Eliassen, Severin Aas
Hatlø, Andreas Moe
Nøstdahl, Jahn-Willy Aasen

Bacheloroppgave

Sjakkarena

En webapplikasjon for organisering av sjakkturneringer

Mai 2020

NTNU
Norges teknisk-naturvitenskapelige universitet.
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for IKT og realfag

NTNU
Kunnskap for en bedre verden

Bacheloroppgave

2020

NTNU

Kunnskap for en bedre verden

Eliassen, Severin Aas
Hatlø, Andreas Moe
Nøstdahl, Jahn-Willy Aasen

Sjakkarena

En webapplikasjon for organisering av sjakkturneringer

Bacheloroppgave
Mai 2020

NTNU
Norges teknisk-naturvitenskapelige universitet.
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for IKT og realfag

NTNU
Kunnskap for en bedre verden

Obligatorisk egenerklæring/gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

Du/dere fyller ut erklæringen ved å klikke i ruten til høyre for den enkelte del 1-6:		
1.	**Jeg/vi erklærer herved at min/vår besvarelse er mitt/vårt eget arbeid, og at jeg/vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.**	☒
2.	**Jeg/vi erklærer videre at denne besvarelsen:** ☐ ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. ☐ ikke refererer til andres arbeid uten at det er oppgitt. ☐ ikke refererer til eget tidligere arbeid uten at det er oppgitt. ☐ har alle referansene oppgitt i litteraturlisten. ☐ ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	☒
3.	**Jeg/vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§14 og 15.**	☒
4.	**Jeg/vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert i Ephorus, se Retningslinjer for elektronisk innlevering og publisering av studiepoenggivende studentoppgaver**	☒
5.	**Jeg/vi er kjent med at høgskolen vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens studieforskrift §31**	☒
6.	**Jeg/vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider**	☒

FORORD

Denne rapporten dokumenterer det arbeidet som er gjort i forbindelse med utviklingen av sjakkturneringsapplikasjonen, Sjakkarena. Prosjektet har blitt gjennomført i samarbeid med Aalesunds Schaklag ved Arne Unneland.

Grunnen til at sjakklubben ønsket en slik applikasjon er at løsningen som har blitt benyttet tidligere føles utdatert og ikke egner seg godt til uoffisielle turneringer. Vi håper at den utviklede løsningen vil være med på å forenkle organiseringen av turneringer og være enkel å bruke.

Gruppen vil gjerne takke oppdragsgivers kontaktperson, Arne Unneland, for all hjelp og forslag han kom med underveis i utviklingen av prosjektet.

En stor takk går til Anniken Karlsen for veiledning i rapportskriving, utførelsen av prosjektet og for å stadig pushe gruppen til å få mest mulig ut av oppgaven.

INNHOLD

SAMMENDRAG

Aalesunds Schaklag har gitt i oppgave å lage en applikasjon som kan benyttes til å organisere uoffisielle sjakkturneringer. Denne rapporten inneholder det teoretiske grunnlaget for utviklingen av applikasjonen, en beskrivelse av det ferdige produktet, samt en beskrivelse av utviklingsprosessen.

I applikasjonen skal turneringsverter kunne opprette turneringer som spillere kan melde seg på. Det er tenkt at applikasjonen i hovedsak vil bli brukt i samlingslokaler hvor turneringsvert viser informasjon om turneringen på en storskjerm. Slik turneringsinformasjon er blant annet turnerings-id, som spillere vil bruke til å melde seg på turneringer, og tabeller som viser spillernes plasseringer og pågående partier.

Spillere som skal melde seg på en turnering må ha en egen enhet. På sin enhet oppgir spillerne en turnerings-id gitt av turneringsvert og et, for turneringen, unikt spillernavn. I løpet av turneringen vil spillerne bli tildelt partier. Ved slutten av hvert parti vil spillerne legge inn resultatet av partiet. Basert på spillerens prestasjoner i turneringen, vil spilleren få tildelt en ny motstander når resultatet er lagt til.

I utviklingen av applikasjonen har det vært lagt spesielt stor vekt på interaksjonsdesign, turneringsoppsett og effektiv kommunikasjon mellom server og klienter. For å få et godt brukergrensesnitt har bl.a. gestaltprinsippene og Donald Norman sine designregler blitt benyttet. Systemet for å organisere sjakkturneringer i applikasjonen er basert på turneringssystemet Monrad og tilpasset oppdragsgivers ønsker. For å kommunisere mellom frontend og backend har det blitt benyttet WebSockets for å oppdatere klienter når de ikke vet når ny informasjon er tilgjengelig på server. Restkall over HTTP har blitt brukt når dette ikke er tilfellet.

Applikasjonen består av en frontend som er utviklet med JavaScript-rammeverket Vue, en Java backend utviklet med Spring Boot og en tilhørende MySQL database. Applikasjonen ble laget som en webapplikasjon som passer på alle typer skjermstørrelser. Spillerdelen har blitt utviklet med et spesielt fokus på mobilskjermer. Delen av applikasjonen som brukes av turneringsverter er tenkt skal benyttes på større skjermer, men fungerer også på mindre enheter.

English summary

Aalesunds Schaklag has asked for an application which can be used to organize informal chess tournaments. This report contains the theoretical foundation for the development of the application and a description of the finished product. Additionally, the process of development is described.

By using the application, tournament hosts can create tournaments which players can enroll in. The application is designed for being used in conjunction with physical gatherings. The tournament host should display tournament information found in the application on a large screen visible for all the contestants. Such information is, among other things, a tournament id the players will use to sign up for the tournament and a leaderboard.

Each player who want to sign up for a tournament will use his own device. To sign up for a tournament, the player uses the tournament id and creates his own unique username. During the tournament the players will be provided games. By the end of each game, the players are responsible for adding the result of the game. Based on the performance of a player in the tournament, the player will receive a new game after the result has been registered.

During the development of the application the main focus has been the user interface, the system for organizing tournaments and the way the server and clients communicates. The gestalt principles and Donald Normans design rules have had a

considerable influence on the user interface. The system for organizing tournaments is based on the Monrad system and has been adapted to the wishes of Aalesunds Schaklag. WebSocket has been used in the communication between the server and clients in any case where the clients are not aware of new information being available. In all other cases, REST calls over HTTP are used.

The frontend of the application is built using the Javascript framwork, Vuejs. Spring Boot has been used to create the backend of the application. In addition, data is stored in a MySQL database. The part of the application meant for players will mainly be shown on mobile screens, while the part of the application meant for tournament hosts will mainly be shown on large screens. The designs of these parts have been influenced by which screens they mainly will be displayed on. However, all parts of the application can be used on all types of screens.

TERMINOLOGI

Begreper

Attributt	«En attributt er navn til en verdi eller flere verdier i en entitet eller post i en database» (Bratbergsengen, 2019).
Kandidatnøkkel	«One or more attributes whose values uniquely identify individual entities» (Ponniah, 2003, p. 219).
Primærnøkkel	«One of the candidate keys selected to serve as the unique identifier for the entity type» (Ponniah, 2003, p. 219).
Fremmednøkkel	«En attributt eller gruppe av attributter som er nøkkel i en annen tabell kalles fremmednøkkel» (Bratbergsengen, 2017).

Forkortelser

JWT	JSON Web Token
MVC	Model-view-controller
NPM	Node Package Manager
URI	Uniform ressursidentifikator
DOM	Document object model
ECDSA	Elliptic Curve Digital Signature Algorithm

1 INNLEDNING

De siste årene har sjakk økt sin popularitet i Norge. Dette ser en blant annet i forbindelse med verdensmesterskapene i sjakk hvor NRK sender timevis med sjakkrelaterte programmer. Også turneringer som Norway Chess, Tata Steel Chess og Grand Chess Tour blir kringkastet via Tv2s sportskanaler. I tillegg har Norges sjakkforbund hatt en betydelig økning av sitt medlemstall (Norges Sjakkforbund, 2020).

Den økte sjakkinteressen har økt behovet for hjelpemidler til å organisere sjakkturneringer. Aalesunds Schaklag bruker i dag et program kalt Turneringsservice. De mener at dette programmet ikke egner seg for uoffisielle turneringer. I tillegg ser de på programmet som utdatert. Sjakklubben har derfor bedt om en applikasjon som spillere skal kunne bruke til å melde seg inn i turneringer. Videre er det applikasjonens ansvar å organisere turneringen. Organiseringen innebærer blant annet å finne motstander til deltakerne, starttidspunkt og bord for partiene. Underveis i turneringen skal turneringsverten kunne administrere turneringen. Deltakerne skal kunne følge med på egne resultater og tabell for turneringen. Det er deltakernes oppgave å registrere resultatet fra sine parti. En mer detaljert kravspesifikasjon finnes i vedlegg 1.

I denne rapporten vil følgende problemstilling bli besvart:

- Hvordan lage en applikasjon for organisering av uoffisielle sjakkturneringer?

For å besvare denne problemstillingen vil følgende delproblemstillinger bli besvart:

- Hvordan sikre at applikasjonen har et godt brukergrensesnitt?
- Hvordan lage et turneringsoppsett som tillater at spillere melder seg på turneringen etter start?
- Hvordan dele applikasjonsrelevant informasjon effektivt mellom brukerne?

Denne rapporten inneholder en teoridel hvor relevante teorier og prinsipper blir presentert. Etter dette har rapporten en materiell- og metodedel hvor prosjektets fremgangsmåte blir beskrevet. Fremgangsmåten inkluderer blant annet organisering av prosjektet og hvilke verktøy og programmeringsspråk som har blitt brukt. En beskrivelse av prosjektets resultat blir presentert i resultatdelen av rapporten. En diskusjon rundt resultatet og gjennomføring av prosjektet finnes i drøftingsdelen. Til slutt vil de viktigste resultatene og erfaringene bli oppsummert i konklusjonsdelen.

2 TEORETISK GRUNNLAG

Dette kapittelet inneholder teorier som har blitt brukt til å utvikle sjakkturneringsapplikasjonen. Dette er blant annet teorier om interaksjonsdesign, datamodellering, objektorientert programmering og turneringsplanlegging.

2.1 Interaksjonsdesign

I et system som skal samhandle med brukere, er det viktig å ha et godt designet brukergrensesnitt. Aalesund Schaklag har blant annet ønsket at applikasjonen skal være intuitiv med oversiktlige sider og formålstjenlig bruk av tekst og farger (se vedlegg 1).

Til hjelp for designere av brukergrensesnitt har det blitt laget en rekke designprinsipper og regler. Noen av disse designprinsippene og reglene blir presentert i denne seksjonen.

2.1.1 Donald Normans designregler

Donald Norman presenterte i sin artikkel «Design Rules Based On Analyses of Human Error» (1983) flere regler for brukergrensesnittdesign. Hovedfokuset til disse designreglene er hvordan man kan redusere eller håndtere feilbruk av et system.

Bedre tilbakemelding for å unngå modusfeil

I artikkelen beskriver Norman modusfeil som noe som finner sted når en bruker tar feil av hvilken tilstand programmet han eller hun bruker er i. Dette kan føre til at brukeren forsøker å gjøre ting som ikke er mulig. Et eksempel på en slik modusfeil er når en bruker trykker på en knapp uten at det tilsynelatende skjer noe. Selv om programmet har registrert knappetrykket og er i gang med å prosessere dette, kan brukeren begynne å tvile på at knappetrykket er registrert. Dette kan så føre til at brukeren forsøker å trykke på samme knapp for å få programmet til å gjøre det det allerede er i gang med å gjøre. Det er også mulig at brukeren forsøker å trykke på andre knapper som normalt ville ha satt i gang prosesser som ikke lenger er mulig.

En løsning på slike modusfeil er ifølge Norman å gi tilstrekkelig tilbakemelding på de handlinger som brukeren gjør i systemet.

Bedre systemkonfigurasjon for å unngå beskrivelsesfeil

Donald Norman definerer i samme artikkel en beskrivelsesfeil som når det er uklart beskrevet hvilken konsekvens en handling har. Et eksempel på en slik feil er når man har flere like knapper som gjør forskjellige ting. Dersom en bruker av programmet ønsker å gjøre en ting, kan han eller hun komme til å trykke på en knapp som gjør noe annet. Kun på grunn av at brukeren ikke forstod hva som var forskjellen mellom knappene. Ved å gi klare beskrivelser eller hint om hva en handling vil medføre, kan man unngå beskrivelsesfeil.

Dersom man ikke er konsekvent kan det oppstå flere feil

Norman mener at brukere ofte benytter seg av tidligere erfaringer når de bruker et program. En bruker kan være vant til å gjøre noe på en bestemt måte for å oppnå en bestemt ting. For eksempel kan brukeren ha erfart at en knapp utformet på en bestemt måte har en bestemt funksjon. Dersom en tilsvarende utformet knapp har en annen funksjon enn det brukeren er vant til, kan dette føre til at brukeren trykker på denne knappen for å oppnå noe annet enn det knappen gjør. En måte å unngå slike feil på er å

være konsekvent i utformingen av brukergrensesnittet. Dette kan blant annet innebære å bruke designstandarder brukt i andre programmer.

Overlappende kommandosekvenser kan bidra til fangstfeil[1]

I sin artikkel beskriver Norman en fangstfeil som noe som oppstår når to handlinger, A og B, har en overlappende sekvens av kommandoer. Slike kommandosekvenser kan f.eks. være å måtte trykke på flere knapper etter hverandre. Dersom handling A blir gjennomført vesentlig oftere en B, kan brukeren av vane gjøre handling A når han eller hun vil gjennomføre handling B.

Som løsning for å unngå slike fangstfeil, foreslår Norman å begrense antall overlappende kommandosekvenser så mye som mulig. I tillegg bør programmet forsøke å fange opp slike feil når de oppstår. Dette vil kreve at systemet vet hvilke intensjoner brukeren har på forhånd.

Påminnelser kan redusere aktiveringsfeil

Aktiveringsfeil oppstår, ifølge Norman, når det blir gjennomført en upassende handling eller når brukeren ønsker å gjennomføre en passende handling, men denne ikke blir gjort. En upassende handling kan for eksempel oppstå i forbindelse med fangstfeil. Man kan la være å gjøre en passende handling dersom det kommer en forstyrrelse før brukeren rekker å gjennomføre handlingen. En løsning på dette er å vise brukeren at det er mulig å gjennomføre handlingen som var planlagt etter at hendelsen som forstyrret er over.

Lag programmet slik at det ikke lar seg påvirke av feil

Ifølge Norman er det uunngåelig at brukerne gjør feil. Derfor er det viktig å gjøre konsekvensene av feilbruk så små som mulig. En mulig løsning er å la handlinger kunne reverseres, slik at feil kan rettes. Der det ikke er mulig å la brukeren reversere sine feil, bør det være vanskelig å utføre handlingen.

2.1.2 Gestaltprinsipper

Gestaltprinsippene er grunnregler ofte brukt i forbindelse med interaksjonsdesign. Prinsippene baseres på arbeid innen gestaltpsykologien, hvor fokuset ligger på hvordan mennesker tolker sine sanseinntrykk (Svartdal, 2019). Slik kunnskap kan brukes til påvirke brukerens opplevelse av applikasjonen.

Ifølge Johnson (2013) er de viktigste gestaltprinsippene: nærhet, likhet, kontinuitet, lukkethet, symmetri, forgrunn/bakgrunn og felles skjebne.

Nærhet

Prinsippet om nærhet innebærer at objekter som ligger nært hverandre relativt til andre objekter oppfattes ofte som en gruppe. Objekter som logisk hører sammen bør derfor grupperes ved å plassere disse nært hverandre.

Likhet

Et annet prinsipp som kan brukes til å vise gruppetilhørighet for objekter, er likhet. Likhetsprinsippet sier at objekter som ser like ut ofte antas å tilhøre samme gruppe.

[1] Engelsk: Capture Error

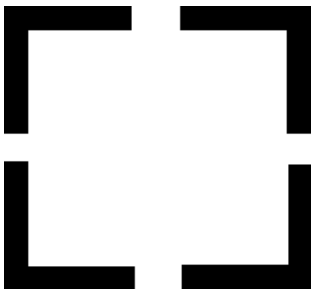
Kontinuitet

Kontinuitetsprinsippet innebærer at objekter heller oppfattes som kontinuerlige former enn frakoblede segmenter. For eksempel blir de røde feltene i Figur 2-1 ofte tolket som én rød linje i stedet for to røde linjer.

Figur 2-1 Eksempel på kontinuitetsprinsippet

Lukkethet

Lukkethetsprinsippet går ut på at åpne figurer ofte tolkes som lukkede (Johnson, 2013). Figur 2-2 illustrere dette. Samlet oppfattes de fire sorte hjørnene ofte som en firkant.



Figur 2-2 Illustrasjon av lukkethet

Symmetri

Symmetriprinsippet går ut på at hjernen automatisk tolker former slik at de fremstår som minst mulig kompleks og mest mulig symmetrisk. Dette prinsippet gjør det mulig å tegne todimensjonale bilder som fremstår som tredimensjonale (Johnson, 2013).

Forgrunn/bakgrunn

Prinsippet om forgrunn/bakgrunn forteller at det en ser blir delt opp i forgrunn og bakgrunn. Forgrunn er det en fokuserer på. Alt annet er bakgrunn. Ved å legge noe nytt over det brukeren fokuserer på kan man bruke dette prinsippet til å lede brukerens oppmerksomhet.

Felles skjebne

Prinsippet om felles skjebne sier at objekter som beveger seg sammen oppfattes som en gruppe og hører sammen.

2.2 Objektorientert programmering

I dette prosjektet blir objektorientert programmering brukt til å utvikle backendapplikasjonen. Hovedprinsippet bak objektorientert programmering er å dele kode inn i klasser. Hver klasse består så av felt som holder på data og metoder for å

behandle dataen. Et objekt defineres av en klasse. Det vil si at et hvert objekt inneholder de feltene og metodene som en klasse inneholder. Forskjellen mellom en klasse og et objekt er at en klasse kun kan beskrive hvilken data et objekt kan inneholde, mens et objekt faktisk inneholder data (Trætteberg, 2020).

2.2.1 Kohesjon

Ifølge Barnes og Kölling (2016, p. 259) er kohesjon et mål på «antallet og mangfoldet av oppgaver en enhet av en applikasjon er ansvarlig for». En enhet kan for eksempel være en klasse, modul eller metode. I en applikasjon med høy kohesjon har hver enhet én bestemt oppgave. Høy kohesjon gjør det mulig å gjenbruke kode.

2.2.2 Kobling

Innen objektorientert programmering er kobling et mål på hvor avhengige klasser er av hverandre. Jo mindre avhengige klasser er av hverandre, desto løsere er koblingen mellom dem. Om en har en fast kobling mellom klasser vil en endring i en klasse med stor sannsynlighet føre til endringer i de andre klassene. En løs kobling vil innebære at man i større grad kan gjøre endringer i en klasse uten at det får innvirkning på de andre klassene. For å oppnå god kodedesign bør en derfor forsøke å ha en løs kobling mellom klassene (Barnes & Kölling, 2016).

2.3 Turneringsplanlegging

Et av kravene fra oppdragsgiver er at applikasjonen skal sette opp partier for spillerne (se vedlegg 1). Det finnes en rekke systemer for å løse dette problemet. I noen av systemene blir partiene er bestemt før turneringsstart, mens de i andre systemer blir bestemt underveis i turneringen.

2.3.1 Round-Robin

Round-Robin blir ofte kalt for «alle møter alle». Dette turneringssystemet bestemmer hvilke konkurrenter som skal møte hverandre og når de skal møtes før turneringen begynner (Lujan, 2007). I sjakksammenheng, hvor farge også har betydning kan round-robin være i enkelte spilleres favør. Dette kan være i form av sterke spillere spiller hvit mot svakere spillere. For å omgå dette kan det bli lagt til to runder, slik at alle spillere møter hverandre med både hvit og sort.

2.3.2 Monrad

I motsetning til Round-Robinturneringer blir rundene i Monradturneringer ikke bestemt før turneringsstart. Monrad kan brukes når det er et stort antall deltakere i turneringen. Grunnen til dette er at Monrad ikke krever at alle skal spille mot hverandre. Dette turneringssystemet bestemmer første runden med uttrekking og neste runde blir bestemt basert på kumulative resultater i turneringen. Hovedsakelig skal spillere som møter hverandre ha så lik poengsum som mulig. To spillere skal ikke spille mot hverandre mer

enn en gang. En mer detaljert beskrivelse av Monrad finnes på sjakkforbundets nettside (Norges Sjakkforbund, u.d.).

2.4 Kommunikasjonsprotokoller

Det ferdige produktet består av både en backend- og en frontendapplikasjon. Dersom en skal få disse applikasjonene til å kommunisere med hverandre er en nødt til å velge en felles kommunikasjonsprotokoll.

2.4.1 HTTP

HTTP er en applikasjonslagsprotokoll som ofte er brukt når applikasjonene er delt opp i klienter og tjenere (Kurose & Ross, 2016). Klienten sender HTTP-forespørsler til serveren og får tilbake HTTP-responser. HTTP er en tilstandsløs protokoll. Dette innebærer at serveren ikke holder på informasjon om klientene. Som transportprotokoll bruker HTTP TCP. Dette medfører at mottaker er garantert å få all data som sendes.

Forespørsel

En HTTP-forespørsel består blant annet av felt for metode og URL, samt header-linjer og en «entity body».

Av metoder finner man blant annet: GET, POST, PUT, PATCH og DELETE. GET-metoden er brukt når klienten ønsker å få noe data tilbake. Når data fra et skjema skal sendes og klienten forventer å få data tilbake, er det POST-metoden som typisk blir brukt. PUT-metoden blir ofte brukt når data skal sendes til server og klienten ikke forventer data tilbake. For å gjøre endringer i data brukes PATCH. Dersom data skal slettes på serveren, brukes DELETE-metoden.

Header-linjene inneholder tilleggsinformasjon til forespørselen. Dette kan blant annet være hvilket språk klienten aksepterer, hvilken nettleser forespørselen ble sendt fra eller hvorvidt vedvarende tilkobling er ønsket.

Dersom noe data skal sendes til serveren plasseres denne ofte i «Entity body»-en. Et alternativ i noen tilfeller er å sende data i URIen.

Respons

Statuskode og frase, header-linjer og «entity body» er noe av det en HTTP-respons inneholder.

Statuskode og frase gir en kort tilbakemelding på forespørselen som ble sendt. For eksempel blir statuskode «200» og statusfrasen «OK» brukt til å fortelle at ønsket data er funnet og blir sendt.

Informasjon om når data-en som blir sendt sist ble endret eller hvilken type server data-en ble sendt fra, er eksempler på tilleggsinformasjon som kan bli lagt i header-linjene.

«Entity body»-en kan inneholde data som skal sendes til klienten.

2.4.2 HTTPS

HTTPS er en kommunikasjonsprotokoll hvor man i tillegg til å bruke HTTP, krypterer kommunikasjonen ved bruk av protokollene Transport Layer Security eller Secure Socket

Layer. Krypteringen gjør det tilnærmet umulig for uvedkommende å lytte på eller endre informasjonen som kommuniseres (Bartnes, 2017).

2.4.3 Vedvarende og ikke-vedvarende forbindelse

Dersom TCP blir brukt som transportslagsprotokoll kan man velge mellom å bruke vedvarende eller ikke-vedvarende forbindelser. Ved vedvarende forbindelser beholdes samme TCP-forbindelser til flere forespørsler og responser mellom maskiner. Når ikke-vedvarende forbindelser er brukt opprettes en ny TCP forbindelse for hvert forespørsel/respons-par (Kurose & Ross, 2016).

2.4.4 Websocket

Websocket er en protokoll som tillater en toveis kommunikasjon over samme TCP-kobling (Fette & Melnikov, 2011). Denne protokollen gjør det mulig å unngå polling, der klienten spør jevnlig om ny informasjon fra server. I stedet kan serveren sende informasjon til klienten ved behov.

2.4.5 STOMP

STOMP er en protokoll som kan kjøres over Websocket-protokollen. I hovedsak tilbyr STOMP-protokollen to funksjoner: send og abonner (subscribe). Ved å bruke send-funksjonen kan klienter sende en melding til bestemte destinasjoner. Destinasjonene defineres med en uniform ressursidentifikator (URI). De klientene som vil motta disse meldingene må bruke abonner-funksjonen for å fortelle at de vil ha meldinger sendt til disse destinasjonene (Stomp, u.d.).

2.5 Datastrukturer

Når data skal sendes mellom frontend- og backendapplikasjonen må den være strukturert på en måte som gjør at mottaker vet hvordan dataen skal tolkes. Til det brukes noen kjente datastrukturer.

2.5.1 JSON

JSON er format brukt ved datautveksling. JSON definerer to datastrukturer: en JSON-tabell og et JSON-objekt. JSON-objektet består av nøkkel/verdi-par. En JSON-tabell kan bestå av en eller flere JSON-objekter eller verdier (Bray, 2017).

2.5.2 JSON Web Token

JSON web token (JWT) er en standard blant annet brukt til identifisering av et datasystems brukere. En JWT har tre hoveddeler: en header, en del med identifiserende informasjon (payload) og en signatur. Headeren består av metainformasjon slik som hvilken tokentype som brukes og hvilken algoritme som er brukt til å signere JWT-en.

Både headeren og payloaden er representert av Base64URL-kodede JSON-Objekter. For å kunne bestemme om JWT-en er autentisk eller ikke, signeres den. Signaturen er en hashing av header, payload og en hemmelig nøkkel. JWT sendes som en tekststreng med header, payload og signatur separert med et punktum (Jones, et al., 2015).

2.6 Datamodellering

I dette prosjektet blir det benyttet en database for å holde oversikt over turneringer, spillere og spill (se avsnitt 4.1). En slik oversikt er nødvendig blant annet for å bestemme tilhørighet og unngå å sette opp parti med spillere fra ulike turneringer. For å strukturere denne dataen på en god måte, er der noe teori som bør følges.

2.6.1 Funksjonell avhengighet

I databaseteori betyr en funksjonell avhengighet fra en attributtmengde, A, til en attributtmengde, B, at for hver verdi av A hører det til én og bare én verdi av B (Kristoffersen, 2016).

2.6.2 Partiell avhengighet

Dersom en attributtmengde som ikke er en delmengde av en kandidatnøkkel er funksjonelt avhengig av en ekte delmengde av en kandidatnøkkel, har man en partiell avhengighet (Kristoffersen, 2016).

2.6.3 Transitive avhengighet

Dersom det finnes en funksjonell avhengighet fra en attributtmengde, B, til en attributtmengde, C, og hverken B eller C er med i en kandidatnøkkel, har man en transitiv avhengighet (Kristoffersen, 2016).

2.6.4 Normalisering

Normalisering av en database innebærer å redusere mengden redundans og uregelmessigheter ved å omorganisere dataen i databasen. Det finnes ulike grader av normalisering, kalt normalformer. En økning i normaliseringsgrad innebærer en reduksjon av redundans og uregelmessigheter. I følge Sumathi og Esakkirajan (2007, p. 297) er den tredje normalformen generelt regnet som god nok. Videre følger en beskrivelse av de tre første normalformene fra Kristoffersen (2016).

Første normalform

Dersom alle attributter i en entitetstype[2] kun inneholder verdier som ikke kan bli oppdelt (atomiske verdier), er entitetstypen på første normalform. Dette vil si at et attributt ikke kan inneholde lister dersom entitetstypen skal være på første normalform.

[2] Begrepene tabell og entitetstype brukes ofte om hverandre. Blant annet bruker Bostrøm (1999) og Ponniah (2003) entitetstype der Kristoffersen (2016) bruker tabell. I denne besvarelsen vil begrepet entitetstype bli brukt.

Andre normalform

Dersom en entitetstype er på første normalform og ikke inneholder en partiell avhengighet, er den på andre normalform.

Tredje normalform

En entitetstype på andre normalform og som ikke inneholder transitive avhengigheter, er på tredje normalform.

2.7 Scrum

Scrum er et rammeverk brukt til å organisere arbeid bl.a. innen programvareutvikling. I Scrum har man ulike roller, hendelser og artefakter (Sutherland & Schwaber, 2017).

En av de mest grunnleggende idéene i Scrum er en sprint. En sprint er en arbeidsperiode på opptil en måned, hvor en forhåndsbestemt mengde arbeid gjøres. Det arbeidet som skal gjøres i forbindelse med utviklingen av et produkt fordeles i sprinter.

Produktkø og sprintkø er artefakter definert i Scrum. Produktkøen er en liste over alle oppgaver som skal gjøres for å lage produktet, mens sprintkøen er en liste over de oppgavene som skal gjøres i en bestemt sprint. Oppgavene i sprintkøen blir hentet fra produktkøen.

Scrum Master, produkteier og utviklingsteam er roller i Scrum. Scrum master har ansvaret for at prinsippene i Scrum blir fulgt, utviklingsteamet er de som utvikler prosjektresultatet, mens produkteier er ansvarlig for å styre produktkøen og gjøre verdien av arbeidet til utviklingsteamet høyest mulig. Utviklingsteamet er selvorganiserende og styrer sitt eget arbeid.

Sprintplanlegging, daglig scrum, sprintgjennomgang og sprinttilbakeblikk er alle hendelser som Scrum definerer. I sprintplanleggingen går scrumteamet sammen for å bestemme hva som skal utvikles i den kommende sprinten. Før hver arbeidsdag holdes det et 15-minutters langt møte kalt daglig scrum. I løpet av en daglig scrum går utviklingsteamet igjennom det arbeidet hvert enkelt teammedlem gjorde dagen før møtet og forklarer hva de vil gjøre den dagen møtet holdes. Ved sprintgjennomgang går man igjennom det som har blitt gjort i en sprint og vurderer om det skal gjøres noen endringer i produktkøen. Sprinttilbakeblikk brukes til å vurdere hva som gikk bra i foregående sprint og hvordan scrumteamet kan arbeide bedre.

2.8 Sikkerhet

Dersom en ikke har tatt hensyn til sikkerheten i applikasjonen, risikerer man at brukere ikke føler det er trygt å ta den i bruk. Noen sikkerhetsteorier har derfor blitt anvendt i forbindelse med utviklingen av applikasjonen.

2.8.1 XSS – Cross server scripting

XSS er en form for angrep mot webapplikasjoner der en bruker får til å sende skadelig skriptkode til andre brukere. Nettlesere klarer ikke å skille mellom skadelige skript og ikke skadelige skript siden begge har samme opprinnelse. Med slike angrep kan personer få tilgang til sensitiv informasjon som er lagret i nettleseren, slik som JSON Web Tokens

eller cookies (OWASP, u.d.(a)). Dette kan skje alle plasser der en webapplikasjons bruker kan gi inndata som ikke valideres og der inndataen ikke blir kodet slik at den senere vil bli tolket som tekst og ikke som skriptkode. For eksempel, dersom karakteren > kodes som &gt vil nettlesere tolke karakteren som en del av en tekst og ikke som slutten på en tag.

2.8.2 SQL – injeksjon

Som ved XSS (avsnitt 2.8.1), er SQL-injeksjoner også et problem som oppstår ved manglende validering og sikring av brukerinput. SQL-injeksjoner brukes til å hente sensitivt innhold fra databasetabeller som brukere vanligvis ikke har tilgang til. SQL-injeksjoner kan også ødelegge databasen ved å sende spørringer som sletter deler av eller hele tabeller. Dette gjøres ved å sende deler av en SQL-spørring som input til applikasjonen. Dersom input brukes direkte i en SQL-spørring kan input gitt av brukeren dermed påvirke hva spørringen gjør (OWASP, u.d.).

3 MATERIALER OG METODE

I denne delen av rapporten beskrives framgangsmåten for hvordan prosjektoppgaven ble løst. Dette innebærer blant annet hvordan prosjektet har vært organisert og hvilken dokumentasjon som har blitt utarbeidet. I tillegg beskrives de tekniske sidene ved prosjektet. Dette innebærer blant annet hvilke programmeringsspråk og rammeverk som er brukt, og hvordan utviklingsmiljø og verktøy har vært satt opp.

3.1 Metode

Her vil prosjektets fremgangsmåte bli presentert. Dette innebærer blant annet hvordan prosjektet har vært organisert, hvilken utviklingsmetodikk som har vært brukt og hvordan samhandlingen med oppdragsgiver har vært.

3.1.1 Prosjektorganisering

Scrum har blitt brukt som utviklingsrammeverk (se avsnitt 2.7). En i gruppen har vært prosjekteier og har hatt ansvar for produktkøen.

Et gruppemedlem ble utpekt som Scrum-master, med ansvar for å se til at prinsippene i Scrum ble fulgt. I tillegg har alle vært deltakere i utviklingsteamet.

Utenom de vanlige scrumrollene har et gruppemedlem vært sekretær. Denne rollen har hatt ansvaret for å skrive møtereferat.

3.1.2 Utviklingsmetodikk

Det ble bestemt av gruppen at prosjektet skulle utvikles med smidige metoder i form av Scrum (avsnitt 2.7). Dette innebærer at utviklingen har både vært iterativ og inkrementell. Utviklingen var iterativ ettersom arbeidet ble gjort i sprinter og inkrementell fordi applikasjonen etter hver sprint ble utvidet til en ny versjon. Varigheten til en sprint ble satt til to uker.

Daglig scrum har blitt gjennomført hver arbeidsdag. I perioder hvor man har arbeidet hjemmefra har rapportering av eget arbeid stort sett vært gjort skriftlig.

Sprintplanlegging, sprintgjennomgang og sprinttilbakeblikk har blitt gjennomført i forbindelse med overgangen fra en sprint til en annen.

3.1.3 Prosjektplanlegging

Utarbeidelse av prosjektplanen ble gjort i forbindelse med skrivingen av forprosjektrapporten (vedlegg 5). Det bla da identifisert en rekke hovedaktiviteter for prosjektet. Prosjektplanen startet med informasjonsinnhenting om teknologier og teorier som kunne brukes i programvareutviklingen. Deretter ble det planlagt at prosjektet skulle konseptualiseres. Som resultat av konseptualiseringen, skulle man sitte igjen med kravspesifikasjon, designskisser, trådrammer og UML-diagrammer.

Selve programvareutviklingen ble delt opp i tre deler: databasedesign, utvikling av frontendapplikasjonen og utvikling av backendapplikasjonen. Design av databasen ble

planlagt å skulle gjennomføres først, mens utviklingen av frontend- og backendapplikasjonene skulle utvikles samtidig. Det ble også planlagt å skrive rapporten parallelt med programvareutviklingen.

Ettersom at Scrum ble brukt i utviklingen ble det ikke planlagt i detalj hva som skulle utvikles når. Dette ble bestemt før hver sprint.

3.1.4 Prosedyrer for godkjenning av kode

For at alle skulle få med seg endringer i koden, og finne feil, har GitHub sin «Branch protection» blitt benyttet. Det ble bestemt at gruppen sin develop-grein skulle være beskyttet mot merging av pull requests før minst en annen på gruppen hadde lest gjennom endringene og godkjent koden. Om koden ble godkjent, men det blir lastet opp noe mer før merge, ble godkjenningene kastet og måtte gjøres på nytt. Det var også lagt inn at Travis-CI sine tester måtte bestås.

3.1.5 Samhandling med oppdragsgiver

For å holde oppdragsgiver oppdatert på utviklingen i prosjektet ble det avholdt møter på sjakklubben. Noen møter ble avlyst og da ble oppdatering sendt pr epost. Midt i prosjektet ble det uaktuelt med fysiske møter på sjakklubben på grunn av pandemien Covid-19. Pandemien gjorde at regjeringen forbydde all organisert kultur, idrettsarrangement og anbefalte befolkningen om å redusere sosial omgang for å begrense smitten. Sjakklubben stengte lokalene og all dialog med oppdragsgiver ble flyttet til epost.

For at det skulle være enklere å vise progresjon, og at kontaktpersonen kunne følge med på utviklingen, ble han gitt tilgang til GitHub-repositorier og ble tilsendt en lenke til testserveren applikasjonen ligger på. Denne ble oppdatert hver eller annen hver uke, alt ettersom hvor mye funksjonalitet som hadde blitt oppdatert eller laget.

3.2 Tekniske sider ved prosjektet

I utviklingen av applikasjonen har en rekke programmeringsspråk og rammeverk blitt brukt. I tillegg har flere utviklingsmiljø og verktøy blitt brukt. Disse vil bli beskrevet i denne delen av rapporten. Det vil også forklares hvordan testoppsettet har blitt gjennomført.

3.2.1 Programmeringsspråk og rammeverk

Denne delen inneholder en kort beskrivelse av programmeringsspråkene og rammeverkene som har blitt brukt i utviklingen av applikasjonen. I tillegg forklares det hvordan vi har benyttet oss av disse.

Spring Boot

Spring Boot er et rammeverk for Java som har blitt brukt til å lage backendapplikasjonen. Spring boot er laget slik at det skal gå med minst mulig tid på å sette opp systemet for utvikleren. Dermed kan mer av tiden brukes på applikasjonsutvikling. Av de tjenester som Spring Boot tilbyr, finner man blant annet en

innebygd Tomcat-server (Spring, u.d.) og muligheten til å håndtere websocket- og REST-kommunikasjon. I tillegg gir Spring Boot tilgang på funksjoner for databasespørringer.

Vue.js

I dette prosjektet har Vue.js blitt brukt til å lage frontendapplikasjonen. Vue.js er et rammeverk skrevet i JavaScript som følger «Model-view-viewmodel»-rammeverket (MVVM). MVVM skiller utviklingen mellom brukergrensesnittet, forretningslogikken og presentasjonslogikken i applikasjonen. I praksis betyr dette at de ulike områdene kan utvikles mer uavhengig av hverandre. (Microsoft, 2012). Vue.js brukes til å utvikle view-laget i MVVM-rammeverket (Vue, u.d. (a)).

Å jobbe med rammeverk som Vue.js er mer effektivt sammenlignet med å jobbe med ren HTML, JavaScript og CSS/SASS. Det skyldes at man i stedet for å skrive alt fra grunnen av bruker JavaScript til å legge til og endre funksjonalitet.

Vuetify

Vuetify er et gratis open-source-rammeverk som brukes sammen med Vue. Vuetify består av ferdig stylet Vue.js komponenter som følger «Material Design»-standarden utviklet av Google (vuetify, n.d.).

MySQL Community Edition

Som databasehåndteringssystem har vi brukt MySQL Community Edition. MySQL Community Edition er bygd på åpen kildekode og er basert på Structured Query Language (Oracle, u.d. (a)).

Jest

For å teste frontendapplikasjonen har rammeverket Jest blitt brukt. Jest krever lite konfigurasjon og er et av de mest utbredte rammeverkene for enhetstesting i JavaScript (Jest.js, u.d.). Når Jest blir brukt sammen med Vue.js, og andre liknende JavaScript-bibliotek, blir hver komponent testet isolert og testene blir kjørt parallelt. Jest gir også muligheten til å teste nestete komponenter. Alle testene blir skrevet i JavaScript.

Cypress

Cypress er et testrammeverk som har blitt benyttet til å teste applikasjonsflyt. Med dette testrammeverket kjøres alle testene i en nettleser. Alt som kan kjøres i en nettleser kan bli testet med Cypress. Det finnes flere fordeler med Cypress, blant annet at man ikke trenger å kombinere flere rammeverk og verktøy for å kunne kjøre testene (Cypres, u.d.). Testene kjører også like fort som nettleseren klarer å laste inn sidene. En annen fordel er at når testene kjører, tar Cypress «bilder» slik at man kan se hva som har skjedd i hvert teststeg. Det blir også automatisk tatt bilder når testen feiler. Underveis blir en video som viser hele testen laget, og denne videoen kan senere brukes som en demonstrasjon av applikasjonsflyten. Cypresstester blir skrevet i JavaScript.

Vue router

Vue router brukes til å håndtere linking mellom de ulike filene i frontendapplikasjonen (Vue, u.d (b)). Dette gjør det mulig å lage en enkeltside applikasjon.

Axios

Axios er et bibliotek som har blitt brukt til å gjøre HTTP-kall fra klientsiden (Axios, u.d.).

Service worker

Service workers er JavaScript-kode som kjører i bakgrunnen på en egen tråd, uavhengig av nettstedet (Mozilla, u.d.). Dette gjør at scriptet ikke har tilgang til localStorage eller DOM-elementer. Service workers er mest brukt til push-varsler og til hente data fra cache når brukeren mangler internettforbindelse, geolokasjon eller liknende.

3.2.2 Utviklingsmiljø og verktøy

IntelliJ

IntelliJ er ett utviklingsmiljø utviklet av JetBrains. Dette utviklingsmiljøet har god innebygd støtte for de ulike programmeringsspråkene og rammeverkene som ble benyttet i oppgaven, samt tilkobling til database. Støtten for database, frontend og backend gjør at vi kan koble disse sammen når vi arbeider, uten bruk av flere verktøyer samtidig.

MySQL Workbench

Dette er en applikasjon med ett visuelt brukergrensesnitt for å designe, utvikle og administrere MySQL databaser. Applikasjonen gir mulighet for å sette opp databasetabeller og definere relasjoner mellom tabellene. En kan også skrive SQL kode og få laget en fil med koden som trengs for å kunne sette opp databasen i en server. Workbench inneholder også funksjoner for brukeradministrasjon, serverkonfigurasjon og migrering fra andre typer databaser til MySQL. Workbench er en gratis applikasjon (Oracle, u.d (b)).

