Universitetet i Sørøst-Norge
Fakultet USN Handelshøyskolen
Institutt for økonomi og IT

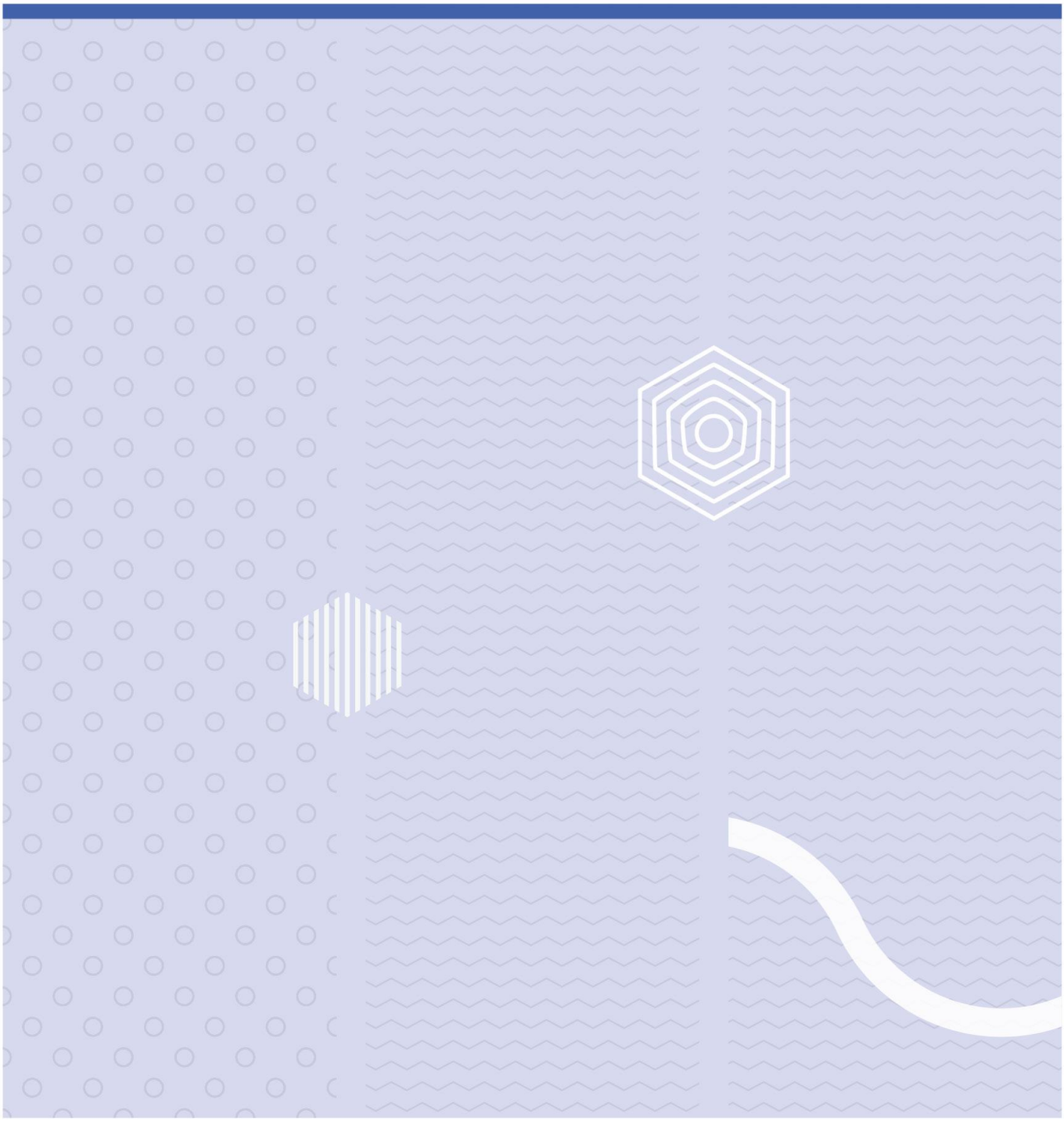
–

Prosjekt
06/05/2019
6117 Bachelorprosjekt

113049 - Kristian Stang

Utvikling av moderne JavaScript-applikasjoner

WebChess – en webbasert sjakkapplikasjon



Innholdsfortegnelse

1 Innledning

1.1 Idéen bak oppgaven

Idéen for prosjektoppgaven oppstod under arbeidet med en obligatorisk oppgave i kurset «6109 – Objektorientert programmering», høsten 2017. Denne oppgaven gikk ut på å lage en veldig enkel Java-applikasjon, der man skulle vise et sjakkbrett og tegne brikker på brettet, samt kunne gjøre enkle trekk ved hjelp av tekstlig input. Det var kun krav om å implementere deler av logikken for et par av brikketypene i spillet, og førte av denne grunn til en ganske ufullstendig applikasjon sett fra en sjakkspillers ståsted.

Som både sjakk- og programmeringsinteressert, fikk jeg under arbeidet med denne oppgaven raskt tanker om å lage en mer fullstendig applikasjon, der alle sjakkens regler var hensyntatt. Samtidig hadde jeg på denne tiden fått sansen for JavaScript-rammeverket «React», og var på jakt etter idéer om noe jeg kunne lage for å få trening i dette rammeverket.

Resultatet av begge disse faktorene ble idéen om å lage en webbasert sjakkapplikasjon. Ettersom sjakk er interaktivt og krever to spillere var vurderingen at dette konseptet måtte egne seg ypperlig som en webapplikasjon, og samtidig ville et slikt prosjekt både la meg arbeide med og utvikle noe jeg allerede er svært interessert i, og samtidig gi meg god trening i bruken av «React», samt andre rammeverk og biblioteker som ofte brukes sammen med «React».

Ettersom utviklingen ville være langt mer krevende enn den obligatoriske oppgaven nevnt innledningsvis, bestemte jeg meg dermed for å gå videre med denne ideen som mitt bachelorprosjekt.

1.2 Oppgavens hovedmål

Hovedmålet for oppgaven har vært å utvikle en fullverdig webapplikasjon der to brukere kan spille sjakk med hverandre over internett. Hovedfokuset har ligget på en kort liste over kjernefunksjonalitet som jeg betrakter som det viktigste for at en slik applikasjon skal lykkes.

- Brukere skal kunne opprette og bli med på sjakkpartier ved hjelp av noen få klikk.
- Brukere skal enkelt kunne invitere venner til partiet ved å dele en generert lenke.
- Applikasjonen skal være enkel og oversiktlig, og gi relevant informasjon om partier på en ren og ryddig måte. Det skal ikke oppstå forvirring rundt noen av GUI-elementene.
- Brikker på brettet må kunne flyttes ved hjelp av både individuelle klikk, og ved hjelp av klikk-og-dra-funksjonalitet.
- Applikasjonen må støtte dataoverføring i sanntid.

I tillegg til disse målene har jeg også satt opp noen mer personlige mål.
Gjennom arbeidet med denne oppgaven har jeg som mål å tilegne meg mer kunnskap om ulike moderne rammeverk og biblioteker som brukes i produksjonsmiljøer på internett i dag. Jeg ønsker å få trening og erfaring i disse rammeverkene og bibliotekene, da spesielt med tanke på «React», som har vært av interesse for meg siden idéen oppstod.

I tillegg ønsker jeg erfaring med selve utviklingsprosessen, og å lære mer om hvordan det faktisk er å utvikle større applikasjoner i JavaScript. Ettersom denne oppgaven har et mye større omfang enn noe annet jeg har utviklet tidligere, vil dette prosjektet gi meg gode muligheter til nettopp dette.

