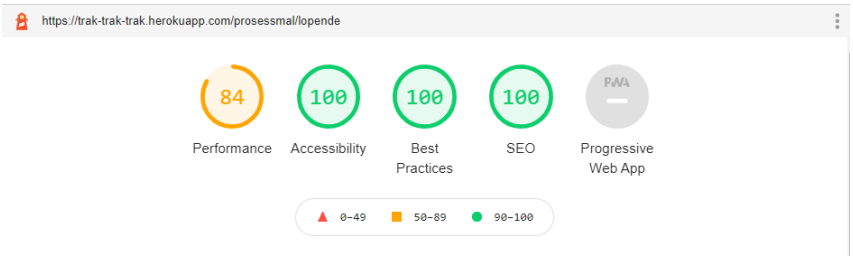


Figur 28: Lighthouse for mine ansatte

[3]En tekst som merker et HTML-element. Benyttes av skjermlesere



Figur 29: Lighthouse for mine oppgaver



Figur 30: Lighthouse for prosessmal

Systemet skulle være intuitivt i bruk slik at personalansvarlige enkelt kan benytte seg av systemet. Det er blitt foretatt brukertester underveis for å sikre nettopp dette. Hvordan brukertestene er gjennomført og notatene fra disse kan finnes i vedlegg F Brukertester.

Videre skulle systemet fungere på Google Chrome, Safari og Mozilla Firefox. Dette er også kjørt manuelle tester for å verifisere at systemet fungerer som det skal på de ulike nettleserne.

Til slutt har systemet blitt utviklet slik at det er brukbart på mobil. Det vil si at det fungerer på mobiler, men det er anbefalt å bruke dette på en datamaskin eller nettbrett.

4.2.2.2 Sikkerhet

Sikkerheten er godt ivaretatt, og har tatt utgangspunkt i OWASP Top Ten fra 2017. Dette er skrevet mer om i Vedlegg C Systemdokumentasjon kapittel 7.

4.2.2.3 Testing

Alle endepunktene er godt testet for et robust API. Mer om testingen kan leses i Vedlegg C Systemdokumentasjon kapittel 10.2.

4.2.2.4 Dokumentasjon

Kodebasen er dokumentert med TypeDoc. Mer om dokumentasjonen kan leses i Vedlegg C Systemdokumentasjon kapittel 9.

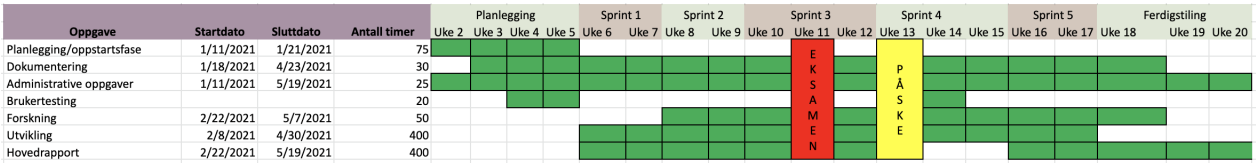
4.2.2.5 Vedlikehold

Prosjektet er utviklet med konvensjonell prosjektstruktur for Next.js og all kode er godt dokumentert. Koden er splittet opp i gjenbrukbare komponenter og teamet har brukt mye tid på å passe på at kodekvaliteten er god.

4.3 Administrative resultater

Dette delkapittelet skal ta for seg de ulike aspektene ved utvikling under prosjektet.

4.3.0.1 Fremdriftsplan



Figur 31: Fremdriftsplan

Millepæl:	Frist	Forklaring
Visjonsdokument	Uke 4	Ferdigstille visjonsdokument slik at det er klart for innlevering
Wireframes	Uke 5	Wireframes skal være ferdig, og klare for brukertesting
Brukertesting på wireframes	Uke 5	Utføre brukertester, og gjøre nødvendige endringer basert på tilbakemeldingene
Kravdokumentasjon	Uke 7	Skrive ferdig kravdokumentasjon
Problemstilling	Uke 7	Problemstillingen skal være ferdig formulert, og godkjent av veileder
Prototype	Uke 12	
Brukertesting av prototype	Uke 13	Ferdigstille brukertesting på prototype
Systemtester	Uke 17	
Ferdig produkt	Uke 17	Produktet skal være helt ferdig og klar til å benyttes av Blank
Innlevering prosjekt	Uke 20	Ferdigstille hovedrapport og alle vedlegg

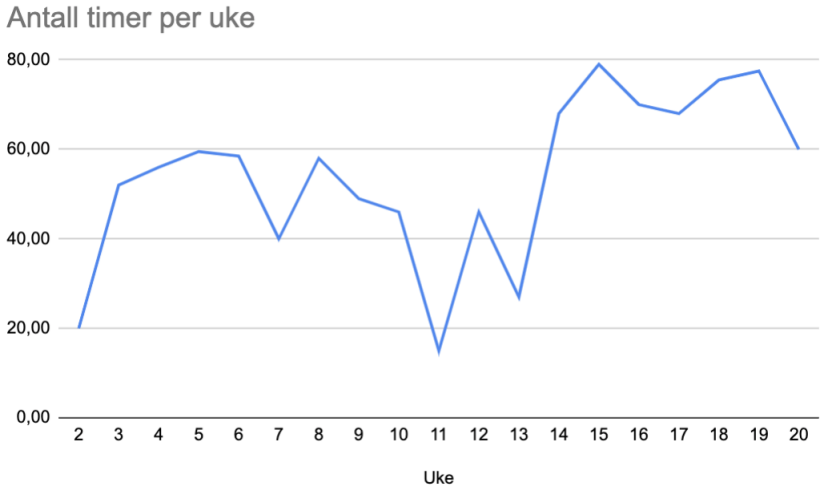
Figur 32: Milepæler

4.3.0.2 Timeforbruk

Alle timer som er ført på timelistene er fordelt på oppgavene som står oppført i figur 33, med unntak av totalt 22 timer som er brukt til obligatorisk skole i løpet av perioden. En fullstendig oversikt over timer i løpet av prosjektet kan ses i Vedlegg D Prosjekthåndbok kapittel 3 Timelister med statusrapporter.

Aktivitet	Planlagte timer	Faktiske timer	Differanse
Planlegging	75	142,50	67,50
Dokumentering	30	26,00	−4,00
Utvikling	400	406,50	6,50
Brukertesting	20	12,00	−8,00
Forskning	50	65,50	15,50
Administrative oppgaver	25	7,00	−18,00
Hovedrapport	400	343,50	−56,50
Totalt	1000	1 025,00	25,00

Figur 33: Oversikt over timeforbruk



Figur 34: Timer per uke

4.3.0.3 Scrum

Vedlegg D Prosjekthåndbok kapittel 1 Prosess har dokumentasjon av alle sprintene. Alle sprintene har blitt dokumentert med sprintmål, sprint backlog, burndown-chart og sprint retrospektiv.

