BACHELOROPPGAVE:

BO19E-18 PASIENTINFOFLYT

Lars Vågenes
Andreas Valen

31. mai. 2019

Dokumentkontroll

Rapportens tittel: BO19E-18 Pasientinfoflyt / Verat	*Dato/Versjon* 31. mai. 2019/1.00
	Rapportnummer: B019E-18
Forfatter(e): Lars Vågenes Andreas Valen	*Studieretning:* 16HKOM
	Antall sider m/vedlegg 42
Høgskolens veileder: Knut Øvsthus	*Gradering:* Delvis åpen
Eventuelle Merknader: Vi tillater at rapporten kan publiseres, men kildekode er konfidensiell.	

Oppdragsgiver: Egendefinert	*Oppdragsgivers referanse:*
Oppdragsgivers kontaktperson(er) (inklusiv kontaktinformasjon):	

Revisjon	**Dato**	**Status**	**Utført av**
0.01	25.01.19	Forstudie	Lars Vågenes, Andreas Valen
1.00	30.05.19	Ferdigstilling	Lars Vågenes, Andreas Valen

Forord

Vi ønsker å takke lærerne fra faget ING101 da de hjalp oss å komme i kontakt med Bergen legevakt. Det var nemlig gjennom en presentasjon i dette faget at Bergen legevakt presenterte problemstillingen som inspirerte oss til å utvikle dette systemet. Spesielt vil vi takke Ruth Donovan, som er læreren som videre fikk oss i kontakt med *Avans*. Selv om vi ikke har utviklet systemet i samarbeid med *Avans,* gav de oss noen gode innspill til planleggingen, og tilba oss et samarbeid. Vi vil derfor også gi en takk til *Avans*. Videre vil vi også takke sykepleierne på Bergen legevakt. De var veldig hyggelig å samarbeide med når vi har var på besøk for å samle informasjon, og når vi presenterte hva vi har laget så langt.

Sammendrag

Oppgaven vi tok for oss gikk ut på å løse Bergen Legevakt sin problemstilling. De hadde presentert et behov for å forbedre oversikten over det fordelte ansvaret på sengeposten deres. Dette ble originalt presentert i sammenheng med faget *ING101 Teknologiledelse, økonomi og nyskaping,* og oppgaven vår var en videreføring av problemstillingen vi tok for oss den gangen. Vi fikk i møte med legevakten kommet frem til kravspesifikasjoner i henhold til funksjonalitet og design. Vi har også på befaring fått satt oss grundig inn i hvordan løsningen må kunne fungere, og hvilke utfordringer som ventet oss.

Innledningsvis hadde vi planer om å samarbeide tett med IT-selskapet *Avans* da de hadde mye erfaring med helsesektoren og sensitiv data, og de skulle lage en API vi kunne koble oss opp mot for å benytte oss av deres journalsystem. I møte med *Avans* ble vi enige om å lage løsningen vår med en hexagonal arkitektur. Dette innebar at hovedkoden skulle være i sentrum av applikasjonen og alle komponenter som database, web API ol. skulle knyttes til gjennom en API. På denne måten ble komponentene i systemet enkle å bytte ut uten å måtte endre hele systemet. *Avans* skulle da kobles til systemet vårt og vi skulle bruke journalsystemet deres som en del av databasen vår.

Vi endte opp med å utvikle en løsning som skal kjøre lokalt på legevakten istedenfor. *Avans* sitt journalsystem ble da ikke lenger nødvendig, og databasekomponenten ble byttet ut med vår egen som vil ligge på serveren vi utplasserer på legevakten. Det kan bli relevant med et eksternt journalsystem i fremtiden, men da er det ikke sikkert *Avans* er de som nødvendigvis passer oss best.

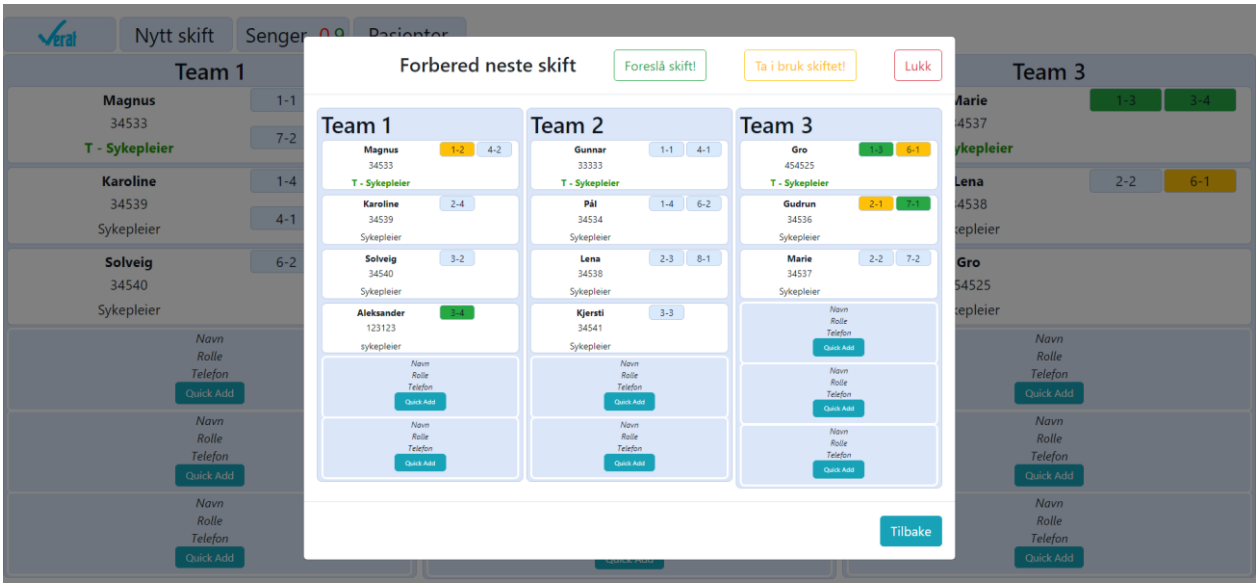
En av hovedutfordringene under utviklingen av løsningen vår var å få en så høy grad av automasjon som mulig, uten å ha muligheten til å hente data fra legevakten sine etablerte systemer. Dette løste vi ved å lage en algoritme som tar imot en liste med ansatte og foreslår et skift basert på hvordan skiftene har vær tidligere. Dette er fortsatt et stykke unna et helautomatisk program, ettersom listen med ansatte må settes i sammen og sendes inn av brukeren, men det sparer fortsatt enormt mye tid som medførte at funksjonen ble svært godt likt av legevakten under en demo.

Brukergrensesnittet vi lagde hoster vi på en server på samme hardware som resten av webapplikasjonen. Den kan nås via nettleser, og er derfor umiddelbart tilgjengelig via alle smartenheter og operativsystem. Nettsiden er profesjonell, intuitiv og ikke minst smidig. Endringene skjer i sanntid sømløst på alle skjermene som er koblet opp samtidig. Dette har vi fått til ved å bruke *React.js.* Dette er et bibliotek/rammeverk som driftes av *Facebook* og er Open-Source. *React.js* har hjulpet oss til å lage et grensesnitt som kan oppdatere deler av skjermen, istedenfor å måtte laste inn hele siden på nytt igjen når en endring i databasen har blitt oppdaget av front-enden. Dette er kritisk for en bra brukeropplevelse når flere folk skal legge til og endre informasjon via webapplikasjonen samtidig. Front-enden er en av de tingene som må jobbes mye med før utplasseringen av løsningen, da det har blitt lite tid til testing av løsningen på smartenheter. Dette har ført til at det har blitt oppdaget sårbarheter i systemet som vi ikke har rukket å fikse før første levering.

Under det siste møtet med legevakten kom det fram at de mener løsningen vår, med noen mindre endringer, fyller deres behov og de er svært interessert i å inngå et samarbeid med oss. Planen er at vi skal jobbe med systemet kontinuerlig, og utvikle det basert på tilbakemeldingene vi får fra brukerne selv. Legevakten foreslo at de kan ansatte oss som konsulenter, slik at vi kan få lønn for å drifte, samt videreutvikle systemet. Alle anskaffelser gjort offentlige institusjoner skal i teorien legges ut som anbud, og det er regler om konkurranse. Dette kan vi unngå så lenge summen på legevaktens direkte kostnader ikke overstiger 100.000kr. Vi er i prosessen av å skrive en kontrakt der alle forholdene

klargjøres. Planen er å starte med en rimelig pris, og å reforhandle etter en avtalt periode. Da har vi som utviklere muligheten til å få mye tilbake for innsatsen vi legger i å forbedre programmet, ettersom programmets verdi vil re-evalueres under de nye forhandlingene.

For å fremstå som profesjonelle har vi valgt navnet *Verat* som er det latinske ordet for "organisert", og gått til anskaffelse av domenet verat.no. Her er planen å legge inn litt reklame for oss selv slik at det blir et bra førstegangsmøte med folk som søker på *Verat*. Planen vår er å ha *Verat* som en 20 -30% stilling fremover, men dette kan endre seg dersom vi opparbeider oss en større kundeportfolie.



Figur 1 Skjermbildet av frontenden til webapplikasjonen slik som vi leverer den. Skjermbildet er tatt under forberedelse av det neste skiftet.

1 Innhold

1 Innledning

Denne oppgaven omhandler utvikling av et datasystem til bruk av sykepleiere og leger på sengepost. Dette skal være et system for lett tilgang til informasjon som en sykepleiers trenger i løpet av arbeidsdagen. Systemet skal bidra til å effektivisere sykepleiernes arbeid, og dermed også gi pasientene en tryggere og bedre opplevelse.

1.1 Motivasjon

Legevakten i Bergen presenterte diverse problemstillinger de møter i sin hverdag for studentene på høyskolen. De har et håp om at studentene kan planlegge, og etter hvert implementere, en mulig løsning. Blant problemene som ble introdusert, var det spesielt et hvor vi så potensiale for stor forbedring. «Pasientflyt» problemstillingen, som de hadde døpt det, går ut på å effektivisere tungvidt arbeid som sykepleiere må gjøre hver dag.

Legevakten presenterte arbeidsmetoden sin slik som den var, og påpekte flere tungvinte aspekter ved en normal arbeidsdag. De ønskte hjelp til å finne måter å redusere denne arbeidsmengden. Problemene som virket å skape mest frustrasjon var systemet for romfordeling. Dette innebærer organisering av personale, senger og pasienter på avdelingene. Ved å lage et system som løser dette problemet bidrar vi til å forbedre pasientenes behandling.

1.2 Problemstilling

Per dags dato opererer legevakten med en felles «whiteboard» tavle som skal presentere informasjon om blant annet:

- Hvilke ansatte som er på jobb og tilgjengelige.
- Hvilke rom, med det også indirekte hvilke pasienter, de er ansvarlig for.
- Telefon nummer, stilling, team og rolle innad i team.
- Pasientinfo som: "smittefare", "vasket", "skal hjem i dag" m.m.

Alt dette blir formidlet manuelt via håndskrift og magneter i farger, noe som gjør at det går tregt å oppdatere tavlen og man må fysisk besøke tavlen, eller ringe noen som har den i syne for å få med seg endringene. På et feltbesøk avslørte sykepleierne at dette er så tungvint at de ofte ikke engang bruker de fargekodete magnetene, altså de magnetene som forteller om status på rommene. Videre så vi også andre flaskehalser som lett kan unngås med et bedre system.

Figur 2 Problem

På bildet «Problem» ovenfor ser mann hvor mye informasjon og hvor mange folk som skal innom tavlene mange ganger for dagen. Denne tavlen finnes det 3 kopier av, som ideelt sett skulle vert lik til alle tider, samt en ekstra tavle for vaskehjelpen. Videre ser vi en utskrift som er så uoversiktlig at bemanningen sitter seg ned og fargelegger denne hver gang de de starter en vakt.

Vi skal lage et system som tar hånd om de overnevnte problemene. Systemet skal samle informasjon og automatisere synkronisering av informasjonen. På denne måten skal systemet bidra til å effektivisere arbeidet til de ansatte og øke tryggheten for pasientene.

2 Kravspesifikasjon

Den underliggende listen er en beskrivelse av den funksjonaliteten som vi ser for oss er nødvendige å ha, hvis systemet skal være en fullverdig erstatning av det legevakten bruker i dag. Vi har utarbeidet denne listen basert på en presentasjon fra Bergen legevakt samt et besøk hos dem. Vi har da fått gode innspill til hva som er nødvendig i systemet. Denne kravspesifikasjonen er ikke utarbeidet av legevakten, og trenger da heller ikke bli fulgt til punkt og prikke. Vi vil derfor tillate oss selv å avvike i liten grad, om vi ser bedre alternativer etter hvert som vi utvikler systemet. Eventuelle avvik vil begrunnes i 6.3 Design.