Kravspesifikasjon finnes også i prosjektbeskrivelsen (vedlagt).

2 Administrative forhold

2.1 Oppdragsgiver

Ettersom jeg har hatt en idé for bachelorprosjektet på forhånd og formulert oppgaven selv, har jeg valgt å være min egen oppdragsgiver. Jeg kom fram til at dette ville være mest hensiktsmessig, ettersom sjakkreglene er svært statiske samtidig som de er veldokumenterte.

På grunn av dette har den ene delen av prosjektet vært en mer eller mindre direkte implementasjon av disse i en egen sjakkmodul, mens resten av prosjektet har gått ut på utforming av web-applikasjonen som skal bruke sjakkmodulen. Ettersom jeg allerede kjenner disse sjakkreglene godt fra før, har det ikke vært behov for noen oppdragsgiver med tanke på veiledning og domenekunnskap, og jeg konkluderte med at jeg ikke ønsket dette.

2.2 Prosjektgruppe

I utgangspunktet var det planlagt at det skulle være tre deltakere på dette prosjektet. Ved prosjektstart valgte imidlertid den ene av dem å gå ut i praksis, mens den andre valgte å arbeide med et annet prosjekt tilknyttet arbeidsgiveren sin. Som resultat av dette har jeg endt opp som eneste gruppemedlem. Jeg valgte allikevel å fortsette med dette prosjektet, etter å ha vurdert arbeidsmengde og krav til applikasjonen på nytt.

2.3 Utstyr og programvare

Ettersom jeg var alene om prosjektoppgaven uten oppdragsgiver, hadde jeg ikke behov for noe felles arbeidslokale eller utstyr. Alt arbeid har blitt utført på egen PC hjemme hos meg selv.

Fordi prosjektet er et rent utviklingsprosjekt, fantes det ikke særlig store krav til programvare og annet utstyr. Alt arbeid kunne gjøres fra en hvilken som helst vanlig PC, ved hjelp av standard kodeprogramvare.

Som teksteditor/IDE har jeg brukt programmet Visual Studio Code, som er en åpen/fri kodeeditor utviklet av Microsoft. Den har støtte for plugins, og kan enkelt tilpasses og utvides med ekstra moduler som letter arbeidet med kode betraktelig.

Jeg valgte å bruke Node.js som utviklingsplattform, ettersom JavaScript er det kodespråket jeg kjenner absolutt best. I tillegg er det hensiktsmessig å bruke JavaScript som språk når man skal lage webapplikasjoner, ettersom dette språket er støttet i de fleste, om ikke alle, nettlesere. Man får dermed den fordelen av å ha samme språk i hele applikasjonen, både på frontend og backend. Gjennom Node.js har jeg også hatt tilgang til mange ferdige pakker og biblioteker, takket være det enkle pakkesystemet «npm».

Til tross for at jeg arbeider alene, har jeg brukt Git og GitHub for versjonskontroll og lagring av kode. Selv om Git sine største fordeler først og fremst merkes når man jobber i grupper, har jeg hatt stor nytte av å lagre koden på GitHub, ettersom det gir muligheter for både backup og enkel «rollback» dersom jeg skulle gjøre store feil i arbeidet med applikasjonen.

Jeg har brukt Google Chrome som nettleser og som debugger i arbeidet med prosjektet. En fordel man har når man utvikler webapplikasjoner er nettopp at de fleste nettlesere har funksjonalitet som gjør det lett å se hvordan applikasjonen oppfører seg for brukerne, og man kan lett finne feil ved å logge verdier til konsollet.

2.4 Arbeidsmetode

Til tross for at jeg har vært alene om prosjektet, har jeg prøvd å strukturere arbeidet ved å følge arbeidsmetodikken Kanban. Grunnen til dette er at kravene til applikasjonen allerede måtte endres kraftig før prosjektet i det hele tatt hadde startet, og det var derfor uklart hvilken funksjonalitet som skulle støttes i den ferdige applikasjonen. En smidig arbeidsmetodikk passer utmerket til prosjekter der kravene kan måtte endres underveis, og er hovedgrunnen til at jeg valgte dette.

Når det gjaldt valg av Kanban over Scrum, falt dette på at jeg foretrekker å arbeide med delmålene i et prosjekt til de er fullstendig utført, og det passet derfor bedre å jobbe etter Kanban, som begrenser antall delmål det skal jobbes med samtidig. Dette i kontrast med Scrum, som er tidsbasert.

I tillegg til dette følte jeg at jeg som uerfaren utvikler ikke ville være i stand til å estimere tidsbruken for hver oppgave særlig nøyaktig, noe som er viktig når man bruker Scrum som arbeidsmetodikk. Jeg følte derfor at det var mer hensiktsmessig å fullføre de viktigste delmålene først, og la arbeidet ta den tiden det trengte, før jeg eventuelt gjorde nye vurderinger senere med tanke på gjenstående tid i prosjektet, og kunne justere kravene deretter.

2.5 Framdriftsplan

Da jeg startet arbeidet med prosjektet, satte jeg opp en grov plan for arbeidet utover semesteret. Jeg regnet med å legge inn ca. 20 timer arbeid per uke over 17 uker, som gir 340 arbeidstimer totalt.

#	Oppgave	Ferdig	Beskrivelse
1	Planlegging	05.02.2019	Planlegging av utviklingsarbeidet. Valg av arbeidsmetode. Valg av plattform og språk.
2	Sjakkmotor	05.02.2019	Sjakkmotoren er ferdigutviklet. Sjakkpartier kan spilles gjennom metodekall. Dokumentasjon er påbegynt
3	Webtjener	12.03.2019	Webtjeneren er ferdigutviklet. Sjakkpartier kan spilles gjennom manuelle trekk i GraphQL-hjelpemiddelet «GraphQL Playground». Dokumentasjon er påbegynt.
4	Webklient	30.04.2019	Webklienten er ferdigutviklet. Sjakkpartier kan spilles i GUI. Applikasjonen er funksjonell. Dokumentasjon er påbegynt.
5	Dokumentasjon	07.05.2019	Applikasjonsdokumentasjonen og prosjektdokumentasjonen er ferdig. Finpussing av applikasjonen.
6	Prosjektslutt	07.05.2019	Innlevering av prosjektrapporten. Innlevering av kildekode.

Fristene i milepælsplanen er lagt opp etter datoene for milepælspresentasjonene utover semesteret. Disse presentasjonene ble naturlige start- og stoppunkter for de ulike fasene, og delte prosjektet opp i tidsperioder som passet godt med oppgavene i prosjektet.

2.6 Utfordringer og vurderinger

Som nevnt innledningsvis hadde jeg planlagt å jobbe sammen med to andre studenter på dette prosjektet, men ettersom begge disse endret planer like før semesterstart støtte jeg på utfordringer umiddelbart. Allerede før prosjektet hadde startet hadde jeg gjort noen tanker om arbeidsfordelinger og tidsbruk, men ble nødt til å forkaste dette og starte på nytt da jeg endte opp alene på prosjektgruppa. Det innebar at jeg var nødt til å kaste ut mange av de funksjonelle kravene jeg hadde planlagt å implementere, ettersom jeg nå var svært usikker på om jeg hadde tid til å fullføre det jeg hadde tanker om. Samtidig var det ikke mye jeg kunne ha gjort for å forutse dette, ettersom de andres beslutninger ble tatt ganske sent opp mot prosjektstart.