5 Diskusjon

5.1 Vitenskapelige resultater

Som nevnt i kapittel 1.2 lyder problemstilling som følgende:

"Hvilke fordeler og ulemper er det ved å bruke et no-code rammeverk sammenlignet med tradisjonell utvikling"

For å konkretisere dette ned i mindre deler har teamet valgt å undersøke følgende forskningsspørsmål:

1. Hvilke begrensninger og muligheter medfører no-code rammeverk for det visuelle og funksjonelle, sammenlignet med tradisjonell utvikling?
2. Vil det være gunstig å benytte et no-code rammeverk med tanke på utviklingshastighet?
3. I hvilken grad vil sikkerheten bli ivaretatt for et no-code rammeverk sammenlignet med tradisjonell utvikling?
4. Kan man med et no-code rammeverk oppnå samme tilgjengelighet som man kan oppnå med tradisjonell utvikling?

I dette kapittelet vil vi drøfte problemstillingen ved å ta utgangspunkt i kapittel 4.1 og deler av kapittel 4.2

5.1.1 Hvilke begrensninger og muligheter medfører no-code rammeverk for det visuelle og funksjonelle, sammenlignet med tradisjonell utvikling?

For å få en bedre forståelse av de to systemene har teamet laget en videodemonstrasjon som viser systemene side om side. Slik kan man få en forståelse av hvordan det visuelle og funksjonelle er løst i begge systemer, samt hvilke begrensninger systemet har. Videoen kan ses i vedlegg G Videodemonstrasjon av systemene.

For å sammenligne begrensningene og mulighetene man har med et no-code rammeverk sammenlignet med tradisjonell utvikling vil vi først undersøke det visuelle aspektet. Basert på resultatene fra kapittel 4.1 og 4.2 kan man se at det ikke er mye som skiller de to versjonene rent visuelt. I tillegg gir no-code rammeverk god støtte for å utvikle responsive løsninger slik at systemet er tilpasset alle plattformer. Til svyende og sist er begge systemene bygget opp ved hjelp av HTML og CSS, slik at man kan utvikle nærmest identiske løsninger.

Angående det funksjonelle, er det klare likhetstrekk mellom flere av sidene. Samtidig finnes visse begrensninger på enkelte. Grunnfunksjonaliteten til begge systemene er identiske; en bruker kan logge inn med Google-brukeren sin, opprette og endre faser, opprette og endre oppgaver. I tillegg har hver bruker mulighet til å fullføre både sine og andres oppgaver. Teamet erfarte at det var utfordrende å forstå hvordan rammeverket fungerte i startfasen, men at arbeidet ble svært effektivt så fort man hadde forstått hvordan ting fungerte. Deretter var det relativt simpelt å implementere flere av sidene så lenge funksjonaliteten ikke var særlig kompleks.

En begrensning som no-code rammeverk har, er at det er svært utfordrende å gjøre komplekse spørringer og filtreringer som også er vanskelig med tradisjonell utvikling. For eksempel løses automatisk opprettelse av oppgaver med den tradisjonelle utviklingen med cron-jobber. Cron-jobbene blir kjørt ved hjelp av en ekstern planlegger. Med Bubble var det mulig å planlegge oppgaver som skulle bli kjørt, men dette var funksjonalitet som man fikk mot betaling. Samtidig var det mange kompliserte sjekker og spørringer som måtte gjøres, men det er ikke gitt at dette

er mulig med Bubble. I beste fall er det ingen enkel og intuitiv måte å gjøre det på, slik at vi ikke med sikkerhet kan si at det faktisk er mulig.

For tradisjonell utvikling har vi erfart at det i langt større grad er mulig å implementere komplekse løsninger. Her har man større frihet for hvordan man velger å løse problemstillinger. I et no-code rammeverk blir man derimot låst og begrenset til funksjonaliteten rammeverket tilbyr. Dette gjør at man kan oppleve at det er vanskelig å utvikle funksjonalitet som man gjerne skulle ha implementert. Slik ble det også i vårt tilfellet hvor vi måtte nedjustere kompleksiteten til blant annet ansatt-, mine-ansatte- og mine-oppgaver-siden.

5.1.2 Vil det være gunstig å benytte et no-code rammeverk med tanke på utviklingshastighet?

Fra kapittel 4.1.2 kan man se de loggførte timene for de ulike aktivitetene i prosjektet. Det er verdt å merke seg at enkelte av aktivitetene for no-code er merket som begrenset sammenlignet med den tradisjonelle, slik det blir diskutert i 5.1.1. Til tross for dette anser vi det som et godt estimat for differansen i utviklingshastigheten mellom no-code og tradisjonell utvikling.

Å implementere innlogging ved hjelp av OAuth er noe som har blitt stadig mer utbredt. Flere no-code rammeverk har fått implementert støtte for dette, og med Bubble tok dette ikke mer enn 15 minutter å sette opp. Dette var noe som fulgte med ut av boksen, og var svært enkelt og intuitivt å implementere. Ved å trykke på påloggingsknappen ble nødvendig informasjon aksessert fra Google-brukeren, og brukeren ble så opprettet i databasen. På den andre siden tok dette opp imot 16 timer å implementere ved hjelp av tradisjonell utvikling.

Man ser en klar trend i at det går fort å utvikle noe med et no-code rammeverk. For eksempel ble selve prosjektet opprettet og rullet ut på rundt tolv minutter. Dette i kontrast til tradisjonell programvareutvikling, hvor det ble brukt hele fire timer. Dette kan implisere en av styrkene med et no-code rammeverk om at enkelte funksjoner følger med ut av boksen. På den måten trenger man ikke benytte seg av en ekstern server for å rulle ut nettsiden da dette allerede følger med. Dermed er det heller ikke behov får sette opp kontinuerlig utrulling eller lignende, ettersom dette allerede er implementert i rammeverket.

Som et resultat av hvordan rammeverket er bygget opp faller også nødvendigheten for versjonskontroll og oppsett av ulike miljøer (utvikling, produksjon osv.) bort. På mange måter understreker dette hvor tidsbesparende det kan være å benytte seg av et no-code rammeverk.

5.1.3 I hvilken grad vil sikkerheten bli ivaretatt for et no-code rammeverk sammenlignet med tradisjonell utvikling?

Fordelen ved å bruke et no-code rammeverk er at sikkerhet i og rundt applikasjonen er håndtert for deg. Bubble bruker SSL og Cloudflare for sikker kommunikasjon over HTTP og de nyeste serverene fra AWS for hosting. Dermed er tilkoblingen til systemet sikker og kryptert, slik at data delt med systemet er ivaretatt på en sikker måte. I tillegg har brukeren mulighet til å sikre data i databasen ved å gjøre den privat, slik at brukeren kan begrense hvem som har tilgang til hvilken data. Dette kan være nyttig når man eksempelvis lager en blogg der alle kan se bloggen og kommentarene, men kun forfatteren kan se hvor mange som har sett bloggen. Rammeverket har også innebygd sesjonshåndtering og kryptering av passord som legger på lag med sikkerhet.

Bubble håndterer som en standard *cross-site scripting*. I tillegg blir injeksjoner håndtert ved å *escape* input fra brukeren. Videre kan utvikleren velge hvilke brukere som skal ha tilgang til

de ulike sidene. I så måte kan rammeverket dekke de viktigste kravene til OWASP uten ekstra arbeid fra brukeren.

