



# Accenture Gaming

## **Sluttrapport - Gruppe 43**

Bachelorprosjekt ved OsloMet - Storbyuniversitetet  
Våren 2020

**Asim Abazi**

**Hamza Aftab**

**Jakob Ramstad**



Institutt for Informasjonsteknologi

Postadresse: Postboks 4 St. Olavs plass, 0130 Oslo

Besøksadresse: Holbergs plass, Oslo

PROSJEKT NR.  
43  
TILGJENGELIGHET  
Offentlig

Telefon: 22 45 32 00

## BACHELORPROSJEKT

HovedProsjektets Tittel  Accenture gaming	DATO 25.05.2020
	ANTALL SIDER / BILAG 132
PROSJEKTDeltakere Asim Abazi Hamza Aftab Jakob Ramstad	INTERN VEILEDER Qian Meng

OPPDRAUGSGIVER Accenture Norge AS	KONTAKTPERSON Daniel Meinecke - <a href="mailto:daniel.meinecke@accenture.com">daniel.meinecke@accenture.com</a> Kjell Olav Dale – <a href="mailto:kjell.olav.dale@accenture.com">kjell.olav.dale@accenture.com</a>
--------------------------------------	---

### SAMMENDRAG

Accenture Norge har en intern spillgruppe, som ønsker en løsning for å planlegge og administrere arrangementer. Vi skal utvikle en “event app” for spillgruppa til accenture. Løsningen vil være en mobil applikasjon som kan kjøres på ios og android.

3 STIKKORD
Event App
React Native
Spring Boot

# Forord

Dette dokumentet er sluttrapporten for gruppe 43 sitt bachelorprosjekt. Rapporten vil være optimalisert for digital visning med linker til ulike deler av dokumentet, men kan også leses helhetlig i papirformat. Rapporten som helhet er hovedsaklig rettet mot sluttvurdererne fra oslomet, men består av ulike deler som fyller ulike formål.

Rapporten er delt opp i en presentasjonsdel, prosessdokumentasjon, produktdokumentasjon, brukermanual og en avsluttende del. Presentasjonsdelen gir et kort introduksjon til prosjektet, hva det går ut på, og aktørene involvert. Prosessdokumentasjonen dokumenterer hvordan planleggingen og utviklingen foregikk, samt hvordan vi kom fram til de ulike beslutningene og valgene som ble gjort underveis i prosjektet. Produktdokumentasjonen skal gi en gjennomgående beskrivelse av systemene som har blitt utviklet. Den er i tillegg skrevet med hensyn til eventuelle videreutviklere av systemet.

Til sist vil vi takke de ulike partene som har vært involvert og hjulpet oss gjennom prosjektet. Vår veileder Qian Meng fra Oslomet. Accenture og veilederne våres Lars Henrik Nordli og Daniel Tafjord som bistå med god hjelp og veiledning under prosjektet.

# Innhold

<b>Forord</b>	<b>3</b>
<b>2 Presentasjon</b>	<b>11</b>
2.1 Innledning	11
2.2 Oslomet	11
2.2.1 Accenture	11
2.2.2 Studentgruppen	12
2.3 Bakgrunn	12
2.4 Oppgaven	12
<b>3 Prosessdokumentasjon</b>	<b>13</b>
3.1 Innledning	14
3.2 Kravspesifikasjon	14
3.2.1 Form Workshop	14
3.2.2 Round robin	14
3.2.3 Difficulty matrix	16
3.2.4 Backlog	16
3.2.5 Brukerhistorier	17
3.2.6 Rammebetingelser:	19
3.2.7 Funksjonelle mål:	19
3.2.8 Ikke-funksjonelle mål:	19
3.3 Planlegging og metode	19
3.3.1 Hvorfor smidig?	19
3.3.2 Scrum	20
3.3.3 Kanban	22
3.3.4 Hvorfor Scrum?	22
3.3.5 Tilpasning av scrum og arbeidet underprosjektet	23
3.3.6 Arbeid under sprintene	24
3.3.7 Fasene i prosjektet	25
3.3.8 Risikoanalyse	26
3.3.9 Retrospektiv-risikoanalyse	28
3.3.10 Valg av plattform og løsning	31
3.3.11 Prototyping og utforming av front end applikasjonen	33
3.4 Utviklingsprosessen	36
3.4.1 Sprintene i utviklingsfasen	36
3.4.2 Sprint 1	36
3.4.3 Sprint 2	37
3.4.4 Sprint 3	39
3.4.5 Sprint 4	40

3.4.6 Sprint 5	41
3.4.7 Tiltak som følge av coronavirus	42
3.5 Avslutning	42
<b>4 Produktdokumentasjon</b>	<b>44</b>
4.1 Forord	44
4.2 Introduksjon	44
4.3 Systemarkitektur	45
4.3.1 Serversiden	46
4.3.2 Klientside	47
4.3.3 Overføring av data	47
4.4 Serversiden	47
4.4.1 Filstruktur	48
4.4.2 REST	48
4.4.3 Hovedserveren	52
4.4.4 Spring	56
4.4.5 Spring Boot	58
4.4.6 Object-relational mapping	58
4.4.7 Controller	60
4.4.8 Service	62
4.4.9 Repository	68
4.4.10 Databasemodellen	69
4.4.11 Databasemigrasjon	77
4.4.12 Server-Sent Events	77
4.4.13 Notifikasjonserveren	79
4.4.14 Node.js	81
4.4.15 Notifikasjoner	82
4.4.16 Sikkerhet	85
4.4.17 Admin side	88
4.4.18 Dokumentasjon	88
4.4.19 Serversiden i produksjon	90
4.5 Klientsiden	90
4.5.1 React Native (RN)	91
4.5.2 Expo	91
4.5.3 React.js	91
4.5.4 Arkitektur	91
4.5.5 React-admin	92
4.5.6 API Routes	92
4.5.7 Modal	92
4.5.8 Design	96
4.5.9 Paginering	97

4.5.10 SSE	97
4.5.11 Notifikasjoner	97
4.5.12 Autentisering og autorisering	100
4.6 Verktøy	100
4.6.1 IntelliJ IDEA	101
4.6.2 Visual Studio Code	101
4.6.3 Postman	101
4.6.4 Git Bash	101
4.6.5 Diagrams.net	101
4.6.6 Github Desktop	101
4.7 Forslag til videreutvikling	102
4.7.1 Pagination i frontend	102
5 Testing	<b>102</b>
5.1 Brukertest	102
5.2 Enhetstester	109
<b>6 Brukermanual</b>	<b>111</b>
6.1 Forord	111
6.2 Innledning	111
6.3 Brukerflyt	111
6.3.1 Innlogging og registrering	112
6.3.2 Event og funksjonaliteter	113
6.3.3 Poll og funksjonaliteter	116
6.4 Administrasjon	121
6.4.1 Admin webapplikasjon	123
6.5 Feilmeldinger og advarsler	127
6.5.1 Innlogging	128
6.5.2 Poll	128
6.5.3 Event	128
6.5.4 Admin-webapplikasjon	130
<b>Ordliste</b>	<b>131</b>

# 2 Presentasjon

## 2.1 Innledning

Dette kapitelet vil være en innledende del som vil gi leseren en introduksjon til prosjektet. Først vil vi presentere prosjektets aktører. Deretter vil vi gjøre rede for bakgrunn for oppgaven, problemstillingen og tanker rundt oppgaven.

## 2.2 Oslomet

OsloMet – Storbyuniversitetet er et universitet med hovedcampus i Oslo sentrum. Det fikk universitetsstatus i 2018 og har rundt 20 000 studenter. Institutt for informasjonsteknologi tilbyr de tre bachelorstudiene anvendt datateknologi, informasjonsteknologi og dataingeniør.

Vi fikk tildelt en intern veileder av oslomet, Qian Meng, til å veilede gruppen gjennom prosjektperioden. Qian Meng er førsteamanuensis på OsloMet ved Institutt for informasjonsteknologi. Hun har kompetanse innenfor rekke felt som programvareutvikling, data integrasjon, digitalisering, business intelligence, prosjektledelse, bærekraft, strategi- og innovasjonsutvikling. Hun er medlem av forskningsgruppen Diagnostisk teknologi og bedre medisinerer - DIAMED, som har som hovedmål å utvikle nye og mer effektive løsninger innenfor diagnostisk teknologi og medisinerer.

Qian Meng - [Qian.Meng@oslomet.no](mailto:Qian.Meng@oslomet.no)

### 2.2.1 Accenture

Accenture er et globalt konsern som tilbyr konsulent-, teknologi- og outsourcingtjenester. Accenture har mer enn 469 000 ansatte i 56 land, og selskapet hadde i 2018 en omsetning på 39,6 milliarder amerikanske dollar. Accenture Norge har rundt tusen ansatte og har hovedkontor på fornebu. Gruppen fikk tildelt arbeidsplass på kontoret som kan benyttes gjennom hele prosjektarbeidet.

Fra Accenture fikk vi tildelt to interne veiledere:

- Daniel Tafjord - [daniel.tafjord@accenture.com](mailto:daniel.tafjord@accenture.com)
- Lars Henrik Nordli - [Lars.henrik.nordli@accenture.com](mailto:Lars.henrik.nordli@accenture.com)

### 2.2.2 Studentgruppen

Oppgaven ble utført av en gruppe på tre studenter som går på dataingeniørstudiet ved oslomet.

#### Medlemmene

- Jakob Braseth Ramstad
- Asim Abazi
- Hamza Amir Aftab

## 2.3 Bakgrunn

Det eksisterer foreløpig ingen dedikert løsning for å håndtere planlegging og organisering av arrangement. Dette har tidligere vært håndtert gjennom utsendelse av e-poster og kommunikasjon i microsoft teams gruppe.

## 2.4 Oppgaven

Accenture har gitt oss et prosjekt som handler om å lage en applikasjon for deres interne spillgruppe. De har bestemt at appen skal være todelt. Den ene appen skal holde styr på utlån av utstyr, mens den andre skal være for å holde rede for arrangement som spillgruppen skal holde. Vår gruppe skal ta for oss "events" delen av applikasjonen. Våre første tanker er å produsere en slik app, men med preg av sosiale medier. Appen vil ha et innloggingssystem som vil kreve innloggingsdetaljer knyttet til deres bedrifts-id. Når man logger inn får man opp alle events, det skal være mulig å filtrere og/eller sortere basert på relevante verdier (spill, dato, kategori, osv...). Events skal kunne holde spørreundersøkelser og det vurderes å legges til kommentarfelt for at informasjon skal flyte sømløst. Kjernen av applikasjonen vil være rundt arrangement og hvordan brukere samhandler med disse arrangementene. Tilsvarende skal det også være tillegg av administrator funksjonalitet som skal ha makt til å styre og holde rede.



## **3 Prosessdokumentasjon**

## 3.1 Innledning

I denne delen av dokumentet vil vi ta for oss utviklingsprosessen valg av teknologi, risikoanalyse og arbeidsforhold. Kapittelet starter med en beskrivelse av utviklingsmetodikken vi har benyttet oss av i arbeidet, og hvordan vi har tilpasset denne til vårt prosjekt. Deretter følger en kort beskrivelse av verktøyene vi har brukt til hjelp i planleggingen. Etter dette skriver vi litt om hvordan vi har lagt til rette for at gruppe medlemmene har kunnet tilegne seg den kompetansen de har hatt behov for i løpet av prosjektet.

## 3.2 Kravspesifikasjon

Vi fikk ikke oppgitt noen spesifikk kravspesifikasjon. Utover det skulle være en event app for spillgruppa. Vi utarbeidet derimot en kravspesifikasjon gjennom et FORM workshop kurs som ble arrangert av Accenture 16. januar. I tillegg til samtale med lederen av spill gruppen.

### 3.2.1 Form Workshop

Design thinking består overordnet av tre hovedprinsipper.

Design thinking er menneske sentrert og starter med empati for å forstå mennesker.

### 3.2.2 Round robin

Round robin er en metode der man sammen arbeider med utfordringer og design muligheter, og raskt finner ulike svar og løsninger til disse. Metoden oppmuntrer til mangfoldighet og presisering av ideer og løsninger. I aktiviteten kritiserer og bearbeider man konsepter og ideer iterativt, ved hjelp av ulike perspektiver.

Konkret går metoden ut på at man fyller inn ulike seksjoner på et worksheet i runder. Hver runde fyller alle deltakerne inn en seksjon deretter sendes arbeidsarket til nestemann. Først fyller man inn en "challenge statement", dette kan være en ide eller problemstilling som omhandler et ønskelig mål eller resultat. Deretter skal en annen person fylle ut en foreslått løsning. Dette kan være en skisse eller forklaring på hvordan man kan løse problemet. En ny person skal deretter kritisere løsningsforslaget og finne potensielle svakheter eller grunner til at den vil feile. Neste person skal deretter komme opp med en endelig løsning på utfordringen basert på den tidligere løsningen og kritikken.<sup>1</sup>

---

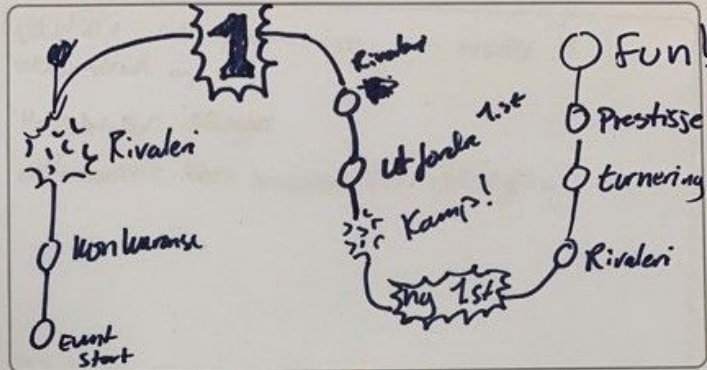
1

#### CHALLENGE STATEMENT

Hvordan kan vi ~~gjøre~~ holde  
systemet interessant for  
Brukerne

#### PROPOSED SOLUTION

Come up with an unconventional way to address the challenge.



#### WHY THE SOLUTION WILL FAIL

Review the proposed solution, and find a reason that it will fail.

This is your chance to be the armchair critic!

- Samme person/personer forblir 1st. Demotiverende dersom noen er overlegne eller ligger langt foran.
- For krevende å implementere. Går ut over viktigere deler av oppgaven

#### FINAL CONCEPT

Review the critique. Then, quickly generate an idea that resolves the issues raised.

Utfordre 1st til et annet spill, slik at det er konkurranse om best spiller og ikke best spiller i et spill.  
Fokuser på de viktigste delene først, deretter ta dette som ekstra funksjonalitet.

### 3.2.3 Difficulty matrix



Figur 3.2 av en difficulty matrix som ble laget under FORM kurs.

En vanskelighet matrise er en matrise som har en akse med vanskelighetsgrad og en akse med viktighetsgrad. En slik matrise tilbyr en gruppe å sette prioriteringer på funksjonaliteter. Det blir også lagt frem mange ideer som hjelper gruppen med å forstå tankegangen til alle medlemmene.

vanskelighetmatrisen vår har blitt sett fra en brukers perspektiv og prosjektets omfang. De funksjonalitetene som en bruker vil sette pris på og de funksjonalitetene som er relevante for prosjektet. Basert på diskusjoner og vanskelighetmatrisen kom vi fram til at noen funksjonaliteter krevde mye og hadde

lav viktighetsgrad. Disse er markert i grå farge. Vi valgte å nedprioritere disse og dersom det var forsvarlig å implementere de skulle vi gjøre det.

### 3.2.4 Backlog

Vi begynte å lage brukerhistorier basert på funksjonalitet vi kom fram til i FORM-kurset. Disse var basert på kravene som vi hadde satt og ble brukt som produkt backlog. Backlogen er en prioritert liste med krav som utgjør utgangspunktet for arbeidsoppgavene i en sprint. Ofte er dette en samling av brukerhistorier som skal implementeres. Ved begynnelsen av hver sprint vil man planlegge arbeid som skal utføres i sprinten basert på backlogen. Backlogen endres kontinuerlig under prosjektet. Innhold og prioritering endres basert på input fra product eier.

### 3.2.5 Brukerhistorier

Brukerhistorier er en metodikk som brukes svært ofte innenfor smidige utviklingsprosjekter. De gjør at man spesifiserer prosjekter og krav ved å utrede funksjonalitet basert på ulike aktørers bruksområder. I en setning klarer man å oppsummere både for hvem vi skal gjøre det, hva som skal fungere og ikke minst hvorfor. Brukerhistorier har formen "Som AKTØR kan jeg GJØRE HANDLING slik at jeg FÅR VERDI".

Administrator - "Som administrator ønsker jeg ... "	Prioritet
Å opprette events for å planlegge et fremtidig event	høy
Å kunne avlyse events for å fjerne events som ikke vil bli gjennomført.	medium
Å kunne endre events for å oppdatere info om events	medium
Å kunne slette brukere for å administrere tilgang	medium
Å kunne endre events for å oppdatere info om events	medium
Å kunne tildele medaljer for å administrere belønninger	lav
Å kunne forfremme nye roller for å administrere	lav

Moderator/Arrangører - "Som moderator ønsker jeg ... "	Prioritet
Å opprette events for å planlegge et fremtidige event	medium
Å kunne avlyse eller fjerne events	medium

<b>Bruker - "Som bruker ønsker jeg ..."</b>	<b>Prioritet</b>
Å se oversikt over ulike events for å se tidligere og kommende events	Høy
Å se oversikt over polls for å se tidligere og aktive polls	Høy
Å lage poll for å holde en meningsmåling.	Høy
Å se resultatet av polls for se hva andre medlemmer mener/ønsker.	Høy
Å kunne avgi stemme på meningsmålinger for å gi min mening.	Høy
Å se detaljer om event for å få vite mer om eventet	Høy
Å logge inn for å få tilgang til applikasjonen	Høy
Å registrere meg som ny bruker for å kunne logge inn på applikasjonen	Høy
Å skrive innlegg på event slik at jeg kan kommunisere med andre som skal på eventet	Høy
Å få notifikasjoner om events slik at jeg blir påminnet om hva og når ting foregår	Middels
å kommentere innlegg slik at jeg kan kommunisere med deltakere	Middels
å legge til bilder i innleggene slik at jeg kan dele øyeblikk	lav
å legge til bilder i innleggene slik at jeg kan dele info	lav
å filtrere events basert på tittel slik at jeg lettere kan finne spesifikke event	lav
å se medaljer slik at jeg kan se hvilke utmerkelser jeg eller andre har.	lav

### 3.2.6 Rammebetingelser:

- Løsningen skal være støttet på iOS & Android.
- Gruppen skal benytte smidig metodikk.
- Løsningen skal testes (hovedsakelig enhetstester)
- Det skal være dokumentasjon nok til at etterkommere kan videreutvikle og/eller vedlikeholde løsningen.

### 3.2.7 Funksjonelle mål:

- Opprette events (spill arrangement)
- Se oversikt over events
- Opprettelse av polls (spørringer)
- Se oversikt over polls
- Innloggingssystem
- Innlegg m/bilde mulighet
- Kommentarer
- Filtrering Events
- Profiler
- Medaljer (oppnåelse/bragd -system)

### 3.2.8 Ikke-funksjonelle mål:

- Intuitivt design
- Brukervennlighet
- Animasjoner
- Sikkerhet

## 3.3 Planlegging og metode

Denne delen vil omhandle planlegging av prosjektet og metodikk. Først vil vi ta for oss vårt prosjekt og hvorfor vi valgte å bruke et smidig rammeverk. Deretter vil vi gå nærmere inn på hvorfor og hvordan vi brukte scrum rammeverket i vår utviklingsmetodikk. Vi vil deretter gå inn arbeidsmetodikk under arbeidet og til slutt planen for prosjektet og de ulike fasene i prosjektet.

### 3.3.1 Hvorfor smidig?

Ut i fra oppgaven som var gitt og kravspesifikasjonen kom vi raskt frem til at vi skulle bruke smidig metodikk under utviklingen. Dette fordi tradisjonelle plandrevne utviklingsprosesser innebærer at hele prosjektet deles opp i separate

faser der man først planlegger alt, så bygger alt. Fokuset på planlegging og dokumentasjon gir mye overhead i mindre prosjekter.<sup>2</sup> I IT-prosjekter er det også mange faktorer som kan endre seg og er vanskelig å forutse som: brukerens behov og interesser, og rammeverk og teknologi som stadig er i endring. Kravene i de fleste IT-prosjekter er komplekse og stadig skiftende. Å forsøke å definere et fullstendig og detaljert sett med krav tidlig i prosjektet ville derfor vært kontraproduktivt og restriktivt.<sup>3</sup>

Smidig metodikk har blitt mer og mer utbredt. Det legger til rette for at man kan lettere endre kravspesifikasjon underveis. Dette fordi man planlegger og definerer krav underveis. I motsetning til å forsøke å følge en fastsatt plan til punkt og prikke, gir dette fleksibilitet i utviklingen. Vi kom derfor frem til at vi skulle bruke smidig metodikk.

### 3.3.2 Scrum

Scrum er et smidig utviklingsrammeverk som er mye brukt i programvare utviklingsprosjekter. Scrum bygger på agile prinsipper og legger til rette for fleksibilitet og effektivt teamsamarbeid. Utviklingsarbeidet i scrum foregår i iterasjoner som kalles en sprint og varer typisk i 2-4 uker. Resultatet av hver sprint utgjør et inkrement i utviklingen. Når en sprint begynner, er varigheten fast og kan ikke forkortes eller forlenges. I scrum er det definert fem ulike hendelser: Sprint, Sprint Planning, Sprint Review, Sprint Retrospective og Daily Scrum.<sup>4</sup> Disse forklares nedenunder.

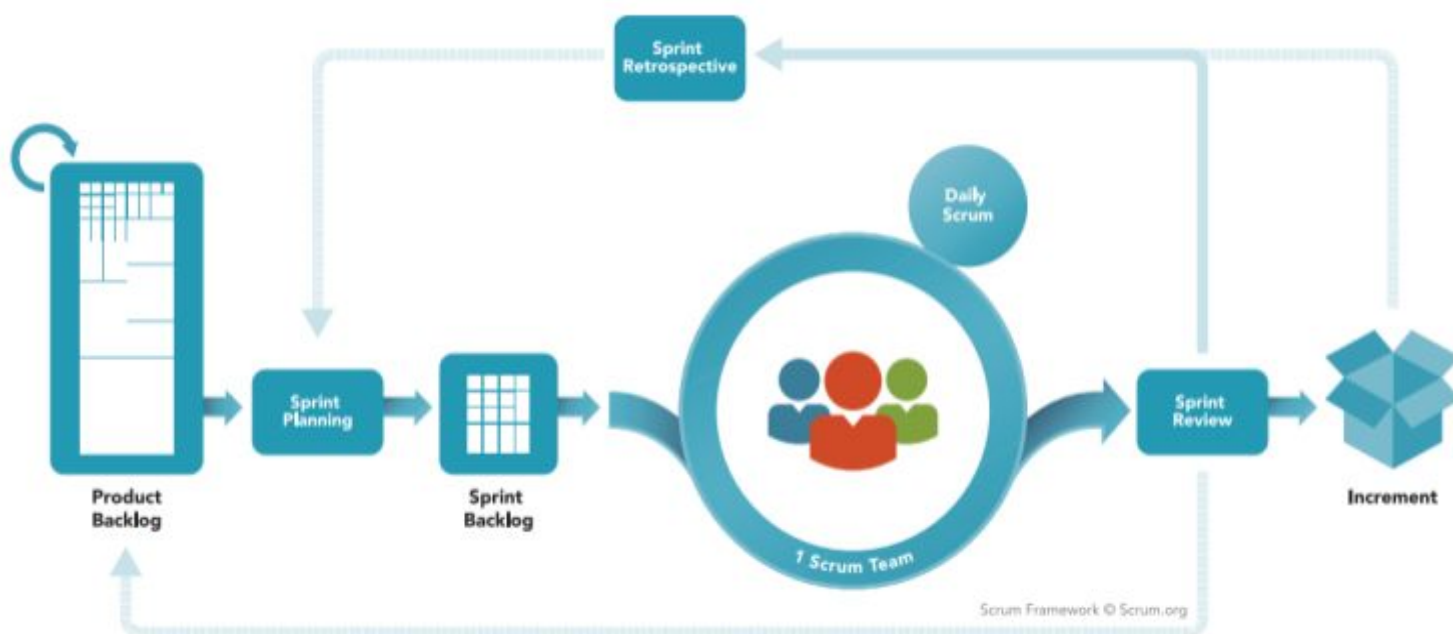
---

<sup>2</sup> Ian Sommerville, Software Engineering 10ed, s75, Agile software development

<sup>3</sup> <https://www.nycon.no/brukerhistorier-er-ikke-en-kravspesifikasjon/>

<sup>4</sup> <https://www.scrum.org/resources/what-is-scrum> What is scrum?, [www.scrum.org](https://www.scrum.org)





Figur 3.3 Scrum rammeverket<sup>5</sup>

### Sprint Planning

Sprint Planning er et møte som foregår før starten på hver sprint, der hele scrum teamet deltar. I møte planlegges arbeidet som skal utføres i sprinten. Dette gjøres ved å velge arbeid fra produkt backlogen som skal ingå i sprinten, og lage en sprint backlog. I møtet etableres også overordnede mål for sprinten.

### Daily Scrum

Daily scrum er et møte som holdes hver dag i sprinten. Dette varer vanligvis rundt 15 minutter og er kun med utviklerne og scrum-master. I møtet gjennomgås vanligvis hva som er gjort, hva man skal gjøre og hindringer man har for å gjøre arbeidet sitt. Møtet gjør at kommunikasjon opprettholdes mellom teamet og at arbeidsstatus og hindringer blir synlige.

### Sprint Review

Sprint Review holdes på slutten av hver sprint. Hele scrum teamet deltar på møtet. På møtet går man igjennom og demonstrerer resultatet av sprinten, og får tilbakemeldinger. Man diskuterer deretter hva som skal gjøres videre og vurderer og bearbeider product backlogen.

<sup>5</sup> <https://www.scrum.org/resources/scrum-framework-poster>

### Sprint Retrospective

Sprint Retrospective holdes etter Sprint Review og gir en mulighet for scrum teamet til å forbedre utviklingsprosessen. Her diskuteres hva som gikk bra i sprinten, hva som kan forbedres og hva som skal gjøres for å forbedre neste sprint.

### Roller i Scrum

I scrum er det definert tre roller. Tabellen under viser de tre rollene med en beskrivelse av de.

Scrum master	<ul style="list-style-type: none"><li>• Leder sprint review, retrospektive møter og det daglige scrum møtet.</li><li>• Sørger for at teamet følger scrum.</li><li>• Jobber for å optimalisere utviklingsprosessen.</li></ul>
Produkteier	<ul style="list-style-type: none"><li>• Representerer kunden.</li><li>• Ansvarlig for å håndtere produkt backlogen og godta fullførte inkremitter.</li></ul>
Utviklingsteam	<ul style="list-style-type: none"><li>• Teamet som utvikler produktet. Gerne 5-9 personer.</li></ul>

### 3.3.3 Kanban

Kanban er et annet rammeverk for smidig utvikling. Det som kjennetegner kanban er at man har en arbeidstavle som man fyller med kort. Hvert kort representerer en arbeidsoppgave. Kortenes plassering på tavlen avhenger av hvor de er i arbeidsflyten. Under utviklingen endres plasseringen til kortene ettersom arbeidsoppgaver blir fullført. Kanban handler om å visualisere arbeidet og maksimere arbeidsflyt og effektivitet. Dette oppnås ved at arbeidet er visualisert i form av kort og dens plassering på kanban brettet, slik at alle kan se arbeidsoppgavene og hvor de er i arbeidsflyten. Arbeidsflyten ivaretas ved at arbeidet ikke tidsbegrenses, igangsatt arbeid minimeres i tillegg til at endringer og leveranse skjer kontinuerlig.<sup>6</sup>

### 3.3.4 Hvorfor Scrum?

Vi valgte å ta i bruk Scrum ettersom det var et kjent og gunstig rammeverk for prosjektet. Samarbeid og engasjement er verdier som er sterkt vektlagt i smidige metodikker. Scrum rammeverket tilbyr en god løsning for å fremme dette. Daily standup er et godt element som fremmer dette og sørger for en fast daglig

<sup>6</sup> <https://www.atlassian.com/agile/kanban> What is kanban?

kommunikasjon blant gruppemedlemmene. Møtet fører til at alle er oppdatert på status og hva som arbeides med av hvert medlem. Det vil ofte også skinne et lys på eventuelle problemer eller hindringer underveis i prosjektet og gjøre det lettere å ta tak i. Hendelsene mellom hver sprint sikrer også tett dialog mellom gruppen og produkteier.

Å dele prosjektet inn tidsboksede sprinter med tilknyttede mål er svært hjelpsomt for å identifisere framgangen i prosjektet. I tillegg tilrettelegger dette for å kunne adressere problemstillinger som oppstår underveis. Dette er viktig da det ofte er mange usikkerheter og hindringer i IT prosjekter. Særlig når man jobber med ny teknologi som man ikke har lang erfaring med, i tillegg til endringer i krav.

Scrum er et rammeverk som vi har lært mye om gjennom studiet, i tillegg til at det er mye brukt i næringslivet. Dette ville derfor gi oss en verdifull erfaring.

### *3.3.5 Tilpasning av scrum og arbeidet underprosjektet*

Scrum er et stort rammeverk som vanligvis utnyttes av et noe større arbeidslag. For oss kunne det virke noe overdrevet å bruke alle elementene i scrum, da vi er et lite team og prosjektperioden er kort. Dette ble også pekt ut av veilederne fra accenture. Vi valgte derfor å ta i bruk de elementene fra scrum som vi mente var mest gunstig for prosjektet vårt, og tilpasse andre.

For å bedre holde oversikten over arbeidet under sprintene tok vi i bruk trello. Her er arbeidsoppgavene visualisert på en tavle i kanban-stil. Dette brukte vi for å holde orden og få en oversikt over status på arbeidsoppgaver i sprint-backlogen og hvem som utførte de under utviklingen.

#### **Roller under prosjektet**

I Scrum så er det tre definerte roller produkteier, scrum-master og utvikler. Siden vi alle kun hadde den samme teoretiske erfaringen om scrum gjennom studiet og kun var 3 personer valgte vi å rullere på hvem som var scrum-master. Lederen av Accenture sin spillegruppe fungerte som produkteier.

#### **Møte med veiledere og produkteier**

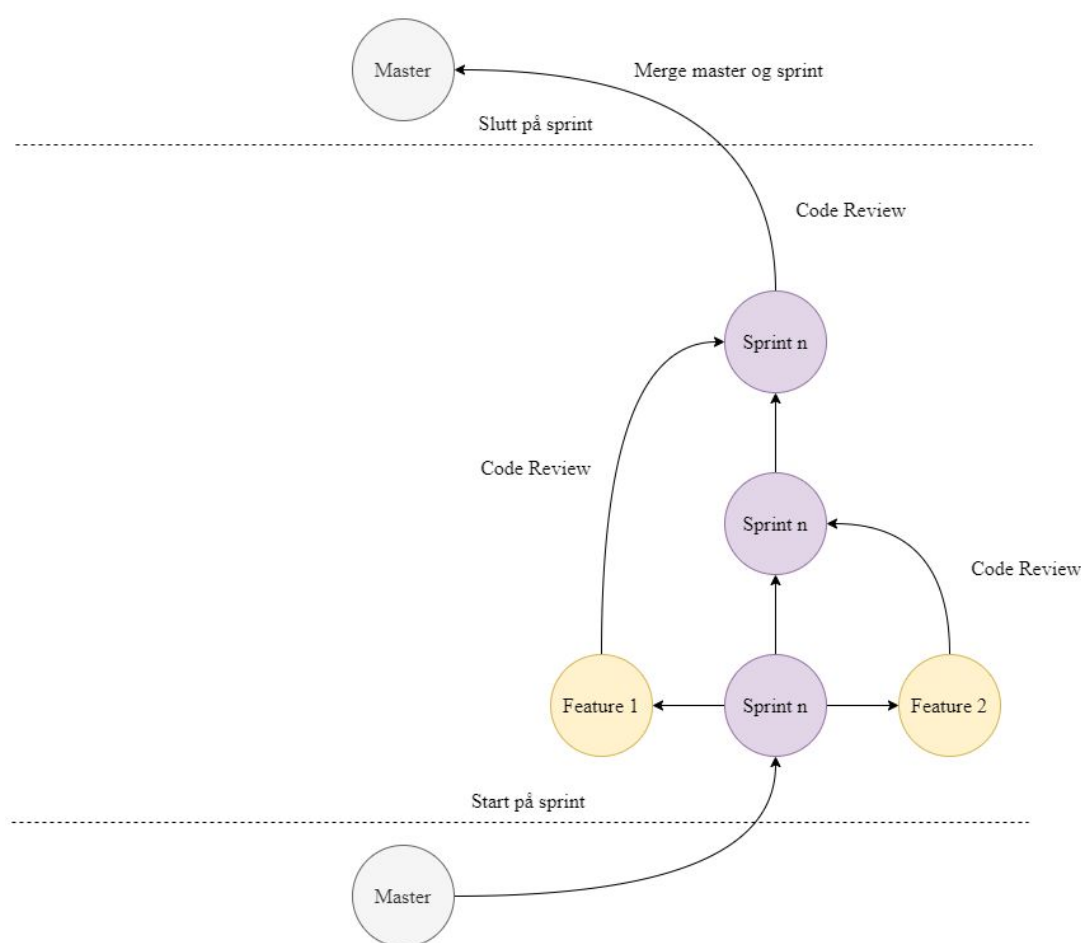
På slutten av hver sprint holdt vi møte med veilederne og produkteier. Under møte presenterte vi resultatene og det vi hadde kommet frem til i sprinten. Veilederne og produkteier kom med mange nyttige tilbakemeldinger underveis på løsningene vi presenterte. Vi diskuterte deretter fremtidig funksjonalitet i appen og prioriterte backlogen. Dette ga oss verdifull input til planlegging av neste sprint.