En annen utfordring jeg støtte på underveis i arbeidet var at jeg undervurderte arbeidsmengden som krevdes ved utviklingen av sjakkmotoren. Mye av logikken bak brikkenes bevegelse og trekklogikken hadde jeg tenkt gjennom på forhånd, men enkelte funksjoner krevde langt mer komplisert logikk enn jeg hadde tenkt. Spesifikt kan jeg nevne bygging av algebraisk notasjon for hvert trekk, som jeg i utgangspunktet kun trodde ville kreve navnet på brikken og navnet på feltet. Det viste seg at for å generere nøyaktig notasjon, så er det nødvendig å huske stillingen før trekket, og undersøke om det finnes andre brikker av samme type som kan flytte til det samme feltet. I så fall må notasjonen inneholde mer informasjon enn ved vanlige trekk.

I tillegg til dette oppstod det også et annet problem i arbeidet med sjakkmotoren. Ettersom jeg planla å arbeide med sjakkmotoren først, og deretter jobbe med webapplikasjonen, fantes ingen enkel måte for meg å visualisere det jeg hadde gjort i sjakkmotoren. Derfor var jeg nødt til å endre litt på planen ganske tidlig, og implementere deler av webserver og -klient tidligere enn planlagt. Som et minimum var jeg nødt til å kunne se en grafisk representasjon av brikkene og brettet, noe som lettet arbeidet betraktelig i forhold til å kun bruke konsollet.

Disse to faktorene gjorde at jeg måtte bruke mer tid på første del av prosjektet, og at sjakkmotoren ikke var fullført innen den planlagte fristen.

Når det gjelder utfordringer ved selve arbeidsmetodikken og fremdriften merket jeg at mens man har mye frihet når man jobber alene og selvstendig, er det lett å begynne å gå utenfor planen. Ved å ha andre mennesker å jobbe med, kan man lettere holde seg motivert for arbeidet og «pushe» hverandre, og det blir lettere å holde tempoet jevnt oppe fordi man ender opp med en ansvarsfølelse ovenfor andre på gruppa. Jeg merket underveis at det var litt for lett å bare sette i gang med programmeringen av ny funksjonalitet, uten at det nødvendigvis var del av planen på det tidspunktet.

Til tross for dette synes jeg planen jeg hadde lagt opp fungerte veldig godt. Ved å dele inn arbeidet i tre hoveddeler følte jeg at jeg hadde tidsfrister å forholde meg til, og dette hjalp meg med å forbli effektiv. Metodikken jeg valgte fungerte godt, med tanke på at jeg ble nødt til å endre noen av kravene underveis, noe som Kanban takler fint. Jeg hadde hele tiden noe å teste underveis, og etter å ha implementert de grunnleggende webfunksjonene var det mulig å få andre til å teste applikasjonen uten å måtte kjenne til det som lå bak. Dette bidro også til å motivere, gjennom positive tilbakemeldinger fra andre underveis, samt konstruktiv kritikk om ting som måtte forbedres.

3 Prosjektarbeidet

3.1 Planlegging

Som nevnt innledningsvis startet planleggingsarbeidet allerede på høsten 2017. Jeg kom frem til en grov idé om hva som hadde vært interessant å gjøre for å kombinere sjakk med webutvikling, men skrev ikke ned noe konkret før begynnelsen av dette semesteret.
Da arbeidet med prosjektet begynte for fullt, satte jeg meg ned og lagde en liten skisse av hvordan brukergrensesnittet skulle se ut, og dette hjalp meg med å visualisere hvilken funksjonalitet applikasjonen burde tilby, og avdekket en rekke funksjonelle krav.

Mye av planleggingsarbeidet ble faktisk gjort på arbeidsplassen min, KIWI. Jeg har brukt mye tid der på å tenke gjennom hvordan jeg skulle løse ulike utfordringer, og mange av idéene bak applikasjonen har oppstått her. I perioder hvor det har vært stille og ingenting å gjøre, har jeg skrevet ned kodesnutter og skisser av GUI som jeg har tenkt på mens jeg har jobbet på kvitteringspapir, og tatt med disse hjem for ikke å glemme dem. Noen av de mest utfordrende aspektene ved utviklingen av applikasjonen har blitt løst gjennom mye tenking på jobb, og enkelte deler av implementasjonen har faktisk vært direkte avskrift fra disse kvitteringsrullene.

Ellers har brukergrensesnittet vært sterkt inspirert av sjakktjenesten Lichess (http://lichess.org). Mange av idéene har oppstått som følge av å ha sett dem her, og fargetemaet på sjakkbrettet er tilnærmet identisk med min implementasjon.

Når det gjelder planleggingen av sjakkmotoren har arbeidet vært svært enkelt, ettersom sjakkreglene stort sett har vært uendret i århundrer, og reglene er godt kjent av meg i tillegg til å være godt dokumentert i litteratur og på internett.

3.2 Kravspesifikasjon

Dette er den endelige kravspesifikasjonen etter flere runder med vurderinger og endringer, nødvendig med hensyn til utfordringene beskrevet i avsnitt 2.6.

- Brukergrensesnittet skal være veldig enkelt og forståelig. Brukere skal kunne opprette partier med innstillinger etter eget ønske ved hjelp av noen få klikk. Tilsvarende skal brukere kunne delta i eksisterende partier ved hjelp av kun noen få klikk.
- Brukere skal kunne invitere andre personer til eksisterende partier, ved at det genereres en unik URL som brukeren kan sende til andre.
- Applikasjonen skal støtte flytting av brikker både gjennom individuelle klikk på felter i sjakkbrettet, og ved klikk-og-dra-funksjonalitet for at trekk kan skje raskt og effektivt.
- Brukere skal kunne se en liste over sjakkpartier de deltar i, og kunne klikke i listen for å raskt kunne bytte mellom dem. Brukere skal dermed også kunne delta i flere partier samtidig.
- Brukere skal kunne se en liste over åpne partier, enkelt kunne klikke på dem for å bli med i vilkårlige partier.
- Brukere skal få oppdateringer om partiene sendt i sanntid, slik at partiene kan spilles uten unødvendig venting.
- Applikasjonen skal implementere samtlige sjakkregler, og fungere som forventet selv av profesjonelle sjakkspillere.

3.3 Prosjektets tre deler

Jeg valgte tidlig i utviklingsprosessen å dele opp arbeidet i tre logiske deler. Felles for alle delene er at de er skrevet i JavaScript på plattformen Node.js, en serverimplementasjon av JavaScript-motoren V8, som også brukes i Google sin nettleser Chrome. De tre delene ble utviklet hver for seg i henhold til fremdriftsplanen vist i avsnitt 2.5, men alle modulene kommuniserer tett med hverandre når applikasjonen kjører. Jeg valgte å gjøre denne inndelingen fordi det gjorde det lettere å fokusere på en ting om gangen, og gjorde arbeidet mer strukturert.

3.3.1 Sjakkmotoren

Sjakkmotoren er selve kjernen i applikasjonen, og er enkelt forklart en implementasjon av sjakkreglene i JavaScript. Sjakkmotoren ble utviklet som en egen modul, og et av målene for denne var at den i tillegg til å skulle tas i bruk av webserveren også skulle fungere godt som en frittstående sjakkmotor i andre applikasjoner.

Sjakkmotoren sørger for logikken bak sjakkspillet, og inkluderer klasser for sjakkpartier, sjakkbrett og sjakkbrikker. Den styrer gangen i et sjakkparti, og holder styr på brettet, stillingen i sjakkpartiet, hvem sin tur det er til å trekke, og alle andre relevante opplysninger som teknisk direkte vedgår sjakkpartiet. Implementasjon av hvordan de ulike sjakkbrikkene skal kunne flytte seg eller ta andre brikker er en av de mest sentrale aspektene ved denne modulen.