På den andre siden har man tradisjonell utvikling hvor man er nødt til å bestemme selv hvordan sikkerheten skal bli implementert. Her kan man velge å implementere sikkerheten på egenhånd, men dette anses som svært risikabelt da det ofte fører til sikkerhetshull. Eksempelvis kan man implementere den feil, eller i verste tilfelle ikke implementere sikkerheten i det hele tatt. Eventuelt kan man velge å ta i bruk allerede eksisterende løsninger som anses som sikre. Man må uansett ta stilling til hvordan man skal løse det, og implementere en løsning i systemet.

Det ble besluttet å benytte allerede eksisterende biblioteker for å ivareta sikkerheten for systemet. Teamet brukte lang tid på å implementere dette, og teste at det var tilstrekkelig. Til tross for tidsbruken har teamet fått utviklet et system med god nok sikkerhet uten store utfordringer.

5.1.4 Kan man med et no-code rammeverk oppnå samme tilgjengelighet som man kan oppnå med tradisjonell utvikling?

Figurene 14 til 17 viser Google Lighthouse-karakterene for systemet utviklet med no-code rammeverket, mens figurene 27 til 30 viser karakterene til systemet utviklet med tradisjonell utvikling. Det er tydelig at Bubble får en merkbart lavere karakter. Det impliserer at rammeverket ikke tilbyr tilstrekkelig tilgjengelighet som en standard. Fra kapittel 4.1.2 kan man observere at det er brukt minimalt med tid på tilgjengeligheten i no-code rammeverket. Dette kommer av at Bubble ikke tilbyr mulighet til å kunne tilpasse tilgjengeligheten til systemet. For eksempel er det ikke mulig å sette tab-indeksering[4] på de ulike elementene. Dette fører til at det er svært utfordrende å ikke ha tilgang på datamus når man benytter seg av systemet. Videre er det heller ikke mulighet for å sette *aria-labels*. Dette gjør at en bruker med synshemninger vil ha problemer med å benytte seg av systemet, ettersom skjermleseren ikke kan tolke de ulike elementene på en god måte. I kombinasjon med manglende mulighet til å benytte tastaturet for å navigere kan det oppstå misforståelser og forvirring.

I 4.1.2 ser man at det er brukt lang tid på å sikre tilgjengelighet og brukbarhet i systemet utviklet med tradisjonell utvikling. Teamet gikk gjennom systemet med både skjermleser og ved å kun benytte tastatur. I tillegg til dette ble det anvendt ulike verktøy for å verifisere at systemet oppfyller WCAG-kravene. På den måten kunne man tilpasse koden underveis slik at kriteriene ble oppfylt. Dette er en klar fordel i favør tradisjonell utvikling, da man har mulighet til å direkte endre HTML-elementene i koden - dersom det skulle være mangler i henhold til eksempelvis WCAG.

5.2 Ingeniørfaglige resultater

5.2.1 Funksjonelle krav

5.2.1.1 Oversikt over mine oppgaver

Kravet om å få en enkel oversikt over kommende oppgaver anses som innfridd. Brukeren får en enkel oversikt over hvilke oppgaver som må gjøres, og kan også få ekstra informasjon om oppgaven dersom det er nødvendig.

Oppgavene er sortert på forfallsdato, men det er ikke mulig å velge andre attributter å sortere på. Dette er noe som kan implementeres i senere versjoner.

[4] En attributt som indikerer at et element kan fokuseres ved hjelp av blant annet tastatur [19]

Med hensyn til akseptansekriteriene er disse også oppfylt. Oppgaver som er forfalt vil kategoriseres i en egen gruppe, og merkes med en tydelig rød skrift. Videre vil man kun vise oppgaver tre måneder frem i tid da det ikke er relevant å se lenger frem enn det. Dermed er også dette akseptansekriteriet innfridd.

5.2.1.2 Oversikt over alle oppgaver

Dette kravet regnes som innfridd da de samme kravene som i 5.2.1.1 gjelder her.

5.2.1.3 Fullførte oppgaver

En bruker kan navigere seg til fullførte oppgaver ved å benytte seg av sidenavigeringen. Det har kommet tydelig frem i brukertestene (Vedlegg F Brukertesting) at dette er intuitivt. Derfor kan kravet anses som innfridd.

5.2.1.4 Filtrere oppgaver

I filtrering kan man filtrere på stikkord og/eller prosesser. Fordelen med å kunne filtrere på stikkord er at det gir stor frihet til brukerne av systemet. Eksempelvis kan man velge å lage et stikkord som heter "prioritert", og dermed si at alle oppgaver med dette stikkordet er prioriterte oppgaver. Som følge av det slipper man å lage egne attributter i databasen, og lar det heller være opp til brukeren å gjøre dette innad i systemet.

Hvorvidt det burde være mulig å filtrere på mer enn det som er implementert er vanskelig å si, men det har ikke kommet frem i brukertestene at det har vært behov for dette. Dermed anses dette kravet som innfridd.

5.2.1.5 Søke etter oppgaver

I tillegg til å kunne filtrere kan en bruker også søke etter oppgaver. Søkingen er blitt implementert med tilnærmet string samsvar, også kalt "fuzzy-search". Styrken med dette er naturligvis at brukeren ikke må ha kjennskap til det konkrete ordet som skal søkes etter. Dette er også med på å øke brukertilfredsheten.

Ulempen er at ytelsen minker ved bruk av en slik algoritme da det er mer arbeid som må gjøres sammenlignet med normal søking. Denne reduseringen av ytelse er såpass minimal at teamet konkluderer med at implementeringen gir flere fordeler enn ulemper. Kravet er derfor innfridd.

5.2.1.6 Markere oppgaver

Ved siden av enhver oppgave er det en avkrysningsboks. For å markere at en oppgave er fullført må denne trykkes på. Brukeren vil få flere tegn på at oppgaven er markert som fullført; avkrysningsboksen blir krysset av, oppgaveteksten blir streket over og man vil få presentert en grønn tilbakemelding om at alt gikk bra. Derfor kan vi si at akseptansekriteriet om at brukeren skal få tydelig beskjed om at oppgaven er markert er innfridd.

Ikke minst har også brukertestene indikert dette er svært intuitivt. Dermed er kravet innfridd.

5.2.1.7 Delegere arbeidsoppgaver

Å delegere bort en arbeidsoppgave er i følge brukertestene svært intuitivt. I tillegg er også akseptansekriteriet om at personen som oppgaven blir delegert til skal motta varslinger i systemet

og/eller på Slack innfridd.

En svakhet med måten dette er løst på er mangelen på paginering. Per nå vil alle ansatte bli hentet ut, og det kan bli et ytelsesproblem ved mange ansatte. For å løse problemet burde man hente ut et begrenset antall. Ved bruk av søkefeltet vil man burde kunne søke direkte til databasen. Slik vil systemet ikke oppleve problemer ved mange ansatte.

5.2.1.8 Onboarding

Kravet om å opprette et onboardings-kort når en ansatt skal slutte er innfridd. Dette løses ved hjelp av en cron-jobb (2.1.6) som er satt opp til å kjøre en gang om dagen.

Cron-jobben kjøres ved at en ekstern kilde kaller på et API-endepunkt som gjør oppgavene spesifisert i 4.2.1.8, 4.2.1.9 og 4.2.1.10. Grunnen til at det er løst på den måten er hvordan rammeverket Next.js er bygget opp. Det er ikke mulig å planlegge oppgaver ettersom man ikke har en dedikert tjener som kjører til enhver tid. Dette problemet løses ved å la en ekstern kilde gjøre kallet. For å legge på et ekstra lag med sikkerhet må det sendes med en hemmelig nøkkel i HTTP-forespørselen. På den måten blir det sikret at det er kun planleggeren som kan gjøre kallet ettersom planleggeren er den eneste som har kjennskap til den hemmelige nøkkelen.