### Planlegging av sprintene

Før hver sprint møttes gruppa og planla i lag hva vi skulle gjøre i sprinten. I hovedsak består dette av å lage en sprint backlog ut fra produkt backlog-en, og etablere mål for sprinten. På møtet diskuterte vi også teknologier, programvarebibliotek og arkitektur som var relevante for sprinten. Vi ble så enige om arbeidsområdet for gruppemedlemmene i sprinten. Eventuelle endringer i forhold til arbeidet og prosessen tas opp i møtet.

### 3.3.6 Arbeid under sprintene

Under arbeidet brukte vi versjonskontroll verktøyet git. Ved hver sprint lagde vi en ny branch. Ved utviklingen av en ny feature opprettet vi en ny branch for denne featuren. Når denne var ferdig gikk vi igjennom funksjonaliteten sammen før vi pushet denne featuren inn i hoved branchen. Figuren under viser prosessen.



Figur 3.4  
av

prosessen

## Brukersentrert utvikling

Utviklingen under sprintene var hovedsaklig brukersentrert. Vi tok hovedsakelig utgangspunkt i å implementere funksjonalitet basert på brukerhistorier. Trello ble brukt til å visualisere arbeidsoppgavene og vi delte inn arbeidet basert på funksjonalitet. Under utviklingen tok vi blant annet i bruk elementer fra ekstreme programming. Vi brukte en in-memory database h2. Med riktig konfigurering gjorde dette det mulig å kompilere og kjøre backend applikasjonen fortløpende under utvikling. Vi kunne også kompilere og kjøre frontend applikasjonen i expo appen. Dette gjorde at vi kunne bygge hele systemet fortløpende under utviklingen. Backend implementerte vi enhetstester slik at testing kunne utføres før hver kompilering.

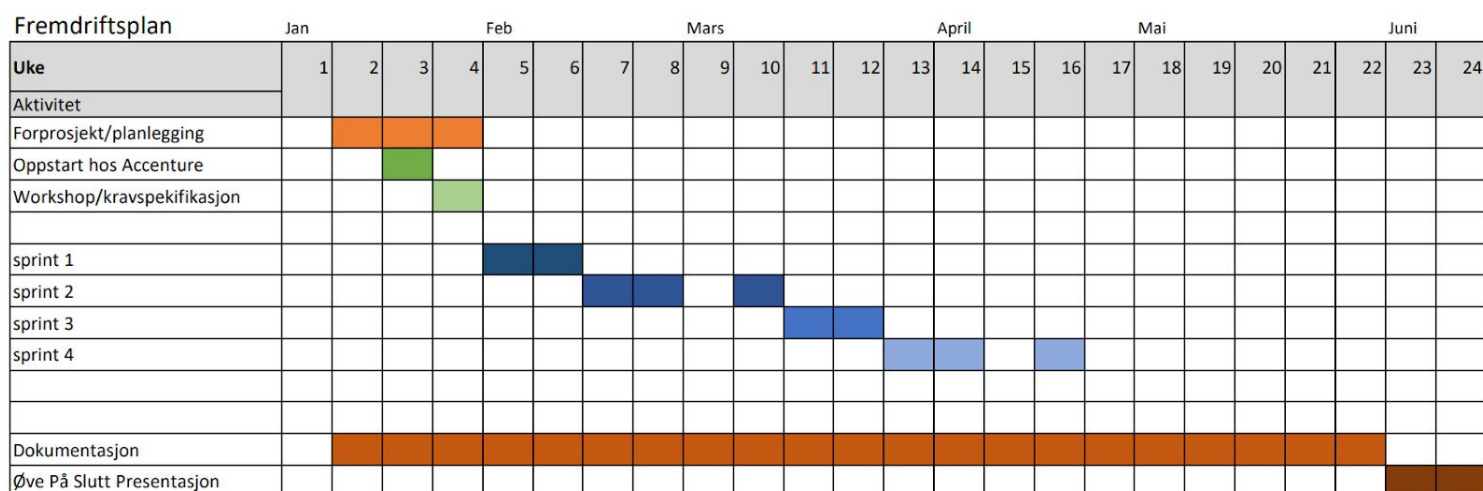
## Arbeidsplan

I planleggingsfasen lagde vi en oversikt over hvor og når vi skulle arbeidet. Vi hadde fått tildelt plasser på accenture sitt kontor i fornebu, og valgte å benytte disse tre dager i uken: mandag, tirsdag og onsdag. Torsdager arbeidet vi på osloMet og hadde også veiledning der noen av dagene. Fredag var avsatt til andre fag og deltidsjobb.

### *3.3.7 Fasene i prosjektet*

Under planleggingen valgte vi å definere tre faser for prosjektet. En planleggingsfase, utviklingsfase og avslutningsfase. Fasene deler opp prosjektet i perioder som omhandler distinktive aktiviteter og oppgaver.

Vi lagde et Gant Diagram som viser planlagt fremdrift for prosjektet, ettersom OsloMet oppfordret til å lage en framdriftsplan i form av et gant diagram. Diagrammet viser de ulike aktivitetene og perioden de ble utført i.

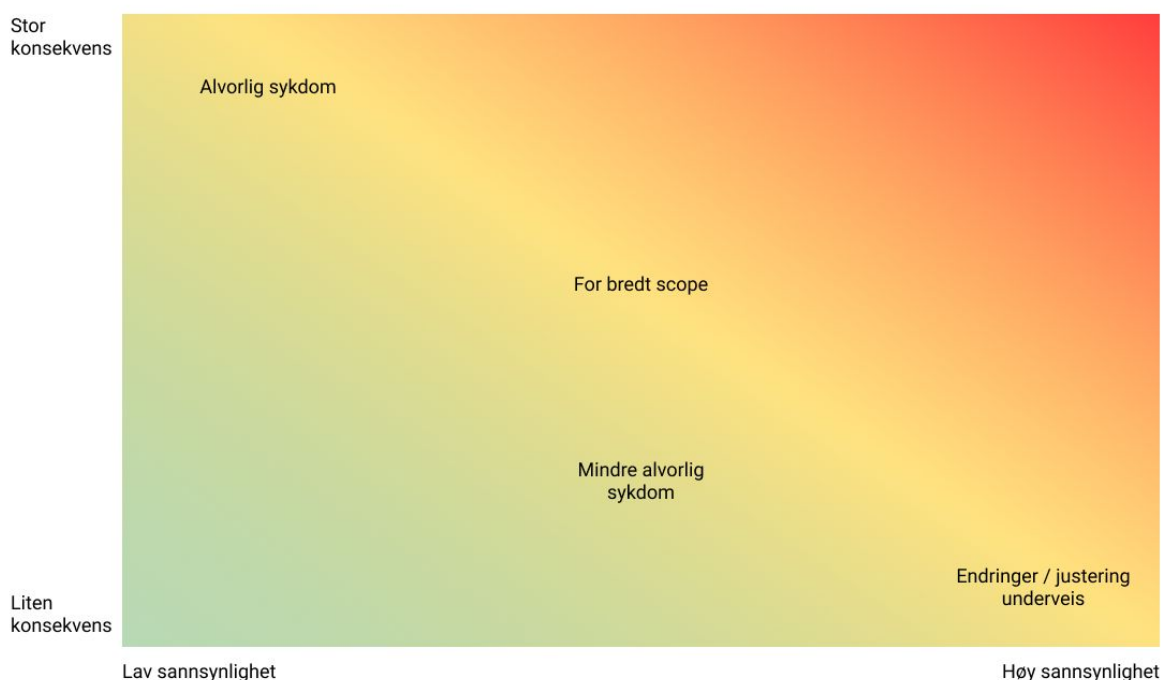


Figur 3.5 Fremdriftsplan - Gant Diagram

Diagrammet viser at vi startet arbeidet i uke to med planlegging og forprosjekt. Deretter hadde vi oppstart hos oppdragsgiver i uke tre og jobbet med kravspesifikasjon og workshop i uke fire. Dette utgjør planleggingsfasen. Videre arbeidet vi i sprints med varighet 2-3 uker, som utgjør utviklingsfasen. Til slutt en avslutningsfase der vi i hovedsak jobbet med dokumentasjon, testing og deployment.

### 3.3.8 Risikoanalyse

Under planlegging av prosjektet valgte vi å lage en risikomatrix for å kartlegge sannsynligheten for forskjellige scenarioer og graden av konsekvens. I tillegg valgte vi å skrive om tiltak som kunne tas i bruk i de forskjellige tilfellene.



Figur 3.6 av risikomatrise

Risiko	Sannsynlighet	Konsekvens	Tiltak
Alvorlig sykdom	lav	høy	Sørge for at alle gruppe­medlemme­ r fortsatt holdes oppdatert angående prosjektet. Finne løsninger for kommunikasjon over nett slik som Discord, Teams eller Skype. Fordele oppgaver slik at ved sykdom kan noen andre ta seg av det.
For bredt scope	middels	middels	Sette prioriteringer på oppgaver og funksjonalitet slik at det produkteier ønsker blir sett på som nødvendig. Snakke med veiledere for å få et bedre omfang av oppgaven.
Mindre alvorlig sykdom	middels	lav	Informere gruppe­medlemme­ r om sykdom og prøve å arbeide der det kan gjøres, kanskje skrive rapport. Satse på kommunikasjon over nett i form av Skype eller Discord.
Endringer/justerin­ ger underveis	høy	lav	Ved bruk av smidig arbeidsmetodikk

			kan konsekvenser av endringer være minimale. Sørge for å god kommunikasjon med produkteier slik at endringer er oppdaterte.
--	--	--	---

### 3.3.9 Retrospektiv-risikoanalyse

Under prosjektet har vi opplevd tre av våre risiko tilfeller. Alvorlig sykdom i form av pandemi skapte uro og hindringer i prosjektarbeidet. Basert på tiltak for risikoen *Alvorlig sykdom* valgte vi å ha møter internt med gruppen over Discord. Vi ble stilt ovenfor andre problemstillinger som hvordan vi skulle holde kontakt med veiledere. Dette ble løst ved å snakke over Microsoft Teams og Discord. I tillegg var det uvisst om hvordan vi skulle kunne teste applikasjonen ettersom at det var satt karantener og smitteregler som måtte følges. Vår valg av teknologi viste seg å være gunstig ettersom at man kunne teste applikasjonen vår over nett med kun en QR-kode. Utfallet av pandemien førte fortsatt til svekket arbeidsinnsats og mangel på motivasjon. Dette var ikke forventet før vi skrev risikomatriksen og dersom det hadde blitt tatt i betraktning kunne motivasjonen og arbeidsinnsatsen blitt tatt hånd om.

Det andre tilfelle vi møtte på var endringer og justeringer av prosjektet underveis. Ettersom at vi valgte en smidig utviklingsprosess var det forventet noen endringer underveis. Utfallet av dette ble at vi måtte endre prioriteringer på forskjellige funksjonaliteter som produkteier ville. God kommunikasjon med både veiledere og produkteier førte til at prosjektets omfang ikke ble for drastisk endret. Det tredje tilfelle av risikomatriksen vi møtte på var mindre alvorlig sykdom. Et gruppemedlem hadde blitt syk og informert gruppen om dette. Gruppen løste dette ved å la det syke gruppemedlemmet skrive på rapporten, mens arbeidsoppgavene ble fordelt på de gjenværende medlemmene.

I etterkant av prosjektet sitter gruppen med en forståelse for hvordan uforventede hindringer kan skape trøbbel og hvor viktig det er med kommunikasjon, planlegging og risikohåndtering fra starten av. Dette er lærdom vi vil ta med oss.



### 3.3.10 Valg av plattform og løsning

Et av de første problemstillingene vi stod overfor var hvilke plattform vi skulle utvikle applikasjonen for. Her var det flere muligheter som alle hadde negative og positive sider. Tabellen under viser en oversikt over fordeler og ulemper for ulike løsninger og plattformer.

Alternativ Løsning	Fordeler	Ulemper
Android App	<ul style="list-style-type: none"><li>• Flere gruppemedlemmer har tidligere erfaring</li><li>• Optimalisert ytelse og brukeropplevelse for plattformen</li></ul>	<ul style="list-style-type: none"><li>• Kun tilgjengelig for de med android.</li></ul>
IOS App	<ul style="list-style-type: none"><li>• Kan ta kortere tid enn å utvikle til android dersom man har lik kompetanse med begge teknologier.<sup>7</sup></li><li>• Optimalisert ytelse og brukeropplevelse for plattformen</li></ul>	<ul style="list-style-type: none"><li>• Vil kun være mulig å bruke for de med IOS.</li><li>• Krever tilgang til Xcode som vi ikke har.</li></ul>
Webapplikasjon	<ul style="list-style-type: none"><li>• Plattformuavhengighet</li><li>• Tidligere erfaring blant alle medlemmer</li></ul>	<ul style="list-style-type: none"><li>• Ingen/lite tilgang til native funksjonalitet som kamera og notifikasjoner.</li><li>• Kan generelt sett virke tregere enn en mobilapplikasjon</li></ul>
Hybrid mobilapplikasjon	<ul style="list-style-type: none"><li>• Kan kjøres på både android og ios.</li><li>• Bygges i rammeverk vanligvis basert på kjent språk og arkitektur fra web utvikling.</li></ul>	<ul style="list-style-type: none"><li>• Krever mye tid for å skaffe ny kunnskap og bli kjent med teknologi/verktøy</li><li>• Funksjonalitet som er avhengig og spesifikk for plattformen kan</li></ul>

7

<https://infinum.com/the-capsized-eight/android-development-is-30-percent-more-expensive-than-ios>  
s Android Development Is 30% More Expensive Than iOS. And We Have the Numbers to Prove It!.

		være begrenset eller ikke støttet.
<b>Web og hybrid mobilapplikasjon</b>	<ul style="list-style-type: none"> <li>• Kan brukes på både mobil, nettbrett og pc.</li> <li>• Størst tilgjengelighet.</li> </ul>	<ul style="list-style-type: none"> <li>• Krever utvikling av to applikasjoner.</li> <li>• Mye arbeid og større tidspress.</li> </ul>

Å utvikle en native mobilapplikasjon var en av løsningene som ble vurdert. En av fordelene med en native app er at man får optimalisert ytelse og brukeropplevelse. Alternativene her var å utvikle til IOS eller Android. Statistikk over markedsandeler for mobile plattformer viser at litt over halvparten av brukerne har iPhone, mens litt under halvparten har android.<sup>8</sup> Å utvikle til en android app ville passet gruppen godt, da to av medlemmene har god erfaring med dette gjennom faget apputvikling hvor vi utviklet android applikasjoner. Ingen av oss hadde noen tidligere erfaring med utvikling til IOS, og vi har heller ikke tilgang til Xcode som kreves for å utvikle native applikasjoner til IOS. Dette passet oss derfor dårlig. Vi konkluderte til slutt med å ikke utvikle en native app fordi den ikke ville vært tilgjengelig for mange av medlemmene.

Å lage en webapplikasjon ble sterkt vurdert. Gruppen hadde tidligere god erfaring med å lage webapplikasjoner og var godt kjent med teknologi som kan brukes til dette. En responsiv nettside vil også være godt tilgjengelig på mobile plattformer. Ulempene var liten eller ingen tilgang til mobil spesifikk funksjonalitet, og at det generelt sett kan trege enn en mobilapplikasjon.

Valget falt til slutt på å lage en hybrid mobilapplikasjon i react native med expo cli. Denne løsningen skal kunne kjøres på både android og ios, og derfor kunne brukes av alle sluttbrukerne. Expo løste mange ulemper som vi hadde sett for oss med en hybrid app. Rammeverket skulle gjøre det mulig å benytte det meste av native funksjonalitetene som vi så på som relevante, som notifikasjoner, enhets posisjon og kamera, uten å måtte lage native moduler til dette til android og IOS.

### 3.3.11 Prototyping og utforming av front end applikasjonen

For å komme fram til et godt design brukte vi i sprint 1 og 2 en del tid på å undersøke lignende applikasjoner. Dette var hovedsakelig for å få inspirasjon til hvordan design og layout i appen skulle være. Dette ga oss inspirasjon til å komme i

<sup>8</sup> <https://gs.statcounter.com/os-market-share/mobile/norway> Mobile Operating System Market Share Norway.

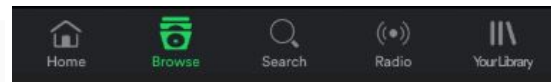
gang med prototyper. Senere i utviklingen var det et mindre behov for dette da vi hadde et godt utgangspunkt å arbeid utifra og kunne endre basert på tilbakemelding fra produkteier, veiledere og brukertesting.

## Navigasjon

Vi så først på hvordan navigasjonen fungerte i andre applikasjoner. Apper vi så på var blant annet facebook,spotify og instagram



Figur - Navigasjonsbaren i instagram



Figur 3.7 - Navigasjonsbaren i spotify

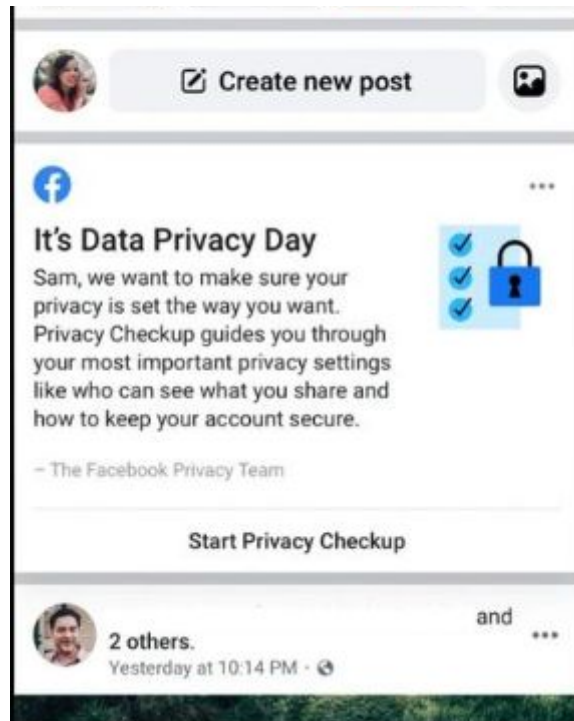
Et mye brukt element var å ha en navigasjons bar nederst på skjermen for å navigere mellom forskjellig innhold. Vi tok dette i bruk allerede i sprint 1 da vi satt opp navigering mellom events og polls og en profil skjerm i applikasjonen. Vår navigasjonsbar vises på bildet under.



Figur 3.8 - Navigasjonsbaren i Accenture Gaming

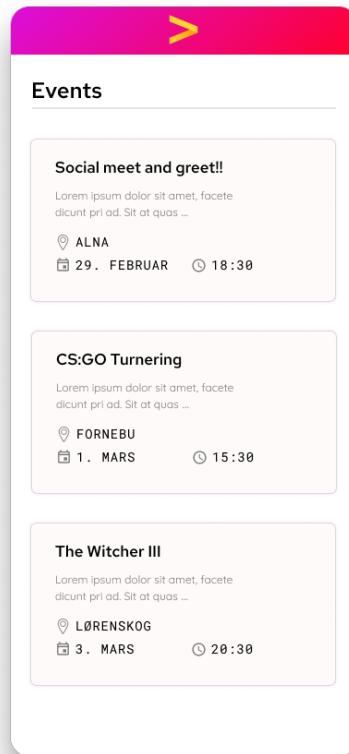
## Visning av events og polls

En sentral del av appen var visning av events og polls. Vi så på apper som facebook og instagram for inspirasjon til dette. Noe som var gjentakende blant flere apper var måten de viste oversikter og samlinger av elementer. Disse kjennetegnes ved at hvert element hadde en lignende utforming og at man kunne dra nedover på skjermen for å se flere elementer. Figuren under viser hvordan dette så ut i facebook med forskjellige typer elementer.

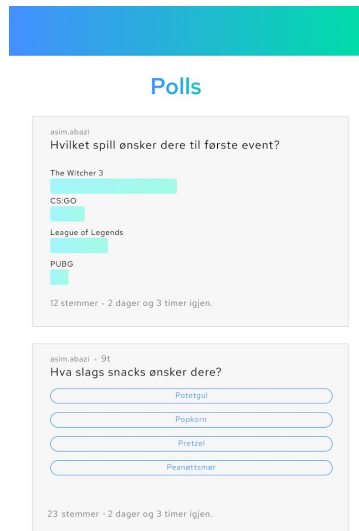


*Figur 3.9 - Facebook feed*

Vi brukte dette til å lage prototyper til visning av events og polls, der et event og et poll tilsvarte et element i samlingen.



Figur 4.0 - viser prototypen for visning av events



Figur 4.1 - viser prototypen for visning av polls

## 3.4 Utviklingsprosessen

Denne delen vil gi en beskrivelse av utviklingsprosessen og prosjektets framdrift fra ide til løsning. Her vil vi først gå nærmere inn på hva vi gjorde i planleggingsfasen og valgene vi måtte ta. Deretter tar vi for oss planlegging og utviklingsarbeidet som ble gjort i hver sprint, og utfordringene vi møtte på underveis.

### 3.4.1 Sprintene i utviklingsfasen

Som tidligere nevnt utførtes utviklingen i sprinter. Vi vil her gi en overordnet beskrivelse av planleggingen av hver sprint og hva som ble gjort i hver sprint. Til hver sprint følger det med en oversikt som viser hvilke funksjonalitet i form av brukerhistorier som sprinten tar utgangspunkt i å utvikle en løsning for, i tillegg til mål for hver sprint. Viktige valg som måtte tas underveis vil også diskuteres.

### 3.4.2 Sprint 1

#### Planlegging

Ved starten av sprint én hadde vi dannet oss en god ide av hva som krevdes av appen. Workshopen og kravspesifikasjonen vi hadde kommet fram til ga oss et godt bilde av funksjonaliteten som kreves av programmet. Vi hadde også kommet fram til mye av teknologien som skulle brukes i prosjektet. Gruppen hadde imidlertid lite erfaring med dette og mye var nytt. Under første sprint måtte vi derfor bruke en del tid på å bli kjent med disse teknologiene og rammeverkene. Vi startet derfor ikke i denne sprinten med å forsøke å implementere funksjonalitet fra backlogen.

Sprint 1	Varighet 14 dager
Mål	<ul style="list-style-type: none"><li>• Modellere en første utgave av database entitetene</li><li>• Oppsett av backend prosjektet</li><li>• Kunne sende request fra en klient til backend</li><li>• Implementere database entiteter</li><li>• Oppsett av frontend prosjekt</li><li>• Bli kjent med teknologien</li></ul>

#### Rollene under sprinten

Rolle/Arbeidsområde	Person
---------------------	--------

Scrum Master	Asim Abazi
Backend	Asim Abazi / Jakob Ramstad
Frontend	Hamza Aftab

## Resultat

I sprinten lærte vi videre om rammeverkene og teknologien vi skulle bruke og ble bedre kjent med dem. Vi fikk satt opp backend applikasjonen med endepunkt som vi kunne kommunisere med via kall fra en klient, samt frontend applikasjonen med enkel navigasjon. Vi lagde også et første utkast av en ER modell i dbdiagram.io og fikk implementert entitetene i backend applikasjonen. Dette ga oss en god oversikt over data modellene som kunne komme til å bli brukt av systemet. Vi fikk dermed et godt grunnlag til å implementere funksjonalitet i neste sprint.

### 3.4.3 Sprint 2

#### Planlegging

I denne sprinten hadde vi som mål å få implementert en rekke funksjonalitet definert i produkt backloggen, da vi hadde et godt utgangspunkt i fra forrige sprint. Vi valgte den funksjonaliteten som var mest sentral og høyest prioritert.

Sprint 2	Varighet 21 dager
Brukerhistorier	<ol style="list-style-type: none"> <li>1. Som bruker ønsker jeg å oversikt over ulike events for å se tidligere og kommende events</li> <li>2. Som bruker ønsker jeg å se oversikt over polls for å se tidligere og aktive polls</li> <li>3. Som administrator ønsker jeg å opprette events for å planlegge et fremtidig event</li> <li>4. Som moderator ønsker jeg å opprette events for å planlegge et fremtidig event</li> <li>5. Som bruker ønsker jeg å lage poll for å holde en meningsmåling.</li> <li>6. Som bruker ønsker jeg å logge inn for å få tilgang til applikasjonen</li> </ol>
Mål	<ul style="list-style-type: none"> <li>• Visning av kommende events i en egen feed.</li> <li>• Visning av polls.</li> <li>• Opprettelse av polls og event i app.</li> <li>• Rollebasert aksesskontroll</li> <li>• Pålogging</li> </ul>

## Rollene under sprinten

Rolle/Arbeidsområde	Person
Scrum Master	Hamza Aftab
Backend	Asim Abazi / Jakob Ramstad
Frontend	Hamza Aftab

## Utfordringer

Under sprinten så vi fort at vi ikke ville rekke å implementere dette fullt ut. React native var utfordrende spesielt for medlemmet som ikke tidligere hadde erfaring med react. Arbeidsmetodikken endret vi derfor fra å ta utgangspunkt i funksjonalitet til å dele inn i arbeidsområde basert på backend og frontend. under sprinten valgte å fokusere på serversiden. Dette var for å tidlig etablere en god struktur i backend applikasjonen med tester og lagdeling. Formålet var at det skulle gå lettere å legge inn ny funksjonalitet og bygge videre på. Vi valgte derfor å være to som arbeidet med backend.

## Resultat

I backend fikk vi på plass de endepunktene som krevdes av den planlagde funksjonaliteten under sprinten. Dette var crud funksjonalitet knyttet til events og polls. Vi fikk også implementert funksjonalitet for autentisering og aksesskontroll. Dette gjorde det mulig å autentisere ved å sende en request fra en klient med enterpriseld og passord, man fikk da returnert en token som brukes til autorisering av videre kommunikasjon.

Fullt ut fikk vi implementert funksjonalitet for å vise en oversikt over events og polls. Hvert event og poll vises som et element i en liste visning i hver sin feed og det brukes en navigasjonsbar til å navigere mellom disse.

## 3.4.4 Sprint 3

### Planlegging

I denne sprinten fokuserte vi først på å fullføre den funksjonaliteten som vi arbeidet med i forrige sprint. Funksjonaliteten for dette var ferdig i backend og vi arbeidet derfor frontend for å ferdiggjøre denne funksjonaliteten.



Vi tok også sikte på å implementere en del ny funksjonalitet. Backend delen av applikasjonen var relativt moden med en god struktur. Dette gjorde den svært utvidbar og ny funksjonalitet var lett å implementere.

Sprint 3	Varighet 14 dager
Brukerhistorier	<b>Fra Sprint 2</b> <ul style="list-style-type: none"> <li>Som bruker ønsker jeg å logge inn for å få tilgang til applikasjonen.</li> <li>Som administrator ønsker jeg å opprette events for å planlegge et fremtidig event</li> <li>Som moderator ønsker jeg å opprette events for å planlegge et fremtidig event</li> <li>Som bruker ønsker jeg å lage poll om events for å se hva andre medlemmer mener og ønsker.</li> </ul> <b>Nye</b> <ul style="list-style-type: none"> <li>Som bruker ønsker jeg å se resultatet av polls for se hva andre medlemmer mener/ønsker.</li> <li>Som bruker ønsker jeg å kunne avgi stemme på meningsmålinger for å gi min mening.</li> <li>Som bruker ønsker jeg å registrere meg som ny bruker for å kunne logge inn på applikasjonen</li> </ul>
Mål	<ul style="list-style-type: none"> <li>Kunne opprette events</li> <li>Kunne opprette polls</li> </ul>

### Rollene under sprinten

Rolle/Arbeidsområde	Person
Scrum Master	Jakob Ramstad
Polls	Asim Abazi
Events og innlogging	Hamza Aftab / Jakob Ramstad

### Resultat og utfordringer

Under sprinten gikk det opp for oss at etter hvert som appen ble brukt kunne det fylle seg opp med svært mange polls og events. Dette kan føre til ytelsesproblemer på lange sikt. Mobil skjermen er også liten og kan kun vise et begrenset antall elementer på et skjermbilde. Det vil derfor være unødvendig å laste inn potensielt flere hundre elementer i liste visningen. Vi valgte derfor å se på løsninger for "paginering". Dette innebærer å blant annet returnere begrenset antall elementer, her events og polls, basert på api kall som sendes under henting av data til sidene med oversikt over events og polls. Etter som brukeren "scroller" nedover på

skjermen vil det gjøres forespørsler til hovedserveren og flere elementer vil vises i oversikten.

En av de mindre utfordringene vi støtte på under sprinten gikk ut på at det hadde skjedd en lettere misforståelse innad i gruppen. React navigation et bibliotek vi brukte for navigasjon hadde kommet med en ny versjon i løpet av sprinten. En av medlemmene som tidligere kun hadde jobbet backend brukte dette under implementering frontend. Problemet lå i at den nyere versjonen hadde introdusert store endringer og var ikke kompatibel med tidligere versjoner. Vi måtte derfor skrive om koden knyttet til denne versjonen på nytt. Etter dette valgte vi å føre en oversikt over biblioteker og rammeverk som viste hvilke versjon vi brukte. Vi fikk ellers implementert ferdig den planlagte funksjonaliteten under sprinten.

### 3.4.5 Sprint 4

#### Planlegging

Under tidligere møte med produkteier hadde det kommet fram at det var ønskelig med en web applikasjon for å lage events. Dette fordi det kunne bli tungvint å skrive mye informasjon om eventene på en mobil. Vi valgte derfor å utvikle en web applikasjon med denne funksjonaliteten og en rekke annen funksjonalitet for å administrere løsningen. Dette ble utført av et medlem av gruppen som hadde tidligere erfaring med web applikasjoner i react. Resten av gruppen fortsatte implementeringen av funksjonalitet i applikasjonen.

Sprint 4	Varighet 21 dager
Brukerhistorier	<ul style="list-style-type: none"><li>Som bruker ønsker jeg å se detaljer om event for å få vite mer om eventet.</li><li>Som bruker ønsker jeg å skrive innlegg på event slik at jeg kan kommunisere med andre som skal på eventet</li><li>som bruker ønsker jeg å legge til bilder i innleggene slik at jeg kan dele info</li><li>Som administrator ønsker jeg å</li><li>Som bruker ønsker jeg å få notifikasjoner om events slik at jeg blir påminnet og varslet om events.</li></ul>
Mål	<ul style="list-style-type: none"><li>Fullføre visning av event detaljer og funksjonalitet for innlegg i eventer.</li><li>Fullføre støtte for notifikasjoner ved nye events</li></ul>

## Rollene under sprinten

Rolle/Arbeidsområde	Person
Scrum Master	Asim Abazi
Events	Jakob Ramstad
Administrasjons løsning	Asim Abazi
Notifikasjoner	Hamza Aftab

## Resultat og utfordringer

Under sprinten så vi at det ikke ble tid til å gjøre web løsningen helt ferdig. Vi valgte derfor å legge til en sprint til for å fullføre denne med den planlagde funksjonaliteten. For at dette ikke skulle gå utover dokumentasjonen og sluttrapporten var det kun et medlem som arbeidet med denne i neste sprint, mens de andre fokuserte på dokumentasjonen. Det ble heller ikke tid til å implementere funksjonalitet for å legge inn bilder i innlegg. Dette var imidlertid ikke en høyt prioritert funksjonalitet og vi valgte å ikke innføre noen tiltak.

Vi fikk implementert en egen skjermvisning av event detaljer når man trykker på et event i listen. Man kan også publisere og lese innlegg fra andre medlemmer under eventet. Det ble også implementert støtte for notifikasjoner, slik at brukere får notifikasjon når nye eventer ble opprettet.

### 3.4.6 Sprint 5

Sprint 5	Varighet 14 dager
Brukerhistorier	<ul style="list-style-type: none"><li>Som administrator ønsker jeg å kunne avlyse events for å fjerne events som ikke vil bli gjennomført</li><li>Som administrator ønsker jeg å kunne endre events for å oppdatere info om events</li></ul>
Mål	<ul style="list-style-type: none"><li>Fullføre administrering delen av applikasjonen</li></ul>

Under sprinten fikk vi fullført web løsningen. Den funksjonaliteten som var planlagt og høyest prioritert ble implementert i løsningen.

### 3.4.7 Tiltak som følge av coronavirus

Korona-viruset også kjent som Covid-19 er en pandemi som påvirket Norge på mange måter. Dette har gått ut over hvordan vi som en gruppe har løst vår oppgave.

En konsekvens var at vi ikke lenger kunne møtes fysisk på accenture og oslomet. Vi fikk mail fra vår kontaktperson Kjell Olav 13. mars om at vi ikke lenger kunne jobbe og møtes på accenture. Samme uke stengte også oslomet for studenter.<sup>9</sup> Gruppen måtte derfor samarbeide over nett.