Sjakkmotoren utvikles objektorientert, og vil hovedsakelig inneholde en rekke klasser. Jeg valgte å gjøre dette fordi jeg følte at det ville bygge videre på oppgaven vi hadde i «Objektorientert programmering», og på den måten var jeg allerede i kjent territorium.

Utover dette har sjakkmotoren blitt bygget fra bunnen av, uten bruk av andre biblioteker eller moduler som allerede finnes ute på nett.

3.3.2 Webserveren

Webserverens hovedoppgave er å ta i bruk sjakkmotoren som en modul i Node.js, og bruke denne til å opprette og styre ulike partier basert på HTTP-forespørsler fra brukere. Den holder styr på alle brukerne og hvilke partier de er involvert i ved hjelp av sesjoner, og koordinerer trekk og andre hendelser med involverte brukere. Eksempelvis vil webserveren ved mottak av HTTP-forespørsel for å gjøre et trekk, kalle på riktig metode fra sjakkmotoren og deretter sende informasjon tilbake til klienten om ny posisjon, brettoppstilling og andre relevante data. På denne måten er webserveren kjernen i applikasjonen i med tanke på kommunikasjon mellom spillere, i og med at all kommunikasjon foregår via denne serveren.

Serveren er også utvikles som en Node.js-applikasjon, og jeg har valgt å ta i bruk en rekke ulike moderne biblioteker og rammeverk for å gjøre arbeidet enklere.

- GraphQL

  Webserveren benytter seg av «GraphQL», et relativt nytt spørrespråk for utveksling og manipulering av data, som er utviklet av Facebook. Dette er implementert både på webserveren og på webklienten, og tar seg av HTTP-forespørsler som et slags bindeledd mellom de to. Jeg har valgt å bruke biblioteket «graphql-yoga» som serverimplementasjon av GraphQL. Et viktig aspekt ved dette systemet er at det muliggjør overføring av data i sanntid, noe som er essensielt i dette tilfellet.
- Express

  Express er et veldig enkelt rammeverk for å opprette HTTP-servere i Node.js. GraphQL-biblioteket «graphql-yoga» bruker dette bak kulissene og tar seg på denne måten av all HTTP-kommunikasjon.

I tillegg til disse har jeg brukt en rekke små moduler som tar seg av andre ting som f.eks. sesjoner (express-session), generering av unike ID-koder (alphanumeric-id), og lignende.

3.3.3 Webklienten

Jeg har valgt å la webklienten være så tynn som mulig, noe som vil si at dens oppgaver er begrenset til å vise frem sjakkbrettet og annen informasjon fra webserveren, og å muliggjøre trekk for brukerne ved hjelp av eventhandlere. Den mottar data fra webserveren i sanntid, og oppdaterer fortløpende grensesnittet med oppdaterte data. Jeg har valgt å la webserveren ta seg av sjakkreglene og utregningene som hører med, og dermed foregår det ingen tunge operasjoner på klientsiden av applikasjonen.

Som med webserveren er klienten utviklet som en Node.js-applikasjon, og tar i bruk en rekke moderne rammeverk.

- React

  En av hovedårsakene til at jeg valgte å gå for dette prosjektet var at det ville gi meg en mulighet til å få trening og erfaring med bruk av frontend-rammeverket «React». Dette er også utviklet av Facebook, og er et av de mest brukte rammeverkene på verdensbasis i dag for utvikling av såkalte «Single Page Applications». SPA innebærer at alt innholdet foregår på én enkelt webside, der innholdet oppdateres dynamisk ved hjelp av JavaScript i browseren. På denne måten vil navigering mellom ulikt innhold skje nærmest umiddelbart, og brukeren får en langt bedre opplevelse av websiden.
- Apollo Client

  Apollo Client er en klientimplementasjon av spørrespråket GraphQL, og inneholder i tillegg ekstra funksjonalitet for å lette arbeidet med dataoverføring mellom klient og server. Den gjør det lett å kjøre GraphQL-spørringer mot webserveren, og tilbyr i tillegg caching/mellomlagring av data og overføring av data i sanntid over WebSocket.
- Styled components

  Et lite bibliotek som gjør at man lett kan knytte CSS-stiler til React-komponenter. Jeg valgte å bruke dette biblioteket fordi det kan være utfordrende å få til styling av elementer i React på en ryddig måte. Dette biblioteket nærmest eliminerer dette problemet, og er veldig enkelt å forstå og bruke.

3.4 Erfaringer og vurderinger

Strategien jeg har valgt som går ut på å dele opp arbeidet med applikasjonen i tre hoveddeler har stort sett fungert utmerket. Ved å ha avgrensede tidsperioder der hovedfokus ligger på en av de tre delene har jeg klart å holde en balanse mellom å fokusere på enkeltdeler og å ha en oversikt over prosjektet som helhet.

Innenfor hver av de tre tidsperiodene har jeg tatt for meg små biter av funksjonalitet om gangen, og latt andre biter vente til jeg har gjort meg helt ferdig med dem. For hver av de tre delene har jeg hatt en liste over ting som skal implementeres, og valgt hva som står for tur basert på en vurdering av hvor viktige de er for applikasjonen. Dette fordi det var litt usikkerhet rundt hva jeg hadde tid til å implementere, gitt at jeg ble sittende med prosjektet på egenhånd.

Som nevnt i avsnitt 3.1 har en stor del av planleggingen foregått på min arbeidsplass, og i noen tilfeller har det hendt at jeg har fokusert på annen funksjonalitet enn den som har vært i fokus på det tidspunktet, på grunn av idéer som har oppstått her. Dette har ikke vært noe problem, ettersom disse idéene ofte har hatt med viktige kjernekomponenter å gjøre. For eksempel har mye av logikken bak flytting av brikker blitt kladdet på jobb.

Når det gjelder å vurdere arbeidet opp mot Kanban-konseptet har noe fungert godt, og andre ting fungert dårlig. Til tross for å ha vært alene på gruppe, har jeg klart å følge Kanban-konseptet med tanke på å la utviklingen være drevet av funksjonelle krav, og ikke ha for meg for mye av gangen. Derimot kunne jeg ha vært flinkere til å benytte Kanban-tavlen som hjelpemiddel, og vært enda mer strukturert, men dette var et valg jeg tok da jeg endte opp alene på gruppa. Ved å være alene, forsvinner en stor del av fordelen ved å bruke en slik tavle, ettersom man ikke lenger har det samme kommunikasjonsbehovet, og kan klare å ha oversikt over hele systemet på egenhånd.

4 Systemdokumentasjon

Jeg vil her beskrive mer tekniske detaljer rundt funksjonaliteten til applikasjonen. I første omgang vil jeg gi en generell oversikt over strukturen og hovedflyten i applikasjonen, ved hjelp av både diagrammer og forklaringer. Jeg betrakter det å flytte en brikke som den mest kompliserte operasjonen i applikasjonen, og vil derfor også gå spesielt i dybden på denne.

Merk at kildekoden er tungt kommentert, og jeg vil anbefale å se på denne for å få bedre oversikt og en mer detaljert beskrivelse av hvordan applikasjonen fungerer.

En fungerende demo av applikasjonen vil være tilgjengelig her:

http://chess.cojiro.no

For å teste applikasjonen vil jeg anbefale å åpne to nettleservinduer, hvorav det ene er i inkognitomodus. På denne måten vil vinduene knyttes til hver sin sesjon, og betraktes som to forskjellige spillere av applikasjonen.

I tillegg følger instrukser for å kjøre applikasjonen på egen maskin.

4.1 Installasjonsinstruks

Her følger en oppskrift på hvordan man kan kjøre applikasjonen på egen maskin, ved å bruke den vedlagte kildekoden.