5.2.1.9 Løpende

Som nevnt i 5.2.1.8 benyttes det cron-jobber for å opprette kortene årlig. Tidligere i utviklingen ble det besluttet at brukerene av systemet skulle definere hvilken dato kortene skulle opprettes. Det vil si at hvis brukeren satt at oppgavene i kvartal 1 skulle opprettes 25.12 hvert år ville oppgavene kun opprettes denne datoen. Ettersom systemet må benytte seg av en ekstern kilde for å gjøre et API-kall var denne tilnærmingen problematisk fordi man ikke har kontroll over hvorvidt kilden gjør kallet eller ikke. Altså er det muligheter for at kilden ikke får utført kallet ved enkelte anledninger. I så fall ville man ved ytterste konsekvens ikke fått opprettet oppgaver for et helt kvartal.

Løsningen ble å gjøre denne logikken datouavhengig slik at eventuell kall som ikke ble gjort kunne gjøres på nytt dagen etter uten konsekvenser. Ved å gjøre det på denne måten vet man at oppgavene vil bli opprettet før eller siden, og kan dermed konkludere med at kravet er innfridd.

Akseptansekriteriet om at oppgavene skal opprettes en øke før fasen starter er så langt det lar seg gjøre innfridd. Dersom kallet ikke skulle skje vil det forsøke å la seg gjøre dagen etterpå. Men så lenge alt går som det skal vil dette kriteriet være innfridd.

5.2.1.10 Offboarding

Måten oppgavene i offboarding blir opprettet på er svært likt som i onboarding.

På bakgrunn av at kravene i 5.2.1.8 er innfridd er også kravene innfridd.

5.2.1.11 Se prosessmaler

Kravet om å se alle prosessmaler med tilhørende oppgaver er innfridd. Akseptansekriteriene er også oppfylt.

5.2.1.12 Oppdatere oppgaver

En bruker av systemet kan enkelt legge til, fjerne og endre oppgaver. Det kommer tydelig frem av brukertestingen at dette er intuitivt. Dersom en bruker ikke fyller ut de nødvendige feltene vil oppgaven heller ikke opprettes. Dermed kan man anse akseptansekriteriet som innfridd.

5.2.1.13 Oppdatere faser

I de ulike prosessmalene kan en bruker foreta endringer på de forskjellige fasene. Ettersom de løpende oppgavene opprettes årlig måtte også måten man oppretter/oppdaterer en fase på være forskjellig sammenlignet med onboarding og offboarding. Til tross for denne forskjellen tilsier brukertestene at dette er forståelig for brukeren. I tillegg kan brukeren få ekstra informasjon om hva de ulike feltene betyr ved hjelp av *tooltips* dersom det er nødvendig. Kravet anses som innfridd.

5.2.1.14 Mine ansatte

Dette kravet er innfridd. Siden viser alle ansatte som har den innloggede brukeren som personalansvarlig, og hvilken fase de for øyeblikket er i.

5.2.1.15 Alle ansatte

Tilsvarende som mine ansatte. Kravet er innfridd.

5.2.1.16 Søke etter ansatte

På samme måte som i 5.2.1.5 benyttes *fuzzy-search* for å søke etter ansatte. Dette gir gode søkeresultater for brukeren, og man kan dermed konkludere med at kravet er innfridd.

5.2.1.17 Oppgaver tilknyttet ansatt

Fra mine/alle ansatte og mine/alle oppgaver kan man navigere seg frem til ansatt-siden. Basert på hvor du trykker deg inn fra vil den vise den aktuelle prosessen med tilhørende faser og oppgaver. Kravet er innfridd.

5.2.1.18 Ansatt-historikk

Dersom en ansatt har oppgaver fra tidligere år eller andre prosesser vil man kunne velge å se disse fra nedtrekksmenyen. Kravet er innfridd.

5.2.1.19 Legge til individuelle oppgaver

På ansatt-siden kan man legge til individuelle oppgaver. Det kommer også tydelig frem hvilke oppgaver som er individuelle og hva som kommer fra prosessmalen. Kravet anses som innfridd.

5.2.1.20 Endre forfallsdato

Som beskrevet i 4.2.1.20 kan en bruker endre forfallsdato for en spesifikk fase. Forfallsdatoen blir validert slik at en bruker ikke kan endre til en ugyldig dato. Kravet anses som innfridd.

5.2.1.21 Filopplastning / notering

Tidlig i planleggingsfasen var det ønskelig å se på mulighet for å enten laste opp dokumenter eller notere på systemet. Hensikten var å kunne knytte notater til ulike samtaler man har iløpet av et år. Teamet, sammen med oppgavestiller konkluderte med at dette ikke var nødvendig likevel. Først og fremst fordi at oppgavestiller allerede hadde en eksisterende løsning som fungerte godt. I tillegg var det ikke ønskelig å ha flere lagringsplasser for filer knyttet til de ansatte.

5.2.1.22 Varslinger

Varslinger er implementert slik at man kan få varslinger i selve systemet eller på Slack. For at man skal kunne ta i bruk varslinger på Slack er det nødvendig å ha installert en såkalt "App" tilhørende firmaets workspace. Kravet anses uansett som innfridd.

5.2.1.23 Bestemme hvilke varslinger

Brukeren kan endre hvilke varslinger vedkommende ønsker i systemet eller på Slack. Per nå er det begrenset til fem statiske valg. Det vil si at det ikke er mulig å endre disse uten å måtte endre på kildekoden. I fremtiden burde man se på løsninger som er mer dynamiske.

Det har ikke kommet frem at det er ønskelig med flere muligheter for varslinger, og derfor anser teamet dette kravet som innfridd.

5.2.1.24 Logge inn

En bruker kan logge seg inn ved å benytte seg av sin Google-konto. Dette ble benyttet da alle ansatte i firmaet har en Google-konto tilknyttet seg, og det kan brukes for å verifisere de som forsøker å logge seg inn. Har man ikke en konto lagret i databasen vil man heller ikke kunne logge inn.

Det kunne vært aktuelt å implementere flere innloggingsmuligheter med andre tredjeparter enn Google, men det ble bestemt at ikke var nødvendig med første versjon.

5.2.1.25 Innstillinger

Innstillinger består kun av hvilke varslinger man ønsker (5.2.1.22). Det er ikke blitt implementert flere innstillinger for brukeren foreløpig. Kravet anses uansett som innfridd.

5.2.1.26 Vanlig ansatt

Mulighetene for dette ble undersøkt, men ikke implementert. Det er noe som ikke er vanskelig å implementere, men teamet og oppgavestiller ble enige om å ikke prioritere dette i denne versjonen.

5.2.2 Ikke-funksjonelle krav

5.2.2.1 Brukbarhet

Det ble stilt en del krav til brukbarhet fra oppgavestiller. Først og fremst skulle systemet innfri de viktigste kriteriene i WGAC 2.1 for å sikre universell utforming. I tillegg til de automatiske testene har det blitt foretatt flere manuelle tester som verifiserer at man blant annet kan benytte systemet ved å kun bruke tastatur, bruke skjermleser osv. Systemet får også gode tilbakemeldinger fra de automatiske testene som forteller at det ikke er blitt funnet konvensjonelle feil med hensyn på WCAG. Teamet anser derfor dette kravet som innfridd.

