Filmatory

APP2000

Applikasjonsutvikling og Web

Universitetet i Sørøst-Norge

Gruppe 10

Wiseflow gruppenummer 1.

Govert Dahl - Studentnr: 233565

Sigve Aleksander Enoksen Eliassen - Studentnr: 233511

Sivert Barsgård Heisholt - Studentnr: 233518

Ørjan Dybevik - Studentnr: 233530

Innleveringsdato: 30.05.2021

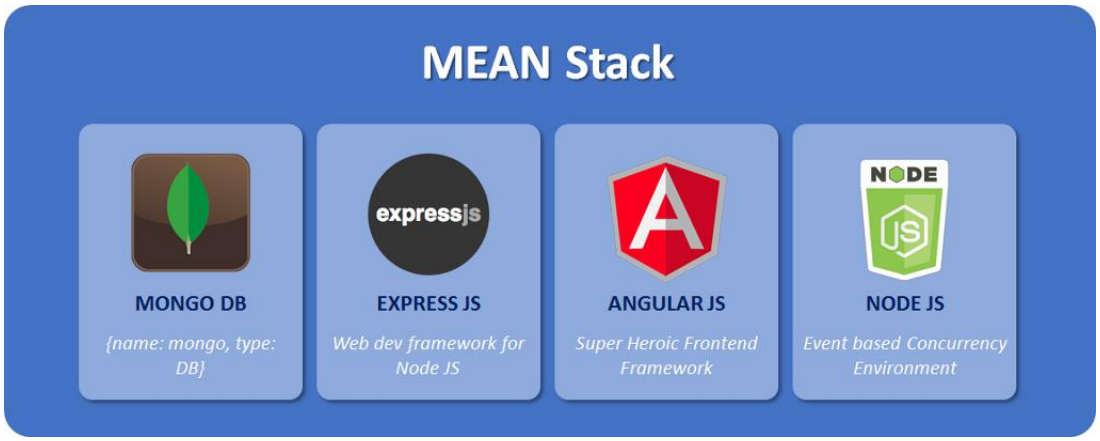
Innhold

Problemstilling

Produktet skal kunne løse problemer som er tilknyttet filmer/serier. En plass der brukere kan kunne gå inn for å finne filmer/serier, filtrere søk, lese anmeldelser, se statistikker i fine grafer og se anbefalinger. Kort sagt for å fylle inn manglende informasjon til brukeren. Hva om du ikke vet navnet på en skuespiller? Ja da kan du bare gå inn på Filmatory og søke opp filmen skuespilleren spilte i. Da skal du få opp liste over skuespillere i den filmen.

Teknologi

Da vi skulle bestemme oss for hvilke teknologier vi skulle bruke, så var det veldig viktig for oss at det var noe som var spennende og gav oss mye læringsutbytte. Vi så først på LAMP stack, men denne hadde vi allerede prøvd oss på tidligere. Derfor endte vi opp med MEAN stack. Denne ser slik ut:



Her har vi MongoDB som databasesystem, denne kjører vi på Mongo Atlas. Vi bruker Express JS som framework for NodeJS som er runtime. AngularJS er front-end, og vi bruker da standard javascript her. Dette gjør at vi får javascript på både front-end og back-end.

På toppen av dette, så bruker vi forskjellige bibliotek. Angående Express så har vi pug som gjør det mulig at vi kan lage front-end mye raskere og bedre. Vi kombinerer da dette med UiKit som er modulær front-end rammeverk. Denne ligner på Bootstrap, men er litt mer moderne. Andre små bibliotek vi bruker er: jQuery, HighChartsJs og Socket.IO. HighChartsJs er for fine diagrammer på front-end. Socket.IO gir oss muligheten for realtime kommunikasjon mellom front-end og back-end.

Vår applikasjon skal håndtere store mengder data. Dette er data som vi ikke kan lagre og derfor må vi hente denne dataen fra en API. Det er hovedsakelig TMDB sin API vi bruker til

applikasjonen. Under kan du se en del av koden som gjør det mulig for oss å hente informasjonen, denne koden søker etter en film ved hjelp av tittelen:

```
Tmdb.prototype.getMovieResults = function getMovieResults(movieTitle) {
    var url = `https://api.themoviedb.org/3/search/movie?api_key=${this.token}
&query=${movieTitle.replace(/ /g, "+")}`;
    return fetch(url).then(res => {
        makeLog(res, url);
        return resultHandler(res);
    })
}
```

Vi lager en funksjon som heter getMovieResults, denne tar inn movieTitle som parameter. Fetch kommer fra en pakke som heter node-fetch, den sender en http request til API'en. Vi får da et svar i JSON format som vi kan bruke.