4.1.1 Med Docker

Applikasjonen støtter bruk av containersystemet Docker, og kan startes ved hjelp av en enkelt kommando dersom Docker er installert og konfigurert på maskinen.

Naviger til katalog:

chess-web

Kjør kommando:

docker-compose up

4.1.2 Uten Docker

Applikasjonen kan også kjøres uten Docker. Dette forutsetter at det finnes en lokal installasjon av Node.js og pakkesystemet npm på maskinen.

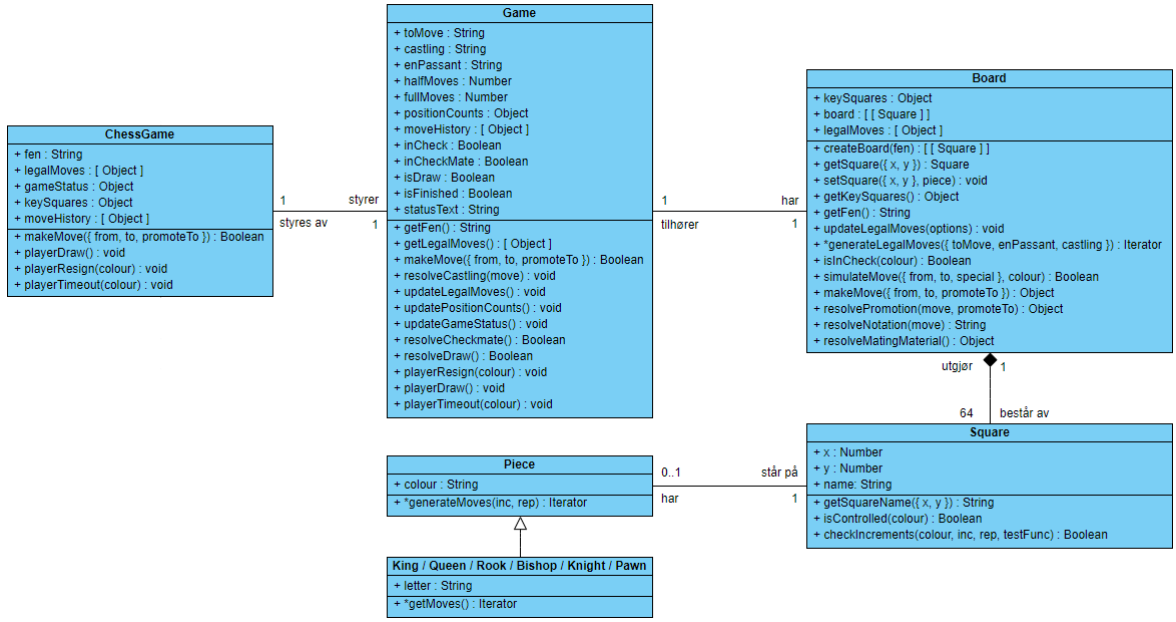
Server	Klient
Naviger til katalog: chess-web/server Kjør kommandoer: npm install npm start	Naviger til katalog: chess-web/client Kjør kommandoer: npm install npm start

4.2 Struktur

4.2.1 Sjakkmotoren

Som nevnt tidligere i rapporten er sjakkmotoren skrevet på objektorienterte prinsipper. Selv om JavaScript ikke i utgangspunktet har et ekte klassesystem, tillater språket at vi definerer klasser som i hovedsak oppfører seg som man skulle forvente fra et objektorientert språk.

Klassediagram



En større versjon av bildet finnes som vedlegg eller ved å klikke her.

Game-klassen kan betraktes som hovedklassen i systemet, og er den som holder styr på det aller meste. ChessGame-klassens oppgave er å fungere som et innkapslingslag, ettersom JavaScript foreløpig ikke har noen offisiell syntaks for å deklarere private variabler.

Sjakkmotoren baserer seg på en såkalt FEN-string, som i sjakkmiljøet er kjent som en tekststreng som beskriver stillingen på brettet, samt informasjon om hvem som er i trekket, hvilke rokademuligheter spillerne har, om det spesielle bondetrekket «en passant» er mulig, samt informasjon om hvor mange trekk som har gått. Applikasjonen bruker denne strengen og analyserer dem for å bygge brettet når partier opprettes. Samtidig vil sjakkmotoren

beskrive stillingen utad ved hjelp av denne notasjonen, og behandles igjen i brukergrense-snittet for å tegne sjakkbrettet for spillerne.

Board-klassen styrer det som har direkte med sjakkbrettet å gjøre, uten å ha noen kjennskap til ting som f.eks. spillerne. Den inneholder operasjoner relatert til ting som å flytte brikker rundt på brettet og å avgjøre om noen står i sjakk, og holder styr på brikkenes posisjon gjennom Square-klassen og deretter Piece-klassen.

Square-klassen representerer felter på brettet, og forteller hvilke brikker som står på dem. Brikkene er representert av superklassen Piece hvorfra de individuelle brikkene arver egenskaper. Brikkene har mulighet til å oppgi hvilke felter de kan flytte til i enhver gitt stilling, og dette er kjernen bak metoden for å generere lovlige trekk på brettet. Den vil beskrives senere i dette avsnittet.

4.2.2 Webserveren

I kjernen av webserveren ligger biblioteket «graphql-yoga», en implementasjon av GraphQL på serversiden i Node.js. Dette biblioteket sørger både for å ta imot HTTP-forespørsler og sende responser tilbake, og for å hente nødvendig data fra riktig kilde og sende dem tilbake.

GraphQL fungerer ved at man definerer egne datatyper og definerer hvilke attributter disse datatypene skal ha, samt attributtenes datatyper. Dette gjøres i denne applikasjonen i filene i «typeDefs»-katalogen. Samtidig definerer man såkalte «resolvers», som er funksjoner som forteller GraphQL hvor data skal hentes for de ulike datatypene og deres attributter. Nytten ved dette systemet er at når man spør etter data, kan man velge hvilke attributter man ønsker. GraphQL vil da kun hente og returnere data for disse attributtene ved hjelp av de tilsvarende resolverne, uten å risikere å måtte hente store mengder unødvendig data som kun forkastes. GraphQL returnerer data i JSON-format, og JSON-objektet vil ha samme struktur som den man bruker i spørringen. Dermed har man som frontend-utvikler stor grad av forutsigbarhet når det gjelder dataen man mottar fra serveren.

I denne applikasjonen sender jeg GraphQL-forespørsler når brukerne gjør operasjoner i frontend, som for eksempel å flytte en brikke. Da vil GraphQL på serveren motta og behandle denne forespørselen, se på hvilke parametere som ble sendt med og sende disse til resolveren. Videre vil resolveren finne riktig sjakkparti basert på IDen sendt fra frontend, og utføre den ønskede operasjonen i sjakkpartiet. Deretter vil GraphQL avgjøre hvilke data som skal sendes tilbake til klienten, basert på de attributtene som ble definert i spørringen.

Også verdt å nevne er behandlingen av sesjoner i webserveren. Ved hjelp av biblioteket «express-session» legger jeg inn en unik ID for hver bruker som besøker siden. Denne IDen brukes så i GraphQL-resolverne til å gjenkjenne brukeren og sørge for at kun de brukerne som er involvert i et parti har mulighet til å påvirke det. Se kildekoden for nærmere forklaring.