1. Tavlevisning av informasjon
    - 1.1. Informasjon som normalt skal vises
        - 1.1.1. Ansatte som er på vakt, med navn
        - 1.1.2. Hvilken rolle hver ansatt har
        - 1.1.3. Hvilket lag hver ansatt tilhører
        - 1.1.4. De Ansattes jobb-telefon nummer
        - 1.1.5. Senger/pasienter som tilhører hver ansatt
        - 1.1.6. Status på seng:
            - 1.1.6.1. Smittefare
            - 1.1.6.2. Okkupert av pasient
            - 1.1.6.3. Kan reise hjem
            - 1.1.6.4. Må vaskes
            - 1.1.6.5. Klar for ny pasient
        - 1.1.7. Tid og dato
    - 1.2. Funksjonalitet som skal støttes
        - 1.2.1. Ansatt
            - 1.2.1.1. Legge til ansatt
            - 1.2.1.2. Slette ansatt
            - 1.2.1.3. Endre ansatt
                - 1.2.1.3.1. Velge navn
                - 1.2.1.3.2. Velge telefonnummer
                - 1.2.1.3.3. Velge lag til ansatt
                - 1.2.1.3.4. Velge rolle for ansatt
            - 1.2.1.4. Legge seng til ansatt
            - 1.2.1.5. Fjerne seng fra ansatt
        - 1.2.2. Seng
            - 1.2.2.1. Legge til seng
            - 1.2.2.2. Fjerne seng
            - 1.2.2.3. Endre seng (Legge til pasient)
            - 1.2.2.4. Koble seng til pasient
            - 1.2.2.5. Fjerne seng fra pasient
            - 1.2.2.6. Markere seng som rengjort
        - 1.2.3. Pasient
            - 1.2.3.1. Legge til pasient
            - 1.2.3.2. Endre pasient
            - 1.2.3.3. Fjerne pasient

1.2.4. Vaktplan
1.2.4.1. Forberede neste vakt
1.2.4.1.1. Velge Ansatt til vakt
1.2.4.1.2. Fjerne ansatt fra vakt
1.2.4.1.3. Velge om vakt automatisk skal tre i kraft.
1.2.4.2. Endre til forberedt vakt
1.2.4.3. Endre aktive vaktplan
1.2.5. Vise info relatert til en ansatt

2. Systemet skal
2.1. Være tilgjengelig for PCer, tavledisplay og mobile enheter på nettet
2.2. Logge alle vakter, inkludert endringer
2.3. Kryptere informasjon som sendes over nettverket
2.4. Kreve sikker pålogging

3. Følgende skal lagres i systemet
3.1. Ansatt
3.1.1. Navn
3.1.2. Rolle
3.1.3. tilknyttet TLF
3.1.4. Lag
3.2. Seng
3.2.1. Plassering – Lagres som «romnummer-sengenummer» eks: «1-2»
3.2.2. Status
3.2.3. Tilhørende pasient
3.2.4. Tilhørende ansatt
3.2.5. Renhold
3.3. Pasient
3.3.1. Innsjekk/Utsjekk
3.3.2. Kommentar
3.4. Vaktplan
3.4.1. Teams med ansatte
3.4.2. Forberedte planer
3.4.2.1. Når forberedte planer trer i kraft

3 Analyse av problemet

For å bygge en forståelse av problemet har vi vert i møter med legakten i Bergen samt *Avans*, som er en leverandør av journalsystem, og har erfaring med IT i helsesektoren. Gjennom det har vi utarbeidet en plan for systemet som er utgangspunktet for kravspesifikasjonen.

I analysen av problemet går vi inn i detalj på kravspesifikasjonen, og beskriver bruksmønster for alle mulige interaksjoner bruker skal ha med systemet. På denne måten avsløres alle behov vi kan se i forkant av utviklingen.

Bruksmønster for Ansatt:

1.2.1.1.	**Legge til ny ansatt i listen over ansatte.**	
Beskrivelse	Dette brukermønsteret beskriver hvordan brukeren navigerer informasjonssystemet for å legge inn en ny Ansatt.	
Forutsetninger	1. Informasjonssystemet er operativt. 2. Instansen informasjonssystemet er aksessert fra håndterer inndata.	
Normal arbeidsflyt:		
1.	Brukeren aksesserer systemet via smartenhet eller datamaskin.	
2.	Brukeren velger hovedmeny.	
3.	Systemet presenterer valgalternativer.	
4.	Brukeren velger handlingen "Legg inn ny ansatt".	
5.	Systemet ber bruker fylle ut den ansattes tilhørende infoceller.	
6.	Systemet lagrer personen i listen over ansatte og loggfører endringen.	
7.	Menyen lukkes og skjermen går tilbake til infooversikten.	
Alternativ arbeidsflyt:		
6a.	Hvis den ansatte allerede finnes i systemet.	
1.	Systemet viser en feilmelding med valg: Avbryt, Endre Ansatt.	
2.	Avbryt: Ingen endringer lagres, meny lukkes.	Endre Ansatt: Brukermønsteret fortsetter fra steg 5.
6b.	Hvis ikke tilstrekkelig info er fylt ut når brukeren prøver å lagre.	
1.	Systemet viser en feilmelding med valg: Avbryt, Endre Ansatt.	
2.	Avbryt: Ingen endringer lagres, meny lukkes.	Endre Ansatt: Brukermønsteret fortsetter fra steg 5.
1.2.1.2.	**Fjerne ansatt fra listen over ansatte.**	
Beskrivelse	Dette brukermønsteret beskriver hvordan brukeren navigerer informasjonssystemet for å fjerne en ansatt fra listen over ansatte.	
Forutsetninger	1. Informasjonssystemet er operativt. 2. Instansen informasjonssystemet er aksessert fra håndterer inndata. 3. Den ansatte som skal fjernes eksisterer i listen over ansatte.	
Normal arbeidsflyt:		
1.	Brukeren aksesserer systemet via smartenhet eller datamaskin.	
2.	Brukeren velger hovedmeny.	
3.	Systemet presenterer valgalternativer.	
4.	Brukeren velger handlingen "Fjerne ansatt".	
5.	Systemet viser en oversikt over registrerte ansatte.	
6.	Brukeren velger den ansatte som skal fjernes.	
7.	Systemet summerer ønsket handling og ber brukeren bekrefte eller avbryte.	
8.	Brukeren bekrefter.	
9.	Systemet fjerner den ansatte fra listen over ansatte og loggfører handlingen.	
Alternativ arbeidsflyt:		
5a.	Hvis ingen ansatte er registrert, eller systemet ikke klarer å aksessere listen over ansatte.	
1.	Systemet viser en feilmelding.	
2.	Meny lukkes.	
8a.	Brukeren velger avbryt.	

1.	Ingen endringer lagres, meny lukkes.	
1.2.1.3.	**Endre ansatt**	
Beskrivelse	Dette brukermønsteret beskriver hvordan brukeren navigerer informasjonssystemet for å endre en eksisterende ansatt fra listen over ansatte. Brukermønsteret er identisk for **1.2.1.3.1-4.**	
Forutsetninger	1. Informasjonssystemet er operativt. 2. Instansen informasjonssystemet er aksessert fra håndterer inndata. 3. Den ansatte som skal endres eksisterer i listen over ansatte.	
Normal arbeidsflyt:		
1.	Brukeren aksesserer systemet via smartenhet eller datamaskin.	
2.	Brukeren velger hovedmeny.	
3.	Systemet presenterer valgalternativer.	
4.	Brukeren velger handlingen "Endre ansatt".	
5.	Systemet viser en oversikt over registrerte ansatte.	
6.	Brukeren velger den ansatte som skal endres.	
7.	Systemet viser den ansattes tilhørende redigerbare infoceller.	
8.	Brukeren utfører ønskede endringer og velger å lagre.	
9.	Systemet lagrer endringene og loggfører handlingen.	
Alternativ arbeidsflyt:		
2a.	Den ansatte brukeren ønsker å endre er tilgjengelig fra hovedsiden og velges direkte fra siden.	
1.	Systemet presenterer den ansattes profil.	
2.	Brukeren velger "Endre ansatt".	
3.	Systemet fortsetter fra steg 7.	
8a.	Brukeren velger avbryt.	
1.	Ingen endringer lagres, meny lukkes.	
8b.	Hvis ikke tilstrekkelig info er fylt ut når brukeren prøver å lagre.	
1.	Systemet viser en feilmelding med valg: Avbryt, Endre Ansatt.	
2.	Avbryt: Ingen endringer lagres, meny lukkes.	Endre Ansatt: Brukermønsteret fortsetter fra steg 7.

1.2.1.4.	**Legge en seng til en ansatt.**
Beskrivelse	Dette brukermønsteret beskriver hvordan brukeren navigerer informasjonssystemet for å legge en seng til en ansatt.
Forutsetninger	1. Informasjonssystemet er operativt. 2. Instansen informasjonssystemet er aksessert fra håndterer inndata. 3. Den ansatte sengen skal legges til eksisterer. 4. Sengen som skal legges til den ansatte eksisterer.
Normal arbeidsflyt:	
1.	Brukeren aksesserer systemet via smartenhet eller datamaskin.
2.	Brukeren velger hovedmeny.
3.	Systemet presenterer valgalternativer.
4.	Brukeren velger handlingen "Knytt seng til ansatt".
5.	Systemet viser en oversikt over registrerte ansatte.
6.	Brukeren velger den ansatte som en seng skal legges til.
7.	Systemet viser liste over pasienter representert med senger.
8.	Brukeren velger sengen fra listen og velger å lagre.
9.	Systemet knytter sengen til den ansatte og loggfører handlingen.
Alternativ arbeidsflyt:	
2a.	Den ansatte brukeren ønsker å legge pasienten til er tilgjengelig fra oversikten og velges direkte derifra
1.	Systemet presenterer den ansattes profil.
2.	Brukeren velger "Knytt pasient til ansatt".
3.	Systemet fortsetter fra steg 7.

5a.	Hvis ingen ansatte er registrert, eller systemet ikke klarer å aksessere listen over ansatte.
1.	Systemet viser feilmelding, meny lukkes.
7a.	Hvis ingen pasienter er registrert, eller systemet ikke klarer å aksessere listen over pasienter.
1.	Systemet viser feilmelding, meny lukkes.
8a.	Brukeren velger avbryt.
1.	Ingen endringer lagres, meny lukkes.

1.2.1.5.	**Fjerne en seng fra en ansatt**
Beskrivelse	Dette brukermønsteret beskriver hvordan brukeren navigerer informasjonssystemet for å fjerne en seng fra en ansatt
Forutsetninger	1. Informasjonssystemet er operativt. 2. Instansen informasjonssystemet er aksessert fra håndterer inndata. 3. Den ansatte sengen skal fjernes fra. 4. Sengen er knyttet til den ansatte som den skal fjernes fra.

Normal arbeidsflyt:	
1.	Brukeren aksesserer systemet via smartenhet eller datamaskin.
2.	Brukeren velger hovedmeny.
3.	Systemet presenterer valgalternativer.
4.	Brukeren velger handlingen "Fjern seng ansatt".
5.	Systemet viser en oversikt over registrerte ansatte.
6.	Brukeren velger den ansatte som sengen skal fjernes fra.
7.	Systemet viser liste over den ansattes tilhørende senger.
8.	Brukeren velger sengen fra listen og velger å lagre.
9.	Systemet fjerner sengen fra den ansatte og loggfører handlingen.
Alternativ arbeidsflyt:	
2a.	Den ansatte brukeren ønsker å fjerne sengen fra er tilgjengelig fra oversikten og velges direkte derifra
1.	Systemet presenterer den ansattes profil.
2.	Brukeren velger "Fjern seng".
3.	Systemet fortsetter fra steg 7.
5a.	Hvis ingen ansatte er registrert, eller systemet ikke klarer å aksessere listen over ansatte.
1.	Systemet viser feilmelding, meny lukkes.
7a.	Hvis ingen senger er registrert, eller systemet ikke klarer å aksessere listen over senger.
1.	Systemet viser feilmelding, meny lukkes.
8a.	Brukeren velger avbryt.
1.	Ingen endringer lagres, meny lukkes.

Bruksmønster for Seng:

I informasjonssystemet vil de ansatte ha ansvar for senger og ikke pasienter. Sengen blir da være bindeleddet mellom den ansatte og pasienten. Dette valget er grunnet i lovene rundt personvern.

1.2.2.1.	**Legge til ny seng i listen over senger.**
Beskrivelse	Dette brukermønsteret beskriver hvordan brukeren navigerer informasjonssystemet for å legge inn en ny seng.
Forutsetninger	1. Informasjonssystemet er operativt. 2. Instansen informasjonssystemet er aksessert fra håndterer inndata.

Normal arbeidsflyt:	
1.	Brukeren velger handlingen "Legg inn ny seng".
2.	Systemet ber bruker fylle ut den sengens tilhørende infoceller.
3.	Systemet lagrer sengen i listen over senger og loggfører endringen.
4.	Menyen lukkes og skjermen går tilbake til oversikten.