NPM

For å håndtere diverse pakker og verktøy benyttet vi NPM. NPM er forkortelsen for «node package manager». Dette er et register for pakker benyttet i JavaScript (NPM, u.d.). Registeret gjør det raskt og enkelt for utviklere å bruke andre sine pakker/moduler i sine egne prosjekt. Jest og Cypress er eksempler på verktøy som NPM ble brukt til å skaffe.

Docker

Docker er en konteinerløsning brukt til virtualisering av programvare. Ved å sette prosjektet i en konteiner kan vi flytte denne konteineren til ulike maskiner. I denne konteineren finnes all programvare som applikasjoenn trenger for å kunne kjøre. Denne konteineren er isolert fra alle andre programmer på maskinen Docker kjøres på (Docker, u.d.).

Travis

Travis CI er et verktøy for kontinuerlig integrering. Dette verktøyet blir brukt til å teste programvaren. Disse testene blir kjørt før hver merge i GitHub, slik at vi vet at branch-ene kan slås sammen og at koden kan kjøres etter mergen. Dette gjør at vi kan teste hver del vi legger til i programvaren, og få vite om der er feil. Slik oppnås en bedre utviklingsflyt, hvor hver feil som oppdages er mindre sammenliknet med feil en hadde oppdaget om en kun testet mot slutten av prosjektet (Travis, u.d.).

EsLint

EsLint er en «plugin» som analyserer koden som blir skrevet og fremhever syntaksfeil (EsLint, u.d.). I EsLint kan man selv skrive regler for hva som skal regnes som riktig

syntaks. På den måten kan man tilpasse EsLint til avtalt kodestil. EsLint kommer også med forskjellige forhåndsdefinerte innstillinger. Denne har hjulpet til med å holde en konsekvent kodestil gjennom prosjektet.

3.2.3 Testoppsett

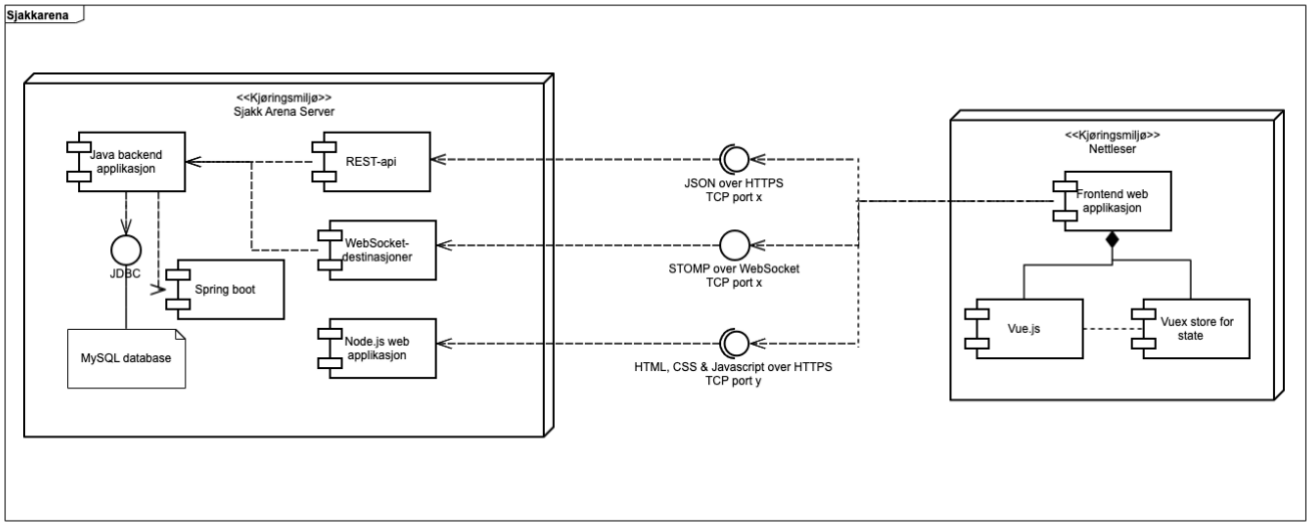
Server

For å vise applikasjonen til veiledere, oppdragsgiver og andre som kan gi tilbakemeldinger har det blitt satt opp en server. Serveren ble satt opp ved hjelp av NTNU Ålesund sitt «Auto Deploy» system. Der blir det satt opp en virtuell datamaskin med seks forskjellige konfigurasjoner. Gruppen valgte standard konfigurasjon som er en Ubuntu-server med versjon 18.04 LTS. Den har 1 CPU, 2 GB RAM og 40GB HDD. For å få både «frontend» og «backend» til å fungere måtte det installeres node, maven, OpenJDK og Docker.

Etter hvert som applikasjonen ble utviklet kom et behov om å kunne benytte HTTPS (avsnitt 2.4.2). Behovet kom under implementasjon av push-varsler, der en service worker (3.2.1) måtte benyttes. Av sikkerhetsmessige grunner kan disse kun benyttes i HTTPS-kontekst og under utvikling av applikasjon på «localhost». Etter noen forsøk på å få dette til ved den eksisterende serveren ble det bestemt at applikasjonen skulle flyttes til Heroku hvor det er mulig å sette opp HTTPS. De tilbyr også kredit til studenter som gjør det mulig å ha en gratis server.

4 RESULTATER

Sjakkturneringsapplikasjonen, Sjakkarena, vil bli kjørt både på en server og på klienter. Som deploymentdiagrammet i Figur 4-1 viser vil serveren ha en Java-backendapplikasjon og en Node.js-webapplikasjon. Spring Boot har blitt brukt til å utvikle backendapplikasjonen. I tillegg vil serveren ha en MySQL-database. Backendapplikasjonen tilbyr et REST-API og WebSocket -destinasjoner hvor klienter kan sende og motta meldinger. På klientene vil en frontendapplikasjon kjøre. Til å utvikle denne applikasjonen har rammeverket Vue.js blitt brukt.



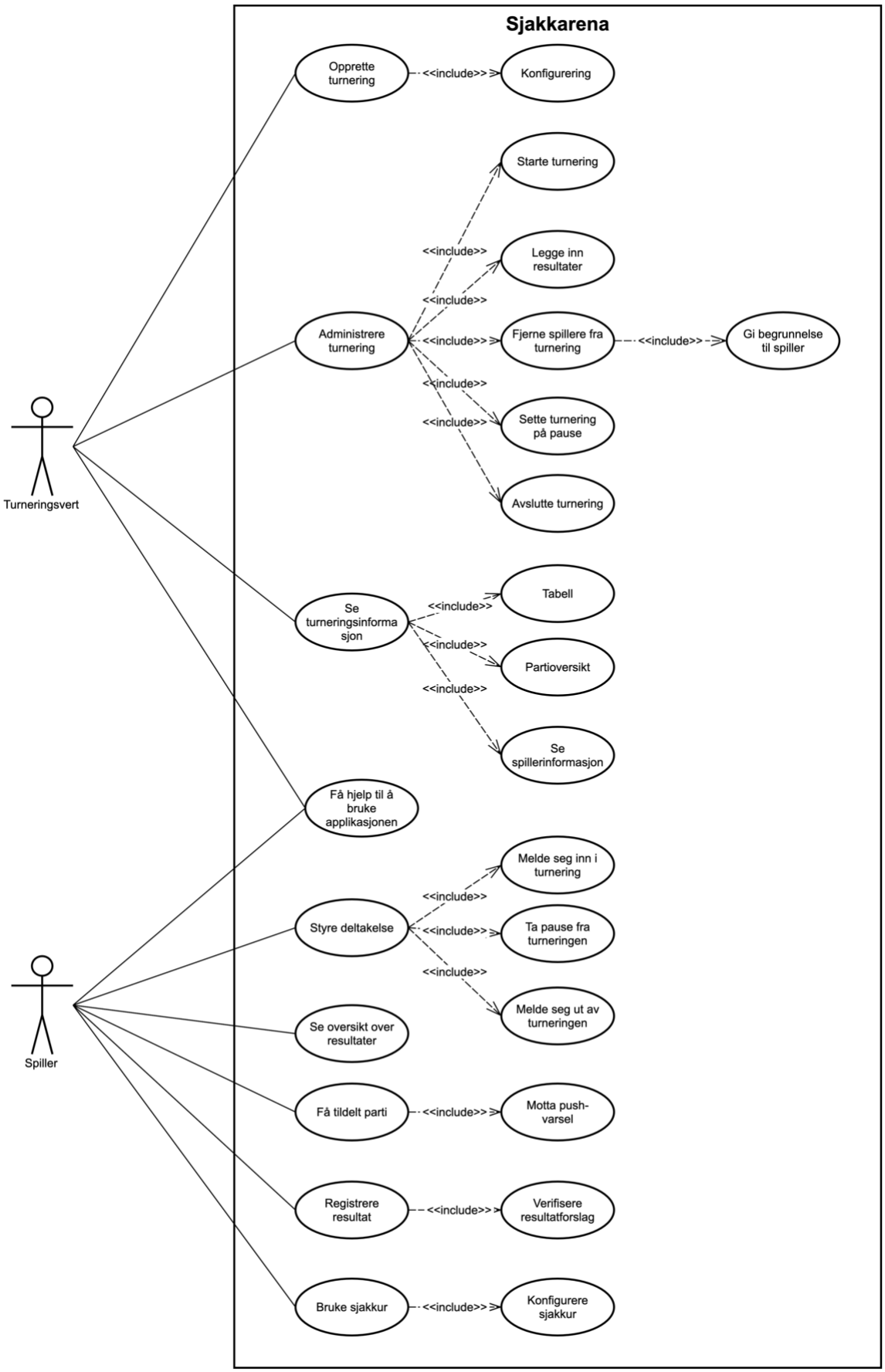
Figur 4-1 Deploymentdiagram for Sjakkarena

Figur 4-2 viser et bruksmønsterdiagram for applikasjonen. Dette diagrammet inneholder oversikt over den funksjonaliteten brukerne av applikasjonen kan benytte seg av. Det finnes to ulike brukertyper av applikasjonen: turneringsvert og spiller.

Som turneringsvert har man mulighet til å opprette og administrere en turnering. Dette innebærer blant annet å starte, slutte og sette turneringen på pause, samt legge til resultater på spilte parti. I tillegg har turneringsvert mulighet til å fjerne spillere og samtidig gi spillerne begrunnelse for hvorfor de har blitt fjernet.

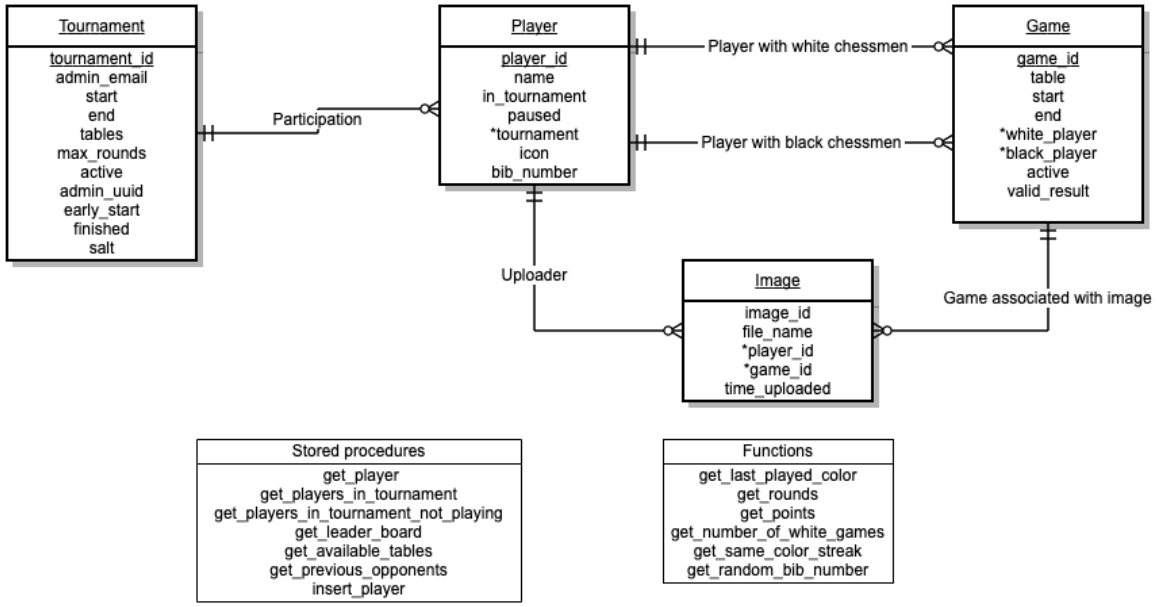
Spillere har mulighet til å melde seg på, ta pause fra og melde seg ut av turneringer. I tillegg vil spillere få tildelt parti som de kan legge til eller validere resultat for. Spillere har også muligheten til å se oversikt over tidligere spilte parti. Applikasjonen har også et sjakkur som spillere kan bruke for å styre tidsbruk når partiene spilles.

Både turneringsverter og spillere har tilgang på en hjelpeside som forklarer hvordan applikasjonen kan brukes.



Figur 4-2 Bruksmønsterdiagram for Sjakkarena

4.1 Datamodell



Figur 4-3 E/R-diagram over datamodell

Det er behov for å lagre en del data som genereres i forbindelse med bruk av applikasjonen. Slik data omfatter turneringer, spillere og parti og vil bli lagret i en database. For å bestemme hvordan denne dataen skal struktureres, har det blitt laget en datamodell. Figur 4-3 viser et E/R-diagram over prosjektets datamodell. Datamodellen består av fire entitetstyper og 13 prosedyrer og funksjoner.

4.1.1 Entitetstyper

Tournament

Entitetstypen Tournament representerer turneringer. Tournament har følgende attributter:

tournament_id	Hver turnering er identifisert med en unik id som blir lagret under tournament_id-attributtet. Det er denne id-en som spillere bruker for å fortelle hvilken turnering de vil melde seg på. Attributtet tournament_id er primærnøkkel for tournament-entiteten.
tournament_name	Den som oppretter en turnering må gi et navn til turneringen.
admin_email	Attributtet admin_email holder på e-postadressen til den som opprettet turnering. E-postadressen blir brukt til å sende e-post med en unik id som turneringsvert kan bruke til å logge seg inn med.
start	Start viser starttidspunkt for turneringen. Starttidspunktet er bestemt av turneringsvert før turneringsstart.
end	End viser sluttidspunktet for turneringen. For turneringsvert er det valgfritt å oppgi sluttidspunkt for turneringen.
tables	Tables holder på antall bord som er tilgjengelig i turneringen. Et bord tilsvarer et spillebrett.
max_rounds	Hvor mange runder en spiller maksimalt kan spille, føres opp under max_rounds-attributtet.
active	Active-attributtet settes til 1 dersom turnering spilles og 0 dersom turnering ikke spilles.
admin_uuid	Admin_uuid er en unik id som turneringsverter kan bruke til å logge seg på en turnering.
early_start	Turneringsverten har mulighet til å velge at turnering skal starte i det to spillere er påmeldt. Dersom turnering skal starte når to spillere er påmeldt settes early_start til 1, ellers settes den til 0.
finished	Holder på verdien 1 dersom turneringen er avsluttet, ellers 0.
Salt	Salt brukt i hashingen av admin_uuid

Attributtene til denne entitetstypen er atomiske. Dermed er entitetstypen på første normalform.

Den eneste attributtmengden som bestemmer alle attributter er en mengde kun bestående av tournament_id. Entitetstypen inneholder derfor ingen partielle avhengigheter. På den måten er entitetstypen på andre normalform.

Det er heller ingen transitive avhengigheter. Entitetstypen er derfor minimum på tredjenormalform etter definisjonene beskrevet i avsnitt 2.6.4. I følge Sumathi og Esakkirajan (2007, p. 297) er en entitetstype på tredje normalform ofte god nok. I forbindelse med utviklingen og testingen av applikasjonen har man sett at dette også stemmer i dette tilfellet.

Hver turnering kan ha flere eller ingen spillere.

Player

Entitetstypen Player representerer spillere i en turnering. En spiller kan kun være med i én turnering. Entitetstypen har følgende attributter:

player_id	Player_id er en unik id som identifiserer en spiller
name	Navnet på spilleren. Sammen med tournament-attributtet, identifiserer name-attributtet en spiller. Det vil si at for hver turnering kan kun en spiller ha et bestemt navn.
in_tournament	Dersom en spiller forlater eller blir kastet ut av turneringen, skal ikke spillerens statistikk forsvinne. Derfor brukes in_tournament-attributtet for å vise om en spiller er med i turneringen eller ikke. Verdien 1 betyr at spiller er i turneringen. Verdien 0 betyr at spiller ikke lenger er med i turneringen.
paused	Paused-attributtet blir brukt til å fortelle om en spiller har tatt pause eller ikke. Verdien 1 betyr at spiller har tatt pause. Verdien 0 betyr at spiller ikke har tatt pause.
tournament	Tournament er en fremmednøkkel som kobler samme Player-entitetstypen og Tournament-entitetstypen.
icon	I frontendapplikasjonen vises spillernavn sammen med et ikon. En beskrivelse av disse ikonene plasseres under icon-attributtet.
bib_number	Bib_number holder på spilleres startnummer.

Player-entitetstypen er på første normalform, da den kun vil bestå av atomiske verdier.

Det finnes to attributtmengder som er kandidatnøkler. Den ene mengden består av bare player_id-attributtet, mens den andre mengden består av både name- og tournamentattributtene. Ingen av de andre attributtene er funksjonelt avhengig av name- og tournamentattributtene hver for seg. Entitetstypen Player er derfor på andre normalform.

Det eksisterer heller ingen transitive avhengigheter da det ikke eksisterer funksjonelle avhengigheter mellom attributtmengder som ikke er med i kandidatnøklene. Etter definisjonene gitt i avsnitt 2.6.4 er entitetstypen derfor på tredje normalform.

En spiller kan spille flere eller ingen spill.

Game

Entitetstypen Game representerer spill som en spiller har spilt. Hvert spill har to spillere: en spiller med hvite brikker og en spiller med sorte brikker. Entitetstypen har følgende attributter:

game_id	Game-id er en unik id som identifiserer et spill.
table	Table holder på hvilket bord spillet spilles på.
start	Start viser tidspunktet spillet startet.
end	End viser tidspunktet spillet sluttet.
white_player	White_player er en fremmednøkkel som kobler sammen enitetstypen Game med entitetstypen Player. Dette attributtet viser hvilken spiller som har spilt med hvite brikker.
black_player	Black_player er en fremmednøkkel som kobler sammen entitetstypen Game med entitetstypen Player. Dette attributtet viser hvilken som spiller som har spilt med sorte brikker.
white_player_points	White_player_points viser hvor mange poeng hvit spiller oppnådde i spillet.
active	Active holder verdien 1 dersom spillet spilles og 0 dersom det ikke har begynt eller er ferdig.
Valid_result	Valid_result er en boolsk variabel hvor 1 signaliserer at partiet har et gyldig resultat enten fordi spillerne er enige eller fordi turneringsvert har bestemt resultatet. 0 signaliserer at partiet har et ugyldig resultat.

Game inneholder ingen atomiske-verdier. Det er heller ingen partielle avhengigheter, da eneste kandidatnøkkel er game_id. Entitetstypen har heller ikke transitive avhengigheter og er derfor på tredje normalform etter definisjonene gitt i avsnitt 2.6.4.

Image

I forbindelse med innmelding av resultat kan spillere legge ved et bilde som kan være til hjelp dersom turneringsvert skal avgjøre resultatet i partiet. Entitetstypen image holder på informasjon som kobler sammen et bilde med en spiller og et parti. Denne entitetstypen har følgende attributter:

image_id	En unik id som identifisere hvert bilde.
file_name	Navnet på bildefilen. Navnet må være unikt.
player_id	Fremmednøkkel som kobler sammen Player og Image. Player_id er id-en til spilleren som lastet opp bildet.
game_id	Fremmednøkkel som kobler samme Game og Image. Dette er id-en til partiet som bildet er koblet til.
time_uploaded	Tidspunktet bildet ble lastet opp til server.

Både image_id og file_name er de eneste kandidatnøklene i Image. Det finnes ingen funksjonelle avhengigheter mellom de andre attributtene. Derfor inneholder entitetstypen ingen transitive eller partielle avhengigheter og er derfor minimum på tredje normalform.

4.1.2 Prosedyrer og funksjoner

For at applikasjonen skal virke, trenger man å hente ut mer informasjon om en spiller enn den informasjonen Player holder på. Slik informasjon er blant annet antall poeng en bruker har fått i løpet av turneringen og hvor mange parti spilleren har spilt. I tillegg, for å sette opp nye spill, trenger man informasjon om hvor mange parti en spiller har spilt med hver farge, hvilken farge spilleren spilte sist med og hvor mange parti det er siden spilleren spilte med en annen farge. All denne informasjonen finnes i entitetstypen Game, men kan ikke hentes ut med én enkel spørring. Derfor har det blitt laget funksjoner og prosedyrer for å hente ut denne informasjonen. Skript for prosedyrer og funksjoner ligger i vedlegg 7.

For å hente ut en spiller, har get_player-prosedyren blitt laget. I tillegg til å hente ut den informasjonen som finnes i Player, henter den ut poeng, antall runder, sist spilte farge, antall parti siden spilleren spilte med en annen farge og antall parti med hvite brikker. For å finne informasjon som ikke ligger i Player, brukes diverse funksjoner.

Det har blitt laget ulike varianter av get_player-prosedyren. I stedet for å finne én spiller med get_player-prosedyren, kan man finne alle spillere som deltar i en turnering med en egen prosedyre. Det har også blitt laget en prosedyre for å finne spillere som ikke spiller et parti.

Prosedyrer for å finne tilgjengelige bord, finne tidligere motstandere til en spiller og for å sette inn en spiller i databasetabellen er også laget.

Når en spiller blir lagt inn i databasen, tildeles et tilfeldig valgt startnummer. En funksjon har blitt laget for å genere dette startnummer.

Figur 4-4 viser funksjonen for å finne en spillers poeng. Funksjonen tar inn id-en til spilleren og finner alle spill som spilleren har spilt. For hvert spill regner funksjonen ut poengene som spilleren fikk. Deretter summeres disse poengene.

```
-- ------------------------------------------------------
--  get_points
-- ------------------------------------------------------
DROP FUNCTION IF EXISTS sjakkarena.get_points;
DELIMITER //
CREATE FUNCTION sjakkarena.get_points(player_id INT(11))
  RETURNS DECIMAL(5, 1)
  READS SQL DATA
BEGIN
  DECLARE points DECIMAL(5, 1) DEFAULT 0;
  SET points = (SELECT SUM(CASE
                             WHEN white_player = player_id
                               THEN white_player_points
                             ELSE (1 - white_player_points)
                END)
                FROM `sjakkarena`.`game`
                WHERE (`white_player` = player_id OR `black_player` = player_id)
  );
  RETURN points;
END//
DELIMITER ;
```

Figur 4-4 Funksjon for å finne en spillers poeng

4.2 Frontendapplikasjon

Allerede på første møtet med oppdragsgiver ble det sagt at brukergrensesnittet blir viktig og at det burde bli lagt ned innsats i å få applikasjonen intuitiv og enkel å bruke. Kahoot

ble nevnt som en mulig inspirasjonskilde for å oppnå dette. Det var ønskelig at applikasjonen ikke skulle være distraherende. Fokuset til brukerne skulle være på å prestere godt i turneringen og minst mulig på noe annet.

Som nevnt har Vuejs blitt brukt til å utvikle frontendapplikasjonen. I Vuejs så bygger man sidene opp av komponenter. Eksempelvis kan et søkefelt eller en liste være en komponent. Disse komponentene blir lagt inn i en «view»-komponent som danner sidene. Komponenter som importerer en annen komponent, blir kalt for forelder og de importerte komponentene er barn.

```
<template>
  <div class="player-wrapper">
    <p class="player-name"> {{ playerName }} </p>
    <i :class="playerPiece"></i>
  </div>
</template>

<script>
export default {
  name: 'Player',
  props: {
    playerName: {
      type: String,
      required: true
    },
    playerPiece: {
      type: String,
      required: false
    }
  }
}
</script>

<style scoped>
  .player-name{
    font-size: 1.3em;
    font-weight: bold;
  }
```

Figur 4-5 Spillerkomponent

Figur 4-5 viser koden for spillerkomponenten som er benyttet i Figur 4-22. Hver komponent og «view» er bygget på samme prinsippet som denne spillerkomponenten. Hver komponent starter med <template>-taggen og slutter med </template>-taggen. All ønsket html – både standard html-elementer eller andre VueJs-komponenter– legges imellom template-taggene.

Under html-koden, imellom <script>- og </script>-taggene, ligger all tilhørende JavaScript kode. Her blir blant annet funksjoner til knapper definert, data blir hentet fra det sentrale tilstandslageret, egendefinerte komponenter blir importert, props blir definert etc. Props definerer hvilke data forelderkomponenten skal overføre til komponentene når de lastes inn. For å sikre mot feilbruk blir datatypen til variablene definert. Man kan også sette en standardverdi for variablene. Det kan og bli lagt til en validering som for eksempel sjekker om en dato eller klokkeslett har riktig format.

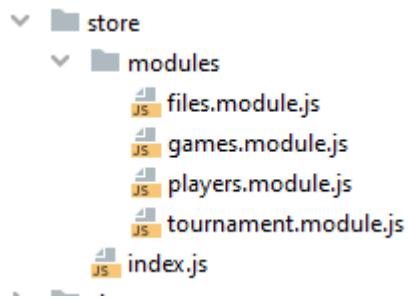
Alle komponentene avsluttes med styling imellom <style scoped>- og </style>-taggene. I stylingen av komponentene er det lurt å benytte seg av scoped. Dette er fordi om to komponenter inneholder den samme CSS-velgeren, så vil den ene overskrive stylingen til den andre. Uten scoped vil stylingen bli tolket som global. Dette skjer selv om komponentene ikke blir lastet samtidig eller i samme side.

«View»-komponentene blir importert inn i index.js som benytter seg av Vue-Router til å håndtere sidenavigasjonen i applikasjonen. I Figur 4-6 ser vi hvordan index.js er bygget opp. Index.js inneholder en konstant, routes, som er en tabell med objekter. Objektene består minimum av sti, navn og komponent (som refererer til «View»-komponenten som skal benyttes). Her ser vi at Home vil ha url «domenenavn.no/», NotFound vil bli brukt under alle stier som ikke finnes, og About finner vi under stien «/about». Objektene kan inneholde flere verdier, eksempelvis meta, der tittelen til siden blir definert. Sidene som ikke inneholder meta vil ha tittelen Sjakk Arena.

```
import ...

Vue.use(VueRouter)

const DEFAULT_TITLE = 'Sjakk Arena'

const routes = [
  {
    path: '/',
    name: 'home',
    component: Home
  },
  {
    path: '*',
    name: 'NotFound',
    meta: {
      title: '404 - Side ikke funnet'
    },
    component: () => import('../views/NotFound')
  },
  {
    path: '/about',
    name: 'about',
    meta: {
      title: 'Om sjakk arena'
    },
```

Figur 4-6 index.js

For å holde på tilstanden til applikasjonen har det blitt benyttet Vuex. Der blir blant annet turneringsobjektet, listen over partier og listen over spillere lagret. Dette sentrale tilstandslageret har tre typer funksjoner: «actions», «mutations» og «getters». I «actions» blir blant annet websocket-callbacks definert og nettverkskall håndtert, «mutations» blir brukt til å endre tilstanden og «getters» blir brukt til å hente data fra tilstandslageret. I de tidlige fasene av utviklingen lå actions, mutations og getters i egne filer. Dette ble etter hvert uoversiktlig og rotete. Da ble det bestemt at det skulle deles opp i moduler, der hver enkelt modul inneholder actions, mutations, getters og state. Det ble laget moduler for turnering, spiller, spill og filer. Modulene blir importert i index.js som ligger i store mappen. Det er denne filen som blir tilstandslageret.



Figur 4-7 sentral tilstand fil-indeling

```
import Vue from 'vue'
import Vuex from 'vuex'
import { deleteToken } from '../common/jwt.storage'
import { API_SERVICE } from '../common/api'

import * as games from '@/store/modules/games.module'
import * as players from '@/store/modules/players.module'
import * as tournament from '@/store/modules/tournament.module'
import * as files from '@/store/modules/files.module'

Vue.use(Vuex)

export default new Vuex.Store( options: {
  modules: {
    games,
    players,
    tournament,
    files
  },
  actions: {
    resetAllStatesAndToken: ({ dispatch : Dispatch }) => {
      deleteToken()
      API_SERVICE.clearHeader()
      dispatch('resetTournament')
      dispatch('resetPlayer')
      dispatch('resetGames')
    }
  }
```

Figur 4-8 store - index.js

Figur 4-7 viser hvordan mappestrukturen for den sentrale «storen» er. Og Figur 4-8 viser importering av modulene og en «actions»-metode som brukes til å tilbakestille all data fra tilstanden til de forskjellige modulene.

```
/**
 * Subscribes to active games. Receives active games from the server when the list of active games changes
 * @param commit
 */
subscribeToActiveGames: ({ commit }) => {
  let activeGamesCallback = function (res) {
    let activeGames = JSON.parse(res.body)
    commit('setActiveGames', activeGames)
  }
  let sub = {
    path: 'tournament/active-games',
    callback: activeGamesCallback
  }
  WEBSOCKET_SERVICE.connect(sub)
},
```

Figur 4-9 eksempel "action" fra games.module.js

Figur 4-9 er et eksempel på en metode definert under actions i games.module.js. Her settes det opp en «callback» metode som definerer hva som skal bli gjort når «WEBSOCKET_SERVICE» får en melding av backend fra «/tournament/active-games». Når websocket får en liste over spill blir listen gjort om fra en JSON-streng til et JavaScript-objekt, for så å bli sendt til en mutation som videre setter listen i tilstandslageret. All manipulering av «state» må skje gjennom mutations. Dette er en begrensing i Vuex for å forhindre feil.

4.2.1 Mixins

For å redusere kodeduplikasjon i frontendapplikasjonen blir det benyttet en funksjonalitet som er innebygget i VueJs som heter «mixins» (Vue, u.d. (c)). I «mixins» blir kode som flere filer trenger lagret. Dette kan være alt i fra props til diverse metoder. Det som er viktig å huske på når man bruker disse er at om det er felles «hook»-funksjoner[3] i «mixin» og komponentene, blir komponenten sin kalt til slutt. Dersom metoder definert i «mixin» har samme navn som egendefinerte metoder i komponenten, vil Vue ignorere «mixin» sin metode og benytte seg av komponenten sin metode.

[3] En hook-funksjon er funksjoner som kalles på ulike steg i initiliaseringen av komponenter (Vue, u.d. (d)). beforeCreate() og created() er eksempler på slike funksjoner.

```
export const playerMixin = {
  props: {
    tournamentName: {
      type: String,
      default: 'Sjakk-Arena turnering'
    },
    tournamentStart: {
      type: String,
      required: true
    },
    tournamentEnd: {
      type: String,
      required: false
    },
    playerName: {
      type: String,
      required: true
    },
    points: {
      type: Number,
      required: true
    }
  }
```

Figur 4-10 playerMixin.js

```
import { playerMixin } from '../../mixins/playerMixin'
```

Figur 4-11 importering av mixin

```
mixins: [
  leavePageWarningMixin,
  playerMixin
],
```

Figur 4-12 playerPlaying.vue mixin

```
<!-- Tournament name -->
<h1 class="bigInfo">{{ tournamentName }}</h1>

<!-- Basic user details -->
<h3 class="minorInfo">{{ playerName }}</h3>
<h3 class="mediumInfo">Poeng: {{ points }}</h3>

<v-divider></v-divider>
```

Figur 4-13 playerPlaying bruk av mixin

I Figur 4-10 ser vi et «mixin»-objekt som definerer felles props for to komponenter. Disse objektene blir laget i en vanlig JavaScript-fil. For å benytte disse må de importeres (se Figur 4-11) inn i komponentene og legges i «mixins» listen (se Figur 4-12). Etter at de har blitt lagt i denne listen er det rett fram å benytte seg av det som er definert. Figur

4-13 viser hvordan turneringsnavn, spillernavn og poeng blir hentet. Dette fungerer på samme måte som når props er definert rett i komponentene.

4.2.2 Språk

Applikasjonens språk er norsk. Med tanke på at denne applikasjonen skal brukes av folk med svært varierende kjennskaper til sjakk, er det derfor svært viktig at man ikke bruker sjakktermologier som de fleste ikke er kjent eller komfortabel med. Skulle det likevel være noen termologier som bruker ikke forstår, kan ordet finnes og bli forklart i applikasjonens ordliste i om-siden (avsnitt 4.2.13).

4.2.3 Responsivt design

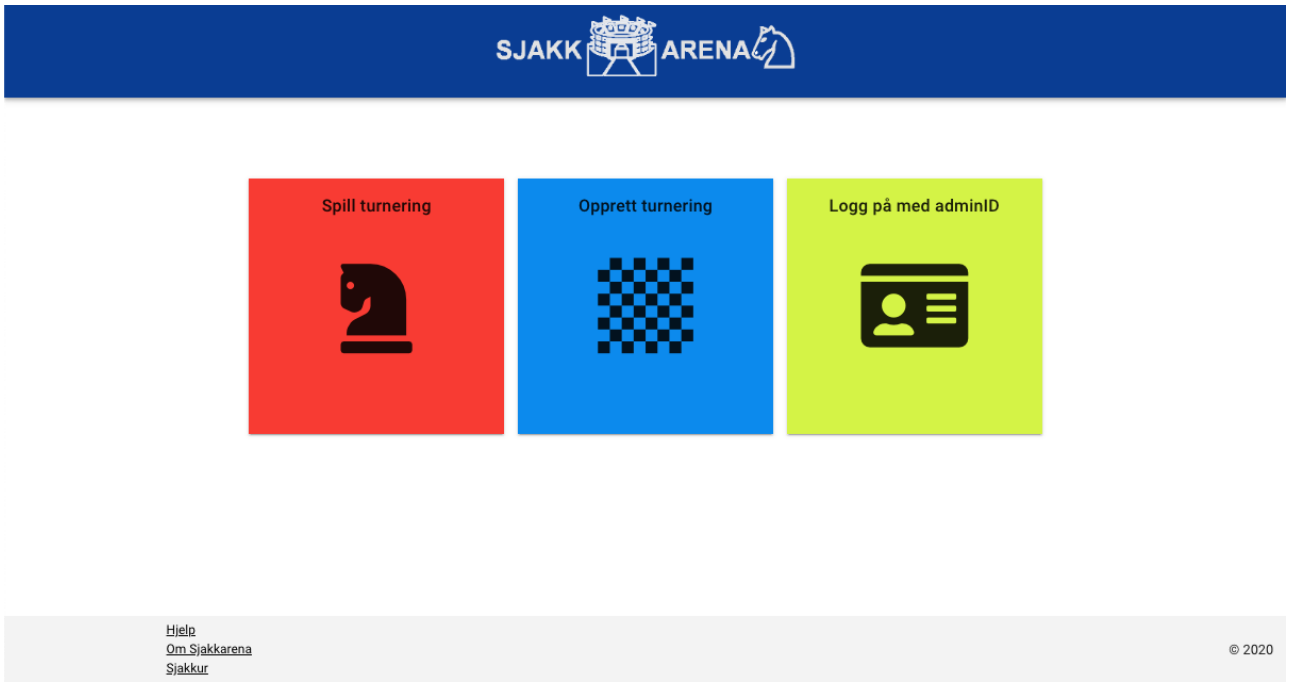
Sidene vi har laget har hatt mobile enheter som førsteprioritet ettersom dette er enheter majoriteten av brukere vil benytte seg av. Prioriteringen av mobile enheter skal likevel ikke gå på bekostning av funksjonalitet eller utseende på større enheter. Brukere skal ikke få følelsen av å bli straffet for å bruke andre enheter enn sin smarttelefon eller nettbrett.

4.2.4 Hovedside

Figur 4-14 viser hovedsiden i applikasjonen. Når man kommer til hovedsiden møter man tre knapper som representerer de ulike handlingene man kan utføre på siden. Dette er å bli med i en turnering, opprette en turnering eller logge inn med en adminID. Disse tre knappene er skilt fra hverandre med mellomrom slik at det blir tydelig at de er separate knapper. Fargene er valgt for å ikke gjøre knappene vanskelig å tyde for hverken fargeblinde eller de med normalt syn (se avsnitt 4.2.17 for mer informasjon). Knapene er likt utformet med unntak av ikonet, den forklarende teksten og fargen. I tillegg er knappene plassert nær hverandre for å vise at de alle representerer valgalternativer i det første valget brukeren skal foreta seg på siden. På den måten har gestaltprinsippene om nærhet og likhet (avsnitt 2.1.2) blitt brukt.

Øverst på siden finner man headeren som inneholder applikasjonens logo. Logoen i applikasjonen er sammensatt av tekst og to vektorer/bilder som er hentet fra pixbay. Pixbay er en side som inneholder gratis bilder og vektorer som kan brukes uten å be om tillatelse eller betale brukeren, også ved kommersiell bruk.

Nederst på siden, i footeren, finner vi lenker til sjakkur (avsnitt 4.2.12), om-siden (avsnitt 4.2.13) og hjelpe-siden (avsnitt 4.2.14).



Figur 4-14 Hovedside

For å unngå kodeduplikasjon har det blitt laget en MenuTile-komponent. Hver av knappene i hovedsiden er en instans av denne komponenten. For hver MenuTile-instans må det oppgis farge, ikon, tekst og lenke til siden applikasjonen skal rutes til når knappen trykkes. Figur 4-15 viser HTML-koden for MenuTile-komponenten. Som en kan se av figuren, er komponenten i hovedsak satt sammen av Vuetify-komponentene v-card og v-card-title. For å utnytte mest mulig av den funksjonaliteten som Vuetify tilbyr, blir styling-informasjon gitt via propsene til Vuetify-komponentene. Alternativt kunne en ha lagt stylingen imellom <style>-tagene. Ulempen med denne tilnærmingen er at en kan ende opp med styling som ikke følger samme standard som Vuetify-komponentene.

```
<template>
    <v-card :color="color"
            :to="link"
            tile
            width="20vw"
            height="20vw"
            max-height="300px"
            max-width="300px"
            min-height="230px"
            min-width="230px"
            class="ma-2"
    >
      <v-card-title class="justify-center text-center text-no-wrap">
        {{ tileText }}
      </v-card-title>
      <div id="tile-icon">
        <i :class="[commonClass, icon]"></i>
      </div>
    </v-card>
</template>
```

Figur 4-15 HTML for komponenten MenuTile

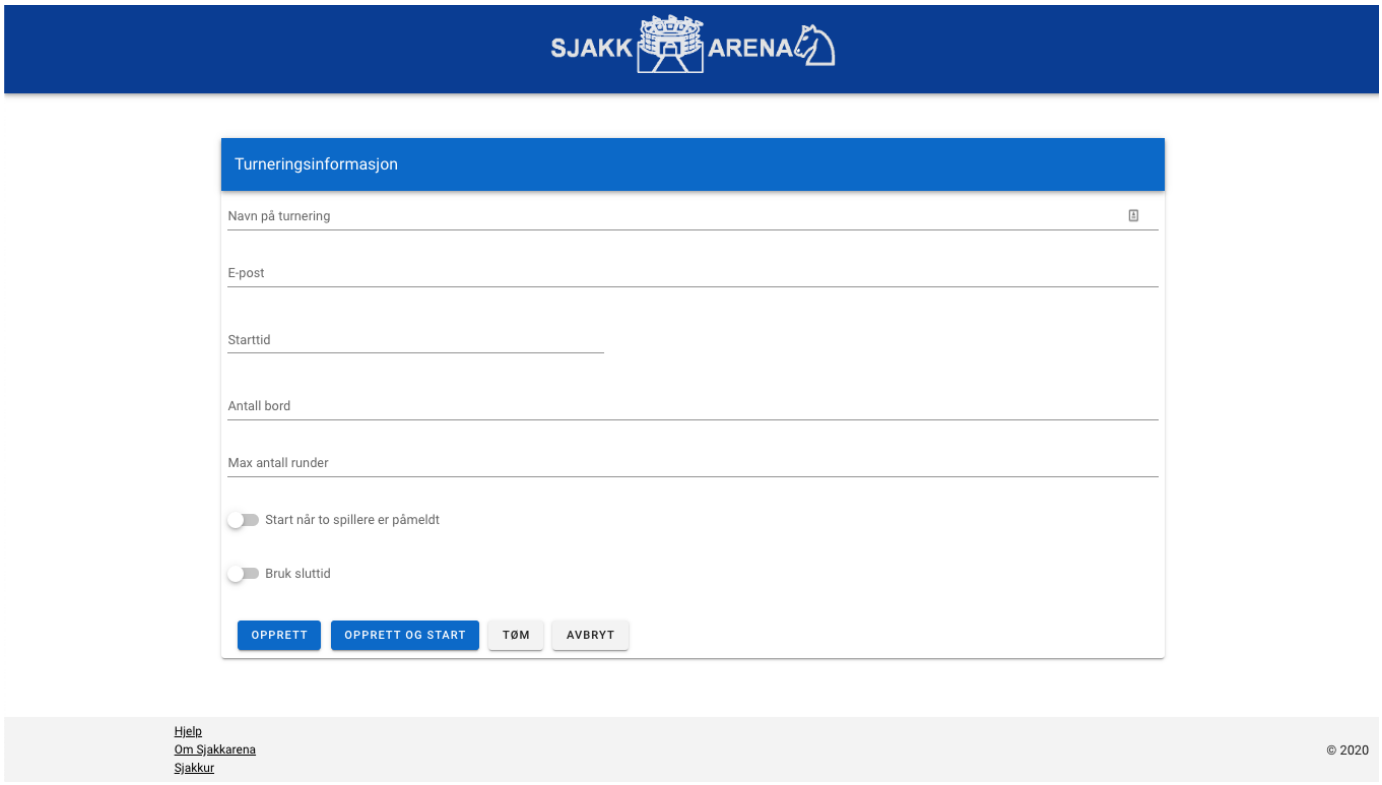
4.2.5 Side for å opprette turnering

Figur 4-16 viser siden hvor man kan opprette en turnering. Denne siden består i hovedsak av et skjema. I skjemaet skal man oppgi et navn for turneringen, sin egen e-

postadresse, starttid, antall tilgjengelige bord og maksimalt antall runder en spiller har lov til å spille. I tillegg kan man velge at turneringen skal starte automatisk når to spillere er påmeldt. Det er også mulig å sette en sluttid for turneringen. På siden finnes også fire knapper: én for å opprette turneringen, én for å opprette og starte turneringen, én for å tømme skjemaet og én for å avbryte prosessen med å opprette en turnering.