Gruppen avtalte å holde de daglige møtene over discord eller Microsoft Teams. Discord og Microsoft teams er programvare som legger til rette for kommunikasjon mellom brukere i kanaler i form av tekst, lyd, bilde og video. Dette var gunstig da discord hadde en god løsning for skjermdeling, slik at vi lett kunne vise til kode over nett og samarbeide. Møtene med veileder og produkteier ble videre utført over Microsoft teams og Discord.

## 3.5 Avslutning

Prosessdokumentasjonen tar for seg planleggingen og arbeidet gruppen har vært igjennom under bachelorprosjektet. Vi har kommet fram til en god løsning som vi mener oppfyller krav og løser problemstillinger rundt det å arrangere events for spillgruppen.

Under prosjektet har arbeidsprosessen vært smidig rammeverk. Vår tilpasning av scrum og planlegging av arbeidet ga oss en god struktur på prosessen. Faste møter og definerte arbeidsmetoder førte til et godt samarbeid og god kommunikasjon i gruppa. Dette la til rette for en produktiv utvikling. Av krav og funksjonalitet definert i kravspesifikasjonen har vi fått implementert det som skal gi høyest verdig til sluttbrukerne gjennom kontinuerlig prioritering underveis i prosjektet.

Samarbeidet under prosjektet har vært sterkt og kritisk under arbeidet. Kommunikasjon i gruppa og med veiledere har vært viktig under arbeidet for å komme fram til et produkt med verdifull funksjonalitet. Dette var også kritisk for å håndtere de problemstillingene som kommer opp underveis.

---

<sup>9</sup> <https://student.oslomet.no/en/siste-nytt/-/nyhet/koronaviruset-og-studenter-i-praksis>

# 4 Produktdokumentasjon

## 4.1 Forord

Formålet med produktdokumentasjon delen er å gi leseren teknisk innsikt i hvordan systemet fungerer. Mye av denne delen vil inneholde detaljer og resonnementer for hvordan konkrete deler av systemet funker. Dette er for å gi leseren en god forståelse av produktet og en pekepinne til hvordan man burde utvikle produktet videre.

Det er forutsatt at leseren har kunnskaper innen objekt orientert programmering. Kunnskap innenfor java og javascript forutsettes også. Leserens burde ha forståelse for relasjonsdatabaser. Som følge av dette egner denne delen seg mer for vedlikeholdere og videreutviklere av produktet.

## 4.2 Introduksjon

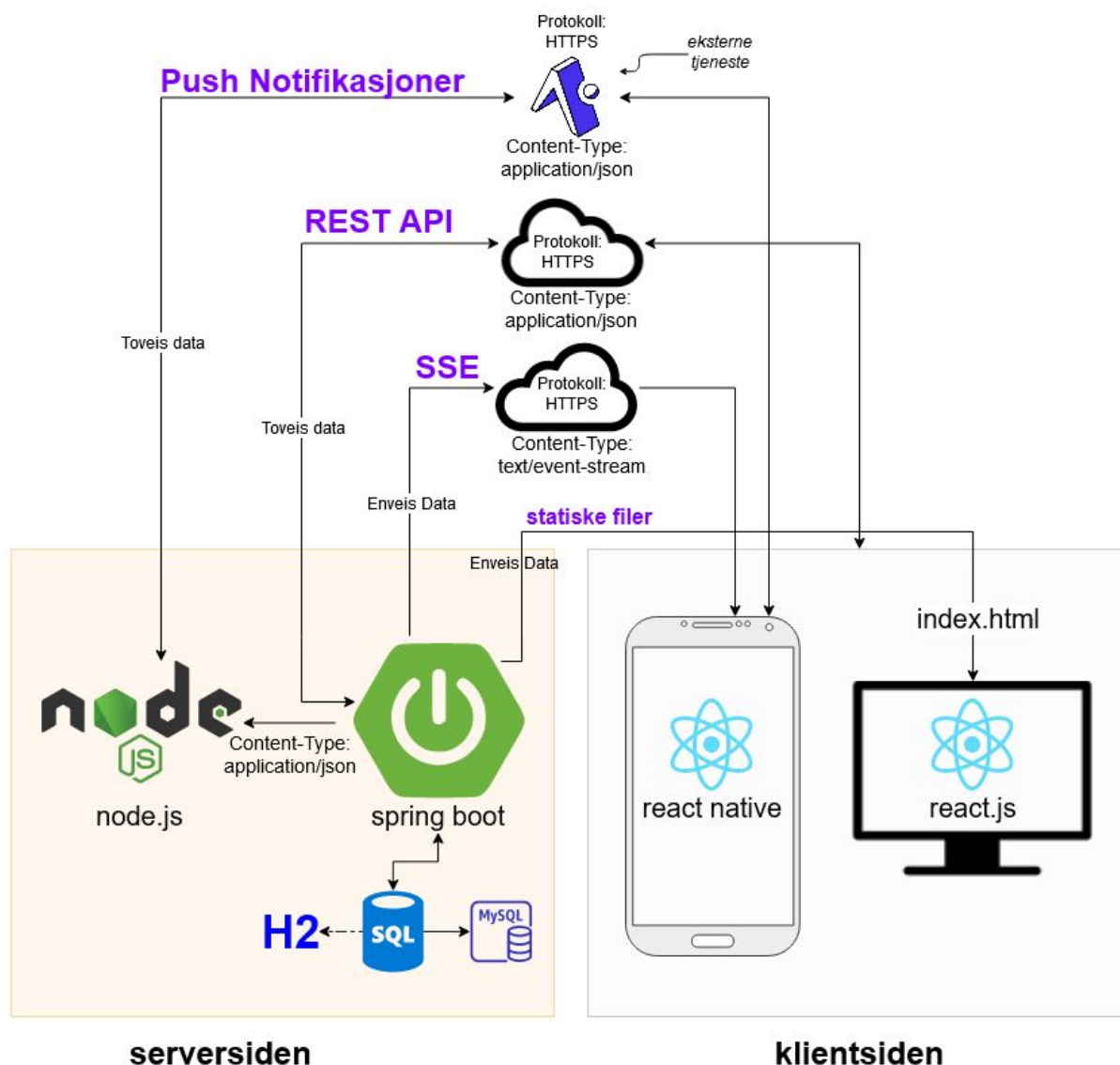
Produktdokumentasjonen er hovedsakelig delt inn i serverside produktet og klientside produktet. I tillegg er det en del om forslag til videre utvikling og litt snakk om verktøy som er blitt brukt. De forskjellige delene vil innledningsvis alltid ha en form for overblikk av et konsept etterfulgt av en utdyping av konseptet. De delene med overblikk har gjerne figurer eller bilder. Inniblant trekker vi inn hvordan vi mener at etterkommere av prosjektet burde gjøre noe, ofte med begrunnelse.

Det som er vektlagt mest i produktdokumentasjonen er eksempler. Grunnen til dette er at vi føler eksempler formidler noen ting lettere enn bare ord.

Det hender at norsk og engelske ord blir brukt om hverandre. Dette er naturlig i IT-verden. Vanligvis er disse ordene fagord og blir forklart i samme avsnitt. Hvis ikke burde man sjekke [ordlisten](#) på slutten av dokumentet.

## 4.3 Systemarkitektur

I denne delen skal leseren få et overordnet blikk av hvordan systemet fungerer i helhet. Hver del av prosjektet skal bli forklart veldig enkelt.



Figur 4.1 av systemarkitekturen

Arkitekturen til systemet er tydelig delt inn i to deler. Den ene delen omtales som server-side, backend eller serversiden. Den andre omtales som klient-side, frontend eller klientside. Sammen utgjør de et forhold kjent som klient-server forhold.<sup>10</sup> Kort beskrevet betegnes serversiden som alle operasjoner gjort på en server.<sup>11</sup> Dette er forskjellig fra klientsiden der det er klienter som utfører operasjoner.<sup>12</sup> Serversiden og klientsiden kommuniserer over https og det blir brukt flere formater for å sende data. Mest brukt format for overføring av data i prosjektet er JSON.

<sup>10</sup> [https://en.wikipedia.org/wiki/Client%E2%80%93server\\_model](https://en.wikipedia.org/wiki/Client%E2%80%93server_model)

<sup>11</sup> [https://developer.mozilla.org/en-US/docs/Learn/Server-side/First\\_steps/Introduction](https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction)

<sup>12</sup> <https://www.cloudflare.com/learning/serverless/glossary/client-side-vs-server-side/>

Systemet er delt opp slik nevnt over for å tydelig skille mellom oppgaver. Det er noen oppgaver eller operasjoner som er mer egnet å bli utført på en server samt så er det noen oppgaver som er mer egnet å bli utført hos klienten. Oppgaver eller operasjoner som krever anvendelse av forretningslogikk eller konfidensiell data blir gjerne gjort server-side. Operasjoner eller oppgaver som anvendes for å visualisere data eller presentere data blir gjort klientsiden.

### **Eksempel**

Når en bankkunde ønsker tilgang til nettbanktjenester med en nettleser (klienten), initierer klienten en forespørsel til bankens server. Gjerne gjennom et brukergrensesnitt. Kunden sender påloggingsinformasjonen videre til bankens server der serveren vil utføre operasjoner mot et databasesystem for å bekrefte at kundens påloggingsinformasjon er korrekt. Gitt at kunden har oppgitt riktig informasjon vil så typisk bankens servere anvende noe forretningslogikk for å returnere en visning til kunden.

Eksemplet over tar for seg et typisk bruksområde for klient-server forhold og henter til hvordan det fører til skille mellom oppgaver.

#### *4.3.1 Serversiden*

Serversiden til prosjektet er hovedsakelig monolittisk oppbygget med et lite preg av modularitet. Den sentrale applikasjonen i serversiden er spring boot applikasjonen. Den blir omtalt som *hovedserveren* fremover. I den utføres all forretningslogikk, datatilgang, server-sent events, servering av statiske filer og den håndterer http forespørsler.

I tillegg til spring boot applikasjonen kjøres det en node.js applikasjon som håndterer push notifikasjoner. Den blir kalt *nofikasjonserver* fremover. notifikasjonserveren får tilsendt data om en notifikasjon og hvem notifikasjon burde bli sendt til fra hovedserveren. Notifikasjonserveren fungerer helt uavhengig av hovedserveren og er fullstendig modulær. Dette betyr at den betegnes som en mikrotjeneste.

Lagring av data skjer i en relasjonsdatabase. Under utvikling og testing har det blitt benyttet en in-memory database, kalt H2, for å kunne lettere tilpasse serversiden lettere. I produksjon brukes MySQL som relasjonsdatabase.

#### *4.3.2 Klientside*

Klientsiden er delt i to deler. Den første delen som er den viktigste er react native (expo) mobilapplikasjon utviklet for mobiltelefoner og fungerer uavhengig av

OS-plattform. Den andre er en webapplikasjon som bruker react-admin rammeverket og reactjs.

Mobilappen er laget med en rekke teknologier, men i hovedsak så er den bygget på react native rammeverket. Mobilappen skal tilgjengeliggjøres for medlemmer av spillgruppen hos accenture. Den skal hovedsakelig konsumere api-et og få tilgang til aktuelle ressurser som applikasjonen i helhet har å tilby, som for eksempel events og polls.

Webapplikasjonen er bygget på et frontendrammeverk som heter *react-admin*. Rammeverket er utviklet for å enkelt lage administratorsider. Administratorsider blir brukt for å gi enkelte med tilstrekkelig autorisasjon tilgang over tjenestens ressurser.

Mobil og web app-en har separat kodebase fordi det er store nok forskjeller blant teknologiene at kodegjenbruk blir lite hensiktsmessig.

### 4.3.3 Overføring av data

Overføring av data eller informasjon mellom server og klient er skjer over internett gjennom http-protokollen. Det har blitt lagt til konfigurasjoner som krypterer alle overføringene med TLS. Det vil si at serveren støtter https.

Serversiden tilgjengeliggjør en del forskjellig data. Den vil alltid sende data-en i et hensiktsmessig format. I figuren som illustrerer systemets arkitektur er det merket med fet og lilla skrift hva data-en i pilene symboliserer. Systemet sender data slik som: Push notifikasjoner, REST API, Server Sent Events (SSE) og statiske filer.

## 4.4 Serversiden

I denne delen beskrives serversiden av produktet i større grad. Hensikten med denne delen er å gi leseren dypere kunnskaper av teknologiene som brukes i serversiden av produktet. Det er en forutsetning at leseren har hvertfall lest om [systemarkitekturen](#) tidligere.

### 4.4.1 Filstruktur

Serversiden har en tydelig filstruktur som må bli fulgt strengt. Under er et ascii-tre med aktuelle mapper. Mappene er i fet skrift og hvis det er punktum så betyr det en kjede av mapper som ligger inni hverandre. Det blir forklart kort om hver mappe for å gi leseren noe oversikt over prosjektet. Noen mapper er såpass omfattende at de blir forklart mer i en senere del.



Det er anbefalt å ha så flat pakkestruktur som mulig. Det vil si at hvis ikke absolutt nødvendig så burde det ikke være flere mapper inni hverandre. Hensikten med en tydelig pakkestruktur er at man at prosjektet blir mer organisert.

- └── **main** - I main skal alle java filer.
  - └── **java.com.accenture.spillgruppeneventmanagement** - Prosjektet
    - └── **config** - Konfigurasjoner som endrer Spring Boot miljøet skal her.
    - └── **controller** - Controller klasser. Må ha "Controller" som suffiks per fil.
    - └── **dto** - Data transfer objects. Speiler aktuelle verdier til entiteter
    - └── **entities** - Database entiteter.
    - └── **exceptions** - Unntak slik som BadRequestException (status kode 404)
    - └── **messages** - SSE Meldinger
    - └── **notifications** - Notifikasjoner
    - └── **paramfiltering** - Forretningslogikk knyttet til http forespørsler
    - └── **repository** - Datatilgang
    - └── **requests** - Modeller på forespørsler.
    - └── **response** -Modeller på svar.
    - └── **security** - Klasser knyttet til sikkerhet, slik som JWT.
    - └── **service** - Service klasser. Må ha "Service" som suffiks på klassene.
    - └── **util** - Diverse funksjonalitet som har for bred nytte eller utrolig smått.
    - └── SpillgruppenEventManagerApplication.java - her starter app-en.
  - └── **resources** - Ressurser som applikasjonen bruker f.eks. statiske filer.
    - └── **keystore** - Her skal digitale sertifikat som krypterer http.
    - └── **public** - Statiske filer som hovedserveren skal serve. F.eks. admin side.
    - └── application.properties - konfigurasjonsfil for spring boot
    - └── data.sql - sql fil for å fylle opp databasen med data
- └── **test** - i test skal alle tester
  - └── **java.com.accenture.spillgruppeneventmanagement** - Test prosjektet
    - └── **service** - e.g. på test mappe. Har tester knyttet til service i prosjektet.

... her fortsetter det med flere testmapper som speiler mappene med forretningslogikk i prosjektet over.

#### 4.4.2 REST

REST står for **RE**presentational **S**tate **T**ransfer. Det betyr at når man utfører en REST forespørsel, vil serveren sende klienten en representasjon av *tilstanden* (state) til den forespurte *ressursen*. Det er en arkitekturstil som ofte blir brukt for å designe løst koblete applikasjoner over HTTP. Enhver tjeneste som følger REST kravene blir gjerne kalt en RESTful API eller bare REST API.<sup>13</sup> Denne delen informerer om REST,

---

<sup>13</sup> <https://restfulapi.net/>

især de trekkene med REST som vi mente var viktig å få frem. Med det sagt, så blir mange ting tatt for gitt fordi spring boot abstraherer bort kompleksiteten.

## Ressurs

Ressurs er et grunnleggende konsept når man snakker om REST. En ressurs er et objekt med en type, tilknyttede data, forhold til andre ressurser og et sett med metoder som fungerer den selv.<sup>14</sup> Det ligner på en instans av et objekt som ofte omtales i objektorienterte språk. Den største forskjellen blant en ressurs og et objekt instans er at ressurser har noen faste metoder. Det er nyttig å merke seg at den tilknyttede data-en til en ressurs kan være en annen ressurs. Med det sagt, blir nevnt ressurs omtalt som en *sub-ressurs* fremover.

## Ressursmetoder

Når ressursmetoder omtales menes det metoder som utfører en handling på ressursen. I dette tilfellet er ressursmetodene det samme som http-forespørsel metoder. Mer konkret: HTTP GET, POST, PUT, PATCH, DELETE og i noen tilfeller OPTION. Vi har utviklet håndtering av http-metodene som per anbefaling av [RFC7231](https://tools.ietf.org/html/rfc7231).

**Eksempel:** Hvis en forespørsel blir gjort på å slette en ressurs vil serveren sende statuskoden 204 (No Content) hvis ressursen blir vellykket slettet.

## Ressursidentifikator (Resource Identifier)

REST APler bruker Uniform Resource Identifiers (URIs) for å kartlegge ressurser. En URI som kartlegger til en ressurs blir også kalt en ressursidentifikator. Utvikler må selv velge hva ressursidentifikatorer skal være. Det er anbefalt å følge en navneplan. Vi har valgt å følge denne: <https://restfulapi.net/resource-naming/>. Den viktigste regelen er at enhver ressursidentifikator skal være i flertallsform e.g. "/users". Det er noen unntak, disse unntakene er kommentert i koden der de inntreffer.

### Eksempel på ressursidentifikator:

1. URI-en "/events" kartlegges til en samling av event-ressurser.
2. URI-en "/events/1" kartlegges til en konkret event-ressurs med 1 som id/indeks.
3. URI-en "/events/1/participants" kartlegges til en konkret event-ressurs med 1 som indeks sin sub-ressurs.

En ressursidentifikator er også kjent som et API endepunkt eller endepunkt.<sup>15</sup> Begrepene brukes om hverandre, men begge betyr URI-en som kartlegger til en ressurs.

<sup>14</sup> <https://restful-api-design.readthedocs.io/en/latest/resources.html>

<sup>15</sup> <https://smartbear.com/learn/performance-monitoring/api-endpoints/>

## Representasjon

Representasjon står for RE i REST. Måten man representerer tilstanden til en ressurs er veldig viktig. Vi har valgt å bruke JSON for å representere ressurser. Leser burde være godt kjent med JSON på forhånd av noe videreutvikling eller vedlikehold.

### Eksempel på representasjon av en enkel ressurs:

En HTTP GET forespørsel til ressursidentifikatoren `"/users/1"` vil få en slik representasjon (JSON) av ressursen som respons:

```
{
  "id": 1,
  "enterpriseId": "asim.abazi"
}
```

### Eksempel på representasjon av en samling av ressurser:

En HTTP GET forespørsel til ressursidentifikatoren `"/users"` vil få en slik representasjon (JSON) av ressursen som respons:

```
[
  {
    "id": 1,
    "enterpriseId": "asim.abazi"
  },
  {
    "id": 2,
    "enterpriseId": "aftab.hamza"
  },
  {
    "id": 3,
    "enterpriseId": "jakob.braseth"
  }
]
```

### Eksempel på representasjon av en sub-ressurs:

En HTTP GET forespørsel til ressursidentifikatoren `"/events/1/participants"` vil få en slik representasjon (JSON) av ressursen som respons:

```
{
  "participants": [
    {
      "id": 1,
      "enterpriseId": "asim.abazi"
    },
    {

```

```

    "id": 2,
    "enterpriseId": "aftab.hamza"
  }
]
}

```

## Spørringsparameter

Samlinger av ressurser kan være enorme. Med det sagt kan man legge til ekstra data til ressursidentifikatoren slik at REST API-et svarer med en representasjon som er mer innskrenket. Den ekstra data-en blir omtalt som en query parameter (spørringsparameter på norsk).<sup>16</sup> Spørringsparametere er nøkkel-verdi par som legges på slutten av en ressursidentifikator. Man legger til et spørsmålsteget (?) på slutten av en ressursidentifikator for å spesifisere at man skal legge til en spørringsparamter. Det er dessuten lov å legge til mer enn en spørringsparamter. Man gjør dette ved å legge til en ampersand (&) mellom spørringsparamterene.

### Eksempel på spørringsparameter: "/users?\_start=0&\_end=30"

Ressursidentifikatoren "/users" har fått to spørringsparametere, den første er `_start=0` og den andre er `_end=30`. Dette betyr i vårt at man ønsker en sub-samling av ressurser i intervallet for indekser mellom 0 og 30 inklusivt.

Spørringsparametere blir ikke bare brukt til å avgrense ressursamlinger. De blir generelt brukt til tilføye informasjon til en forespørsel. Noen andre vanlige spørringsparameter:

sortering - velge hvilken verdi man ønsker å sortere ressursen etter

søkeord - hente alle ressurser som inneholder noe som passer søkeordet.

## REST Standard

For å holde REST API-et organisert har vi brukt to kilder for å lage en standard. Den første er [restfulapi.net](https://restfulapi.net/rest-api-design-tutorial-with-example/) sin design tutorial.<sup>17</sup> Den andre er [typicode](https://github.com/typicode/json-server) sin JSON-Server spesifikasjon.<sup>18</sup> Utvikler burde ta en titt på disse før man velger å videreutvikle eller vedlikeholde systemet.

Valget av disse to kildene var gjort for å gjøre utviklingen av REST API-et enklere. Ved å ha en standard å jobbe mot trenger man ikke å gruble og utforske på måter man skal designet API-et på. JSON-Server spesifikasjonen var valgt for å gjøre utviklingen av klientsiden til adminsiden enklere. Spesifikasjon valget blir nærmere sett på i [adminside](#) delen lengre ned.

<sup>16</sup> <https://rapidapi.com/blog/api-glossary/parameters/query/>

<sup>17</sup> <https://restfulapi.net/rest-api-design-tutorial-with-example/>

<sup>18</sup> <https://github.com/typicode/json-server>

Serversiden har flere lag med abstraksjoner som gjør det mulig å endre standard. Dette er oppnådd ved å lage abstraksjoner i forretningslogikken som håndterer kartlegging av ressursidentifikatorer, spørringsparamtere og opprettelse av representasjonen. Dette blir diskutert i mer detalj lengre ned i [abstraksjoner av forretningslogikk](#) under [service](#).

### 4.4.3 Hovedserveren

Som nevnt i [systemarkitektur delen](#) refereres spring boot applikasjonen i systemet som *hovedserveren*.

#### Valg av teknologi

Valg av hvilken teknologi som skulle stå bak hovedserveren var påvirket av to faktorer. De faktorene var: Utvidbarhet og modenhet. Disse faktorene var viktig fordi vi ønsket å lage et godt produkt, samt lage et produkt som kan utvides på en rimelig måte.

Vi visste veldig tidlig at prosjektet skulle handle om å lage et REST API. Fra tidligere emner og personlige prosjekt hadde vi erfaringer med å lage REST API-er i C# sin ASP.NET CORE og express.js på node.js. Disse erfaringene ga solide grunnlag for å kunne designe et REST API, men var ikke definitive når det gjelder valg av teknologi som skulle serve api-et.

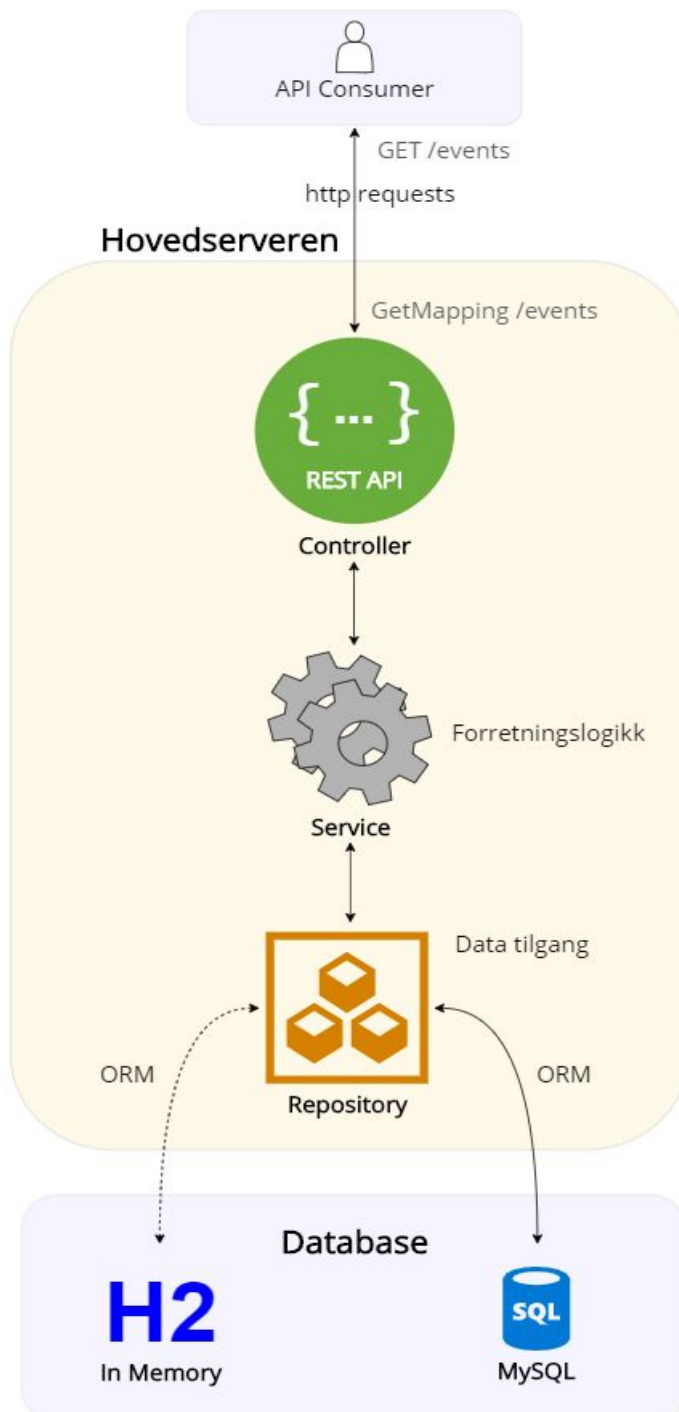
Under en rekke samtaler med veilederne fra Accenture fant vi ut om Spring. Det viser seg at spring blir hyppig brukt på Accenture, dette betyr at det skulle ikke være vanskelig å finne noen fra Accenture som kan videreutvikle produktet. I tillegg til dette arrangerte Accenture et Spring kurs som vi deltok på. Kursholder ga innblikk i hva spring støtter og hvordan man burde lage en spring boot applikasjon. Etter det kurset var vi overbevist om at spring var modent og utvidbart for vårt tilfelle. Det var også veldig positivt at man utviklet spring boot applikasjoner i java, da dette er et språk vi alle har god kjennskap til.

#### Arkitektur

REST-API delen av hovedserveren er bygget etter en fire lags arkitektur. Hvert lag har et begrenset bruksområde som utvikler(ene) må følge strengt. Ved å dele opp applikasjonen i lag, får utviklere muligheten til å endre eller legge til bestemte lag. Slik modularitet fører til at man ikke trenger å bearbeide hele applikasjonen når man videreutvikler applikasjonen.

De fire lagene er: *Controller*, *Service*, *Repository* og *Object Relation Mapping* (ORM) -layer.

Her har vi valgt å inkludere ORM som et lag, fordi kartleggingen fra objektorientert domenemodell til en relasjonsdatabase er omfattende i Spring og Spring Boot applikasjoner. Under er en figur av arkitekturen og en forklaring for hvert lag. Forklaringen er kort og skal gi et overblikk. Det skal være en del per lag i [serversiden](#) delen lengre ned.



Figur 4.3 av hovedserver arkitekturen (REST)

**Controller:** Her håndteres http forespørsler. Metoder i controller klassen annoteres med api-ruter slik som `@GetMapping("/events")`. Den metoden i controlleren vil alltid bli kjørt hvis en GET http forespørsel til `"/events"` blir gjort. Metoden i controller kaller typisk på en metode i service og det forventes tilbake noe data som skal bli sendt til den som utfører http forespørselen.

**Service:** I service bearbeides ofte http forespørsler. Det blir utført forretningslogikk på aktuell data. Data kommer oftest fra repository laget under. En hver *public* metode skal returnere en DTO (data transfer object) som skal opp til controlleren og

**Repository:** I dette laget defineres hvordan innhenting av data gjøres og hvilken data.

**ORM:** ORM er laget som bearbeider objekter til noe en relasjonsdatabase kan håndtere. Her innebærer det å lage objekter og annotere objektene og deres relasjoner til andre objekter. ORM er det som generer sql kode.

Lagdeling er dessuten et grunnleggende REST krav spesifisert av Roy Fielding i sin originale avhandling om REST.<sup>19</sup> Det er et krav fordi det lag konseptet fører til mer isolasjon av tjenester. Denne isolasjonen kan utnyttes på flere måter for å få bedre ytelse og skalerbarhet av kodebase. I vårt tilfelle har lagdeling blitt brukt for skalerbarhet av kodebase.

Hver enkel operasjon i REST API delen av hovedserveren er synkron. Det vil si at operasjoner utføres en og en. Hvis en operasjon tar lang tid vil neste operasjon måtte vente helt til tidligere nevnt operasjon er ferdig. Dette var et designvalg som vi tok veldig tidlig. Valget var hovedsakelig avgjort basert på informasjon vi hadde fått om medlemmer i spillgruppen (under 500 i Norge).

Hovedserveren består av to tilleggs tjenester. Begge tjenestene er webtjenester, men de har ingenting med REST å gjøre. Den første tjeneste er server-sent events (SSE) og den andre er servering av statiske filer som i dette tilfellet gjelder admin siden. Begge tjenestene er viktig for helheten av systemet.

### Overblikk av Server-Sent Events (SSE)

SSE er en server push teknologi.<sup>20</sup> I et klient-server forhold kan man bruke SSE for å sende meldinger fra serveren til klienten. Meldingene blir sendt over http og blir sendt som strøm av tekst karakterer (text/event-stream). Det er veldig nyttig at meldingene blir sendt over http fordi det kreves ikke noen tilleggs konfigurasjon eller endringer til hovedserveren. Disse meldingene kan klienten handle på og i vårt tilfelle vil klienten sitt brukergrensesnitt oppdateres.

Meldinger blir sendt når en hendelse på serversiden inntreffer. Hendelser slik som når noen stemmer på et poll blir sendt til alle som er på den delen av brukergrensesnittet. Dette betyr at SSE er utviklet til å være hendelsesdrevet.

SSE er en asynkron funksjonalitet. Den delen av hovedserveren blir delegert til en annen tråd og det fører til at SSE og REST API-et blir håndtert uavhengig av hverandre. Dette var et designvalg som vi gjorde fordi vi tenkte på videreutvikling av serveren.

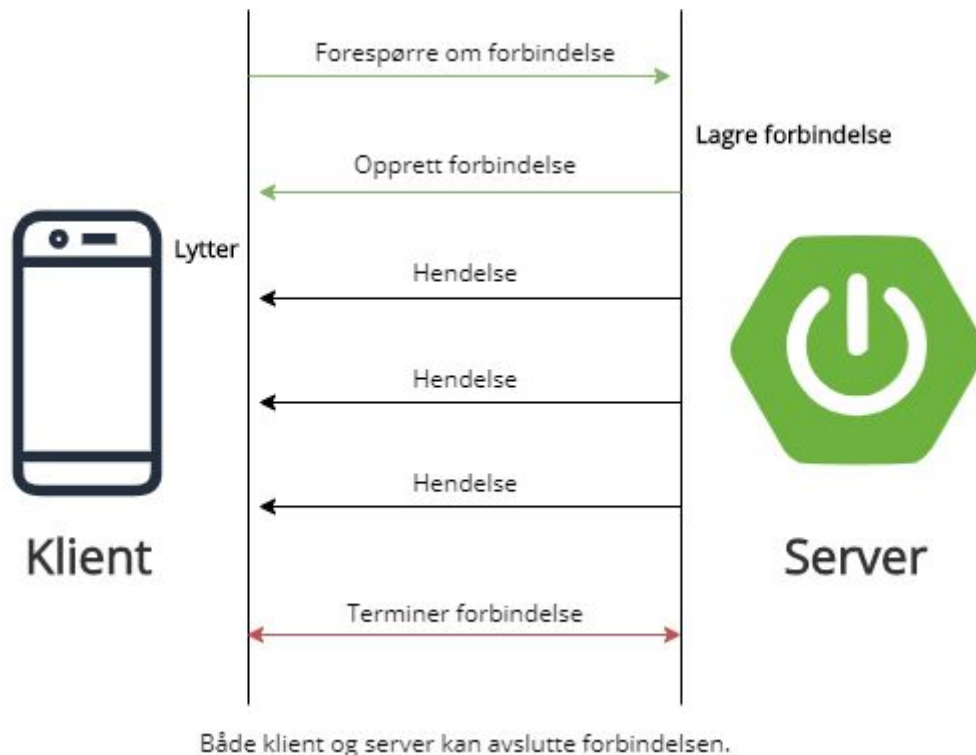
Under er en figur av hvordan forholdet mellom klienten og serveren ser ut med tanke på SSE funksjonaliteten. En viktige detalj man merker ut ifra figuren er at serveren lagrer forbindelsen mellom den selv og klienten. Dette betyr at SSE delen av serveren ikke er *stateless*. Som en konsekvens av dette vil den aldri oppfylle kravene for REST arkitektur. Dette er en forskjell vi prøver å tydeliggjøre ofte for å

---

<sup>19</sup> [https://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)

<sup>20</sup> [https://en.wikipedia.org/wiki/Server-sent\\_events](https://en.wikipedia.org/wiki/Server-sent_events)

ikke skape forvirringer, altså SSE  $\neq$  REST API. En annen konsekvens av dette er at systemet regnes som mindre skalerbart fordi den ikke er stateless.