Alternativ arbeidsflyt:		
3a.	Hvis den sengen allerede finnes i systemet.	
1.	Systemet viser en feilmelding med valg: "**Avbryt**", "**Endre seng**".	
2.	**Avbryt**: Ingen endringer lagres, meny lukkes.	**Endre seng**: Brukermønsteret fortsetter fra steg 2.
3b.	Hvis ikke tilstrekkelig info er fylt ut når brukeren prøver å lagre.	
1.	Systemet viser en feilmelding med valg: **Avbryt**, **Endre seng**.	
2.	**Avbryt**: Ingen endringer lagres, meny lukkes.	**Endre seng**: Brukermønsteret fortsetter fra steg 2.

1.2.2.2.	**Fjerne seng fra listen over senger.**
Beskrivelse	Dette brukermønsteret beskriver hvordan brukeren navigerer informasjonssystemet for å fjerne en seng.
Forutsetninger	1. Informasjonssystemet er operativt. 2. Instansen informasjonssystemet er aksessert fra håndterer inndata.

Normal arbeidsflyt:	
1.	Brukeren velger handlingen "Fjerne seng".
2.	Systemet viser en liste over registrerte senger.
3.	Brukeren velger sengen som skal fjernes.
4.	Systemet fjerner sengen fra listen og loggfører endringen.
5.	Menyen lukkes og skjermen går tilbake til oversikten.
Alternativ arbeidsflyt:	
3a.	Brukeren velger avbryt.
1.	**Avbryt**: Ingen endringer lagres, meny lukkes.

1.2.2.3.	**Endre seng.**
Beskrivelse	Dette brukermønsteret beskriver hvordan brukeren navigerer informasjonssystemet for å endre en eksisterende seng.
Forutsetninger	1. Informasjonssystemet er operativt. 2. Instansen informasjonssystemet er aksessert fra håndterer inndata.

Normal arbeidsflyt:	
1.	Brukeren velger handlingen "Endre seng".
2.	Systemet viser en liste over registrerte senger.
3.	Brukeren velger sengen som skal endres.
4.	Systemet ber bruker fylle ut den sengens tilhørende infoceller.
5.	Systemet endrer sengen og loggfører endringen.
6.	Menyen lukkes og skjermen går tilbake til oversikten.
Alternativ arbeidsflyt:	
3a.	Brukeren velger avbryt.
1.	**Avbryt**: Ingen endringer lagres, meny lukkes.

1.2.2.4.	**Legge pasient til en seng.**
Beskrivelse	Dette brukermønsteret beskriver hvordan brukeren navigerer informasjonssystemet for å knytte en pasient til en eksisterende seng.
Forutsetninger	1. Informasjonssystemet er operativt. 2. Instansen informasjonssystemet er aksessert fra håndterer inndata. 3. Sengen eksisterer og er ledig. 4. Pasienten eksisterer i systemet.

Normal arbeidsflyt:	
1.	Brukeren velger handlingen "Knytt pasient til seng".
2.	Systemet viser en liste over registrerte senger.
3.	Brukeren velger sengen en pasient skal knyttes til.
4.	Systemet viser liste over pasienter og tilhørende senger.
5.	Brukeren velger pasient. Systemet legger pasienten til sengen og loggfører endringen.
6.	Menyen lukkes og skjermen går tilbake til oversikten.
Alternativ arbeidsflyt:	
3a.	Brukeren velger avbryt.
1.	**Avbryt**: Ingen endringer lagres, meny lukkes.

1.2.2.5.	**Fjerne pasient fra en seng.**

Beskrivelse	Dette brukermønsteret beskriver hvordan brukeren navigerer informasjonssystemet for å fjerne en pasient fra en seng.
Forutsetninger	1. Informasjonssystemet er operativt. 2. Instansen informasjonssystemet er aksessert fra håndterer inndata. 3. Sengen eksisterer og har en pasient knyttet til seg.
Normal arbeidsflyt:	
1.	Brukeren velger handlingen "Fjern pasient fra seng".
2.	Systemet viser en liste over registrerte senger.
3.	Brukeren velger sengen en pasient skal fjernes fra.
4.	Bruker bekrefter fjerningen når spurt.
5.	Systemet fjerner pasienten fra sengen og loggfører endringen.
6.	**Sengen er nå markert som skitten og beholder egenskapene fra pasienten etter at den er borte.**
Alternativ arbeidsflyt:	
4a.	Brukeren velger avbryt.
1.	**Avbryt**: Ingen endringer lagres, meny lukkes.

1.2.2.6.	**Markere seng som ren.**
Beskrivelse	Dette brukermønsteret beskriver hvordan brukeren navigerer informasjonssystemet for å registrere at en seng har blitt klargjort til en ny pasient.
Forutsetninger	1. Informasjonssystemet er operativt. 2. Instansen informasjonssystemet er aksessert fra håndterer inndata. 3. Sengen er markert som skitten.
Normal arbeidsflyt:	
1.	Brukeren velger handlingen "Rengjort seng".
2.	Systemet viser en liste over registrerte senger.
3.	Brukeren velger sengen som er rengjort.
4.	Bruker bekrefter rengjøringen.
5.	Systemet markerer sengen som ren og loggfører endringen.
Alternativ arbeidsflyt:	
4a.	Brukeren velger avbryt.
1.	**Avbryt**: Ingen endringer lagres, meny lukkes.

Bruksmønster for Pasient

Pasienter skal i all hovedsak hentes direkte fra en liste, men det gis muligheten til å registrere pasienter om det skulle oppstå tekniske feil, eller om en pasient ønsker å være anonym.

1.2.3.1.	**Legge til ny Pasient i listen over Pasient.**	
Beskrivelse	Dette brukermønsteret beskriver hvordan brukeren navigerer informasjonssystemet for å legge inn en ny pasient.	
Forutsetninger	1. Informasjonssystemet er operativt. 2. Instansen informasjonssystemet er aksessert fra håndterer inndata.	
Normal arbeidsflyt:		
1.	Brukeren velger handlingen "Legg inn ny pasient".	
2.	Systemet ber bruker fylle ut den sengens tilhørende infoceller.	
3.	Systemet lagrer personen i listen over pasienter og loggfører endringen.	
4.	Menyen lukkes og skjermen går tilbake til oversikten.	
Alternativ arbeidsflyt:		
3a.	Hvis den pasienten allerede finnes i systemet.	
1.	Systemet viser en feilmelding med valg: "**Avbryt**", "**Endre pasient**".	
2.	**Avbryt**: Ingen endringer lagres, meny lukkes.	**Endre pasient**: Brukermønsteret fortsetter fra steg 2.
3b.	Hvis ikke tilstrekkelig info er fylt ut når brukeren prøver å lagre.	
1.	Systemet viser en feilmelding med valg: **Avbryt**, **Endre pasient**	

2.	**Avbryt**: Ingen endringer lagres, meny lukkes.	**Endre pasient**: Brukermønsteret fortsetter fra steg 2.

1.2.3.2.	**Fjerne pasient fra listen over pasienter.**
Beskrivelse	Dette brukermønsteret beskriver hvordan brukeren navigerer informasjonssystemet for å fjerne en pasient.
Forutsetninger	1. Informasjonssystemet er operativt. 2. Instansen informasjonssystemet er aksessert fra håndterer inndata. 3. Pasienten eksisterer

Normal arbeidsflyt:	
1.	Brukeren velger handlingen "Skriv ut pasient".
2.	Systemet viser en liste over registrerte pasienter.
3.	Brukeren velger pasienten som skal fjernes.
4.	Systemet fjerner pasienten fra listen og loggfører endringen.
5.	Menyen lukkes og skjermen går tilbake til oversikten.
Alternativ arbeidsflyt:	
3a.	Brukeren velger avbryt.
1.	**Avbry**t: Ingen endringer lagres, meny lukkes.

1.2.3.3.	**Endre pasient.**
Beskrivelse	Dette brukermønsteret beskriver hvordan brukeren navigerer informasjonssystemet for å endre en eksisterende pasient.
Forutsetninger	1. Informasjonssystemet er operativt. 2. Instansen informasjonssystemet er aksessert fra håndterer inndata. 3. Pasienten eksisterer.

Normal arbeidsflyt:	
1.	Brukeren velger handlingen "Endre pasient".
2.	Systemet viser en liste over registrerte pasienter.
3.	Brukeren velger pasienten som skal endres.
4.	Systemet ber bruker endre pasientens tilhørende infoceller.
5.	Systemet endrer pasienten og loggfører endringen.
6.	Menyen lukkes og skjermen går tilbake til oversikten.
Alternativ arbeidsflyt:	
5a.	Hvis ikke tilstrekkelig info er fylt ut når brukeren prøver å lagre.
1.	Systemet viser en feilmelding med valg: **Avbryt**, **Endre pasient**.
2.	**Avbryt**: Ingen endringer lagres, meny lukkes. / **Endre pasient**: Brukermønsteret fortsetter fra steg 2.

Bruksmønster for vaktplan

Vaktplanen omhandler for det meste det som vises på tavlen, samt det som skal vises på tavlen senere. Den er delt inn i tre lag, heretter omtalt som "teams", og operer på en 8 timers syklus.

1.2.4.1.	**Forberede en ny vaktplan**
Beskrivelse	Dette brukermønsteret beskriver hvordan brukeren navigerer informasjonssystemet for å legge inn en ny vaktplan som ikke skal implementeres umiddelbart.
Forutsetninger	1. Informasjonssystemet er operativt. 2. Instansen informasjonssystemet er aksessert fra håndterer inndata.

Normal arbeidsflyt:	
1.	Brukeren velger handlingen "Forbered ny vakt".
2.	Systemet ber bruker fylle ut lagene med ansatte.
3.	Systemet lagrer laglistene og systemet loggfører
4.	Brukeren velger om systemet skal automatisk ta i bruk vaktplanen ved neste syklus eller ikke.
5.	Menyen lukkes og skjermen går tilbake til oversikten.

Alternativ arbeidsflyt:		
3a.	Hvis den ansatte allerede finnes i vaktplanen på et annet team.	
1.	Systemet viser en feilmelding med valg: "**Avbryt**", "**Tving endring**".	
2.	**Avbryt**: Ingen endringer lagres	**Tving endring**: Den ansatte flyttes til det nye teamet

1.2.4.2.	**Ta i bruk forberedt vaktplan**
Beskrivelse	Dette brukermønsteret beskriver hvordan brukeren navigerer informasjonssystemet for å ta i bruk en vaktplan som tidligere har blitt klargjort.
Forutsetninger	1. Informasjonssystemet er operativt. 2. Instansen informasjonssystemet er aksessert fra håndterer inndata. 3. Der finnes en forberedt vaktplan

Normal arbeidsflyt:	
1.	Brukeren velger handlingen "Hent vaktplan".
2.	Systemet viser tidligere forberedt vaktplan.
3.	Brukeren velger at vaktplanen skal tre i kraft med umiddelbar virkning.
4.	Systemet endrer aktive vaktplan til den som tidligere var forberedt og loggfører endringen.
5.	Menyen lukkes og skjermen går tilbake til oversikten.

1.2.4.3.	**Endre den aktive vaktplanen**
Beskrivelse	Dette brukermønsteret beskriver hvordan brukeren navigerer informasjonssystemet for å gjøre endringen på den aktive vaktplanen
Forutsetninger	1. Informasjonssystemet er operativt. 2. Instansen informasjonssystemet er aksessert fra håndterer inndata.

Normal arbeidsflyt:		
1.	Brukeren velger handlingen "Endre vaktplanen".	
2.	Systemet ber bruker fylle ut lagene med ansatte.	
3.	Systemet lagrer laglistene og systemet loggfører	
4.	Vaktplanen som vises oppdateres til den nye listen.	
5.	Menyen lukkes og skjermen går tilbake til oversikten.	
Alternativ arbeidsflyt:		
3a.	Hvis den ansatte allerede finnes i vaktplanen på et annet team.	
1.	Systemet viser en feilmelding med valg: "**Avbryt**", "**Tving endring**".	
2.	**Avbryt**: Ingen endringer lagres	**Tving endring**: Den ansatte flyttes til det nye teamet

3.1 Tilbakeblikk og reflektering

Når vi etter ferdigstilt oppgave ser tilbake på overnevnte brukermønster ser vi at det er flere forskjeller på slik vi så for oss brukergrensesnittet skulle se ut, til hvordan det ble. Det samme gjelder også på backend der vi i brukermønstrene går ut ifra at pasientene lastes inn fra en liste fra legevakten, mens i realiteten er systemet vårt helt frittstående og har ingen kommunikasjon med legevakten sitt journalsystem.