I tillegg til universell utforming skulle systemet være enkelt og intuitivt i bruk. Som beskrevet i 4.2.2.1 har det blitt foretatt brukertester for å bekrefte dette. Brukerne har, uten støtte fra utviklerne, klart å benytte seg av systemet og de funksjonelle egenskapene uten problemer. I de tilfellene hvor funksjonaliteten har vært uklar har teamet tatt hensyn til brukernes tilbakemeldinger, og gjort de nødvendige endringene.

Mobilvisning har blitt implementert slik at det er brukbart. Systemet vil ikke brekke ved eventuell mobilvisning, men det er likevel en anbefaling om å bruke systemet på en datamaskin eller nettbrett. I senere versjoner burde dette forbedres.

5.2.2.2 Sikkerhet

Kravet om sikkerhet er implementert. Blant annet gjennom å sørge for autentisering av brukerne slik at ingen andre enn ansatte skal ha tilgang til systemet. Det har også blitt foretatt flere manuelle tester for å verifisere at ingen uten adgang skal få tilgang til systemet. Derfor anser teamet dette kravet som innfridd.

5.2.3 Refleksjon rundt sluttprodukt

5.2.3.1 Styrker ved løsningen

Hovedfordelen med systemet er at den automatisk oppretter oppgaver uten direkte interaksjon med brukeren. Systemet trenger kun å koble seg opp til bedriftens internsystemer, og gjør deretter all logikken selv. Dette står i kontrast til dagens situasjon hvor prosessen er svært manuell. Effekten av å automatisere dette gjør opplevelsen av å være personalansvarlig mye bedre. Videre bidrar det også til en bedre opplevelse for alle ansatte da det legges større fokus på å få gjort alle oppgaver. Dette gjøres ved å minne de ansvarlig på at "disse oppgavene må fullføres". På den måten kan systemet være med på å gi hele bedriften et løft.

Systemet bruker en åpen løsning der alle har mulighet til å se hvilken arbeidsoppgaver enhver har ansvar for og hvor langt i prosessen enhver personalansvarlig har kommet med hver ansatt. Dette fører til et underliggende "press" om at oppgaver må utføres før de forfaller.

Det har kommet frem av brukertestene (Vedlegg E) at systemet er meget intuitivt, og er lett å bruke. Dette er en styrke ved systemet da det ikke er behov for å måtte bruke lang tid på å forstå systemet.

5.2.3.2 Svakheter ved løsningen

Selv om løsningen gjør det kunden har forespeilet finnes det flere svakheter med løsningen. Blant annet har ikke teamet fått fokusert på paginering ved innlasting av data. Det vil si at selv om en bedrift skulle ha veldig mange ansatte vil alle disse bli hentet ut og presentert på nettsiden. Det har blitt gjort tester for opp mot 200 ansatte, og det er merkbart at siden får en liten nedgang i ytelse.

Mobilvisning kan også forbedres. Som nevnt i 4.2.2.1 er dette blitt implementert, men er ikke optimalisert. Derfor burde det utvikles en løsning hvor mobilvisning vil fungere tilnærmet likt som på en datamaskin.

5.2.3.3 Valg av teknologier

For ethvert systemutviklingsprosjekt er det kritisk å velge de riktige verktøyene og rammeverkene for jobben. Valget av Next.js er teamet til syvende og sist svært fornøyd med. Next.js genererer siden på tjeneren (se 2.1.3), og dette fører til at systemet oppfattes svært raskt og

responsivt. I tillegg gir det også mulighet for å implementere et API i samme økosystem som klienten. Dermed hadde vi både klienten og tjeneren i ett og samme prosjekt, hvilket ga teamet flere fordeler som gjorde at utvikling gikk raskt og effektivt.

Videre ble Material UI benyttet som komponentbibliotek. Dette er et meget godt dokumentert bibliotek som gir mange fordeler ut av boksen. Blant annet har det blitt utviklet med universell utforming i fokus slik at mange av komponentene som en standard har innfridd flere kriterier fra WCAG 2.1. Dette gjorde at teamet sparte mye tid, og samtidig fikk utviklet gode løsninger.

Bruken av PostgreSQL var ukjent for teamet, men var totalt sett helt uproblematisk. I tillegg ble ORM-rammeverket Prisma benyttet. Dette var et rammeverk som var relativt nytt, og det viste seg at det det dukket opp en del bugs underveis. I tillegg til dette ble det hyppige lansert nye versjoner av rammeverket hvor enkelte måter å gjøre spørringer på ikke lenger ble støttet. Dette gjorde at teamet måtte skrive om flere av de allerede fungerende spørringene underveis. Til tross for disse problemene anser teamet som bruken av Prisma som svært gunstig for utviklingen.

Alt i alt er teamet svært godt fornøyd med valgene som ble tatt, og sitter igjen med en solid mengde erfaringer og kunnskap knyttet til de teknologiske rammeverkene som er benyttet.

5.2.3.4 Profesjonsetiske problemstillinger

I utgangspunktet er systemet utviklet slik at man ikke lagrer sensitiv data om de ansatte i systemet. Det er uansett verdt å merke seg at systemet skal koble seg opp mot den interne databasen til firmaet. Her ligger data som ikke er ment for offentligheten. Derfor er det viktig for oss som dataingeniør å begrense adgangen til systemet for alle utenom ansatte, samtidig som vi begrenser sensitiv informasjon og kun henter ut nødvendig data.

5.3 Administrative resultater

5.3.1 Fremdriftsplan

Fremdriftsplanen ble benyttet for å gi teamet et helhetlig bilde av hvordan vi så for oss prosjektperioden. Den ble delt opp i samme kategoriene som ble brukt i timeføringslistene hvor vi estimerte hvor mye tid teamet jobbet med de ulike aktivitetene. Teamet satte også opp milepæler med frister, slik at teamet alltid hadde klare mål som må oppnås gjennom hele prosjektet.

Teamet har fint klart å følge planen gjennom hele perioden og har truffet de fleste milepæler på veien. Hovedproblemstillingen ble definert til riktig tid, men senere har teamet valgt å dele opp problemstillingen i flere forskningsspørsmål. Disse ble ikke ferdig formulert før flere uker etter fristen.

Det har blitt gjort mindre justeringer underveis på grunn av prioritetsendringer, men har fortsatt klart å gjennomføre alt til riktig tid.

5.3.2 Timeforbruk

Teamet har forsøkt å holde et jevnt timeforbruk gjennom hele prosjektet, men det viser seg at det likevel har vært varierende hvor mye man har fått gjort i løpet av perioden. I startfasen av prosjektet var det også undervisning i et annet fag, hvilket gjorde at antall arbeidstimer for bachelor-prosjektet måtte nedjusteres.

Når det gjelder de ulike aktivitetene har antall faktiske timer vært rimelig med tanke på hva vi forespeilet oss. Vi brukte mer tid på planleggingen enn vi hadde tenkt. Det ble fullført fem iterasjoner med design av skisser, slik at antall timer knyttet til planleggingen naturligvis ble høy. Likevel er dette noe teamet i ettertid ser nytten av, og mener at det absolutt ga store fordeler videre i prosjektet.