Figur 4.4 av forhold mellom klient og server (SSE)

Data beveger seg bare i en retning etter forbindelse mellom klient og server er opprettet. Det heter *Server-sent events* av god grunn. Etter at forbindelsen er opprettet vil klienten aldri få mulighet til å sende data gjennom forbindelsen. Serveren derimot vil få kobling til klienten der data i form av tekst-strøm kan bli sendt. I vårt tilfelle sendes oftest JSON over.

Resultatet av SSE er at klientsiden får sanntidsoppdateringer på hendelser.

### Statiske filservering

Den siste tjeneste hovedserveren har er statisk filservering. I vårt tilfelle serveres `index.html` filen (og alle avhengigheter) for admin-siden. Filen regnes også som en ressurs og man får tilgang til den med ressursindikatoren `"/"`. Vi har latt servering av statiske filer forbi på rot URI-en av enkelhets skyld. Det er fullt mulig å endre konfigurasjonen til noe annet om man skulle ønske.



#### 4.4.4 Spring

Spring ble lansert i 2003 som en løsning på kompleksitet for serverside java applikasjoner.<sup>21</sup> Spring framework var den første skapelsen i spring familien. Med spring framework kom det en rekke kjerneteknologier som er høyst konfigurerbare. Disse kjerneteknologiene er delt opp i moduler og utviklere kan selv velge å bruke, konfigurere eller utvide disse modulene.

##### **Eksempler på viktige moduler:**

*Core:* Dette er den viktigste modulen i spring framework. Denne modulen definerer inversion of control (IoC) og aspect oriented programming (AOP).

*Spring Web MVC* - Brukes for å bygge webapplikasjoner og følger Model-View-Controller designmønster. Det er også mer kjent som Spring MVC.<sup>22</sup>

*Data Access* - Denne modulen brukes til datatilgang og samspillet mellom datatilgangslaget og virksomhets- eller tjenestelaget

##### **Inversion of Control (IoC)**

Inversion of control er et prinsipp innen programvareutvikling. Hensikten med IoC er å kunne lettere endre på oppførsel uten at du må gjøre store endringer i applikasjonen.<sup>23</sup> Hvis man bruker IoC prinsippet og ønsker å modifisere oppførsel til en del av applikasjonen kan man gjøre slik:

Utvide oftest *interface* eller *klassen* den oppførselen er knyttet til. Deretter kan man definere ny oppførsel. Begge oppførselene er bundet av en felles klasse eller interface, derfor vil programmet fortsatt kjøre.

Det er en rekke fordeler ved å følge IoC prinsippet.<sup>24</sup> Slik som:

1. Skille utførelsen av en oppgave fra implementeringen
2. Det er lettere å veksle mellom forskjellige implementeringer
3. Mer modularitet i applikasjonen.
4. Letthet i å teste et program ved å isolere en komponent eller hånet avhengighetene og la komponenter kommunisere gjennom kontrakter (interface)

---

<sup>21</sup> <https://docs.spring.io/spring/docs/current/spring-framework-reference/overview.html>

<sup>22</sup>

<https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/web.html#spring-web>

<sup>23</sup> <https://www.martinfowler.com/articles/injection.html>

<sup>24</sup> <https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring>

## IoC Container

En IoC Container er spring framework sin implementasjon av IoC prinsippet. IoC er også kjent som Dependency Injection (Avhengighetsinjeksjon). En IoC Container fungerer ved at et objekt definerer sine avhengigheter (andre objekter som kan brukes) bare på forhåndsbestemte måter.<sup>25</sup> Den måten vi har brukt er å definere avhengigheten i konstruktøren til klassen. Det er flere måter å definere avhengigheter men eksemplet viser den vi bruker.

### Eksempel på spring dependency injection (DI):

@Controller

```
public class EventController {  
  
    final  
    EventRepository eventRepository;  
  
    public EventController(EventRepository eventRepository) {  
        this.eventRepository = eventRepository;  
    }  
  
}
```

Dette er et simplifisert eksempel. Det hender ofte at det er mange flere avhengigheter.

**Forklaring til eksempel på DI i Spring:** EventController er skal være et objekt, det skal EventRepository også. I dette eksemplet skal EventController objektet bruke noe fra EventRepository. EventController har definert en instansevariabel av EventRepository kalt eventRepository. Ved definere den som final og legge den til i sin konstruktør vil IoC Containeren til spring framework plugge inn en instans av EventRepository inn i EventController. Det vil si at man kan bruke EventRepository instansen inni EventController.

Dette er veldig viktig fordi alle avhengigheter blir håndtert av IoC Containeren. En veldig viktig detalj er at alle objekter som styres av IoC Container-en i et Spring prosjekt er singleton-er.

## 4.4.5 Spring Boot

Spring boot er også en del av Spring familien. Hensikten med spring boot er å få en allerede konfigurert versjon av spring framework. Den skal inneholde akkurat det

---

<sup>25</sup>

<https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html#beans-introduction>

man trenger for å lage en frittstående, produksjonsklar Spring applikasjon.<sup>26</sup> Det er alltid mulig å konfigurere selv. Dette gjør man ved å endre på en fil som heter "application.properties".

### Eksempel på egen konfigurering:

```
spring.datasource.driverClassName=org.h2.Driver
```

Her setter datakilden til å være h2. Det vil si at applikasjonen prøver å koble seg opp mot en h2 database.

### 4.4.6 Object-relational mapping

Object-relational mapping eller kort sagt ORM er et veldig viktig lag. Det stod sist under i forklaringen av [arkitekturen](#) til hovedserveren, men det er knyttet til veldig viktige konsept som egner seg å tatt opp først.

Hensikten med ORM er i bunn og grunn å oppnå persistens av data.<sup>27</sup> Det oppnås i dette tilfellet ved hjelp av en relasjonsdatabase. utfordringen er å kunne gjøre objekter samsvare med hvordan man lagrer data i en relasjonsdatabase. Denne uoverensstemmelse har ført til at blir laget rammeverk som håndterer den utfordringen.

### JPA

Java Persistence API (JPA) er en API som definerer beskriver håndtering av relasjonsdata i java-applikasjoner.<sup>28</sup>

### Hibernate

Hibernate er et ORM rammeverk som implementerer JPA. Den følger med en del annotasjoner som skal brukes på objekter og objekter sine datafelt. Disse annotasjonene beriker java-objektet med metadata. Metadata-en blir brukt av Hibernate til å kartlegge objektet sin tilstand til en relasjonsdatabase per JPA. REST API delen bruker hibernate som ORM rammeverk.

Et hvert kall til databasen vil skje i gjennom hibernate.

### Entitet

En entitet er en Java-klasse med tilstand som vanligvis blir vedvart til en tabell i en relasjonsdatabase. Forekomster av en slik entitet tilsvarer en rad i tabellen. Med

---

<sup>26</sup> <https://docs.spring.io/spring-boot/docs/2.2.4.RELEASE/reference/htmlsingle/#boot-documentation>

<sup>27</sup> <https://hibernate.org/orm/what-is-an-orm/>

<sup>28</sup> [https://en.wikipedia.org/wiki/Java\\_Persistence\\_API](https://en.wikipedia.org/wiki/Java_Persistence_API)

Hibernate må alle entiteter annoteres med *@Entity* og ha en primærnøkkel som er annotert med *@Id*.

## Relasjon

Entiteter har vanligvis relasjoner til andre entiteter. Disse forholdene kommer til uttrykk i datafeltet i java-objektet som referanser til andre java-objekt. For at Hibernate skal tolke referansene riktig må man tilføre annotasjon over referansen. Annotasjonen må inneholde hva slags relasjon det er og på hvilket datafelt relasjonen gjelder på.

Forskjellige de vanligste relasjonene er prosjektet er følgende: *@OneToMany*, *@ManyToOne*, *@ManyToOne*.

### Eksempel på en entitet med relasjon til en annen entitet:

Under et to entiteter. Legg merke til annotasjonene på entitetene.

```
@Entity
public class Bruker {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank
    private String brukernavn;

    @OneToMany(
        mappedBy = "bruker"
    )
    List<Applikasjon> applikasjoner;
}
```

```

@Entity
public class Applikasjon {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank
    private String applikasjonNavn;

    @ManyToOne
    Bruker bruker;
}

```

De to entitetene har en relasjon. Det er slik at en bruker kan ha flere applikasjoner. Det vil si at en applikasjon må ha bruker knyttet til den. Oversatt til relasjonell domene så betyr det at tabellen for applikasjoner vil måtte inneholde en kolonne for primærnøkkelen til bruker (id). Eksemplet viser også hvor annerledes objektorientert domene er i forhold til relasjonell domene.

Hibernate i ORM-laget har stort omfang. Det betyr at det kan være vanskelig å vite hvordan man burde vedlikeholder og utvider dette laget. Måten vi anbefaler er å tenke database først. Det vil si at man designer databasene med å tenke i den relasjonelle domene. Deretter kan man bruke Hibernate User Guide til å for å få komme frem til hvordan man implementerer databasen.<sup>29</sup> Grunnen til at vi anbefaler fremgangsmåten er at det vil være lettere å lage en god database i høvelig normalform hvis man tenker på relasjonelt fremfor objektorientert. Realistisk sett vil man ikke trenge å skrive noe SQL-kode.

#### 4.4.7 Controller

Controllerlaget er inngang og utgangs -punktet til applikasjonen. Enhver http-forespørsel på en ressurs blir håndtert av en dedikert controller. I controller skal det kartlegges ressursindikatorer og ressursmetoder til metoder som skal videre bearbeide forespørsler. Ressurser og controller er tett knyttet til hverandre.

<sup>29</sup> [https://docs.jboss.org/hibernate/orm/5.4/userguide/html\\_single/Hibernate\\_User\\_Guide.html](https://docs.jboss.org/hibernate/orm/5.4/userguide/html_single/Hibernate_User_Guide.html)

Det er slik at hvis man ønsker å gjøre en ressurs tilgjengelig så må man lage en controller for den overordnede ressursen. Navngivingen til controlleren skal være slik: [Ressurs]+Controller.

### **Eksempel på forhold mellom ressurs og controller:**

Hvis man ønsker å gjøre `"/events"` ressursen tilgjengelig må man opprette en klasse *EventController* i mappen *Controller*. Da blir event ressursen den overordnede ressursen. Hvis man ønsker å få tilgang til sub-ressurser av event slik som `"/events/1/participants"` skal håndteringen av den ressursen forbli i *EventController*.

For at en klasse skal bli gjenkjent som en gyldig controller må man konfigurere klassen riktig. Systemet er hovedsakelig en REST API så man må annotere klassedefinisjonen med *@RestController* slik at Spring kan gjenkjenne at utvikler har intensjoner om å bruke klassen som en controller. Ved å legge til *@RestController* vil alle objekter som controlleren sine metoder returnerer bli gjort om til JSON som standard.

Innad controller klassen trengs det metoder som Spring MVC bruker for å håndtere http-forespørsler. Disse metodene må annoteres med ressursidenfikator og hvilken http-metode som utføres for at ressurser blir kartlagt riktig. Dette gjør man med annotasjonen: *@GetMapping*, *@PostMapping*, *@PutMapping*, *@DeleteMapping*, *@PatchMapping* og *@OptionMapping*. Navneplanen på annotasjonene tilsvarer *@[httpmetode]Mapping*. Man kan valgfritt også legge til ressursidenfikator hvis man ønsker å spesifisere ressursen slik som: *@GetMapping("/events")*. For å tydeliggjøre oppbyggingen av metodene

Hver metode i controller klassen skal returnere et *ResponseEntity<T>* objekt, der T er et vilkårlig objekt. Dette er for å forsikre at representasjonen er konsistent på tvers av ressurser. Et *ResponseEntity* objekt følger med Spring Framework og er ikke noe mindre enn et objekt T og en statuskode. Objektet T er oftest en Data Transfer Object (DTO).

### **Eksempel og gjennomgang av metode i controller:**

Under er et kode tatt ut i fra prosjektet og gjort små justeringer for å simplifisere det for eksemplets skyld. Enkelte deler er nummerert for å fremheve dem, de delene forklares under eksemplet.

```
@GetMapping("/events")1
public ResponseEntity<?>2 getList(
    @RequestParam(required = false)3 Map<String, String> params4) {

    Response<?> res = eventService.getEventsFilter(params);5
```

```
return new ResponseEntity<>(res, HttpStatus.OK6);  
}
```

1. Annotasjon som spesifiserer http-metode og ressursidentifikator til metoden.
2. Retur-verdien til metoden. Det er av typen `ResponseEntity<?>`, der `?` er et vilkårlig objekt.
3. Spørringsparameter annotasjon. Den kan tilføyes en boolean for å spesifisere om spørringsparamterer nødvendig.
4. Spørringsparametere blir tolket og lagt til en `Map<String, String>` objekt.
5. Utfører et metodekall. Metoden er i en service klasse og der bearbeides spørringsparameterene for å lage en representasjon av den forespurte ressursen. Det blir returnert et `Response<?>` objekt der `?` er et vilkårlig objekt, mest sannsynlig en DTO.
6. Metoden returnerer et `ResponseEntity` objekt med `res` som innhold og en statuskode OK (200).

De fleste metodene i controllerklassene vil ha lignende struktur som eksemplet over.

Det er veldig viktig at man unngår å utføre forretningslogikk i controller. Hvis man gjør det bryter man arkitekturen til systemet. Det er imidlertid ikke umulig å utføre forretningslogikk og dermed er utviklerens ansvar å forholde seg til dette.

#### 4.4.8 Service

I servicelaget holder forretningslogikk til. Etter at en http-forespørsel har blitt kartlagt riktig i controllerklassen så er det naturlig at controllerklassen kaller på en metode fra en serviceklasse. Controllerklassen forventer at serviceklassen skal returnere en representasjon av tilstanden til ressursen som ble forespurt. Likt som controllerklassene så må serviceklassene bli satt opp på en bestemt måte for å fungere slik som den skal og for å holde arkitekturen til serveren konsistent.

Regelen for navngiving og service sitt forhold til en ressurs er tilsvarende som i controller. Det vil si at man skal kalle serviceklassen for `[Ressurs]+Controller`. Det er også greit å ha utførelse forretningslogikk på sub-ressurser av den overordnede ressursen i samme serviceklasse. Med det sagt så er det ikke uvanlig å lage en dedikert service for sub-ressursene også. Det er vanlig å organisere det slik i de tilfeller der sub-ressursen er mer omfattende og krever mer kode.

### Eksempel på service av sub-ressurser:

Det ønskes funksjonalitet som tillater at alle brukere kan endre eller legge til valg på noen andre sitt poll. Ressursen `"/polls/{id}/choices"` har nå gått fra å være uforanderlige til å bli foranderlig, men også fra forskjellige brukere (potensielt samtidig).

Det er en rekke utfordringer med å legge til slik funksjonalitet. Mange av dem vil være å håndtere edge-cases og forsikre at data forblir konsistent. Tillegget av funksjonaliteten vil gjøre PollService ikke vedlikeholdbar fordi den vil bringe masse forretningslogikk knyttet til bare sub-ressursen. I dette tilfellet er det rådet å opprette en ChoiceService klasse for å skille oppgaver.

Serviceklasser må konfigureres mindre enn controllerklasser. Det er vanlig å legge til `@Service` i klassens definisjon. Det er likevel ikke påkrevd av Spring og man kan bruke `@Component` for å bli gjenkjent av IoC Containeren. Likevel insisterer vi på at serviceklassene skal bli annotert med `@Service`.

Den vanligste oppgaven som blir gjort i servicelaget er å lage representasjon av data. I dette tilfellet blir representasjonen bygget opp av data som kommer fra relasjonsdatabasen. Selve implementasjonen av innhenting av data-en skjer i datatilgangslaget (repository), men servicelaget har tilgang til å gjøre kall på datatilgangslaget. Disse kallen blir kalt *repositorykall* fremover. Oppbyggingen av representasjonen skjer ved å kartlegge verdier til entiteter i databasen til *Data Transfer Objects* sine variabler. Det er viktig å huske på at representasjonen på dette punktet er et java-objekt og blir ikke gjort om til JSON før i controllerlaget. Denne metoden kalles *data mapping* (datakartlegging) og er en sentral teknikk som blir brukt i prosjektet.<sup>30</sup>

Repositorykall som blir gjort i servicelaget kan mutere data. Det er opp til utvikleren å passe på dette. I det tilfellet burde utvikler opprettholde C-en i ACID-prinsippet som er konsistens (consistency). For å oppnå dette må metoden man utfører repositorykallet være annotert med `@Transactional`. Spring vil tolke det som at man ønsker å starte en transaksjon på starten av metoden, også utføre den (commit) når metoden er ferdig.

### Data transfer object (DTO)

Data transfer objects er objekter som bærer eller holder på data mellom prosesser.

<sup>31</sup> I denne konteksten er de bare vanlige objekter som speiler aktuelle variabler en entitet har. Nøkkelordet her er *aktuelle variabler* fordi selv om man kan kartlegge

---

<sup>30</sup> [https://en.wikipedia.org/wiki/Data\\_mapping](https://en.wikipedia.org/wiki/Data_mapping)

<sup>31</sup> [https://en.wikipedia.org/wiki/Data\\_transfer\\_object](https://en.wikipedia.org/wiki/Data_transfer_object)



alle verdiene så betyr det ikke at man burde. Noen variabler som f.eks. passord burde aldri forlate databasen. Dette er en stor grunn til at DTO-er eksisterer. De fungerer ofte som et abstraksjonslag over entiteter.

### Eksempel på DTO:

```
public class UserInfo {  
  
    private long id; // getter og setter kreves også  
    private String enterpriseId; // getter og setter kreves også  
  
}
```

Vi har navneplan og konvensjoner som vi følger for opprettelse av DTO-er. Vår konvensjon er at alle DTO-er som speiler alle variabler så skal DTO-en hete [Entitet]+Info, den skal også være plassert i *dto* mappen. Så i tilfellet over speiler DTO-er variablene til en User entitet. Det er viktig å merke at variablene det gjelder er de som ikke har en relasjon. Hvis DTO-en inneholder alle variabler, inkludert de med relasjoner så skal den hete [Entitet]+Details. Details objektet kan potensielt være mange ganger større og derfor burde de brukes varsomt. Hvis et details objekt inneholder  $n$  antall sub-ressurser, må det gjøres databasekall på alle de  $n$  sub-ressursene.

### Datakartlegging og ModelMapper

For å gjøre det enklere å datakartlegge brukes et datakartleggingsbibliotek med navn ModelMapper. Den blir lagt til som en avhengighet i maven og kan så brukes hvor som helst ved å lage en instanse av et *ModelMapper* objekt.

To viktige konsepter ModelMapper er *Source Model* (Kildemodell) og *Destination Model* (Målmodell). Kildemodellen er den modellen man ønsker å *trekke* data ut ifra mens målmodellen er den modellen man ønsker å kartlegge den uttrukne data-en til.

## Eksempel på kildemodell og målmodell:

### Source model

```
Anta at alle klassene har getter og setter
class Order {
    Customer customer;
    Address billingAddress;
}

class Customer {
    Name name;
}

class Name {
    String firstName;
    String lastName;
}

class Address {
    String street;
    String city;
}
```

### Destination Model

```
// Anta at klassen har gettere og settere
class OrderDTO {
    String customerFirstName;
    String customerLastName;
    String billingStreet;
    String billingCity;
}
```

Bilde av *ModelMapper* sin kildemodell eksempel & målmodell eksempel.<sup>32</sup>

## Kodeeksempel av kartleggingen:

```
// ModelMapper instans
```

```
ModelMapper mapper = new ModelMapper();
```

```
// Order -> OrderDTO kartlegging
```

```
OrderDTO orderDTO = mapper.map(order, OrderDTO.class);
```

Slik vil alle verdiene i *Order* objektet (som er initiert) bli kartlagt til et nytt *OrderDTO* objekt. Det er viktig å merke seg at variabelnavnene på feltene er identiske. Dette er et krav for at *ModelMapper* skal fungere ordentlig. Hvis man ikke er tilfreds med variabelnavnet i målmodellen kan man annotere den variabelen med `@JsonProperty(" ")` og legge til ønsket verdi mellom sitat tegnene.

<sup>32</sup> <http://modelmapper.org/getting-started/#how-it-works>

Kartlegging kan utføres på vilkårlige objekt. Det hyppigste bruken av ModelMapper vil være for å kartlegge entiteter til DTO-er. Med det sagt så burde man huske på at det ikke er et krav og man kan kartlegge to vilkårlige objekter.

Ved å bruke et bibliotek for data mapping blir kodebasen mer ryddig og oversiktlig. Hvis vi skulle ha gått for gammeldags data mapping som innebærer bruk av modeller sine getter og setter metode så ville det blitt veldig mye kode. Dette er ikke nødvendigvis problematisk, men mindre kode er lettere å tolke og vedlikeholde.

## Paginerings

Paginerings er ofte nyttig når man har et en stor samling av ressurser og man ønsker å presentere det for klienten i mindre samlinger. Implementasjonen av paginerings avhenger av tilleggsinformasjon som klienten må legge til i http-forespørselen. I dette tilfellet må klienten holde styr på spørringsparameterene *\_start* og *\_end*. Behandlingen av spørringsparamterene og resultatet av den behandlingen regnes som forretningslogikk.

Det er to steg til å implementere paginerings for en ressurs. Det første steget er å opprette et *Pageable* objekt, som inneholder informasjon om paginerings.<sup>33</sup> For å opprette en *pageable* må man minst ha *\_start* og *\_end* spørringsparametere. Det andre steget er å utføre et repositorykall med *Pageable* objektet. Repositorykallet vil returnere et *Page<T>* objekt der T er en entitet.

REST API-en sin paginerings er drevet av to viktige komponenter. Den første er interface *Pagination* og den andre er klassen *JSONServerPagination*.

```
public interface Pagination {
    Pageable generatePagable(Map<String, String> filter);
}

public class JSONServerPagination implements Pagination {
    @Override
    public Pageable generatePagable(Map<String, String> filter) {
        // filter holder
        // Forretningslogikk som beregner eller bedømmer hva
        // pageNr, pageSize og sort skal være.

        // Lager Pageable basert parametrene over
```

---

<sup>33</sup>

<https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/domain/Pageable.html>

```

        return PageRequest.of(pageNr, pageSize, sort);
    }
}

```

JSONServerPagination er den som blir brukt i servicelaget for å opprette Pageable objektet.

```

Pagination jsonServerPagination = new JSONServerPagination();
Pageable pageable = jsonServerPagination.generatePagable(params);

```

Det er opp til utvikler å benytte seg av paginering. Vi har benyttet oss av paginering for alle samlinger av ressurser. Derfor mener vi at etterkommer av prosjektet burde vurdere å gjøre det samme.

### Vedlikehold av forretningslogikk

Vedlikeholdbarhet er noe vi har satt stor fokus på. Dette gjelder især i servicelaget der forretningslogikk tar sted. For å oppnå Vedlikeholdbarhet har vi fulgt *Dependency Inversion Principle*.<sup>34</sup> Kort fortalt så skal ikke kode i servicelaget være avhengig av en konkret implementasjon, men heller en abstraksjon.

#### Eksempel på Dependency Inversion Principle i servicelaget:

Som nevnt i [paginering](#) brukes spørringsparamtere `_start` og `_end` for å lage en Pageable. Hvis man ønsker en alternativ måte å avgrense samlingen av ressurser kan man opprette en ny klasse for å definere den nye oppførselen. Et alternativ kan være f.eks. å ha spørringsparamtere `_pageSize` og `_pageNumber`. Så lenge man implement *Pagination* interfacet vil overgangen av spørringsparamtere bli relativt enkel. Dette er fordi paginering er avhengig av en abstraksjon (Pagination interface) isteden for en konkret implementasjon (JSONServerPagination).

Det er meget anbefalt å lage abstraksjoner på all forretningslogikk. Dette er såklart opp til utvikleren, men eksemplet over mener vi trekker frem fordelene tydelig.

Den viktigste abstraksjonen i service laget er RESTServerStandard. Den står for å opprettholde REST Standarden til typicode sin JSON-Server som nevnt i [REST Standard](#). Den ser slik ut:

```

public interface RESTServerStandard<S, R> {

    S getOne(long id);

    PageResponse<S> getList(Map<String, String> params, JSONServerFilter<R>
filterHandler);
}

```

<sup>34</sup> [https://en.wikipedia.org/wiki/Dependency\\_inversion\\_principle](https://en.wikipedia.org/wiki/Dependency_inversion_principle)

```

    PageResponse<S> getList(Map<String, String> params);

    List<S> getMany(List<Long> ids);

    S createEntity(S request, Class<R> entityType);

    void deleteEntity(long id);
}
JSONServerService implementerer RESTServerStandard. Der defineres den
konkrete forretningslogikken som skal ta seg av å holde seg til standarden.
public class JSONServerService<S, R> implements RESTServerStandard<S, R> {

    private String resource;
    private Repository<R, Long> repo;
    private Class<S> type;
}

```

Det er opp til utvikler å bedømme om JSON-Server standarden er tilstrekkelig ut i fra behovet. Hvis den ikke er det kan man alltid lage en ny implementasjon som tilfredstiller RESTServerStandard abstraksjonen.

For mer detaljer angående implementasjon av standarden burde man se til dokumentasjonen som følger med klassen. Der er det beskrevet i større detalj.

### 4.4.9 Repository

Repositorylaget tar seg av datatilgang. Når man ønsker en entitet eller en mengde entiteter må man gå igjennom repositorylaget. Når man snakker om repositorylaget snakker man om repositoryinterface som er knyttet til en entitet. På lik linje med controller og service -laget sine klasser må repositoryinterface bli satt opp på en bestemt måte for å fungere slik som den skal og for å holde arkitekturen til serveren konsistent. Alle repositoryinterface skal hete [Entitet]+Repository og skal være interface.

Implementasjonen av repositoryinterface-ene ble hovedsakelig håndtert av Spring Data JPA. Det er et bibliotek som er en del av Spring familien. Hoved konseptet bak biblioteket er å redusere kode som kreves for å utføre ofte brukte spørringer.<sup>35</sup> Den oppnår dette ved å introdusere JPA-Repository.

JPA-Repository skaper et abstraksjonslag mellom en repositoryinterface og en entitet. For å få nytte av JPA-Repository må man utvide en av JPA-Repositoryene,

---

<sup>35</sup> <https://spring.io/projects/spring-data-jpa>

blant annet: JpaRepository, CrudRepository og PagingAndSortingRepository. Utvikler burde sjekke dokumentasjonen på disse JPA-Repositoryene for å bedømme hvilken passer utviklerens behov.<sup>36</sup>

Abstraksjonslaget lar deg definere metoder i repositoryinterfacet ved å bruke forhåndsbestemte nøkkelord. Man bruker nøkkelordene til å lage en metode som skal tilsvare en spørring.

### Eksempel på JPA-Repository spørring:

```
public interface UserRepository extends JpaRepository<User, Long> {  
    List<User> findByEmailAddressAndLastname(String emailAddress, String lastname);  
}
```

Metoden findByEmailAddressAndLastname blir gjort om til:

```
select u from User u where u.emailAddress = ?1 and u.lastname = ?2
```

Vi anbefaler vedlikeholder eller videreutvikler av prosjektet å gå lese om Query Creation avsnittet i dokumentasjonen til Spring Data JPA.<sup>37</sup>

## 4.4.10 Databasemodellen

Databasemodellen er bygd etter den første kravspesifikasjonen vi ble tildelt og fokuset ble lagt på at applikasjonen skulle følge databasemodellen og ikke omvendt. For at brukeropplevelsen skal være optimal valgte vi å lage entiteter rundt en bruker. Brukeren skal ha lagret historikk som førte til relasjoner mellom entiteter også ble prioritert.

Når applikasjonen følger databasen skapes mer stabilitet ved utvikling, ettersom at databasen definerer kravene og omfanget til prosjektet. Dette er gunstig fordi man kan teste koden, sjekke relasjoner og sørge for at dataflyten er som forventet i et brukergrensesnitt. Dersom man følger applikasjonen først kan det forekomme problemer som at omfanget blir for bredt, ved testing av relasjoner må det lages et helt brukergrensesnitt før det kan kjøres og dersom applikasjonen endrer design kan det hende at databasestrukturen må endres og dette er ikke ønsket.

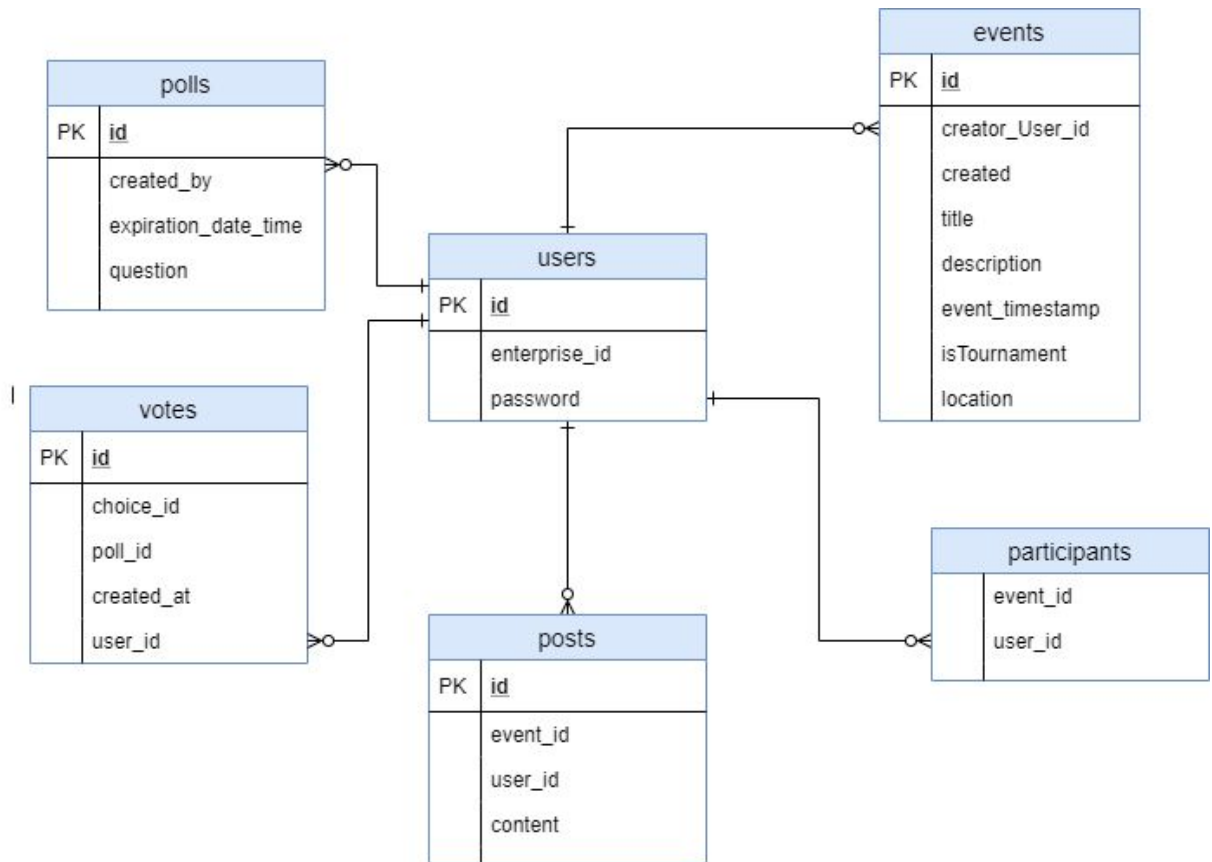
---

<sup>36</sup> <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories.core-concepts>

<sup>37</sup> <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods.query-creation>



## User



Figur 4.6

*User* entiteten har metadata som id, enterprise id og passord, dette definerer hva en User er. User har relasjoner med andre entiteter som *Event*, *Poll*, *Post* og *Role*. Dette betyr at en bruker skal ha tilgang til *Event* entiteten, og det skal bli lagret data om hvilke events en bruker har deltatt i, dette blir lagret i en binde tabell *participants*.