Som beskrevet i avsnitt 3.2.1 har vi brukt Vuetify til å utforme sidene. Vuetify er som nevnt bygd på «Material Design»-standarden. En av fordelen med å basere seg på «Material Design»-standarden er at flere prinsipper for interaksjonsdesign er fulgt. Blant annet kan man se i Figur 4-16 at etiketter for inntastingsfelt og verdien som er skrevet inn ligger nært hverandre. Samtidig er det større avstand mellom hvert inntastingsfelt enn det er mellom inntastingsfeltet og dets etikett. Dette gjør at man følger gestaltprinsippet om nærhet (avsnitt 2.1.2).

Alle inntastingsfeltene er omsluttet av en firkant for å vise at de tilhører samme skjema.



Figur 4-16 Side for å opprette turnering

For å unngå blant annet det Donald Norman kalte aktiveringsfeil (avsnitt 2.1.1) har det blitt lagt til validering av input fra brukeren. Figur 4-17 viser et eksempel på hvordan dette gjøres. Dersom input fra brukeren ikke er gyldig, lar det seg ikke gjøre å opprette turneringen. Et annet eksempel er starttid som ikke kan settes til et tidspunkt senere enn eller lik sluttid, og som ikke kan settes til et tidspunkt tidligere enn da turneringen opprettes. Dette gjøres ved å begrense valgene i tidsvelgeren.

Figur 4-17 Validering av e-postadresse

Validering gjøres ved å gi instansene av inputfeltkomponentene en liste med regler skrevet på et spesielt format. Figur 4-18 viser hvordan regelen for e-post-feltet i skjemaet er skrevet. Når skjemaet blir validert vil en sjekke om den inntastede e-postadressen samsvarer med en regex som beskriver ønsket format på e-postadresser. Dersom de ikke samsvarer vil inputfeltet lyse rødt og en forklarende tekst vil bli vist, slik som i Figur 4-17.

```
emailRules: [
  v => /^[A-ZÆØÅa-zæøå0-9._%+-]+@[A-ZÆØÅa-zæøå0-9.-]+\.[A-ZÆØÅa-zæøå]{2,6}$/.test(v) ||
    'Du må skrive inn en gyldig e-postadresse'
],
```

Figur 4-18 Valideringsregler for e-postaddresser

I de tilfeller hvor turneringsvert ønsker å sette en sluttid på turneringen vil det bli testet om sluttid er etter starttid. Til dette har det blitt laget en egen funksjon, kalt checkTime. Denne funksjonen er vist i Figur 4-19. Funksjonen sjekker først om starttid eller sluttid er udefinert, eller om sluttid ikke er samme dag som starttid. Siden datovelgeren for sluttid kun godtar datoer etter eller samme dato som når turneringen opprettes, vil den dermed anta at starttiden er før sluttiden og godkjenne tidene. Dersom alle tre betingelsene ikke er sanne, sjekkes det først om timer i sluttid er større enn timer i starttid og, dersom dette er sant, om minutter i sluttid er større enn minutter i starttid.

```
checkTime() {
  if (this.endTime === undefined || this.endDate !== this.currentDate || this.startTime === undefined) { return true }
  let startTimeH = this.startTime.toString().split( separator: ':')[0]
  let endTimeH = this.endTime.toString().split( separator: ':')[0]
  if (parseInt(startTimeH) < parseInt(endTimeH)) {
    // Error not displayed since the start time is smaller than the end time
    return true
  } else {
    // Display error if endtime is less than or equal to start time.
    let startTimeM = this.lastNumberInTime(this.startTime)
    let endTimeM = this.lastNumberInTime(this.endTime)
    return parseInt(startTimeM) < parseInt(endTimeM)
  }
},
```

Figur 4-19 Funksjon for å sjekke om starttid er før sluttid

For å følge Normans regel om at tilbakemeldinger fører til færre modusfeil (avsnitt 2.1.1), vises en innlastingssirkel når forespørsel om å opprette turnering er sendt til server og klienten ennå ikke har fått respons. Figur 4-20 viser en slik innlastingssirkel. Innlastingssirkelen er en Vuetify-komponent som benytter seg av lukkethetsprinsippet (avsnitt 2.1.2) til å lage sirkel som roterer. Innlastingssirkler blir også brukt andre plasser i applikasjonen. Blant annet når en bruker venter på å få tildelt et nytt parti.

Figur 4-20 Innlastingssirkel

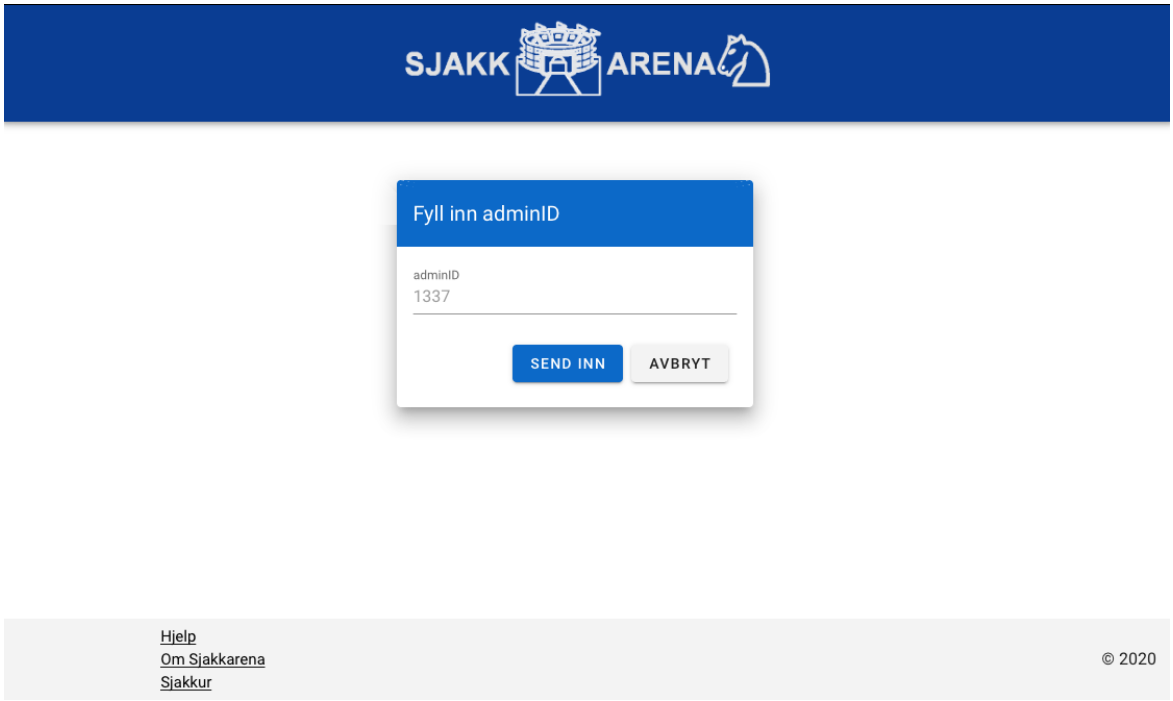
Farger på knapper blir brukt til å signalisere hva man ønsker brukeren skal trykke. Et gjennomgangstema for brukergrensesnittet er at de knappene som vil holde oppe en

brukers aktivitet i applikasjonen er farget blå. Knapper som vil redusere en brukers aktivitet har en svak gråfarge. Knappene for skjemaet i Figur 4-16 viser dette. Dersom brukeren trykker på «Opprett» og «Opprett og start» vil dette bidra til økt aktivitet i applikasjonen. Derfor er de farget blå. Knappene «Tøm» og «Avbryt» vil ikke bidra til økt aktivitet og har derfor en svak gråfarge. Det samme kan observeres i figurer nedenfor. Her har man benyttet seg av Donald Normans prinsipp om å være konsekvent, beskrevet i avsnitt 2.1.1. Bruker av systemet trenger kun å lete etter en blå knapp for å vite hvor han eller hun skal trykke for å komme videre i turneringen. Brukeren skal ikke villedes og knappenes plassering er forutsigbare og lette for brukeren å finne.

4.2.6 Side for å logge inn som turneringsvert

Etter at man har opprettet en turnering (avsnitt 4.2.5) får man tilsendt en unik ID, adminID, til e-postadressen som ble oppgitt i opprettingsskjemaet. Denne ID-en kan bli brukt til å få tilgang til lobbysiden for turneringer (avsnitt 4.2.7). Figur 4-21 viser siden hvor man må oppgi adminID-en. Innmeldingsskjemaet likner det som er brukt til å melde seg på turneringer (avsnitt 4.2.10) og har i likhet med dette to knapper: en knapp for å sende inn skjemaet og en knapp for å avbryte prosessen med å logge inn.

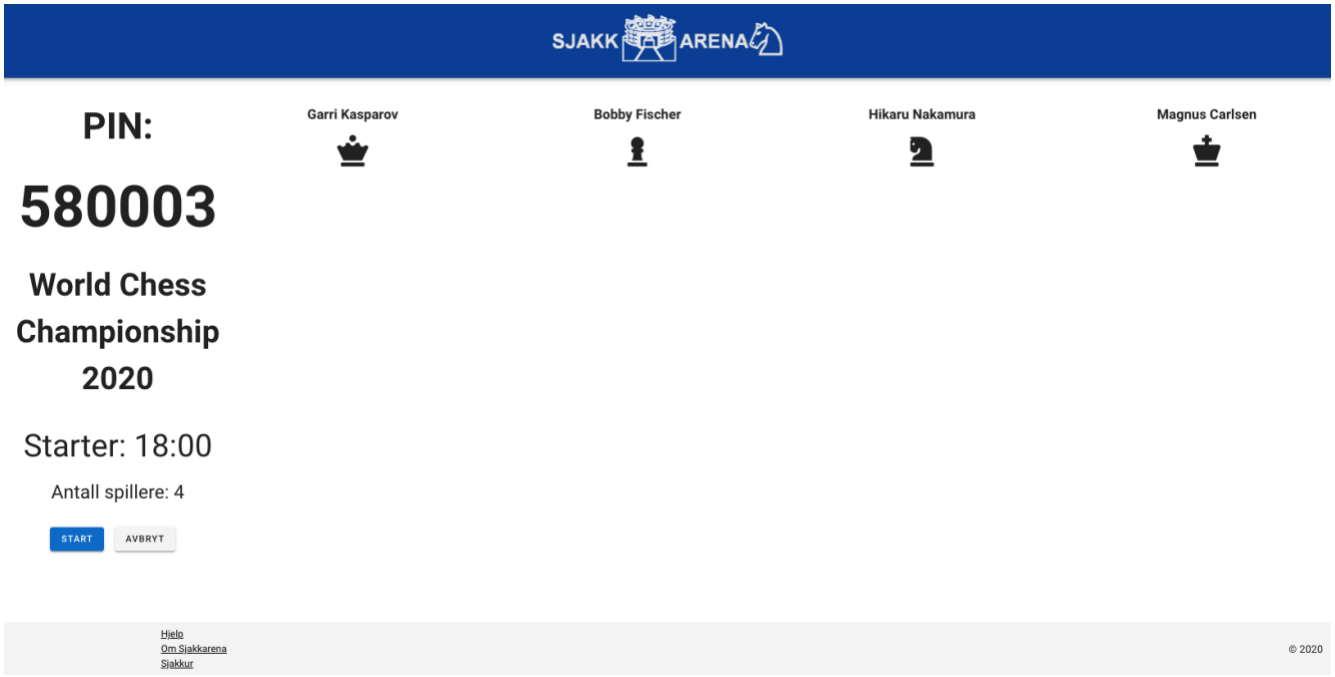
Det er også her validering av inputdataen. Dersom adminID-en ikke er korrekt, gis det beskjed om dette.



Figur 4-21 Side for å logge inn som turneringsvert

4.2.7 Turneringslobby

Dette er siden man kommer til når en turnering har blitt opprettet. Siden består av to deler: en informasjonssidebar og en liste over spillerne som blir med i turneringen. Listen inneholder en spillerkomponent som viser navn og en sort sjakkbrikke. Hvilken brikke spillerne får blir tilfeldig tildelt av backendapplikasjonen. Hver spiller får en brikke for å få applikasjonen til å føles mere «levende» istedenfor å bare ha en liste med navn.



Figur 4-22 Side etter turnering er opprettet

I Figur 4-22 kan vi se at sidebaren viser en pin som spillere bruker til å bli med i turneringen (avsnitt 4.2.10), navnet til turneringen, starttid for turneringen og en teller på hvor mange spillere som er med i turneringen. I tillegg er der to knapper. Disse blir brukt til å starte eller avbryte turneringen.

Gestaltprinsippene om nærhet og likhet (avsnitt 2.1.2) har blitt brukt til å designe denne siden. Nærhet er brukt når all informasjon om turneringen er plassert i et tenkt rektangel på venstre side, mens spillerlisten er i et tenkt rektangel på høyre side. I tillegg plasseres ikon og navn nært hver andre, noe som gjør at disse kobles til samme spiller. Likhetsprinsippet blir brukt når alle spillere representeres på samme måte.

Hovedinnholdet til siden er listen over spillere. Den blir oppdatert med en gang en spiller blir med i turneringen. Klient vet ikke når nye spillere har blitt registrert. Derfor har websocket (avsnitt 2.4.4) blitt brukt for å unngå for stor belastning på server. Antall spillere per linje vil forandre seg når skjermstørrelsen blir mindre eller større, men er ikke mer enn 4 spillere pr linje. Dette er for at listen ikke skal føles «klumpete» og at navn som er lange ikke skal overlappe.

Figur 4-23 viser hvordan spillerlisten er bygget opp. Den baserer seg på Vuetify sitt grid-system med offset som bryter ned listen når skjermstørrelsen endrer seg. Måten offset tallene ble bestemt på var ved å manuelt krympe vinduet og sette verdien som var best for hvert intervall av skjermstørrelser. Denne kodesnutten ligger videre inn i en container som lager en kolonne for informasjonssiden og en for spillerlisten.

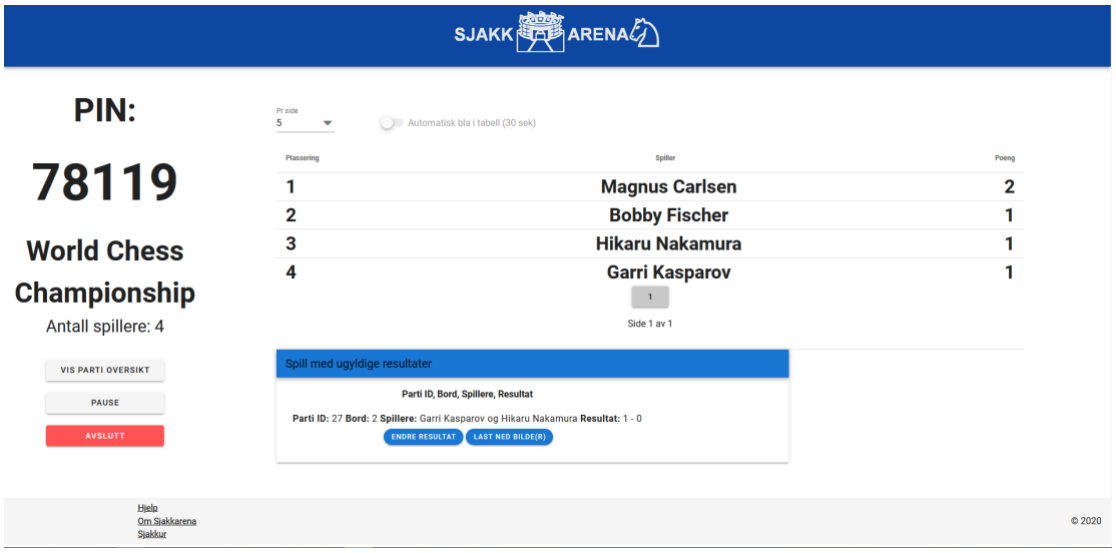
```
<!-- The player list. Offset to better fit smaller screens -->
<v-col
  offset="5"
  offset-sm="3"
  offset-md="2"
  offset-lg="1"
  offset-xl="0"
>
  <v-row
    align="start"
    justify="start"
    class="text-center"
    v-if="this.getAllPlayers.length !== undefined"
  >
    <!-- The individual players -->
    <player
      class="player"
      v-for="(player, index) in this.getAllPlayers"
      @click.native="handleRemovePlayer(player, player.user_id)"
      :player-name="player.name"
      :player-piece="player.icon"
      :key="index"
      :id="'player' + index"
    />
  </v-row>
</v-col>
```

Figur 4-23 Kode for å vise listen over spillere

4.2.8 Side når turneringen har startet

Når turneringen starter, blir verten navigert til siden vist i Figur 4-24. Her beholder verten samme strukturen som i lobby siden (avsnitt 4.2.7) med informasjonssidebar til venstre og diverse tabeller i midten av skjermen. Knappene i sidebaren har blitt byttet ut med tre knapper som er mere relevant etter at turneringen har startet.

Måten siden er designet på er å få det mest relevante i fokus. Det blir gjort ved å bruke fet skrift og bestemme størrelse på skriften med tanke på hvor interessant informasjonen til teksten er. Som Figur 4-24 viser er antall spillere for eksempel ikke uthevet og er mye mindre enn pin. Pin er mer interessant siden spillere bruker denne til å bli med i turneringen, mens antall spillere ikke er viktig for å spille i turneringen.



Figur 4-24 Side for vert under pågående turnering

I rangeringstabellen er alle tabelloppføringene klikkbare. Ved å klikke på en av spillerne blir en ny fane åpnet. Den nye fanen inneholder en spillerdetaljside (avsnitt 4.2.9). På grunn av at vertdelen av applikasjonen hovedsakelig vil bli vist på en storskjerm, blir siden med spillerdetaljer vist i en ny fane. På den måten vil listen over pågående partier og spillertabellen fortsatt være synlig for spillerne i lokalet.

Når man benytter seg av knappen for å vise partioversikt (se sidebar i Figur 4-24) vil innholdet i rangeringstabellen bli byttet ut med en oversikt over pågående sjakkpartier. Oppføringene i denne tabellen er ikke klikkbare og inneholder bordnummer, navn til begge spillerne og starttidspunkt for partiet. Det å vise starttidspunktet kan være interessant for spillere som venter på å få tildelt et nytt parti. Da kan en spiller som vil se et sluttspill gå til bordet som har spilt lengst eller om man vil se mer av et parti kan man gå til partiet som har spilt kortest.

Under rangeringstabellen/partitabellen er det en ny tabell. Denne inneholder informasjon om partier der spillerne ikke ble enige om resultat. Blir ikke spillerne enige kan de ta kontakt med turneringsansvarlig, så kan han bestemme resultatet. Den eneste forutsetningen for at turneringsansvarlig skal kunne bestemme et resultat, er at spillerne ikke klarer å bli enige.

Denne siden er i hovedsak bygd opp av komponentene TournamentInfo, Table og InvalidGames. TournamentInfo er også brukt i lobbyside (4.2.11) og instanser av Table brukes til å vise både tabell og partioversikt. Figur 4-25 og Figur 4-26 viser hvordan en har utnyttet komponenter for å unngå kodeduplikasjon. Disse figurene vise henholdsvis hvordan en spillertabell- og en partioversikt-instans av Tabell-komponenten blir laget. Ettersom at siden kun skal vise to typer tabeller brukes en boolsk-verdi, showLeaderBoard, til å bestemme om det er spillertabell eller partioversikt som skal vises. Tabeller henter innhold fra spillerlisten og listen over aktive parti. I tillegg er det spesifisert hvilken funksjon som skal kalles når en oppføring i spillertabellen blir trykt.

```
<Table
  class="tournament-table"
  v-if="showLeaderBoard"
  :object-list="Array.from(playerList)"
  :heading-list="leaderBoardHeadingList"
  :autoScrollOption="true"
  @entryClicked="handlePlayerClicked"
/>
```

Figur 4-25 Kode for å lage spillertabell-instans av Table-komponent

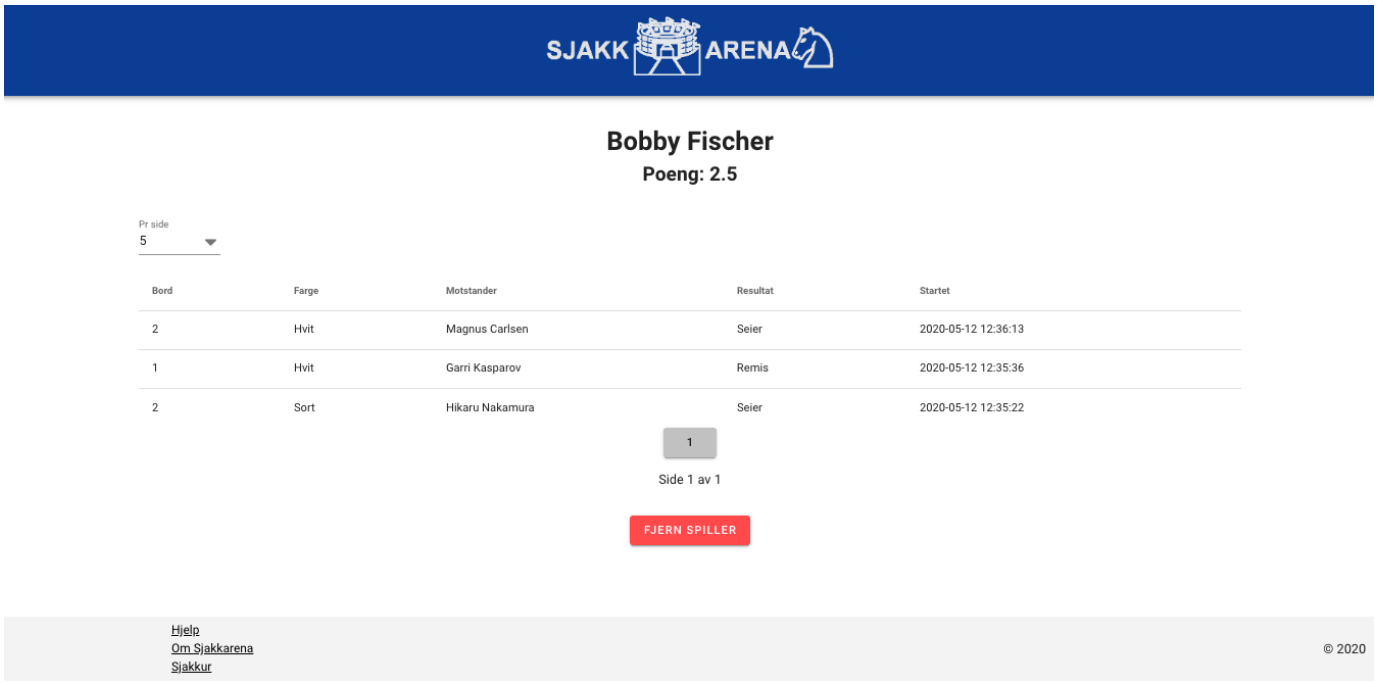
```
<Table
  class="tournament-table"
  v-if="!showLeaderBoard"
  :object-list="Array.from(gamesList)"
  :heading-list="activeGamesHeadingList"
  :autoScrollOption="true"
/>
```

Figur 4-26 Kode for å lage partioversikt-instans av Table-komponent

InvalidGames-komponenten viser oversikt over partier som ikke har gyldig resultat. I tillegg til å vise denne oversikten er det også mulig for verten å laste ned bilder spillerne har lastet opp for partiet. Denne funksjonen har en egen knapp: «last ned bilde(r)». Trykkes denne vil backend applikasjonen søke gjennom databasen og finne bilder som tilhører partiet. Applikasjonen vil da samle disse opp og legge dem sammen i en zip-fil som blir sendt til verten. Det ble bestemt å gjøre dette på denne måten ettersom det ikke kan garanteres at spillerne bare laster opp et bestemt antall bilder. Hvordan opplastningen foregår blir forklart i avsnitt 4.2.11. Den endelige zip-filen vil ha navnet som partinummeret i turneringen, og filene vil beholde sine originale navn de ble opplastet med.

4.2.9 Spillerdetaljer

Ved å trykke på en spiller i spillertabellen (avsnitt 4.2.8) vil spillerdetaljsiden bli åpnet i en ny fane. Denne siden inneholder informasjon om hvor mange poeng en spiller har oppnådd i turneringen, samt en tabell over tidligere spilte parti. Hver oppføring i tabellen inneholder bordnummer, brikkefarge, mostander, resultat og tidspunkt for partistart. Tabellen er her, i likhet med tabellene beskrevet i avsnitt 4.2.8, en instans av Table-komponenten.



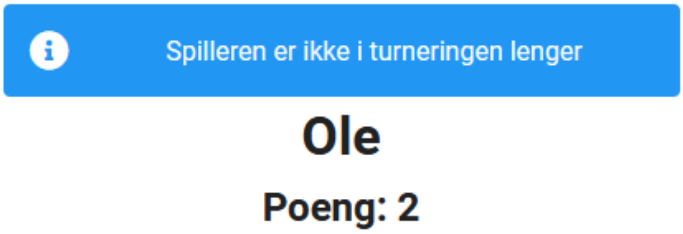
Figur 4-27 Side som viser detaljer om en spiller

Siden inneholder også en knapp som turneringsvert kan trykke på for å fjerne spilleren fra turneringen. Når denne knappen trykkes åpnes en dialogboks hvor turneringsvert kan bekrefte at han ønsker å fjerne spilleren, samt gi en begrunnelse for hvorfor spilleren ble fjernet.

Når turneringsvert har bekreftet at spilleren skal fjernes, kalles funksjonen removePlayerFromTournament (Figur 4-28). Denne funksjonen samler sammen nødvendig informasjon for å fjerne spilleren og sender denne til serveren. Om spilleren blir fjernet, vil en grønn boks med tekst vises i toppen av siden. Dersom det oppstår en feil, vil en rød boks med feilmelding vises.

```
removePlayerFromTournament() {
  this.kickDialog = false
  let payload = {
    id: this.player.user_id,
    started: true,
    msg: this.msg !== '' ? this.msg : 'blank'
  }
  this.removePlayer(payload).then(res => {
    this.color = 'green'
    this.removedMessage = 'Spiller fjernet! Du kan nå lukke denne fanen'
    this.icon = 'check'
  }).catch(err => {
    this.handleError(err)
  })
  this.removed = true
},
```

Figur 4-28 Funksjonen removePlayerFromTournament i spillerdetaljsiden

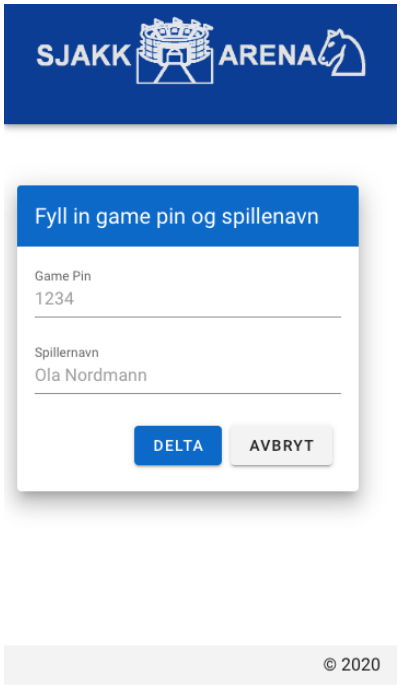


Figur 4-29 Spiller som lenger ikke er med i turneringen

Når en spiller har forlatt eller blitt kastet ut av turneringen vil boksen i Figur 4-29 vises over navnet til spilleren. Knappen for å fjerne spiller vil også være deaktivert for å forhindre unødvendige feil. Denne boksen har samme plassering som boksen for suksess eller feil når spiller blir kastet ut.

4.2.10 Side for å melde seg inn i turnering

Figur 4-30 viser siden hvor en spiller kan melde seg inn i en turnering. Siden inneholder et innmeldingsskjema hvor spilleren må oppgi en id, game pin, som representerer turneringen han skal melde seg inn i og et brukernavn. Brukernavnet må være unikt for turneringen. Skjemaet inneholder to knapper: en knapp for å melde seg inn i turneringen og en knapp for å avbryte prosessen med å melde seg inn.



Figur 4-30 Side for å melde seg inn i turnering

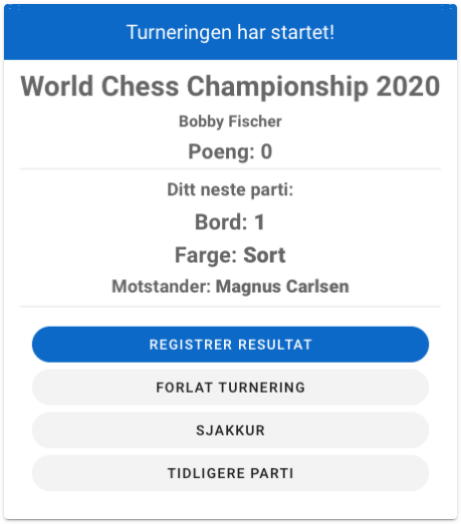
Begge inntastingsfeltene må være fylt ut for at skjemaet skal kunne sendes inn. I tillegg gis det tilbakemelding dersom inntastet «game pin» ikke er gyldig eller om spillernavnet allerede er tatt. I begge tilfeller kan spilleren forsøke å melde seg inn på nytt. Dette bidrar til at applikasjonen følger Donald Normans regel om å ikke la applikasjonen bli påvirket av feil (avsnitt 2.1.1).

4.2.11 Spillerlobby

Spillerlobby er siden spillere i hovedsak skal forholde seg til når de deltar i turneringer. Her vil spillere få instruksjoner om partiene, slik som motspiller, farge og bord som det skal spilles på. Hovedutfordringen med denne siden er å lage den slik at den kan presentere mye informasjon og ha flere funksjoner uten at den virker uoversiktlig og uforståelig.

Figur 4-31 Spillerlobby før turneringsstart

Figur 4-31 viser spillerlobby. Denne siden vises dersom en spiller blir med i turnering før turneringsstart. Ettersom applikasjonen tillater spillere å melde seg inn før turneringen starter, er det behov for å la brukere vite at de har blitt meldt opp og nå venter på at turneringen skal starte. Her får ikke brukeren tilgang på noen funksjoner ettersom det ikke er noe å gjøre enda. Spiller ser starttid og sluttid for turneringen, dens navn og spillerens navn i turneringen.



Figur 4-32 Spillerlobby når turnering har startet

Spillerne blir varslet om turneringsstart over Websocket (avsnitt 2.4.4). Etter turneringsstart vil spillere bli presentert for en side som i Figur 4-32. Denne siden kan deles opp i tre hovedseksjoner basert på hvordan de er fordelt som komponenter i Vue. Den øverste er seksjon med generell informasjon. Seksjonen med generell informasjon viser navnet på turneringen, etterfulgt av spillernavnet og poengscoren brukeren har opptjent. Seksjonen under er spillseksjonen med informasjon om spillerens neste parti.

Som Figur 4-32 illustrerer, inneholder denne seksjonen hvilket bord som skal benyttes, fargen brukeren skal spille med og navnet på motspilleren. Denne seksjonen endrer seg mest i livssyklusen til applikasjonen.

Tilbakemelding

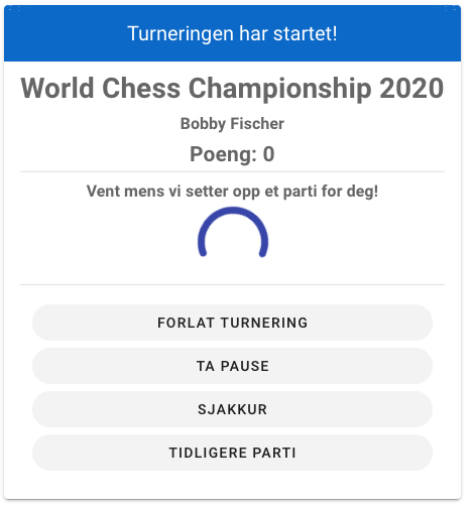
På sider hvor flere ting kan lastes inn uavhengig av hverandre, som i spillerlobbyen, har det blitt lagt til tekst som forteller at informasjon er i ferd med å lastes inn. Figur 4-33 viser slik tekst for feltet som inneholder en spillers poengsum. Dette er gjort for å gi brukerne bedre tilbakemelding slik som beskrevet i Don Normans designregler (avsnitt 2.1.1).

Figur 4-33 Innlastingstekst

Knapper

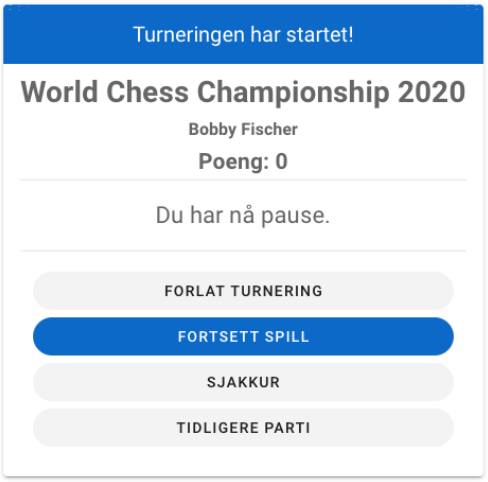
Under spillseksjonen finner vi knappeseksjonen. Her finner vi knapper som brukeren kan trykke på underveis i turneringen. Knappeseksjonen og spillseksjonen har påvirkning på hverandre og bestemmer hva som skal vises hvor. Når en spiller ikke har pause, eller er i et parti, vil spill- og knappe-seksjonen være slik som i Figur 4-34. Spillseksjonen viser nå en lastesirkel og knappeseksjonen har endret seg. Knappen for å registrere resultat er ikke lenger der, og knappen for å ta pause har kommet frem. Ingen av knappene er farget. Dette er for at ingen av funksjonene hos knappene fremmer spilleraktivitet.

For å hindre overlappende kommandosekvenser kan det meste av tilgjengelig funksjonalitet nås fra spillerlobby-siden med ett knappetrykk. På den måten unngår man det Norman kalte fangstfeil (avsnitt 2.1.1), hvor brukere av vane trykker på en knapp de ikke har intensjon om å trykke på.



Figur 4-34 Spillerlobby, spiller venter på parti.

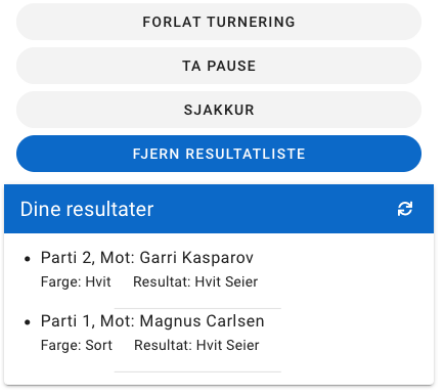
Når spiller trykker på pauseknappen, vil spillseksjonen endres til å bare vise teksten «Du har nå pause.» (se Figur 4-35). Knappeseksjonen vil vise en blå knapp for å fortsette spill. Når applikasjonen er i denne tilstanden, vil ikke spiller motta flere partier før «FORTSETT SPILL»-knappen trykkes. Brukeren kan fortsatt forlate turneringen eller se oversikt over sine tidligere partier.



Figur 4-35 Spillerlobby, spiller har tatt pause

Trykker brukeren på knappen for å se tidligere parti vil resultatseksjonen vises. Resultatseksjonen viser oversikt over brukeren sine 10 sist spilte parti (Figur 4-36). Her får man se informasjon om partier man har spilt, slik som navnet på motspillere, hvilken farge man har spilt og hvilket resultat som ble registrert.

I spillerlobbyen er det også en knapp for å åpne et sjakkur. Når sjakkur-knappen trykkes, åpnes et sjakkur i en ny fane (avsnitt 4.2.12).



Figur 4-36 Spillerlobby, spiller ser på sine tidligere resultater

Om brukere trykker på «forlat turnering»-knappen i knappeseksjonen, eller tilbake knappen i nettleseren, vil varselvinduet vist i Figur 4-37 dukke opp. Her kan man se at knappen for å avbryte er tydeligere enn knappen for å forlate turneringen. Dette varselvinduet dukker også opp for verten i turneringslobby. For å ta vekk oppmerksomheten fra alt annet som er på skjermen, legges varselvinduet oppå de andre bestanddelene. Samtidig vil det som varselvinduet ikke dekker bli uskarpt. På den måten oppnår man en forgrunn/bakgrunn-effekt i henhold til gestaltprinsippene (avsnitt 2.1.2). Oppmerksomheten til brukeren vil derfor flyttes til avgjørelsen om å forlate eller fortsette turneringer.

Figur 4-37 Forlate turnering

Det har blitt laget en OvalButton-komponent. Det er denne komponenten som blir brukt til å lage knappene i spillerlobbyen. Hver OvalButton-knapp består av en stylet v-btn med tekst. HTML for denne komponenten er vist i Figur 4-39. Blant annet teksten og hvilken funksjon som skal kalles når knappen trykkes spesifiseres i forbindelse med instansieringen av komponenten (Figur 4-38).

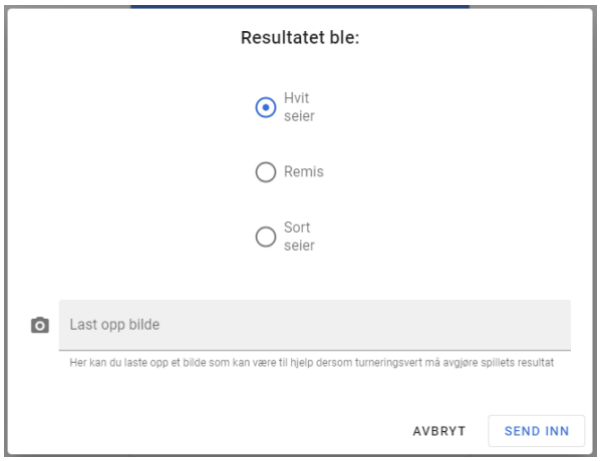
```
<oval-button
  text="Forlat turnering"
  @buttonClicked="leaveTournament"
/>
```
Figur 4-38 Instans av OvalButton

```
<template>
  <v-btn
    class="btn"
    :color="buttonColor"
    block
    rounded
    depressed
    @click="buttonClicked"
  >
    {{ text }}
  </v-btn>
</template>
```
Figur 4-39 HTML for OvalButton-komponent

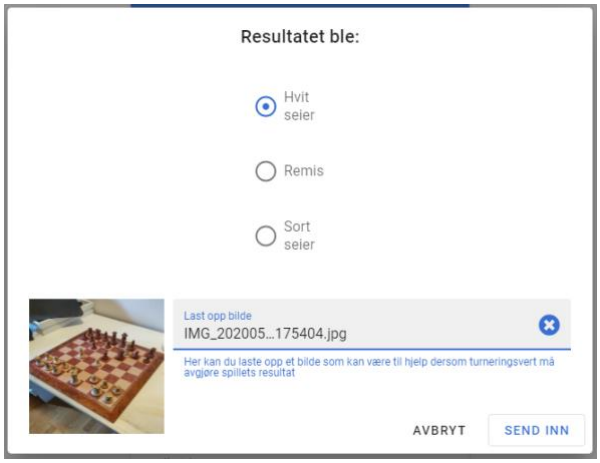
Registrering av resultat

Når et parti er fullført, skal spillerne registrere resultatet. Dette gjøres ved å trykke på «registrer resultat»-knappen i spillerlobby. Brukeren vil bli møtt av en dialog illustrert i Figur 4-40. Brukere velger resultatet som ble oppnådd i sitt parti ved å trykke på en av radio knappene som er tilgjengelig. Skulle brukere ønske å laste opp et bilde som bevis for resultatet, må feltet hvor det står «last opp bilde» trykkes. Dette bildet kan brukes for å bevise sjakkmatt, patt eller utløpt tid. For å forhindre at brukere laster opp feil bilde har vi satt inn en fremvisning (Se Figur 4-41) for valgt bilde. På denne måten får brukerne se valgt bilde for å forsikre seg om at det er riktig. Når brukere trykker «send inn» vil både resultat og eventuelt bilde bli lastet opp. Bildet blir lagrer i en egen mappe i backend applikasjonen, og det blir innført assosiering mellom partiet, spiller som lastet opp og bildet i databasen.