Totalt sett er teamet svært tilfreds med timeforbruket. Vi har klart å holde oss til å jobbe fra 8-16 stort sett hver dag, og brukte ikke mye mer tid enn vi hadde sett for oss.

5.3.3 Scrum

På et overordnet nivå har bruken av Scrum fungert meget godt for teamet. Hver sprint bestod av sprintplanlegging med oppgavestiller, hvor det ble valgt hvilken brukerhistorier som skulle prioriteres og estimert tid for hver brukerhistorie. Det ble holdt ukentlige stand-ups med oppgavestiller slik at teamet skulle ha noe å strekke seg etter på ukentlige basis. Dette fungerte veldig godt, og var svært drivende for teamet. Her fikk vi vise progresjon og hadde muligheten til å stille eventuelle spørsmål. Dette var med på å holde kontakten med oppgavestiller god, og ga stor verdi.

På slutten av hver sprint ble oppgavestiller med på gjennomgang av sprinten med presentasjon av løsningen på brukerhistoriene. Her bestemte oppgavestiller om løsningen var god nok, eller om teamet måtte justere på arbeidet som var blitt gjort. Til slutt gjennomførte teamet retrospektiv for å forbedre samarbeidet og arbeidsflyten til neste sprint.

Som nevnt i Vedlegg D Prosjekthåndbok kapittel 1 ble det valgt en Scrum-inspirert utviklingsprosess. Dette innebar blant annet at teamet ikke hadde en dedikert Scrum-master. Ettersom teammedlemmene hadde god kjennskap til Scrum fra tidligere prosjekter var dette problemfritt. Teamet var svært selvdrevne, og ordnet problemer som dukket opp underveis selv.

Lengden på sprintene var på to uker. Dette fungerte godt, og teamet følte at man fikk gjort mye produktivt i løpet av sprintene. Et problem som stadig dukket opp var at det var svært vanskelig å estimere hvor mye man trodde man skulle få gjort i løpet av sprinten. Vi endte ofte opp med å legge til flere brukerhistorier underveis, hvilket kan bli sett på Burndown-rapportene fra Vedlegg D Prosjekthåndboka kapittel 1 hvor grafen stiger. Dette forsøkte vi å løse ved å sette opp oppgaver tilsvarende arbeidsmengden vi klarte de foregående sprintene. Selv om estimeringen fortsatt ble feil, var det klar fremgang i de senere sprintene.

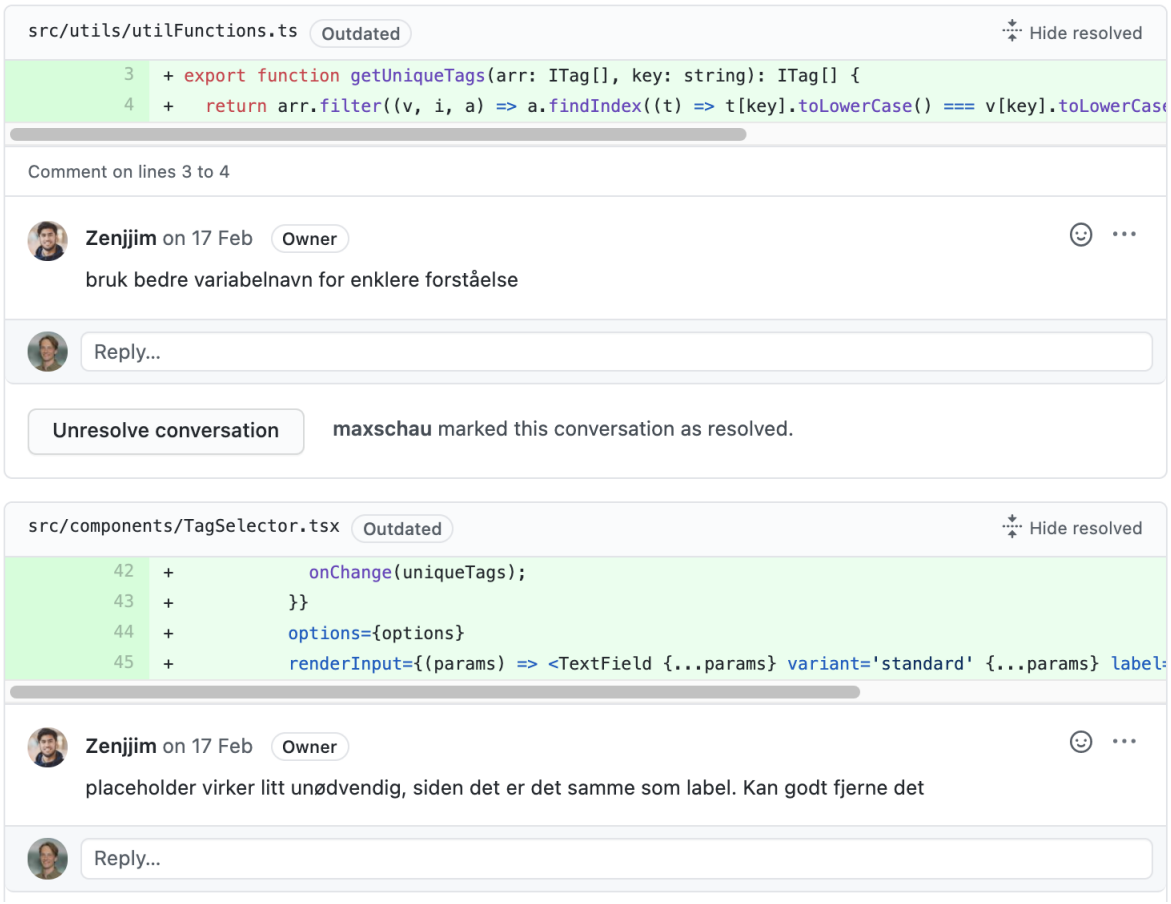
Oppsummert var valget av en agil utviklingsmetode svært vellykket, og teamet mener at dette har bidratt til økt produktivitet i prosjektperioden.

5.3.4 Arbeidsmetodikk

5.3.4.1 Versjonskontroll

For å holde oversikt over egen kode og unngå å endre på hverandres kode under arbeid benyttet teamet seg av *feature branches*. Dette er et branch-mønster hvor enhver funksjonalitet skal ha sin egen branch. Når den er ferdig utviklet kan den flettes inn i hovedbranchen. Teamet mente dette var den beste Git-flyten ettersom vi jobbet med enkle brukerhistorier som stort sett var uavhengig av andre funksjonaliteter. På denne måten kunne hvert teammedlem jobbe med hver sin brukerhistorie på en egen branch slik at arbeidet ikke ble påvirket av andre. Dette fungerte meget godt, og det var aldri store problemer knyttet til Git. En annen fordel med feature branches var at det ga teamet en historikk over når funksjonalitet ble implementert i prosjektet. I tillegg fikk man også oversikt over når det hadde gått galt i prosjektet.