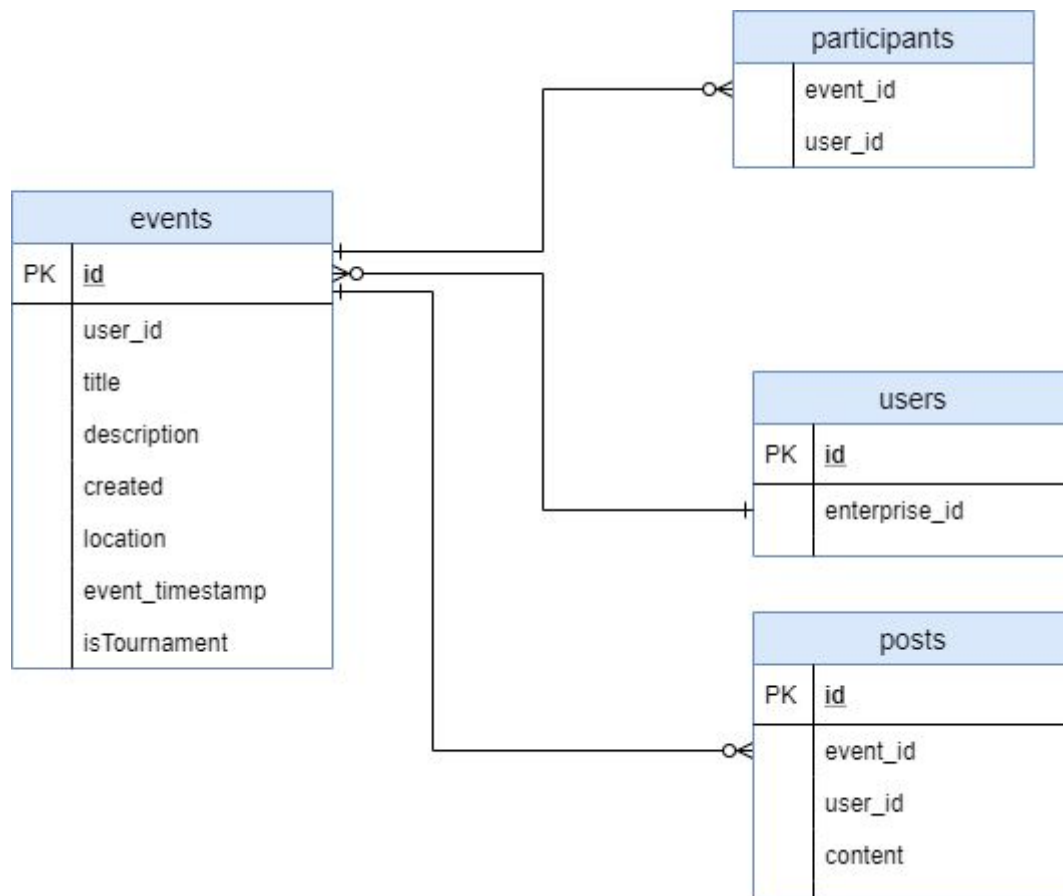
Figur 4.7

Relasjonen *Poll* og *Post* er for at man skal kunne gå gjennom en bruker og sjekke hvilke polls og posts de har laget. *Role* relasjonen er satt slik at en bruker kan være enten en Admin, Moderator, User eller flere av disse. dette er for å sikre at



ressurser ikke blir tildelt til uønskede brukere. *User* blir brukt som en hovedkomponent i applikasjonen og dataflyten følger hovedsakelig denne entiteten.

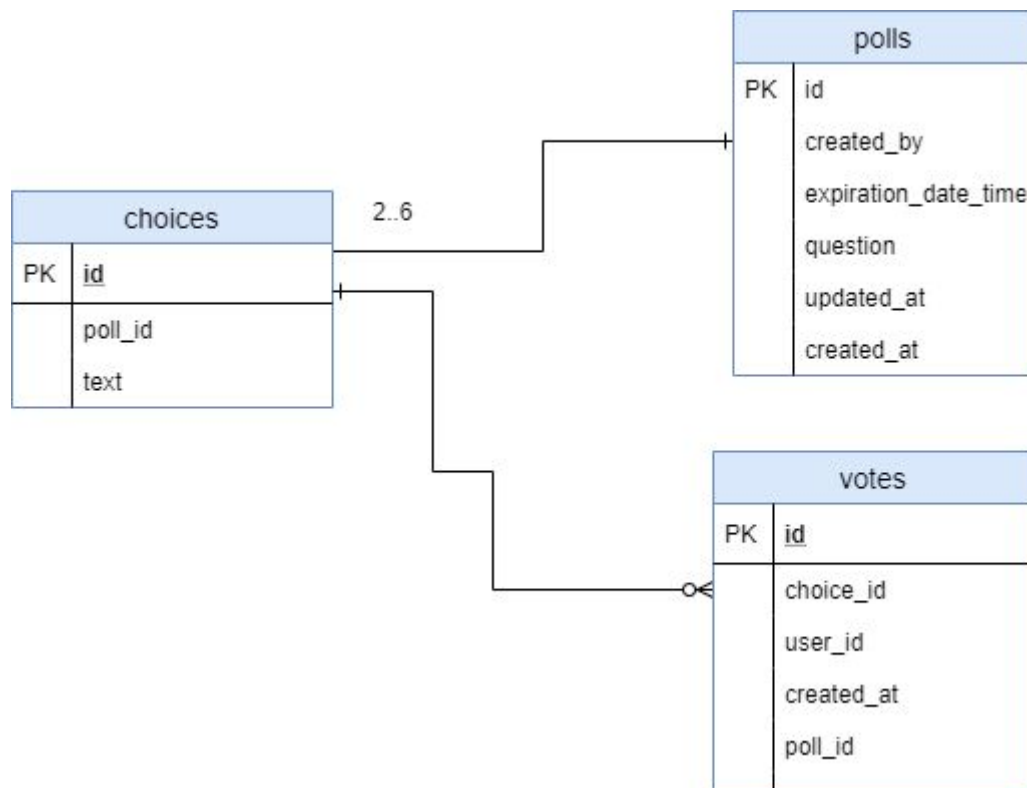
## Event



Figur 4.8

*Event* entiteten lagrer all informasjon om et event, en id, hvem som lagde eventet, når eventet ble opprettet, tittel, når eventet starter, om det er en turnering, beskrivelse av eventet og lokasjon. Eventet har relasjoner med både *Post* og *User*. Dette sørger for at man har lagret informasjonen om hvilke posts som tilhører hvilke events. *User* relasjonen sørger for at man kan se hvem som har deltatt i et event. Entiteten *Event* blir brukt i applikasjonen for at data om et event blir lagret, hvilke posts som tilhører et spesifikt event og hvem som har deltatt i dette eventet.

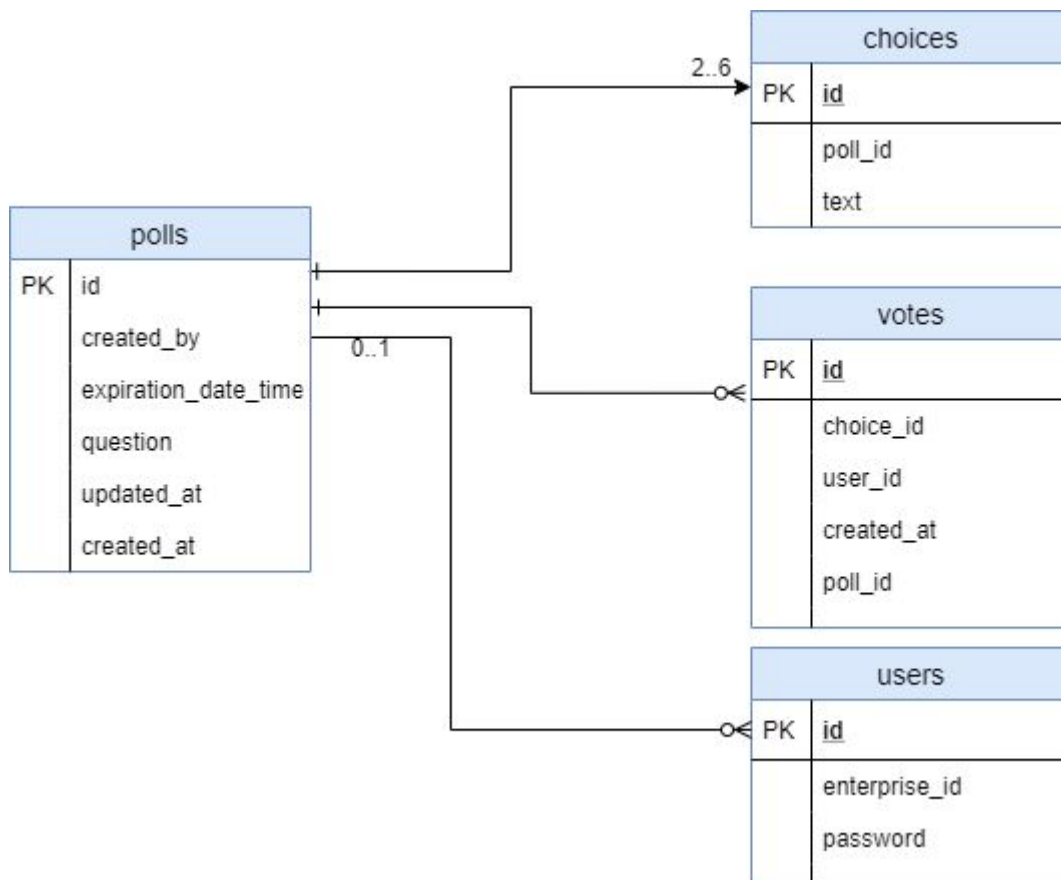
## Choice



Figur 4.9

*Choice* entiteten består av en id og en tekst. *Choice* har kun relasjoner med *Poll* entiteten. Dette er for at man skal kunne hente inn riktige valgmuligheter til riktig undersøkelse. Denne entiteten blir i hovedsak brukt for å ha valgmuligheter i en meningsmåling.

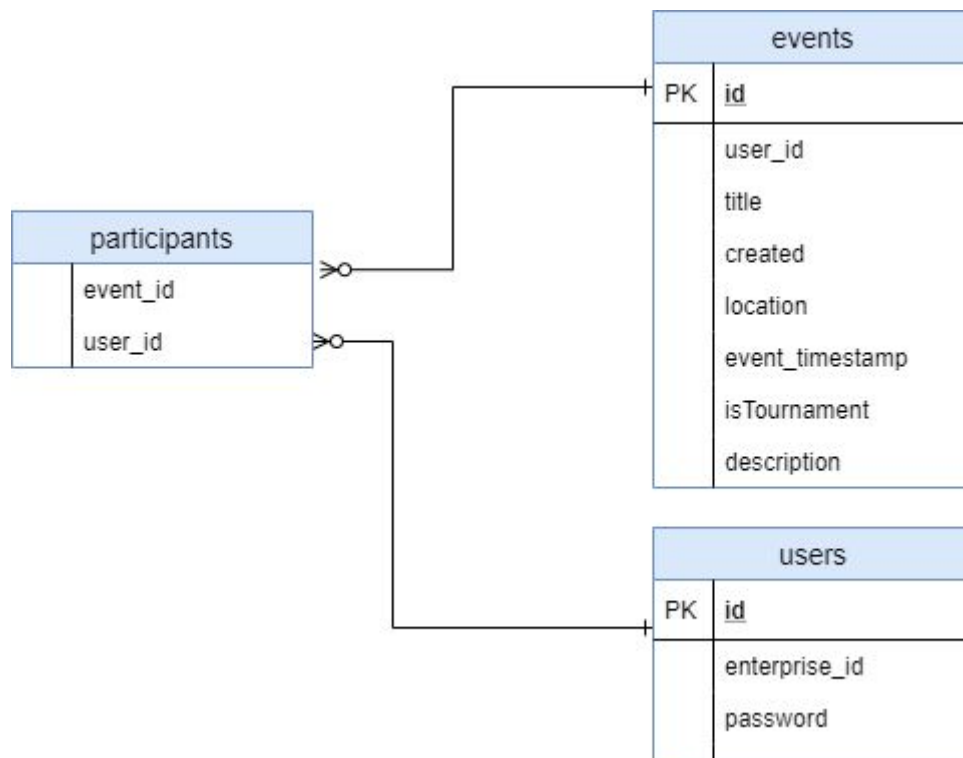
## Poll



Figur 5.0

*Poll* entiteten har metadata som en id, et spørsmål, hvem som lagde pollen, en liste av valgmuligheter og en utløpsdato. *Poll* har relasjoner med *User* og *Choice*, dette er for at man skal kunne se hvem som har laget en meningsmåling og hvilke valgmuligheter som tilhører denne. Entiteten lagrer all data knyttet til en meningsmåling. Dataen blir brukt for å fremvise andre brukere hva som har blitt spurt om og hva de kan svare.

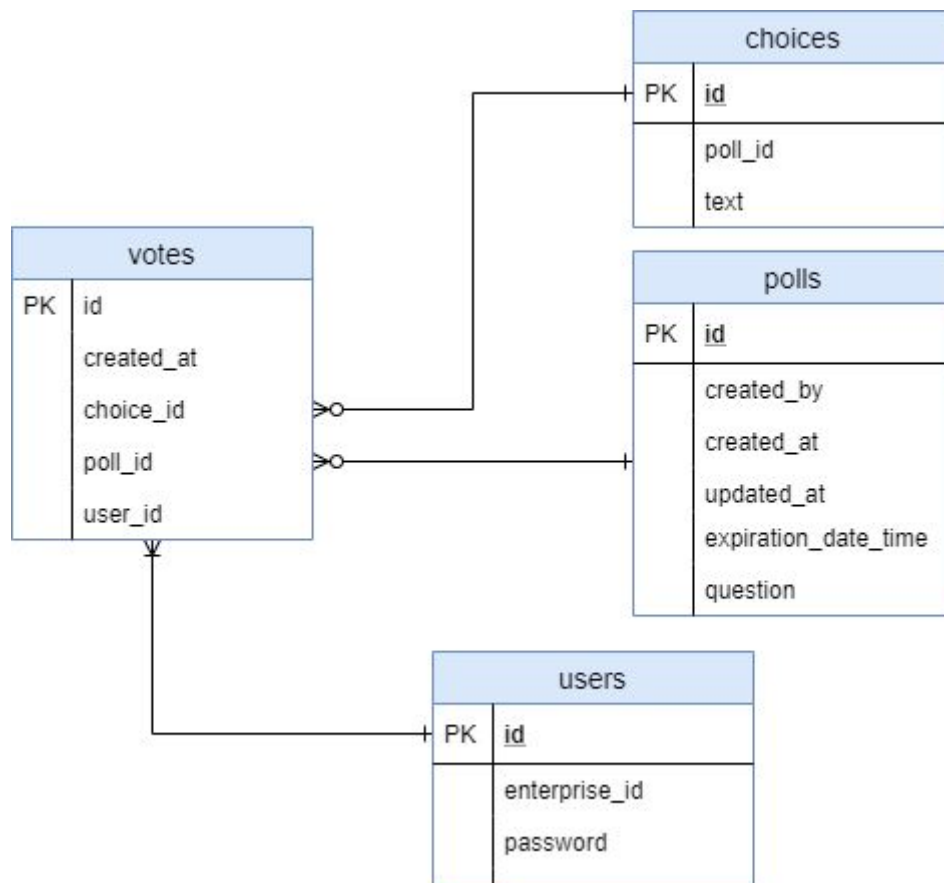
## Participants



Figur 5.1

*Participants* er ikke en entitet i seg selv, men en bindetabell som blir brukt for å lagre hvilke brukere som deltar på hvilket event. Denne entiteten blir i hovedsak brukt for å sjekke antall deltakere i et event i frontend.

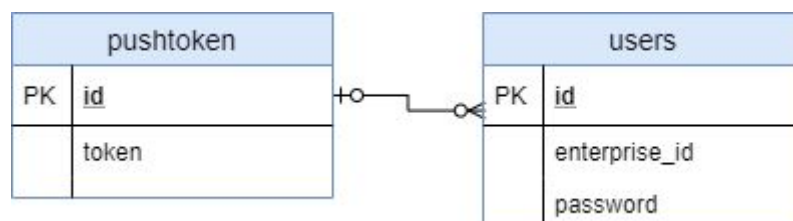
## Vote



Figur 5.2

*Vote* entiteten består av en id, et valg id, en bruker id og en poll id. Dette sørger for at når man skal sjekke en *Vote* kan man se hvilket valg det var, hvilken undersøkelse den tilhørte og hvilken bruker som valgte. Denne entiteten blir brukt for å kunne ta valg og oppdatere antall valgte av en valgmulighet i applikasjonen.

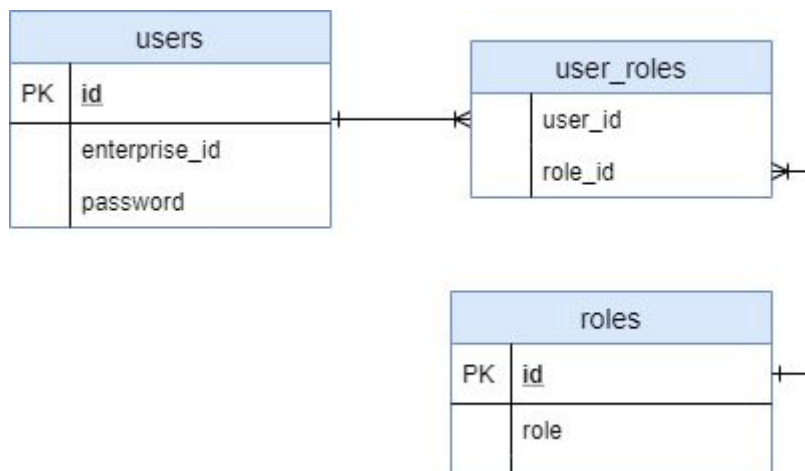
## PushToken



Figur 5.3

*PushToken* entiteten lagrer en id, en bruker id, og en token. Denne entiteten blir brukt for å sende notifikasjoner til brukere, dette gjøres ved å bruke feltet token som inneholder en unik string som identifiserer en mobil enhet.

## Role



Figur 5.4

*Role* entiteten inneholder data om en id og et rollenavn. Rollenavn er et enum og der er det tre valg, Moderator, Admin og User. Disse tre rollene blir brukt i sikkerhetsaspektet av applikasjonen for å gi tilgang til forskjellige ressurser. Entiteten blir brukt for å lagre hvilke typer roller som finnes for vår applikasjon og det kan bli lagt til nye roller.

### 4.4.11 Databasemigrasjon

Endringer av en databasemodell vil føre til at man må gjøre endringer på databasen. Under utviklingen kan det skje hyppige endringer på databasemodellene. For å unngå å gjøre endringer til databasen valgte vi å bruke en minnebasert database. En minnebasert database vil måtte rives ned for hver gang applikasjonen slås av samt bygges opp hver gang applikasjonen starter opp. Fordelen med dette er at hvis man gjør endringer til databasemodellen trengs det bare å starte applikasjonen på nytt for at databaseskjemaet skal oppdateres. Vi har valgt å bruke H2 som utviklingsdatabase.<sup>38</sup>

En konsekvens av dette er at minnebaserte databaser ikke oppfyller C-en i ACID-prinsippet (consistency). Dermed er det viktig å være OBS. på at den H2 ikke vil fungere i produksjon.

### 4.4.12 Server-Sent Events

Det er anbefalt at leser har lest [overblikk av Server-Sent Events \(SSE\)](#).

SSE krever en controller (SSEController). Det er nødvendig å tilgjengeliggjøre et endepunkt for at funksjonaliteten skal fungere. Det skal bare være et endepunkt

<sup>38</sup> <http://www.h2database.com/html/tutorial.html>

per overordnet ressurs og ikke noe mer. Endepunktet skal starte med `"/sse"` etterfulgt av ressursen den skal håndtere. Et SSE endepunkt for `"/events"` ressursen vil se slik ut: `"/sse/events"`.

Det kreves noe state for at SSE skal fungere. Når en forbindelse med en klient blir opprettet lagres forbindelsen som et `SseEmitter` objekt.<sup>39</sup> Denne forbindelsen lagres så inn i et `ConcurrentHashMap` objekt. **Slik som her:**

```
public final Map<Long, SseEmitter> pollSSE = new ConcurrentHashMap<>();
```

Det er nødvendig å ha en `ConcurrentHashMap` per SSE endepunkt.

Man sender data til klienter ved å bruke en instanse av `SseEmitter`. `SseEmitter` inneholder en metode `"send"` som tar imot et `String` objekt. Det man putter inn i `send` metoden blir sendt til klienten som forbindelsen er knyttet til.

Sending av meldinger med SSE regnes som forretningslogikk. Med det sagt, er det passende å lage en serviceklasse for SSE (`SSEService`). Slik som det er nå itereres alle forekomster av `SseEmitter` i `ConcurrentHashMap` og det utføres en `send` operasjon per forekomst.

**Eksempel på sending av melding med SSE til alle forbindelser:**

```
public void sendMessage(Messageable msg) {
    sseController.pollSSE.forEach((k, emitter) -> {
        single.execute(() -> {
            try {
                emitter.send(msg.message());
            } catch (IOException e) {
                e.printStackTrace();
            }
        });
    });
}
```

Metoden krever en `Messageable`. Dette er en abstraksjon for å håndheve at bare strøm av tekst kan bli sendt gjennom med SSE.

```
public interface Messageable {
    String message();
}
```

Den vanligste `Messageable` objektet er:

```
public class JSONMessage<T> implements Messageable {
    String type;
```

---

<sup>39</sup>

[https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/servlet/mvc/meth  
od/annotation/SseEmitter.html](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/servlet/mvc/method/annotation/SseEmitter.html)

```

    T data;

    @Override
    public String message() throws JsonProcessingException {
        ObjectMapper mapper = new ObjectMapper();
        return mapper.writeValueAsString(this);
    }
}

```

Den vil omgjøre all data om til en String formatert i JSON format. Vi anbefaler å fortsette å bruke denne implementasjonen fordi klientsiden håndterer JSON veldig godt. Med det sagt så kan utvikler enkelt føre inn en ny implementasjon hvis det skulle være behov. Det er isåfall viktig å huske på å håndtere implementasjonen i klientsiden også.

Sending av meldinger gjennom SSE skjer på en annen tråd. I serviceklassen til SSE defineres det en enkel tråd gjennom ExecutorService.

```
private final ExecutorService single = Executors.newSingleThreadExecutor();
```

På den tråden utføres all sending av meldinger. Hvis applikasjonen skulle måtte håndtere mange flere forbindelser på tvers av SSE endepunkt vil den enkeltråden vise seg å yte dårlig. Dermed anbefales det å utforske mer av ExecutorServices.

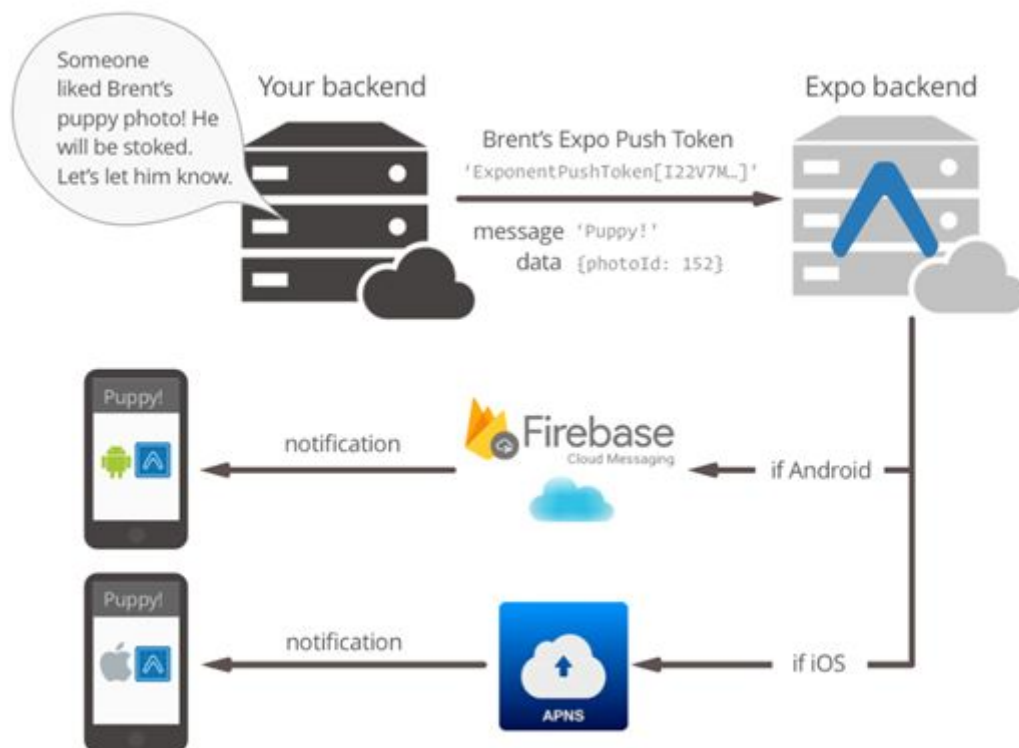
#### 4.4.13 Notifikasjonserveren

Et av målene [sprint 4](#) var å legge til funksjonalitet som sender varslinger til brukere om et nytt event blir opprettet. For å oppnå dette gikk vi igjennom Expo sin dokumentasjon for å få finne ut om Expo støttet varslingsfunksjonalitet.

##### Expo server

Expo server er en løsning som er tilrettelagt expo applikasjoner. Det er relativt lett å konfigurere og integrere en expo server i en allerede eksisterende expo applikasjon. Ved å lese igjennom dokumentasjonen kom vi fram til at det er en gunstig løsning for vår applikasjon.





Figur 5.5

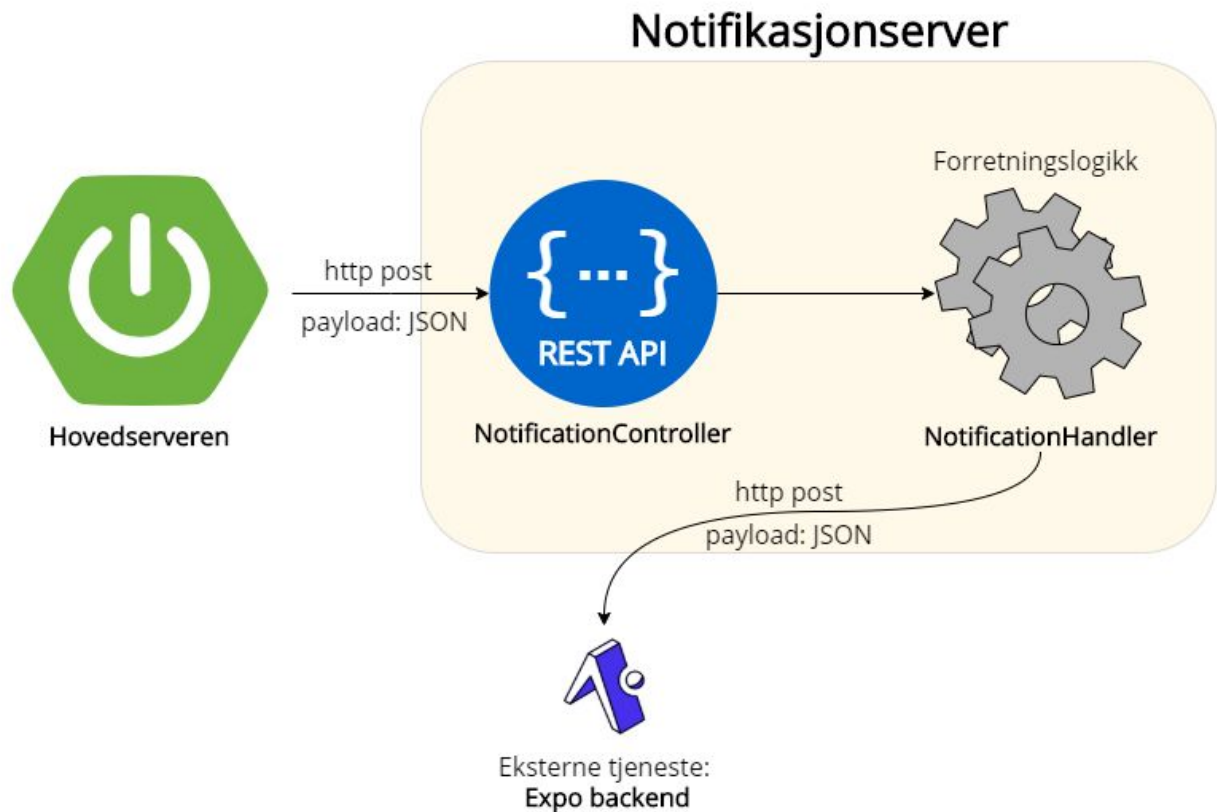
Expo tilbyr en løsning som er beskrevet i bildet ovenfor. Arkitekturen går utpå at vår hovedserver sender en forespørsel til notifikasjonsserveren ved en hendelse som krever at en bruker får informasjon. Notifikasjonsserveren sørger for at en bruker får denne informasjonen i form av en push notifikasjon.

Denne løsningen var attraktiv ettersom at Expo tilbyr et robust og effektivt system. Expo serveren støtter batch forespørsler (flere kall i en forespørsel), dette er gunstig fordi problemet med REST API er ofte at HTTP forespørsler skaper flaskehals. Istedenfor at vår hovedserver sender 1000 forespørsler for 1000 notifikasjoner så trenger vi bare å sende en forespørsel som inneholder 1000 kall. Dette effektiviserer kommunikasjon mellom servere. I tillegg har Expo serveren god støtte for applikasjoner med flere titalls tusener brukere. I vårt tilfelle er det et relativt lite omfang av brukere på rundt 100 brukere.

### Valg av servermiljø

Expo har skrevet løsningen sin i flere servermiljø. Vi ville gjerne ha et språk vi er godt kjent med og vi kom fram til at løsningen skrevet i javascript for et Node.js servermiljø ville være et fornuftig valg. Gruppen har god kunnskap innenfor javascript fra tidligere prosjekter som forsterket dette valget.

## Arkitektur



Figur 5.6 av arkitekturen til notifikasjonsserveren.

Systemarkitekturen i notifikasjonsserveren følger en to lags arkitektur. Disse lagene har vært sitt bruksområde som sørger for å skape modularitet. Det vil si at ved videreutvikling trenger ikke utviklerne å endre hele applikasjonen. Denne arkitekturen er basert på Expo sin dokumentasjonsstandard.

**Hovedserveren:** Det blir sendt en post-forespørsel fra hovedserver til notifikasjonsserveren.

**NotificationController:** Her blir post-forespørselen tatt imot. Controller håndterer forespørsler, slik som tidligere beskrevet i [hovedserveren](#).

**NotificationHandler:** I *handler* blir forespørselen bearbeidet. *Handler* fungerer i prinsippet på samme måte som service gjør i [hovedserveren](#). Konvensjonen følger Expo sin dokumentasjon der *handler* betyr håndterer.

**Expo backend:** Expo backend er et API-endepunkt som *handler* sender forespørselen til. Her håndterer Expo selv logikk for å sende varsler til mobile enheter.

#### 4.4.14 Node.js

Expo teamet har laget løsningen deres på flere programmeringsspråk og løsningen vi gikk for var javascript for Node.js server miljø. Vi valgte denne løsningen fordi gruppen har tidligere drevet med prosjekter i Node.js server miljø og hatt gode resultater.

Node.js er et runtime-system som ble lansert i 2009. Runtime-systemet ekskaverer javascript og kan kjøre på tvers av plattformer. I hovedsak blir det brukt for server og nettverksapplikasjoner. Node.js følger en asynkron modell som betyr at den ikke venter på at en operasjon skal fullføre før den kjører neste linje. Når operasjonen er fullført blir det kalt på en callback-funksjon som prosesserer resultatet.<sup>40</sup> Node.js er kjent som en effektiv og skalerbar måte å kjøre kode på.<sup>41</sup>

For at notifikasjonserver skal kunne håndtere forespørsler på en effektiv og lettvinnt måte valgte vi å bruke web rammeverket Express.js. Express rammeverket er et av de mest populære web rammeverkene og i profesjonell sammenheng blir det brukt av selskaper som PayPal, Uber og Accenture.<sup>42</sup> Express simplifiserer oppgaven med å håndtere og sette opp routes i tillegg kan man skrive få linjer for å kjøre en webserver. I tidligere prosjekter har vi også brukt Express som gjorde det til et attraktivt valg for vårt prosjekt.

#### 4.4.15 Notifikasjoner

For å kunne tilby en fullstendig løsning for mobile enheter valgte gruppen å se gjennom alternativer for levering av meldinger fra serverside til mottakere. Formålet er at en bruker får informasjon slik at de er oppdaterte om nye hendelser og at applikasjonen holdes aktiv i lenger tid. Løsningen vi valgte har en profesjonell standard for effektiv kommunikasjon mellom servere. Implementasjonen av koden er basert på utvidbarhet slik at videreutvikling skal være simplifisert.

##### Push notifikasjoner:

Push notifikasjoner blir definert som et kommunikasjonsmiddel og har blitt standardisert i dagens mobilteknologi. Statistikk viser at ved bruk av push notifikasjoner øker engasjering med applikasjon opp mot 88%, i tillegg at 65% av brukerne returnerer til applikasjonen i løpet av 30 dager. Statistikken viser også at det å sende push notifikasjoner kan øke applikasjonens retensjonsrate (forholdet mellom beholdt kundegruppe i løpet av en periode) mellom tre til ti ganger så mye.

---

<sup>40</sup> <https://nodejs.org/en/about/>

<sup>41</sup> <https://www.monterail.com/blog/nodejs-development-enterprises>

<sup>42</sup> <https://expressjs.com/en/resources/companies-using-express.html>

For å kunne bruke notifikasjoner i vårt system tok vi i bruk Expo sin tilnærming for et slikt system.

## Flyt

Flyten i backend er som følgende: Etter at push token er lagret og det blir opprettet et nytt event vil hovedserveren hente alle push tokens fra databasen, lage et notifikasjonsobjekt og sende den til notifikasjonserveren. Notifikasjonserveren vil ta imot forespørselen i **NotificationController** og sende det til **NotificationHandler**. Her valideres alle token og deretter sende det til Expo sitt API endepunkt for at alle de mobile enhetene skal få notifikasjonen. Når notifikasjonserveren sender det til Expo sitt API endepunkt er responsen en kvittering om notifikasjonen ble sendt eller ikke, denne kvitteringen blir brukt for feilhåndtering. Notifikasjonen som blir sendt til den mobile enheten inneholder data som kan bli brukt i applikasjonen frontend messig.