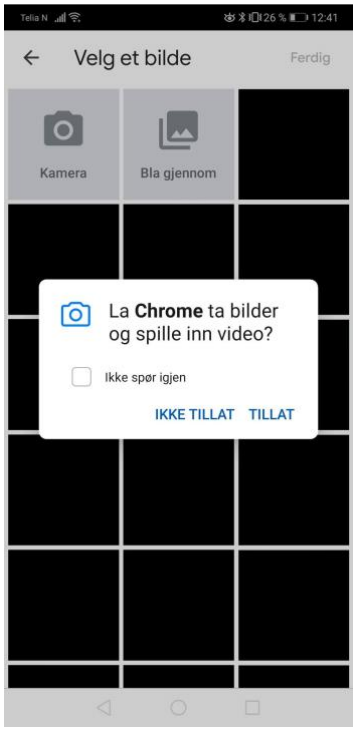
Skulle brukeren være på mobil og trykker på bildefeltet vil brukeren bli spurt om å dele kamera med applikasjonen for å ta bilde (Figur 4-42Figur). Dette vil gi brukeren en bedre brukeropplevelse enn å lukke nettleser, åpne kamera, ta bilde, gå tilbake til applikasjonen for å så lete gjennom filutforskeren på telefonen.



Figur 4-40 Registrer resultat dialog med mulighet for bildeopplastning.



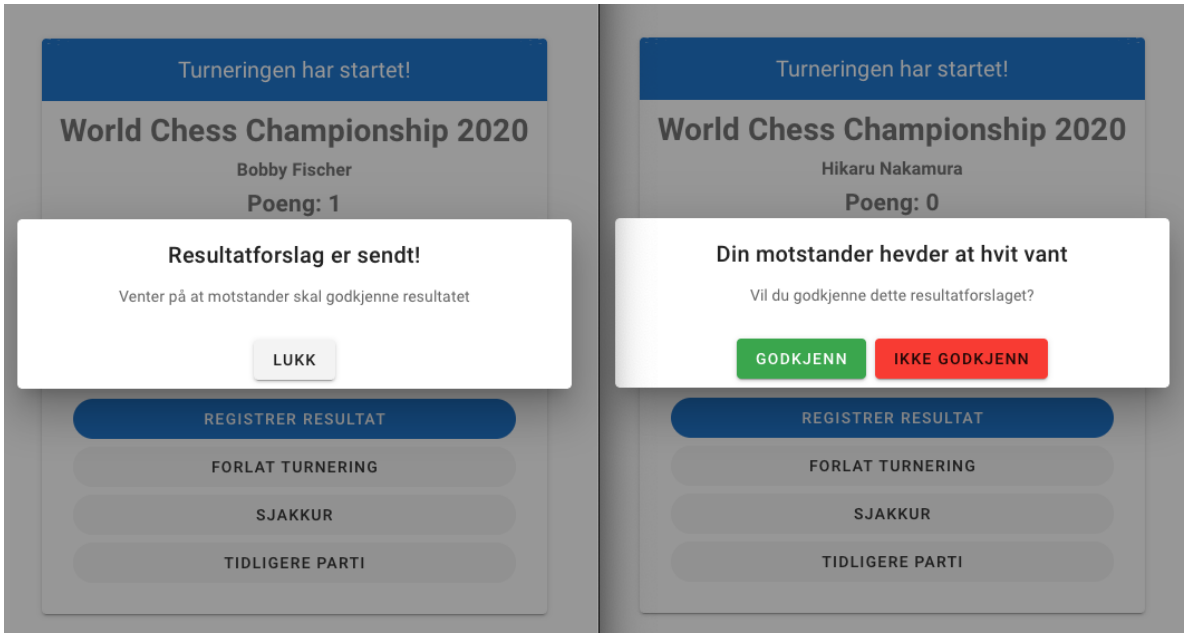
Figur 4-41 Registrer resultat og last opp bilde



Figur 4-42 Skjermbilde av mobil når spurt om tillatelse til kamera

Validering

For at et resultat skal regnes som gyldig må én av to betingelser være oppfylt. Enten må begge spillerne være enige om hva resultatet ble eller så må turneringsvert ha bestemt resultatet. Når en spiller legger inn et resultatforslag, sendes resultatforslaget til motstanderen. Motstanderen får så et valg om å godkjenne eller ikke godkjenne resultatet. Dersom motstanderen ikke godkjenner resultatet, sendes partiet til turneringsvert (avsnitt 4.2.8) slik at han kan legge inn et resultat. Selv om partiet er sendt til turneringsverten, kan spillerne forsøke å legge inn resultatet på nytt. Figur 4-43 viser hvordan validering av resultat foregår. Til venstre i bildet ser vi skjermen til spilleren som har sendt resultatforslaget. Til høyre er skjermen til spilleren som skal godkjenne forslaget. Her har man benyttet seg av gestaltprinsippet om forgrunn/bakgrunn (avsnitt 2.1.2). Dialogbokser legges over det spillerne vanligvis ser i applikasjonen for å flytte fokuset over på resultatvalideringen. Det at spillere kan legge inn resultatet på nytt om de er uenige, bidrar til å følge Donald Normans prinsipp om å la programmet være upåvirket av feil (avsnitt 2.1.1).



Figur 4-43 Validering av resultat

Skulle motspilleren trykke på «IKKE GODKJENN» vil begge spillere få et varsel om dette. Begge spillerne får da mulighet om å prøve igjen, eller gå til turneringsvert. Turneringsverten har da autoritet til å bestemme resultatet for partiet.

Utkasting

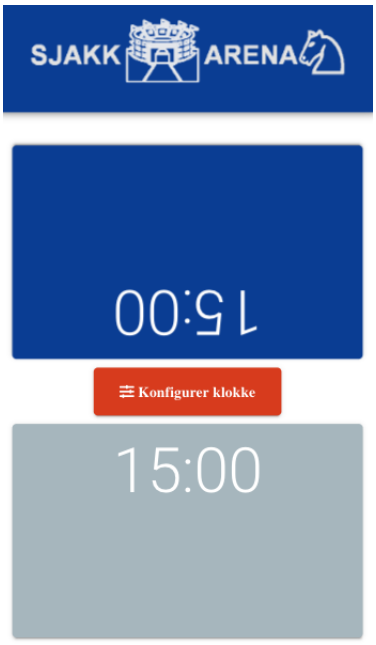
Når spillere blir kastet ut av turneringen vil de få beskjed om dette. De vil få en dialog som teller ned til at spilleren automatisk blir navigert til hovedsiden (avsnitt 4.2.4). Dialogen inneholder også begrunnelsen verten gir for utkastelsen, og en knapp for å gå tilbake til hjemmesiden.

Figur 4-44 Spiller som har blitt kastet ut

4.2.12 Sjakkur

Fra spillerlobbyen kan spillere få tak i et sjakkur som de kan bruke til å holde styr på tiden under et sjakkparti. Som Figur 4-45 viser, består sjakkuret av to knapper – én for hver spiller – som brukes til å starte nedtellingen av motstanderens tid. I tillegg har sjakkuret en knapp for å konfigurere klokken. Sjakkuret er designet for enheter med touchskjerm. Under bruk av sjakkuret skal enheten ligge flatt på et bord med knappene vendt mot hver sin spiller. Tiden til hver spiller er vist øverst på spillernes respektive knapp. Den ene knappen er gitt en lys farge, mens den andre er gitt en mørkere farge. Spilleren med hvite brikker skal bruke den lyse knappen, mens spiller med sorte brikker skal bruke den mørke. Sjakkuret er laget slik at spilleren med sorte brikker må trykke på sin knapp for at sjakkuret skal starte nedtellingen. Når en spiller trykker på sin knapp, stopper nedtellingen av spillerens tid, mens motspillerens klokke begynner å telle ned.

Ved å trykke på knappen for å konfigurere klokken, kan spillerne bestemme antall minutter og sekunder de skal starte med, samt antall sekunder man skal få per trekk.



Figur 4-45 Sjakkur

De to knappene som blir brukt til å starte motspillerens tid er laget i en Vue-komponent kalt ChessClockButton. Disse komponentene ligger i ChessClock-viewet som styrer hver ChessClockButton. ChessClock gir ChessClokkButton informasjon om hvor mye tid hver spiller starter med, hvor mye tid som skal legges til for hvert trekk, om nedtelling skal pågå og om klokken skal nullstilles.

Figur 4-46, Figur 4-47 og Figur 4-48 viser de viktigste funksjonene i ChessClockButton. Når komponenten får beskjed om å starte nedtelling kalles funksjonen countDown vist i Figur 4-46. Denne funksjonen kalles hver gang inputvariabelen, countDown, til ChessClockButton endrer verdi. Dersom nedtelling skal startes, og knappen ikke er i ferd med å nullstilles, vil funksjonen reduceRemainingTime (Figur 4-47) kalles hvert 100. millisekund. Dersom gjenstående tid er mindre enn null, vil en «times-up»-hendelse bli publisert.

Som Figur 4-48 viser, vil ChessClockButton slutte å redusere gjenstående tid i intervaller når den trykkes. I tillegg vil tiden som spillerne skal få per trekk legges til gjenstående tid. En «update:count-down»-hendelse publiseres til ChessClock for å fortelle at nedtelling skal stanses.

```
watch: {
  countDown: function (countDown) {
    if (countDown && !this.reset) {
      this.intervalStart = new Date().getTime()
      this.countDownInterval = setInterval(this.reduceRemainingTime, timeout: 100)
    }
  },
```

Figur 4-46 Funksjon for å starte nedtelling i ChessClockButton

```
reduceRemainingTime: function() {
  if (this.remainingTime > 0) {
    let now = new Date().getTime()
    this.remainingTime -= now - this.intervalStart
    this.intervalStart = now
  } else {
    this.$emit( event: 'times-up')
  }
}
```

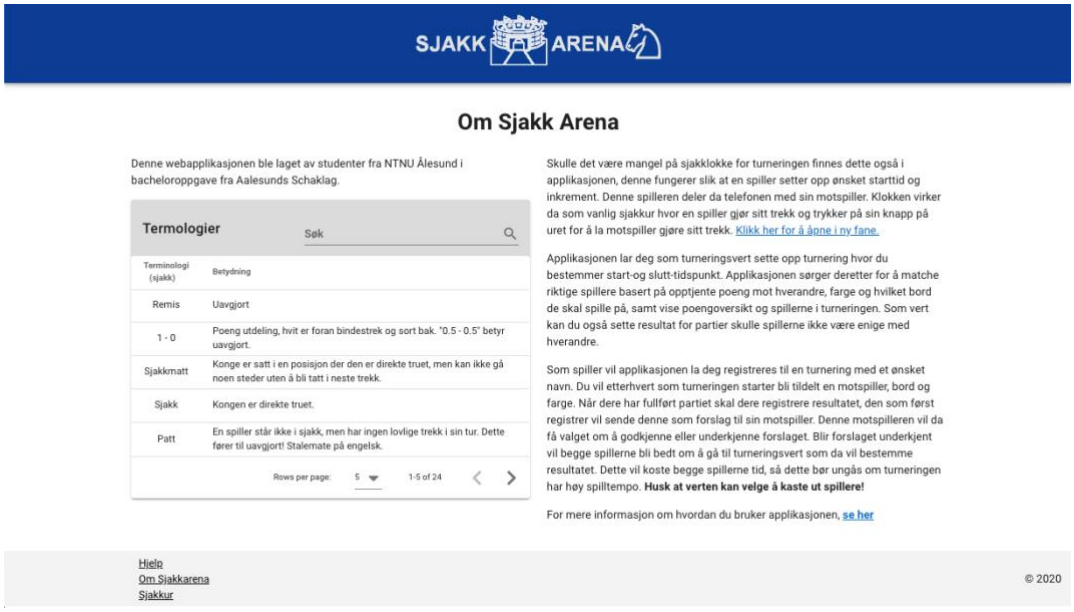
Figur 4-47 Funksjon for å redusere gjenstående tid for en spiller i ChessClockButton

```
clicked: function () {
  if (this.countDown) {
    this.remainingTime += this.additionalTimePrMove * this.millisecondToSecondRatio
    clearInterval(this.countDownInterval)
    this.$emit( event: 'update:count-down',  args: false)
  }
```

Figur 4-48 Funksjonen som kalles når en knapp trykkes i ChessClockButton

4.2.13 Om-siden

På om-siden (Figur 4-49) står det om konteksten applikasjonen ble laget i og hvilke funksjoner som finnes for spillere og turneringsverter. En lenke til denne siden finner man i footeren til applikasjonen. Siden åpnes automatisk i en ny fane når lenken trykkes. Dette for å ikke la brukerne forlate turnering med uhell eller andre uønskede hendelser. I tillegg til funksjonsforklaringen finnes det også en tabell som viser ulike regler og terminologier som blir brukt i sjakk. Denne ser man i Figur 4-50. I denne figuren ser man også at der er mulig å søke i tabellen. Dette betyr at brukeren ikke trenger å kaste bort tid på å bla igjennom tabellen, og kan fokusere mest mulig på spillet. Denne søkeboksen leter etter ord både i terminologi- og betydningskolonnen. Dette fører til at brukeren kan få opp flere resultater enn ett, og får større sjanse for å få ønsket resultat frem.

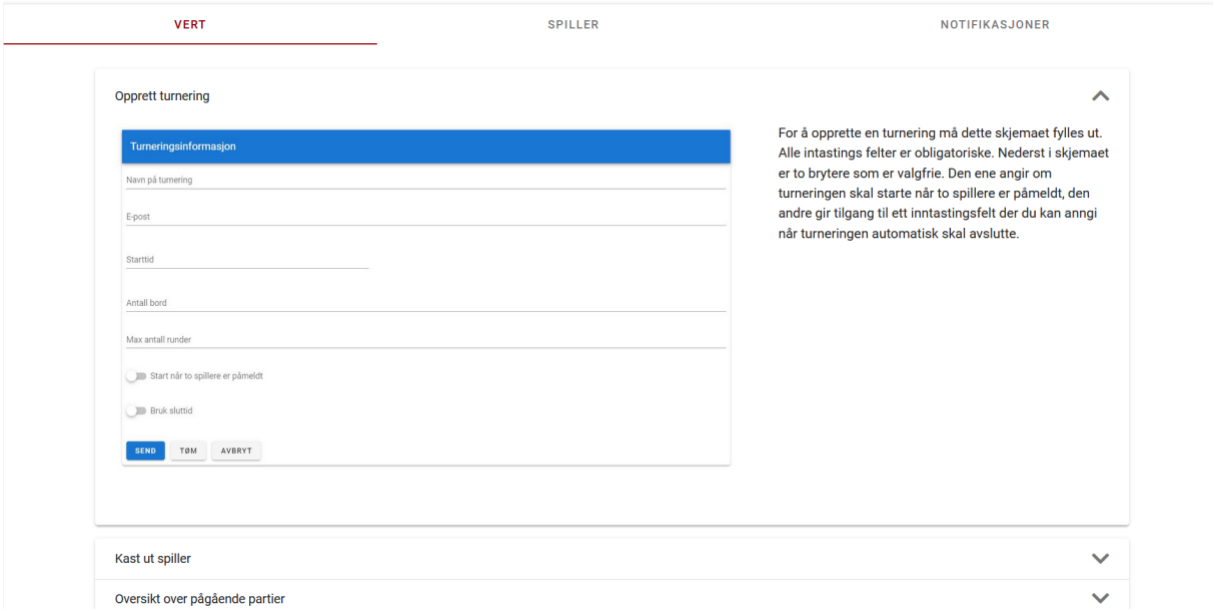


Figur 4-49 Omsiden

Figur 4-50 Søking i terminologilisten

4.2.14 Hjelpeside

For å gi brukerne av applikasjonen en enda bedre brukeropplevelse har det blitt laget en hjelpeside. I denne siden finner man informasjon om hvordan applikasjonen fungerer, for eksempel hvordan man som turneringsvert kan opprette en turnering (Figur 4-51) eller kaste ut en spiller. Alle delene av denne siden inneholder bilder med tekst som beskriver hva man skal/kan gjøre.



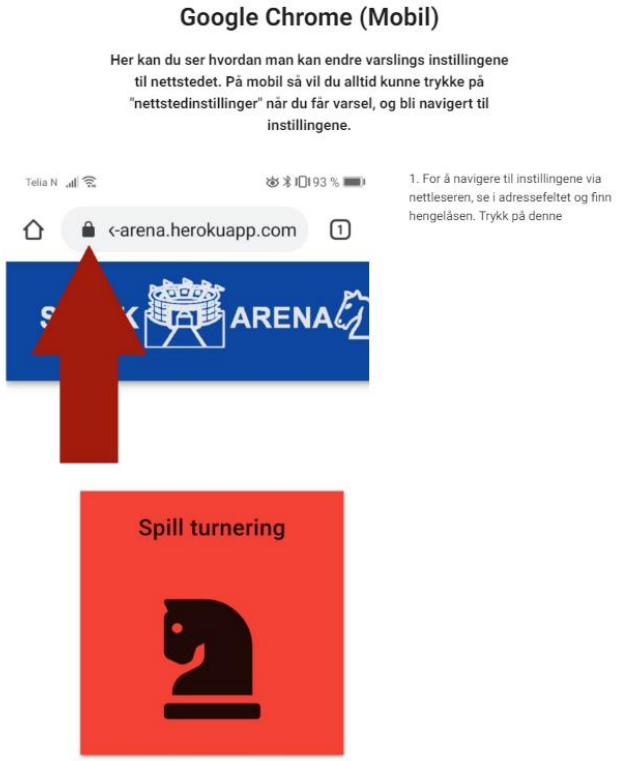
Figur 4-51 Hjelpeside, vert - opprett turnering

Figur 4-52 Hjelpeside navigering

Navigeringen i hjelpesiden består av en meny der man velger hva en trenger hjelp med. Under notifikasjoner så er der også en undermeny der en kan velge hvilken nettleser som blir benyttet. Nettleserne applikasjonen har informasjon om er Firefox, Google

Chrome for PC, Google Chrome for Android og Samsung internett. Under notifikasjoner er der også en avkryssingsboks som er deaktivert. Denne er der for å vise om brukeren har godtatt eller avslått muligheten til å få notifikasjoner. Om brukeren skulle benytte seg av en enhet eller nettleser som ikke støtter push notifikasjoner vil undermenyen erstattes av en tekst som forteller brukeren om dette Figur 4-53.

Figur 4-53 Enheten eller nettleseren støtter ikke push varsler



Figur 4-54 Hjelpeside, Notifikasjoner - Google chrome

Hjelpesiden inneholder også informasjon om hvordan brukere kan endre om applikasjonen har tillatelse til å sende push notifikasjoner (Figur 4-54). Dette er med ettersom ikke alle vet hvordan man endrer innstillingene for push-varsler.

Komponenten som viser hvordan man endrer notifikasjonstillatelse, er bygget på en slik måte at det enkelt skal kunne utvides til å vise for flere nettlesere. Dette er gjort ved at all tekst og alle bilder blir definert i en JavaScript-fil. Komponenten renderer antall faner og ikon etter hvor mange som er definert i filen.

```
const notificationHelpPageData = [{
  title: 'Mozilla Firefox',
  icon: 'fab fa-firefox-browser',
  description: DESKTOPDESCRIPTION,
  texts: [
    'For å se hvilke tillatelser du har gitt dette nettstede så går du til adressefeltet og ser etter en hengelås. ' +
     'Når du har funnet hengelåsen trykker du på den, og en informasjons boks vil vises. ' +
     'Ved å trykke på "X" vedsiden av "Sende varsler" vil du ikke bli sendt varsler lenger, men du vil bli spørt på nytt ' +
     'når du går inn i en ny turnering. For å da ikke mota varsler er det bare å velge "tillat aldri" Dette kan endres ' +
     'om du angrer på å ikke godta varsler så endres dette på samme måte som beskrevet over og i bildet. '
  ],
  images: [
    require('@/assets/help-page/firefox/firefox2.png')
  ],
  altTags: [
    'Dialog box for å endre varslings instillinger'
  ]
},
```

Figur 4-55 Javascriptfil med bilder og tekst for hjelpe siden

Komponenten som inneholder informasjon om notifikasjoner, er laget på en slik måte at det skal være enkelt å utvide med flere nettlesere senere. Dette gjøres ved å definere et objekt i listen som blir eksportert fra JavaScript-filen (Figur 4-55). Der definerer man tittel, ikonet som skal vises i menyen, en kort beskrivende tekst, teksten til hvert bilde og bildene en skal ha. Der skal også være en liste over «alt» tekst som benyttes i alt-attributtene til bilde elementene. Det kan bli definert to ikoner eller ikon og en etikett, for eksempel for å vise Google Chrome for Android. Siden tekstene og bildene er definert i hver sin liste, må teksten og tilhørende bilde ha samme posisjon i listene. Det vil si at bildet på plass N i bildelisten vil bli koblet til teksten på plass N i tekstlisten.

```
    <v-tab
      v-for="item in data"
      :key="item.title">
      <v-icon>
        {{ item.icon }}
      </v-icon>
      <v-icon v-if="item.icon2">
        {{ item.icon2}}
      </v-icon>
      <v-label v-if="item.label">
        {{ item.label }}
      </v-label>
    </v-tab>
  </v-tabs>
```

Figur 4-56 Undermenyen til notifikasjoner

Figur 4-56 viser hvordan applikasjonen lager en meny utfra hvor mange objekter det er i listen og gir ikonene og etiketten som er ønsket.

```
<v-tab-item v-for="(item, index) in data"
            :key="index">
<h1>
  {{item.title}}
</h1>
  <h3 v-if="item.description">
    {{ item.description }}
  </h3>
  <div class="content-container" v-for="(image, index) in item.images"
      :key="index+10">
  <img :src="image"
      :title="item.altTags[index]"
      :alt="item.altTags[index]"/>
  <p>
    {{ item.texts[index] }}
  </p>
  </div>
</v-tab-item>
</v-tabs-items>
```

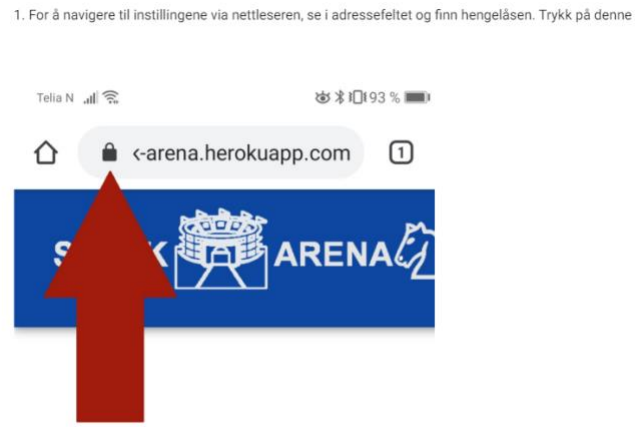
Figur 4-57 Rendering av tittel, beksrivelse, bilde og tekst

Når det blir navigert til notifikasjonsdelen av hjelpesiden blir det laget en «v-tab-item» for hvert objekt i listen (Figur 4-55). «v-tab-item» er en Vuetify komponent som tar seg av hvilket av de definerte objektene som skal vises. Figur 4-57 viser at det blir laget et nytt element for hvert bilde i objektet som skal vises. Dette elementet er blitt stylet som et rutesystem ved hjelp av CSS. Rutesystemet består av to ruter, en for bilde og en for tekst.

```
}
@media(max-width: 800px){
  p{
    grid-column: 1/2;
    grid-row: 1/2;
  }
  img{
    grid-row: 2/2
  }
  h3{
    max-width: 90%;
  }
}
</style>
```

Figur 4-58 Css for å endre grid oppsett

For å få hjelpesiden brukervennlig på store og små skjermer er den blitt laget av et gridsystem. For alle skjermstørrelser med bredde mindre enn 800 piksler vil oppsettet til siden endre fra bilde til venstre og tekst til høyre (Figur 4-54) til å ha teksten over bildet (Figur 4-59).



Figur 4-59 Hjelpeside på enhet med bredde mindre enn 800 piksler

4.2.15 Pagineringsknapper

Pagineringsknapper er et essensielt tillegg til tabeller og andre objekter som blir delt inn i sider. Disse knappene er her bygget som en selvstendig komponent som består av en '«' knapp som hopper til venstre, en liste av knapper og en '»' knapp som hopper til høyre. Alle disse ligger på en rekke. Under rekken er en liten tekst som beskriver hvor mange sider det er og hvilken side man er på. For å tydeligere vise hvilken side man er på, får den knappen som tilhører den aktuelle siden mørkere bakgrunn. Dette er også et eksempel på hvordan Don Normans designregel om god tilbakemelding (avsnitt 2.1.1) er brukt i designet av pagineringsknappene. Pr. nå blir pagineringsknappene kun benyttet i tabellkomponenten, men er designet slik at den kan benyttes flere plasser senere.

Figur 4-60 paginering tilstand 1

Pagineringskomponenten har fire forskjellige tilstander (se Figur 4-60, Figur 4-61, Figur 4-62 og Figur 4-63). I tilstand én er det ingen knapper som er utenfor det synlige intervallet. Det er tenkt at siden alle knapper er synlige er det ikke nødvendig med hoppeknappene.

Figur 4-61 paginering tilstand 2

I tilstand to er det flere sider til høyre enn det som vises og '»' knappen vil dukke opp (se tilstand 3 for funksjonalitet til '»' knappen).

Figur 4-62 paginering tilstand 3

I tilstand tre er det flere sider på begge ender av de synlige knappene. Hovedfunksjonen til '«' og '»' er å hoppe til den første knappen som er utenfor de som er synlige. For å holde antall sider den hopper konsistent, blir det regnet ut hvor mange sider den skal hoppe når den laster inn. Det vil si at i Figur 4-62 vil '»'-kappen hoppe til side 7, men i tilstand 2 (Figur 4-61) vil den hoppe til side 4.

Figur 4-63 paginering tilstand 4

Tilstand fire er samme prinsippet som tilstand to, bare motsatt vei. Her er det ingen knapper til høgre for det synlige intervallet.

```
visibleButtons () {
  let button = 1
  let buttonsArr = []
  // Creates an array of all the buttons.
  while (buttonsArr.length < this.lastButton) {
    buttonsArr.push(button)
    button++
  }
  switch (true) {
    case this.maxVisibleButtons >= this.lastButton:
      this.alterShowJumpLeft( show: false)
      this.alterShowJumpRight( show: false)
      return buttonsArr
    // If there are no buttons to the left of the visible range
    case !this.hasButtonsLeft():
      this.alterShowJumpLeft( show: false)
      this.alterShowJumpRight( show: true)
      return buttonsArr.slice(0, this.maxVisibleButtons)
    // No buttons to the right of the visible range
    case !this.hasButtonsRight():
      this.alterShowJumpLeft( show: true)
      this.alterShowJumpRight( show: false)
      return buttonsArr.slice(this.lastButton - this.maxVisibleButtons, this.lastButton)
    default:
      this.alterShowJumpLeft( show: true)
      this.alterShowJumpRight( show: true)
      // There are buttons to both right and left of the visible range.
      return buttonsArr.slice(this.minButtonNr(), this.maxButtonNr())
```

Figur 4-64 PaginationButtons tilstander

```
// Returns if there is any buttons to the left outside of the visible ones
hasButtonsLeft() {
  return this.minButtonNr() >= 1
},
// Checks if there is any buttons to the right outside of the visible ones
hasButtonsRight() {
  return this.maxButtonNr() < this.lastButton
},
// Returns the index of the lowest visible number
minButtonNr() {
  return Math.floor( x: this.activeButton - (this.maxVisibleButtons / 2))
},
// Returns the highest number that should be visible.
maxButtonNr() {
  return Math.floor( x: this.activeButton + (this.maxVisibleButtons / 2))
},
/*
```

Figur 4-65 funksjoner å bestemme tilstand

```
},
lastButton () {
  let lastButton = Math.ceil( x: this.numberOfItems / this.prPage)
  return lastButton > 0 ? lastButton : 1
}
```

Figur 4-66 Funksjon for å regne ut tallet til den siste knappen

Figur 4-64 og Figur 4-65 viser litt av hvordan pagineringsknappene er programmert. I forelderkomponenten blir det bestemt hvor mange tallknapper som maksimalt kan være synlig på en gang. Om det ikke blir bestemt i forelderkomponenten, vil det være maksimalt fem synlige knapper.

Ved den første tilstanden i switch-casen (Figur 4-64) sjekkes det om antall knapper er større enn indeksen til den siste knappen i listen. Om den er det, trenger man ikke «hoppe»-knappene og returnerer bare listen. I alle andre tilstander trenger man «hoppe»-knappene. Tallknappene man trenger blir returnert med slice-funksjonen. Slice funksjonen fungerer slik at det blir laget en liste som inneholder alle elementene mellom startindeks og sluttindeks. Elementet på startindeks vil følge med, men ikke element på sluttindeks.
Eksempel på utregning i komponenten:

Antall objekt i listen: 92
Objekt per side i liste: 5
Aktiv knapp: 6
Siste knapp: $\lceil 92/5 \rceil = 19$ (For at det skal fungere må tallet alltid rundes opp siden det er antall fulle sider som blir regnet ut før avrunding)
minButtonNr: $6 - \lfloor 5/2 \rfloor = 3$ (Rundes ned siden vi vil ha heltall som indeks)
maxButtonNr: $6 + \lfloor 5/2 \rfloor = 8$ (Rundes ned siden vi vil ha heltall og for å få riktig antall knapper med tanke på indekser i lister)

Denne utregningen gir at hasButtonsLeft vil gi **True** det samme vil hasButtonsRight. Som vil gi tilstand 4. Og knappene [«][4][5][**6**][7][8][»] vil vise.

4.2.16 Testing

Unit

Som nevnt i 3.2.1 benyttes JEST for å teste frontendapplikasjonen. JEST-testene er ment for å teste forretningslogikken til komponentene. Testene skal være til hjelp for å forstå hvordan komponenter fungerer og passe på at ingenting blir ødelagt ved endring av kodene. Det testes bl.a. at elementer som skal være synlige er synlig, at egendefinerte hendelser har bestemt data og om innholdet i komponenten samsvarer med et øyeblikksbilde. Alle testene som blir laget for frontend har filnavnet komponentNavn.spec.js. Ut ifra kompleksiteten til komponentene har testene forskjellig antall sjekker.

```
import { createLocalVue, mount } from '@vue/test-utils'
import Vue from 'vue'
import Vuetify from 'vuetify'

import PaginationButtons from '../../../src/components/PaginationButtons'

Vue.use(Vuetify)

const localVue = createLocalVue()

const factory = (propsData) => {
  return mount(PaginationButtons,  options: {
    localVue,
    propsData: {
      ...propsData
    }
  })
}
```

Figur 4-67 PaginationButtons.spec.js setup

Figur 4-67 viser hvordan en testfil blir satt opp. Siden vi benytter Vue, importer vi vue-test-utils med JEST. Test-utils inneholder diverse «mount» funksjoner som hjelper til å bygge komponentene i isolasjon. Hver test må importere test-utils og Vue. Den første funksjonen som blir definert er «factory». Denne blir brukt i hver enkelt test og setter opp komponentene med nødvendige data.

```
it( name: 'should emit event with the number of the clicked button', fn: async() => {
  const wrapper = factory( propsData: {
    numberOfItems: 20,
    prPage: 3 })
  wrapper.find( selector: '#button5').trigger( eventName: 'click')
  await wrapper.vm.$nextTick()
  expect(wrapper.emitted( event: 'update:page')[0]).toEqual( expected: [5])
})
it( name: 'should match snapshot', fn: () => {
  const wrapper = factory( propsData: {
    numberOfItems: 20,
    prPage: 3 })
  expect(wrapper.html()).toMatchSnapshot()
})
```

Figur 4-68 PaginationButtons tester

Figur 4-68 Viser to av testene som er definert for pagineringsknappene (avsnitt 4.2.15). Den første testen sjekker at knappen som blir trykket på avløser den forventede hendelsen med det forventede innholdet. Den andre testen definert i figuren er en test av øyeblikksbilde. Denne tar et «bilde» av komponentene første gang den kjører. Ved senere tester så sjekker den om bildene er like og gir beskjed til utvikleren om hva den fant og hva den forventet. På den måten kan utvikleren bestemme om det var en uforventet endring og koden må fikses, eller om endringen var forventet og ett nytt «bilde» skal lagres.

Cypress

Som nevnt i avsnitt 3.2.1 ble Cypress også benyttet til å teste applikasjonen. Testene som ble laget i Cypress sjekker blant annet om elementer er synlige når de skal være det og om applikasjonen reagerer som forventet ved feil på nettverkskall. Det er laget tester for nesten hver side i applikasjonen og det er laget to «full flyt»-tester som går gjennom applikasjonen som en spiller eller som en vert som oppretter turnering. Disse to testene blir brukt mere som korte demonstrasjoner enn tester av applikasjonen.

Det er ikke alle testene som er fullt ut «end to end»-tester, men der Cypress heller går mellom og fungerer som en server. Dette gjør det enklere og raskere å teste applikasjonen om spesielle situasjoner skulle oppstå.

Figur 4-69 viser en av testene der vi ber Cypress sette opp en server. Denne testen sjekker at det blir vist en suksessboks og at fjernknappen blir gjort utilgjengelig når verten får til å kaste ut en spiller i siden for spillerdetaljer (avsnitt 4.2.9). Deretter sjekker den om en feilmelding blir gitt dersom server svarer med «*http 400: Bad Request*».

Cypress tillater å legge til egne funksjoner eller kommandoer. Cy.kickplayer() som er vist i figuren er eksempel på dette. Disse funksjonene legges i en egen fil som heter commands.js i mappen support.

```
it( title: 'Should remove player, display error when failed and success on success', fn: () => {
  cy.server()
  cy.route({
    method: 'PATCH',
    url: 'http://localhost:8080/tournament/player/inactivate/*',
    response: {}
  }).as( alias: 'removePlayer')
  cy.playerDetailsKick()
  cy.get('[data-cy=remove]').should( chainer: 'be.disabled')
  cy.get('[data-cy=alert-box]').should( chainer: 'be.visible')

  cy.visit('/tournament/player/133')
  cy.route({
    method: 'PATCH', // Route all GET requests
    url: 'http://localhost:8080/tournament/player/inactivate/*',
    delay: 0,
    status: 400,
    response: {}
  }).as( alias: 'removePlayer')
  cy.playerDetailsKick()
  cy.get('[data-cy=remove]').should( chainer: 'be.disabled')
  cy.get('[data-cy=alert-box]').should( chainer: 'be.visible')
})
```

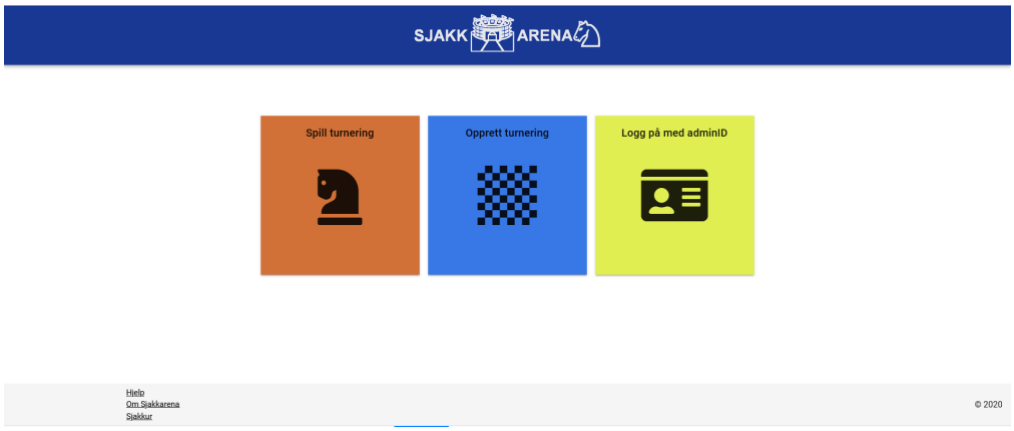
Figur 4-69 Cypress test for å kaste ut spiller.

4.2.17 Tilgjengelighet

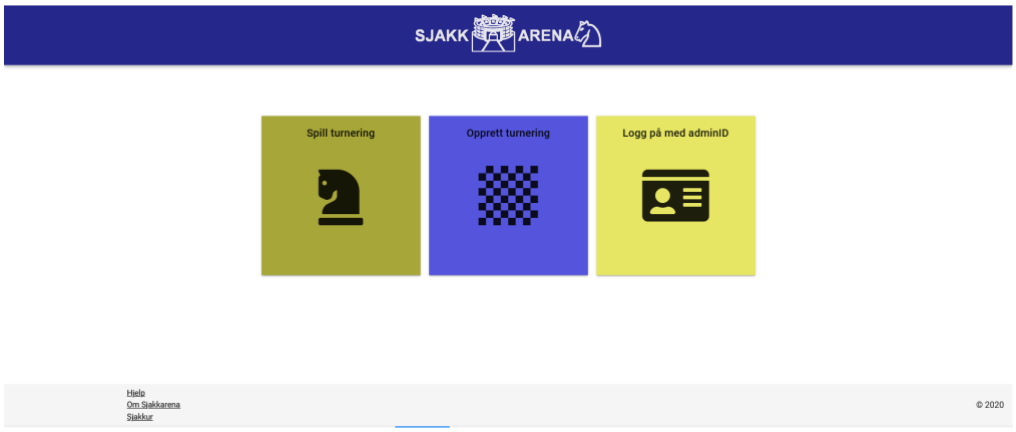
Under design av frontendapplikasjonen var tilgjengelighet i fokus, uten at dette var et spesifikt krav fra oppdragsgiver. Det er ikke mange forskjellige farger i applikasjonen, men der det er har det vært satt søkelys på å vær konsistent med hva for eksempel en knapp med fargen rød gjør. En rød knapp representerer en negativ handling, slik som å avslutte en turnering eller å ikke godkjenne et resultat. Grønn representerer et «klarsignal», som å godkjenne et resultat. Mens blå knapper tar en videre i applikasjonen.

På hovedsiden i applikasjonen har vi valgt å bruke «fliser» med forskjellige farger for å navigere. Her ble det viktig å huske på de forskjellige gradene og typene av fargeblindhet som finnes. For å finne de rette fargene ble hovedsiden sett på med simulert fargeblindhet. Dette kan bli gjort på flere måter. Det er nettsteder som tilbyr simulering ved å linke til siden din, laste opp bilder av applikasjonen. Enkelte nettlesere har også funksjonalitet for dette. I for eksempel Mozilla Firefox konsollen er der en tab som heter «accessibility», der man kan se om nettsiden oppfyller forskjellige krav og simulere fargeblindhet. I Google Chrome kan man bruke plugin for fargeblindhet.

I Figur 4-14 ser vi hvordan hovedsiden ser ut for personer med vanlig fargesyn.



Figur 4-70 simulert deuteranomali – grønnsvakhet



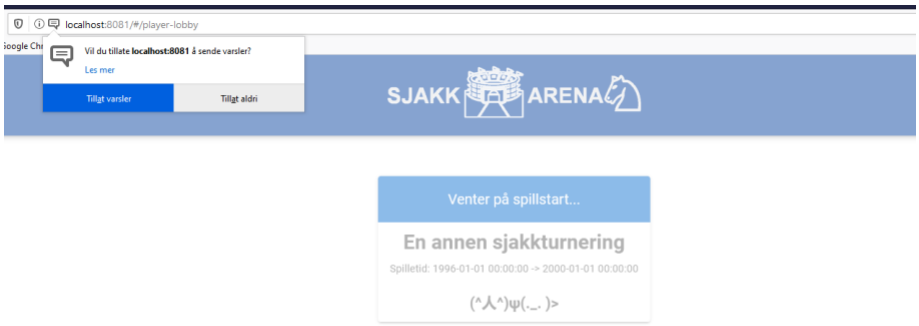
Figur 4-71 simulert protanopi – rødblindhet

Figur 4-70 viser en simulering av grønnsvakhet og Figur 4-71 viser rødblindhet. Begge disse figurene er simulert ved hjelp av Mozilla Firefox sin innebyggede funksjon. I figurene er det fortsatt relativt enkelt å skille mellom de forskjellige elementene.

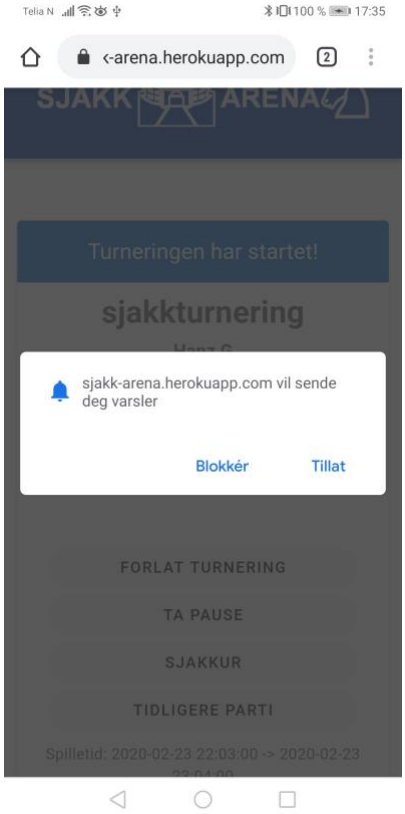
En annen ting som er gjort for å gjøre applikasjonen mere tilgjengelig er å benytte tittel- og alt-attributtene for alle bildeelementene i applikasjonen. Dette gjør at de som benytter seg av skjermlesere får en beskrivelse av hva bildet inneholder.