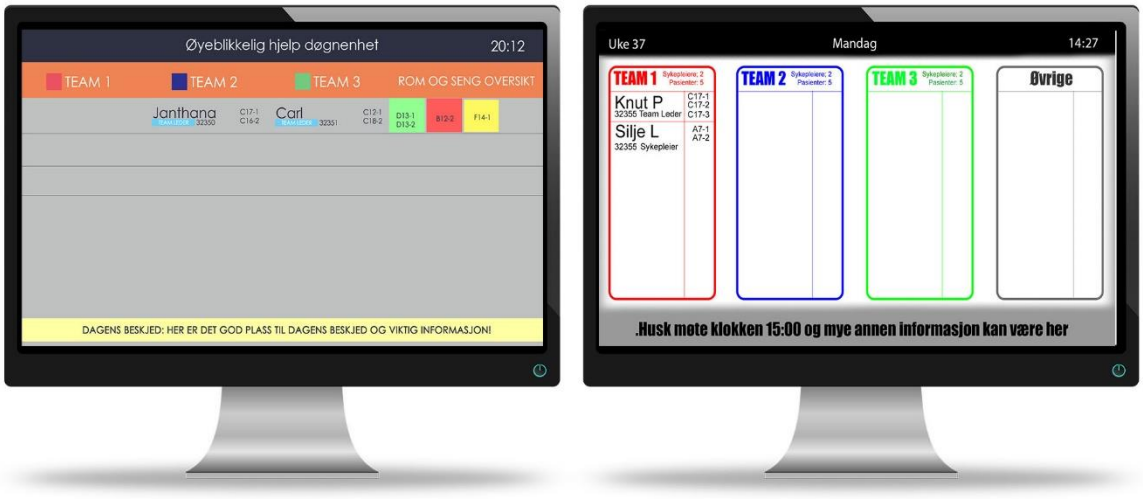
4 Realisering av løsning

Vi har bygget løsningen ut fra tavlen som vises i «Figur 2 Problem», da det er den vi skal erstatte. Vi har bygget opp informasjonen som skal vises basert på hva som skrives på tavlene, og har som målsetning å effektivisere hvordan det gjøres. Dette oppnår vi ved å opprette et digitalt system.

4.1 Design for løsningsforslag

Som løsning på problemstillingen foreslo vi en digital løsning. Løsningen skal håndtere all den nødvendige informasjonen rundt romfordeling hos legevakten. Informasjonen skal være automatisk oppdatert på skjermer alle steder hvor en tavle tidligere ble brukt. I tillegg skal brukerne kunne hente

enten all informasjonen, eller informasjon relevant til seg selv fra en pc eller mobil enhet internt på nettet. Hensikten med systemet er å automatisere flere oppgaver som de ansatte gjør, i tillegg til å forenkle andre oppgaver. Videre skal systemet forhindre feilkilder som dårlig håndskrift og dobbel/trippelregistreringer av informasjon. Systemet skal logge aktivitet, noe som gjør det lettere å finne feilkilder, og vil gi mer data om pasientens behandling.



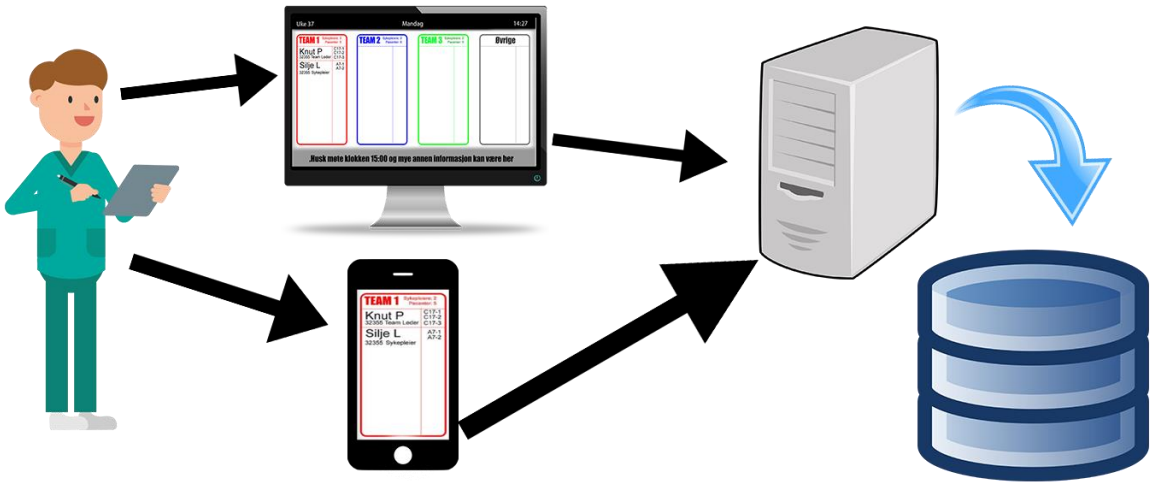
Dette er konsept layouter og ikke nødvendigvis likt vårt endelige resultat*

Figur 3 Her vises tidlig versjon av hvordan layoutene til en løsning kunne ha sett ut.

Systemet skal presentere informasjon om alle ansatte som er på jobb, dette er hovedsakelig sykepleiere, men også leger. På sengepost hos legevakten deles sykepleierne inn i lag, som jobber tettere sammen. Pasientene har egen seng som videre blir knyttet til en ansvarlig sykepleier. Alle ansatte skal derfor kunne legges inn på team, og senger skal kunne knyttes til ansvarlige ansatte. Senger skal kunne markeres med nyttig informasjon som smittefare, ren/skitten, og om pasienten er klar for å dra hjem. Videre skal brukerne kunne bytte på senger og lag, samt endre status på senger. Ved vaktskifte må mange endringer gjøres da det er nye ansatte som kommer på jobb, derfor vil systemet tillate brukerne å forberede et fullstendig skift og gi støtte til forslag av skift.

4.2 Utforming av mulige løsninger

For å oppnå et velfungerende system ut fra kravspesifikasjonen, ser vi noen komponenter som må inngå i løsningen. Dette er et eller flere program som presenterer informasjon, samt lar bruker registrere og endre informasjon. Videre må det være en komponent som tar vare på informasjonen, og distribuerer denne til alle visningsenheter.



Figur 4 Visualisering av en løsning tilgjengelig på tvers av enheter

4.2.1 Løsningsalternativ 1

En server lages for å holde kontroll på data, og denne distribuerer dataen i ønsket format til og fra skjermene, som er et program i Windows, eller eventuelt også Mac. Videre lages det egne applikasjoner for Android og iOS, som også kommuniserer med server på samme måte som skjermene. Til slutt skal server ha tilgang til database for å logge og lagre informasjon. Dette kan være i et journalsystem, eller lokalt.

Denne fremgangsmåten gir gode muligheter til å tilby all funksjonalitet som er ønsket, da man utvikler rett mot operativsystemet til de ulike enhetene. Ulempen med denne metoden er at utviklingen blir ganske omfattede, og krever mange forskjellige teknologier.

4.2.2 Løsningsalternativ 2

En webserver lages lokalt på nettverket, ikke tilgjengelig fra internett. Denne serveren gir data til skjermer og telefoner i samme format. Skjermene og mobilene kjører en HTML basert løsning. Også i dette tilfellet lagres data i et journalsystem eller lokalt.

HTML løsningen vil medføre at det det bare er behov for å utvikle en løsning som vil fungere på alle plattformer. Dermed kan samme brukergrensesnittet tilpasses til både vår mobile og stasjonære løsning. Dette vil spare mye tid, men kan føre til begrensninger, da vi ikke har så lett tilgang på operativsystemenes funksjoner.

4.2.3 Vurderinger av verktøy

For alternativene ovenfor kreves det flere ulike verktøy. For alternativ 1 kan vi i bacheloroppgaven begrense oss til å utvikle for Android og Windows. Det krever at vi bruker Java i Android Studio til å utvikle mobilapplikasjonen, og C# i Visual Studio for å produsere Tavlevisningen. Dette er verktøy vi har erfaring med, og mindre tid vil gå til å lære seg ny teknologi. Men det vil fortsatt være en fordel å undersøke noen rammeverk og biblioteker som gjør jobben lettere.

For det andre alternativet trenger vi HTML, CSS og Javascript, gjerne i tillegg til C#. Her vil brukergrensesnittet skrives med HTML, CSS og Javascript. Serveren kan skrives i C# med ASP.NET CORE, og med nodeJS for å bygge opp nettsidene. Dette er verktøy vi ikke har erfaring med, men verktøy som er utviklet for web utvikling. Dette gjør at det er attraktive verktøy å kunne, og gir stor sannsynlighet for at systemet lett kan bli videreutviklet av andre webutviklere.

4.2.4 Konklusjon

For å realistisk gjennomføre oppgaven, må vi velge en løsning som kan gi oss et fungerende resultat raskt. Derfor velger vi å gjennomføre alternativ 2. Dette innebærer at vi må lære oss veldig mange nye verktøy, men vil gi oss den mest elegante løsningen. Grunnen til dette er at vi ikke ser noe behov for funksjonalitet som bare støttes av native løsninger, og vi ønsker å bruke verktøy som hjelper oss å levere en god løsning raskt. Vi skal utvikle systemet på en slik måte at front-end komponentene kan endres og erstattes med native løsninger i fremtiden, uten at det må gjøres endringer på serversiden. På den måten vil systemet være lett skalerbart.

4.3 Realisering av systemet

Som nevnt Konklusjonovenfor har vi valgt å lage en webserver basert løsning. Dette medfører at vi kan bruke veletablerte verktøy, og ligger godt til rette for videreutvikling. Vi har delt utviklingen inn i 2 hovedkategorier: back-end og front-end. Det skyldes at arbeidet i de kategoriene er nokså ulikt, og har et veldig naturlig skille. Front-end og spesielt back-end deles videre opp i underkategorier hvor arbeidet har ulik form. De ulike delene blir så knyttet sammen igjen på en måte som gjør delene lett utbyttbare.

4.3.1 Back-end

Back-end er gjerne sett på som det som skjer på serveren i et system. (Wikipedia, 2019) Vi har Laget en web server med ASP.NET Core. Fordelen med ASP.NET Core er at det bygges med C# som er et veletablert språk som vi har god kjennskap til. Videre kan da serveren kjøres på andre operativsystemer enn bare Windows, noe som tillater bruk av Linux servere, som ofte er billigere enn Windows. (Heroix, 2018)

4.3.1.1 Arkitektur

Vi har bygget opp løsningen med en hexagonal arkitektur. En hexagonal arkitektur, også kjent som «ports and adapters pattern», som foreslått av Alistar Cockburn i 2005, innebærer at vi har hovedkoden i sentrum av applikasjonen og alle komponenter som database, web API ol. Knyttes til gjennom en API. På denne måten blir komponentene i systemet enkel å bytte ut, og det bidrar til å unngå duplikat kode. Videre samles lignende arbeid i egne komponenter, noe som tillater utviklerne å fokusere på en ting om gangen. (Vuollet, 2018)

Hos oss er business logikken i senter, og gjennom bruk av interfacer bygges resten av systemet rundt dette. Web APIen, som er nærmere forklart nedenfor, kaller metoder i logikken for å initiere prosesser som å knytte sykepleiere til senger ol. Databasen er knyttet til gjennom at logikken gjør kall til en interface som er implementert i en aksesskode mot databasen. Vi har også en algoritme for å beregne forslag til et nytt skift, denne er på samme måte som databasen knyttet til logikken gjennom et interface. Grunnen til at dette ikke inngår i logikken, er for å separere bekymringer. Ved å skjule algoritmen bak en liten API vil den være lett å forbedre i fremtiden, uavhengig av den resterende koden i logikken.

4.3.1.2 Web API

ASP.NET Core er et Cross-Platform, Open-Source rammeverk for å lage moderne applikasjoner med tilknytning til internett. (Microsoft, 2019) Vi har brukt dette til å lage en web API som fungerer som programmets styreenhet for logikken. Det er her brukeren sender inn og henter data i form av JSON objekter. Denne er bygget opp som en RESTful api, men med hovedfokus på GET, POST og DELETE, ettersom objektene som sendes inneholder all nødvendig informasjon.

For å aksessere data fra systemet kan http GET forespørsler sendes, enten det skal hentes informasjon om ansatte eller hentes et fullstendig skift. For å legge inn data brukes POST, og det sendes et objekt med informasjonen som skal endres. Er det en ansatt som skal endres, inneholder objektet den ansattes id og den nye informasjonen som skal lagres. Når objektet når web serveren omformes det til et c# objekt som håndteres av logikklaget. I logikklaget vil objektets id avgjøre om et nytt objekt skal opprettes eller om et eksisterende objekt skal endres. Av den grunn ser vi ikke behov for å implementere PUT og PATCH forespørsler. For krypteringens del passer det også godt å beholde iden i objektet, istedenfor som en del av URLen. Https benyttes for at informasjonen skal sendes kryptert over nettet.