## Kode

For at notifikasjon systemet skal være utvidbart og robust har vi fokusert på implementasjon av kode. I hovedserveren er det to scenarier ved registrering av notifikasjoner, om brukeren eksisterer eller ikke: Ved tillatelse av notifikasjoner for første gang blir det gjort en post forespørsel fra applikasjonen til vår hovedserver og deretter blir den validert og lagret til databasen. Dersom en bruker sletter applikasjonen og laster den ned på nytt vil det bli generert en ny push token, i dette tilfelle blir det sjekket om bruker id-en finnes i databasen og om push token er den samme, dersom den er ulik vil det bli gjort en oppdatering på denne raden.

```
else {  
    PushToken updateToken =  
notificationRepository.findPushTokenByUserId(req.getUserId());  
    updateToken.setToken(req.getToken());  
    notificationRepository.save(updateToken);  
    return true;  
}
```

Den forrige push token blir fjernet fra Expo sine servere etter en periode så man skal sørge for å ikke sende unødvendige forespørsler når token ikke lenger er gyldig. Dette sørger for at tabellen i databasen ikke lagrer null verdier og at dataen ikke er redundant.

## Oppretting av en notifikasjon

Etter at all data har blitt lagret i databasen blir den hentet dersom en hendelse inntreffer, i vårt tilfelle ved å opprette et nytt event. Dette kaller på en metode som

henter inn alle tokens fra databasen og oppretter et objekt av typen **EventNotification**.

```
public class EventNotification extends
BaseNotification<EventNotificationData> {
    public EventNotification(List<String> tokens) {
        super(tokens);
    }
}
```

**EventNotification** er en klasse som arver fra klassen BaseNotification. I tillegg blir det sendt data av typen EventNotificationData. Konstruktøren kaller på BaseNotification sin konstruktør og sender en liste av tokens. Dette er alle de lagrede brukere som har tillatt notifikasjoner.

```
public class EventNotificationData {
    private long eventId;
    public final String navigation = "Event";
    public EventNotificationData(long eventId) {
        this.eventId = eventId;
    }
}
```

**EventNotificationData** er en klasse som inneholder det som kreves i klientside fra en event-notifikasjon. Det vil si den dataen som navigerer en bruker til det nye opprettede eventet. Her kreves en eventId som er id-en til det nye eventet og en string som referer til hvilken komponent det handler om.

```
public interface Notifiable<T extends BaseNotification<?>> {

    T notification(List<String> tokens);

}
```

**Notifiable** er et interface som kun representerer en metode. Dersom denne metoden skal bli brukt må objektet arve fra BaseNotification. Dette setter restriksjoner på koden slik at man må følge et mønster for å lage og bruke en ny type notifikasjon. Alt som trengs å gjøres er å lage en ny klasse som arver fra BaseNotification og at den klassen har data som skal sendes til klientside.

**Slik blir en ny notifikasjon opprettet og sendt:**

```
notificationService.sendNotification((Notifiable<EventNotification>) tokens
-> {
```

```

        EventNotification notification = new EventNotification(tokens);
        notification.setData(new EventNotificationData(createdEvent.getId()));
        notification.setTitle(createdEvent.getTitle());
        notification.setMessage("Nytt event har blitt opprettet!");
        notification.setSubtitle("Nytt event");

        return notification;
    });

```

Det blir opprettet en ny notifikasjon av typen EventNotification som inneholder tokens. Data-feltet blir satt til det nye opprettede eventet sin id, tittel blir lagret som navnet til det nye eventet, message og subtitle blir satt av utviklerne.

### Kode for å sende en notifikasjon:

```

public void sendNotification(Notifiable<?> notification) {
    try {
        URI uri = new URI(baseUrl);
        List<String> tokens = getAll();
        ResponseEntity<String> result =
restTemplate.postForEntity(uri, notification.notification(tokens),
String.class);
    } catch (URISyntaxException e) {
        e.printStackTrace();
    }
}

```

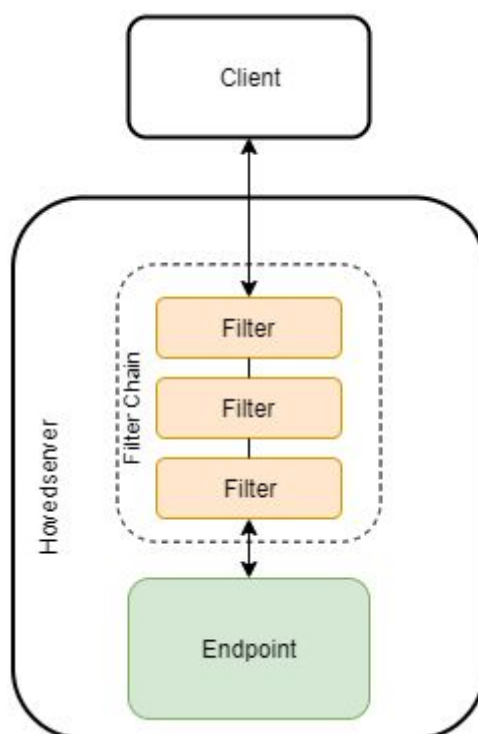
Den nye notifikasjonen blir sendt til metoden sendNotification som en parameter, deretter blir det laget en post forespørsel som sender denne notifikasjonen til notifikasjonserveren. Endepunktet til notifikasjonserveren er forhåndsdefinert og lagret som variabelen baseUrl.

## 4.4.16 Sikkerhet

Et av kravene som var høyt prioritert var sikkerhet. Denne delen vil beskrive hvordan dette oppnås i backend applikasjonen. Vi vil først ta for oss rammeverk og teknologiene som brukes. Deretter hvordan dette brukes til autentisering og autorisering av kommunikasjon.

## Spring security

Spring security er et stort rammeverk for autentisering og aksesskontroll. Det er standard rammeverket for å sikre spring applikasjoner. Mye av styrken til rammeverket er at det er veldig utvidbart.<sup>43</sup> Et hovedaspekt i spring security er servlet filtre. Det finnes mange forskjellige filtre. Disse brukes til å behandle en forespørsel før den når et endepunkt i applikasjonen. Basert på typen av forespørselen og ressursen den aksesserer opprettes en filter kjede som består av de filtrene som forespørselen behandles av. Man kan utvikle egne filtre for å tilpasse håndteringen av autentisering og autorisering. Figuren under viser hvordan kommunikasjon fra klienter blir behandlet av filtre i filter kjeden før den når et endepunkt.



Figur - Spring security filter basert håndtering av requests

## Autentisering og Autorisering

Sikkerhets konfigurering av applikasjonen finnes i klassen "SecurityConfiguration". Det er her definert slik at enhver request til serveren må være autentisert. Det er utviklet to egendefinerte filtre til dette, disse finnes i mappen "security". Ved innlogging med brukernavn og passord håndteres autentisering av forespørselen av filteret "JWTFilterAuthentication". Dersom brukernavn og passord er korrekt vil det returneres en JWT til klienten. Dette tokenet inneholder claims med

<sup>43</sup> <https://spring.io/projects/spring-security>

informasjon om bruker id, roller og gyldighet, og brukes inngår i påfølgende forespørsler fra klienten.

For autorisering av forespørsler til endepunkts ressurser brukes filteret "JWTFilterAuthentication" her valideres tokenet som inngår i forespørselen. Dersom tokenet er gyldig settes det en autentisering basert på brukeren, som tokenet tilhører, i sikkerhets konteksten. Denne konteksten er unik for hver forespørsel og brukes til å håndtere aksess kontroll til hver forespørsel.

Spring security tilbyr løsninger for sikkerhetshåndtering på metode nivå. Dette bruker vi i controllerne for å administrere tilgang til hver metode basert på roller. Dette gjøres gjennom annotasjoner hvor man kan deklarere hvilke roller som skal ha tilgang til metoden i controlleren.

## Kryptering av http

HTTPS er en sikrere versjon av HTTP.<sup>44</sup> Den store forskjellen er at data-en som sendes over nettet blir kryptert og dermed vil være uforståelig å lese før den har blitt dekryptert igjen.

Det var veldig viktig å sette opp HTTPS støtte for prosjektet. Grunnen til at det var viktig var fordi uten HTTPS vil JWT være sårbar under transport. Hvis man bruker app-en på et offentlig nett vil det øke sannsynligheten for at noen pakkesniffer seg frem til JWT-en. Med JWT vil trusselaktøren kunne utgi seg som den originale eieren av JWT.

I Spring var det relativt lett å konfigurere og sette opp HTTPS. Konfigurasjonene skjedde i *application.properties* filen. Under er de aktuelle verdiene som må bli satt.

**server.ssl.key-store-type=PKCS12**

**server.ssl.key-store=classpath:keystore/x.p12**

**server.ssl.key-store-password=**

**server.ssl.key-alias=**

Verdier er ikke satt med vilje av konfidensiell årsaker. "x.p12" er en sertifikatfil som bruker PKCS12 kryptering

Sertifikatet som blir brukt for å kryptere http må være i en mappe kalt keystore inni *resource* mappen. Det må være av riktig type og samsvare med konfigurasjon satt for type.

Det er viktig at sertifikatet er signert av en pålitelig *Certificate Authority* (CA), hvis ikke kan det oppstå problemer med http-forespørsler fra mobiltelefoner. IOS for

---

<sup>44</sup> <https://www.cloudflare.com/learning/ssl/what-is-https/>



eksempel tillater ikke å utføre http-forespørsler over en https forbindelse med kryptert med et selvsignert sertifikat.<sup>45</sup> Man må isåfall manuelt godkjenne sertifikatet gjennom iOS sine innstillinger.

Anskaffelse av et pålitelig CA signert sertifikat overlates til Accenture.

#### 4.4.17 Admin side

Admin siden konsumerer API-et etter typicode sin JSON-Server spesifikasjon. Hvis man ønsker å gjøre en ressurs tilgjengelig i admin siden må serviceklassen til ressursen implementere *AdminEndpoint* interfacet.

```
/** @param <T> Data transfer object e.g. EventInfoDTO.  
public interface AdminEndpoint<T> {  
  
    PageResponse<T> getList(Map<String, String> filter);  
  
    T getOne(long id);  
  
    List<T> getMany(List<Long> ids);  
  
    T createEntity(T entity);  
  
    T updateEntity(long id, T entity);  
  
    void deleteEntity(long id);  
  
}
```

Deretter må serviceklassen implementere alle metodene knyttet til interfacet. Dette burde være lett i dette tilfellet fordi JSONServerService klassen har allerede implementert forretningslogikken for disse metodene samt følger typicode sin JSON-Server spesifikasjon.

Om man ønsker at admin siden skulle konsumere API-et etter en annen standard må man gjøre større endringer i klientsiden. Mer om dette i [react-admin](#) i [klientsiden](#).

#### 4.4.18 Dokumentasjon

Serversiden består av to typer dokumentasjon. Den første er forretningslogikk dokumentasjon og den andre er REST API dokumentasjon.

Forretningslogikk dokumentasjon blir gjort ved å bruke Javadoc notasjon.<sup>46</sup>

---

<sup>45</sup> <https://medium.com/collaborne-engineering/self-signed-certificates-in-ios-apps-ff489bf8b96e>

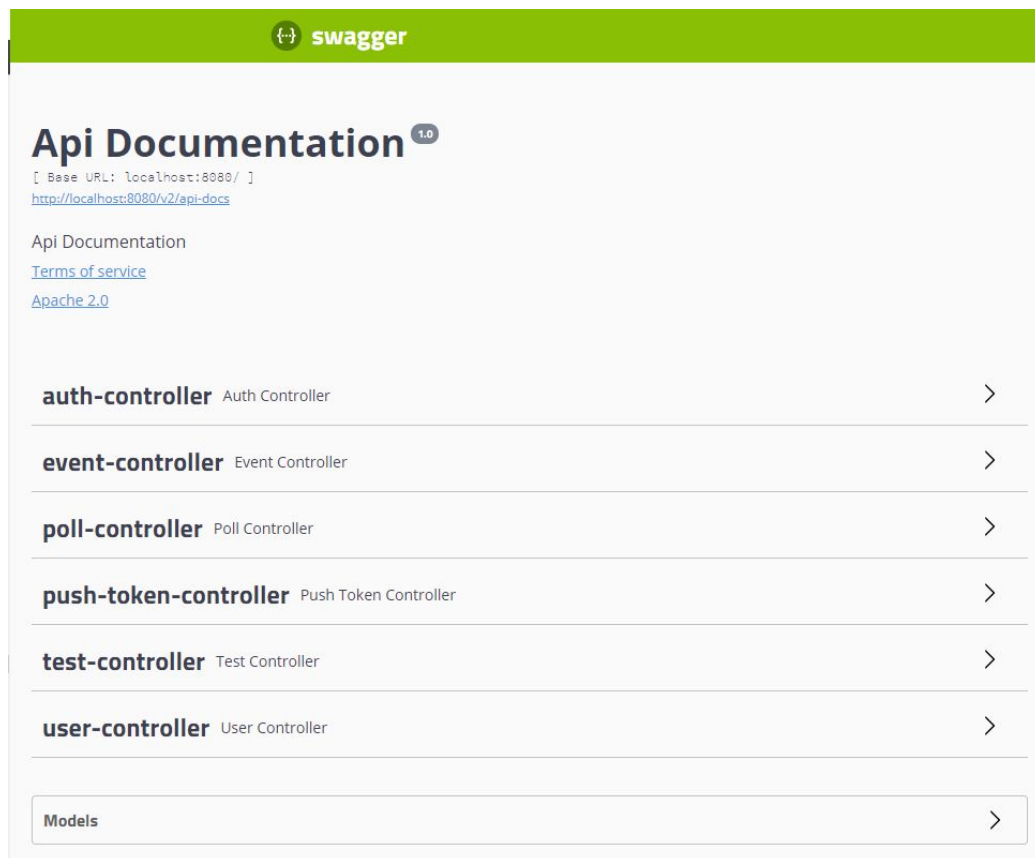
<sup>46</sup> <https://en.wikipedia.org/wiki/Javadoc>

## Eksempel på forretningslogikk dokumentasjon i javadoc notasjon:

```
/**
 * Generell implementasjon som følger marmelab/ra-data-json-server sin implementasjon
 av dataprovider.
 * @param <S> DTO
 * @param <R> Entity
 */
public class JSONServerService<S, R> implements RESTServerStandard<S, R>
```

REST API dokumentasjon er informasjon om et tilgjengelig endepunkt. Det blant annet informerer om: ressursen, http-metoden og spørringsparamtere. I tillegg kan det ofte hende at endepunktet inneholder kommentarer om hva det brukes til og hvordan man burde bruke det. Dokumentasjonen blir hovedsakelig generert av et bibliotek som heter Swagger.<sup>47</sup>

Swagger skanner controllerklassene hver gang applikasjonen starter. Etter skanningen vil det blir produsert en fint formatert side med oversikten over alle endepunkt. Man akseesserer `/swagger-ui.html` for å få opp siden.



Bilde av swagger hovedskjerm

<sup>47</sup> <http://swagger.io>

user-controller User Controller	
GET	/api/users getList
POST	/api/users createEntity
GET	/api/users/{id} getOne
PUT	/api/users/{id} updateEntity
DELETE	/api/users/{id} deleteUser
OPTIONS	/api/users/{id} deleteUser

Bilde av endepunkt tilknyttet til en controller.

#### 4.4.19 Serversiden i produksjon

For å få serversiden ut i produksjon må man sette opp tre ting. Den første er å ha en relasjonell database, database kan kjøre samme sted som hovedserveren. Den andre man må sette opp er hovedserveren og til slutt notifikasjonserveren.

Det er viktig å huske på å ha riktig konfigurasjon for MySQL serveren.

**spring.jpa.hibernate.ddl-auto=update**

**spring.jpa.database-platform=org.hibernate.dialect.MySQL5Dialect**

**spring.datasource.url=jdbc:mysql://localhost:3306/spillgruppe**

**spring.datasource.username=**

**spring.datasource.password=**

Username og passord er tom av konfidensielle årsaker.

Etter riktig konfigurasjon er satt kan man kjøre "mvn package" enten i intellij eller i terminal og gitt at alle testene slår korrekt og man ikke har kompileringsfeil vil man få en .jar fil i /target. Vi starter opp serveren med: "nohup java -jar filnavn.jar&" for å starte prosessen i en ny tråd og at den skal logge hendelser til fil.

For notifikasjonserver starter man programmet ved å kjøre Node NotificationController.js.

## 4.5 Klientsiden

Klientsiden er bygget med relativt ny teknologi. Dette betyr at det er færre sterke meninger om hvordan man skal gjøre noe. Dette er i sterk kontrast til serversiden, der det var nesten bare en måte å implementere visse funksjonaliteter. Det er anbefalt at leser er

### 4.5.1 React Native (RN)

React Native er mobilapplikasjon rammeverk.<sup>48</sup> Rammeverket kom først ut i 2015 og det betyr per dags dato at det bare er 5 år gammelt. Dette regnes som en veldig ung teknologi. Hensikten med rammeverket er å utvikle mobilapplikasjoner med javascript. Hovedsakelig blir javascript og jsx brukt for UI og noe forretningslogikk. Det som gjør RN mektig er at man kan i tillegg til javascript bruk native programmeringsspråk for mobil (Objective-C/Swift for iOS og Java for Android).

### 4.5.2 Expo

Expo er et rammeverk som bygges rundt React Native.<sup>49</sup> Hensikten med rammeverket er å eliminere behovet for å skrive i det native programmeringsspråket til mobilen. Expo fungerer som et mellomledd i mellom react native og utvikleren.

### 4.5.4 Arkitektur

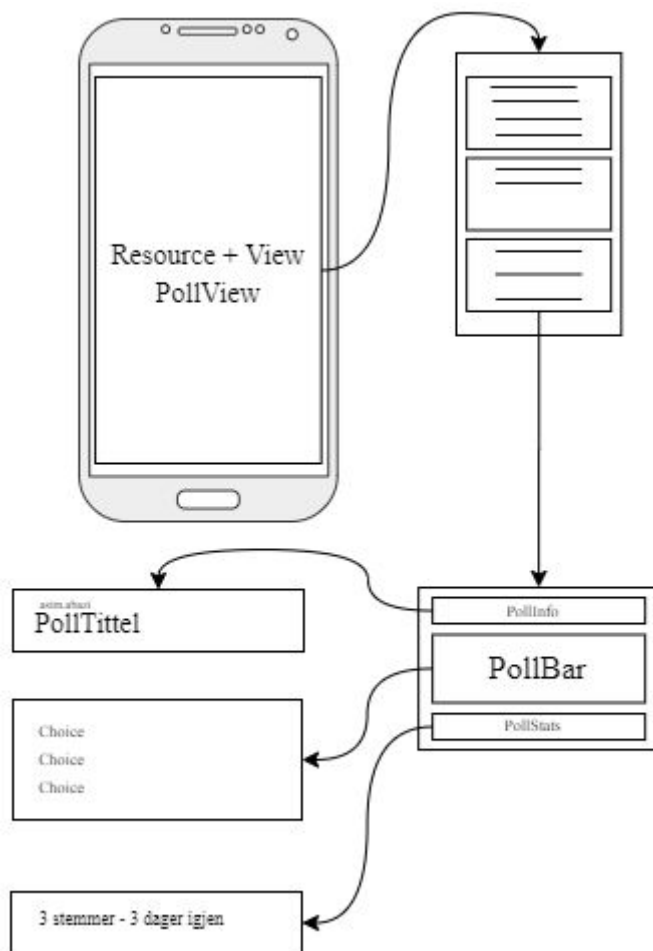
Arkitekturen for mobilappen er veldig simpel. Det innebærer å ta en ressurs som er tilgjengelig i serversiden og lage en overordnet komponent for å håndtere ressursen. Den overordnede komponenten skal hete [Ressurs]+View. Den overordnede komponenten skal være den eneste som gjøre http-forespørsler mot REST API-et.

Den overordnede komponenten har ansvaret for å sende data ned til underkomponenter. Det er i underkomponentene man styler brukergrensesnittet (UI).

---

<sup>48</sup> [https://en.wikipedia.org/wiki/React\\_Native](https://en.wikipedia.org/wiki/React_Native)

<sup>49</sup> <https://docs.expo.io/>



Figur av UI-arkitektur til mobilappen.

#### 4.5.7 Modal

Modal er en design teknikk som blir brukt for å presentere en midlertidig kontekst som er adskilt fra den forrige konteksten.<sup>50</sup> Den blir også brukt for å gi brukeren et fokus på en oppgave. Det er flere elementer i applikasjonen som krever bruken av en modal. Dette førte til at gruppen valgte en modulær løsning som gjorde modal komponenten lett å implementere i andre komponenter.

Modal komponenten blir brukt for å vise en bruker en oppgave som er tilknyttet til hoved konteksten. Modal blir brukt i to deler av applikasjonen, event og poll. Det som er til felles blir lagret i en komponent, og det som er forskjellig blir sendt inn som parametere. Dette gjør komponenten gjenbrukbar og lettere å implementere i andre komponenter.

```

export function AppModal(props: IProps) {
  return (

```

<sup>50</sup> <https://developer.apple.com/design/human-interface-guidelines/ios/app-architecture/modality/>

```

    <Modal
      animationType="slide"
      visible={props.modalVisible}
      onRequestClose={() => {
        props.setModalVisible(false);
      }}
    >
      {props.children}
    </Modal>
  );
}

```

Dette er den fundamentale modalen AppModal, det vil si at det som skal være til felles mellom alle modal skal bli definert her. Vi har forhåndsdefinert animasjon typen og hva som skal skje når noen prøver å lukke modalen. AppModal tar imot parametere som props.children. Dette betyr at denne AppModal kan ta imot andre komponenter, for eksempel event komponenten. Restriksjonene satt av AppModal blir fulgt, men design og logikk blir definert i komponenten som blir sendt inn.

## Modal for nye eventer

Modal komponenten blir brukt i event-delen av applikasjon for å kunne opprette et nytt event. Brukerens kontekst blir fokusert på å fylle ut de nødvendige feltene for å lage et event.

```

export default function CreateEventModal() {
  const [modalVisible, setModalVisible] = useState(false);
  return (
    <View>
      <AppModal modalVisible={modalVisible}
        setModalVisible={setModalVisible}>
        <EventInput
          setModalVisible={setModalVisible}
          modalVisible={modalVisible}
        />
      </AppModal>
      <FAB onPress={() => setModalVisible(true)} icon="plus"></FAB>
    </View>
  );
}

```

Denne komponenten lager en EventModal, den inneholder en AppModal som tar imot en komponent som heter EventInput.

Det er EventInput som inneholder all logikk og

Cancel

Submit

## New Event

Event Title

Event Title is required

Oslo

design for hvordan denne modalen skal se ut for event-delen av applikasjonen. EventInput inneholder en tittel, en lokasjon, om det er en turnering, en beskrivelse, dato og tid. Når dataen er ferdig utfylt blir det gjort en post forespørsel til hovedserveren ved å trykke på submit-knappen. Det som blir sendt er i JSON format og kartlegges til et objekt i hovedserveren.

EventInput validerer alle felt, dersom det er et felt som mangler blir det gitt en feilmelding. Dato og tid blir også validert, dersom datoen har vært blir det vist en dialogboks som opplyser brukeren om dette. Submit knappen blir tilgjengelig ved godkjent validering.

Cancel

Submit

## New Event

CS:GO

Oslo

Tournament

Vi skal ha en CS:GO turnering, blir det ut premier!

Event Date

21. Mai

17:25

### Poll modal

Modal komponenten blir brukt i poll-delen av applikasjonen for å kunne opprette en ny meningsmåling. Dette skal være tilgjengelig for alle brukere og konteksten skal fokusere på det

som kreves for å kunne opprette en meningsmåling.

```
export default function PollModal() {  
  const [modalVisible, setModalVisible] = useState(false);  
  return (  

```

```

    <View>
      <AppModal modalVisible={modalVisible}
setModalVisible={setModalVisible}>
        <PollInput
          modalVisible={modalVisible}
          setModalVisible={setModalVisible}
        />
      </AppModal>
      <FAB
        onPress={() => {
          setModalVisible(true);
        }}
        icon="plus"
      ></FAB>
    </View>
  );
}

```

Koden er implementert i samme struktur som i event-delen. AppModal tar imot en komponent som heter PollInput. PollInput sørger for å definere design og logikk. En funksjon som er verdt å legge merke til er mapping funksjonen. PollInput komponenten starter med et input felt for spørsmål og deretter to input felt for alternativer. Ved å trykke på legg til knappen blir det lagt til flere input felt, dette gjøres ved å iterere gjennom en liste som for hvert element legger til et input felt.

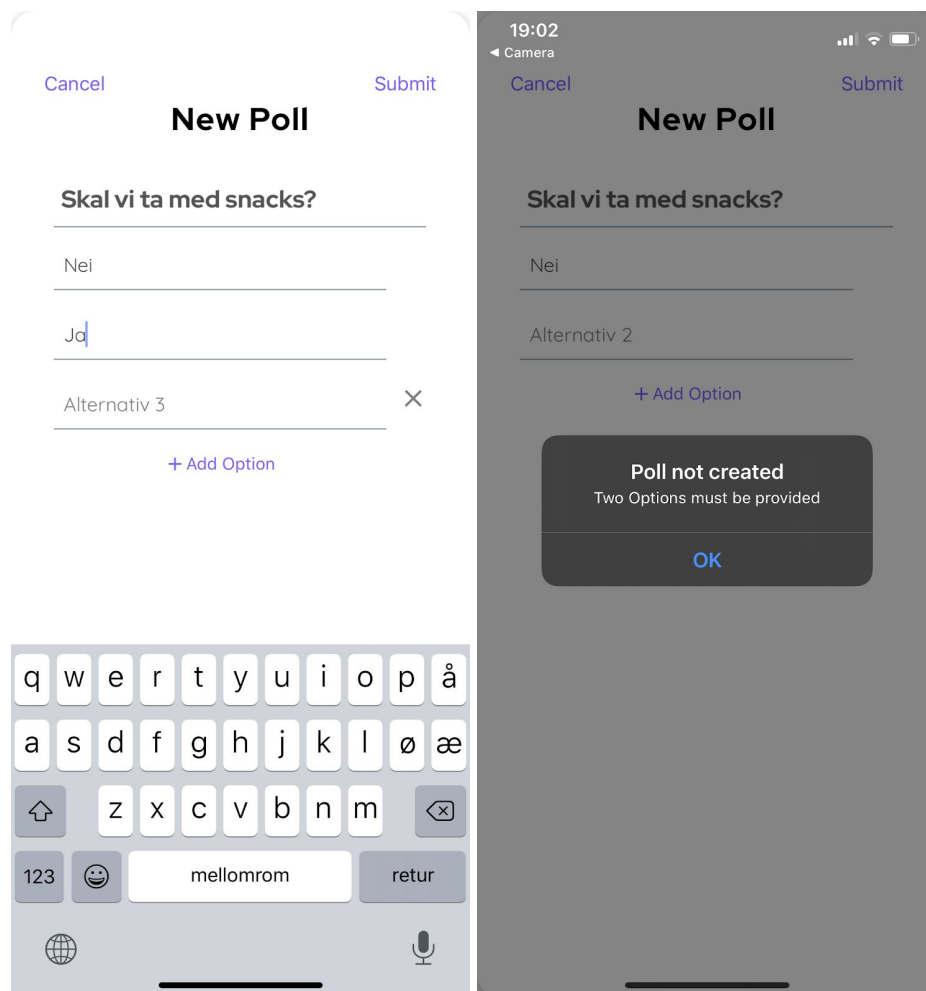
```

choices.map((choice, index) => {
  return (
    <OptionRow
      placeholderText={`Alternativ ${index + 1}`}
      key={index}
      onChangeText={(e) => handleChange(e, index)}
    >

```

Her er choices en liste av string verdier. Den blir instansiert med to elementer og ved å legge til et nytt input felt blir det lagt til en ny string. Deretter blir det kalt en metode som ved endring av tekst oppdaterer det valgte elementet i choices. Når submit knappen blir trukket på blir det gjort en post forespørsel til hovedserveren. Denne forespørselen inneholder et objekt som består av et spørsmål, liste av alternativer og hvilken bruker som opprettet denne meningsmålingen.





Det kan bli lagt til flere alternativer, men det kreves minimum to. Dersom det blir fylt ut kun et alternativ blir det gitt en feilmelding. Ved korrekt utfylling blir det laget en ny meningsmåling.

#### 4.5.8 Design

Design burde samsvare med Apple sin Human Interface Guidelines. Eksempel på hvordan applikasjonen vår håndhever retningslinjene er bruken farge. Farge blir brukt for å trekke oppmerksomheten til brukeren.

#### 4.5.9 Paginering

Paginering er utviklet i klientsiden som en uendelig bla funksjon. Dette betyr at når man blar nedover i applikasjonen så vil

#### 4.5.10 SSE

SSE blir håndtert i en komponent med navn SSEModule. Det er en fullstendig modulær løsning som kan plassere i hvilken som helst View komponent. Ved å tilføre sse endepunkt vil View komponentene automatisk få tilgang til meldinger om

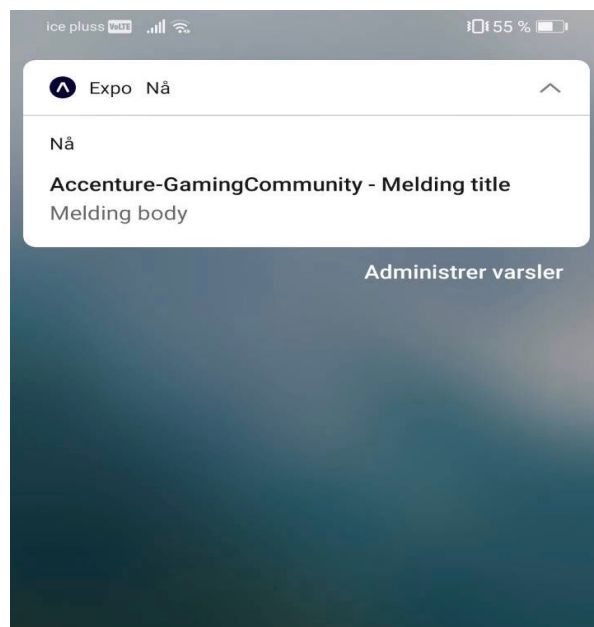
hendelser. Det er opp til utvikler å håndtere hva man gjør når man har fått meldingene.

#### 4.5.11 Notifikasjoner

Push notifikasjon i klientsiden er en melding som dukker opp på en mobil enhet.<sup>51</sup> Dette skal være informasjon som informerer en bruker om en hendelse. Implementasjonen av koden krever en rekke veldefinerte steg som sørger for en god brukeropplevelse.



Skjerm bilde av notifikasjon på iOS.



Skjerm bilde av notifikasjon på android 10.

En push notifikasjon skal opplyse en bruker om en hendelse. Vi har valgt å ta i bruk Apple sin human interface guideline for å kunne sørge for at informasjonen følger en profesjonell standard. Noen av prinsippene de følger er å ha notifikasjoner som er lett leselige og forklarende, ikke sende ut konfidensiell informasjon om en bruker, ikke send flere notifikasjoner angående samme hendelser og sørge for å få tillatelse til å sende notifikasjoner<sup>52</sup>. Disse prinsippene blir fulgt nøye.

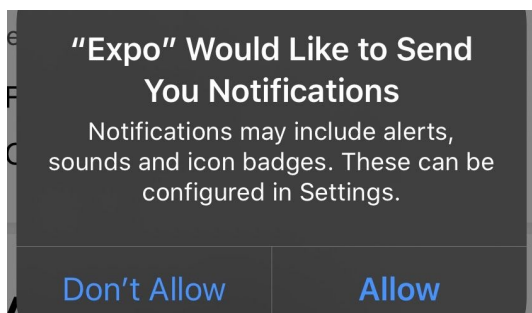
<sup>51</sup> <https://www.airship.com/resources/explainer/push-notifications-explained/>

<sup>52</sup>

<https://developer.apple.com/design/human-interface-guidelines/ios/system-capabilities/notifications/>



Informasjonen som vi formidler må opplyse en bruker om at det har blitt opprettet et nytt event. Strukturen starter med en tittel som er navnet på det nye eventet som ble laget og meldingen er at et nytt event har blitt opprettet. Denne notifikasjonen inneholder også i bakgrunn data som blir brukt i klientsiden for navigering. Slik vi ser i figuren til høyre er notifikasjonen forklarende og teksten er lett å lese.



Ved oppstart av applikasjonen blir en forespørsel om tillatelser til å sende push notifikasjoner vist slik som Apple sin human interface guideline ber om.

Den overordnede flyten for notifikasjoner er som følgende: Ved kjøring av applikasjonen blir det gjort en forespørsel om en bruker vil tillate push notifikasjoner, dersom brukeren tillater push notifikasjoner vil en unik push token bli sendt og lagret til vår hovedserver. Hvis brukeren ikke tillater notifikasjoner vil applikasjonen ikke sende noen push token. Når det blir opprettet et nytt event blir det sendt en notifikasjon til alle brukere som har tillatt notifikasjoner. Når en bruker trykker på notifikasjonen vil den sende brukeren til det nye eventet.