Ekstra bibliotek

Node.js tilbyr mange bibliotek som er laget av andre personer. Vi har brukt en del pakker som har gjort arbeidet vårt mye lettere og spart oss for mye tidsbruk. Denne funksjonen heter NPM (Node package manager). Se vedlagt fil «Bibliotek» for en oversikt over alle NPM-pakker vi har brukt og beskrivelse av disse.

Systembeskrivelse

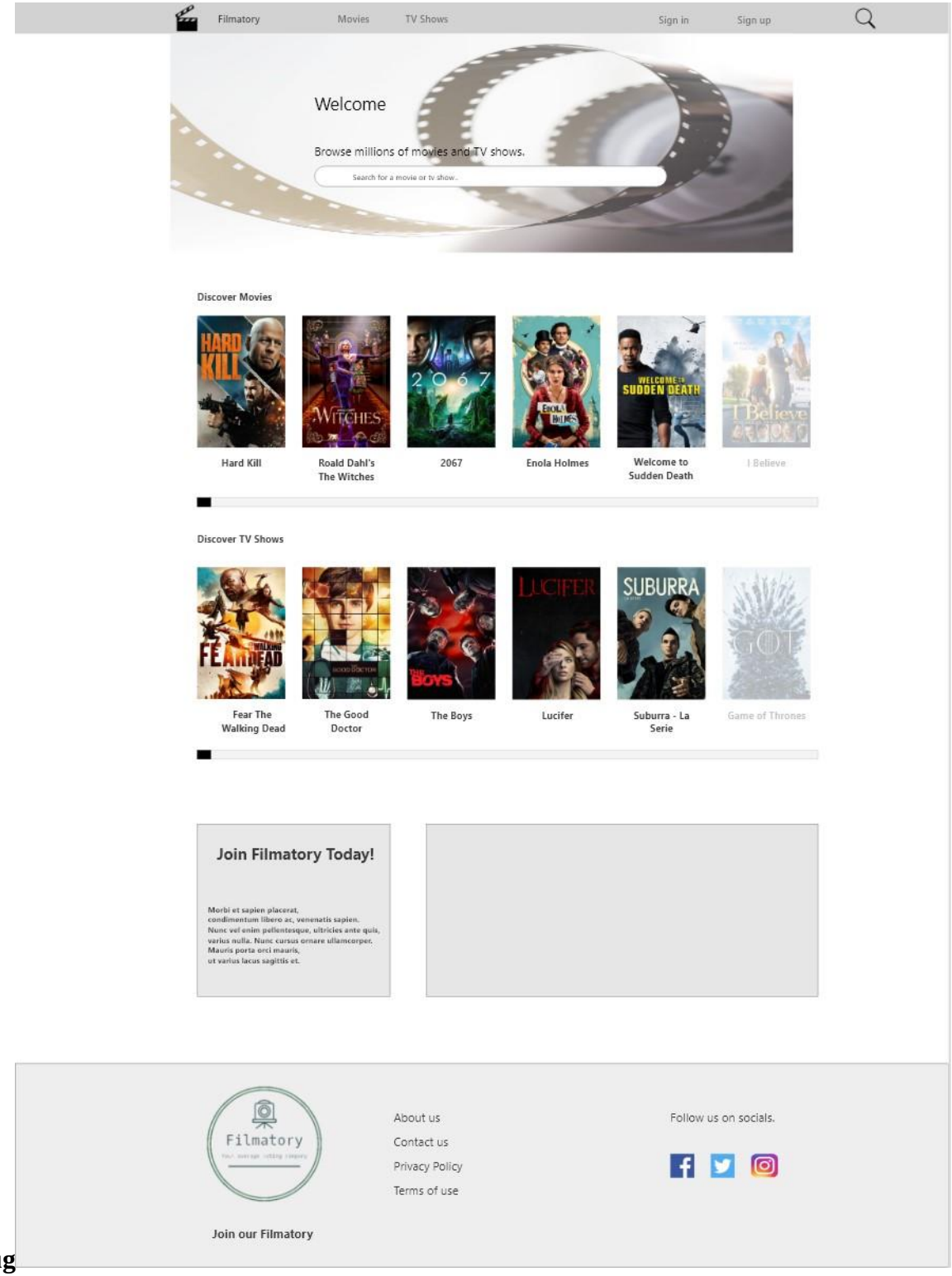
Overordnet beskrivelse

Løsningen er bygd opp på den måten at klienten sender en forespørsel til serveren. Serveren tar deretter imot denne forespørselen og ruter til riktig sted. I vårt tilfelle så har vi controllere som tar seg av GET og POST forespørsler. Vi har også «middlewares» som kjøres før controlleren kjøres, dette er for å gjøre ting som skal skje før