For å gjøre klassene vennlige til å bli sendt over nettet benytter vi model-klasser som tar dataen og omformer den til noe som er enkelt å omforme til JSON objekter, som igjen enkelt kan omformes tilbake til brukbare klasser når det når front-end. Dette bidrar også til å minske mengden data som er sendt, og tar bort bekymringer om format på objekter i logikken.

4.3.1.3 Algoritme

Når et skift skal opprettes må mange ansatte legges inn på lag, og alle sengene må tilegnes de ulike ansatte. For å gjøre denne jobben enklere har vi implementert en algoritme for å beregne den mest sannsynlige fordelingen av roller gitt en liste av ansatte. Videre fordeles også sengene til de ansatte, slik at algoritmen bygger opp et fullstendig forslag til hvordan skiftet skal organiseres.

Algoritmen tar imot en liste av ansatte samt en referanse til en database som kan gi historikken av tidligere skift. I gjengjeld returnerer algoritmen et forslag til skift. På denne måten er ikke algoritmen bunden til logikken, og kan enkelt endres for å ta hensyn til nye ønsker. Det vil også være gode muligheter til å implementere maskinlæring for å gi virkelig gode forslag til brukerne.

Algoritmen er implementer til å gi et enkelt forslag basert på hvem som har hatt hvilken rolle oftest de x antall siste skiftene. Videre balanseres rollene ved bl.a. å forflytte de ansatte som har minst sannsynlighet for å være på et lag, til neste lag for at det skal være en jevn fordeling av ansatte på lagene. Sengene fordeles ikke på et veldig godt grunnlag, men gjør en fornuftig jobb. Algoritmen må videre tilpasses etter ønsker fra brukerne etter hvert som de får erfare behovene sine i løsningen.

4.3.1.4 Database

For databasen er det benyttet en enkel Sqlite database som gjør oppsettet enkelt, men med oppbyggingen av en fleksibel kode vil dette være lett å endre. Videre har vi benyttet Entity Framework til å forenkle arbeidet. Entity Framework er en ORM (Object Relational Mapper) for ADO.NET, dens oppgave er å ta seg av forbindelser med databasen og gjennomføre kommandoer. (Tutorialspoint, 2019) Dette tillatte oss å bruke en «code-first» tilnærming for å opprette, og holde styr på databasen.

For utviklerens del ser databasen ut som velkjente lister med objekter hvor man kan bruke LINQ til å enkelt aksessere data. Dette var en veldig stor fordel for oss da det betydde en mindre ting å lære seg

på kort tid, siden vi slapp å sette oss inn i SQL. Men ulempen med det er at vi trolig ikke har gjort de mest effektive databasekallene ol. Dette er likevel ikke et problem da vi i databaseklassene implementerte en interface som bygger en API for logikken. Med det er utskifting av problematiske metoder en enkel jobb, og en fullstendig utskifting av databaseimplementasjon er fult mulig.

Likt som for web APIen har vi model klasser som omformer objekter til noe som enklere kan lagres. Selv om Entity Framework har støtte for å lagre objekter med referanser, vil det være mye mer komplekst enn å omforme dataen til et mer lagringsvennlig format. Det oppnår de samme fordelene som for web APIen.

4.3.1.5 Logikken

Business logikken ligger sentralt i arkitekturen og er ansvarlig for å holde kontroll på den aktuelle dataen og hvordan den behandles. Når forespørsler kommer gjennom web APIen gjøres det kall til logikken. Det er da logikkens ansvar å benytte alle andre nødvendige komponenter for å gjøre endringer og gi et fornuftig svar tilbake.

Vi har bygget opp logikken som et skift tilsvarende det man kan se på tavlen i Figur 2 Problem. Et skift består av en liste med senger. Sengene har ulike verdier som kan endres gjennom ulike API kall. Sengene har også referanser til en ansatt og en pasient. Videre er det også en liste over ansatte som ikke har ansvar for senger. De ansatte og pasientene har også, som sengene, egne verdier som kan endres med ulike API kall. Dette er grovt sett all dataen som sendes til brukeren når de vil se informasjon relatert til sine ansvarsområder.

4.3.2 Front-end

Front-end sees ofte på som det som skjer på skjermen sluttbrukeren møter. Her bestemmes hva som blir vist, og hvordan brukeren kan interagere med programvaren. Vi har valgt en nettleserbasert løsning for å lettest være tilgjengelig på tvers av enheter og operativsystem. Web-applikasjonen er en React.js-applikasjon som kjøres på en Node.js-server som vil ligge på samme hardware som back-enden.

4.3.2.1 Web-applikasjonen sett fra nettleseren

Web-applikasjonen brukeren besøker består i all hovedsak av tre kodespråk som jobber sammen. **HTML**, *Hypertext Markup Language*, brukes for formateringen og for å strukturere elementer og innhold. Her gis de forskjellige elementene kjennetegn som klasser og id-er, og settes opp i forhold til hverandre.

CSS, *Cascading Style Sheets*, brukes for utsmykningen av elementene. Alt fra plassering, farge og størrelser gjøres via CSS. Det samme gjelder også noen dynamiske hendelser som hvis et element endrer stil når musepekeren flyter over. CSS benytter seg av strukturen gitt i HTML-koden og opererer med en foreldre/barn syntaks.

JavaScript er et høynivå-programmeringsspråk og regnes som en av grunnsteinene i moderne web-utvikling. JavaScript brukes for å kjøre kode i nettleseren i bakgrunnen og er vesentlig for å ha en dynamisk nettside. Dette kan omfavne alt fra kode som håndterer når brukeren interagerer med nettsiden, til enkle skript som holder øye klokken og gir mørkere bakgrunnsfarger på kvelden.

JavaScript kan legge til og fjerne elementer fra nettsiden. Dette er ideelt når vi får når vi mottar data fra server som ikke kan vises direkte på skjermen, men må endres først.

4.3.2.2 Bootstrap

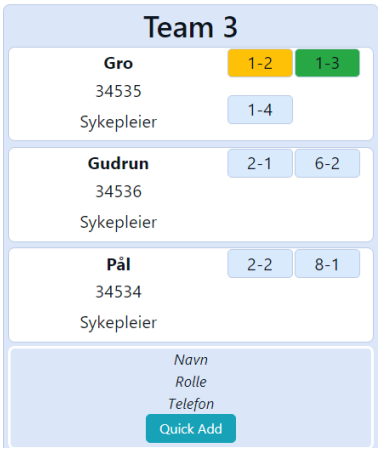
Vi har tatt i bruk *Bootstrap*-biblioteket for å raskt kunne gjøre kosmetiske endringer på web-applikasjonen. Den normale fremgangsmåten er å sette en klasse på et html-element, for og deretter referere til denne klassen via CSS-koden. Derifra setter man stilen alle elementer med samme klasse som referert til skal ta i bruk. Hvis to elementer skal ha forskjellig utseende, må de ha forskjellige klasser.

Bootstrap-biblioteket er lagt opp på en måte der en hel rekke klasser med forskjellige stiler allerede er opprettet for deg, og du setter de klassene som passer på html-elementet. Dette sparer en for mye repetitiv koding. *Bootstrap* kommer i tillegg til CSS-koden, så man vil fortsatt alltid kunne ha egendefinerte klasser utenom.

Det finnes et eget *React-Bootstrap*-bibliotek som er skreddersydd for React.js-applikasjoner, men det har vi gått bort fra ettersom det kreves lisens for å bruke det.

4.3.2.3 React.js

React.js er et JavaScript-bibliotek for å bygge brukergrensesnitt ved hjelp av egendefinerte komponenter. Det er vedlikeholdt av blant annet Facebook, og er spesielt egnet for en enkeltside applikasjon. Vi valgte derfor React.js ovenfor andre lignende løsninger som Anuglar.js.



Figur 5 viser et egendefinert Team-element fra løsningen som inneholder flere egendefinerte Ansatt-elementer, som igjen inneholder egendefinerte Seng-elementer.

En klasse komponent laget i React.js inneholder en status, livs-syklus, kode som skal kjøres mens komponenten renderes, egenskaper fra foreldrekomponenter og en returfunksjon som returnerer JSX-element. Det finnes funksjonelle komponenter i React.js uten status, men de ser vi bort i fra da vi ikke har brukt dem i løsningen vår.

Statusen til React.js komponenten blir brukt for å lagre verdier og objekter hos komponenten selv som skal kunne aksesseres selv etter renderingsfunksjonen er ferdig utført. Feltene som blir lagret i status fungerer mye likt som en global variabel innad i komponenten, og kan nåes fra alle funksjoner og stadier av livs-syklusen.

Egenskaper blir brukt av foreldrekomponenten for å sende verdier, objekter, referanser og funksjoner til barnekomponenten. Dette åpner for kommunikasjon mellom komponentene der

foreldrekomponentene sender informasjon via egenskaper, og barnekomponentene svarer via mottatte funksjoner eller referanser til foreldrekomponenten.

Livs-syklusen til React.js komponenten gjør det mulig å kjøre spesifikke operasjoner ved spesifikke hendelser. Her kan vi bestemme hva som skal skje når komponenten først tas i bruk, og hva som skal gjøres annerledes når den bare endres eller oppdateres.

Renderingen til React.js komponenten skjer når komponenten tas i bruk, og hver gang det oppdages en endring i det som returneres fra komponenten. React.js er elegant på den måten at det er kun den delen av komponentene som endres som renderes på nytt. Oppdateringer fra backend oppleves av erfaring derfor sømløse da bare de nødvendige elementene på skjermen endrer seg, i motsetning til at hele siden skulle ha lastet inn på nytt.

JSX-elementet som returneres fungerer mye likt et html-element. Dette er reflektert i måten React.js komponenter implementeres i appen på. Når vi bestemmer innholdet og strukturen til appens elementer bruker vi React.js komponentene i samkjør med, og på lik måte, som de allerede etablerte html-elementene. Resultatet blir kode som er forvekslende lik vanlig html.

4.3.2.4 Axios

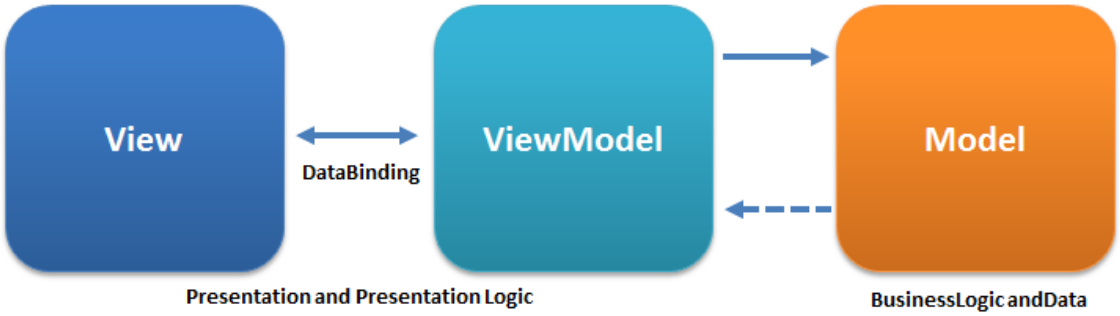
React.js-applikasjonen bruker Axios.js rammeverket for alle HTTP-forespørsler. Axios er et grensesnitt mot REST API som fungerer som en enkel HTTP klient som sender og mottar JSON objekter. Axios er løfte-basert som gjør at vi kan benytte oss av asynkron kode, samt avskjære og kansellere HTTP forespørsler.

4.3.2.5 Arkitektur: MVVM: Model-View-ViewModel

Det har vært litt debatt om hvilket designmønster React.js biblioteket best kan beskrives som. Noen mener React.js-applikasjoner står for View-seksjonen til en MVC arkitektur, andre mener designmønsteret til MVVM er en bedre beskrivelse. Tar vi for oss arkitekturen til MVC, *model-view-controller,* kan det kort oppsummeres til:

- Model: holder på data
- View: viser frem data
- Controller: flytter på data

Sett opp mot strukturen til React.js komponentene ser ikke det ut til stemme ettersom hver komponent holder på data i statusen, og presentasjonen av dataen kan endres basert på den. Vi ser tydelig ved hjelp av figuren under at React.js-applikasjonen følger en MVVM arkitektur.



Figur 6 MVVM designmønster

5 Testing

For å bygge en forståelse av systemets brukbarhet må det utføres ulike tester. Nedenfor beskriver vi noen tester som vi har valgt å bruke for å verifisere programmets brukbarhet. Målet med testene er å bekrefte om systemet fungerer for sitt formål, men også for å lære hva som kan forbedres gjennom fremtidig arbeid.

5.1 Stresstest

For å bekrefte at systemet er brukbart, må det testes om det kan håndtere all trafikken som er nødvendig. Og det må testes hvor kraftig hardware som skal til for at programmet skal kunne kjøres problemfritt kontinuerlig. Kravet til systemet er gitt ut fra det maksimale behovet vi kan se for oss at legevakten skulle trenge, og det er et ganske mildt krav.