Koden i klientside er skrevet i tre hovedsteg:

1. Forespørsel om tillatelse til å sende push notifikasjoner.
2. Sende push token og bruker id til hovedserver.
3. Håndtere notifikasjonen etter at en bruker har trukket på den.

Det første steget er implementert ved å asynkront sende en forespørsel til en bruker. Basert på om brukeren velger å tillate notifikasjoner eller ikke vil en variabel bli tillagt denne verdien. Dersom en bruker gir tillatelse vil man kunne asynkront få en push token. Denne push token blir autogenerated og unikt identifiserer en mobil enhets expo applikasjon.

```
handleSubmit = async () => {  
    const { status } = await Permissions.askAsync(Permissions.NOTIFICATIONS);  
}
```

Det andre steget tar denne push token og bruker id-en som en bruker er logget inn med og sender en post forespørsel til hovedserveren. Bruker id-en er encoded og må bli decoded før den blir sendt til hovedserveren.

```
const RegisterTokenRequest = (PushToken) => {  
    return {  
        method: "POST",  
        body: JSON.stringify({  
            userid: jwt_decode(this.context.user.token).sub,  
            token: PushToken,  
        })),  
        headers: {  
            Accept: "application/json",  
            "Content-Type": "application/json",  
            Authorization: this.context.user.token,  
        }  
    }  
}
```

Etter at push token har blitt sendt fra klientside og validert i hovedserveren vil en bruker kunne bli tilsendt en notifikasjon. Det tredje steget vil være å håndtere denne notifikasjonen. Når en notifikasjon blir sendt fra notifikasjonserveren er den bygd opp med en melding, tittel, undertittel og data.

Data er det feltet som blir brukt for å håndtere notifikasjonen i klientside. Fra vår hovedserver blir data definert som det klientsiden trenger for å navigere, altså en string som referer til hvilket view og en tallverdi som referer til hvilket event det er snakk om. Når en bruker får en notifikasjon så er det et av to alternativer som kan forekomme, at en bruker trykker på notifikasjonen eller ikke. I klientside er det satt en listener som venter på notifikasjoner, og dersom en bruker trykker på notifikasjonen blir brukeren navigert til det nye eventet.

```
_handleNotification = (notification) => {  
    if (notification.origin !== "selected") {  
        return;  
    }  
}
```

```
    this.props.navigation.navigate(notification.data.navigation, {
      eventId: notification.data.eventId,
    });
  };
};
```

### 4.5.12 Autentisering og autorisering

Autentisering og autorisering er en sentral del av løsningen som ble høyt prioritert. Utviklingen av frontend løsningen for dette ble utført i sprint 3. Her vil det beskrives hvordan dette er implementert i appen og funksjonalitet som er relatert til dette.

#### JWT

JWT er kort for JSON web token og er en token som brukes til autentisering. jwt er json basert og består av en header, payload og signature. JWT kan inneholde informasjon i claims som finnes i payload delen. Dette kan ofte være informasjon om en bruker som id eller brukernavn. Sikkerheten ligger i at tokenet inneholder en signatur som kan brukes til å verifisere at tokenet er gyldig. I frontend inngår dette i alle api requests til serveren. Brukerens roller som inngår som claim i tokenet brukes også til å til å rendere funksjonalitet som er spesifikk for admin og moderator.

#### AsyncStorage

AsyncStorage er et asynkront lagringssystem i react native. Dette lagrer data i en sikker kontekst slik at andre applikasjoner ikke har tilgang til denne dataen. I vår app brukes dette til å lagre og lese jwt.

#### Innlogging

Når man først åpner applikasjonen vil det sjekkes om brukeren allerede er innlogget og har en gyldig token i AsyncStorage. Dersom token er gyldig vil man komme rett inn til innholdet i appen og token lagres i usercontext. Hvis det ikke finnes noen token eller den er ugyldig vil brukeren tas til en innloggingsskjerm.

På innloggingsskjermen blir brukeren bedt om å skrive inn enterpriseld og passord for å logge inn. Når brukeren trykker på "sign in" vil det sendes en login forespørsel til serveren. Dersom forespørselen er gyldig og blir autentisert vil klienten motta en respons med en json web token som lagres i Async Storage og settes i usercontext. Bruker vil deretter navigeres inn til appen innholdet. Hvis passordet eller enterpriseld er feil vil backend autentiseringen feile og en feilmelding vil vises i

appen. Dersom brukeren ikke er registrert vil man kunne navigere til en “sign up” skjerm hvor man kan registrere en ny bruker.

## 4.6 Verktøy

Gruppen benyttet hovedsakelig tre typer verktøy. Utviklingsverktøy, versjonskontrollverktøy og illustrasjonsverktøy. Vi har benyttet oss av både gratis verktøy og verktøy som koster penger, men er inkludert i studiet.

### 4.6.1 IntelliJ IDEA

IntelliJ IDEA faller under utviklingsverktøy kategorien, mer spesifikt så er intellij et integrert utviklingsmiljø. Gruppen har mye erfaring med intellij fordi det har blitt brukt gjennom hele utdanningsperioden for å skrive java kode. Verktøyet ble brukt hovedsakelig under utviklingen av spring boot applikasjonen. IntelliJ har en rekke funksjoner som er veldig nyttige for utvikling av en spring boot applikasjon. Funksjoner som: avansert kodefullføring, kodenavigering som gjør det mulig å hoppe til en klasse eller deklarerer i koden direkte, kodeinnsetting, debugging og integrert maven støtte.<sup>53</sup>

### 4.6.2 Visual Studio Code

Visual Studio Code eller bare “vscode” faller også under kategorien utviklingsverktøy. Vscodet er annerledes fra intellij fordi det er ikke et integrert utviklingsmiljø, men heller en kildekode editor. Gruppen valgte å bruke vscode fordi det var en rekke gratis utvidelser som gjorde utviklingen av klientsiden lettere.

### 4.6.3 Postman

For å enklere jobbe med utviklingen av REST API-et benyttet vi oss av et utviklingsverktøy som heter postman. Postman var veldig nyttig å sjekke om funksjonalitet knyttet til api-et fungerte som det skulle. Den har funksjonaliteter som: sette tilgangstoken og organisering av request url.<sup>54</sup>

### 4.6.4 Git Bash

Git bash er en bash emulator for windows.<sup>55</sup> Den følger med git programmet som ble brukt for versjonskontroll.

---

<sup>53</sup> <https://www.jetbrains.com/idea/features/>

<sup>54</sup> <https://www.postman.com/use-cases/application-development/>

<sup>55</sup> <https://gitforwindows.org/>

#### *4.6.5 Diagrams.net*

Illustrasjonsverktøy som tilgjengelig på nettleser. Alle alle figurer i denne sluttrapporten er laget i diagrams.net. Det tidligere kjent som draw.io.

#### *4.6.6 Github Desktop*

Github er et GUI versjonskontrollverktøy. Det blir brukt for å håndtere merge konflikter og under code reviews.

### 4.7 Forslag til videreutvikling

#### *4.7.1 Paginering i klientsiden*

Paginering i klientsiden har preg duplikatkode. Det kan være aktuelt å lage en Higher order component som håndterer forretningslogikken.

#### *4.7.2 Serveren ut i produksjon*

Å få serveren ut i produksjon er litt slitsomt. Det kan være lurt å ta en titt på containermiljø som f.eks. docker.

## 5 Testing

Gruppen hadde ikke på forhånd av prosjektet mye kunnskap om testing. Den eneste formen for testing som var kjent til en begrenset grad var enhetstesting.

## 5.1 Brukertestning

Brukertestning innebærer å la brukere som er representative av sluttbrukerne teste produktet. Dette er viktig for å teste funksjonalitet uten at det gjøres antagelser basert på implementeringskunnskap som utviklere og produkt nære personer har. Dette vil gi et innblikk i hvordan applikasjonen fungerer fra et brukerperspektiv og kan derfor avsløre feil og mangler.

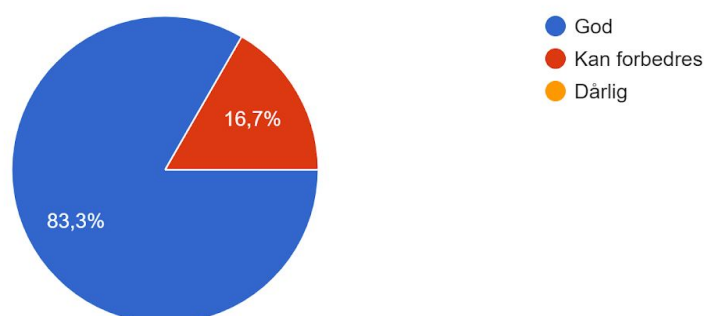
Opprinnelig hadde gruppen planlagt at brukertestning kunne arrangeres på accenture i person og utføres på en testenhed. Korona situasjonen fjernet dette alternativet. Vi valgte isteden å lage et spørreskjema i google skjemaer og tilrettelegge for at personer kunne teste appen ved å laste den ned gjennom expo appen og gi tilbakemelding via google skjema. Vi spurte veilederne om de kunne kontakte medlemmer av spill gruppa da det er disse som ville være slutt brukerne og gi mest relevant tilbakemelding. For å få noen flere testbrukere valgte vi å spørre bekjente om å også gjennomføre testing og svare på skjema.

Nedenunder vil vi gå gjennom tilbakemeldingene hvordan vi tolket dem og hvilke tiltak vi utførte.

### Innlogging og registrering

Hva syns dere om logg på delen av applikasjonen?

6 svar





Er det noe som kan forbedres med logg på siden, i så fall hva?

2 svar

Log in delen var helt fin, klart design. Fikk feilmelding ved feil enterprise.id/passord kombinasjon.

Logg på delen var fin, men registrerings delen kunne gjerne hatt litt mer funksjonalitet som krav til enterprise.id og passord.

### Figur - innlogging brukertesting

Her viser det at de fleste var fornøyd med innlogging og registrering delen av applikasjonen. Vi fikk to tilbakemeldinger på spørsmål om hva som kan forbedres med log in delen. Den ene var bare positiv, mens den andre foreslo mer funksjonalitet på registrerings skjermen som krav til enterprise.id og passord.

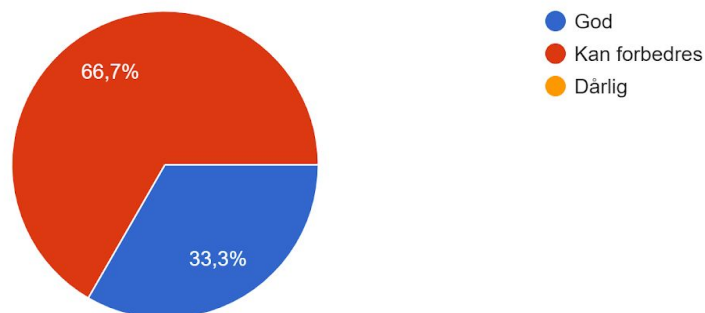
### Tiltak

Vi så ikke noe grunn til tiltak med logg in delen, men valgte å implementere krav til passord lengde ved registrering.

### Events

Hva synes dere om event delen av applikasjonen?

6 svar



Er det noe som kan forbedres med event delen av applikasjonen, i så fall hva?

5 svar

Fikk en feilmelding nederst på skjermen når jeg gikk inn på event innlegg.

Kanskje se på innlegg-funksjonaliteten(måtte gå ut - inn av eventet før kommentaren ble vist). Kanskje også ha en "skal ikke" alternativ

Burde vært en fargeendringer i eventlisten som viser hvilke du har svart på og hvilke du ikke har svart på, og eventuelt hvilke du ikke har sett på.

Fikk feilmelding nederst på skjermen når jeg gikk på innlegg.

Kanskje ha siste innlegg øverst? Og når man deler et innlegg på et event, må man gå ut og inn av eventet før man ser innlegget man la til. "Skal"-knappen er resetta når man går ut og inn på eventet. Får opp en feilmelding når jeg trykker på "Innlegg".

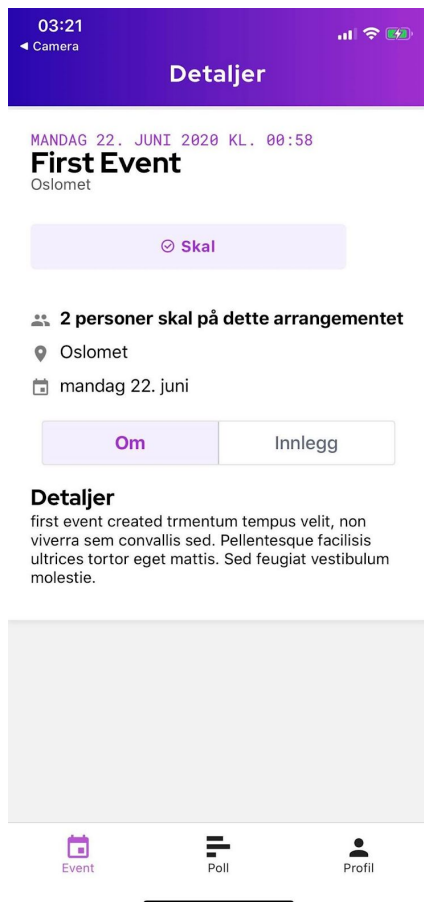
### *Figur - Events Brukertesting*

Det var totalt 4 som mente event delen kunne forbedres mens to svarte at den var bra. I tilbakemeldingene om hva som kan forbedres var det 4 tilbakemeldinger som påpekte at det kom opp en feilmelding nederst på skjermen. Av to av brukerne ble det påpekt at man måtte gå inn og ut av eventet før innlegget ble vist når man opprettet et nytt innlegg og at "skal knappen blir resetta når man går inn og ut". Det var også forslag om farge endring i oversikten over events.

### **Tiltak**

Etter å ha tolket svarene kommer det frem at alle brukerne som svarte "kan forbedres" ga tilbakemelding om at det vises en feilmelding. Dette var en advarsel og vil ikke vises i et produksjons tilfelle og er ikke relevant for sluttproduktet.

Fra de to brukerne med andre forslag var det tilbakemelding om at "skal" knappen ble resetta. Vi har sett på dette og forsikret oss at dette nå ikke forekommer. Dette var en feil som kun skjedde ved noen events og er nå rettet. Vi fikk også tilbakemelding om at det var vanskelig å vite at komponenten som åpnet deltaker modal var klikkbar. Vi endret derfor skriften på den trykkbare komponenten for å vise deltakere, slik at den så mer markert ut ved å endre tekst stilen. Endringen er vist under.

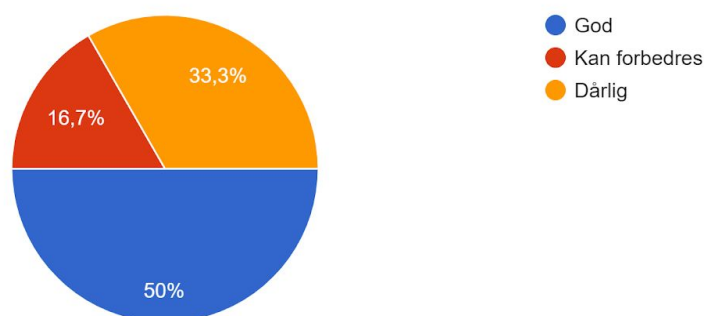


Figur - Event detaljer og endring av tekst stil

## Event Opprettelse

Hva syns dere om funksjonaliteten til lag event?

6 svar



Er det noe som kan forbedres med lag event modalen?

5 svar

Fikk feilmelding når jeg lagde event. Ellers bra.

var grei den, men jeg fikk ikke opprettet et event

input feltene var fine. Men fikk en feilmelding ved opprettelse.

Får feilmelding her response status 500 når jeg forsøker å opprette et event. Ellers kan det kanskje tydeliggjøres litt hva man kan endre på, selv om det sier seg selv, men bare for å se forskjell på hva man kan og ikke kan klikke på.

Alt funka men fikk feilmelding når jeg lagde event

### *Figur - Opprett event*

Her var det to tilbakemeldinger som mente funksjonaliteten til lag-event var dårlig, en som mente den kan forbedres og tre som mente den var god. På de skriftlige tilbakemeldingene som omhandlet hva som kan forbedres kom det imidlertid fram at de fikk en feilmelding når de opprettet event.

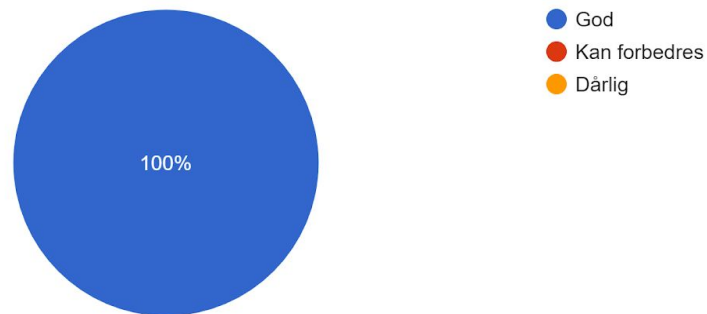
### **Tiltak**

Etter å ha undersøkt dette fant vi ut at det kom opp en feilmelding for testerne. Feilmeldingen kom opp på grunn av at notifikasjons serveren ikke var kjørende under testingen. Løsningen fungerte ellers som den skulle og bruker testerne fikk opprettet eventer, men det bare så ikke slik ut på grunn av feilmeldingen som vises. Dette forklarer mesteparten av de negative tilbakemeldingene. Det eneste andre forslaget var å tydeliggjøre det man kan endre på. Vi vurderte dette men kom fram til at det var tydelig nok i modalen, og ikke svært viktig da dette kun er funksjonalitet som skal brukes av admin og moderator som kjenner appen godt.

## Polls

Hva syns dere om poll delen av applikasjonen?

6 svar



Er det noe som kan forbedres med poll delen av applikasjonen, i så fall hva?

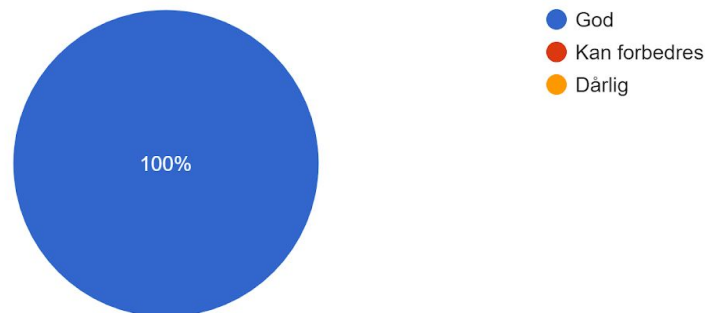
0 svar

Det finnes foreløpig ingen svar på dette spørsmålet.

### Figur - Poll brukertesting

Hva syns dere om funksjonaliteten til lag poll?

6 svar



Er det noe som kan forbedres med lag poll, i så fall hva?

0 svar

Det finnes foreløpig ingen svar på dette spørsmålet.

### Figur - Opprett Poll brukertesting

Tilbakemeldingene på visning og opprettelse av poll var alle bra. Det var ingen forslag om forbedring. Vi vurderte derfor ingen endringer basert på dette.

## Design og flyt

Hva syns dere om designet til applikasjonen iforhold til farger?

6 svar

Header på event og poll tabene burde ikke være så annerledes. Kanskje litt lite farger.

det var bra

skrevet øverst

Header burde være lik. Kunne vært mer farger.

"Events" overskriften kan sammenstemme med overskriften til "Poll".

Syntes det kunne vært noe mere farger

Ved spørsmål om design og farger fikk vi blant annet kommentarer på header i event og poll oversikt, og event detaljer som ikke var like. Vi fikk også kommentarer på at det kunne vært mer farger.

## Tiltak

Headeren til i event oversikten og ble oppdatert fra å være hvit til å matche den som brukes i polls. Dette var noe vi hadde planlagt å gjøre men ikke bestemt ferdig hvordan skulle se ut. Skjermen som viser event detaljer ble også oppdatert til å matche de andre, men med annerledes farge.

Hva syns dere om designet til applikasjonen iforhold til størrelser og proposjoner?

6 svar

Greit det

bra

god

greit

Størrelsene ser gode ut.

Flott

Ved spørsmål om design i forhold til størrelser og proposjoner svarte alle positivt. Vi gjorde ingen endringer basert på dette.

Hva syns dere om flyten i applikasjonen i helhet?

Helt grei

bra

grei

Noen ganger så hakker den (vet ikke om den bare "refresher"). Når man trykker på knappene for å komme inn i en fane, burde den kanskje scrolle til toppen (smak og behag antar jeg).

Flott

Ved spørsmål om flyt svarte de fleste positivt. Det var en kommentar om at appen noen ganger hakker. Vi gjorde ingen endringer basert på dette.

## Resultater

Alt i alt var vi fornøyd med testene. Det var noen negative tilbakemeldinger, men etter å ha undersøkt dette fant vi ut at de fleste skyldes at notifikasjons løsningen ikke var kjørende noe som forårsaket feilmeldinger under testingen. Under testingen kom det også fram en feil knyttet til deltakelse på events, som vi fikk rettet. Ellers fikk vi gode tilbakemeldinger på applikasjonen. Vi fikk også gjort appen

mer brukervennlig gjennom å bedre kommunisere hva som er klikkbart, i tillegg til å oppdatere farger og design.

## 5.2 Enhetstester

Enhetstestning var gjort med JUnit 5. Testningen var gjort på forretningslogikken i serversiden.

Skjermdump av resultatet.

```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 55, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 25.604 s
[INFO] Finished at: 2020-05-25T07:03:09+02:00
[INFO] -----
```



# 6 Brukermanual

## ACCENTURE GAMING

### *Oppbygging av brukermanual*

## 6.1 Forord

Accenture gaming er en eventhåndterings applikasjon. Den er designet med et mål om god brukervennlighet. Funksjonalitet skal derfor være intuitiv og applikasjonen kunne brukes uten spesifikk forkunnskap. Vi har likevel laget en brukermanual for å presentere appen, og illustrere flyten og funksjonaliteten som tilbys.

Bruerveiledningen er fordelt i to deler, første delen tar for seg hva en bruker har tilgang til av funksjonaliteter og den andre delen omhandler administrering av applikasjonen. Brukermanualen skal sørge for at leseren får forståelse av hva applikasjonen tilbyr og hvordan den løser forskjellige problemer. Vi har også lenket til [feilmeldingsmanualen](#) til Expo dersom det er noe avvik i notifikasjonserveren.

## 6.2 Innledning

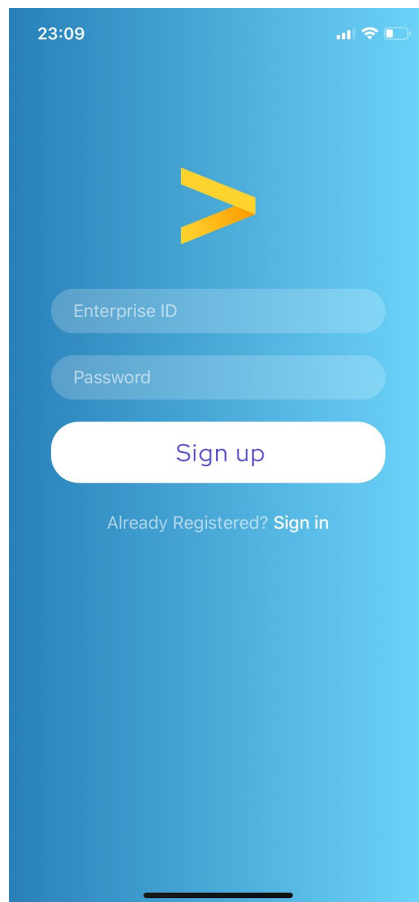
Applikasjonen er beregnet for alle som er i spillgruppen til Accenture og aktivt deltar i events. Formålet er å simplifisere oppgavene knyttet til det å arrangere eventer. Dette kan være å delta eller ikke, å opprette nye meningsmålinger og å se innlegg fra andre brukere. I tillegg er det laget en webapplikasjon for administrasjon. Mobilapplikasjonen er tilgjengelig for både iOS og Android.

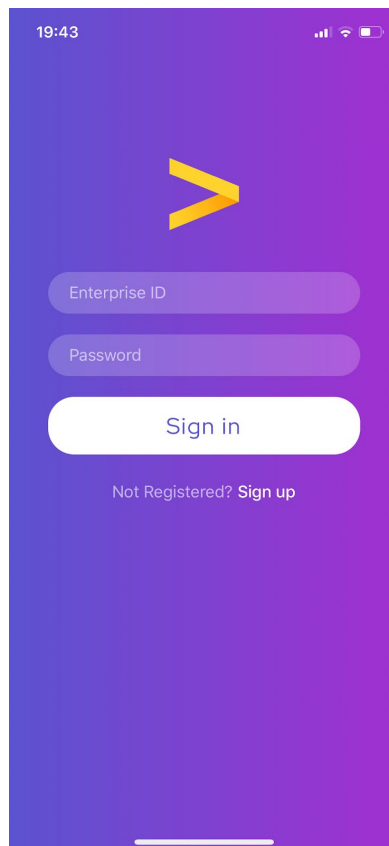
## 6.3 Brukerflyt

Brukerflyten viser hvordan en bruker kan registrere seg og ta i bruk applikasjonens funksjonaliteter.

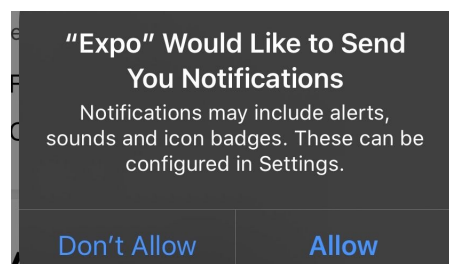
### *6.3.1 Innlogging og registrering*

Når applikasjonen kjøres for første gang blir brukeren presentert en registreringsside. Her må brukeren registrere seg, dersom det er første gang de bruker applikasjonen.

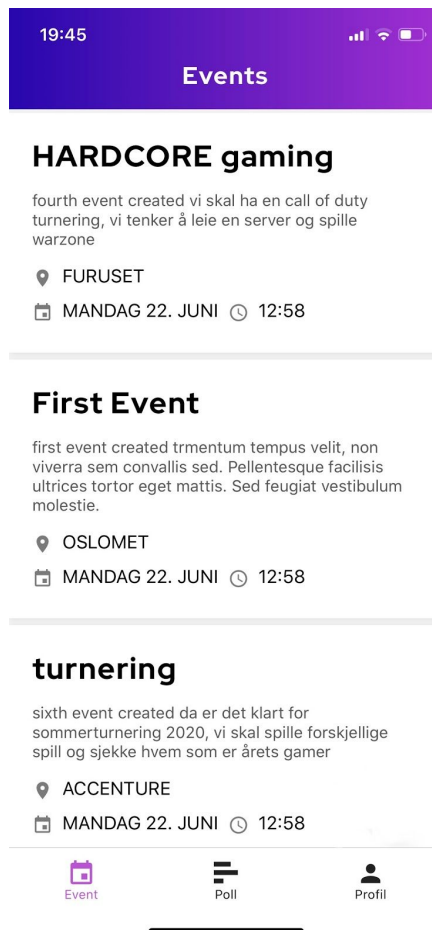




Etter at en bruker er registrert blir de vist til innloggingsside. Her skal en bruker kunne logge seg inn ved å oppgi brukernavn og passord.

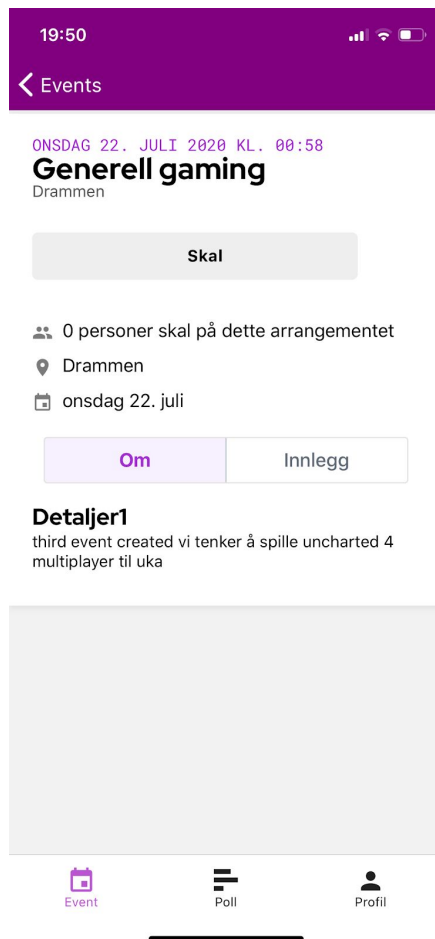


Ved godkjent brukernavn og passord blir brukeren logget inn til å se en dialogboks som spør om tillatelse til å sende notifikasjoner.

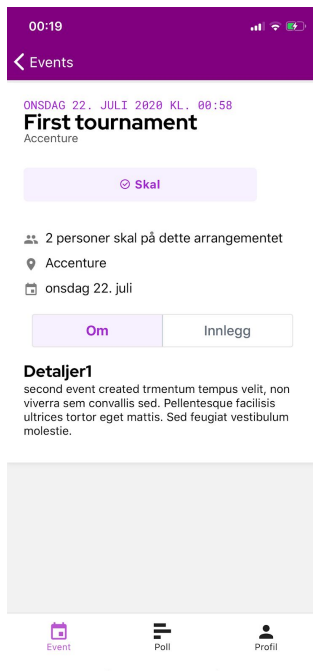


### 6.3.2 Event og funksjonaliteter

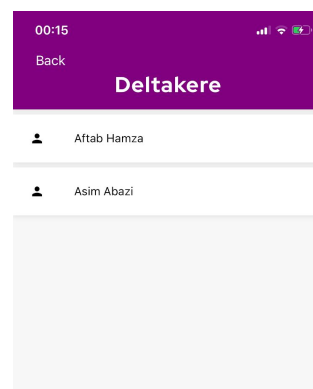
Brukeren blir presentert en liste av elementer som har en tittel, en beskrivelse, en lokasjon, dato og tid. Dette utgjør et event. Det er en navigeringsbar nederst som kan sende brukeren til meningsmålinger og profil.



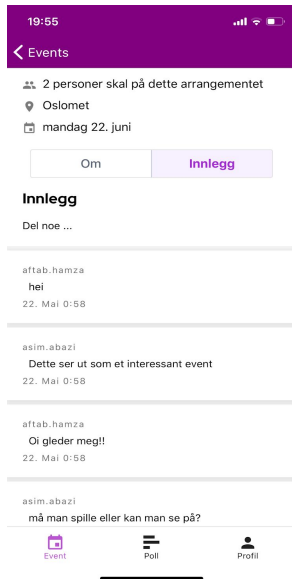
Når en bruker velger et av elementene blir de sendt til en ny side som gir informasjon om dette eventet. Her blir det vist hvor mange som deltar, lokasjon, beskrivelse og innlegg. Klikkbare objekter er feltene *Skal*, *Personer*-ikonet, *Om* og *Innlegg*.



Ved å trykke på *Skal*-knappen skal blir det lagret at denne brukeren skal delta på eventet og fargen endres. Dette for å gi brukeren et signal på at de har valgt å delta.

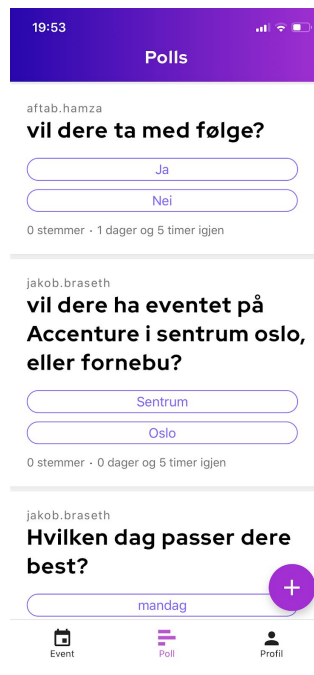


Ved å trykke på *Personer*-ikonet vises det en liste av alle som deltar på dette eventet. Dersom brukeren trykker på tilbake knappen blir de sendt til den forrige siden.



Ved å trykke på *Innlegg*-knappen blir det vist en liste av innlegg. Der står det hvem som laget innlegget, en tekst og en dato for når innlegget ble skrevet.

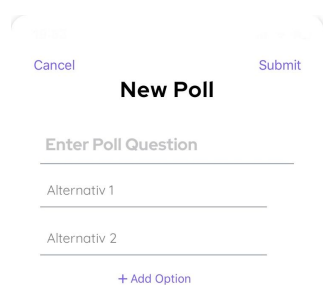
### 6.3.3 Poll og funksjonaliteter



Ved å navigere til *Poll*-siden blir brukeren vist en liste av meningsmålinger. Hvert element viser et navn, et spørsmål, en liste av alternativer, antall stemmer og en

utløpsdato. Denne dataen utgjør en meningsmåling. En bruker kan trykke på alternativene og *Pluss*-ikonet.

Ved å velge et av alternativene blir elementet oppdatert og det blir vist hvor mange prosent som har stemt på hvert alternativ. I tillegg blir antall stemmer oppdatert.



The screenshot shows a modal window titled "New Poll". At the top left is a "Cancel" button and at the top right is a "Submit" button. Below the title is a text input field labeled "Enter Poll Question". Underneath this are two more text input fields labeled "Alternativ 1" and "Alternativ 2". At the bottom of the modal is a link that says "+ Add Option".