4.2.3 Webklienten

Når det gjelder webklienten, er denne skrevet ved hjelp av frontendrammeverket «React», utviklet av Facebook. React fungerer ved at man definerer egne GUI-komponenter, som egentlig bak kulissene er funksjoner som returnerer HTML-elementer. Komponenter kan kalles fra andre komponenter, og man kan også sende inn data som parametere til komponentene og på denne måten endre innholdet eller strukturen på de returnerte HTML-elementene. En typisk React-applikasjon vil ha en rotkomponent som kaller andre komponenter på en strukturert måte, og på denne måten bygger en hel webside i JavaScript. Til slutt settes den genererte HTML-koden inn i det virkelige DOM-et.

Fordelen ved bruk av React er at man oppnår det man kaller en «Single Page Application». Dette er en webside som kun bruker én ordentlig side, der innholdet oppdateres fortløpende ved hjelp av JavaScript. Ved å oppdatere komponentene ved bruk av event listeners og andre datakilder oppnår man en dynamisk webside som oppdateres lynraskt. Dermed får brukerne en mye bedre brukeropplevelse enn om man måtte laste hver enkelt side på tradisjonelt vis.

I sjakkapplikasjonen har jeg bygd hele komponentstrukturen i React, og jeg ønsker å trekke frem strukturen bak sjakkbrettkomponenten her.

Sjakkbrettkomponenten består først av en rammekomponent, som har som oppgave å indikere feltenes koordinater. Kolonnene navngis fra A til H, mens radene navngis fra 1 til 8. Det interessante med disse koordinatene er at dersom brukeren som laster siden spiller med svarte brikker, vil rekkefølgen på disse tallene og bokstavene snus. Effekten av dette er at spilleren ser brettet som om han eller hun befinner seg på den andre siden. Dette oppnås ved å hente data om partiet, og dermed finne ut om den aktuelle brukeren har svarte brikker i dette partiet. Avhengig av dette vil bokstav- og tallkomponentene tegnes inn i riktig rekkefølge.

Sjakkbrettkomponenten består videre av 64 underkomponenter med navn «Square», som representerer feltene på brettet. Disse feltene genereres på bakgrunn av partiets FEN-string, som hentes fra webserveren når siden for partiet lastes. En funksjon løper igjennom hvert tegn i denne strengen, og avgjør hvilken komponent som skal tegnes inn på bakgrunn av dette. Om det står en brikke på feltet, tegnes det i tillegg inn en «Piece»-komponent på feltet, og basert på hvilken brikke som skal tegnes inn, endres Piece-komponentens bakgrunnsbilde slik at det samsvarer med den aktuelle brikken. Dersom brukeren som ser på siden har svarte brikker i det aktuelle partiet, vil feltene tegnes inn i motsatt rekkefølge, på samme måte som med tallene og bokstavene nevnt tidligere.

Hvert av feltene på sjakkbrettet er tilknyttet en event handler, som fyres av når brukeren klikker på feltet. En annen event handler fyres av når brukeren slipper museknappen, og på denne måten har jeg implementert klikk-og-dra-funksjonalitet for brikkene. Så lenge museknappen er nede, tegnes det inn en ekstra komponent som følger musepekeren, og har samme utseende som den brikken som ble klikket på. Slik oppnår jeg at brikken ser ut som om den blir flyttet rundt på brettet. Når knappen slippes, registreres det hvilket felt musepekeren befant seg over, og på bakgrunn av de to feltene som var involvert i klikket, fyres det av en funksjon som ber om at brikken flyttes på serveren.

For styling av komponenter har jeg brukt biblioteket «Styled components». Dette biblioteket gjør det lett å opprette komponenter og knytte CSS-stildefinisjoner til dem. Det nyttige ved dette biblioteket er at du kan variere CSS-stilene basert på parameterne man sender inn til komponenten, og dermed få dynamisk stylede komponenter. Disse komponentene kan da brukes videre i applikasjonen på vanlig måte, og man oppnår en veldig enkel og effektiv måte å style komponentene på, noe som vanligvis kan være noe utfordrende i React.

For kommunikasjon med serveren bruker jeg biblioteket «Apollo Client», som er en GraphQL-klient med mange nyttige tilleggsfunksjoner. For å ta dette i bruk må jeg først opprette en instans av Apollo-klienten, der jeg oppgir URL-en til serveren jeg ønsker å knytte meg til. Deretter gjør jeg klienten tilgjengelig for bruk i alle komponentene i applikasjonen min, og kan videre bruke den i de komponentene som har behov for data. Jeg skriver egne GQL-spørringer basert på de dataene jeg trenger, og kjører dem når komponenten lastes inn. Alternativt kan jeg knytte spørringene til elementer (knapper) på websiden og kontrollere spørringene manuelt. Når serveren sender data tilbake vil komponenten kunne tegne innholdet som tenkt.

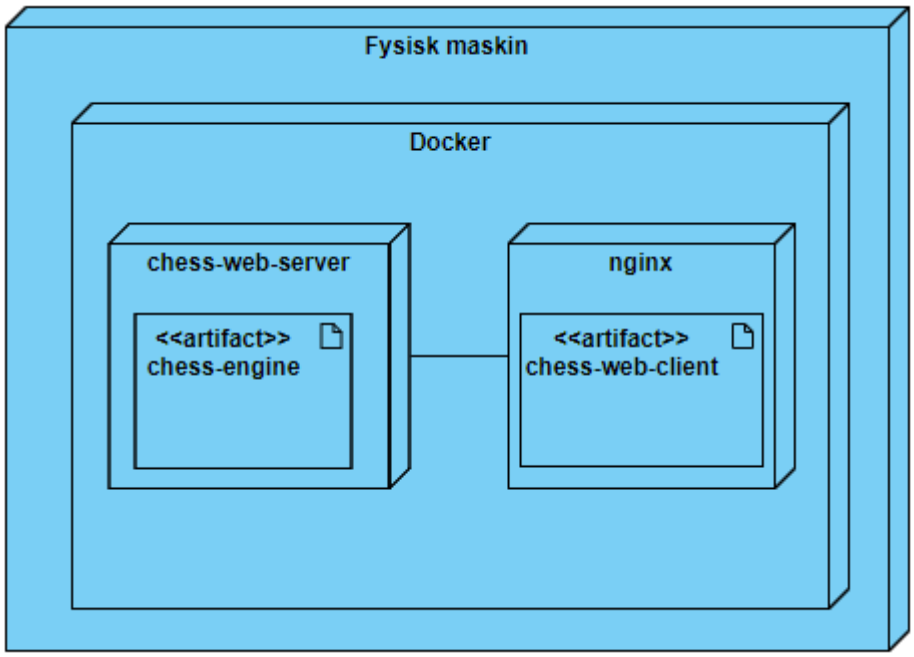
4.2.4 Docker

Jeg bestemte meg ganske sent i utviklingsprosessen for å prøve å ta i bruk programmet «Docker». Dette programmet gjør at man kan kjøre applikasjoner i såkalte virtuelle containere, der applikasjonene kan kjøre isolert med kun den programvaren den trenger for å fungere. Containere har mye til felles med virtuelle maskiner, men forskjellen ligger i at containere kjøres av den samme operativsystemkjernen, og krever dermed mindre totalt sett av vertsmaskinen enn tradisjonelle virtuelle maskiner.

Docker fungerer kort fortalt ved hjelp av såkalte Dockerfiles, som forteller Docker hvordan en applikasjon eller en modul skal bygges og pakkes til et image. Disse imagene kan videre kjøres i Docker-containere. Man har også mulighet til å lage større «pakkeløsninger» ved å sette sammen containere i virtuelle nettverk, og la dem kommunisere med hverandre over dette nettverket.

Sjakkapplikasjonen har støtte for dette, og ved hjelp av Docker-modulen «docker-compose» kan hele applikasjonen bygges og startes i produksjonskonfigurasjon i løpet av svært kort tid på enheter som har Docker installert.