Legevakten skal hovedsakelig ha 3 skjermer som er på til enhver tid. Videre er det god mulighet for at noen ønsker å ha programmet tilgjengelig i et vindu. Dette er vinduer som alle skal oppdateres når en endring blir gjort, men som ikke gjør mye ut av seg ellers. I tillegg vil det være mulighet for at flere gjør endringer samtidig, dette kan for eksempel være at noen forbereder et nytt skift, og ber noen andre gå inn å hjelpe dem med oppsettet. Samtidig kan vaskehjelpen markere en seng vasket, og en annen ansatt kan legge inn en ny pasient. Av den grunn ønsker vi å teste med 4 enheter som gjør endringer, med minimum 5 andre enheter som lytter.

For å avgjøre hvor kraftig hardware som er nødvendig for å kjøre systemet vil vi prøve å kjøre serveren på ulike maskiner med varierende prosessorkraft. Vi forventer at systemet skal kunne kjøres på nokså beskjeden hardware, og vil teste på det vi har tilgjengelig.

Vi prøvde å sende mange forespørsler til serveren og å åpne klienten i mange vinduer. Vi har ikke tilgang på så mange enheter til å teste fra, men det vil ikke bli for ulikt å åpne mange vinduer fra samme pc. Serveren kunne håndtere like mange klienter som vi klarte å åpne fra en pc, uten at den fikk problemer med å utføre kommandoer. Vi prøvde også å sette stort trykk på PC-en som kjører serveren. Til tross for at PC-en slet med å åpne nye vinduer og lignende, svarte fortsatt serveren på alle forespørsler. Det eneste vi kunne legge merke til var at forslagsalgoritmen brukte lenger tid på å generere et svar. Dette tyder altså på at vi ikke kommer til å få noen problemer med høyt stress ved en implementering hos legevakten. All hardware vi brukte i testingen var tilstrekkelig til å kjøre serveren.

5.2 Brukertest

For å få tilbakemeldinger på hvordan systemet er å bruke, skal det testes sammen med noen sykepleiere fra Legevakten. Dette gjør vi for å få reelle tilbakemeldinger på brukervennlighet, samt innspill fra brukere med faglig kunnskap.

Sammen med legevakten skal vi gå gjennom hoved funksjonaliteten som beskreven i Brukerdokumentasjonen. For hver funksjonalitet ser vi på brukervennlighet og effektivitet. Sykepleierne får si sin mening og eventuelt komme med forslag. Vi ser også på den visuelle utformingen, for å bekrefte at all nødvendig informasjon er til stede, og om sykepleierne ser behov for noe mer.

Testingen hos Bergen legevakt ga gode resultater på eksisterende funksjonalitet. Videre fikk vi ønsker om utvidet funksjonalitet. Blant annet trenger leger noen spesialmuligheter som ikke sykepleierne trenger, og vi ble enige om noen endringer i hvordan informasjon skal logges. Dette er utbedringer som vil bli del av fremtidig arbeid.

5.3 Tidstest

Når ting tar for lang tid blir totalopplevelsen av systemet dårligere. Derfor må vi teste hvor lang tid ulike prosesser tar. I denne testen vil vi gå gjennom ulike bruksmønstre og vurdere hvor lang tid man generelt bruker for å oppnå målene.

Først skal vi vurdere tiden det tar å legge inn data. Dette innebærer å legge til ansatte, legge til pasienter og legge til senger. Videre skal vi gjøre endringer som å flytte ansatte mellom lag, og bytte senger mellom ansatte. Vi skal også teste endring av informasjon. Det omfatter blant annet endringer på ansattes telefonnumre og sengers statuser. Til slutt skal vi vurdere hvor lang tid det tar å sette opp en ny vakt. Vi skal teste oppsett av en ny vakt manuelt, samt å benytte oss av forslagsalgoritmen og deretter gjøre nødvendige endringer. For å simulere et realistisk resultat vil vi gjøre noen ekstra endringer etter forslagsalgoritmen gir et resultat. Vi skal flytte en ansatt fra et team til et annet, endre et telefonnummer og flytte 2 senger fra en ansatt på et lag til en ansatt på et annet lag.

En bruker med lav til moderat erfaring bruker i snitt 16 sekunder for å fra startsiden opprette en ansatt, pasient eller seng. Pasient tar litt lengre tid å opprette enn resten.

Den kombinerte handlingen å endre en ansatts telefonnummer, sette til teamleder og gjøre ansvarlig for fire forhåndsbestemte senger på en ansatt tok i snitt 40 sekunder.

Oppretting av et helt nytt skift med 9 ansatte og 15 senger, der sengene og pasientene finnes fra før tar i snitt rundt 6 minutter hvis telefonnumrene er de samme som de var forrige gang.

Det samme skiftet med endringene som er beskrevet over tok under 2 minutter i snitt å lage ved hjelp av forslagsalgoritmen.

Vi kan konkludere med at brukeren sparer mye tid ved å bruke systemet. Ansatte som er opprettet blir i systemet helt til de fjernes manuelt, så man slipper å registrere samme personen flere ganger. Vi ser også at når brukeren benytter seg av forslagsalgoritmen tar det bare en tredjedel av tiden det vanligvis tar å opprette et skift, selv når forslaget ikke er perfekt og ekstra endringer må gjøres.

5.4 Varighetstest

For å bekrefte at systemet ikke begynner å slite over tid, må det testes i normalt daglig bruk. For å replisere legevaktens bruk trenger vi at minimum 3 enheter er tilkoblet til enhver tid, samt at minst 3 ekstra enheter kobler seg til og kan gjøre endringer samtidig.

For å teste dette setter vi opp en lokal server som skal kjøre 3 døgn i strekk. I løpet av denne tiden skal vi fullstendig bytte ut skift 3 ganger for dagen: når vi står opp, midt på dagen, og når vi legger oss på kvelden. I tillegg skal vi gjøre endringer underveis, som å legge inn nye pasienter, og bytte ut ansatte.

Målet med denne testen er å bekrefte at systemet ikke får problemer over lengre kjøretider. I tillegg vil det gi oss en mulighet til å oppleve bruken av systemet på en måte som ligner den sykepleierne vil oppleve.

I løpet av denne testen møtte vi ikke på noen problemer. Systemet kjørte fint hele veien, vi prøvde også å starte serveren på nytt og fortsette inntasting av data. Vi opplevde et kort øyeblikk hvor tjenesten var nede, men så fort den var oppe igjen kunne man fortsette der hvor man slapp med all dataen intakt. Et unntak her er skiftet som forberedes, da vi ikke ser behov for at dette lagres før det tas i bruk.

6 Diskusjon

Vi startet arbeidet fra grunnen av ved å designe et system fra en ide. Det har vært en omfattende prosess å gå gjennom alle stegene fra start til slutt, og det er enda mye potensiale for mer arbeid. Vi hadde som mål å bygge et system som kunne effektivisere og gjøre arbeidet på sengepost tryggere. Innenfor rammene til bacheloroppgaven ble vi nødt å avgrense dette til å lage en «proof of concept» løsning som inneholder det meste av funksjonaliteten som Legevakten skulle ønske. Den skal også bygge grunnlaget for en programvare som skal videreutvikles, til det blir et produkt som legevakten kan benytte seg av.

6.1 Samarbeidspartnere

Under hele startfasen av prosjektet har vi jobbet ut ifra at IT-selskapet *Avans* skulle lagre dataen vår i deres journalsystem. Dette har vært planen helt siden konseptualiseringsfasen i innovasjon og teknologiledelse faget *ING101* der behovet for *PasientInfoFlyt* løsningen først ble presentert. Vi har vært på flere møter med *Avans*, både under teknologiledelse faget og i forbindelse med bacheloren.

Planen vi utviklet sammen med *Avans* var å lage et system med hexagonal arkitektur der *Avans* sitt journalsystem skulle være den ene armen. De var villig til å lage en API vi kunne koble oss opp mot for å kunne lagre dataen vår trygt i deres systemer som allerede brukes til taushetsplikts belagt informasjon.

Selv om dette samarbeidet var veldig verdifullt, fant vi ut at en bedre løsning for vår del er å kjøre webserveren lokalt, og ha vår egen database på legevakten. Da slipper vi å sende informasjon som kan tolkes som sensitiv over internettet, og informasjonen blir begrenset til legevakten sitt intranett. Vi kan se for oss et scenario under videreutviklingen av *Verat* der vi kanskje trenger ekstern lagring av data. Hvis vi opparbeider oss flere kunder på forskjellige områder kan vi lettere rettferdiggjøre den arbeidsmengden det kreves for å trygt sende informasjonen over internettet istedenfor. Kanskje *Avans* blir en aktuell samarbeidspartner igjen hvis den tid kommer, og med hexagonal arkitektur er det fort gjort å bytte database.

6.2 Kunden

I vårt tilfelle kan vi se på Bergen Legevakt som kunden vår. De har et behov de ønsker å fylle, og vi vil levere et produkt som gjør dette. Vi har vert på flere møter med Bergen Legevakt hvor vi har samlet informasjon, planlagt og fått tilbakemeldinger på fremgang. Kravspesifikasjonen til oppgaven er utviklet basert på de møtene vi hadde med legevakten under teknologiledelse faget. Utformingen av løsningen er basert på befaringen vi hadde, som gav enda mer innsikt i utfordringene legevakten og vi kom til å ha ved utplassering av systemet.

I de fleste kundeforhold er det normalt å betale for tjenestene man får, i det minste dekke leverandørens utgifter. Ettersom Bergen Legevakt er en offentlig institusjon, er det visse regler som gjelder. Alle kjøp legevakten gjør skal legges ut som anbud, og det er krav om rettferdig konkurranse fra potensielle leverandører (Doffin, 2019). *Verat* er også ikke et etablert selskap enda, så legevakten kan ikke betale oss rett i lommen, på tross av at vi privat kommer til å ha løpende utgifter med drift og vedlikehold.

Den foreslåtte løsningen fra Bergen Legevakt er å ansette oss begge som konsulenter, og dermed legge oss inn i legevaktens lønnssystemer. I tillegg kom det under det siste møtet med legevakten frem at de kan gjøre innkjøp av programvarer og tjenester for opptil 100.000kr uten at det må lyses ut som anbud. Vi er foreløpig i dialog med private ressurser for å få konstruert en kontrakt som vi skal bruke som en del av fremdriftsplanen vår videre, for å benytte oss av disse foreslåtte mulighetene.

6.3 Design

Hvis vi ser bort ifra at vi gikk over til en lokal løsning, har ikke designet sporet særlig av den originale planen noen steder. Dette er nok mye som følger av at løsningen er levert basert på hva legevakten ønsket seg. De har vært konsekvent på hva de trenger og hvilke behov de har, så vi har ikke måtte gjøre noen drastiske endringer på designet vårt underveis.

Målet har alltid vært å lage en løsning som sømløst fungerer på alle enheter. Som følger av vanskeligheter med testing på mobil og nettbrett før den endelige versjonen var klar, har dette blitt nedprioritert. Vi har derfor ikke lagt så mye arbeid i drag-and-drop funksjonalitet, og brukergrensesnittet kan se noe uprofesjonelt ut på mobil til tider, da lange navn strekker seg utenfor beholderne sine. Funksjonalitet på smartenheter, spesielt da nettbrett, er uttrykte ønsker fra kunden selv, så i videreutviklingen vil det bli lagt høy prioritering på å få disse løsningene på plass.

Under demoen vi hadde på legevakten gav de inntrykk av å være svært begeistret for utseende, og brukeropplevelsen av systemet. Dette var første gang de fikk se noe håndfast, og konsensusen var at de var fornøyd, og at de ønsker at vi skal videreutvikle løsningen til det punktet at de kan se for seg å ta den i bruk. Visuelt sett er vi der allerede, men basert på tilbakemeldingene er det et par funksjonaliteter som må på plass før systemet kan utplasseres. Det kom også frem funksjonalitet som de mente kunne være greit å ha, men som ikke var nødvendig å ha på plass før utplasseringen, her foreslo de at de kunne bestille utvikling av funksjonaliteter etter hvert som de ble oppdaget å være nødvendig eller ønsket.

Designet til løsningen er veldig vellykket. Vi har gjort noen mindre avvik fra den initiale planen og kravspesifikasjonen. Vi har valgt å organisere den lagrede informasjonen litt annerledes, og noen av brukermønstrene er endret. Endringene er gjort for å oppnå økt brukervennlighet. Punkt 2.2 «Autentisering» fra kravspesifikasjonen er ikke implementert. Grunnen til at vi gjorde dette valget, er

fordi vi ikke har kommet til enighet om hvordan dette skal implementeres for best mulig opplevelse. Vi skal i møte med IT-avdelingen til Bergen legevakt for å diskutere hvilke krav de stiller. Videre har vi testet med kryptering i henhold til kravspesifikasjon punkt 2.3, men dette vil først bli fullstendig implementert i samband med utplassering av systemet. Likevel oppnår systemet de fleste målene som er satt i oppgaven, og danner et veldig fleksibelt grunnlag for videreutvikling. Legevakten er fornøyd, og da er vi fornøyde.