controllerne. Dette kan for eksempel være språk for alle sider, session, bruker eller sjekker. Controlleren sender til slutt informasjonen til klienten som gjør at brukeren kan se nettsiden. Løsningen er også bygd opp ved hjelp av websockets som gjør det mulig for sanntidskommunikasjon mellom klient og server etter at nettsiden er prosessert.

Brukergrensesnitt

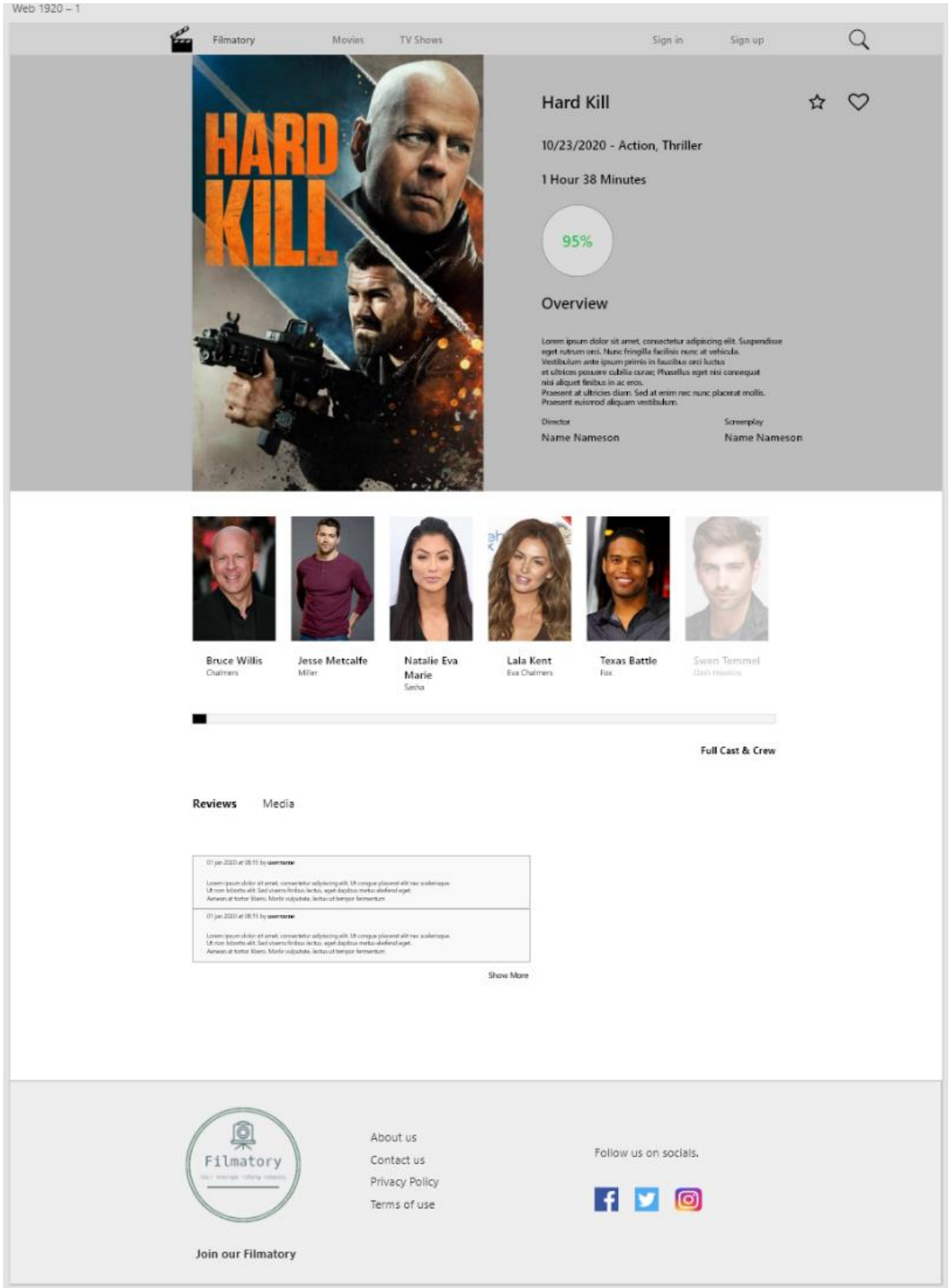
Før vi startet prosjektet, lagde vi en prototype av brukergrensesnittet i Adobe XD. Slik fikk vi dannet et bilde av hvordan det ville se ut, og hva slags funksjoner vi kan produsere. Bilde 1 og 2 viser hvor det visuelle startet med Filmatory. Når vi først skulle designe web-applikasjonen med HTML/CSS, brukte vi disse to designene som vi laget tidligere.