Utplasseringsdiagram



Her er en grov skisse av hvordan strukturen i applikasjonen ser ut når den kjøres i Docker. Applikasjonens webserver kjører her som en egen container, og sjakkmotoren brukes som en modul av denne. Den andre containeren er basert på HTTP-serveren nginx, som betjener en kompilert utgave av React-applikasjonen som statisk HTML.

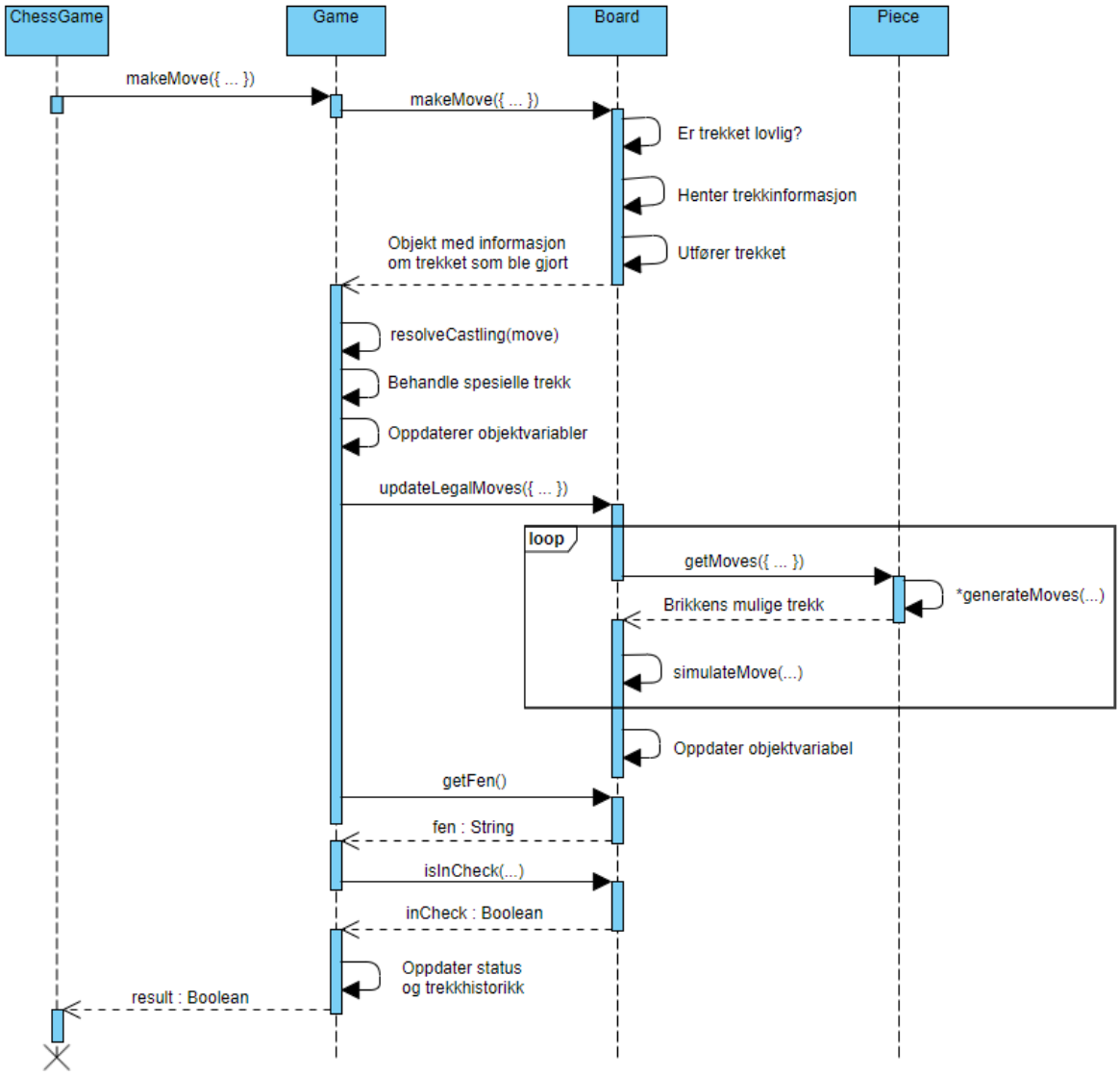
Nginx vil ta imot forespørsler utenfra på port 80, og basert på URL-en som etterspørres vil den enten sende tilbake React-applikasjonen, eller omdirigere trafikken til sjakkserveren i de tilfeller der forespørselen er en GraphQL-operasjon.

Konfigurasjonsfilen for nginx finnes i applikasjonens kildekode.

4.3 Flytting av brikker

Den viktigste og samtidig mest kompliserte operasjonen i hele sjakkapplikasjonen er den som angår flytting av brikker. Når brikker flyttes er alle ledd i applikasjonen i bevegelse, og derfor ønsker jeg å gå gjennom hele prosessen og beskrive den i detalj.

Sekvensdiagram for utførelse av trekk i sjakkmotoren



En større versjon av bildet finnes som vedlegg eller ved å klikke her.

Det hele begynner ved at en spiller klikker eller drar en brikke fra et felt til et annet i frontend, og utløser GraphQL-mutasjonen «makeMove». Denne sendes til webserveren, som behandler forespørselen i makeMove sin resolverfunksjon. Der hentes parti-ID ut fra forespørselens parameterliste, og søker i den interne listen over aktive partier etter partiet med denne IDen. Når partiet er funnet, kjøres metoden makeMove({ … }) i ChessGame-objektet som representerer partiet. Sekvensdiagrammet på forrige side begynner her.

ChessGame-objektet videresender metodekallet til Game-objektet, som i sin tur sender det videre til Board-objektet, der behandling av trekket kan begynne. Det første som sjekkes er om trekket faktisk er lovlig, ved å se om trekket finnes i brettobjektets interne liste over lovlige trekk som ble generert etter forrige trekk. Den henter informasjon om trekket dersom det er lovlig, og utfører trekket på brettet. Informasjon om trekket som ble gjort returneres til Game-objektet for videre behandling.

Dersom trekket var gyldig er det nå Game-objektet sin tur til å behandle det. Det undersøker om trekket var et konge- eller tårntrekk, og i så fall må rokademulighetene for spilleren oppdateres. En rekke betingelser sjekkes videre, og en rekke objektvariabler oppdateres på bakgrunn av disse betingelsene. Etter at dette er utført skal listen over lovlige trekk oppdateres slik at neste trekk kan behandles senere. Denne prosessen er også noe komplisert, og dekkes i detalj i neste avsnitt.

Etter at listen over lovlige trekk er oppdatert, er det igjen Game-objektet som kjører. Det som gjenstår nå er å bygge en ny FEN-string som kan returneres, samt annen generell info som skal returneres til spillerne. Gjennom metodekallene getFen() og isInCheck(…) i Board-objektet hentes nødvendig informasjon, og til slutt legges trekket inn i listen over tidligere trekk i partiet. Game-objektet returnerer «true» til ChessGame-objektet, som i sin tur returnerer «true» til resolvermetoden som startet prosessen i sjakkmotoren.

Jeg har valgt å la tidskontroll for spillerne være utenfor sjakkmotoren, og heller la webserveren ta hånd om dette. Dette fordi det finnes ulike måter å implementere dette på, og jeg ønsker å la det være opp til brukeren av sjakkmotoren å finne en løsning. Jeg har selv løst det ved å sammenligne nåværende tidspunkt med tidspunktet for forrige trekk i partiet.