4.2.18 Push-varsler

For at brukerne ikke skal trenge å ha telefonen oppe og åpen på nettsiden hele tiden under turneringen, har det blitt lagt til push-varsler for å gi beskjed når en spiller har fått en ny mostander. Dette er ingen påtvunget funksjonalitet, men er noe brukeren vil bli spurt om å bruke når de blir med i en turnering. Dette skjer først etter at man blir med i en turnering for å vise brukeren hva siden skal med push-varsler. Dersom det blir spurt om med engang noen går inn på nettsiden vil det være vanskeligere å forstå hva dette skal brukes til og dermed vil mange sikkert blokkere varsler. Brukerne vil kun bli spurt om de vil aktivere varsler første gangen de blir med i en turnering i en nettleser. Og vil ikke bli spurt flere ganger i den nettleseren. Valget vil ikke lagres forbi denne nettleseren. Om en bruker skulle skifte nettleser eller bli med på en annen enhet vil vedkommende bli spurt på nytt.

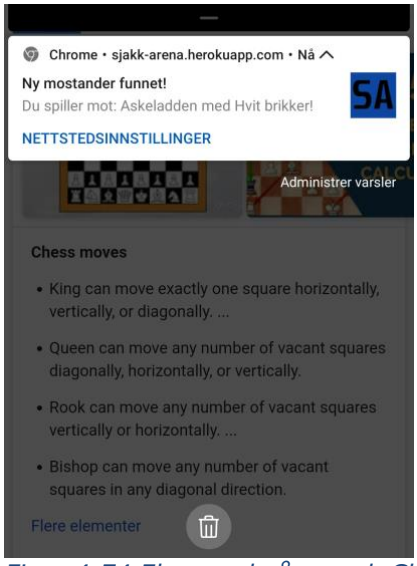


Figur 4-72 Bruker blir spurt om varsler pc



Figur 4-73 Mobilbruker får spørsmål om å motta pushvarsler (chrome)

Figur 4-72 og Figur 4-73 viser hvordan en bruker blir spurt om varsler når de blir med i en turnering. For å få flest brukere til å svare på om de vil ha varsler eller ikke, blir siden gjort delvis gjennomsiktig slik at spørsmålsboksen kommer i fokus. Her benyttes gestaltprinsippet forgrunn/bakgrunn (avsnitt 2.1.2) for å flytte oppmerksomheten til brukeren. Når brukeren tillater eller blokkerer varsler, vil siden gå tilbake til normal.



Figur 4-74 Eksempel på varsel, Chrome - Android 9

Figur 4-74 viser hvordan et varsel vil se ut på en Android-telefon med Google Chrome. Varslene applikasjonen sender inneholder et ikon, tittelen «Ny mostander funnet!» og en tekst som sier hvilken farge mottakeren har og hvem som er motstanderen. Den inneholder også en standardlenke som tar brukeren til innstillingene nettleseren har for varsler. Der kan brukeren skru av denne funksjonaliteten.

```
self.addEventListener( type: 'push', listener: event => {
  let job = event.data.json()
  let promiseChain = getMatchingWindowClient().then(windowClient => {
    if (windowClient === null || !windowClient.focused) {
      return self.registration.showNotification( title: 'Ny mostander funnet!',
        options: {
          'body': 'Du spiller mot: ' + job.opponent + ' med ' + job.colour + ' brikker!',
          'icon': './notificationIcon.jpg',
          'vibrate': [500] // Not available in Android Oreo or higher.
        })
    }
  })
  event.waitUntil(promiseChain)
})
```

Figur 4-75 Notifikasjon

Figur 4-75 viser koden for å gi disse push-varslene til brukeren. Det har blitt satt opp en Service worker (avsnitt 3.2.1) som lytter etter push-hendelser. For at applikasjonen ikke skal føles masete, vil den ikke sende varsler om nettsiden er i fokus. Er den ikke i fokus – dvs. den er lukket, en annen fane er i fokus eller nettleseren er lukket (kun mobil) – vil den sende varsel om nytt parti.

Når brukeren trykker på varselet, vil nettleseren åpne spillsiden. Om siden allerede er åpen, men er i en annen fane, vil fane bli byttet. Er den ikke åpen vil siden bli åpnet i en ny fane.

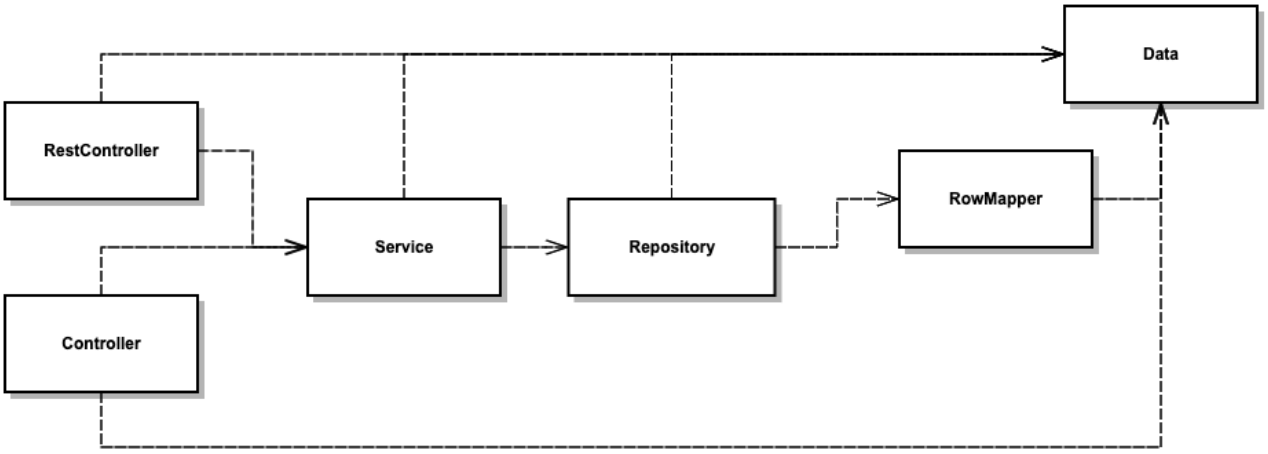
For å kunne mota push varsler må frontend først hente ut public key fra backend også sende denne nøkkelen til push service API-et i nettleseren. Dette gjøres gjennom en «service worker»-instans. Når nøkkelen er sendt vil frontendapplikasjonen få tilbake et push objekt som inneholder en lenke fra push service API-et, og to p256 nøkler som skal bli brukt til å kryptere data som blir sendt i et push varsel. Når dette objektet er sendt til backend trenger frontend sin service worker bare å lytte etter push hendelser.

4.2.19 Sikkerhet

Som beskrevet i avsnitt 2.8.1 er cross server scripting en form for å sende skadelig scriptkode til andre brukere. For å beskytte applikasjonen mot slike angrep har vi prøvd å følge Vuejs sine sikkerhetstips (Vue, u.d. (e)) og de generelle kodestilsrådene til Vuejs. Eksempelvis så har vi konsekvent benyttet {{}} taggene når applikasjonen skal vise input fra en bruker til en annen. Det disse taggene gjør er å be nettleseren formatere teksten på en slik måte at spesielle tegn som < blir gjort om til <. Dette gjør at nettleseren ikke vil tolke tilførsel av JavaScript kode som JavaScript, men som ren tekst og dermed ikke kjøre koden.

4.3 Backendapplikasjon

Hovedoppgaven til backendapplikasjonen er å sørge for at data generert på klientsiden blir lagret i en database, samt gi data til klienter ved behov. I tillegg skal backendapplikasjonen opprette nye spill i turneringen. Hovedstrukturen i backendapplikasjonen er vist i Figur 4-76. SpringBoot har blitt brukt til å utvikle applikasjonen, noe som har hatt innvirkning på hvilke klasser applikasjonen består av. RestController- og Controller-klassene definerer API-et. Service-klassen håndterer forretningslogikken i applikasjonen, mens Repository-klassene samhandler med databasen. RowMapper-klassene avbilder resultatet fra databasespørringer inn i Data-klasser.



Figur 4-76 Klassediagram for de viktigste klassekategoriene i backendapplikasjonen

4.3.1 REST API

REST-API-et tilbys via RestController-klassene. Disse klassene er plassert i en mappe kalt restcontroller. Videre er klassene delt opp etter hvilken brukertype som skal bruke API-et og hva brukeren skal gjøre. F.eks. er alle RestController-klasser som tilbyr et API for spillere lagt under undermappen «player». Mappen «player» inneholder så klasser for å håndtere spiller-, spill- og turneringsinformasjon henholdsvis kalt PlayerRESTController, PlayersGameRESTController og PlayersTournamentRESTController. Det samme gjelder for brukere som styrer turneringer. I tillegg er det laget en klasse for å håndtere API-et til brukere som ikke er autentisert. Figur 4-77 viser filstrukturen for RestControllerne.

Inputdata til ResetControllerne er stort sett i JSON-format (avsnitt 2.5.1) eller ligger i URI-ene brukt til å identifisere de ulike REST-ressursene.

Figur 4-77 Filstruktur for RestControllere

HTTP-metoder er brukt i henhold til teori beskrevet i avsnitt 2.4.1. GET-metoden er brukt når informasjon om turneringer, spill eller spillere skal hentes ut fra serveren. Når nye brukere skal legges til og klienten skal få en JWT tilbake brukes POST-metoden. Delete brukes når en bruker skal slettes og PATCH brukes når en bruker skal settes på pause. I tillegg blir PUT brukt når et resultat skal legges til.

I Spring blir RestController-klasser merkes med @RestController. Metodene som er grensesnittet mot klientene merkes med @RequestMapping. Figur 4-78 viser hvordan dette er gjort i PlayerRESTController. Alle forespørsler til PlayerRESTController må sendes til «/player» pluss en streng som spesifisere metode. For å sette spiller på pause må klient sende en PATCH-forespørsel til «/player/pause».

```
@RequestMapping(value = "/pause", method = RequestMethod.PATCH)
public ResponseEntity<String> pause() {
    try {
        playerService.pausePlayer(RESTSession.getUserId());
        return new ResponseEntity<>(HttpStatus.OK);
    } catch (TroubleUpdatingDBException e) {
        e.printStackTrace();
        return new ResponseEntity<>(e.getMessage(), HttpStatus.BAD_REQUEST);
    }
}
```

Figur 4-78 Metode i PlayerRESTController for å sette en spiller på pause

4.3.2 WebSocket

Ved å bruke WebSocket kan man sende oppdateringer til spillere og turneringer uten at disse må be om å bli oppdatert (avsnitt 2.4.4). For eksempel blir WebSocket brukt til å oppdatere spillerlisten i turneringslobbyen (avsnitt 4.2.7), og poengsum og parti i

spillerlobbyen (avsnitt 4.2.11). I denne applikasjonen blir protokollen STOMP (avsnitt 2.4.5) brukt over WebSocket-protokollen.

Controller-klasser blir brukt til å motta meldinger over WebSocket. Controller-klassene er organisert og navngitt på samme måte som RestController-klassene (avsnitt 4.3.1). I Spring blir Controller-klasser merket med @Controller og metodene som tar imot meldinger blir merket med @MessageMapping.

Controller-klassene og RestController-klassene deler på de samme Service-klassene.

Figur 4-79 viser metoden, getPoints, som blir brukt til å be om å få tilsendt sin egen poengsum. Metoden getPoints henter brukerId-en til spilleren og ber Service-klassen om å få informasjon om spilleren. Deretter brukes spillerinformasjonen til å sende poengsummen til spilleren.

```
@MessageMapping("/player/points")
public void getPoints(Authentication authentication){
    int playerId = WebSocketSession.getUserId(authentication);
    Player player = playerService.getPlayer(playerId);
    sendPointsToPlayer(player);
}
```

Figur 4-79 Metode for å sende poeng til spillere i PlayerController

4.3.3 Forretningslogikk

Hver RestController har sin egen Service-klasse. Disse Service-klassene har ansvaret for å håndtere forretningslogikken i applikasjonen. Et eksempel på dette er UnauthenticatedUserService som håndtere forespørsler om å registrere brukere. Når UnauthenticatedRESTController mottar en forespørsel fra en klient om å registrere seg som en bruker, ber RestControlleren Service-klassen om å ta seg av denne forespørselen. Figur 4-80 viser metoden som RestControlleren kaller i dette tilfellet. Det metoden gjør er bl.a. å sjekke om brukernavnet brukeren vil bruke finnes fra før. I tillegg vil Service-klassen be om et ikon til brukeren, be om at brukeren legges til i databasen og kalle funksjonen som skal håndtere at en bruker er lagt til i databasen.

```
public String handleAddPlayerRequest(Player player) {
    if (playerRepository.doesPlayerExist(player)){
        throw new NameAlreadyExistsException("Name already taken, try a new one!");
    }
    try {
        player.setIcon(PlayerIcons.getRandomIcon());
        int userId = playerRepository.addNewPlayer(player);
        onNewPlayerAdd(player.getTournamentId());
        return jsonCreator.createResponseToNewPlayer(userId);
    } catch (TroubleUpdatingDBException e) {
        throw new TroubleUpdatingDBException(e);
    }
}
```

Figur 4-80 Metode for å håndtere forespørsler om å legge til en spiller

4.3.4 Data

Alle Service-klasser holder på en eller flere Repository-klasser. Det er Repository-klassene som har ansvar for å kommunisere med databasen via et JdbcTemplate-objekt. Et eksempel på dette er metoden getGame i GameRepository-klassen (Figur 4-81). JdbcTemplate-en bruker i getGame en GameRowMapper til å lage et Game-objekt ut fra resultatet av SQL-spørringen. GameRowMapper og andre RowMapper-klasser er plassert i mappers-mappen. Game, og andre klasser som representerer mulige returobjekt fra databasespørringene, ligger i data-mappen.

```
public Game getGame(int gameId) {
    return jdbcTemplate.queryForObject( sql: "SELECT * FROM sjakkarena.game WHERE game_id = " +gameId, gameRowMapper);
}
```

Figur 4-81 Eksempel på hvordan en Repository-klasse henter ut data fra en database

4.3.5 Hendelser

Hendelser styrer en god del av backendapplikasjonen. Hendelser er delt inn i tre kategorier: spillhendelser, spillerhendelser og turneringshendelser. Spillhendelser er for eksempel at et spill er laget eller at et resultat er lagt til. Spillerhendelser er bl.a. at en spiller er lagt til eller at det har skjedd en oppdatering av spillerlisten. Mens en turneringshendelse kan være at turneringen har startet eller sluttet.

I Spring er en hendelse en klasse som arver ApplicationEvent-klassen. For å varsle de klassene eller metodene som abonnerer på de ulike hendelsene, brukes et ApplicationEventPublisher-objekt. Metoder som abonnerer på hendelser merkes med @EventListener. For å lage hendelsene har det blitt definert EventCreator-klasser, én for hver hendelsestype. Disse klassene har ansvaret for å samle inn nødvendig data i forbindelse med hver hendelse, opprette hver hendelse og for å be ApplicationEvenPublisher-objektet om å publisere hendelsene.

Et eksempel på bruk av hendelser i applikasjonen er følgende. Etter at turnering har startet ber TournamentService (Figur 4-82) TournamnetEventCreator (Figur 4-83) om å lage en TournamentStartedEvent. Etter at TournamentStartedEvent-et er publisert kalles metoder i TournamentController- og PlayersTournamentController-klassene (Figur 4-84 og Figur 4-85). Disse sender så oppdatering over WebSockets til turneringsverten og spillere om at turneringen har startet.

```
private void onTournamentStart(int tournamentId) {
    tournamentEventCreator.createAndPublishTournamentStartedEvent(tournamentId);
}
```

Figur 4-82 Metoden onTournamentStart i TournamentService-klassen.

```
public void createAndPublishTournamentStartedEvent(int tournamentId){
    List<Player> players = playerRepository.getPlayersInTournament(tournamentId);
    applicationEventPublisher.publishEvent(new TournamentStartedEvent( source: this, tournamentId, players));
}
```

Figur 4-83 Metoden createAndPublishTournamentStartedEvent i TournamentEventCreator-klassen.

```
@EventListener
public void onTournamentStart(TournamentStartedEvent tournamentStartedEvent){
    for (Player player : tournamentStartedEvent.getPlayers()){
        informPlayerAboutTournamentState(player.getId(), active: true);
    }
}
```

Figur 4-84 Metoden onTournamentStart i PlayersTournamentController-klassen

```
@EventListener
public void onTournamentStart(TournamentStartedEvent tournamentStartedEvent){
    sendActiveStateToTournament(tournamentStartedEvent.getTournamentId(), active: true);
}
```

Figur 4-85 Metoden onTournamentStart i TournamentController-klassen

4.3.6 Oppgaver

I backendapplikasjonen er det definert to typer oppgaver: StartTournamentTask og EndTournamentTask. Begge oppgavene er klasser som arver Runnable-klassen. Når en turnering opprettes oppgir turneringsvert tidspunkt for turneringsstart og, om ønsket, tidspunkt for turneringsslutt. Tidspunktet for turneringsstart og turneringsslutt blir tidspunktet oppgavene skal gjennomføres. Disse oppgavene blir sammen med tidspunktet for å gjennomføre oppgavene, gitt til et TaskScheduler-objekt. Når tidspunktet for turneringsstart eller turneringsslutt er passert, vil TaskScheduler publisere en hendelse som gjør at metodene for å starte eller slutte en turnering i TournamentService-klassen blir kalt.

4.3.7 Sikkerhet

Autentisering og autorisering

Turneringsbrukere opprettes ved å sende inn et skjema for turneringsoppretting. Ved innlevering av skjemaet blir turneringen tildelt en unik id. Denne iden blir sammen med en beskrivelse av brukerrollen, lagt i en JSON web token (JWT) (avsnitt 2.5.2). JWT-en blir så signert med en hemmelig nøkkel. Det samme skjer når en spiller sender inn sitt innmeldingsskjema. Dersom en turneringsvert har oppgitt riktig e-postadresse, vil han få tildelt en unik id han senere kan bruke til å logge seg inn med.

Springapplikasjonen er konfigurert slik at den delen av REST API-et som er ment for spillere kun kan brukes dersom JWT-en forteller at brukeren er en spiller. Det samme gjelder for API-et for turneringer. Denne konfigurasjonen gjøres i klassen WebSecurityConfig.

Lagring av informasjon

Applikasjonen inneholder en del informasjon som uautoriserte ikke bør få tak i. Eksempel på slik informasjon er AdminID, nøkkel brukt til å signere JWT-er og brukernavn og passord for e-postkontoen brukt til å sende adminID til turneringsvert.

For å redusere sannsynligheten for at AdminId-en blir stjålet, blir den hashet før den lagres i databasen. Et tilfeldig generert salt blir brukt i hashingen av adminId-en.

Når det kommer til krypteringsnøkkel brukt til å signere JWT, blir denne lagret i en fil. De eneste sikkerhetstiltakene som er gjort her er å gi filen et tilfeldig filnavn. I tillegg bør filen lagres på en plass på serveren som kun den brukeren som kjører applikasjonen har tilgang til. Brukernavn og passord til e-postkontoen må skrives inn i kildekoden til

backendapplikasjonen. Et alternativ er å bruke et hvelv til å lagre denne type informasjon.

Det har blitt laget en «proof of concept»-løsning hvor Hashicorps Vault brukes. Denne løsningen finnes i vedlegg 6. I denne løsningen blir den som starter backendapplikasjonen bedt om å oppgi en token for å logge seg på hvelvet. Denne tokenen blir så gitt til Vault-klassen som setter opp kommunikasjonen mellom Spring-applikasjonen og hvelvet (Figur 4-86). Spring Vault har en del innebygget funksjonalitet for å kommunisere med Hashicorps Vault. Dette gjør det enklere å bruke hvelv sammenlignet med fil. Figur 4-87 viser hvordan «proof of concept»-løsningen leser data fra hvelvet. I metoden spesifiseres hvor data skal hentes fra og hvilket objekt data-en skal avbildes inn i.

```
public static void init(String token) {
    VaultEndpoint vaultEndpoint = new VaultEndpoint();
    vaultEndpoint.setScheme("http");
    vaultTemplate = new VaultTemplate(vaultEndpoint, new TokenAuthentication(token));
}
```

Figur 4-86 Initialisering av kommunikasjon med hvelv.

```
public static <T> VaultResponseSupport<T> read(String destination, Class<T> c) {
    VaultResponseSupport<T> response = vaultTemplate.read( path: "kv/sjakkarena/" + destination, c);
    return response;
}
```

Figur 4-87 Metode som leser og returnerer data fra en plass i hvelvet

4.3.8 Turneringsplanlegging

Ettersom at oppdragsgiver hadde noen ønsker som gjorde det vanskelig å bruke et eksisterende turneringssystem, ble Monrad (avsnitt 2.3.2) tilpasset oppdragsgivers ønsker.

I Monrad er det forskjell mellom hvordan man setter opp parti i første runde og hvordan parti senere i turneringen settes opp. I første runde blir spillerne tilfeldig tildelt et startnummer som de beholder ut turneringen. Startnummertildelingen gjøres i SQL-funksjonen «get_random_bib_number» (se vedlegg 7) når en spiller føres opp i databasen. Spillerne sorteres så i en liste etter startnummer og får tildelt et nummer etter plassering i listen[4]. Spillere med partallsnummer skal spille med hvite brikker, mens de med oddetallsnummer skal spille med sorte brikker. Spillere med nummer som er nærmest hverandre skal spille mot hverandre. Tildelingen starter med nummer 1, som skal spille mot nummer 2. Deretter skal nummer 3 spille mot 4 osv.

Etter at et parti i en runde er spilt, trekkes nye parti. Alle spillere som ikke spiller et parti, eller har pause, sorteres etter gjennomsnittspoeng. De spillerne som har likt gjennomsnittspoeng sorteres etter startnummer. En spiller skal møte den spilleren som ligger nærmest seg selv på listen, dersom visse krav er oppfylt. Blant annet skal spillere ikke møte hverandre mer enn én gang i løpet av turneringen. I tillegg skal spillere ikke spille med samme farge mer enn tre ganger på rad. En spiller skal heller ikke spille mer enn halvparten av partiene sine pluss ett parti med én farge.

I mappen «AdaptedMonrad» (se vedlegg 7) finner man de klassene som har blitt brukt til å sette opp parti. Det har blitt laget to ulike klasser for å håndtere ulikheten i hvordan partier blir tildelt ved og etter turneringsstart: «AtTournamentStartAdaptedMonrad» og

[4] Startnumre går ikke fra 1 til n. Se drøfting 5.4 for mer informasjon.

«AfterTournamentStartAdaptedMonrad». Begge disse klassene arver den abstrakte klassen «AdaptedMonrad». «AdaptedMonrad» tilbyr muligheten til å hente ut nye spill via «getNewGames»-metoden. Denne metoden er den eneste som er synlig for andre klasser. «AdaptedMonrad» har også en abstrakt metode, «setupGames», som klassene som arver «AdaptedMonrad» implementerer. Metoden «setupGames» fungerer som startpunktet for å implementere de forskjellige måtene å tildele parti på. Figur 4-88 og og Figur 4-89 viser implementasjonen av setupGames i henholdsvis «AtTournamentStartAdaptedMonrad» og «AfterTournamentStartAdaptedMonrad». For å oppnå løs kobling (avsnitt 2.2.2) er datatyper i input og datatype som returneres så høyt opp i arvhierarkiet som praktisk mulig.

```
@Override
protected List<Game> setupGames(Deque<Player> playersNotPlaying, Queue<Integer> availableTables) {
    List<Player> players = new ArrayList<>(playersNotPlaying);
    List<Game> games = new ArrayList<>();
    int i = 0;
    while ((i + 1 < playersNotPlaying.size()) && !availableTables.isEmpty()) {
        games.add(new Game(availableTables.poll(), LocalDateTime.now().withNano(0).toString(),
                players.get(i+1).getId(), players.get(i).getId(), active: true));
        i += 2;
    }
    return games;
}
```

Figur 4-88 Implementasjon av setupGame i AtTournamentStartAdaptedMonrad

```
@Override
protected List<Game> setupGames(Deque<Player> playersNotPlaying, Queue<Integer> availableTables) {
    List<Game> games = new ArrayList<>();
    while (!playersNotPlaying.isEmpty() && !availableTables.isEmpty()) {
        try {
            Player challenger = playersNotPlaying.pollFirst();
            Player opponent = findOpponent(challenger,
                    new ArrayList(playersNotPlaying));
            playersNotPlaying.remove(opponent);
            ColorDistribution colorDistribution = ColorDistributor.distribute(challenger, opponent);
            games.add(new Game(availableTables.poll(), LocalDateTime.now().withNano(0).toString(),
                    colorDistribution.getWhitePlayer().getId(), colorDistribution.getBlackPlayer().getId(),
                    active: true));
        } catch (NoSuchElementException e) {
            e.printStackTrace();
        }
    }
    return games;
}
```

Figur 4-89 Implementasjon av setupGame i AfterTournamentStartAdaptedMonrad

Hovedelementene i prosessen med å sette opp parti etter turneringsstart er å finne en motstander og fordele farger. Andre metoder enn «setupGames» er ansvarlig for å utføre disse delrutinene. Å finne motstander er det «AfterTournamentStartAdaptedMonrad» som står for, mens ansvaret for å fordele farge er gitt til en egen ChessmenColorDistributor-klasse. Dette er et eksempel på hvordan man har forsøkt å oppnå høy kohesjon (avsnitt 2.2.1). Både når man skal finne mostander og fordele farger er man avhengig av å ha tilgang til ovennevnte fargeregler. Disse har derfor blitt plassert i en egen ColorRules-klasse.

4.3.9 Push varsler

For at applikasjonen skal kunne sende push varsler må kommunikasjonen mellom frontend og backend krypteres. Først må backendapplikasjonen gi frontend en ECDSA P-256 public key. Denne nøkkelen blir sendt videre til pushManager i frontend slik at den vet at det faktisk er backend applikasjonen som sender en push melding. Tilbake får backendapplikasjonen et subscription-objekt. Denne inneholder en lenke og to P-256 nøkler. Lenken er endepunktet applikasjonen skal sende pushmeldinger til og nøklene blir brukt til å kryptere data som blir sendt. Subscription-objektet blir lagret i en HashMap der bruker-Id er nøkkel og objektet er verdi. Hver gang en bruker får tildelt et nytt parti vil subscription-objektet hentes ut fra listen og bli brukt til å sende en push-melding. Når en bruker forlater en turnering, vil vedkommende bli slettet fra HashMap-en. Når en trurnering avsluttes, blir alle brukere som var med i denne slettet fra HashMap-en. I applikasjonen blir det benyttet et eksternt bibliotek til sending av varsler.

4.3.10 Testing av backendapplikasjon

Det er laget Postman-tester for å teste applikasjonens REST-API. I tillegg har det blitt laget JUnit-tester for å teste metodene som brukes til å opprette nye parti. Testene finnes blant kildekoden til backendapplikasjonen, hvor Postman-testene er lagt ved i JSON-format i filen «sjakkarena.postman-collection.json».

Postman-tester

De fleste Postman-tester er avhengig av at det finnes noe testdata i databasen. Testdataen er lagt ved kildekoden til backendapplikasjonen i filen «2_sjakkarena.data.sql». De fleste av testene kjører uavhengig av hverandre, mens andre tester er avhengig av at ny informasjon har blitt lagt til på serveren – av andre tester – før de kjøres. Alle testene tester om server responder med riktig statuskode. I tillegg tester noen om riktig data sendes tilbake til klientene. Det testes for både riktig og feil bruk av API-et.

For å kjøre testene må eksempeldata i databasen og jwt-er brukt i testene oppdateres slik at de passer med nøkkelen som er brukt til å signere jwt-ene.

JUnit-tester

Figur 4-90 viser en JUnit-test som tester genereringen av nye parti i AfterTournamentStartAdaptedMonrad. Denne testen sjekker at regelen om at spillere kun kan møtes én gang i turneringen overholdes. Testen gir ni spillere til AfterTournamentStartAdaptedMonrad hvor bare to spillere – første og siste spiller i players-listen – ikke har spilt mot hverandre tidligere. Testen sjekker så at det kun er disse to spillerne som får tildelt et nytt parti.

```
@Test
public void provideNewGamesWhenOnlyTwoPlayerHasNotPlayedAgainstEachOther() {
    int numberOfPlayers = 9;
    onlyTwoPlayersHasNotPlayedAgainstEachOtherInit(numberOfPlayers);
    List<Game> games = adaptedMonrad.getNewGames(players, availableTables);
    assertEquals( expected: 1, games.size());

    Game game = games.get(0);
    assertTrue( condition: (game.getWhitePlayerId() == 1 && game.getBlackPlayerId() == numberOfPlayers) ||
            game.getWhitePlayerId() == numberOfPlayers && game.getBlackPlayerId() == 1);
}
```

Figur 4-90 Test av AfterTournamentStartAdaptedMonrad

5 DRØFTING

I denne delen vil de viktigste valgene en har tatt i løpet av prosjektarbeidet bli drøftet. Dette gjelder blant annet teknologivalg, datamodellering og kommunikasjon mellom klienter og servere. På slutten av drøftingsdelen vil mulige utvidelser og forbedringer av applikasjonen bli drøftet.

5.1 Valg av teknologi

Frontend

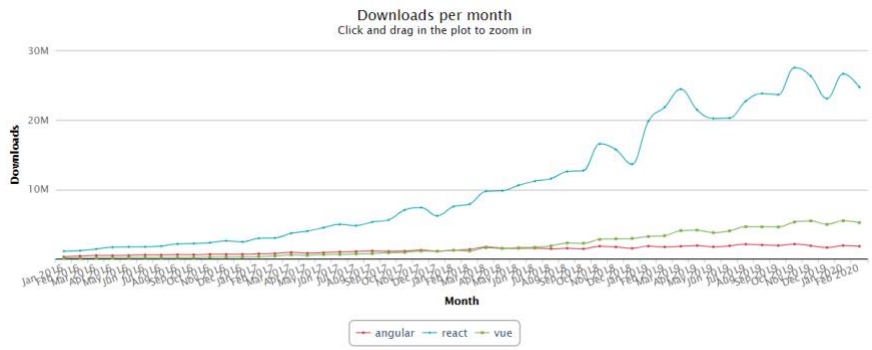
Det finnes flere JavaScript rammeverk som kunne vært brukt til å lage frontendapplikasjonen. I tillegg til VueJS, er Angular og React eksempler på slike rammeverk. Alle har åpen kildekode, men VueJS er det eneste som ikke har et stort selskap som styrer utviklingen.

Alle rammeverkene har hatt en stor fremgang de siste årene og antall brukere har økt. Ved å søke i «node package manager» (NPM) sin statistikk kan vi se denne økningen ved å sammenligne antall nedlastninger pr måned. Figur 5-1 viser denne statistikken. Som man kan se av figuren skiller React seg ut med ha de høyeste nedlastningstallene. Vue kommer på andreplass med litt høyere tall en Angular.

Hvor populært rammeverket er, ble tatt med i vurderingen når valg av rammeverk skulle tas. Populariteten til rammeverkene er viktig med tanke på jobbsøking for gruppemedlemmene etter at studiene er fullført.

Andre ting som ble gjort var å sammenligne prosjekter laget med de ulike rammeverkene. Ett av gruppens medlemmer hadde tidligere erfaring med React. Etter å ha studert Vue- og Angularprosjekter, kom gruppemedlemmet fram til at Vue ville fungere like godt som React. Samtidig mente gruppemedlemmet at det trolig ville være det enkleste å bruke.

Den siste overveiende faktoren var at kontaktpersonen til oppdragsgiver har erfaring med Vue. Dette gjorde det mulig å få innspill underveis. I tillegg vil det gjøre overleveringen og videreutviklingen av prosjektet enklere.



Figur 5-1 Månedlige nedlastinger for React, Angular og Vue. Kilde: (NPM-stats, 2020)

Backend

Java ble brukt som programmeringsspråk ettersom alle i gruppen hadde tidligere erfaringer med Java. WildFly og Dropwizard er andre rammeverk enn Spring Boot som kunne vært brukt til å lage Sjakkarena. Dropwizard brukes først og fremst til å lage REST-API. Ettersom vi besluttet å også bruke WebSockets, ble Dropwizard valgt bort.

Noen fordeler med Spring Boot er at deler av prosjektgruppen hadde brukt dette rammeverket tidligere. Ingen i gruppen hadde erfaring med tilsvarende Java-rammeverk. I tillegg har oppdragsgiver erfaring med Spring Boot.

5.2 Database/datamodell

Alle entitetstypene i datamodellen er på minimum tredje normalform. Dette ble ansett som tilstrekkelig. Utover å ha alle entitetstypene på tredje normalform, ble andre tiltak gjennomført for å redusere redundans. Det hadde f.eks. vært mulig å lagre informasjon som antall runder og poeng til en spiller eksplisitt i databasen. Entitetstypen Player kunne ha fått poeng- og runde-attributt for å lagre denne informasjonen. Men ettersom denne informasjonen ligger implisitt i Game-tabellen ble det besluttet å lage prosedyrer og funksjoner for å hente ut denne informasjonen. En ulempe med denne tilnærmingen er at det tar lengre tid å hente ut informasjon om spillere fra databasen. En fordel – i tillegg til redusert redundans – er at man slipper å sørge for at data i Player-tabellen samsvarer med data i Game-tabellen. For eksempel slipper man å oppdatere begge tabellene når et nytt resultat registreres.

Lagrede prosedyrer og funksjoner kunne vært unngått ved å skrive tilsvarende kode i Java. En av fordelen med å bruke lagrede prosedyrer og funksjoner fremfor java-kode er at java-koden blir ryddigere; I stedet for å gi databasespørringene som en tekststreng til et objekt, kan man skrive SQL-spørringen i et SQL-script. I tillegg får man lettere tilgang på SQL-spesifikke funksjonalitet, som å opprette midlertidige tabeller. En annen fordel er at lagrede prosedyrer reduserer risikoen for SQL-injeksjoner (avsnitt 2.8.2) ettersom at input data ikke blir tolket som kjørbar kode (Microsoft, 2017).

5.3 Kommunikasjon mellom server og klient

Tidlig i prosjektet ble det drøftet hvordan applikasjonen skal håndtere dynamisk data. Det ble diskutert mellom to ulike fremgangsmåter: klientene kan med regelmessige intervaller spørre server om endringer eller server kan sende endringer til brukerne ved behov. Med klienter som utfører spørringer kan det bli mer datatrafikk mellom klientene og server. Dette kan gjøre det tungt for serveren om der er mange klienter på samme tid. Grunnen til at dette kan legge unødvendig press på server er måten slike kommunikasjoner foregår. Fordelen med denne er at det muligens er lettere å implementere.

Alternativ to er å la serveren sender data til klientene over WebSockets. Dette reduserer press for serveren siden det allerede er en kobling mellom klient og server som venter på trafikk. WebSocket er til gjengjeld noe vanskeligere å implementere.

Vi startet med alternativ én hvor klientene gjør periodevis utspørring. Vi innså fort at utspørringer måtte ha korte intervaller for at applikasjonen skulle fungere som vi ønsket. Dette ville til gjengjeld ført til mer dataoverføring og stress på nettverket, skulle det bli mange deltakere. Derfor ble behovet for WebSocket stadig større.

5.4 Turneringsplanlegging

Det finnes mange ulike system for turneringsplanlegging som kunne vært brukt i Sjakkarena. Av disse finner man Round Robin (avsnitt 2.3.1) og Monrad (avsnitt 2.3.2). Fordelen med å bruke system som Round Robin er at man får en viss forutsigbarhet ved at alle parti i turneringen bestemmes før turneringsstart. En slipper da å ta hensyn til regler for partioppsett underveis i turneringen, slik som begrensninger i hvor mange parti

på rad en spiller kan spille med samme farge. En av ulempene med å bruke Round Robin, eller tilsvarende system, er at hvem som skal delta i turneringen må være avklart før turneringsstart. Oppdragsgiver ønsket at spillere skulle kunne melde seg inn i turneringen etter turneringsstart. Dermed kunne ikke Round Robin bli brukt.

Monrad har en rekke fordeler. Med Monrad trenger en ikke å ta hensyn til spilleres FIDE-rating. I stedet blir parti satt opp basert på spillernes prestasjoner i turneringen. Det er dog mulig å bruke FIDE-rating til å sette opp den første runden. Denne muligheten er valgt bort ettersom at de fleste som skal bruke applikasjonen antakeligvis ikke er FIDE-ratet. Det turneringssystemet som ble tatt i bruk i Sjakkarena er en tilpasset versjon av Monrad.

I utgangspunktet skal man ved bruk av Monrad tildele startnummer til spillerne ved turneringsstart. Ettersom at spillere kan melde seg på turneringen både før og etter turneringen har startet, blir startnummer tildelt når spilleren registreres i databasen. I motsetning til i Monrad, er ikke startnumrene i Sjakkarena alle heltall i en mengde {1, ..., n}, men er tilfeldige desimaltall som ligger i intervallet [0, 1). Det sørges for at to spillere i en turnering ikke får tildelt samme startnummer. Man ønsker at tidspunktet en spiller melder seg inn i turneringen skal ha lite å si for hvordan spilleren blir sortert senere i turneringen. Å trekke et tilfeldige desimaltall som startnummer anses å være en bedre løsning enn å skulle sørge for at mengden av startnumre hele tiden består av etterfølgende heltall.

Når nye parti skal tildeles i Monrad skal spillerne sorteres etter poengsum og startnummer. I stedet for å sorteres etter poengsum, blir spillerne i Sjakkarena sortert etter gjennomsnittlig poengsum. Dette gjøres for å ta hensyn til at spillere kan ha spilt et ulikt antall parti når det nye partiet skal tildeles.

Monrad har en rekke regler for å håndtere situasjoner hvor antall spillere i turneringen er et oddetall. Når partiene i en runde trekkes samtidig vil man kunne oppnå at en spiller ikke blir tildelt et parti i runden (walk-over). Blant annet bestemmer disse reglene at ingen skal stå over en runde mer enn én gang. Oppdragsgiver ønsket at spillere skulle få tildelt et nytt parti så snart som mulig etter å ha spilt ferdig sitt parti. En konsekvens av dette er at reglene for walk-over ikke kan anvendes. Når rundene ikke trekkes samtidig vet man ikke hvem som skal delta i runden. Dette innebærer at man ikke vet om antallet spillere i turneringen er et oddetall eller et partall. Man kan derfor ikke peke ut noen på forhånd til å stå over runden.

5.5 Brukertester

Ett av de beste hjelpemiddelene for å lage en brukervennlig applikasjon er å gjennomføre brukertester. Gjennom brukertester kan man studere hvordan brukerne benytter seg av applikasjonen og få konkrete tilbakemeldinger på hva som fungerer og ikke fungerer. Om ikke applikasjonen har et godt designet brukergrensesnitt kan brukerne oppleve irritasjon og forvirring, og dermed ikke ønske å bruke den.

Det var aktuelt å kjøre en runde med brukertesting på sjakklubben i Ålesund. Kontaktpersonen til oppdragsgiver har også hatt dialog med en av eierne i sjakkpuben «The Good Knight» i Oslo, som sa seg villig til å gjennomføre en brukertest. Et annet alternativ var å arrangere brukertesting med studenter. Alle disse mulighetene ville vært viktig for en skikkelig gjennomgang av brukergrensesnittet og for å finne feil som ikke har blitt oppdaget. Ingen av disse brukertestene kunne bli gjennomført på grunn av situasjonen med Covid-19.

5.6 Brukergrensesnitt

Applikasjonen ble laget for fremvisning med projektor på vertsidene og mobil på spillersidene. Dette fordi vi tenker at applikasjonen vil bli brukt med smarttelefoner hos spillere, og verten vil muligens ønske vise poengstillingen mellom alle spillerne, enten for seg selv eller synlig for spillerne. Om dette er beste måten å gå frem på er uvisst, ettersom vi ikke har utført brukertester diskutert i avsnitt 5.5. Totalt sett er vi fornøyde med utformingen av applikasjonen og dens konsistente design. Vi føler oss sikre på at applikasjonen er lett å forstå og lite distraherende for spillere. Men vi føler likevel at en brukertest hadde vært både læringsrikt og nyttig, for vår egen del og for eventuell videreutvikling av applikasjonen. Det er vanskelig å utforme sider for mange ulike skjermstørrelser og uventede problemer kan oppstå. Disse hadde muligens blitt oppdaget ved en slik test.

5.7 Videre arbeid

Det kan være ønskelig å utvikle applikasjonen videre. I så fall må det avklares hvordan de nødvendige ressursene til videreutviklingen skal anskaffes.

5.7.1 Insentiv for drift, vedlikehold og videreutvikling

Dersom applikasjonen skal kunne være tilgjengelig for bruk, må man ha en måte å skaffe ressurser til drift, vedlikehold og videreutvikling av applikasjonen. Underveis i prosjektarbeidet har det blitt drøftet ulike metoder for å skaffe disse ressursene.