For å sikre at systemet var stabilt ved endringer ble det implementert kontinuerlig integrasjon som kjører automatisk bygging av prosjektet og automatiske tester. Videre tok teamet i bruk *pull requests*[5]. Enhver pull request måtte godkjennes av det andre teammedlemmet før det kunne slås sammen med den sentrale koden. Her gikk teammedlemmene gjennom koden og kom med eventuelle endringsforslag for å holde koden lesbar, ryddig og effektiv. På denne måten sikret teamet god kodekvalitet gjennom hele prosjektet. I tillegg fikk hele teamet god oversikt over hele kodebasen.



Figur 35: Eksempel på pull request

5.3.5 Gruppearbeid

Teamet har hatt et godt samarbeid gjennom hele perioden. Det har vært fokus på at begge teammedlemmene skal ta like stor del i alle avgjørelser, og at det derfor ikke har vært en definert "sjef". Dette gjorde at det hele veien ble tatt avgjørelser som teamet var enige i.

Dette fremkommer blant annet av måten teamet jobbet med versjonskontroll (5.3.4.1) på, hvor man måtte godkjenne hverandres arbeid. Slik ble også alt av kode og utvikling godkjent som en felles avgjørelse.

Totalt sett er teamet svært godt fornøyd med måten gruppearbeidet har blitt gjennomført, og ser for seg å jobbe sammen i fremtiden.

[5]En måte å si fra til andre teammedlemmer om at du har gjort endringer på en branch, og ønsker å slå denne branchen sammen med en annen branch

6 Konklusjon og videre arbeid

6.1 Konklusjon

Problemstillingen som ble definert i 1.2 er et åpent spørsmål som ikke lar seg besvare fullstendig. Teamet konkretiserte dette ned i følgende forskningsspørsmål:

1. Hvilke begrensninger og muligheter medfører no-code rammeverk for det visuelle og funksjonelle, sammenlignet med tradisjonell utvikling?
    - Fra resultatene og diskusjonen kan man se at man, rent visuelt, vil ha de samme mulighetene som med tradisjonell utvikling. Når det gjelder det funksjonelle er det definitivt muligheter for å utvikle tilsvarende løsninger som med tradisjonell programvareutvikling. Blant annet var utviklingen av statiske sider med brukerinteraksjon uproblematisk for begge tilnærmingene. Det er derimot klare begrensninger når det kommer til å gjøre komplekse spørringer og logikk med no-code rammeverk. Man blir låst til den funksjonaliteten rammeverket tilbyr, mens man med tradisjonell utvikling har mer spillerom og frihet til å utvikle egne løsninger
2. Vil det være gunstig å benytte et no-code rammeverk med tanke på utviklingshastighet?
    - Det er svært gunstig å benytte et no-code rammeverk med tanke på utviklingshastighet. Resultatene viser at man kan spare mange timer ved å utvikle et system ved hjelp av no-code sammenlignet med tradisjonell utvikling
3. I hvilken grad vil sikkerheten bli ivaretatt for et no-code rammeverk sammenlignet med tradisjonell utvikling?
    - For et no-code rammeverk som Bubble vil sikkerheten følge med ut av boksen. Det har hatt fokus på å abstrahere dette vekk fra brukeren. Dermed blir sikkerheten ivaretatt på en gode måte. For tradisjonell utvikling må man ta stilling til dette selv ved å bestemme seg for om man skal implementere dette selv, bruke eksterne biblioteker og så videre. Det er derfor ikke ivaretatt i utgangspunktet slik som med no-code rammeverk
4. Kan man med et no-code rammeverk oppnå samme tilgjengelighet som man kan oppnå med tradisjonell utvikling?
    - Etter å ha testet ut tilgjengeligheten med ulike verktøy viser det seg at no-code rammeverket Bubble ikke har tilstrekkelig med funksjonalitet til å ha god nok tilgjengelighet. Med tradisjonell utvikling har man ikke de begrensningene som man får med et no-code rammeverk ettersom man blant annet kan endre direkte på HTML-elementene

På bakgrunn av de ingeniørfaglige resultatene og diskusjonene fra henholdsvis kapittel 4.2 og kapittel 5.2 knyttet opp mot målene definert i Vedlegg A Visjonsdokument kapittel 5 og 6 kan vi konkludere med et vellykket prosjekt.

6.2 Videre arbeid

Som nevnt i kapittel 5 anser teamet prosjektet som vellykket da målene spesifisert i oppstartsfasen er nådd. Til tross for dette ser teamet flere muligheter for å utvikle systemet videre.

6.2.1 Paginering

I kapittel 5.2.3.2 nevnes det at det er mangel på paginering, og at dette er noe som burde fokuseres på ved neste versjon. Dette gjelder for både oppgaver- og ansatt-oversikten. Slik det er nå vil ytelsen reduseres ved mye data, og det er noe som ikke er ønskelig. Selv om det ikke er sannsynlig at firmaet vil ha så store mengder data at det går utover ytelsen ser teamet uansett fordelen av å fokusere på skalerbarhet.

6.2.2 Mobilvisning

Se på mulighetene for å gjøre siden mer mobil-kompatibel slik at den er fullt funksjonelt på mobil på lik linje med skrivebordsversjonen.

6.2.3 Booke medarbeidersamtale

Som beskrevet i oppgaveteksten er det ønskelig at systemet burde kunne booke medarbeidersamtale, og er en gylden mulighet for systemet å bli integrert opp til en form for kalender.

6.2.4 Egen ansattplattform

Det er også ønskelig å utforske muligheten for at ansatte som ikke har personalansvar kan logge inn, og få se en oversikt over oppgaver som er tilknyttet dem. Det er ikke tenkt at disse skal ha de samme rettighetene på siden som ansatte med personalansvar.

7 Referanser

Referanser

[1] B. Barron. *2020's Most Surprising WordPress Statistics*. https://www.whoishostingthis.com/compare/wordpress/stats/. Lest: 2021-03-24. 2021.

[2] *Burn Down Charts*. URL: https://sites.google.com/a/effectivepmc.com/www/_/rsrc/1577600917219/blog/agile/information-radiators/burn-down-chart/ReleaseBurnDown.png.

[3] B. J. Case. *Universal design*. http://images.pearsonassessments.com/images/tmrs/tmrs_rg/UniversalDesign.pdf. Lest: 2021-04-15. 2008.

[4] *Crontab – Quick Reference*. https://www.adminschoice.com/crontab-quick-reference. Lest: 2021-04-17. 2021.

[5] R. T. Fielding. «Representational State Transfer (REST)». I: (2000). URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

[6] M. Fowler og M. Foemmel. «Continuous integration». I: *Thought-Works)* 122.14 (2006), s. 1–7. URL: https://moodle2019-20.ua.es/moodle/pluginfile.php/2228/mod_resource/content/2/martin-fowler-continuous-integration.pdf.

[7] *Manifestet for smidig programvareutvikling*. https://agilemanifesto.org/iso/no/manifesto.html. Lest: 2021-03-24. 2021.

[8] Mary K. Pratt. *Low-code and no-code development platforms*. https://searchsoftwarequality.techtarget.com/definition/low-code-no-code-development-platform. Lest: 2021-03-24. 2021.

[9] *Next.js*. https://en.wikipedia.org/wiki/Next.js. Lest: 2021-02-24. 2021.

[10] *Object–relational impedance mismatch*. https://en.wikipedia.org/wiki/Object\OT1\textendashrelational_impedance_mismatch. Lest: 2021-03-24.