Dersom en bruker trykker på *Pluss*-ikonet blir det vist frem en modal med funksjonalitet for å opprette en ny meningsmåling. Her skal det skrives et spørsmål og minimum to alternativer for at meningsmålingen skal bli opprettet. Ved å klikke på *Add Option* blir det satt inn flere input-felt.



The screenshot shows a 'New Poll' form. At the top, there are 'Cancel' and 'Submit' buttons. Below them is the title 'New Poll'. The form contains a text input field labeled 'Enter Poll Question'. Below this are six rows, each with a label 'Alternativ 1' through 'Alternativ 6' and a corresponding input field. To the right of each input field is a small 'X' icon, which is used to remove the alternative. The form is enclosed in a light gray border with rounded corners.

New Poll	
Enter Poll Question	
Alternativ 1	
Alternativ 2	
Alternativ 3	X
Alternativ 4	X
Alternativ 5	X
Alternativ 6	X

Det kan velges hvor mange alternativer en bruker vil gi meningsmålingen, dette tallet må være mellom to og seks. Det kan fjernes alternativer ved å klikke på *Kryss*-ikonet. Ved å fylle ut en meningsmåling korrekt vil en bruker bli sendt tilbake til *Poll*-siden, der den nye meningsmålingen også blir fremvist.



**Aftab Hamza**

---

Sign out



Event



Poll

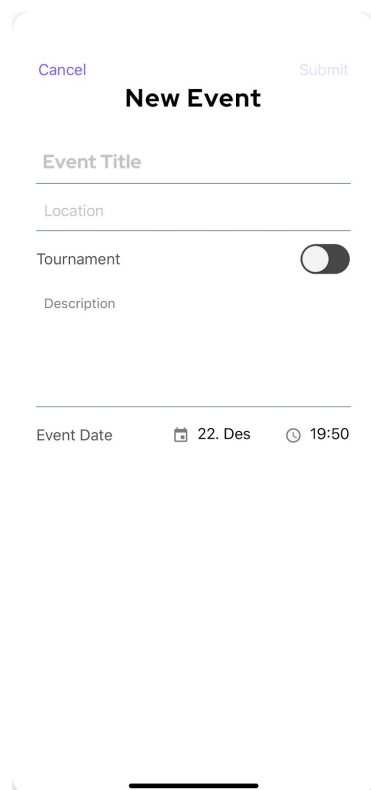
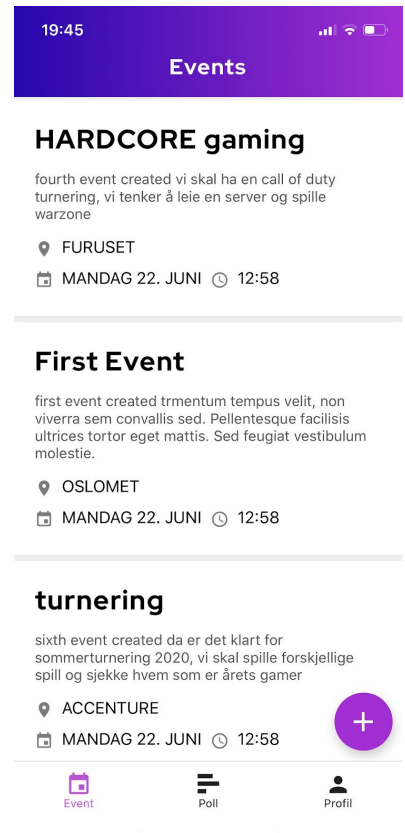


Profil

Profil-siden blir brukt for å logge ut og å sjekke hvem som er logget på.

## 6.4 Administrasjon

Dersom n bruker har rollen *Admin* får de tilgang til mer funksjonalitet i mobilapplikasjonen. Et *Pluss*-ikon blir vist i event siden for å kunne opprette et nytt event. All annen data blir vist som tidligere beskrevet.



Etter å ha trukket på *Pluss*-ikonet blir det fremvist en modal med funksjonalitet for å opprette et nytt event. Her skal det gis en tittel, en lokasjon, om det er en turnering, en beskrivelse, en dato og tid. Nødvendige felt er tittel, dato og tid.

Cancel Submit

### New Event

Test event

---

Oslo

---

Tournament ☒

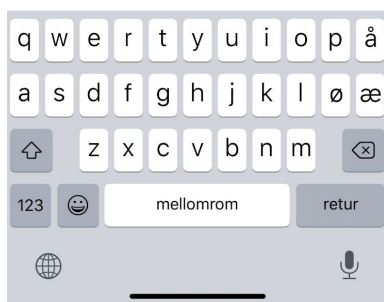
---

Test event

---

Event Date 22. Des 19:50

Etter å ha fylt ut de nødvendige feltene vil *Submit*-knappen bli tilgjengelig. Ved å klikke på *Tournament*-knappen blir fargen endret.



19:52 Cancel Submit

### New Event

Event Title

---

Location

---

Tournament ☐

---

Description

---

Pick a date

19	September	2017
20	October	2018
21	November	2019
22	December	2020
23	January	2021
24	February	2022
25	March	2023

Confirm

Cancel

Ved å trykke på dato og tid komponentene blir det vist en ny modal som fremhever hvert deres element.

19:52 Cancel Submit

### New Event

Event Title

---

Location

---

Tournament ☒

---

Description

---

Pick a time

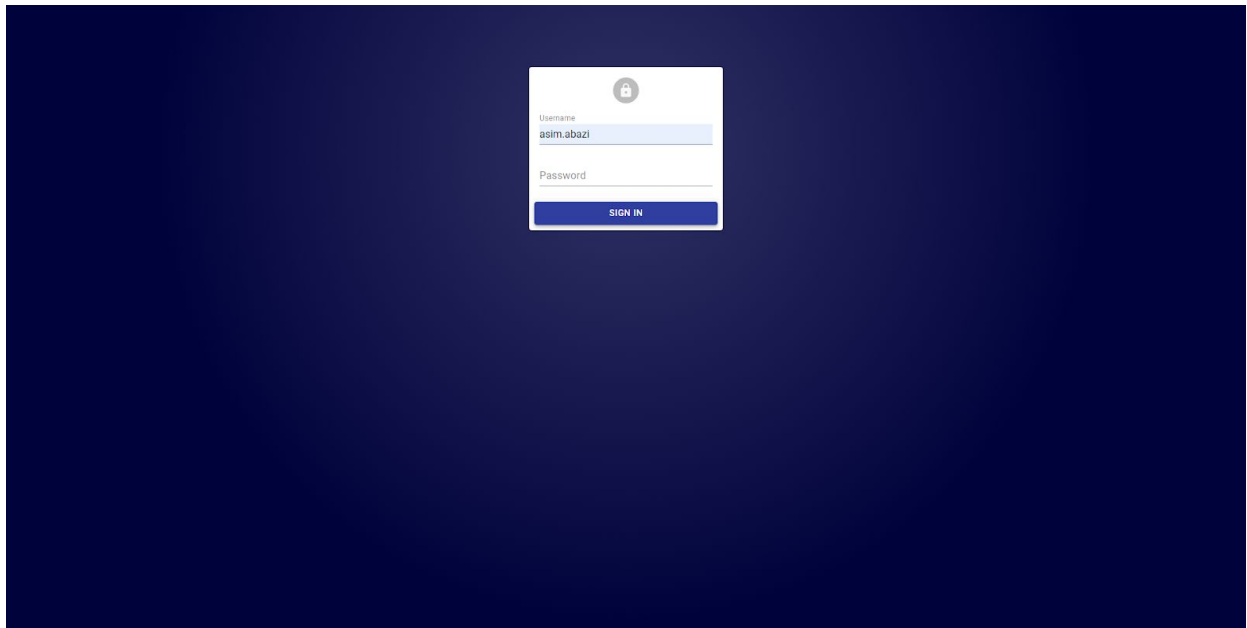
16	29
17	30
18	31
19	32
20	33
21	34
22	35

Confirm

Cancel

### 6.4.1 Admin webapplikasjon

Ved å kjøre webapplikasjonen blir det vist en logginnside. Her må en bruker med



*Admin*-rolle skrive inn brukernavn og passord.

### Users

<input type="checkbox"/>	Id ↕	Enterprise
<input type="checkbox"/>	1	asim.abazi
<input type="checkbox"/>	2	aftab.hamza
<input type="checkbox"/>	3	jakob.braseth
<input type="checkbox"/>	4	lars.henrik.nordli
<input type="checkbox"/>	5	daniel.tafjord
<input type="checkbox"/>	6	adam.asskali
<input type="checkbox"/>	7	sean.elliott
<input type="checkbox"/>	8	josefina.cortes
<input type="checkbox"/>	9	manoe.moll
<input type="checkbox"/>	10	امیرمحمد.زید.ایران
<input type="checkbox"/>	11	malou.christiansen
<input type="checkbox"/>	12	patrick.morris
<input type="checkbox"/>	13	özkan.özgürkey
<input type="checkbox"/>	14	maria.theresia.schier
<input type="checkbox"/>	15	jessie.patterson

Etter å ha logget inn vil *Admin* bli vist en liste av brukere. Dette vil være alle som er med i spillgruppen til Accenture. Det blir vist en id og en enterprise-id. Her kan det søkes og sorteres.

Api/users

Users

Events

maria

EXPORT

<input type="checkbox"/>	id ↑	Enterprise
<input type="checkbox"/>	14	maria-theresia.schier
<input type="checkbox"/>	88	maria.mendez
<input type="checkbox"/>	107	christa-maria.knauf
<input type="checkbox"/>	120	marina.vargas
<input type="checkbox"/>	178	marika.van.toorn
<input type="checkbox"/>	200	marisa.guerin

Rows per page: 1-6 of 6

Ved å trykke på søkefeltet kan en *Admin* søke på andre brukere. Her kan det søkes på både navn og id.

## Events

Api/events

Users

Events

+ CREATE

EXPORT

<input type="checkbox"/>	id	Title	Dato ↑	Opprettet av	
<input type="checkbox"/>	4	HARDCORE gaming	24.6.2020, 02:33:36	aftab.hamza	<a href="#">EDIT</a>
<input type="checkbox"/>	7	turnering	24.6.2020, 02:33:36	jakob.braseth	<a href="#">EDIT</a>
<input type="checkbox"/>	1	First Event	24.6.2020, 02:33:36	asim.abazi	<a href="#">EDIT</a>
<input type="checkbox"/>	6	BRETTSPILL	24.7.2020, 02:33:36	jakob.braseth	<a href="#">EDIT</a>
<input type="checkbox"/>	5	Generell gaming	24.7.2020, 02:33:36	aftab.hamza	<a href="#">EDIT</a>
<input type="checkbox"/>	2	First tournament	24.7.2020, 02:33:36	asim.abazi	<a href="#">EDIT</a>
<input type="checkbox"/>	3	Generell gaming	24.7.2020, 02:33:36	aftab.hamza	<a href="#">EDIT</a>
<input type="checkbox"/>	8	sammenkomst	24.8.2020, 02:33:36	jakob.braseth	<a href="#">EDIT</a>

Rows per page: 10 1-8 of 8

Events inneholder alle events tilgjengelig fra databasen. Her står det en id, tittel, dato og hvem som opprettet eventet. Her kan det sorteres basert på id, tittel, dato og hvem som opprettet eventet.

Api/events

Users

Events

+ CREATE

EXPORT

<input type="checkbox"/>	id	Title	Dato	Opprettet av ↓	
<input type="checkbox"/>	6	BRETTSPILL	24.7.2020, 02:33:36	jakob.braseth	<a href="#">EDIT</a>
<input type="checkbox"/>	8	sammenkomst	24.8.2020, 02:33:36	jakob.braseth	<a href="#">EDIT</a>
<input type="checkbox"/>	7	turnering	24.6.2020, 02:33:36	jakob.braseth	<a href="#">EDIT</a>
<input type="checkbox"/>	5	Generell gaming	24.7.2020, 02:33:36	aftab.hamza	<a href="#">EDIT</a>
<input type="checkbox"/>	3	Generell gaming	24.7.2020, 02:33:36	aftab.hamza	<a href="#">EDIT</a>
<input type="checkbox"/>	4	HARDCORE gaming	24.6.2020, 02:33:36	aftab.hamza	<a href="#">EDIT</a>
<input type="checkbox"/>	1	First Event	24.6.2020, 02:33:36	asim.abazi	<a href="#">EDIT</a>
<input type="checkbox"/>	2	First tournament	24.7.2020, 02:33:36	asim.abazi	<a href="#">EDIT</a>

Rows per page: 10 1-8 of 8

Ved å trykke på pilen ovenfor et av feltene som for eksempel *Opprettet av*, vil listen bli sortert basert på feltet. Her blir listen sortert på brukernavn synkende.

The screenshot shows a web application interface for editing an event. The top navigation bar is blue with a hamburger menu icon, the text 'Api/event #6', and icons for refresh and user profile. On the left, a sidebar contains 'Users' and 'Events' with corresponding icons. The main content area is a form for editing event #6. It includes a 'Title' field with the value 'BRETTSPILL', a 'Description' field with the text 'fifth event created ja folkens det blir en brettspill dag for oss denne uken, kanskje noen slange spill eller uno', an 'Event timestamp' field with the value '24.07.2020 02:33', and a 'Tournament' toggle switch that is currently turned on. At the bottom of the form are two buttons: a blue 'SAVE' button and a red 'DELETE' button.

Ved å trykke på *Edit*-knappen ved et event kan admin redigere eventet. Her kan tittel, beskrivelse, dato og turnering endres. Eventet kan både lagres og slettes.

The screenshot shows a web application interface for creating a new event. The top navigation bar is blue with a hamburger menu icon, the text 'Create Api/event', and icons for refresh and user profile. On the left, a sidebar contains 'Users' and 'Events' with corresponding icons. The main content area is a form for creating a new event. It includes a 'Title' field, a 'Description' field, an 'Event timestamp' field with a placeholder 'dd.mm.åååå --:--', and a 'Tournament' toggle switch that is currently turned off. At the bottom of the form is a blue 'SAVE' button.

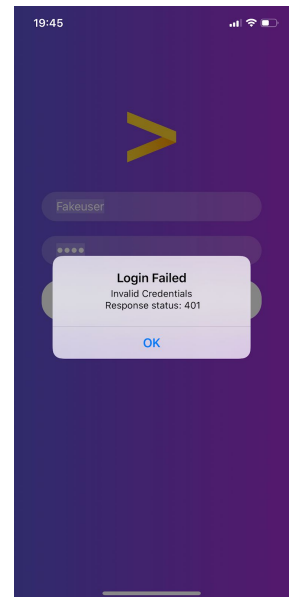
Ved å trykke på *Create*-knappen kan admin opprette et nytt event. Her gis det en tittel, beskrivelse, dato og om det er en turnering eller ikke.



## 6.5 Feilmeldinger og advarsler

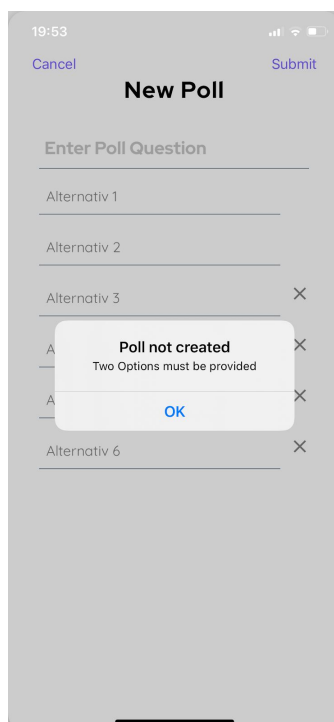
### 6.5.1 Innlogging

Dersom brukeren gir feil brukernavn eller passord blir det vist en dialogboks som fremhever denne informasjonen. Denne feilmeldingen har kode 401, som betyr at opplysningene ikke er korrekte.



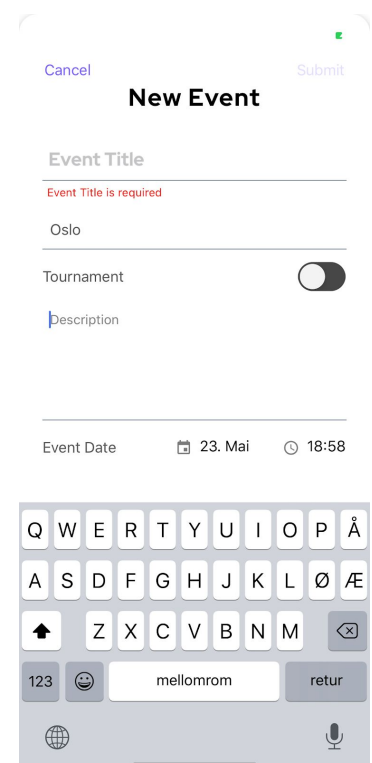
### 6.5.2 Poll

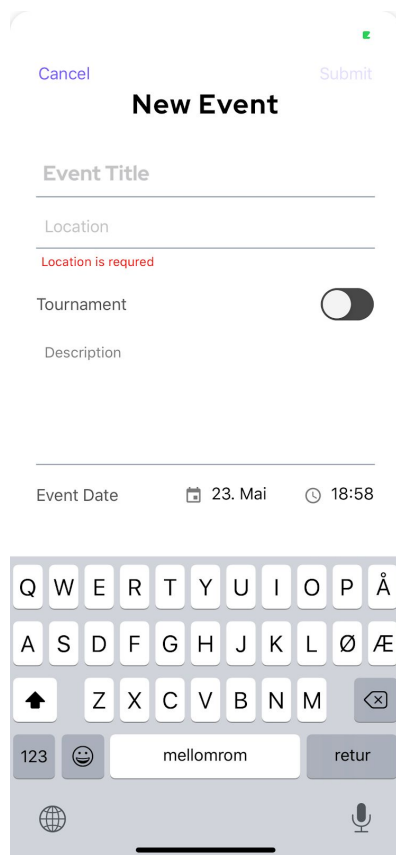
Dersom en bruker ikke fyller ut meningsmålingen og prøver å trykke på *Submit*-knappen, vil det bli vist en dialogboks med en feilmelding. Dette betyr at en bruker må fylle ut minimum to alternativer og et spørsmål.



### 6.5.3 Event

Dersom en bruker med rolle *Admin* ikke gir en tittel vil det bli gitt en feilmelding som sier at tittelen mangler.

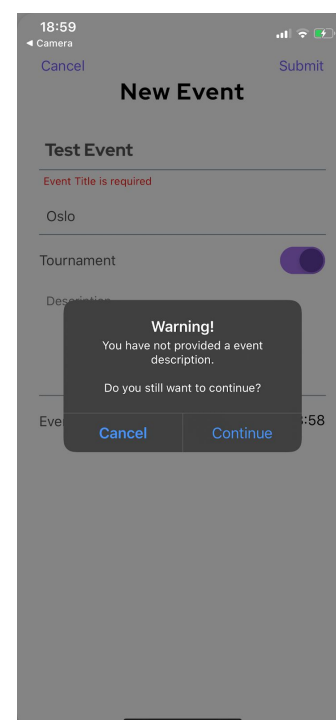




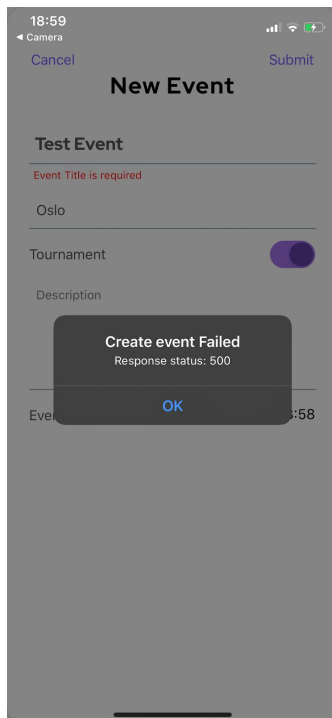
A mobile app interface for creating a new event. At the top, there are 'Cancel' and 'Submit' buttons. The title 'New Event' is centered. Below it are input fields for 'Event Title', 'Location', and 'Description'. A red error message 'Location is required' is visible below the 'Location' field. There is a 'Tournament' toggle switch. At the bottom, the 'Event Date' is set to '23. Mai' and '18:58'. A keyboard is shown at the bottom with the word 'mellomrom' entered in the 'Description' field.

Dersom en bruker med *Admin* rolle ikke skriver en lokasjon, vil det bli gitt en feilmelding som opplyser brukeren om å gi en lokasjon.

Hvis en bruker ikke skriver en beskrivelse av eventet blir det gitt en advarsel som opplyser brukeren om dette.

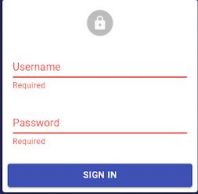


The same 'New Event' form as above, but with a 'Warning!' dialog box overlaid. The dialog box contains the text: 'Warning! You have not provided a event description. Do you still want to continue?'. It has 'Cancel' and 'Continue' buttons. The background form is dimmed, showing the 'Test Event' title and the 'Event Title is required' error message.



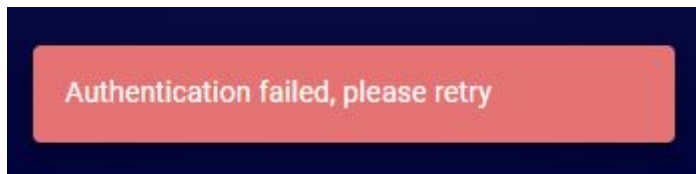
Feilmelding 500 blir gitt dersom det er noe som avviker i notifikasjonsserveren. Feilmeldinger i notifikasjonsserveren er lenket til i forord.

### 6.5.4 Admin-webapplikasjon



The image shows a login form for an admin web application. The form is centered on a dark blue background. It consists of a white box with a lock icon at the top. Below the icon are two input fields: 'Username' and 'Password'. Each field has a red underline and the word 'Required' below it. At the bottom of the white box is a blue button labeled 'SIGN IN'.

Ved å ikke oppgi brukernavn eller passord vil det bli gitt en feilmelding til brukeren om dette. Dette blir markert med røde farger.



Dersom det blir gitt feil brukernavn eller passord blir det vist en dialogboks med feilmelding. Det kreves riktig brukernavn og passord for å gå videre.

# Ordliste

## ACID-prinsippet

A (Atomicity) C (Consistency) I (Isolation) D (Durability) er et sett av krav som sikrer at operasjoner på databaser utføres pålitelig:

<https://no.wikipedia.org/wiki/Database#ACID-prinsippet>

## Administrator

I denne forstand er en administrator en overordnet bruker i datasystemet med tilgang til ekstra administrativ funksjonalitet.

## Administratorside

Administratorside referer til web-brukergrensesnittet for en administrator.

## API

Api, Application Program interface, et grensesnitt i programvare som definerer protokoll for kommunikasjon med ekstern programvare:

<https://no.wikipedia.org/wiki/Programmeringsgrensesnitt>

## Autorisasjon

Autorisasjon i denne kontekst referer til hvilken grad av adgang en bruker har til ressurser:

<https://en.wikipedia.org/wiki/Authorization>

## Backend

Backend referer til den delen av programvaren som ligger nærmest databasen:

<https://snl.no/backend>

## Bibliotek

Et bibliotek refereres til som en samling av funksjoner og klasser som ikke nødvendigvis er et selvstendig program, men som kan benyttes i annen programvare:

[https://no.wikipedia.org/wiki/Bibliotek\\_\(programvare\)](https://no.wikipedia.org/wiki/Bibliotek_(programvare))

## Business logic

Omhandler forretningslogikken i applikasjonen:

[https://en.wikipedia.org/wiki/Business\\_logic](https://en.wikipedia.org/wiki/Business_logic)

## Certificate Authority

Innen kryptografi referer dette til en som utsteder digitale sertifikater. Et digitalt sertifikat forsikrer at den som navngis i sertifikatet er innehaver av en gitt nøkkel. Dette betyr at en som har et digitalt sertifikat kan bli stolt på av andre parter:

[https://en.wikipedia.org/wiki/Business\\_logic](https://en.wikipedia.org/wiki/Business_logic)

## Client

I denne konteksten referer client til maskinvare som kan aksessere en service tilgjengeliggjort av en server:

[https://en.wikipedia.org/wiki/Client\\_\(computing\)](https://en.wikipedia.org/wiki/Client_(computing))

## Client-server forhold

Referer til forholdet mellom maskinvare som aksesserer service fra en server og serveren som tilgjengeliggjør servicen.

## Crud

C (Create) R (Read) U (Update) D (Delete) er fire basis funksjoner som blir brukt for persistent lagring. Hver funksjon kan bli kartlagt til en HTTP metode slik som CREATE blir PUT/POST:

[https://en.wikipedia.org/wiki/Create,\\_read,\\_update\\_and\\_delete](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete)

## Databaseentiteter

En databaseentitet referer til et objekt med data som blir lagret til databasen. Dette objektet blir vanligvis lagret i en egen tabell.

## Data mapping

Data mapping referer til å kartlegge data elementer mellom to distinkte data modeller. Dette kan for eksempel være like attributter mellom to forskjellige objekttyper:

[https://en.wikipedia.org/wiki/Data\\_mapping](https://en.wikipedia.org/wiki/Data_mapping)

## Datasource

Datasource referer til kilden av data, dette kan være en fil, database eller lignende.

## Datatilgangslaget

Omhandler et lag som simplifiserer data aksess til lagrede elementer i en programvare:

[https://en.wikipedia.org/wiki/Data\\_access\\_layer](https://en.wikipedia.org/wiki/Data_access_layer)

## Dependency Inversion Principle

Prinsippet omhandler det å skape et program med minimum avhengigheter mellom moduler:

[https://en.wikipedia.org/wiki/Dependency\\_inversion\\_principle](https://en.wikipedia.org/wiki/Dependency_inversion_principle)

## Edge-cases

I programmering refereres edge-cases til et problem som forekommer i inputverdier ved maksimum eller minimum og krever spesiell håndtering i form av for eksempel en algoritme. Dersom en algoritme fungerer med maksimum parametere og minimum parametere burde alt i mellom også fungere.

## Expo

Expo referer til en plattform som tilbyr å lage applikasjoner for forskjellige operativsystemer med operativsystem-funksjonalitet ved bruk av javascript og react.

## FORM

FORM er navnet på Accenture sin egen arbeidsmetodikk. Den baserer seg på prinsipper og konsepter fra smidig utvikling og Design thinking er et viktig del av metodikken.

## Git

git er et distribuert versjonskontroll verktøy. les mer på <https://git-scm.com/>

## H2 database

H2 database er et relasjonsdatabase system som er skrevet i java. Blir hovedsakelig brukt for testing ettersom at den sletter og setter inn data etter hver økt relativt raskt:

[https://en.wikipedia.org/wiki/H2\\_\(DBMS\)#Main\\_features](https://en.wikipedia.org/wiki/H2_(DBMS)#Main_features)

## Http

En protokoll som primært brukes for å utveksle informasjon:

<https://no.wikipedia.org/wiki/HTTP>

## Http forespørsler

Http forespørsler referer til tilgjengelige forespørselsmetoder som for eksempel GET, DELETE og PUT.

## Https

Https er en sikrere utgave av Http protokollen. Krypterer data og sørger for at uvedkommende ikke kan avlytte eller endre data:

<https://snl.no/HTTPS>

## In-memory database

En in-memory database er et databasehåndteringssystem som bruker RAM for lagring av data. Dette fører til at ved å lukke en økt vil denne dataen bli borte. Blir ofte brukt for testing siden data ikke trenger å hentes fra disk hver gang.

## Instansevariabel

Instansvariabler er variabler definert i en klasse utenfor konstruktøren, metoder og blokker. Blir også referert som et objekts egenskaper.

## Intellij

Et integrert utviklingsmiljø for utvikling av dataprogramvare.

## Interface

Et interface er en type som definerer oppførsel. Interface deklarerer metoder, men implementerer ingen. Metoden skal definert i klassen som implementerer interfacet.



## Java

Java er et programmeringsspråk. Det er et imperativt, objektorientert, kompilert høynivåspråk som kan kjøre på mange forskjellige plattformer ved hjelp av en virtuell maskin: Java Virtual Machine.

## Json

JSON er en enkel tekstbasert standard for å formatere dokumenter som blir brukt for datautveksling. Formatet er bundet av en nøkkel med en verdi:

<https://en.wikipedia.org/wiki/JSON>

## Jwt

JSON Web Token er en internettstandard for å lage JSON-baserte tilgangstokener som hevder et visst antall krav, disse blir brukt for sikkerhetsaspektet av en programvare:

[https://en.wikipedia.org/wiki/JSON\\_Web\\_Token](https://en.wikipedia.org/wiki/JSON_Web_Token)

## Klasse

En klasse er skrevet av en programmerer i en definert struktur for å lage et objekt. Det blir definert et sett av attributter og metoder som tilhører til objekter av denne type klasse:

[https://simple.wikipedia.org/wiki/Class\\_\(programming\)](https://simple.wikipedia.org/wiki/Class_(programming))

## Kodebase

Kodebase er en samling av kildekode som blir brukt for å opprette et software system, en applikasjon eller en programvare:

<https://en.wikipedia.org/wiki/Codebase>

## Konsumere

I denne sammenheng blir konsumere brukt for å forbruke data.

## Metadata

Metadata er definert som data som gir informasjon om annen data:

<https://en.wikipedia.org/wiki/Metadata>

## Mikrotjeneste

Mikrotjenester er en programvareutviklings-teknikk som ordner en applikasjon som en samling av løst koblede tjenester:

<https://en.wikipedia.org/wiki/Microservices>

## Monolittisk

I programvareteknikk er en monolittisk applikasjon en en-lags programvare som kombinerer brukergrensesnitt og datatilgangskode til en plattform:

[https://en.wikipedia.org/wiki/Monolithic\\_application](https://en.wikipedia.org/wiki/Monolithic_application)

## Native (mobil) app

App som er utviklet for en spesifikk plattform, mest vanlig IOS og android. Dette gjøres med teknologi som er spesifikk for utvikling til denne plattformen.

## Navneplan

Referer til hva slags konvensjon det brukes for å tildele navn til forskjellige type dataholdere i et system.

## Objekt instans

En instans av et objekt betyr at det finnes et objekt allokert i minne:

[https://en.wikipedia.org/wiki/Instance\\_\(computer\\_science\)](https://en.wikipedia.org/wiki/Instance_(computer_science))

## OS-plattform

En OS-plattform er et software system som håndterer maskinvare:

[https://en.wikipedia.org/wiki/Operating\\_system](https://en.wikipedia.org/wiki/Operating_system)

## Rammeverk

Et rammeverk er software som simplifiserer oppgaven med å lage en programvare eller et system. Allerede eksisterende kode blir brukt for å håndtere lav-nivå kode, slik at en utvikler kun trenger å fokusere på det de vil skreddersy. Her kan de legge kode til den allerede eksisterende koden og unik programvare eller system.

## React Native

Et rammeverk som blir brukt for å lage applikasjoner med tilgang til OS-funksjonaliteter ved bruk av react.

## Relasjonsdatabase

En relasjonsdatabase er en database som bygger på relasjonsmodellen. Tabellene er forbundet av nøkler mellom seg. Dette fører til at man kan aksessere en tabell via en annen på grunn av nøkkelen som kobler dem sammen:

[https://en.wikipedia.org/wiki/Relational\\_database](https://en.wikipedia.org/wiki/Relational_database)

## Request-url

En request-url er en forespørsel i form av en url som svarer med det som ble spurt om. Dersom det blir sendt en forespørsel om data skal denne bli sendt i en url der serveren tolker den og svarer med den forespurte dataen.

## Screen - page

Brukes til å referere til det som vises på skjermen tilsvarende "page" på et nettsted

## Scrum

Et smidig utviklings rammeverk for utvikling av programvare.

## Separation of concerns

Separation of concerns er et designprinsipp for å skille et dataprogram i distinkte seksjoner slik at hver seksjon adresserer en egen bekymring:

[https://en.wikipedia.org/wiki/Separation\\_of\\_concerns](https://en.wikipedia.org/wiki/Separation_of_concerns)

## Server

En server er en programvare som tilbyr en eller flere tjenester til andre maskinvarer kalt klienter.

## Servermiljø

Et servermiljø er et miljø på en maskinvare som lar en server kjøre.

## Singleton

Singleton er et designmønster som innebærer at man begrenser antall instanser av en klasse til kun et objekt. Dette designmønsteret blir brukt der det kreves kun en instans av et objekt.

## Spring Boot

Spring Boot er et java-basert rammeverk som blir brukt for å lage mikrotjenester.

## Statuskode

Statuskode er en respons fra en server etter at det har blitt gjort en forespørsel fra en klient.

## Tilgangstoken

En tilgangstoken inneholder sikkerhetsinformasjon angående en påloggingsøkt. Den kan bli brukt for å identifisere en bruker, en gruppe eller et annet program. Dette er opp til systemet som har implementert tilgangstoken:

[https://en.wikipedia.org/wiki/Access\\_token](https://en.wikipedia.org/wiki/Access_token)

### **Trello**

En webapplikasjon for å organisere og styre prosjektarbeid. Dette gjøres ved å visualisere arbeidsoppgaver og arbeidsflyt med tavler, lister og kort.

### **url**

En uniform ressurslokator identifiserer og navngir en ressurs ved hjelp av lokaliseringsinformasjon eller en ressurs sin adresse:

[https://no.wikipedia.org/wiki/Uniform\\_Resource\\_Locator](https://no.wikipedia.org/wiki/Uniform_Resource_Locator)

### **Visual Studio Code**

Visual Studio Code er en kildekode editor utviklet av Microsoft.