En mulighet er å gjøre kildekoden tilgjengelig for alle, slik at applikasjonen kan utvikles videre som et «open source»-prosjekt. Fordelen med denne tilnærmingen er at man ikke har behov for penger for å drive med vedlikehold og videreutvikling. Dermed øker muligheten for å la applikasjonen være gratis å bruke. Det vanskelige med å la applikasjonen bli et «open source»-prosjekt er å generere nok interesse til at noen ønsker å bidra. Trolig vil en være avhengig av de som både har interesse for sjakk og applikasjonsutvikling.

Dersom applikasjonen klarer å genere inntekt vil det bli enklere å skaffe de nødvendige ressursene. En mulighet for å skaffe penger er å ta betalt for å bruke applikasjonen. Om dette gjøres, risikerer man at færre personer ønsker å ta applikasjonen i bruk. Dette er selvfølgelig ikke ønskelig.

En annen mulighet er å selge reklameplasser i applikasjonen. Dette vil trolig påvirke antall brukere mindre enn dersom man tar betalt for bruk. Ulempen er at det kreves en del brukere for at inntektene skal nå et nivå som gjør det mulig drifte, vedlikeholde og videreutvikle applikasjonen.

Å åpne for at brukere kan donere penger til drift av applikasjonen, er også en mulighet. Trolig er det en liten andel av brukerne som er villige til å donere penger. Dermed må en ha et stort antall brukere for å få inn nok.

Det mest realistiske er å ha applikasjonen som et «open source»-prosjekt. Denne løsningen stiller minst krav til antall brukere av applikasjonen og deres betalingsvilje.

5.7.2 Mulige utvidelser

I løpet av prosjektarbeidet har det blitt vurdert mange mulige utvidelser av applikasjonen. Noen av disse har blitt implementert, mens andre kan bidra til å øke applikasjonens kvalitet dersom de blir implementert. En av de mulige utvidelsene er å la spillere spille sjakkparti i applikasjonen dersom de ønsker, eller ikke har mulighet til, å spille på fysiske brett. Dette kan gjøres ved å legge til et virtuelt brett slik som lichess.org og chess.com tilbyr. Ved å legge til denne funksjonaliteten vil applikasjonen likne mer på tjenestene de nevnte sjakknettsidene tilbyr. Likevel vil applikasjonen skille seg ut ved å kunne legge til rette for mer sosiale turneringer spilt på fysiske brett.

En annen mulig utvidelse er å legge til rette for at spillerne kan registrere trekk underveis i partier. Dette er noe som gjøres i seriøse langsjakkturneringer og er trolig noe en del kunne ønsket seg å ha muligheten til. Det bør også gjøres mulig å eksportere trekkene som skrives inn til ulike formater, f.eks. Portable Game Notation (PGN). Enkelte applikasjoner, slik som lichess.org, tilbyr muligheten til å få analysert parti som er beskrevet med PGN. For en del vil det derfor være ønskelig å kunne bruke trekkinformasjonen andre plasser.

For å øke nytten av å kunne registrere trekk i applikasjonen, kan man legge til brukere. Dersom en spiller har muligheten til å logge inn på applikasjonen med brukernavn og passord fremfor å logge seg inn med «Kahoot!»-metoden, vil det være mulig å vise informasjon om brukerens tidligere partier og turneringer. Slik informasjon kan bl.a. være trekkene som spilleren og motspilleren gjorde i partiet.

Å bruke FIDE-rating i applikasjonen kan også være en mulig utvidelse. Spillerne kan for eksempel oppgi sin FIDE-rating når de melder seg på turneringen eller så kan FIDE-ratingen hentes fra FIDEs nettsider. Ratingen kan brukes til å finne jevnere motstand til spillere. I tillegg kan ratingen brukes til å gi en vurdering av prestasjonen til hver enkelt spiller ved å sammenlikne nivået på dem de har spilt mot. Dette gjør det mulig å endre reglene for poengtildeling. Spillere kan f.eks. få poeng, ikke bare etter resultatet på partiet, men også etter hvor stor motstand de møtte.

Det finnes også andre endringer som kan gjøres i forbindelse med poengtildeling. En kan gi ekstrapoeng til de som spiller mange parti på rad. I tillegg kan ekstrapoeng tildeles de som har spilt få partier, slik at de som har motstandere som bruker lenger tid på sine parti ikke straffes for dette.

Det er også mulig å implementere flere ulike turneringsplanleggere, slik at applikasjonen kan brukes til å organisere ulike typer turneringer. F.eks. alle møter alle eller at applikasjonen setter spillere blindt mot hverandre uten å vurdere opptjent poeng eller lignende. Dette gjør at applikasjonen kan brukes til flere sammenhenger og derfor gjøre applikasjonen mer konkurransedyktig for et fremtidig marked.

5.7.3 Sikkerhet

SQL-injeksjon
Som beskrevet i 2.8.2 er SQL-injeksjoner en måte å angripe databasen gjennom brukerinput. En metode gruppen har benyttet for å sikre applikasjonen er at der applikasjonen forventer tall, så kan kun tall benyttes. Det finnes også noe innebygget funksjonalitet i SpringBoot for å sikre applikasjonen mot slike angrep. Eksempelvis så aksepterer ikke applikasjonen «;» ved bruk av stivariabler (se Figur 5-2). Så er ikke dette tilstrekkelig. Ved utvidelse av applikasjonen burde SQL-spørringer skrives om til å benytte seg av «Prepared statements». Denne metoden er effektiv mot slike angrep fordi

den sender spørringen til databasen og kompilerer den med midlertidige variabler, deretter sender den verdiene med en annen protokoll og kjører spørringen med de gitte verdiene.

```
org.springframework.security.web.firewall.RequestRejectedException: The request was rejected because the URL contained a potentially malicious String ";"
    at org.springframework.security.web.firewall.StrictHttpFirewall.rejectedBlacklistedUrls(StrictHttpFirewall.java:369) ~[spring-security-web-5.2.1.RELEASE.jar:5.2.1.RELEASE]
    at org.springframework.security.web.firewall.StrictHttpFirewall.getFirewalledRequest(StrictHttpFirewall.java:336) ~[spring-security-web-5.2.1.RELEASE.jar:5.2.1.RELEASE]
    at org.springframework.security.web.FilterChainProxy.doFilterInternal(FilterChainProxy.java:194) ~[spring-security-web-5.2.1.RELEASE.jar:5.2.1.RELEASE]
    at org.springframework.security.web.FilterChainProxy.doFilter(FilterChainProxy.java:178) ~[spring-security-web-5.2.1.RELEASE.jar:5.2.1.RELEASE] <2 internal calls>
```

Figur 5-2 Innebygget forebygging mot SQL-injeksjon

Push varsler – Backend

Pr nå blir offentlig og privat nøkkel hentet fra en nettside[5] og lagt inn i applikasjonen ved hjelp av miljøvariabler. Det vil si at den som skal sette opp applikasjonen må kopiere privat og offentlig nøkkel fra nettstedet og manuelt legge de inn. Dette er ikke noe applikasjonen gjør automatisk. Det er da heller ønskelig at applikasjonen selv oppretter nøkkelpar og skriver dette til en kryptert fil. Men det ble rett og slett for vanskelig og tidkrevende å få nøklene til et format som kunne brukes sammen med push-varsel-biblioteket som ble benyttet.

Lagring av informasjon

På grunn av tidsmangel har det ikke blitt laget til en skikkelig løsning for å lagre informasjon som ikke bør deles med andre, slik som passord og krypteringsnøkler. En mulig løsning på dette problemet er å benytte seg av hvelv, slik som Hashicorps Vault (Hashicorp, u.d.). I hvelvet kan passord og krypteringsnøkler bli lagret. Den største utfordringen med å bruke hvelv, er å sette dette opp på en sikker måte. Det var dette som ville ta for lang tid til at å rekke å implementere denne løsningen. En «proof of conept»-løsning for interaksjon med Vault er beskrevet i avsnitt 4.3.7.

5.7.4 Cypress

Det ble ikke tid til å lage tester for alle sidene i applikasjonen så dette kan arbeides videre med. Det er også et par av sidene det kunne vært mulig og kanskje ønskelig å skrive flere tester til. Å involvere backend er også en ting som kunne vært gjort mere. Det er dog både ulemper og fordeler med å involvere backend. Eksempel på en fordel er at man alltid vil teste mot det applikasjonen kommer til å sende. En måte for å involvere denne mere er å benytte Cypress sin «exec()» metode som gjør det mulig å kjøre kommandoer. Denne kunne da ha blitt brukt til å legge til og slette data fra databasen før hver test.

5 https://web-push-codelab.glitch.me/

6 KONKLUSJON

Resultatet av prosjektarbeidet er en fungerende applikasjon for organisering av sjakkturneringer. Applikasjonen oppfyller de kravene som oppdragsgiver stilte i begynnelsen av prosjektet. I tillegg har det, etter avtale med oppdragsgiver og veileder, blitt lagt til flere funksjoner underveis i prosjektarbeidet.

I applikasjonen kan turneringsverter opprette turneringer, se oversikt over spillere og pågående parti. I tillegg kan verter administrere turneringer. Dette innebærer blant annet å starte og avslutte turneringer, samt sette de på pause. Turneringsverter har også mulighet til å bestemme hvilke spillere som får delta i turneringen og, ved uenighet mellom spillere, bestemme resultatet av spilte parti.

Spillere kan bl.a. melde seg på og av turneringer og få tildelt nye parti med informasjon om motstander, brikkefarge og hvilket brett de skal spille på. De kan også ta pause fra turneringen og bruke sjakkur for å ta tiden under parti. Dersom spillerne ønsker det, kan de få pushvarsler når nye parti er tildelt.

Det har også blitt lagt til en omside og en hjelpe-side for å gjøre det enklere å ta i bruk applikasjonen.

Gestaltprinsippene og Donald Normans designregler har blitt brukt til å sikre et godt brukergrensesnitt. Gestaltprinsippene om nærhet og likhet har i særlig stor grad blitt brukt. I tillegg har Normans prinsipper om viktigheten av å gi tilbakemelding til brukerne og å være konsekvent i utformingen av applikasjon hatt stor innflytelse på designet. Disse teoriene har vært til stor hjelp, ettersom gruppemedlemmene hadde lite erfaring med interaksjonsdesign før arbeidet med bacheloroppgaven begynte.

Selv om eksisterende system for turneringsoppsett, slik som Monrad og Round-Robin, ikke kunne brukes, har disse gitt inspirasjon til systemet brukt i Sjakkarena. En har i hovedsak tatt utgangspunkt i Monrad og tilpasset denne til oppdragsgivers ønsker. Man har blant annet forsøkt å øke aktiviteten i applikasjonen ved å tillate at spillere kan melde seg på så lenge turneringen pågår. I tillegg vil spillere få tildelt nye parti så fort en egnet motstander er ledig.

En annen viktig side av applikasjon er hvordan man har valgt å sende informasjon mellom backend- og frontendapplikasjonen. Der hvor klienter ikke vet når ny informasjon er tilgjengelig, har WebSockets blitt brukt til å sende denne informasjonen. På den måten har man unngått at klienter stadig må spørre server om det er kommet ny informasjon. I de tilfeller hvor klienter selv vet at det er ny informasjon tilgjengelig, eller vil sende ny informasjon, gjør de restkall over HTTP.

Prosjektarbeidet har medført mye læring. Av teori er det først og fremst prinsipper og regler for interaksjonsdesign og for turneringsplanlegging som har vært nytt for oss. Mye av annen teori som bacheloroppgaven bygger på har vært lært i emner vi har hatt. Det er implementasjonen av slik teori som i stor grad har vært nytt. Innen programmering har vi også tilegnet oss ferdigheter i bruk av noen rammeverk og verktøy, slik som Cypress, Vue og Spring Boot.

Med tanke på prosjektarbeid har vi fått god trening i å jobbe med Scrum. Vi har blant annet brukt konsepter som sprint, med tilhørende møter, og holdt hverandre oppdatert med daglige scrum-møter.

Totalt sett er vi selv godt fornøyde med resultatet vi fikk i denne oppgaven og tror at denne applikasjonen kan være til hjelp for mange brukere.

7 REFERANSER

Axios, u.d. *github.com.* [Internett]
Available at: https://github.com/axios/axios
[Funnet april 8 2020].

Barnes, D. J. & Kölling, M., 2016. *Objects First with Java: A Practical Introduction Using BlueJ.* 6 red. s.l.:Pearson Education Limited.

Bartnes, M., 2017. *snl.no.* [Internett]
Available at: https://snl.no/HTTPS
[Funnet 25 februar 2020].

Bostrøm, E., 1999. *Datamodellering - praksis og teori med oppgavesamling.* 1 red. s.l.:s.n.

Bratbergsengen, K., 2017. *snl.no.* [Internett]
Available at: https://snl.no/relasjonsmodellen
[Funnet 1 april 2020].

Bratbergsengen, K., 2019. *snl.no.* [Internett]
Available at: https://snl.no/attributt_-_IT
[Funnet 1 april 2020].

Bray, T., 2017. *ietf.org.* [Internett]
Available at: https://tools.ietf.org/html/rfc8259
[Funnet 19 februar 2020].

Cypres, u.d. *cypress.io.* [Internett]
Available at: https://www.cypress.io/how-it-works/
[Funnet 2 april 2020].

Docker, u.d. *docker.com.* [Internett]
Available at: https://www.docker.com/resources/what-container
[Funnet 4 mars 2020].

EsLint, u.d. *eslint.org.* [Internett]
Available at: https://eslint.org/
[Funnet 8 april 2020].

Fette, I. & Melnikov, A., 2011. *ietf.org.* [Internett]
Available at: https://tools.ietf.org/html/rfc6455
[Funnet 26 mars 2020].

Hashicorp, u.d. *hashicorp.com.* [Internett]
Available at: https://www.hashicorp.com/products/vault
[Funnet 10 mai 2020].

Jest.js, u.d. *jestjs.io.* [Internett]
Available at: https://jestjs.io/
[Funnet 2 april 2020].

Johnson, J., 2013. *Designing with the Mind in Mind.* s.l.:ELSEVIER SCIENCE.

Jones, M., Bradley, J. & Sakimura, N., 2015. *ietf.org.* [Internett]
Available at: https://tools.ietf.org/html/rfc7519
[Funnet 20 februar 2020].

Kristoffersen, B., 2016. *Databasesystemer.* 4. utgave red. s.l.:Universitetsforlaget.

Kurose, J. F. & Ross, K. W., 2016. *Computer Networking: A Top-Down Approach.* 7. utgave red. s.l.:Pearson Education Limited.

Lujan, P., 2007. *Scheduling Guide.* [Internett]
Available at: https://www.ocf.berkeley.edu/~quizbowl/schedules.html
[Funnet 25 februar 2020].

Microsoft, 2012. *docs.microsoft.com.* [Internett]
Available at: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10)
[Funnet 31 mars 2020].

Microsoft, 2017. *docs.microsoft.com.* [Internett]
Available at: https://docs.microsoft.com/en-us/sql/relational-databases/stored-procedures/stored-procedures-database-engine?view=sql-server-ver15
[Funnet 29 april 2020].

Mozilla, u.d. *developer.mozilla.org.* [Internett]
Available at: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API
[Funnet 10 mai 2020].

Norges Sjakkforbund, 2020. *sjakk.no.* [Internett]
Available at: https://www.sjakk.no/2020/01/20/medlemstall-for-2019/
[Funnet 22 januar 2020].

Norges Sjakkforbund, u.d. *sjakk.no.* [Internett]
Available at: https://www.sjakk.no/kunnskap/systemer/norges-sjakkforbunds-monradsystem/
[Funnet 25 februar 2020].

Norman, D. A., 1983. Design Rules Based On Analyses of Human Error. *Communications of the ACM*, April, pp. 254-258.

NPM, u.d. *npmjs.com.* [Internett]
Available at: https://docs.npmjs.com/about-npm/
[Funnet 20 mars 2020].

NPM-stats, 2020. *npm-stats.com.* [Internett]
Available at: https://npm-stat.com/charts.html?package=react&package=vue&package=angular&from=2016-01-01&to=2020-02-26
[Funnet 2 februar 2020].

Oracle, u.d (b). *mysql.com.* [Internett]
Available at: https://www.mysql.com/products/workbench/
[Funnet 8 februar 2020].

Oracle, u.d. (a). *mysql.com.* [Internett]
Available at: https://www.mysql.com/products/community/
[Funnet 2 april 2020].

OWASP, u.d. *owasp.org.* [Internett]
Available at: https://owasp.org/www-community/attacks/SQL_Injection
[Funnet 27 mars 2020].

OWASP, u.d.(a). [Internett]
Available at: https://owasp.org/www-community/attacks/xss/
[Funnet 27 mars 2020].

Ponniah, P., 2003. *Database Design and Development: An Essential Guide for IT professionals.* 1. utgave red. s.l.:John Wiley Sons Inc.

Spring, u.d. *spring.io.* [Internett]
Available at: https://spring.io/projects/spring-boot
[Funnet 19 mars 2020].

Stomp, u.d. *stomp.github.io.* [Internett]
Available at: http://stomp.github.io/stomp-specification-1.2.html#Background
[Funnet 26 mars 2020].

Sumathi, S. & Esakkirajan, S., 2007. *Fundamentals of Relational Database Management Systems.* s.l.:Springer Berlin Heidelberg.

Sutherland, J. & Schwaber, K., 2017. *scrumguides.org.* [Internett]
Available at: https://www.scrumguides.org/scrum-guide.html
[Funnet 3 april 2020].

Svartdal, F., 2019. *snl.no.* [Internett]
Available at: https://snl.no/gestaltpsykologi
[Funnet 1 april 2020].

Travis, u.d. *travis-ci.com.* [Internett]
Available at: https://docs.travis-ci.com/user/for-beginners/
[Funnet 10 mars 2020].

Trætteberg, H., 2020. *ntnu.no.* [Internett]
Available at: https://www.ntnu.no/wiki/display/tdt4100/Faginnhold
[Funnet 26 mars 2020].

vuetify, u.d. *vutifyjs.com.* [Internett]
Available at: https://vuetifyjs.com/en/introduction/why-vuetify

Vue, u.d (b). *vuejs.* [Internett]
Available at: https://router.vuejs.org/
[Funnet april 8 2020].

Vue, u.d. (a). *vuejs.org.* [Internett]
Available at: https://vuejs.org/v2/guide/
[Funnet 31 mars 2020].

Vue, u.d. (c). *vuejs.* [Internett]
Available at: https://vuejs.org/v2/guide/mixins.html
[Funnet 18 april 2020].

Vue, u.d. (d). [Internett]
Available at: https://vuejs.org/v2/guide/instance.html#Instance-Lifecycle-Hooks
[Funnet 22 april 2020].

Vue, u.d. (e). [Internett]
Available at: https://vuejs.org/v2/guide/security.html
[Funnet 4 mai 2020].

VEDLEGG

Vedlegg 1	Kravspesifikasjon
Vedlegg 2	Logg
Vedlegg 3	Framdriftsrapport
Vedlegg 4	Møtereferat
Vedlegg 5	Forprosjektrapport
Vedlegg 6	Proof of concept for interaksjon med Vault
Vedlegg 7	Kildekode for backendapplikasjon, vedlagt i zip-fil. Ligger også på GitHub: https://github.com/Severinzz/Sjakk-Arena-backend
Vedlegg 8	Kildekode for frontendapplikasjone, vedlagt i zip-fil. Ligger også på GitHub: https://github.com/Severinzz/Sjakk-Arena-Frontend

Applikasjonen vil i en periode etter levering av oppgaven kunne bli testet ut på følgende nettside: https://sjakk-arena.herokuapp.com/

Vedlegg 1 Kravspesifikasjon

Kravspesifikasjon

Målutgivelse	1.0
Epic	
Dokumentstatus	UTKAST
Dokumenteier	
Designer	
Utviklere	
Kvalitetssikring	

Mål

- Lage en webapplikasjon for organisering av uformelle sjakkturneringer

Bakgrunn og strategisk passform

Fra oppgavebeskrivelsen: "5-30 personer møtes enten på en Pub som skal ha sjakk-kveld eller på en klubb. Noen kjenner hverandre, andre kjenner ikke hverandre.

De kommer og går til ulike tider. Hvem skal spille sammen? Inspirert av f.eks. Lichess Arena så kan man sette opp en tjeneste som kan brukes fra nettbrett eller mobiler for å melde seg inn og ut av turneringen.

Så vises hvem som skal møtes, resultatliste på en stor TV i lokalet. Etterhvert som noen blir ferdige så starter nye parti frem til man har meldt seg ut eller er ferdig."

Antakelser

- Deltakerne er først og fremst interessert i å bruke applikasjonen på nettbrett og mobiltelefoner

Funksjonelle krav

#	Tittel	Brukerhistorie	Viktighet	Notater
1	Oprette turnering	Som en turneringsvert vil jeg kunne opprette turneringer	MÅ HA	
2	Konfigurere turnering	Som en turneringsvert vil jeg kunne konfigurere turneringer.	MÅ HA	Det som kan bestemmes i prioritert rekkefølge: 1. Start og sluttid 2. Antall bord 3. Tid mellom parti og maks antall runder
3	UUID	Som en turneringsvert vil jeg motta en UUID per epost som vil jeg meg tilgang til å konfigurere turneringen.	BURDE HA	
4	Melde seg inn i turnering	Som en deltaker vil jeg kunne melde meg inn i en turnering, selv om jeg kommer sent.	MÅ HA	Pin tastes inn (kahootliknende system)
5	Registrere resultat	Som en deltaker vil jeg kunne registrere resultatet fra mine spill	MÅ HA	Seier, remis, tap, walk over, avlyst for hvert parti
6	Se resultatoversikt	Som en bruker vil jeg kunne se hvordan ser står til sammenlignet med andre spillere.	BURDE HA	Leaderboard
7	Melde seg ut av turnering	Som en deltaker vil jeg kunne melde meg ut av en turnering	BURDE HA	
8	Avslutte turnering	Som en turneringsvert vil jeg kunne avslutte min turnering.	MÅ HA	Manuell avslutning

9	Valg av motspiller	Som en deltaker vil jeg at applikasjonen skal velge ut en motspiller og farge for meg.	MÅ HA	• Gi deltakere bordnr. • Bestemme hvem som skal spille med hvite brikker og hvem som skal spille med sorte brikker • Valg av motspiller basert på egne poeng og tidligere motspilleres poeng
10	Push-varsel	Som en deltaker vil jeg ha push-varsel når en ny motstander er funnet.	BURDE HA	Varsel for nytt parti
11	Kaste ut deltakere	Som en turneringsvert vil jeg kunne kaste ut deltakere	KAN HA	Noen kan finne på å gå uten å melde seg ut
12	Begrunnelse	Som en bruker vil jeg ha begrunnelse ved utkastelese fra turneringen	KAN HA	
13	Spillerpause	Som en bruker vil jeg kunne ta en pause fra turneringen/stå over en runde uten at det påvirker resultatet.	KAN HA	
14	Turneringspause	Som en turneringsvert vil jeg ha mulighet til å sette turneringen på pause.	KAN HA	
15	Endre resultat	Som en turneringsvert vil jeg ha mulighet til å endre resultatet på ferdig spillte kamper	BURDE HA	Folk kan legge inn feil resultat ved uhell. Kan også vær regelbrudd der endring av resultat er relevant
16	Hjelpeside	Som en bruker vil jeg ha en innføring i bruk av applikasjonen /hjelpeside.	BURDE HA	sjakkordbok, regler, applikasjons assistanse.
17	Klokke	Som en deltaker vil jeg ha en sjakk klokke i applikasjonen	KAN HA	Egen fane hvor spillere kan bruke en smarttelefon som sjakklokke.

Ikke-funksjonelle krav

#	Tittel	Brukerhistorie	Viktighet	Notater
1	Intuitivt design	Som en bruker ønsker jeg at designet er intuitivt og at applikasjonen dermed er enkel å bruke	BURDE HA	Minst mulig tid på å forstå appen, mest mulig sjakk
2	Responsivt design	Som en bruker ønsker jeg å kunne bruke applikasjonen på enheter som en kan forvente at en slik webapplikasjon fungerer på	MÅ HA	TV visning + mobil/pad (ulike skjermstørrelser)
3	Moderne design	Sidene må være oversiktlig, fornuftig bruk av tekst og farger	BURDE HA	
4	Universell utforming	Siden må være laget slik at alle kan bruke den, uansett om man har en synshemming eller lignende.	BURDE HA	

Vedlegg 2 Log

IB303312 Bacheloroppgave
LOG
Utført arbeid i perioden

Navn på studenter:
Stud1: Andreas Moe Hatlø Stud2: Severin Aas Eliassen

Stud3: Jahn-Willy Nøstdahl
Navn på bedrift/organisasjon: Aalesunds Schaklag
Navn på veileder ved bedrift/organisasjon: Arne Unneland
Aktivitetsplan

Uke	Dato	Gjennomført arbeid/Tema/aktivitet	Stud1 Timer	Stud2 Timer	Stud3 Timer
2	08.01	Forprosjektrapport	3	3	3
2	10.01	Forprosjektrapport	5	5	5
3	13.01	Forprosjektrapport, brukerhistorier	5	5,75	5
3	14.01	Forprosjektrapport, UML diagram	5,34	6	4
3	15.01	Forprosjektrapport	5,25	3,83	5
3	16.01	Forprosjektrapport, wireframes, UML-diagram	7	8	7
3	18.01	Forprosjektrapport, databasedesign		1,15	
4	20.01	Wireframes, Forprosjektrapport, Designskisser	3,5	4,5	2
4	21.01	Forprosjektrapport, bachelorrapport, designskisser	7	8	7
Sum timer			41,09	45,23	38

Plan for neste 14-dager, tema (aktivitetsplan)

4-6		Informasjonsinnhenting
4-6		Fortsette på bachelorrapport – innledning og teoridel
4-6	Før 29.02	Lage presentasjon til systemfaget
4-6	Før 01.02	Fullføre forprosjektrapporten
4-6		Ferdigstille designskisser

IB303312 Bacheloroppgave
LOG
Utført arbeid i perioden

Navn på studenter:
Stud1: Andreas Moe Hatlø Stud2: Severin Aas Eliassen

Stud3: Jahn-Willy Nøstdahl
Navn på bedrift/organisasjon: Aalesunds Schaklag
Navn på veileder ved bedrift/organisasjon: Arne Unneland
Aktivitetsplan

Uke	Dato	Gjennomført arbeid/Tema/aktivitet	Stud1 Timer	Stud2 Timer	Stud3 Timer
4	22.01	Bachelorrapport innledning, teori, forprosjektrapport, designskisse, informasjonsinnhenting frontendteknologier	5	5,83	5,5
4	23.01	Forprosjektrapport, bachelorrapport, sette seg inn i Bootstrap, møte med oppdragsgiver	7,6	8,5	7,75
4	25.01	Sette seg inn i Vue			1
5	27.01	Bachelorrapport, forprosjektrapport, pp-presentasjon, informasjonsinnhenting frontend, skrive møtereferat	5,25	6,25	5,25
5	28.01	Design frontend (Vue), presentasjon av oppgave	7	8	6,25
5	29.01	Design frontend (Vue), forprosjektrapport og bachelorrapport	5	2,25	3
5	30.01	Forprosjektrapport, bachelorrapport, design frontend (Vue)	7	8	7
6	03.01	Design frontend	5	6	4,5
6	04.01	Design frontend	7	8	4,25
Sum timer			48,85	52,83	44,5

Plan for neste uke, tema (aktivitetsplan)

6-8		Legge til funksjonalitet i applikasjonen for å opprette og konfigurere turneringer
6-8		Legge til funksjonalitet i applikasjonen for å melde seg inn i turneringer
6-8		Legge til funksjonalitet i applikasjonen for å gi en unik id per epost til turneringsvert
6-8		Fortsette på teoridel i bachelorrapport

IB303312 Bacheloroppgave
LOG
Utført arbeid i perioden

Navn på studenter:
Stud1: Andreas Moe Hatlø Stud2: Severin Aas Eliassen
Stud3: Jahn-Willy Aasen Nøstdahl
Navn på bedrift/organisasjon: Aalesunds Schaklag
Navn på veileder ved bedrift/organisasjon: Arne Unneland

Aktivitetsplan

Uke	Dato	Gjennomført arbeid/Tema/aktivitet	Stud1 Timer	Stud2 Timer	Stud3 Timer
6	05.02	Programmering backend + kontorarbeid	5	6	4,25
6	06.02	Programmering: Spillerlobby, spillerregistrering, Refaktorering, cypress, testing	7	8	5,25
6	07.02	Spillerlobby, refaktorering			5
6	08.02	Bachelorrapport, testing		1,83	
7	10.02	Programmering: jwt, testing, design feedback	4	4,83	3,8
7	11.02	Programmering: generering av turneringsid, brukerregistrering, testing, veiledermøte, frontend API, vuex research, spillerlobby refaktorering.	7	8	6,85
7	12.02	Programmering: autorisering med JWT, testing, frontend API for turneringsoppretting,	5	6,16	4,25
7	13.02	Bachelor rapport, spring boot-læring, Programmering: autorisering med JWT, henting av brukernavn for turneringer, frontend: melde seg inn i turnering og fortsette turnering med turneringsID	7	8	5,5
7	14.02	Lære om spring boot			1,83
8	17.01	Frontend: ny spiller til server, lobby API,	5	5,5	4
8	18.01	Backend spillerikon, hente turneringsinformasjon, slette bruker, testing. Bachelorrapportskriving, lobby API, bugfixing, dokumentasjon av kode	7	9	7
Sum timer			47	57,32	47,70

Plan for neste uke, tema (aktivitetsplan)

8-10		Skrive på bachelorrapport
8-10		Lage til funksjonalitet for å se resultater
8-10		Lage til funksjonalitet for å registrere resultater
8-10		Lage til funksjonalitet for å melde seg ut av turnering

8-10		Om det blir tid: Containerisering med docker og CI med travis.

IB303312 Bacheloroppgave
LOG
Utført arbeid i perioden

Navn på studenter:
Stud1: Andreas Moe Hatlø Stud2: Severin Aas Eliassen

Stud3: Jahn-Willy Aasen Nøstdahl
Navn på bedrift/organisasjon: Aalesunds Schaklag
Navn på veileder ved bedrift/organisasjon: Arne Unneland
Aktivitetsplan

Uke	Dato	Gjennomført arbeid/Tema/aktivitet	Stud1 Timer	Stud2 Timer	Stud3 Timer
8	19.02	Bugfiksing, bacheloroppgave teoridel. Sette opp server. Funksjon for å forlate turnering.	5	6,25	4,25
8	20.02	Backend: Forlat turnering			0,75
8	22.02	Backend: Forlat turnering			0,75
9	24.02	Backend: resultatoversikt, registrere resultat Frontend: Turnering startet for vert, «dato og tid» -velger, Forlat turnering	5	7	5,5
9	25.02	Bachelorrapport, «dato og tid»-velger	2	2,16	
9	26.02	Frontend: registrere resultat, «dato og tid»- velger, bugs, oppdatere server. Møte.	5	6,33	5
9	27.02	Bachelorrapport, turnering startet vert, Veildermøte, frontend refaktorering av API	8	6,5	5,5
9	28.02	Bachelorrapport, Frontend: resultatliste, turnering startet for vert	2	4	3
10	02.03	Bachelorrapport, Frontend: refaktorering av playerlobby, resultatliste, docker	4	5,2	4,5
10	03.03	Bachelorrapport, Frontend: registrere spillere, resultatliste	7	8	7
Sum timer			50	45,44	36,25

Plan for neste uke, tema (aktivitetsplan)

10-12		Lage til funksjonalitet for å se resultatoversikt, frontend
10-12		Lage til funksjonalitet for å ta pause, frontend
10-12		Lage til funksjonalitet for valg av motspiller
10-12		Lage til websocket
10-12		Redesign av turneringsinformasjon i spillerlobby

IB303312 Bacheloroppgave
LOG
Utført arbeid i perioden

Navn på studenter:
Stud1: Andreas Moe Hatlø Stud2: Severin Aas Eliassen
Stud3: Jahn-Willy Aasen Nøstdahl
Navn på bedrift/organisasjon: Aalesunds Schaklag
Navn på veileder ved bedrift/organisasjon: Arne Unneland
Aktivitetsplan

Uke	Dato	Gjennomført arbeid/Tema/aktivitet	Stud1 Timer	Stud2 Timer	Stud3 Timer
10	04.03	Websocket backend, resultatliste (spillerlobby), bachelorrapport, unittesting, oppdatere server	7	8	6
10	05.03	Websocket backend, spillerpause, designendringer, fikse docker, backend, server & andre bugs	7	10,5	8,1
10	06.03	Docker		3,67	
10	07.03	Lære om websocket			2
10	08.03	Backend: Websocket	1		
11	09.03	Backend: Websocket, valg av mostander Frontend: mobilskalering, redesign,	5,5	5,5	5
11	10.03	Backend: Websocket, redesign, Frontend: socket	7	7	6,75
11	12.03	Møte med veileder, gå gjennom forslag for å utvide oppgave. Backend: velge motstandere Frontend: redesign	5	2,5	7
12	16.03	Backend: velge motstandere	2,5		
12	17.03	Backend: velge motstandere, feilhåndtering. Frontend: feilhåndtering. Skrive på bachelorrapport websocket	7	7,5	8,2
Sum timer			42	44,42	43,05

Plan for neste uke, tema (aktivitetsplan)

12-14		Legge til funksjonalitet hvor vert kan endre resultat på spilte spill
12-14		Legge til funksjonalitet hvor vert kan kaste ut spillere fra turneringen. Spiller skal også kunne motta begrunnelse for utkastelsen
12-14		Legge til bedre feilhåndtering i de tilfellene hvor spiller gir feil inputdata.

IB303312 Bacheloroppgave
LOG
Utført arbeid i perioden

Navn på studenter:
Stud1: Andreas Moe Hatlø Stud2: Severin Aas Eliassen
Stud3: Jahn-Willy Aasen Nøstdahl
Navn på bedrift/organisasjon: Aalesunds Schaklag
Navn på veileder ved bedrift/organisasjon: Arne Unneland
Aktivitetsplan

Uke	Dato	Gjennomført arbeid/Tema/aktivitet	Stud1 Timer	Stud2 Timer	Stud3 Timer
12	18.01	Backend: sette opp parti Frontend: spiller detaljer for vert, vanlige feilmeldinger kontorarbeid	8	6,25	6
12	19.03	Backend: sette opp parti, Frontend: tabell og pagination komponenteter	7	7,5	5
12	20.03	Backend: sette opp parti Bachelor rapport, datamodelleringsbok	8	7	4
12	21.03	Backend: sette opp parti Frontend, Pagination buttons	4	4,5	
12	22.03	Backend, sette opp parti, bug hunting Frontend: Pagination buttons, bug hunting	7,5	1,25	
13	23.03	Refaktorering av backend, kaste ut spiller med begrunnelse. Backend: vert endrer resultat på partier Frontend: Spillerdetaljer for vertside, kaste ut spiller med begrunnelse	9,5	7,5	6
13	24.03	Skrive bachelorrapport Frontend: Beskjed til spiller om utkastelse med begrunnelse Backend: Beskjed til spiller om utkastelse med begrunnelse Backend: vert endrer resultat på partier	6,5	8	6
13	25.03	Backend: start applikasjon på starttidspunkt, Beskjed til spiller om utkastelse med begrunnelse, error handling Frontend: Beskjed til spiller om utkastelse med begrunnelse, error handling Frontend: vert endrer resultat på partier	9,5	8,16	5
13	26.03	Skrive bachelorrapport + Frontend, start turnering når backend gir beskjed Div bugfix, datamoddeleringsbok	8	9,33	6
13	27.03	Frontend, start turnering når backend gir beskjed, Oppdatere test serveren, fikse småbugs, bachelorrapport, teste server for xss og SQL – injection.	7	9,33	6,25

13	28.03	Fikse: server/docker, div bugs, vue research		5,45	
13	29.03	Skrive bachelorrapport	2		
14	30.03	Løst diverse «issues», Fikse div bugs.	8	7	7
14	31.03	Løst diverse «issues», Fikse div bugs, unittesting frontend.	8	8	7,5
Sum timer			93	89,27	58,75

Plan for neste uke, tema (aktivitetsplan)

14-16		Avslutte turnering
14-16		Push-varsel
14-16		«Turneringspause»

IB303312 Bacheloroppgave
LOG
Utført arbeid i perioden

Navn på studenter:
Stud1: Andreas Moe Hatlø Stud2: Severin Aas Eliassen

Stud3: Jahn-Willy Aasen Nøstdahl
Navn på bedrift/organisasjon: Aalesunds Schaklag
Navn på veileder ved bedrift/organisasjon: Arne Unneland
Aktivitetsplan

Uke	Dato	Gjennomført arbeid/Tema/aktivitet	Stud1 Timer	Stud2 Timer	Stud3 Timer
14	01.04	Veiledermøte, bachelorrapport, junittesting	8	7	5,5
14	02.04	Bachelorrapport, validering av resultat, push-varsler	8	5,5	6
14	03.04	Bachelorrapport, validering av resultat, push-varsler, turneringspause	8	8	5
14	04.04	Tls/ssl frontend & backend		5,25	
14	05.04	Bachelorrapport, ssl/tls	3	2	
15	06.04	Validering av resultat, ssl, forlat turnering varsel	8	7	6
15	07.04	Veiledermøte, avslutte turnering, bachelorrapport, heroku, sidetitler, ny logo, små frontendbugfikser, turneringsnavn på vertside	8	7	5
15	08.04	Avslutte turnering, bachelorrapport, refaktorisering, tabell for aktive spill (vert)	9	10,5	6
15	09.04	Bachelorrapport, diverse tasks, bugs og styling fiks, varsel når tilbakeknapp trykkes	8	6,25	6,5
15	10.04	Slette data når en avslutter/går ut av turnering, bachelorrapport, heroku, Bachelorrapport	8	5	5
16	13.04	Diverse tasks, bachelorrapport, docker og backend Heroku	8	4,30	4
16	14.04	Bachelorrapport, backend heroku, bare tillate vert til tournament/player/xx	8	8	6
Sum timer			84	76,05	55

Plan for neste uke, tema (aktivitetsplan)

16-18		Fikse bugs
16-18		HTTPS og push-varsel
16-18		Legge til sjakklokke i applikasjonen
16-18		Legge til omside i applikasjonen
16-18		Skrive bachelorrapport

IB303312 Bacheloroppgave
LOG
Utført arbeid i perioden

Navn på studenter:
Stud1: Andreas Moe Hatlø Stud2: Severin Aas Eliassen
Stud3: Jahn-Willy Aasen Nøstdahl
Navn på bedrift/organisasjon: Aalesunds Schaklag
Navn på veileder ved bedrift/organisasjon: Arne Unneland

Aktivitetsplan

Uke	Dato	Gjennomført arbeid/Tema/aktivitet	Stud1 Timer	Stud2 Timer	Stud3 Timer
16	15.04	Veiledermøte, kontorarbeid, heroku backend, bachelorrapport	8	8	6
16	16.04	Heroku, push-varsel, sjakkur, varseldialog	8	7,5	7
16	17.04	Bachelorraport, serviceworker, push-varsel	8	8	5
16	18.04	Refaktorisering frontend, bachelorrapport		11	
16	19.04	Refaktorisering frontend		11,5	
17	20.04	Refaktorisering frontend, fix frontend bugs, bachelorrapport, begrense tilgang på api	8	8	5,5
17	21.04	Service worker, webpush backend & frontend, begrense tilgang på api, dokumentasjon, omside	8	11	7,5
17	22.04	Webpush backend, bugfix, omside	8	7,33	6
17	23.04	Webpush backend, service worker, veildermøte, bugfix, bachelorrapport	8	10	6,5
17	24.04	Webpush backend, service worker, heroku Bachelorrapport, bugfix	8	11	5,25
17	25.04	Push-varsler frontend, bachelor rapport		7	
17	26.04	Push-varsler frontend & backend		5,17	
18	27.04	Hjelpeside, bugfix, dokumentasjon	8	6	6,25
18	28.04	Hjelpeside, bachelorrapport, bugfix, testing, dokumentasjon	8	7	7,5
Sum timer			80	118,5	62,5

Plan for neste uke, tema (aktivitetsplan)

18-20		Fikse siste bugs
18-20		Mulighet til å laste opp bilde når resultat legges inn
18-20		Sjakkbrett for å legge inn resultat
18-20		Testing
18-20		Generell forbedring

IB303312 Bacheloroppgave
LOG
Utført arbeid i perioden

Navn på studenter:
Stud1: Andreas Moe Hatlø Stud2: Severin Aas Eliassen
Stud3: Jahn-Willy Aasen Nøstdahl
Navn på bedrift/organisasjon: Aalesunds Schaklag
Navn på veileder ved bedrift/organisasjon: Arne Unneland
Aktivitetsplan

Uke	Dato	Gjennomført arbeid/Tema/aktivitet	Stud1 Timer	Stud2 Timer	Stud3 Timer
18	29.04	Kontorarbeid, bachelorrapport, bugfix, bilde-opplasting	8	6,33	6
18	30.04	Bachelorrapport, sikkerhetstesting, bugfix, sende og vise spilleres inaktive spill, bilde-opplasting	8	7	7,25
18	01.05	Hjelpe side, bachelorrapport, bilde-opplasting		8,25	5,5
18	02.05	Frontend: Hjelpe side, dokumentering, bilde-opplasting		6	2
18	03.05	Frontend: Dokumentering		3,333	
19	04.05	Frontend: bugs, bachelorrapport, hashing av adminID, bugfix, bilde-opplasting	7	7	5,5
19	05.05	Bachelorrapport, div bugs & QOL, bugfix, hvelv for å lagre secret key, bilde-nedlastning	8	7	8,75
19	06.05	Varsle spiller om at turnering er pauset, vise på turneringssiden at turnering er pauset, veiledermøte, hvelv for å lagre secret key, bilde-nedlastning	8	8	6
19	07.05	Varsle spiller om at turnering er pauset, vise på turneringssiden at turnering er pauset, navigation guarding, bugfix, bachelorrapport	7	10	
19	08.05	Route guarding, refactor leave page warning, bachelorrapport, startknapp ved skjema for turneringsoppretting, fjerne polling av tidligere spilte spill, bilde-nedlastning	8	8	5
19	09.05	Cypress, bilde-nedlastning		7,16	6
19	10.05	Cypress, bachelorrapport	3	9	
20	11.05	Cypress, bachelorrapport, bugfix, bilde-nedlastning	9	11	6,5
20	12.05	Cypress, bachelorrapport	8	13	
Sum timer			74	111,08	57,5

Vedlegg 3 Framdriftsrapporter

IB303312 Bacheloroppgave	Prosjekt Sjakkarena	Antall møter denne periode 1). 1	Firma – Oppdragsgiver Aalesunds Schaklag	Side 101 av 141
Rapport fra prosess **Framdriftsrapport**	Periode/uke(r) 2-4	Antall timer denne per. (fra logg) 123,32	Prosjektgruppe (navn) Andreas Moe Hatlø Severin Aas Eliassen Jahn-Willy Nøstdahl	Dato 22.01

Hovedhensikt / fokus for arbeidet i denne perioden Konseptualisering, Skrive forprosjektrapport, Informasjonsinnhenting
Planlagte aktiviteter i denne perioden Skrive brukerhistorier, wireframes, UML-diagram, designskisse. Møte med veileder, oppdatering av kunde, skrive forprosjektrapport, finne informasjon om database-, frontend- og backendteknologier.
Virkelig gjennomførte aktiviteter i denne perioden Skrive brukerhistorier wireframes, UML-diagram, designskisse. Møte med veileder, oppdatering av kunde, skrive forprosjektrapport
Beskrivelse av/begrunnelse for eventuelle avvik mellom planlagte og virkelige aktiviteter Ikke funnet tilstrekkelig med informasjon om database-, frontend- og backendteknologier. Årsak: nedprioritert på grunn av tidsmangel.
Beskrivelse av /begrunnelse for endringer som nå ønskes i selve prosjektets innhold eller i den videre framgangsmåten - eller framdriftsplanen Ingen endringer er ønsket, så langt.
Erfaring fra denne perioden Vi planla litt for mye arbeid for denne perioden. Lært mer om prosjektdokumentasjon.
Hovedhensikt/fokus neste periode Informasjonsinnhenting, skrive bachelorrapport
Planlagte aktiviteter neste periode Informasjonsinnhenting, fortsette på bachelorrapport (innledning og teoridel), lage presentasjon av prosjektet til systemfaget, fullføre forprosjektrapporten, ferdigstille designskisser, møte med veileder, oppdatering av oppdragsgiver
Annet
Ønske om /behov for veiledning, tema i undervisningen – drøfting ellers Problemstilling og avgrensning

IB303312 Bacheloroppgave	Prosjekt Sjakkarena	Antall møter denne periode 1). 1	Firma – Oppdragsgiver Aalsesunds Schaklag	Side 102 av 141
Rapport fra prosess Framdriftsrapport	Periode/uke(r) 4-6	Antall timer denne per. (fra logg) 146,18	Prosjektgruppe (navn) Andreas Moe Hatlø Severin Aas Eliassen Jahn-Willy Nøstdahl	Dato 05.02

Hovedhensikt / fokus for arbeidet i denne perioden
Informasjonsinnhenting, skrive bachelorrapport

Planlagte aktiviteter i denne perioden
Informasjonsinnhenting, fortsette på bachelorrapport (innledning og teoridel), lage presentasjon av prosjektet til systemfaget, fullføre forprosjektrapporten, ferdigstille designskisser, møte med veileder, oppdatering av oppdragsgiver.