[11] *OWASP Top Ten*. https://owasp.org/www-project-top-ten/. Lest: 2021-05-14. 2021.

[12] S. Pittet. «What is Continuous Deployment?» I: (). URL: https://www.atlassian.com/continuous-delivery/continuous-deployment.

[13] *PostgreSQL: The World's Most Advanced Open Source Relational Database*. https://www.postgresql.org/. Lest: 2021-03-24. 2021.

[14] *Prisma*. https://www.prisma.io/nextjs. Lest: 2021-03-23. 2021.

[15] *Representational state transfer*. https://en.wikipedia.org/wiki/Representational_state_transfer. Lest: 2021-03-24. 2021.

[16] J. Rymer og C. Obdam. *Interview with John Rymer, VP, Principal Analyst at Forrester Research*. Betty Blocks. URL: https://www.bettyblocks.com/john-rymer-chris-obdam-interview.

[17] *Server-side Rendering*. https://nextjs.org/docs/basic-features/pages. Lest: 2021-03-24. 2021.

[18] *Summary of Don Norman's Design Principles*. https://www.csun.edu/science/courses/671/bibliography/preece.html. Lest: 2021-04-14. 2021.

[19] *tabindex*. Mozilla. URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/tabindex.

[20] *The Low Code No Code Debate*. https://www.bettyblocks.com/no-code-vs-low-code. Lest: 2021-04-13. 2021.

[21] A. Torres mfl. «Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design». I: *information and software technology* 82 (2017), s. 1–18.

[22] *Web Content Accessibility Guidelines (WCAG) 2.1*. https://www.w3.org/TR/WCAG21/. Lest: 2021-05-14. 2021.

[23] *What are Norman's design principles?* https://www.educative.io/edpresso/what-are-normans-design-principles. Lest: 2021-04-14. 2021.

[24] *What is a Product Backlog?* https://www.scrum.org/resources/what-is-a-product-backlog. Lest: 2021-03-24. 2021.

[25] *What is a Sprint in Scrum?* https://www.scrum.org/resources/what-is-a-sprint-in-scrum. Lest: 2021-03-24. 2021.

[26] *What is a Sprint Retrospective?* https://www.scrum.org/resources/what-is-a-sprint-retrospective. Lest: 2021-03-24. 2021.

[27] *What is Agile?* https://www.atlassian.com/agile. Lest: 2021-03-24. 2021.

[28] *What is No-Code Application Development?* Betty Blocks. URL: https://www.bettyblocks.com/no-code-low-code-application-development.

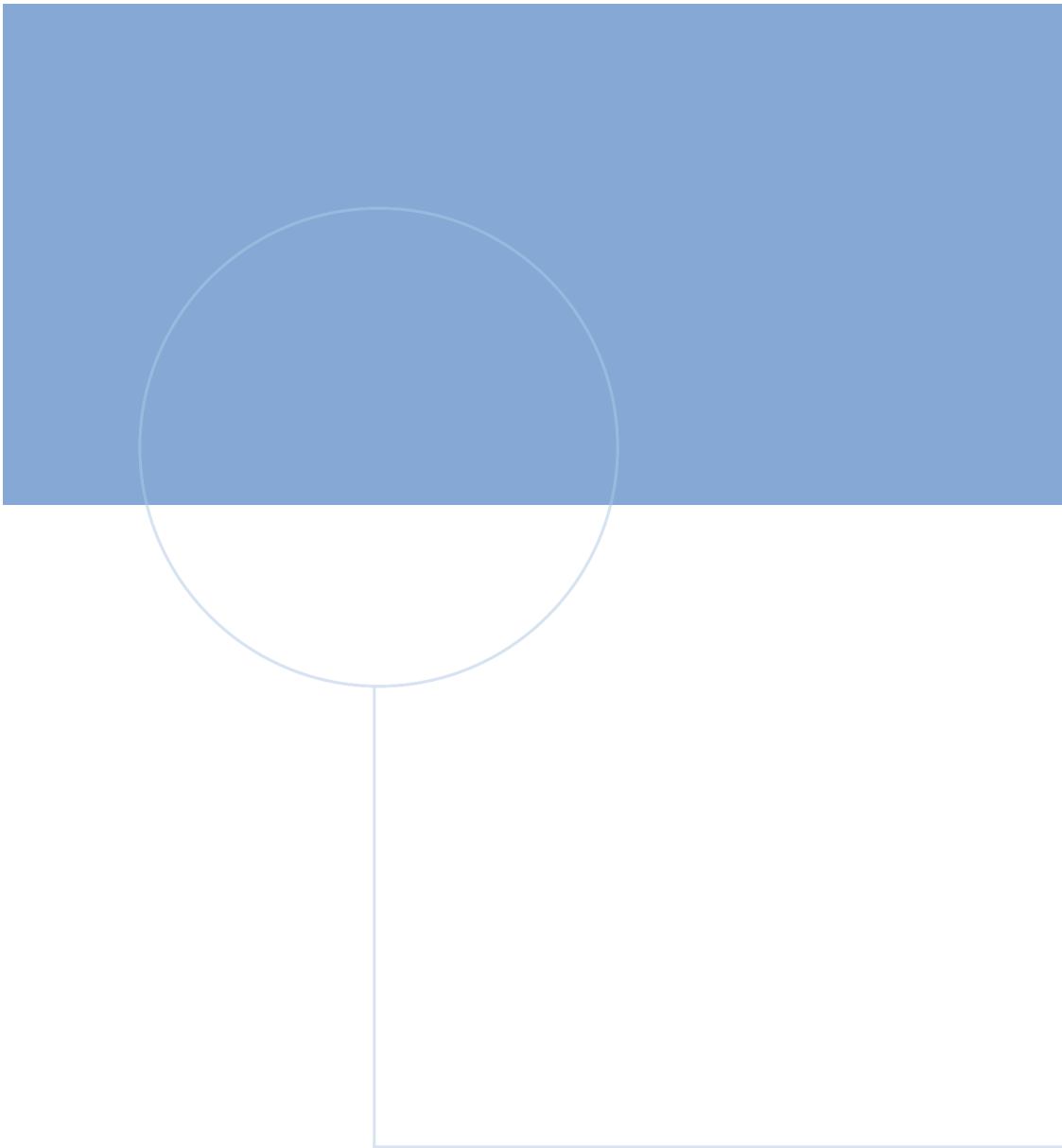
[29] *WHAT IS SCRUM?* https://www.scrum.org/resources/what-is-scrum. Lest: 2021-03-24. 2021.

[30] *What is Sprint Planning?* https://www.scrum.org/resources/what-is-sprint-planning. Lest: 2021-03-24. 2021.

[31] *Wordpress*. https://en.wikipedia.org/wiki/WordPress. Lest: 2021-03-24. 2021.

8 Vedlegg

- Vedlegg A - Visjonsdokument
- Vedlegg B - Kravdokumentasjon
- Vedlegg C - Systemdokumentasjon
- Vedlegg D - Prosjekthåndbok
- Vedlegg E - Simulering av opprettelse av oppgaver
- Vedlegg F - Brukertesting
- Vedlegg G - Videodemonstrasjon av systemene



NTNU

Norwegian University of
Science and Technology