7 Konklusjon

Vi har startet med et ønske fra Bergen legevakt. Dette gikk ut på å effektivisere noen av prosessene de går gjennom hver dag. Vi har så tatt det videre til å utforme en ide som vi presenterte til legevakten. I denne oppgaven har vi utviklet videre på denne ideen, til et produkt som vil fungere som et «proof of concept». Vi tok oss god tid til å finne ut hvordan vi skulle bygge opp systemet, noe som har lønnet seg. Vi har en løsning som er veldig fleksibel for videreutvikling gjennom bruk av «ports and adapters» arkitekturen. Vi har utforsket samarbeidsmuligheter, men konkludert med at det er best å implementere systemet først. Da kan vi se etter samarbeid etter vi har levert systemet, når vi har mer forhandlingskraft.

Gjennom tester som er gjort i 5 Testing ser vi gode resultater. Selv ved mange tilkoblinger så det ikke ut til at systemet fikk noen som helst problemer. Systemet fungerer over tid, og brukerne var positiv til designet. De fleste funksjonaliteter er raske, men noen sjeldnere brukte funksjoner kan gjøres mer tilgjengelig i fremtiden.

For at programmet skal bli fullstendig brukbart gjenstår det enda noe arbeid. Vi må legge til noen funksjonaliteter som vi har kommet frem til sammen med legevakten, og vi må komme til enighet om en form for autentisering, samt at vi må implementere en form for kryptering som ikke bruker et test-sertifikat.

7.1 Fremtidig arbeid

Vi har som del av oppgaven utviklet en løsning som skal være et forslag til hvordan vi kan løse problemene hos Bergen legevakt. Systemet vi har nå gir et godt inntrykk av hva vi kan tilby legevakten, men er ikke nødvendigvis på det stadiet at vi kan implementere det hos legevakten, og gå videre med livet. For at systemet skal være til god nytte for legevakten må det først testes i bruk, og så videreutvikles basert på tilbakemeldinger fra brukerne. I tillegg kan det oppstå bugs eller lignende, som også krever oppfølging. Legevakten er likevel interessert, og vi er i kontakt for en løsning på et videre samarbeid. Vi har intensjoner om å arbeide videre med løsningen, og tilby legevakten noe de faktisk kan benytte.

Det er mye potensielt arbeid som kan gjøres videre. Først og fremst vil dette være å tilpasse systemet etter kundens ønsker, men også noe vedlikehold burde gjøres. Noe av koden i prosjektet er litt hastet gjennom, som implementasjonen av databasen, denne kan gjerne få en oppfriskning i fremtiden. Andre ting som tilleggsfunksjoner i front-enden som mulighet for «dra og slipp» vil også være veldig relevant.

For å kunne tilby det optimale systemet, vil det være en fordel å kunne legge pasientinfo inn i systemet. På den måten kan vi tilby brukerne alt di trenger i på en plass. Men for å gjøre dette kreves mye arbeid, da pasientinformasjon må lagres i et journalsystem. Å lage dette selv vil være en veldig omfattende jobb, selv om vi skulle ønske å bygge det på toppen av et open-source journalsystem som openEHR. Et annet alternativ kan også være å inngå et samarbeid med en leverandør av journalsystemer. I den anledning har vi vert i kontakt med *Avans* som har vert positiv til muligheten for et samarbeid. Men legevakten i Bergen har planer om å endre journalsystem i nær fremtid, og vi velger derfor å vente med en eventuell slik avtale.

Vi har startet en backlog av oppgaver som vi gjerne ville gjort men som ikke er realistisk å gjøre som del av oppgaven. Denne vil deles inn i sprinter og jobbes videre med etter bacheloroppgaven er gjennomført.

Referanser

Doffin. (2019, Mai 30). *Hovvedside*. Hentet fra Doffinn: https://www.doffin.no/

Heroix. (2018, Mai 16). *Heroix*. (Heroix) Hentet Mai 26, 2019 fra https://blog.heroix.com/blog/linux-vs-windows-a-cost-comparison

HiB, UiB, NHH, UiO og Nasjonalbiblioteket. (2014, 12 12). *Søk og Skriv*. Hentet 12 12, 2014 fra http://sokogskriv.no/

Microsoft. (2019, April 7). *Microsoft*. Hentet fra Microsoft: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.2

Tutorialspoint. (2019, Mai 26). *Tutorialspoint*. Hentet fra Tutorialspoint: https://www.tutorialspoint.com/entity_framework/entity_framework_overview.htm

Vuollet, P. (2018, Oktober 24). *ndepend*. Hentet fra ndepend: https://blog.ndepend.com/hexagonal-architecture/

Wikipedia. (2019, Mai 26). *Wikipedia*. Hentet fra Wikipedia: https://en.wikipedia.org/wiki/Front_and_back_ends

Appendiks A Forkortelser og ordforklaringer

ADO.NET	En data aksess teknologi mellom relational og non-relational system
"Agile" metode	Agile er en metode for utvikling som går mye frem og tilbake mellom stadiene.
API	Application Program Interface – Sett av metoder eller prosedyrer som aksesserer et annet program eller system
ASP.NET Core	Et Open-Source web rammeverk, bl.a. for å lage en Web API
Entity Framework	En ORM til bruk med ADO.NET
Git	Er et versjonskontroll-verktøy.
Github	Er en skylagring for kildekode som brukes for å lagre med git.
Issue	"Problem" direkte oversatt fra engelsk. Brukt om oppgaver som må tas hånd om.
Kanban	Kanban er i hovedsak en organsisert oversikt over oppgaver som må gjøres.
LINQ	Language Integrated Query
MVC	*Model-View-Controller*
MVVM	*Model-View-ViewModel*
ORM	Object Relational Mapper – Konverterer data mellom inkompatible type system
React	Et Javascript bibliotek for å bygge brukergrensesnitt.
RESTful API	API som bruker http requests to GET, PUT, POST og DELETE data.
Scrum	Er en lean prosjektmetode for gjennomføring av prosjekter på en elegant måte.
Sengepost	Øyeblikkelig hjelp avdeling hvor pasienter som ikke kan dra til Haukeland blir innlagt.
SQLite	Relational database administrerings system
Waterfall	Beskriver en metode for software utvikling som følger et lite fleksibelt mønster

Appendiks B Prosjektledelse og styring

Vi er en gruppe på 2 studenter og har derfor valgt å dele på ansvaret. Vi styrer hver vår del av oppgaven og tar felles avgjørelser på ting som omfatter hele systemet, samt generelle avgjørelser rundt systemet.

B.1 Prosjektorganisasjon

Siden vi er 2 på gruppen, har vi fordelt arbeidet i 2: front-end og back-end. Lars har ansvar for back-end og Andreas har ansvar for front-end. For hver del bygger vi en organisert arkitektur, og kobler dem sammen gjennom et API. Videre fordeler vi arbeidet i underkategorier. Back-end er bygget opp rundt naturlige skiller. For eksempel er alt som er relatert til databaser gjemt bak et API, og kan dermed organiseres uavhengig av resten av systemet. Front-end er bygget opp med komponenter.

B.2 Prosjektform

Når vi startet oppgaven brukte vi ikke noen organisert prosjektmetode. I forstudien lagde vi en kravspesifikasjon samt en utdypende oversikt over bursmønstre. Vi tok da utgangspunkt i dette når vi startet utviklingen. Det ble altså noe som kan minne om waterfall. Etter noen uker med arbeid innså vi at denne arbeidsmetoden ikke var tilstrekkelig oversiktlig. Det ble vanskelig å få oversikt over gjenstående arbeid så vel som hva som trengte høyest prioritet. I tillegg var det vanskelig å vite hva den andre på gruppen jobbet med, uten å måtte snakke sammen hver gang.

Prosjektet vårt er programvare, og vi bruker git for versjonskontroll. Vi bruker github til å lagre kildekoden. I github kan man også lage "issues" og prosjekter med automatisert kanban. Vi valgte derfor å gi dette et forsøk. Kanban er i hovedsak en organisert oversikt over oppgaver som må gjøres. Man kan se på dette som en tavle med Post-it lapper i ulike kolonner: «to do», «in progress» og «done». En Post-it-lapp representerer da en oppgave, og den flyttes etter hvert som oppgaven jobbes med. Vi bruker da dette til å fullføre noen funksjonaliteter om gangen, noe som minner mer om en "agile" metode.

Denne arbeidsmetoden fungerte veldig godt for oss, og bidrog til å effektivisere arbeidsprosessen vår. Vi opplevde at det ble mye lettere å se hvilket arbeid som måtte prioriteres, noe som førte til færre flaskehalser i arbeidsprosessen. Vi ser for oss å fortsette å bruke denne prosjektmetoden i fremtiden, og gjerne fullstendig implementere en "agile" metode som "Scrum". På den måten kan vi gjøre videreutviklingen av systemet i sprinter hvor vi bruker noen uker på å ferdigstille et sett med ny funksjonalitet.

B.3 Fremdriftsplan

Som del av forarbeidet satt vi sammen en fremdriftsplan som vist på bildet under. Vi har brukt planen som en guide på hvor mye arbeid som skal være gjort. Vi delte planen inn i et sett med noen mindre mål: 1. Lage et «skall», dvs. at vi har knyttet sammen koden, men ikke fylt inn innholdet. 2. Få på plass nokk til å kunne demonstrere systemet. 3. Bli ferdig med de viktigste punktene på kravspesifikasjonen. 4. Fullføre koden. Med denne oppsplittingen er vi litt fleksible innenfor tidsrammer, og har flere delmål som må jobbes inn mot. Det gjør at vi kan dedikere mer tid til å arbeide om vi ser at vi ligger bak når vi

nærmer oss tid for et mål.

					Uke	w2	w3	w4	w5	w6	w7	w8	w9	w10	w11	w12	w13	w14	w15	w16	w17	w18	w19	w20	w21	w22	w23	w24	w25	
					Man	7/1	14/1	21/1	28/1	4/2	11/2	18/2	25/2	4/3	11/3	18/3	25/3	1/4	8/4	15/4	22/4	29/4	6/5	13/5	20/5	27/5	3/6	10/6	17/6	
					Fre	11/1	18/1	25/1	1/2	8/2	15/2	22/2	1/3	8/3	15/3	22/3	29/3	5/4	12/4	19/4	26/4	3/5	10/5	17/5	24/5	31/5	7/6	14/6	21/6	
#	Aktivitet	Start Dato	Slutt Dato	Fram drift	Ansvarlig								SU?							Påske										
1	Forstudie arbeid	7/1	31/1	100 %	Vi	~200 timer																								
2	Metode undervisning	8/1			Institutt		2t	2t	2t	2t				2t			2t													
3	Forstudie innlevering	31/1		100 %	Vi					◆																				absolutt frist
4	Bachelor oppgave arbeid			100 %	Vi						~800 timer																			
5	Skallet er på plass. GUI og server.			100 %	Vi							◆																		
6	Jobbe med en fungerende demo			100 %	Vi												◆													
7	Midtveis presentasjon	1/4	5/4	100 %	Vi													◆												avtales med veileder
8	Ferdig med minimum viktigste: kravspek-> 2.2			100 %	Vi																	◆								
9	Ferdigstille all programvare			100 %	Vi																		◆	◆						
10	Bachelor oppgave seminar/ Siste UV dag	10/5		100 %	Vi																		◆							10 min/gruppe
11	Eksamen	13/5	11/6	100 %	Vi									L											A					
12	Kun rettskriving ol. igjen etter uke			40 %	Vi																			◆						
13	Bachelor oppgave innlevering	31/5		0 %	Vi																					◆				absolutt frist
14	Bachelor oppgave presentasjon	5/6	12/6	0 %	Vi																						◆			avtales med veileder
	EXPO / Avslutningsfest	13/6		0 %	Vi																							◆		alle må stille

Figur 7 Gant-diagram av fremdriftsplanen

Vi opplevde allerede til første mål at vi var litt bak planen, men det skyldtes at vi måtte sette oss inn i veldig mye ny teknologi. Dette inkluderer ASP.NET Core, HTML, CSS, Javascript, React.js, Databaser mm. For å kompensere for dette jobbet vi litt mer, og hentet oss inn igjen til neste mål. Vi oppdaget også underveis at vi ønsker mer funksjonalitet enn hva som er realistisk å levere. Vi måtte derfor prioritere det som var planlagt i kravspesifikasjonen, og sette opp en backlog på videre arbeid som kan gjøres etter fullført bacheloroppgave.