Virkelig gjennomførte aktiviteter i denne perioden
Informasjonsinnhenting, fortsatt på bachelorrapport (innledning og teoridel), laget presentasjon av prosjektet til systemfaget, fullført forprosjektrapporten, oppdatert oppdragsgiver. Laget statiske frontendsider.

Beskrivelse av/begrunnelse for eventuelle avvik mellom planlagte og virkelige aktiviteter
Ferdigstille designskisser: Arbeidet er for omfattende og vil ta for lang tid. Vi har laget noen designskisser for å vise «designtema». Designtemaet har blitt brukt i produksjonen av statiske frontendsider.

Ikke hatt møte med veileder: Veileder har vært sykmeldt

Beskrivelse av /begrunnelse for endringer som nå ønskes i selve prosjektets innhold eller i den videre framgangsmåten - eller framdriftsplanen
Oppdragsgiver vil ha en mer tillitsbasert registrering av resultater enn det vi først hadde planlagt. Vi har også endret teknologi for frontend fra Bootstrap til Vue etter anbefaling fra oppdragsgiver.

Erfaring fra denne perioden
Lært om teknologiene Vue, Vuetify og Vuex.

Hovedhensikt/fokus neste periode
Legge til funksjonalitet i applikasjonen, skrive bachelorrapport

Planlagte aktiviteter neste periode
Fortsette på teoridel i bachelorrapport.
Legge til funksjonalitet i applikasjonen for å:

- Opprette og konfigurere turneringer
- Melde seg inn i turneringer
- Gi en unik id per epost til turneringsvert

Annet

Ønske om /behov for veiledning, tema i undervisningen – drøfting ellers
Problemstilling og avgrensning

IB303312 Bacheloroppgave	Prosjekt Sjakkarena	Antall møter denne periode 1). 1	Firma - Oppdragsgiver Aalesunds Schaklag	Side 103 av 141
Rapport fra prosess Framdriftsrapport	Periode/uke(r) 6-8	Antall timer denne per. (fra logg) 151,72	Prosjektgruppe (navn) Andreas Moe Hatlø Severin Aas Eliassen Jahn-Willy Aasen Nøstdahl	Dato 19.02

Hovedhensikt / fokus for arbeidet i denne perioden

Legge til funksjonalitet i applikasjon, skrive bachelorrapport

Planlagte aktiviteter neste periode

Fortsette på teoridel i bachelorrapport.

Legge til funksjonalitet i applikasjonen for å:

- Opprette og konfigurere turneringer
- Melde seg inn i turneringer
- Gi en unik id per epost til turneringsvert

Virkelig gjennomførte aktiviteter i denne perioden

Fortsette på teoridel i bachelorrapport.

Legge til funksjonalitet i applikasjonen for å:

- Opprette og konfigurere turneringer
- Melde seg inn i turneringer
- Gi en unik id per epost til turneringsvert

Beskrivelse av/begrunnelse for eventuelle avvik mellom planlagte og virkelige aktiviteter

Ingen avvik

Beskrivelse av /begrunnelse for endringer som nå ønskes i selve prosjektets innhold eller i den videre framgangsmåten - eller framdriftsplanen

Vi må ha mer fokus på skriving av bachelorrapport. Dette har blitt nedprioritert den siste perioden.

Erfaring fra denne perioden

Lært mer om: JWT, spring boot, Vue, Vuex, Vuetify, Axios.

Hovedhensikt/fokus neste periode

Legge til funksjonalitet i applikasjonen, skrive bachelorrapport

Planlagte aktiviteter neste periode

Fortsette på teori og metodedel i bachelorrapport.

Legge til funksjonalitet i applikasjonen for å:

- se resultater
- registrere resultater
- melde seg ut av turnering

Om det blir tid: Containerisering med Docker og CI med Travis

Annet

Ønske om /behov for veiledning, tema i undervisningen – drøfting ellers

Hvordan «secret key» bør lagres, hvordan lage push varsler.

IB303312 Bacheloroppgave	Prosjekt Sjakkarena	Antall møter denne periode 1). 2	Firma – Oppdragsgiver Aalesunds Schaklag	Side 104 av 141
Rapport fra prosess Framdriftsrapport	Periode/uke(r) 8-10	Antall timer denne per. (fra logg) 137,15	Prosjektgruppe (navn) Andreas Moe Hatlø Severin Aas Eliassen Jahn-Willy Aasen Nøstdahl	Dato 04.02

Hovedhensikt / fokus for arbeidet i denne perioden

Legge til funksjonalitet i applikasjonen, skrive bachelorrapport

Planlagte aktiviteter i denne perioden

Fortsette på teori og metodedel i bachelorrapport.
Legge til funksjonalitet i applikasjonen for å:

- se resultater
- registrere resultater
- melde seg ut av turnering

Om det blir tid: Containerisering med Docker og CI med Travis

Virkelig gjennomførte aktiviteter i denne perioden

Vi har lagt til all funksjonalitet som var planlagt med unntak av å se resultater for spiller.

Beskrivelse av/begrunnelse for eventuelle avvik mellom planlagte og virkelige aktiviteter

En refaktorerte mens en annen jobbet videre på gammel kode som gjorde at det ble rot når den «gamle» koden skulle slås sammen med den refaktorerte koden.

Beskrivelse av /begrunnelse for endringer som nå ønskes i selve prosjektets innhold eller i den videre framgangsmåten - eller framdriftsplanen

Etter anbefaling fra veileder går vi vekk fra å lage et «tillitsbasert» system, slik som kunde ville ha.

Erfaring fra denne perioden

Lært om asynkron javascript, docker, travis.

Hovedhensikt/fokus neste periode

Gå over fra «polling» til websocket, valg av motspiller, skrive bachelorrapport

Planlagte aktiviteter neste periode

Fortsette på teori og metodedel i bachelorrapport.
Legge til funksjonalitet i applikasjonen for å:

- Se en spillers resultater
- La spiller ta pause
- Velge motspiller

Bruke websocket
Redesign av hvordan turneringsinformasjon blir vist i spillerlobby

Annet

Ønske om /behov for veiledning, tema i undervisningen – drøfting ellers

IB303312 Bacheloroppgave	Prosjekt Sjakkarena	Antall møter denne periode 1). 2	Firma – Oppdragsgiver Aalesunds Schaklag	Side 105 av 141
Rapport fra prosess Framdriftsrapport	Periode/uke(r) 10-12	Antall timer denne per. (fra logg) 129.47	Prosjektgruppe (navn) Andreas Moe Hatlø Severin Aas Eliassen Jahn-Willy Aasen Nøstdahl	Dato 18.03

Hovedhensikt / fokus for arbeidet i denne perioden

Gå over fra «polling» til websocket, valg av motspiller, skrive bachelorrapport

Planlagte aktiviteter i denne perioden

Fortsette på teori og metodedel i bachelorrapport.

Legge til funksjonalitet i applikasjonen for å:

- Se en spillers resultater
- La spiller ta pause
- Velge motspiller

Bruke websocket

Redesign av hvordan turneringsinformasjon blir vist i spillerlobby

Virkelig gjennomførte aktiviteter i denne perioden

Fortsette på teori og metodedel i bachelorrapport.

Legge til funksjonalitet i applikasjonen for å:

- Se en spillers resultater
- La spiller ta pause

Bruke websocket

Redesign av hvordan turneringsinformasjon blir vist i spillerlobby

Beskrivelse av/begrunnelse for eventuelle avvik mellom planlagte og virkelige aktiviteter

Arbeidet med å velge motspiller var mer krevende enn først antatt. Nedstengning av skolen gjorde at vi mistet noen arbeidstimer.

Beskrivelse av /begrunnelse for endringer som nå ønskes i selve prosjektets innhold eller i den videre framgangsmåten - eller framdriftsplanen

Vi ønsker å utvide applikasjonen med litt mer «avansert» funksjonalitet. Dette kan f.eks. være innlogging hvor brukere kan se statistikk over turneringer som de tidligere har deltatt i. Det kan også være muligheten for å spille parti i applikasjonen dersom det skulle være mangel på sjakkbrett i spillelokalet. I tillegg bør vi gå vekk fra en tillitsbasert måte å legge til resultat hvor begge spillere må legge inn likt resultat for at det skal registreres.

Erfaring fra denne perioden

Lært om websockets.

Hjemmekontor

Brukergrensesnitt design ved bruk av ulike prinsipper om brukervennlighet

Hovedhensikt/fokus neste periode

Valg av motspiller, skrive bachelorrapport, legg til funksjonalitet i applikasjonen

Planlagte aktiviteter neste periode

Fortsette på teori og metodedel i bachelorrapport.

Legge til funksjonalitet i applikasjonen for å:

- Endre resultat på spilte spill
- Kaste ut spillere fra turnering. (inkludert begrunnelse).
- Bedre feilhåndtering i de tilfellen hvor spiller gir feil inputdata
- Velge motspiller

Annet
Ønske om /behov for veiledning, tema i undervisningen – drøfting ellers

IB303312 Bacheloroppgave	Prosjekt Sjakkarena	Antall møter denne periode 1). 1	Firma – Oppdragsgiver Aalesunds Schaklag	Side 107 av 141
Rapport fra prosess Framdriftsrapport	Periode/uke(r) 12-14	Antall timer denne per. (fra logg) 241,02	Prosjektgruppe (navn) Andreas Moe Hatlø Severin Aas Eliassen Jahn-Willy Aasen Nøstdahl	Dato 01.04

Hovedhensikt / fokus for arbeidet i denne perioden

Valg av motspiller, skrive bachelorrapport, legg til funksjonalitet i applikasjonen

Planlagte aktiviteter i denne perioden

Fortsette på teori og metodedel i bachelorrapport.

- Legge til funksjonalitet i applikasjonen for å:
- Endre resultat på spilte spill
- Kaste ut spillere fra turnering. (inkludert begrunnelse).
- Bedre feilhåndtering i de tilfellen hvor spiller gir feil inputdata
- Velge motspiller

Virkelig gjennomførte aktiviteter i denne perioden

Fortsette på teori og metodedel i bachelorrapport.
Legge til funksjonalitet i applikasjonen for å:

- Endre resultat på spilte spill
- Kaste ut spillere fra turnering. (inkludert begrunnelse).
- Bedre feilhåndtering i de tilfellen hvor spiller gir feil inputdata
- Velge motspiller

Bugfiksing

Beskrivelse av/begrunnelse for eventuelle avvik mellom planlagte og virkelige aktiviteter

Det ble tid til overs, så vi fikk tid til å rette opp kjente bugs

Beskrivelse av /begrunnelse for endringer som nå ønskes i selve prosjektets innhold eller i den videre framgangsmåten - eller framdriftsplanen

Det har blitt bestemt at applikasjonen skal utvides med muligheten til å spille parti online.

Erfaring fra denne perioden

Lært om turneringsoppsett
Lært om websocket frontend
JUnittesting for frontend er utfordrende

Hovedhensikt/fokus neste periode

Legge til funksjonalitet i applikasjonen, skrive bachelorrapport.

Planlagte aktiviteter neste periode

Legge til funksjonalitet for å:

- Avslutte turnering
- Push-varsel
- Godkjenning av resultat

Bugfix

Annet

Ønske om /behov for veiledning, tema i undervisningen – drøfting ellers

IB303312 Bacheloroppgave	Prosjekt Sjakkarena	Antall møter denne periode 1). 2	Firma - Oppdragsgiver Aalesunds Schaklag	Side 108 av 141
Rapport fra prosess **Framdriftsrapport**	Periode/uke(r) 14-16	Antall timer denne per. (fra logg) 215.05	Prosjektgruppe (navn) Andreas Moe Hatlø Severin Aas Eliassen Jahn-Willy Aasen Nøstdahl	Dato 15.04

Hovedhensikt / fokus for arbeidet i denne perioden

Legge til funksjonalitet i applikasjonen, skrive bachelorrapport.

Planlagte aktiviteter i denne perioden

Legge til funksjonalitet for å:

- Avslutte turnering
- Push-varsel
- Godkjenning av resultat

Bugfix

Virkelig gjennomførte aktiviteter i denne perioden

Legge til funksjonalitet for å:

- Avslutte turnering
- Godkjenning av resultat

Bugfix

Beskrivelse av/begrunnelse for eventuelle avvik mellom planlagte og virkelige aktiviteter

Push-varsel krever bruk av HTTPS. HTTPS var ikke mulig å legge til på autodeploy-serveren som vi har brukt hittil. Det har derfor gått med en del tid på å sette opp applikasjonen på en ny server.

Beskrivelse av /begrunnelse for endringer som nå ønskes i selve prosjektets innhold eller i den videre framgangsmåten - eller framdriftsplanen

Ønsker å legge til en sjakklokke i applikasjonen.

Erfaring fra denne perioden

Nettlesere blokkerer bruk av varsler dersom siden ikke er lastet inn med ssl/tls.

Hovedhensikt/fokus neste periode

Skrive bachelorrapport, legge til ny funksjonalitet i applikasjonen, bugfix

Planlagte aktiviteter neste periode

Legge til:

- HTTPS og push-varsler
- Sjakklokke
- Omside
- Bugfix

Skrive bachelorrapport

Annet

Ønske om /behov for veiledning, tema i undervisningen – drøfting ellers

IB303312 Bacheloroppgave	Prosjekt Sjakkarena	Antall møter denne periode 1). 2	Firma - Oppdragsgiver Aalesunds Schaklag	Side 109 av 141
Rapport fra prosess Framdriftsrapport	Periode/uke(r) 16-18	Antall timer denne per. (fra logg) 261	Prosjektgruppe (navn) Andreas Moe Hatlø Severin Aas Eliassen Jahn-Willy Aasen Nøstdahl	Dato 29.04

Hovedhensikt / fokus for arbeidet i denne perioden

Skrive bachelorrapport, legge til ny funksjonalitet i applikasjonen, bugfix

Planlagte aktiviteter i denne perioden

Legge til:

- HTTPS og push-varsler
- Sjakklokke
- Omside

Bugfix

Skrive bachelorrapport

Virkelig gjennomførte aktiviteter i denne perioden

Legge til:

- HTTPS og push-varsler
- Sjakklokke
- Omside

Bugfix

Skrive bachelorrapport

Beskrivelse av/begrunnelse for eventuelle avvik mellom planlagte og virkelige aktiviteter

Ingen avvik

Beskrivelse av /begrunnelse for endringer som nå ønskes i selve prosjektets innhold eller i den videre framgangsmåten - eller framdriftsplanen

Ingen endringer ønsket

Erfaring fra denne perioden

Push-varsler er mere avansert og tidkrevende enn først antatt

Hovedhensikt/fokus neste periode

Skrive bachelorrapport, bugfix, testing av applikasjon, legge til funksjonalitet

Planlagte aktiviteter neste periode

Fikse bugs, mulighet til å laste opp bilde når resultat legges inn, testing, generell forbedring
Om tid: sjakkbrett for å legge inn resultat

Annet

Ønske om /behov for veiledning, tema i undervisningen – drøfting ellers

IB303312 Bacheloroppgave	Prosjekt Sjakkarena	Antall møter denne periode 1). 1	Firma - Oppdragsgiver Aalesunds Schaklag	Side 110 av 141
Rapport fra prosess Framdriftsrapport	Periode/uke(r) 18-20	Antall timer denne per. (fra logg) 242,58	Prosjektgruppe (navn) Andreas Moe Hatlø Severin Aas Eliassen Jahn-Willy Aasen Nøstdahl	Dato 13.05

Hovedhensikt / fokus for arbeidet i denne perioden
Skrive bachelorrapport, bugfix, testing av applikasjon, legge til funksjonalitet

Planlagte aktiviteter i denne perioden
Fikse bugs, mulighet til å laste opp bilde når resultat legges inn, testing, generell forbedring
Om tid: sjakkbrett for å legge inn resultat

Virkelig gjennomførte aktiviteter i denne perioden
Fikse bugs, testing og generell forbedring. Proof-of-concept løsning for å laste opp bilde når resultat legges inn.

Beskrivelse av/begrunnelse for eventuelle avvik mellom planlagte og virkelige aktiviteter
Det ble ikke tid til å legge til sjakkbrett eller fullføre opplasting av bilde.

Erfaring fra denne perioden

Hovedhensikt/fokus neste periode
Fullføre rapport, lage presentasjon

Vedlegg 4 Møtereferat

Møtereferat

Dato
14.01.2020
Deltakere
Anniken Karlsen, Severin Aas Eliassen, Jahn-Willy Nøstdahl, Andreas Moe Hatlø
Innhold
På møtet ble det snakket om hva ideen vår er rundt prosjektet, og hvordan vi har planlagt å gå fram med å løse oppgaven. Vi fikk noen anbefalinger på hva som var lurt å gjøre framover og ting som må gjøres før neste møte (Tirsdag 28.1.2020, 09.00-10.00).
Ting som skal gjøres før neste møte:

- Lese https://innsida.ntnu.no/bacheloroppgave og lenker der om hvordan man skal skrive en bachelor oppgave.
- Konseptualisere oppgaven (Lage wireframes & andre diagrammer som kan hjelpe med å visualisere & planlegge prosjektet).
- Laste ned bachelorrapportmal og begynne å skrive innledning og legge inn konseptskisser.
- Skrive logg. Antall timer/klokkeslett for start og slutt, og hva en har jobbet med.

Anbefalinger:

- Pencil. Godt verktøy å lage skisser og konsept av hvordan modellen fungerer og utseende.
- Tenke på visuelle/grafiske element man kan bruke til å imponere med. Skape en wow-faktor.
- Lese tidligere bachelor oppgaver for å hente inspirasjon. Finne ut litt mer hva man burde ha med. https://ntnuopen.ntnu.no/ntnu-xmlui//discover?filtertype=doctype&filter_relational_operator=equals&filter=Bachelor+thesis
- Lage liste med funksjoner man kan utvide systemet med senere.

Det ble også sagt at vi må finne ut hva som er insentivet til å videreutvikle modellen, og eventuelle økonomiske modeller som er tenkt å bli brukt. Skal systemet vær gratis? Skal det være en abonnementstjeneste? Og hva er planen med prosjektet når vi er ferdig og leverer. Hvem skal vedlikeholde og videreutvikle systemet? Og hva skal eventuell motivasjon vær om applikasjonen er gratis å bruke. Så tilslutt må vi huske å beskrive hvorfor folk skal velge vår tjeneste over andre lignende tjenester som finnes allerede.

Møtereferat

Dato
23.01.2020

Deltakere
Oppdragsgiver sin kontakt person: Arne Unneland
Jahn-Willy Nøstdahl, Severin Aas Eliassen, Andreas Moe Hatløy

Innhold
Før møtet har gruppen sendt en PDF-fil med wireframes, konseptskisser og UML-diagrammer for å gi en kort oppdatering på hvor langt gruppen har kommet. Under møtet ble det så gitt tilbakemelding på disse diagrammene, og gitt ett par tips som kan hjelpe videre med prosjektet.
Punkt som ble nevnt:

- Kan ha valg om å starte automatisk når to har meldt seg på.
- Se på flytdiagrammene, det ene går fra å opprette turnering over i hvordan spillere interagerer, burde kanskje vær to forskjellige diagrammer.
- Tillitsbasert godkjenning. Begge kan legge inn resultat, blir endret til det den siste legger inn, synliggjør hva som har skjedd.
- Bra at hver enkelt spiller kan ta pause, kan vær lurt å også ha mulighet til å sette hele turneringen på pause.
- Om det er tid til slutt, kanskje bruke cypress til å vise en demonstrasjon av frontendapplikasjonen, eller eventuelt til testing av denne. Ikke anbefalt å gjøre tidlig i prosjektet, store endringer i applikasjonen vil føre til at test scriptene brekker. Cypress er nytt og kan bli en erstatter til selenium.
  https://www.cypress.io/

På møtet så gir kunden et inntrykk om at Vue.js er ønsket som frontendløsning, og trekker frem nyttige hjelpemidler i forbindelse med en slik løsning.

- Vuex, som er en «content store» som holder på tilstanden til alle komponenter på en plass, slik at dersom det blir en endring i en komponent i hierarkiet, så vil andre komponenter få med seg denne endringen. https://vuex.vuejs.org/
- Vuetify, ett komponent bibliotek som inneholder ferdige knapper, lister etc. som kan bidra til enklere styling av applikasjonen.
- Font Awesome, som er et bibliotek av vektorikon og logoer som er gratis å benytte.

Siden kunde har lang erfaring innen utvikling, spurte vi om hva som kan vær lurest løsning for å laste inn brukerne til listen over påmeldte spillere. Kunden tenkte at det kan vær lurt å starte med at frontendapplikasjonen spør server om endringer i ett gitt intervall, og om det er tid til slutt bytte ut denne løsningen med en websocketløsning. En websocketløsning gjør at server kan gi beskjed til frontend om at en spiller har meldt seg på og listen kan da oppdateres fortløpende.

Møtereferat

Dato
11.02.2020

Deltakere
Jahn-Willy, Andreas, Severin, Kjell-Inge Tomren.

Innhold
Møtet startet med en kort oppsummering av hva vi har gjort i prosjektet til nå. Etter oppsumeringen ble det stilt diverse spørsmål både fra biveileder og fra gruppen.

Veileders spørsmål:

- Er det noen plan for videre drift av systemet spesielt med tanke på sjakk klubbens resurser?
    - Pr nå så er det ingen konkret plan for videre drift av systemet. Dette må bli stilt spørsmål ved på neste møte med kunde.

Gruppens spørsmål:

- Gruppen tenker at problemstillingen skal være et spørsmål som rapporten skal svare på, men at det ikke er enkelt i ett slikt prosjekt. Så hvordan formulere en problemstilling i ett slikt softwareprosjekt der det er ganske konkret og relativt "*rett frem*" hva som skal lages?
    - Biveileder er enig i at en problemstilling kan vær vanskelig å formulere. Anbefaler å skrive oppgaven som en beskrivelse av applikasjonen, fortell om alternativ som er vurdert og begrunne alle konkrete valg som er tatt. Forutsetter at de som skal lese rapporten har teknisk bakgrunn så ikke utbroder for mye om generell teori.
    - Gå videre med formuleringen av problemstilling som er beskrevet i forprosjektrapporten. Benytt brukertester, både av spiller og administratorer for å svare på om applikasjonen er intuitiv og enkel å bruke, vis til statistikken som kommer fram.
- Det ble også stilt spørsmål om veileder har noen erfaring med WebSocket i Spring Boot.
    - Det er ikke tilfellet, så ble henvist videre til Girtz eller Mikael.

Det siste som ble nevnt under veiledningsmøte er at siden brukergrensesnittet blir et så sentralt tema i oppgaven, burde gruppen vise frem og eventuelt ta tidlige brukertester av dette. Der vi i testene skal presisere at dette er fortsatt i designfasen slik at endringer er enkelt å foreta. Dette er for å ikke implementere mye logikk slik at eventuelle redesigninger som må bli gjort tar altfor lang tid.

Møtereferat

Dato
27.02.2020

Deltakere
Jahn-willy, Andreas, Severin
Kjell-Inge, Anniken

Innhold
Møtet startet som vanlig med en gjennomgang av progresjonen siden sist møte. Under gjennomgangen ble de stilt forskjellige spørsmål og diverse problemstillinger med prosjektet ble tatt opp. Det ble også nevnt at en tidlig brukertesting burde skje så fort som mulig med hovedmål om å teste brukergrensesnittet før altfor mye logikk er koblet inn. Kjøre testen lokalt med en liten brukergruppe, gjerne med folk uten teknisk bakgrunn.

- **Problemstillinger, spørsmål og forbedringspunkt fra veilederne.**
  - Hva er det prosjektet går ut på? Få fram en bedre beskrivelse av prosjektet..
  - Kan det bli problematisk med folk som bruker lang tid på trekkene siden det ikke er planlagt noen form for tidskontroll i applikasjonen.
  - Mulighet for å trekke seg fra parti?
    - Pr nå så kan man ikke trekke seg uten å gi opp. Ulempe med andre former for å kunne trekke seg er jo at man bare trekker seg fra partiet om man står dårlig
  - Kunden tenkte systemet kunne vær tillitsbasert med tanke på resultatregistrering.
    - Veilederne utrykker at applikasjonen blir altfor enkel om det ikke er slike tester.
  - Databasemodellene er ikke oppdaterte, disse må oppdateres fortløpende. Veiledere anbefaler å finne måter å utvide systemet på.
  - Hva er incentivmodellen etter levering? Og hvordan planlegger sjakklubben å drifte applikasjonen etter den er levert?
    - Dette blir gruppen bedt å skrive om i rapporten.
- **Tips fra veilederne.**
  - Ta en tidlig brukertesting. Kan gjøres lokalt med liten testgruppe, gjerne med folk uten så mye teknisk bakgrunn.
    - Dette for å teste flyten og designet i applikasjonen, og for å se hvordan folk benytter seg av den. Siden kontaktperson for kunden har lang teknisk bakgrunn kan det være lurt å involvere folk uten slik bakgrunn også, siden ikke alle som er målgruppe har dette.
    - Lage en skikkelig testplan som kan benyttes i hovedrapport.
  - Husk å vis til litteratur i rapporten. Vise til litteratur med tanke på brukerdesignet.
    - "Design with the mind in mind"

Møtereferat

Dato
05.03.2020

Deltakere
Oppdragsgiver sin kontakt person: Arne Unneland
Jahn-Willy Nøstdahl, Severin Aas Eliassen, Andreas Moe Hatløe

Innhold
Siden forrige møtet har kunden blitt oppdatert på epost om progresjonen til gruppen. Det har gått litt lang tid siden sist møte på grunn av et avlyst møte i februar og vinterferien. På møtet viste gruppen applikasjonen så langt. Den generelle tilbakemeldingen er at kunden liker prosjektet så lang og gleder seg til å kjøre første test runde. Han kommer også med noen mer konkrete tilbakemeldinger og tips til prosjektet.

Spørsmål fra gruppen:

- Hva er insentivmodellen etter at applikasjonen er levert og hva er planen for videre utvikling?
    - Tanken er å kanskje ha det som ett «community prosjekt», der mange unge sjakkspillere kanskje har lyst til å ha ett hobby prosjekt og kan da være med på å videreutvikle/vedlikeholde applikasjonen.
- Noen tips til hvordan man kan gjøre applikasjonen mere avansert?
    - Benytte seg av rating. Avsnittet nederst.
    - Videreutvikle poengsystemet. Ikke slik som i vanlig sjakk der man får 0 for tap, 0.5 for uavgjort og 1 for seier. Gi ekstra poeng når man har en «winning streak», gi folk som har spilt få parti flere poeng enn folk som har spilt mange parti. Poeng utifra differansen mellom spillerne etc..

Diverse punkt som ble nevnt på møtet:

- Skaleringen treffer ikke helt på tv skjermen applikasjonen ble demonstrert på. Nevner at det kanskje er noe innebygd i vuetify som skalerer slik høyden treffer bedre.
- AdminUUID burde bli «hashet» før den blir lagret i databasen.
- Om rating skal bli implementert så burde kanskje den som lager turneringen få bestemme om det er aktuelt med rating eller ikke.
- La brukerne selv legge inn sin rating fra fide systemet.

Det ble snakket mye om rating i sjakk under møtet. Dette fordi gruppen vil utvide/gjøre applikasjonen mer avansert. Det finnes mange måter man kan implementere rating, man kan utvikle et eget system, adaptere systemet fra fide selv, eller benytte seg av et av de mange ratingsystemene som allerede er utviklet. Finnes mange eksempel fra spill som har dette. Om man benytter fide sin rating så vil der også kanskje være brukere som ikke har rating. Her nevner kunden at man kanskje ikke skal sette en forhåndsfastsatt rating, men heller benytte seg av «rating performance».

Møtereferat

DATO:
12.03.2020
Deltakere
Severin, Jahn-Willy, Andreas
Anniken
Innhold
Det ble gjort en kort gjennomgang av alt som var nytt siden sist, og hva som ble tatt opp og avtalt på møtet med kunden som har vært. Blant annet hva planen til oppdragsgiver med tanke på videreutvikling av prosjektet («open-source community prosjekt»).

Gruppen og veileder blir enig i at applikasjonen bør utvides. Flere ideer ble lagt på bordet, blant annet en brukerinnlogging slik de kan se statistikk i tidligere turneringer, la spiller legge inn sluttstillingen sin. For å legge inn sluttstilling ble de foreslått å la brukere ta bilde av brettet og laste opp dette. Det ble foreslått a ha å ha en «drag and drop»-løsning, eller at brukerne klikker på et felt og får opp en liste med brikker, for å registrere resultat.

Etter møtet skal gruppen ha en idémyldring og gå i dialog med oppdragsgiver for å legge frem forslag til endringer.

Med tanke på rapporten skal gruppen nå sende den i word-format hver fredag slik at veileder kan legge inn kommentarer. Applikasjonen må beskrives bedre i innledninga. Under flere punkt i rapporten burde det skrives en kort innledning og referering burde bli gjort bedre/tydeligere. Veileder nevner gestaltprinsippene og at gruppen burde vise til disse og skrive en kort innledning om at applikasjonen skal være brukervennlig. Boken «datamodellering» av Edgar Bostrøm blir anbefalt som litteratur.

Gruppen må merke all kode som blir tatt eller adaptert fra internett, slik at det ikke blir noen tvil om plagiering. Legge til kode som appendix og dele den opp i seksjoner slik at vi kan vise til koden i rapporten.

Møtereferat

DATO:
01.04.2020
Deltakere

Anniken Karlsen, Andreas Moe Hatlø, Severin Aas Eliassen, Jahn-Willy Nøstdahl

Innhold

Fredag 27.03, så sendte gruppen bachelorrapporten til veileder Anniken, og avtalte et møte på Skype. Skype er på grunn av situasjonen med covid-19. På møtet ble det først gått gjennom rapporten. Generelt sett var veileder fornøyd med rapporten, men der var små ting som kunne gjøres litt bedre.

- På alle definisjoner så ta med referanse. Referer til forfatter og skriv om definisjonen er adaptert.
- Vær forsiktig med for mange forkortelser, og bruk det kun der det er naturlig
- I interaksjonsdesignskapittelet skriv et lite avsnitt om hvorfor det er viktig med godt design med tanke på oppdragsgivers ønsker.
- Finn alternativ litteratur til Don Norman.
- Referer til databasemodellen og skriv en kort innledning i 2.6
- Når vi skriver noe i teksten, må vi skrive hvorfor vi skriver om det.
- I kapittel 3 om metoder, skal vi skrive at vi har en vanlig prosjektplanlegging med systemutvikling vedsiden av.
-

Det ble avtalt at gruppen skal jobbe ganske intensivt de neste dagene slik at all funksjonalitet som mangler blir implementert. Etter det skal gruppen begynne å implementere funksjonalitet som er tenkt på som utvidelser til prosjektet.

Møtereferat

Dato
15.04.2020
Deltakere
Anniken Karlsen, Andreas Moe Hatlø, Severin Aas Eliassen, Jahn-Willy Nøstdahl
Innhold
Møtet startet med en kjapp gjennomgang av utviklingen til applikasjonen og rapporten siden sist. Under gjennomgangen ble det pekt på et par moment som kan være med på å utvide applikasjonen videre.

1. Veileder savna grafikk og/eller bilder i applikasjonen for å løfte den.
2. Legge til at brukere kan laste opp sluttstillingen sin ved å ta bilde av brettet. (Turneringsverten kan se sluttstillingen om spillere skal være uenige om resultatet).
3. Legge til en sjakklokke. Ikke bare for pågående parti, men også som en egen funksjon slik at den kan benyttes uten at man er i en turnering. (Gir mer insentiv til å benytte applikasjonen).
4. En eller annen form for å notere trekk, for eksempel som å spille i applikasjonen (ikke mot motstander).
5. Legge til rangering. Eksempel: spillere rangerer seg selv fra 1-10, legger inn om de har spilt i en klubb etc.
6. Funksjonalitet for at spillere kan spille i applikasjonen
7. En hjelpeside som forklarer hvordan applikasjonen skal brukes.
8. Beskrive sjakkreglene og sjakkterminologien en plass i applikasjonen.
9. Sammenligne tidligere turneringer. (krever innlogging)

Gruppen burde også tenke på hvilken informasjon som kan legges til som kan være viktig/interessant å få ut til brukerne.
For rapporten ble det også nevnt et par punkt som kan forbedres/utvides.

1. Legg til mer grensesnitt-designlitteratur ikke bare gestaltprinsippene.
2. Skrive at vi har henta prinsipp fra scrum eller at vi benytter en tilpasset scrum om vi ikke følger denne utviklings metodikken til punkt og prikke.
3. Legge til rollebeskrivelse i databasetabellene i rapporten (bostrøm boken).
4. Vise mere kode fra frontend og forklare mer hvordan ting er programmert der.
5. Finne C-momentene i koden som vi har utviklet selv og få de direkte inn i rapporten.
6. Flytt deler av koden fra vedlegg, direkte inn i rapporten. Eksempel SQL-prosedyrer.

Etter veiledermøtet tok gruppen et kort internmøte der det ble bestemt å starte med hjelpesiden og sjakklokken som utvidelser. På det korte møtet ble de også nevnt at applikasjonen burde ha informasjon om hva som blir lagret om hver enkelt bruker, selv om pr nå det eneste som blir lagret er en «Json web token» for autentisering. Den holder kun på rollen til brukeren (turnering eller spiller) og en ID. Og uten om dette er det kun det brukere sender inn (vha. inputfelter) som blir lagret.

Møtereferat

Dato
23.04.2020

Deltakere
Anniken Karlsen, Andreas Moe Hatløy, Severin Aas Eliassen, Jahn-Willy Nøstdahl

Innhold
Hensikten med møtet var å oppdatere veileder på fremgangen og hva som er planlagt fremover. Møtet startet med å vise nye elementer på frontend (sjakklokke & om-side), så gikk det over til rapporten.

Rapporten begynner å ta form, gruppen skriver bra, men burde ha mere på drøftings delen. Gruppen burde også skrive om problemstilling tidlig i rapporten og nevne at det finnes liknende sider, men ikke til akkurat det samme formålet. Drøftingen burde inneholde alt som har med å lage en slik applikasjon og valgene gruppen tok. Rapporten «Byggordboka – en digitalisert fagordbok» ble nevnt som en mulig inspirasjon til rapporten.

Videre planlegger gruppen å fikse en del bugs, lage til opplasting av sluttstilling som bilde, legge inn bilder i hjelpesiden, og om det skulle være tid, ett sjakkbrett for å registrere trekk underveis.

Vedlegg 5 Forprosjektrapport

FORPROSJEKT - RAPPORT

FOR BACHELOROPPGAVE

NTNU
Kunnskap for en bedre verden

TITTEL:
Forprosjektrapport for sjakkarena

KANDIDATNUMMER(E):
SEVERIN AAS ELIASSEN, ANDREAS MOE HATLØ, JAHN-WILLY NØSTDAHL

DATO:	EMNEKODE: *	EMNE:	DOKUMENT TILGANG:
30.01.2020	IE303612	Bacheloroppgave (Data)	- Åpen

STUDIUM:	ANT SIDER/VEDLEGG:	BIBL. NR:
DATAINGENIØR	13/2	- Ikke i bruk -

OPPDRAGSGIVER(E)/VEILEDER(E):
Aalesunds Schaklag / Anniken Susanne Thoresen Karlsen, Kjell Inge Tomren

OPPGAVE/SAMMENDRAG:

Oppdragsgiver, Aalesunds Schaklag, ønsker et verktøy for å holde en uformell sjakkturnering. Applikasjonen, som vil være resultatet av dette prosjektet, skal kunne gi dem muligheten til dette. Denne typen applikasjon finnes allerede for turneringer som spilles online, men det er ønskelig å ha en applikasjon til bruk i turneringer som spilles på fysiske brett.

Slike turneringer fungerer på den måten at spillere blir satt opp mot hverandre fortløpende. Spillere som skal møtes skal ha laveste mulig poengforskjell. Man står fritt til å vinne partiene så fort som mulig for så å sette i gang et nytt parti, for å tjene til seg flere poeng.

Databasen skal holde på turneringer og brukere koblet til en turnering.
Applikasjonen skal ikke holde på brukere eller turneringer etter turneringsslutt.
Webapplikasjonen blir utviklet med HTML, CSS, Javascript, SpringBoot, vue.js og MySQL og skal være forholdsvis enkel å videreutvikle og vedlikeholde.
Benyttet arbeidsmetodikk er scrum.

Denne oppgaven er en eksamensbesvarelse utført av student(er) ved NTNU i Ålesund.

Postadresse
NTNU i Ålesund
Postboks 1517
N-6025 Ålesund

Besøksadresse
Larsgårdsvegen 2
Internett
www.ntnu.no

Telefon
70 16 12 00
Epostadresse
postmottak@ntnu.no

Telefax
70 16 13 00

Bankkonto
7694 05 00636
Foretaksregisteret
NO 947 767 880

INNHOLD

1 INNLEDNING

Aalesunds Schaklag har ønsket seg en applikasjon for å sette opp sjakkturneringer. En løsning for turneringsorganisering finnes allerede dersom offisielle turneringer skal organiseres. Det som sjakklubben ønsker seg er et program til bruk i uoffisielle settinger, slik som pubturneringer eller turneringer blant venner. I forbindelse med pubturneringer, eller liknende sammenkomster, er det ofte slik at de fleste deltakere ikke kjenner hverandre fra før. Det er derfor ønsket at deltakerne skal kunne melde seg inn i turneringen fra sine egne enheter og at applikasjonens skal gi instrukser om hvor, når og mot hvem det skal spilles.