Når trekket er utført i sjakkmotoren vil resolvermetoden på webtjeneren stoppe timeren for spilleren som utførte trekket, og samtidig starte en ny timer for den andre spilleren. I praksis betyr dette at dersom en av spillerne går tom for tid, vil webapplikasjonen automatisk kalle playerTimeout-metoden i ChessGame-objektet i sjakkmotoren, og avslutte spillet etter gjeldende regler.

Etter at trekket er ferdigbehandlet, vil GraphQL sende en melding om trekket via WebSockets til alle klienter som er involvert i partiet. Slik oppfylles kravet om sanntidsoppdateringer i GUI.

Når webklienten mottar meldingen om trekket, vil grensesnittet oppdateres med nye data, og reflektere tilstanden til partiet slik det er i sjakkmotoren. Etter dette overlates kontrollen over brikkene til motspilleren, og applikasjonen er klar til å ta imot neste trekk.

4.4 Generering av lovlige trekk

Generering av lovlige trekk starter ved at en instans av Board-klassen løper gjennom alle feltene på brettet og finner feltene som har brikker som tilhører spilleren som er i trekket. Deretter kjøres metoden getMoves({ … }) for hver brikke i utvalget. På denne måten spørres brikkene selv om å oppgi hvilke felter de kan flytte til, basert på regler knyttet til brikketypen. For hvert trekk som returneres fra brikkeobjektet, kjører brettobjektet en simulering av trekket for å undersøke om trekket er lovlig, altså at trekket ikke setter egen konge i sjakk, noe sjakkreglene ikke tillater. Dersom alt er ok, legges trekket inn i listen over lovlige trekk.

Når brikkene blir bedt om å oppgi hvilke felter de kan flytte til, har jeg skrevet en generell metode som fungerer for de fleste brikkene. Metoden baserer seg på at hver brikkes bevegelse kan beskrives av et sett med inkrementer. For eksempel kan en løper kun bevege seg diagonalt, og bruker derfor kun diagonale inkrementer. Et inkrement tilsvarer en retning en brikke kan bevege seg i, og består av en x-verdi og en y-verdi som representerer forflytningen i kolonne- og radretning per enhet. For eksempel vil inkrementet { x: 1, y: 1 } tilsvare en bevegelse diagonalt opp til høyre. I tillegg tar metoden en parameter som avgjør om inkrementet kan repeteres, altså om brikken kun kan flytte et felt om gangen, eller så langt man ønsker.

For hver retning (hvert inkrement) en brikke kan flyttes, vil metoden bevege seg lenger og lenger, helt til den når kanten av brettet eller en annen brikke. For hvert ledige felt på veien vil brikken oppgi målfeltet som et lovlig trekk, og dersom den skulle støte på en annen brikke, vil den kun oppgi målfeltet som et lovlig trekk dersom denne brikken tilhører motstanderen, ettersom man ikke kan ta sine egne brikker. I så fall vil trekket regnes som lovlig, og markeres som et slagtrekk, altså at brikken slår ut en annen brikke.

5 Erfaringer og konklusjoner

Ved prosjektinnlevering betrakter jeg prosjektet som fullført, basert på kravspesifikasjonen beskrevet i avsnitt 3.2.

- Brukergrensesnittet er enkelt og oversiktlig, både etter mitt skjønn og basert på tilbakemelding fra personer som har testet applikasjonen. Det går raskt å gjøre trekk, og å opprette og bli med på partier er gjort på et øyeblikk.
- Applikasjonen støtter invitasjon til partier gjennom deling av unik generert URL.
- Både klikk-og-dra-funksjonalitet og individuelle klikk kan brukes til å gjøre trekk.
- Brukere kan både se hvilke partier de deltar i og en oversikt over åpne partier. Disse listene kan brukes til å navigere mellom ulike partier kjapt og enkelt.
- Applikasjonen støtter oppdateringer i sanntid.
- Alle sjakkregler er implementert, med et lite unntak når det gjelder reglene for bondeforfremmelse. Bønder skal kunne forfremmes til både tårn, springer, løper og dronning, men applikasjonen støtter kun forfremmelse til dronning per dags dato.

Til tross for det lille unntaket når det gjelder bondeforfremmelse, er jeg svært fornøyd med prosjektresultatet. Ettersom forfremmelse til andre brikker enn dronning skjer svært sjeldent, vil jeg påstå at sjakkreglene i akseptabel grad er hensyntatt.

Når det gjelder mine personlige mål for prosjektet, føler jeg også at disse er oppnådd. Jeg har fått svært god trening i bruk av moderne rammeverk og biblioteker som React og GraphQL, da spesielt med tanke på at jeg har tatt dem i bruk i et forholdsvis stort prosjekt på en god måte, da jeg tidligere kun har brukt dem i små prosjekter eller for å prøve dem ut. Jeg føler også at jeg har klart å holde arbeidet strukturert og jevnt, til tross for de utfordringene jeg nevnte innledningsvis med tanke på å være alene om et slikt prosjekt. På denne måten synes jeg også at jeg har lært en god del om å arbeide etter en godt etablert arbeidsmetodikk.

Ved prosjektstart hadde jeg en idé om å lagre ferdigspilte partier i en database, slik at man ved senere besøk til samme URL ville ha fått opp sluttstillingen og trekkhistorikken for partiet. Jeg bestemte meg senere for å droppe dette kravet grunnet begrenset tid. Dessuten var ikke dette særlig relevant for måloppnåelsen i prosjektet, ettersom den ikke ville ha fylt noen av de funksjonelle kravene jeg satt igjen med etter å ha revurdert arbeidsmengden. Om jeg hadde litt mer tid er dette et av punktene jeg nok hadde implementert fullt ut.

Noe annet jeg ville ha fokusert mer på om jeg hadde hatt mer tid, er designet på websiden. Jeg er i utgangspunktet svært dårlig på webdesign, og derfor har jeg valgt å gå for et veldig minimalistisk og kanskje kjedelig design. Dette er noe jeg ønsker å bli bedre på, og jeg hadde definitivt brukt mer tid på å gjøre applikasjonen «penere» dersom det var tid til overs.

Med den erfaringen jeg har fått gjennom arbeidet med dette prosjektet ser jeg definitivt ting jeg ville ha gjort litt annerledes om jeg skulle ha gjort det igjen, men helhetlig sett er jeg veldig fornøyd med prosjektresultatet. Jeg kan godt tenke meg å jobbe videre med applikasjonen etter at prosjektet er ferdig, og utbedre de manglene jeg har påpekt i dette avsnittet.

6 Referanser

Demo

Applikasjonen kan testes her: http://chess.cojiro.no

Prosjektrelatert

Prosjektweb:	https://github.com/svgnAyure/Prosjektweb
Kildekode sjakkmotor:	https://github.com/svgnAyure/chess-engine
Kildekode webapplikasjon:	https://github.com/svgnAyure/chess-web

Sentrale websider

Lichess:	https://lichess.org/
GitHub:	https://github.com/
Sjakkregler:	https://en.wikipedia.org/wiki/Rules_of_chess

Sentral programvare

Visual Studio Code:	https://code.visualstudio.com/
Node.js:	https://nodejs.org/en/
Docker:	https://www.docker.com/
NGINX:	https://www.nginx.com/

Node.js-moduler

React:	https://reactjs.org/ https://github.com/facebook/react
GraphQL:	https://graphql.org/ https://github.com/graphql/graphql-js
graphql-yoga:	https://github.com/prisma/graphql-yoga
express:	https://github.com/expressjs/express
express-session:	https://github.com/expressjs/session
Apollo Client:	https://github.com/apollographql/apollo-client
alphanumeric-id:	https://github.com/cfj/alphanumeric-id
styled components:	https://github.com/styled-components/styled-components