Appendiks C Brukerdokumentasjon

C.1 Brukerdokumentasjon

Systemet leveres med følgende funksjonaliteter:

Ansatt

Funksjonaliteter

Å opprette ny ansatt:

Klikk på "Quick Add" knappen som befinner seg på ledige plassene på team-panelet.

Velg "Opprett ny ansatt" nede i høyre hjørne av pop-up-en.

Fyll ut tekstfeltene, velg den ansattes stilling i nedslagsmenyen og huk av for teamleder om ønskelig.

Den ansatte er nå lagt til i systemet og er klar for bruk med én gang

Å endre en eksisterende ansatt:

Klikk på den ansatte du ønsker å endre.

Pop-up panelet gjenspeiler den ansattes egenskaper. Utfør ønskede endringer og klikk på "Lagre endringer".

Endringene som har blitt gjort er nå lagt til i systemet.

Å legge til og fjerne senger den ansatte er ansvarlig for:

Den ansatte i seg selv har ingen senger registrert på seg innad i systemet. Denne informasjonen ligger kun hos sengen selv. Derfor baserer denne funksjonaliteten seg på funksjonen for å endre senger.

Klikk på en av de tilhørende sengene, eller høyre halvdel, til den ansatte du ønsker å legge til/fjerne senger den ansatte er ansvarlig for.

Panelet som popper opp viser en liste over alle sengene som er på skiftet. Hvis den ansatte allerede ligger inn med senger vil de være synlig markert i listen, samt være oppsummert under listen.

Klikk på senger for å markere dem, eller klikk på markerte senger for å fjerne markeringen. Når riktige senger er markert klikke du på "Fullfør".

Sengene som var markert har nå blitt endret og den nye ansvarlige ansatte har blitt lagret i systemet. Hvis en seng tidligere var registrert på en annen ansatt vil den fjernes derifra.

Det er ikke mulig å:

Endre navn på den ansatte:

Dette er et valg vi har tatt for å unngå at én ansatt blir brukt av flere personer. Dette kan skape stor forvirring når man skal sjekke hvem som var ansvarlig for hva i den loggete historikken.

Slette en ansatt:

Denne funksjonaliteten er gjort klar og vil implementeres som en del av forbedringene vi må gjøre før legevakten tar i bruk systemet.

Team

Funksjonaliteter

Å legge en ansatt til et team:

Teamet i seg selv har ingen ansatte registrert på seg innad i systemet. Denne informasjonen ligger kun hos den ansatte selv. Derfor baserer denne funksjonaliteten seg på funksjonen for å endre den ansatte.

Klikk på "Quick Add" knappen på teamet du vil legge en ny ansatt til.

Velg en ansatt fra nedlagstavlen.

Huk av for teamleder hvis den ansatte som legges til også skal være teamleder. Hvis en teamleder allerede er valgt vil teamlederstatusen automatisk fjernes fra tidligere teamleder.

Klikk "Fullfør". Den ansatte er nå lagt til teamet.

Denne funksjonaliteten kan også gjøres ved å endre den ansatte sitt tilhørende team.

Å fjerne en ansatt fra teamet:

Teamet i seg selv har ingen ansatte registrert på seg innad i systemet. Denne informasjonen ligger kun hos den ansatte selv. Derfor baserer denne funksjonaliteten seg på funksjonen for å endre den ansatte.

Klikk på den venstre halvdelen av den ansatte du ønsker å fjerne fra teamet.

Klikke på den røde "Fjern fra team" knappen

Den ansatte er nå fjernet fra teamet i systemet.

Det er ikke mulig å:

Legge til/fjerne flere ansatte samtidig:

Dette er en funksjon som vil utvikles i tiden frem mot launch hos legevakten.

Legge til en ny ansatt når alle plassene i et team er fylt opp:

Dette vil det implementeres en løsning for samtidig som funksjonaliteten rundt det å legge til/ fjerne flere ansatte samtidig legges til.

Senger

Funksjonaliteter

Å opprette en ny seng:

Klikk på "Senger" knappen i headeren.

Klikk på den grønne "Opprett ny seng" knappen, helt øverst.

Fyll inn informasjon etter ønske, "Seng id" er påkrevd.

Klikk på "Fullfør". Sengen er nå lagt til systemet og er klar for bruk med én gang.

Å gjøre endringer på en allerede registret seng:

Klikk på "Senger" knappen i headeren.

Klikk på den sengen du ønsker å gjøre endringer på.

Fyll inn informasjon etter ønske.

Klikk på "Fullfør". Endringene er nå lagt til systemet.

Det er ikke mulig å:

Slette en allerede registrert seng:

Dette er ønsket funksjonalitet og er derfor en del av framtidsplanen å legge til.

Endre navn på en allerede registrert seng:

Dette er et valg vi har tatt for å forsikre oss om at den historiske loggen blir forståelig og oversiktlig.

Pasienter

Funksjonaliteter

Det lagres ikke noe informasjon om sengen hos pasienten selv. Dette innebærer at når en pasient knyttes opp mot en seng blir en hjelpefunksjon som endrer på den valgte sengen brukt for å legge pasienten sid id på sengen. Ingen endringer blir altså gjort på selve pasienten.

Å opprette en ny pasient:

Klikk på "Pasienter" knappen i headeren.

I vinduet som popper opp klikker på den grønne knappen "Opprett ny pasient" øverst.

Fyll inn ønsket informasjon og kommentarer. Navn er påkrevd.

Klikk deretter "Opprett pasient".

Pasienter er nå lagret til systemet og er klar for bruk med en gang.

Å gjøre endringer på en allerede registret pasient:

Klikk på "Senger" knappen i headeren.

Klikk på den sengen du ønsker å gjøre endringer på.

Fyll inn informasjon etter ønske.

Klikk på "Fullfør". Endringene er nå lagt til systemet.

Det er ikke mulig å:
Slette en allerede registrert pasient:

Dette er svært ønsket funksjon og har høy prioritering når videreutviklingen av Verat starter etter endt studie.

Neste skift

Funksjonaliteter

Det neste skiftet eksisterer alltid og trenger derfor ikke å opprettes på samme slik som andre elementer. Det neste skiftet finnes under "Nytt skift" valget i headeren. Man kan til enhver tid gjøre endringer på det nye skiftet uten at det påvirker det aktive skiftet.

Å gjøre endringen på det nye skiftet:

Det er ingen avvik i brukermønsteret for å gjøre endringer på det neste skiftet, sett opp mot brukermønsteret for å gjøre endringer på det aktive skiftet.

Å få et nytt skift foreslått av serveren:

Klikk på "Nytt skift" valget i headeren.

På vinduet som dukker opp klikker du på den grønne knappen "Foreslå skift!" øverst.

I listen for ansatte markerer du alle dem som skal jobbe på det neste skiftet.

Klikk på "Fullfør". Listen med ansatte sendes til serveren og blir bearbeidet. Det kan ta flere sekunder før serveren har forberedt skiftet. Her blir det nye skiftet satt i sammen basert på hvordan tidligere skift har blitt satt opp.

Det nye skiftet som serveren foreslår erstatter alt som allerede står på det neste skiftet.

Å ta i bruk skiftet:

Velg "Nytt skift" i headeren øverst.

Det nye skiftet popper opp i et nytt vindu. Klikk på den gule knappen "Ta i bruk skiftet!" hvis du er fornøyd med det nye skiftet.

Det nye skiftet erstatter det aktive skiftet.

Det nye skiftet er nå tomt og klar for neste forberedelse av skift.

Det er ikke mulig å:
Registrere flere skift samtidig:

Dette er en funksjon vi ikke ser for oss at kommer til å være nødvendig, men vil bli tatt opp i samtalen med brukerne. Hvis det viser seg å være en ønskelig funksjon kan utviklingen av denne bestilles.

Legge inn hvilken tidsramme skiftet skal gjelde for:

Dette vil bli lagt inn under videreutviklingen av Verat etter endt studie.

Legge inn tidspunkt for når det nye skiftet automatisk skal tre i kraft:

Her må vi snakke med legevakten og jobbe ut ifra tilbakemeldingene deres om dette er noe de faktisk ønsker. Det kan være brukervennligheten blir dårligere hvis skiftet plutselig endrer seg fordi noen glemte at de hadde lagt inn automatisk endring osv.

C.2 Drifts- og vedlikeholdsdokumentasjon

Systemet skal være tilgjengelig på det lokale nettet på legevakten. Det innebærer at vi trenger en server på lokalet som kan kjøre *asp.net core* med *Internet Information Services*. Undersøkelsen vår peker mot at det skal kunne kjøres på *linux*, men vi har ikke fått testet dette på legevakten. Serveren må også kunne kjøre Node.js for å drive webapplikasjonen, men her jobber vi med en ansatt fra *Websystemer AS* for å mulig få dette til å kjøre på enhetene som benytter seg av webapplikasjonen istedenfor før systemet skal utplasseres.

For å ha en best mulig brukeropplevelse bør vi også samarbeide med IT avdelingen til legevakten for å få lagt inn serveren vår i navnetjeneren deres slik at ønsket trafikk kan bli rutet via menneskelesbare adresser, da helst "verat.no" ettersom dette er døpenavnet til systemet.

Vedlikehold utover det som er normalt for IT-utstyr, støv oppsamling osv., skal ikke være nødvendig. Vi er i opprettelsesfasen av å skrive en kontrakt med legevakten der vi kan fikse eventuelle feil som blir oppdaget, eller som oppstår etter hvert. Minnelekkasje er for eksempel noe som ikke nødvendigvis vil oppdages umiddelbart og er vanskelig å teste for, men som kan bli et problem etter systemet har vært i drift over en lengre periode.

For å sette opp server refererer vi til dokumentasjon fra Microsoft: https://docs.microsoft.com/nb-no/aspnet/core/host-and-deploy/iis/index?view=aspnetcore-2.2 Dette innebærer å konfigurere IIS til å kjøre serveren vår, som vi har publisert til en mappe. For å benytte seg av klienten trenger man en nettleser. I nettleseren kobler man seg til IP-adresse og port fra serveren.

Appendiks D Dokumentasjon av kode

Vi har dokumentert mye av koden med strukturerte kommentarer i koden. På denne måten ligger enkle forklaringer tilgjengelig rett i koden, noe som gjør det enkelt for utviklere å forstå hvordan metoder fungerer.

D.1 Backend

Fullstendig dokumentasjon finnes i eget vedlegg: «Back-end Dokumentasjon». Denne er i html format generert med Sandcastle Help File Builder.

D.2 Frontend

Oversikt over *React.js* komponenter brukt i frontend koden, med noe funksjonalitet:

Forms:

Forms-komponentene er basert på stand html-form elementer, men de skiller seg ut der de kan lagre datafelt i statusene og komponenten vet selv om den er en del av et aktivt eller kommende skift. Her er et eksempel på en POST-request som kjøres når et form sendes:

```
handleSubmit = (event) =>{
    var tempAnsatt = this.state;
    var prep='';
    if(this.props.preppy==="true"){
      prep='prep';
    }
    axios.post(`${webServerHostName}/api/${prep}shift/employee`, tempAnsatt)
```

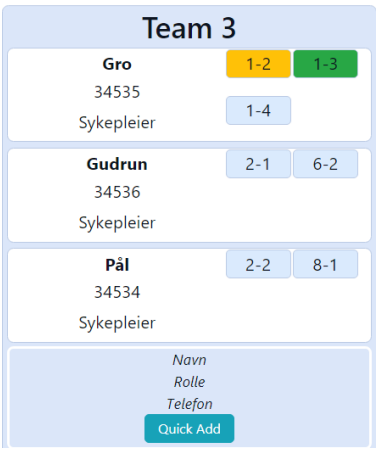
Figur 8 Deler av EndreAnsattForm komponenten sin håndtering av submit.

Det er foreløpig 9 forskjellige forms-komponenter i bruk:

EndreAnsattForm, EndrePasientForm, OpprettAnsattForm, OpprettPasientForm, OpprettSeng, SuggestForm, VelgAnsattFraListe, VelgPasientFraListe, VelgSengFraListe.

Kort:

Kort er komponenter som vises flere ganger ved siden av hverandre og oppfører seg som en blokk. Et kjent eksempel på dette er de individuelle vennene i en venneliste på Facebook.



Hele seksjonen med hvit bakgrunn er en AnsattCard-komponent. Nederst med hvit kant ser man et eksempel på en AnsattCardEmpty-komponent.

De fargete firkantene med tall er sengekomponenter, de er ikke eksempler på "kort".

Modaler:

Modaler er komponenten som popper opp på skjermen og gjør bakgrunnen halvgjennomsiktig. Hver gang man ikke lenger ser det aktive skiftet er man i en eller annen form i kontakt med en modal. Alle modalene trenger hver sin åpne og lukke funksjon.

I løsningen tar vi foreløpig i bruk 7 forskjellige modaler:

AnsattModal, ChangeAnsattModal, ChangeBedModal, ChangePasientModal, PasientModal, PrepModal, SengModal.