Gruppen ønsker å lære mer om utvikling av webapplikasjoner. Dette innebærer blant annet å lære mer om server-klientarkitektur, databaseutvikling og menneske-maskininteraksjon. Oppgaven som er valgt, ser ut til å kunne tilfredsstille dette ønsket. I tillegg vil det kommende arbeidet gi nyttig erfaring i prosjektarbeid og det å arbeide med en ekstern oppdragsgiver.

2 BEGREPER

Klient-serverarkitektur	En måte å strukturere et nettverk av prosesser. Prosessene kan ha en av to roller: server eller klient. Som server venter prosessen på at en klient skal sende forespørsel om tilkobling.
Innebygd system	Datamaskin med lav ytelse som er bygget for å utføre en eller ett sett med funksjoner som en del av ett større system (Omnisci, u.d.).
Issue-tracker	Verktøy for å holde oversikt over problemer som skal løses.
Refaktorering	Omskriving av kode, uten å endre funksjonalitet.
Frontend	Den delen av applikasjonen som brukeren samhandler med. Ofte et visuelt brukergrensesnitt.
Backend	Den delen av systemet som kobler sammen databasen med frontend. I backend gjøres de fleste beregninger.
UML-diagram	Diagram basert på UML-standarden brukt bl.a. til modellering av programvareprosjekter
Wireframe	Tegning av de viktigste delene i en applikasjons brukergrensesnitt.

3 PROSJEKTORGANISASJON

3.1 ***Prosjektgruppe***

Studentnummer(e)
492841 – Severin Aas Eliassen 484151 – Jahn Willy Nøstdahl 471879 – Andreas Moe Hatlø

3.1.1 Oppgaver for prosjektgruppen - organisering

Scrum vil bli brukt som arbeidsmetodikk. Når scrum brukes har man som regel en «scrum-master» som fungerer som leder for gruppen. Utover å ha en «scrum-master» er det ingen bestemt organisering av gruppen annet enn en sekretærrolle som vil gå på rundgang. Scrum beskrives i avsnitt 5.3.

Prosjektarbeidet vil bli delt opp i sprinter på 2 uker. Før starten av hver sprint vil gruppen i fellesskap fordele oppgaver som skal gjøres i løpet av sprinten. Oppgavene er på forhånd prioritert sammen med kunde. Kollektivt eierskap vil bli praktisert. Det vil si at alle gruppemedlemmer er ansvarlig for alt arbeid som gjøres.

3.1.2 Oppgaver for «Scrum-Master»

Scrum-masters oppgave er å sørge for at prinsippene i Scrum blir fulgt av utviklingsteamet (Sommerville, 2015).

3.1.3 Oppgaver for sekretær

Sekretærs oppgave er å skrive møtereferat fra møter med veiledere og oppdragsgiver.

3.1.4 Styringsgruppe (veileder og kontaktperson oppdragsgiver)

Anniken Susanne Thoresen Karlsen er veileder, Kjell Inge Tomren er biveileder, mens Arne Unneland er kontaktperson for oppdragsgiver.

4 AVTALER

4.1 ***Avtale med oppdragsgiver***

Det er ikke inngått en skriftlig avtale med oppdragsgiver.

4.2 ***Arbeidssted og ressurser***

I hovedsak vil gruppearbeidet foregå på rom L167.

Veileder skal oppdateres på avtalte møter. Se avsnitt 7.1. Kontaktperson for oppdragsgiver skal oppdateres etter hver sprint enten via e-post eller ved at det holdes et møte.

4.3 Gruppenormer – samarbeidsregler – holdninger

Normer og samarbeidsregler

Arbeidstid er fra 09:00 til 16:00 mandag til fredag. I hovedsak skal gruppearbeidet gjøres på skolen, men det er mulig å arbeide andre steder dersom det er ønskelig. Ved behov må det også arbeides utenom nevnte arbeidstid.

Det skal legges ned en god innsats av alle gruppemedlemmer.

God kommunikasjon mellom gruppemedlemmer er viktig. Dersom et gruppemedlem er forhindret fra å møte opp til avtalt tidspunkt skal de andre medlemmene informeres om dette i god tid. Det skal være en lav terskel for å spørre hverandre om hjelp.

Holdninger

Dataingeniører som gruppe har stor innflytelse på verden rundt seg. Manges liv er i stor grad preget av, og i mange tilfeller avhengig av, datarelatert teknologi. For eksempel har feil i programvare nylig bidratt til to ulykker med fly av typen Boeing 737 Max (The New York Times, 2020). Dette viser at dataingeniører i mange tilfeller må ta sitt arbeid seriøst og ha et bevisst forhold til den innflytelsen de har.

Lovverk, slik som GDPR, er med på å regulere arbeidet til dataingeniører. I tillegg kan det argumenteres for at dataingeniører trenger etiske retningslinjer. Ved å følge slike retningslinjer kan det hende man får et mer bevisst forhold til dataingeniørers innvirkning på andres liv.

ACM er en organisasjon som har laget etiske retningslinjer for blant annet dataingeniører. Et eksempel fra ACMs etiske retningslinjer er at man skal «strebe etter å oppnå høy kvalitet i prosess og produkt» (ACM, u.d.). Et annet prinsipp er å følge gjeldende lovverk. Noen eksempel på hvordan man kan følge de etiske retningslinjene er:

- Det må legges ned en innsats i å skrive god kode med nok dokumentasjon. Dette gjør det lettere å vedlikeholde og videreutvikle koden.
- Det skal kun lagres informasjon som er helt nødvendig å lagre om brukere. Slik informasjon skal også kun lagres ved samtykke fra brukeren. For å ivareta sluttbrukers interesser så skal det også oppgis hva denne informasjonen skal brukes til.
- Ved bruk av eksterne biblioteker/pakker skal man se til at de er av god kvalitet og at de oppfyller krav til sikkerhet, slik at minst mulig tid senere trenger å bli brukt til å endre pakkene eller tette eventuelle sikkerhetshull.
- Ved bruk av en annens kode, både redigert og uredigert, skal den originale forfatteren krediteres. Dette skal gjøres på en måte slik at det kommer tydelig fram hvem som har skrevet den aktuelle koden og hvor den er hentet fra.

5 PROSJEKTBESKRIVELSE

5.1 Problemstilling - målsetting – hensikt

Problemstilling

Hvordan lage en applikasjon som kan brukes til å organisere uoffisielle sjakkturneringer?

Forskningsspørsmål som vil brukes til å besvare problemstillingen er:

- Hvordan lage en sjakkturneringsapplikasjon som er intuitiv å bruke?
- Hvordan lage et turneringsoppsett som blir godt mottatt av turneringens deltakere?

Resultatmål

Innen 20.05.2020 skal en fungerende sjakkturneringsapplikasjon være ferdig. Applikasjonen regnes som fungerende når den har funksjonalitet som beskrevet i de høyest prioriterte brukerhistoriene (se punkt 5.2 og vedlegg 1).

Effektmål

- Minst 90% av turneringsverter skal mene at applikasjonen gjør det enkelt å organisere sjakkturneringer.
- Minst 90% av applikasjonens brukere skal mene at applikasjonen gjør det enkelt å delta på sjakkturneringer.
- Applikasjonen skal kunne brukes av minimum 20 personer samtidig.

Milepæler/delmål er behandlet i avsnitt 5.7.1.

5.2 Krav til løsning eller prosjektresultat – spesifikasjon

Når scrum brukes som arbeidsmetode er kravspesifikasjonene skrevet som brukerhistorier. Brukerhistoriene kan leses i vedlegg 1.

Brukerhistoriene vil prioriteres i samarbeid med kunde. Optimalt sett er prosjektet fullført når applikasjonen har all den funksjonaliteten som brukerhistoriene forespeiler. Arbeidsmetoden scrum tillater å levere et produkt som ikke har all ønsket funksjonalitet, men som leverer nok til at den likevel kan brukes. Produktet anses derfor som fullført dersom den har funksjonalitet tilsvarende minstekravet til kunde eller mer. Det vil likevel bli forsøkt å levere så mye funksjonalitet som mulig.

Prosjektet har ingen bestemte økonomiske rammer. Det regnes med at arbeidet ikke vil medføre betydelige utgifter.

5.3 Planlagt framgangsmåte(r) for utviklingsarbeidet – metode(r)

Av metoder i programvareutvikling finnes det tre hovedkategorier: «fossefall», «inkrementell utvikling» og «integrasjon og konfigurasjon» (Sommerville, 2015).

Ved bruk av fossefallsmetoden gjennomføres de ulike stegene i en programvareutviklingsprosess (spesifikasjon, utvikling, validering og evaluering) sekvensielt. Fossefallsmetoden innebærer at en på forhånd planlegger alle stegene og så gjennomfører dem i henhold til denne planen.

Sommerville (2015, s. 49) hevder at fossefallsmetoden kun er egnet for innebygde systemer, kritiske systemer og store programvaresystem som inngår i et system utviklet av flere selskap.

Når inkrementell utvikling blir brukt som arbeidsmetode vil produktet leveres i versjoner (inkrementer) hvor en versjon bygger på den foregående versjonen.

Sommerville (2015, s. 50-51) nevner tre fordeler ved bruk av inkrementell utvikling sammenliknet med fossefallsmetoden: kostnader for å gjøre endringer er lavere, det er lettere å få tilbakemelding fra kunde og det åpner for muligheten til å levere produktet raskere til kunde. Inkrementell utvikling gjør det mulig å levere et uferdig produkt som har tilstrekkelig funksjonalitet til å bli tatt i bruk.

Av ulemper med inkrementell utvikling nevner Sommerville (2015, s.51) at prosessen kun evalueres når en versjon leveres og at mange endringer kan føre til en rotete kode. Disse ulempene kan overkommes med jevnlig refaktorering og nøye planlegging med kunde før man begynner arbeidet med å lage neste versjon.

Integrasjon og konfigurasjon innebærer å ta utgangspunkt i eksisterende komponenter og systemer, og ta disse i bruk i produktet en utvikler.

En fordel med integrasjons- og konfigurasjonsmetoden er at selve utviklingsprosessen kan gå raskere. En ulempe er at man risikerer dårligere ytelse fordi komponentene og systemene en tar utgangspunkt i ikke egner seg godt nok til å brukes i produktet.

Dette prosjektet er et mindre prosjekt, som er ikke-kritisk og det ferdige produktet vil ikke være et innebygd system. Derfor er det ikke aktuelt å bruke fossefallsmetoden. Det kan være gunstig å benytte seg av inkrementell utvikling ettersom denne metoden gir noe fleksibilitet. Ettersom at man har begrenset tid til å arbeide med bachelorprosjektet er det også en fordel å kunne levere deler av produktet til oppdragsgiver før alt er ferdig.

Det er mulig å kombinere inkrementell utvikling med integrasjon og konfigurasjon. I prosjektarbeidet vil det vil bli nødvendig å benytte seg av rammeverk som f.eks. Spring Boot, Vuetify eller liknende. Integrasjon og konfigurasjon er derfor en metode som også vil bli tatt i bruk.

Arbeidsmetoden scrum benytter seg av inkrementell utvikling. Scrum innebærer å arbeide i små team i to- til fireukers iterasjoner kalt sprinter. For hver sprint velges det funksjoner og arbeidsoppgaver fra en liste av ugjorte oppgaver som skal fullføres i løpet av sprinten. Etter hver sprint har man potensielt noe en kan levere til kunde og få tilbakemelding på.

Scrum har fordelene med inkrementell utvikling som nevnt ovenfor. Scrum er også en arbeidsmetode som er blitt brukt i forbindelse med arbeid i et annet emne. I tillegg samsvarer scrum med de retningslinjer for arbeidsmetode som er gitt datastudentene i forbindelse med bachelorprosjektet. Alt dette gjør at scrum vil bli brukt som arbeidsmetode i dette prosjektarbeidet.

5.4 Informasjonsinnsamling – utført og planlagt

Utført informasjonssamling

Det finnes enkelte applikasjoner som likner på den turneringsapplikasjonen som skal lages. Lichess.org og chess.com har begge funksjoner for å opprette og organisere turneringer, men hvor partiene spilles i applikasjonen. Utprøving av disse applikasjonene har gitt nyttig informasjon med tanke på hvordan en turneringsapplikasjon kan lages. I tillegg har vi hentet informasjon fra oppdragsgiver som har erfaring med bruk av tilsvarende sjakkturneringssystemer.

Informasjon om programvareutviklingsprosjekter er så langt funnet i pensumbøker brukt i studiet.

Planlagt informasjonssamling

Oppdragsgiver vil fortsatt kunne være en nyttig informasjonskilde..

Veiledere, pensumbøker og dokumentasjon på internett vil være gode kilder til informasjon vedrørende programvareutvikling. I tillegg vil det være nødvendig å finne informasjon om temaer som ikke har vært tilstrekkelig dekket i løpet av studiet, slik som design av brukergrensesnitt. Til dette kan for eksempel bøker som «Designing with the Mind in Mind» (Johnson, 2014) brukes.

5.5 Vurdering – analyse av risiko

Muligheten for å realisere prosjektet innen gitt ramme

Basert på gruppemedlemmenes erfaringer med tidligere prosjekter anses det som sannsynlig at dette prosjektet bli realisert innen gitt ramme. Som nevnt tidligere åpner arbeidsmetoden scrum opp for at prosjektet kan leveres selv om ikke all funksjonalitet er på plass. Dette øker sannsynligheten for å realisere prosjektet. Likevel er det flere hendelser som kan hindre realisering. Eksempler på slike hendelser er:

- Sykdom
- Teknologisvikt. Confluence, Jira, Github eller andre tjenester gruppen er avhengig av blir utilgjengelig
- Store endringer i krav til produktet
- Mangel på kompetanse
- Feilprioriteringer
- Dårlig gruppesamarbeid

Felles for alle hendelsene er at det er stor usikkerhet rundt hvor mye de bidrar til prosjektets risiko. Det lar seg trolig ikke gjøre å bestemme hendelsenes sannsynligheter og konsekvenser. Det kan likevel være lurt å gjennomføre tiltak for å redusere konsekvensen av eller risikoen til slike hendelser. Tiltakene blir presentert i tabell 1.

Tabell 1 Tiltak for å redusere konsekvens av negative hendelser

Hendelse	Tiltak
Teknologisvikt	For å redusere konsekvensen av teknologisvikt blir dokumenter sikkerhetskopiert.
Store endringer i krav til produktet	Kommuniser ofte med kunde slik at begge parter har et riktig inntrykk av hva som er realistisk å gjennomføre.
Mangel på kompetanse	Avgjør tidlig hvilke teknologier som skal brukes, samt lese seg opp på disse.
Feilprioriteringer	Kommuniser ofte med kunde og veileder slik at sannsynligheten for å oppdage feilprioriteringer økes.
Dårlig gruppesamarbeid	Følg normer, samarbeidsregler og holdninger beskrevet i 4.3.

5.6 Hovedaktiviteter i videre arbeid

Tabell 2 Oversikt over prosjektets hovedaktiviteter

Nr.	Hovedaktivitet	Hovedansvar	Tid/omfang
1	**Informasjonsinnhenting**		
1.1	Finne teorier om design av brukergrensesnitt	Jahn-Willy	14.01-20.01
1.2	Finne informasjon om backend-teknologier	Andreas	14.01-10.02
1.3	Finne informasjon om frontend-teknologier	Jahn-Willy	20.01-10.02
1.4	Finne informasjon om databaseteknologier	Severin	20.01-10.02
1.5	Finne informasjon om hvordan en kan organisere sjakkturneringer (oppsett, rangering, ...)	Andreas	01.02-10.02
2	**Konseptualisering**		
2.1	Lage Wireframes	Andreas	14.01-28.01
2.2	Skrive brukerhistorier (kravspesifikasjon)	Hele gruppen	14.01-22.02
2.3	Lage UML-diagram	Severin	14.01-10.02
2.4	Lage designskisse	Jahn-Willy	14.01-28.01
3	**Rapportskriving**	**Hele gruppen**	**14.01-20.05**
4	**Utvikling**		
4.1	Lage database	Severin	10.02-01.03
4.2	Lage backendapplikasjon	Andreas	10.02-10.05
4.3	Lage frontendapplikasjon	Jahn-Willy og Severin	10.02-10.05
4.4	Testing	Hele gruppen	10.02-10.05
5	**Skrive dokumentasjon til kunde**	**Hele gruppen**	
6	**Lage poster**	**Hele gruppen**	**15.05-20.05**

Vedlegg 2 er et Gantt-diagram over hovedaktivitetene.

5.7 Framdriftsplan – styring av prosjektet

5.7.1 Hovedplan

Prosjektet starter med informasjonsinnhenting og konseptualisering. Konseptualiseringen går ut på å danne seg en forestilling om hvordan applikasjonen skal bli. Til det trenger en å få på plass kravspesifikasjoner, designskisser, wireframes og UML-diagrammer. Kravspesifikasjoner gir en beskrivelse av hvordan kunde ønsker at det ferdige produktet skal bli. Designskisser og wireframes fungerer som en veiledning til hvordan det ferdige produktet skal utformes. UML-diagrammene viser en oversikt over hvilken teknologi produktet vil bestå av og hvordan produktet vil bli brukt.

Konseptualiseringen krever at en har gjennomført en del informasjonsinnhenting først. Blant annet må design av brukergrensesnitt gjennomføres etter at en har funnet relevant teori. I tillegg må UML-diagram som omhandler teknologivalg gjennomføres etter at ulike teknologier er blitt vurdert.

Rapportskriving er en aktivitet som vil gjennomføres i hele arbeidsperioden.

Utviklingen av applikasjonen har blitt delt opp i fire deler. Disse delene er database, backend, frontend og testing. Testing vil bli gjennomført underveis i utviklingen.

Når produktet er ferdig skal det skrives dokumentasjon til kunde. Til slutt skal det lages en poster som informerer om det gjennomførte prosjektarbeidet.

I tabell 3 finner man prosjektets milepæler. Bestemmelse av teknologier og ferdigstillelse av applikasjonens bestanddeler har blitt gitt tidsfrister. De øvrige milepælene er de høyest prioriterte brukerhistoriene slått sammen i grupper. Når disse blir gjennomført blir bestemt i forbindelse med hver enkelt sprintplanlegging. Dette er i henhold til hvordan scrum praktiseres.

Tabell 3 Oversikt over milepæler og tilhørende tidsfrister

Milepæl	Tidsfrist
Backend- og frontendteknologier er bestemt	10.02.2020
Turneringer kan bli opprettet og konfigurert	Sprint for gjennomføring bestemmes etter hvert
Brukere kan melde seg inn og ut av turneringer	Sprint for gjennomføring bestemmes etter hvert
Resultat kan registreres og få tildelt ny motstander	Sprint for gjennomføring bestemmes etter hvert
Database er ferdigstilt	01.03.2020
Frontend er ferdigstilt	10.05.2020
Backend er ferdigstilt	10.05.2020

5.7.2 Styringshjelpemidler

Styringshjelpemidlene vi har valgt er Jira og confluence.

Jira er et verktøy hvor man kan holde oversikt over nåværende, tidligere og fremtidige oppgaver laget av gruppen. Dette verktøyet hjelper til med å holde oversikt over hvem som jobber med hva og hvilke oppgaver som må gjøres. Det er i Jira oppgaver blir tildelt for hver sprint. I dette verktøyet får man også sprintrapporter med oversikt over hvilke oppgaver som ble utført av hvem i hver sprint.

I Confluence har man muligheten til å lage ulike prosjektdokumenter som diagrammer, wireframes og kravspesifikasjoner. Via Confluence kan gruppemedlemmer og veiledere få tilgang på disse dokumentene.

Microsoft Teams blir brukt til å lagre dokumenter som ikke er laget i Confluence.

5.7.3 Utviklingshjelpemidler

IntelliJ vil bli brukt som utviklingsmiljø for frontend- og backendapplikasjonene. MySQL Workbench vil bli brukt til databasedesign.

GitHub vil bli brukt til versjonskontroll og distribuering av kode.

5.7.4 Intern kontroll – evaluering

Issue-trackeren i Jira blir brukt til å holde oversikt over oppgaver. I issue-trackeren blir oppgavene markert med «i utvikling» eller «ferdig». På denne måten vil vi ha en oversikt over framdriften til gruppen. Framdriftsrapport og logg vil også bli brukt til å skaffe oversikt over framdriften. I tillegg vil det holdes daglige møter (daglig scrum) hvor hvert gruppemedlem forteller hva de gjorde forrige dag og hva de planlegger å gjøre den dagen møte holdes.

Noen av målene regnes som oppnådd når relevante funksjoner er implementert og applikasjonen fungerer som ønsket for brukerne. Dette gjelder blant annet effektmålene beskrevet i 5.1. For noen mål er det opp til oppdragsgiver å bestemme når de er nådd. I hovedsak gjelder dette resultatmålet som også er beskrevet i 5.1 Gruppen i fellesskap kan bestemme når noen delmål eller milepæler er oppnådd. Dette gjelder blant annet milepælen «backend- og frontendteknologier er bestemt».

5.8 Beslutninger – beslutningsprosess

Beslutninger blir tatt i samhold med oppdragsgiver og eventuelt veileder. I de tilfeller det lar seg gjøre vil gruppen ta beslutninger uten å involvere oppdragsgiver eller veileder. Internt i gruppen vil beslutninger bli diskutert og bestemt demokratisk.

I forbindelse med utarbeidelsen av forprosjektrapporten har bestemmelse av kravspesifikasjoner blitt gjort sammen med oppdragsgiver. Beslutninger vedrørende loggskriving, konseptualisering og skriving av bachelorrapporten er tatt i samarbeid med veileder. Andre beslutninger er så langt tatt av gruppen.

6 DOKUMENTASJON

6.1 Rapporter og tekniske dokumenter

Dokumenter som skal utarbeides

- Daglig logg over hvert gruppemedlems arbeid
- Logg hver 14 dag over gruppens arbeid
- Framdriftsrapport
- Bachelorrapport
- UML-diagram
- Kravspesifikasjon (brukerhistorier)
- Wireframes
- Designskisser
- Møtereferat
- Sprintrapporter
- Brukerveiledning

Rutiner

- Holde daglig oppdateringsmøte for gruppemedlemmer (daglig scrum)
- Oppdatere kunde etter hver sprint
- Møte veileder ca. hver 14 dag (se avsnitt 7.1)
- Skrive logg hver 14. dag
- Skrive personlig logg hver dag
- Sikkerhetskopiere ukentlig alle dokumenter

Godkjennelse

Logger, framdriftsrapport og bachelorapport godkjennes av gruppens medlemmer. UML-diagram, kravspesifikasjon, wireframes, brukerveiledning og designskisser godkjennes sammen med kunde.

Distribusjon

Logger, framdriftsrapport og bachelorrapport sendes til veiledere via e-post. Brukerveiledning sendes til kunde via e-post. For gruppens medlemmer er disse, i tillegg til møterapporter, tilgjengelig via Microsoft Teams. UML-diagram, kravspesifikasjon, wireframes og designskisser distribueres til kunde via e-post. For veileder og gruppens medlemmer er disse, i tillegg til sprintrapporter, tilgjengelig via Confluence.

Oppbevaring

Logger, framdriftsrapport, møtereferat og bachelorrapport oppbevares i Microsoft teams sin skylagringstjeneste. UML-diagram, kravspesifikasjon, wireframes, designskisser, møtereferat og sprintrapporter oppbevares i Confluence.

7 PLANLAGTE MØTER OG RAPPORTER

7.1 *Møter*

Veileder

Avtalte møter med veileder er vist i tabell 4.

Tabell 4 Avtalte møter med veileder

Dato & Tidspunkt
Tirsdag, 14 Jan. 12.00-13.00
Tirsdag, 28 Jan. 09.00-10.00
Torsdag, 13 Feb. 09.00-10.00
Torsdag, 27 Feb. 09.00-10.00
Torsdag, 12 mars. 08.15-09.15
Tirsdag, 31 mars. 08.15-09.00
Tirsdag, 14 Apr. 08.15-09.00
Mandag, 27 Apr. 08.15-10.00
Torsdag, 14 mai. 09.00-11.00

Kunde

Det er avtalt å oppdatere kunde etter hver sprint. Det er ikke avtalt faste møter med kunde.

Arbeidsgruppe

Som nevnt i 5.7.4 vil gruppen hver dag ha ett lite møte kalt «daglig scrum». Det vil også bli avholdt ett møte etter hver sprint der det blir gjort en kort gjennomgang av sprinten som har vært og en planlegging av neste sprint.

7.2 Periodiske rapporter

7.2.1 Framdriftsrapporter (inkl. milepæl)

Framdriftsrapport vil bli skrevet etter hver sprint.

7.2.2 Logg

Gruppemedlemmene skal skrive 14-dagers logg for å holde oversikt over hva som har blitt gjort og hvilke utfordringer som har dukket opp. Disse loggene skal skrives etter hver sprint.

Hvert gruppemedlem skal daglig loggføre sitt arbeid med prosjektet.

8 PLANLAGT AVVIKSBEHANDLING

Ved avvik må gruppen først se om avviket kan bli rettet opp av gruppen selv. Hvis det er en oppgave som ikke har god nok fremdrift vil det, om mulig, flyttes mer arbeidskraft til denne oppgaven. Dersom avviket ikke lar seg rette av gruppen, må kunde eller veileder kontaktes.

Dersom en oppgave ikke blir ferdig innenfor en gitt sprint, vil den følge med videre inn i neste sprint og ha høy prioritet, om ikke annet er avtalt med oppdragsgiver.

Hvert enkelt gruppemedlem har selv ansvar for å varsle om den ser at arbeidet ikke blir ferdig eller er mulig å fullføre innen avtalt tid.

9 UTSTYRSBEHOV/FORUTSETNINGER FOR GJENNOMFØRING

Det vil bli behov for å få tilgang til en server hvor applikasjonen kan kjøres. Confluence-, Jira- og office365-lisenser blir gitt av NTNU. IntelliJ har kostnadsfrie studentlisenser.

10 REFERANSER

ACM, u.d. *acm.org.* [Internett]
Available at: https://www.acm.org/code-of-ethics
[Funnet 14 januar 2020].

Johnson, J., 2014. *Designing with the Mind in Mind.* s.l.:Morgan Kaufmann Publishers.

Omnisci, u.d. *omnisci.com.* [Internett]
Available at: https://www.omnisci.com/learn/resources/technical-glossary/embedded-systems
[Funnet 23 januar 2020].

Sommerville, I., 2015. *Software Engineering.* 3. utgave red. s.l.:Pearson Education Limited .

The New York Times, 2020. *nytimes.com.* [Internett]
Available at: https://www.nytimes.com/2020/01/05/business/boeing-737-max.html
[Funnet 13 januar 2020].

VEDLEGG

Vedlegg 1	Kravspesifikasjon
Vedlegg 2	Gantt-diagram

Kravspesifikasjon

Målutgivelse	1.0
Epic	
Dokumentstatus	UTKAST
Dokumenteier	Andreas Moe Hatlø Severin Eliassen Jahn-Willy Nøstdahl
Designer	Severin Eliassen Jahn-Willy Nøstdahl
Utviklere	Andreas Moe Hatlø Severin Eliassen Jahn-Willy Nøstdahl
Kvalitetssikring	Andreas Moe Hatlø

Mål

- Lage en webapplikasjon for organisering av uformelle sjakkturneringer

Bakgrunn og strategisk passform

Fra oppgavebeskrivelsen: "5-30 personer møtes enten på en Pub som skal ha sjakk-kveld eller på en klubb. Noen kjenner hverandre, andre kjenner ikke hverandre.

De kommer og går til ulike tider. Hvem skal spille sammen? Inspirert av f.eks. Lichess Arena så kan man sette opp en tjeneste som kan brukes fra nettbrett eller mobiler for å melde seg inn og ut av turneringen.

Så vises hvem som skal møtes, resultatliste på en stor TV i lokalet. Etterhvert som noen blir ferdige så starter nye parti frem til man har meldt seg ut eller er ferdig."

Antakelser

- Deltakerne er først og fremst interessert i å bruke applikasjonen på nettbrett og mobiltelefoner

Funksjonelle krav

#	Tittel	Brukerhistorie	Viktighet	Notater
1	Oprette turnering	Som en turneringsvert vil jeg kunne opprette turneringer	MÅ HA	
2	Konfigurere turnering	Som en turneringsvert vil jeg kunne konfigurere turneringer.	MÅ HA	Det som kan bestemmes i prioritert rekkefølge: 1. Start og sluttid 2. Antall bord 3. Tid mellom parti og maks antall runder
3	UUID	Som en turneringsvert vil jeg motta en UUID per epost som vil jeg meg tilgang til å konfigurere turneringen.	BURDE HA	
4	Melde seg inn i turnering	Som en deltaker vil jeg kunne melde meg inn i en turnering, selv om jeg kommer sent.	MÅ HA	Pin tastes inn (kahootliknende system)
5	Registrere resultat	Som en deltaker vil jeg kunne registrere resultatet fra mine spill	MÅ HA	Seier, remis, tap, walk over, avlyst for hvert parti
6	Se resultatoversikt	Som en bruker vil jeg kunne se hvordan ser står til sammenlignet med andre spillere.	BURDE HA	Leaderboard
7	Melde seg ut av turnering	Som en deltaker vil jeg kunne melde meg ut av en turnering	BURDE HA	
8	Avslutte turnering	Som en turneringsvert vil jeg kunne avslutte min turnering.	MÅ HA	Manuell avslutning

9	Valg av motspiller	Som en deltaker vil jeg at applikasjonen skal velge ut en motspiller og farge for meg.	MÅ HA	• Gi deltakere bordnr. • Bestemme hvem som skal spille med hvite brikker og hvem som skal spille med sorte brikker • Valg av motspiller basert på egne poeng og tidligere motspilleres poeng
10	Push-varsel	Som en deltaker vil jeg ha push-varsel når en ny motstander er funnet.	BURDE HA	Varsel for nytt parti
11	Kaste ut deltakere	Som en turneringsvert vil jeg kunne kaste ut deltakere	KAN HA	Noen kan finne på å gå uten å melde seg ut
12	Begrunnelse	Som en bruker vil jeg ha begrunnelse ved utkastelese fra turneringen	KAN HA	
13	Spillerpause	Som en bruker vil jeg kunne ta en pause fra turneringen/stå over en runde uten at det påvirker resultatet.	KAN HA	
14	Turneringspause	Som en turneringsvert vil jeg ha mulighet til å sette turneringen på pause.	KAN HA	
15	Endre resultat	Som en turneringsvert vil jeg ha mulighet til å nedre resultatet på ferdig spillte kamper	BURDE HA	Folk kan legge inn feil resultat ved uhell. Kan også vær regelbrudd der endring av resultat er relevant

Ikke-funksjonelle krav

#	Tittel	Brukerhistorie	Viktighet	Notater
1	Intuitivt design	Som en bruker ønsker jeg at designet er intuitivt og at applikasjonen dermed er enkel å bruke	BURDE HA	Minst mulig tid på å forstå appen, mest mulig sjakk
2	Responsivt design	Som en bruker ønsker jeg å kunne bruke applikasjonen på enheter som en kan forvente at en slik webapplikasjon fungerer på	MÅ HA	TV visning + mobil/pad (ulike skjermstørrelser)
3	Moderne design	Sidene må være oversiktlig, fornuftig bruk av tekst og farger	BURDE HA	
4	Universell utforming	Siden må være laget slik at alle kan bruke den, uansett om man har en synshemming eller lignende.	BURDE HA	

Brukersamhandling og utforming

Spørsmål

Nedenfor er en liste over spørsmål som skal tas stilling til som følge av dette kravdokumentet:

Spørsmål	Utfall
Skal deltakere ha brukere i applikasjonen?	Nei
Skal turneringsleder ha bruker i applikasjonen?	Nei

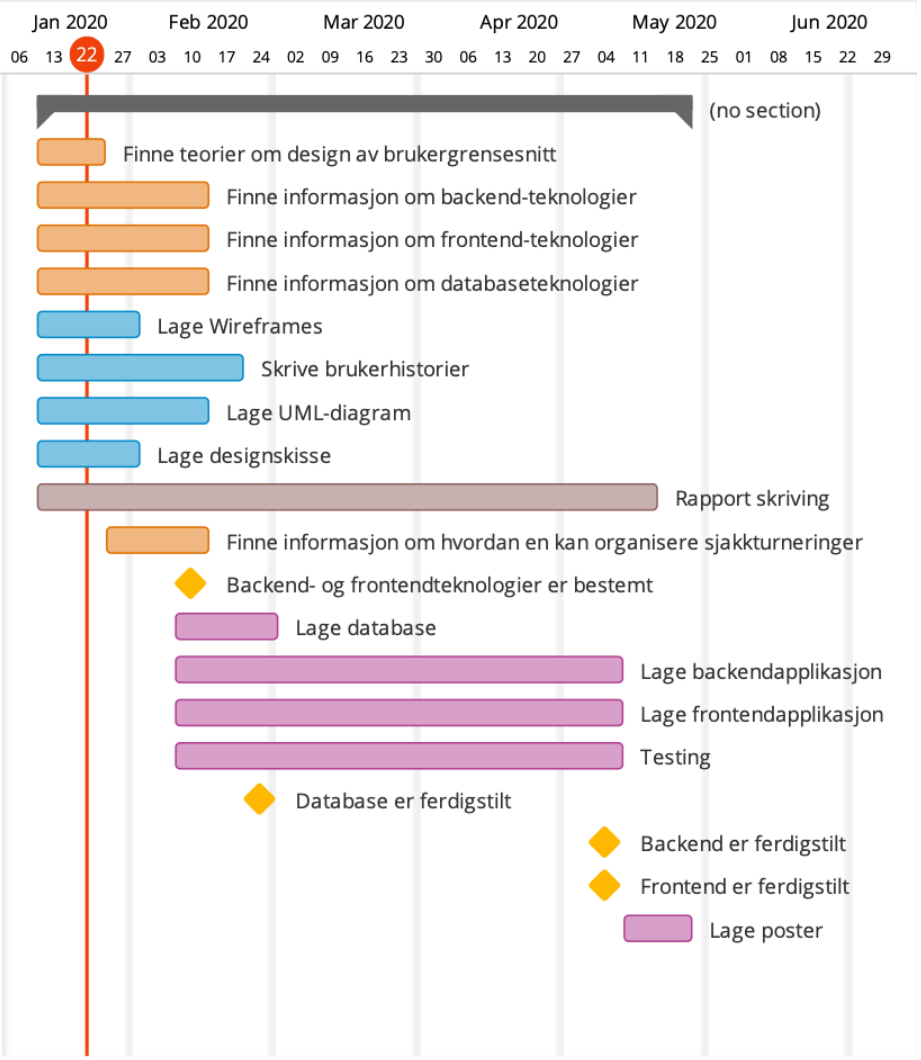
Gjør ikke

- *Bruke fide rating til å sette spillere opp mot hverandre.*
- *Passe på at en spiller ikke får samme farge brikke mange partier på rad.*

SjakkArena

Read-only view, generated on 22 Jan 2020

	ACTIVITIES	ASSIGNEE	EH	START	DUE	%
	(no section)		-	14/Jan	20/May	0%
1	Finne teorier om design av brukergre...	Unassigned	-	14/Jan	20/Jan	0%
2	Finne informasjon om backend-tekno...	Unassigned	-	14/Jan	10/Feb	0%
3	Finne informasjon om frontend-tekno...	Unassigned	-	14/Jan	10/Feb	0%
4	Finne informasjon om databasetekno...	Unassigned	-	14/Jan	10/Feb	0%
5	Lage Wireframes	Unassigned	-	14/Jan	28/Jan	0%
6	Skrive brukerhistorier	Unassigned	-	14/Jan	21/Feb	0%
7	Lage UML-diagram	Unassigned	-	14/Jan	10/Feb	0%
8	Lage designskisse	Unassigned	-	14/Jan	28/Jan	0%
9	Rapport skriving	Unassigned	-	14/Jan	15/May	0%
10	Finne informasjon om hvordan en ka...	Unassigned	-	31/Jan	10/Feb	0%
11	Backend- og frontendteknologier er ...	Unassigned	-	10/Feb	10/Feb	0%
12	Lage database	Unassigned	-	10/Feb	28/Feb	0%
13	Lage backendapplikasjon	Unassigned	-	10/Feb	10/May	0%
14	Lage frontendapplikasjon	Unassigned	-	10/Feb	10/May	0%
15	Testing	Unassigned	-	10/Feb	10/May	0%
16	Database er ferdigstilt	Unassigned	-	28/Feb	28/Feb	0%
17	Backend er ferdigstilt	Unassigned	-	10/May	10/May	0%
18	Frontend er ferdigstilt	Unassigned	-	10/May	10/May	0%
19	Lage poster	Unassigned	-	15/May	20/May	0%
20	Dokumentasjon til kunde	Unassigned	-			0%



Vedlegg 6:
Proof of concept for interaksjon med Vault

```
package no.ntnu.sjakkarena;

import no.ntnu.sjakkarena.utils.KeyHelper;
import no.ntnu.sjakkarena.utils.Vault;
import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

import java.io.Console;
import java.security.Security;
import java.util.Scanner;

@SpringBootApplication
public class SjakkarenaApplication {

    public static void main(String[] args) {
        Vault.init(getVaultToken());
        Security.addProvider(new BouncyCastleProvider());
        KeyHelper.createAndStoreKey();
        SpringApplication.run(SjakkarenaApplication.class, args);
    }

    /**
     * Get vault token from the application user
     *
     * @return The vault token
     */
    private static String getVaultToken() {
        System.out.println("Please enter your vault token: ");
        String token = null;
        if (System.console() != null) {
            Console console = System.console();
            token = new String(console.readPassword());
        } else {
            Scanner scanner = new Scanner(System.in);
            token = scanner.nextLine();
        }
        return token;
```

```
    }
}

package no.ntnu.sjakkarena.utils;

import org.springframework.vault.authentication.TokenAuthentication;
import org.springframework.vault.client.VaultEndpoint;
import org.springframework.vault.core.VaultTemplate;
import org.springframework.vault.support.VaultResponseSupport;

/**
 * Communicates with the vault where secrets are stored
 */
public class Vault {

    private static VaultTemplate vaultTemplate;


    /**
     * Initialises the Vault
     *
     * @param token The token used to open the vault
     */
    public static void init(String token) {
        VaultEndpoint vaultEndpoint = new VaultEndpoint();
        vaultEndpoint.setScheme("http");
        vaultTemplate = new VaultTemplate(vaultEndpoint, new TokenAuthentication(token));
    }

    /**
     * Writes a secret to the specified destination in the vault
     *
     * @param destination The destination where the secret will be stored
     * @param secret      The secret to store
     */
    public static void write(String destination, Object secret) {
        vaultTemplate.write("kv/sjakkarena/" + destination, secret);
    }

    /**
```

```
     * Returns a secret stored at the specified destination
     *
     * @param destination The destination where the secret is stored
     * @param c           The class of the secret to be returned
     * @param <T>         The type of the secret to be returned
     * @return a secret stored at the specified destination
     */
    public static <T> VaultResponseSupport<T> read(String destination, Class<T> c) {
        VaultResponseSupport<T> response = vaultTemplate.read("kv/sjakkarena/" + destination, c);
        return response;
    }
}


package no.ntnu.sjakkarena.utils;

import no.ntnu.sjakkarena.data.EncryptionKey;
import org.springframework.http.converter.HttpMessageConversionException;
import org.springframework.vault.support.VaultResponseSupport;

import java.security.Key;



/**
 * Handles public and private keys
 */
public class KeyHelper {

    private static EncryptionKey key = new EncryptionKey();

    /**
     * Returns a secret key
     *
     * @return a secret key
     */
    public static Key getKey() {
        VaultResponseSupport<EncryptionKey> response = Vault.read("symmetric", EncryptionKey.class);
        return response.getData();
    }

    /**
     * Creates and stores a secret key
```

```
     */
    public static void createAndStoreKey() {
        try {
            getKey();
        } catch (HttpMessageConversionException | NullPointerException e){
            Vault.write("symmetric", key);
        }
    }

    /**
     * Returns public key
     * See this for easy keys in right format https://web-push-codelab.glitch.me/
     *
     * @return Public key
     */
    public static String getPublicKey() {
        return "";
    }

    /**
     * Returns private key
     * See this for easy keys in right format https://web-push-codelab.glitch.me/
     *
     * @return Private key
     */
    public static String getPrivateKey() {
        return "";
    }

}

package no.ntnu.sjakkarena.data;

import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.security.Keys;

import javax.crypto.SecretKey;

/**
 * Represents a symmetric key.
 */
```

```
public class EncryptionKey implements SecretKey {

    private String algorithm;
    private String format;
    private byte[] encoded;



    /**
     * Constructs a symmetric key
     */
    public EncryptionKey(){
        SecretKey secretKey = Keys.secretKeyFor(SignatureAlgorithm.HS256);
        this.algorithm = secretKey.getAlgorithm();
        this.format = secretKey.getFormat();
        this.encoded = secretKey.getEncoded();
    }

    @Override
    public String getAlgorithm() {
        return algorithm;
    }

    @Override
    public String getFormat() {
        return format;
    }

    @Override
    public byte[] getEncoded() {
        return encoded;
    }
}
```