Brukergrensesnittet er stort sett laget med UiKit som er et lett og modulært front-end rammeverk. For å få det resultatet vi ønsket, skrev vi over enkelte funksjoner som vi ikke ønsket fra UiKit. Bilde 3 og 4 er bilde av frontsiden og filmsidens endelige resultat. De har store likheter fra design til endelig produkt.

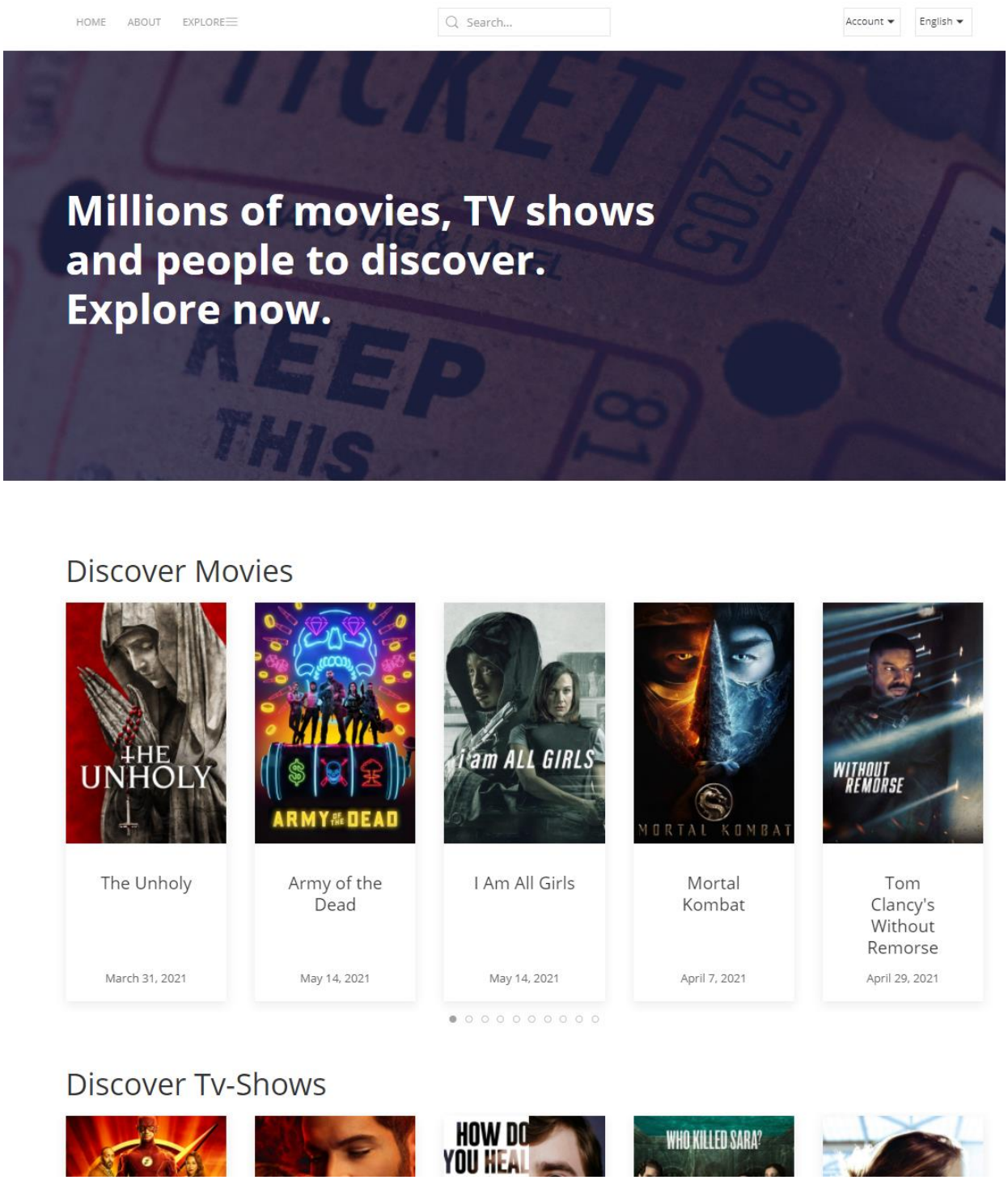


ug

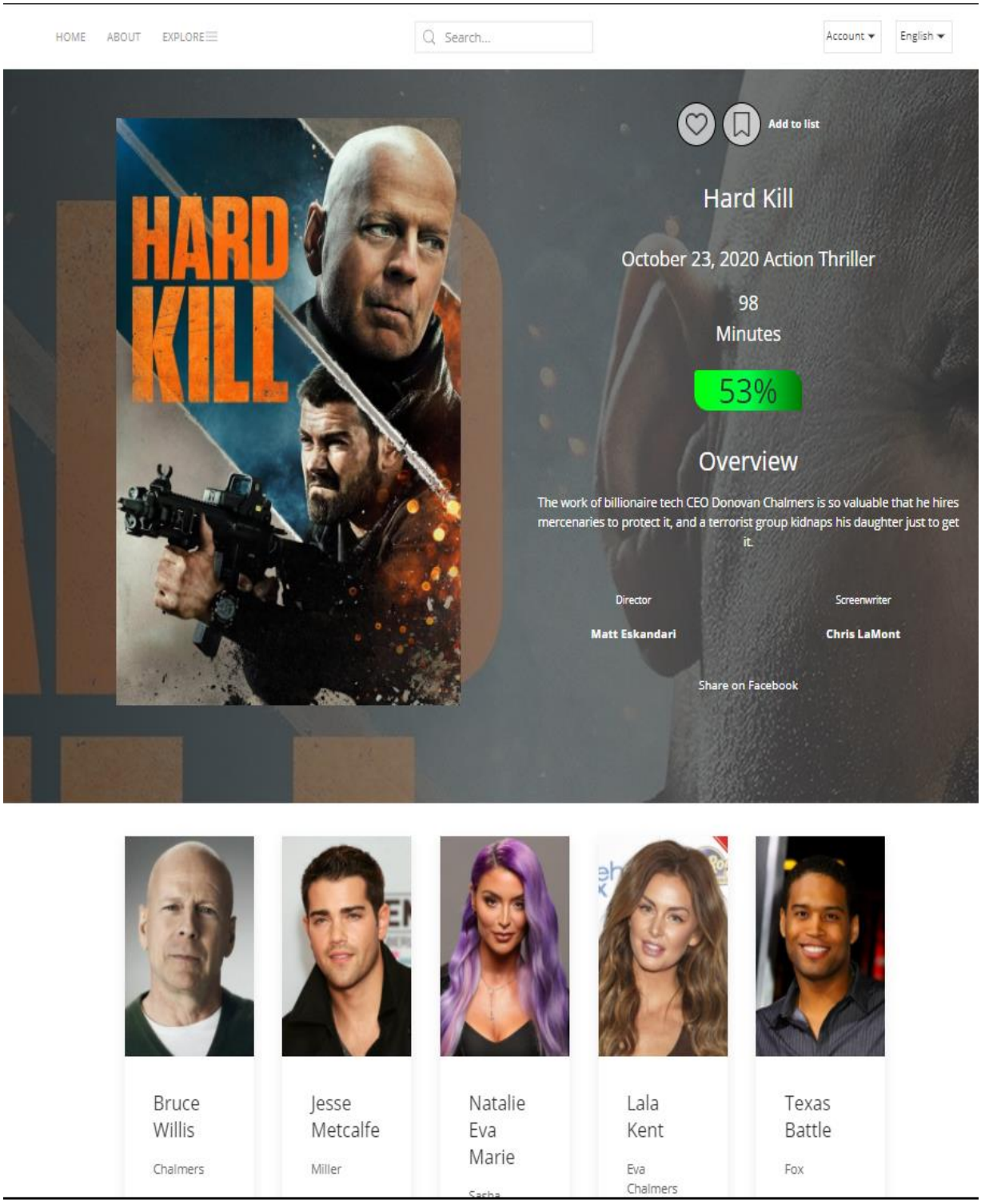
Bilde 1 - Design hvordan vi så for oss frontsiden



Bilde 2 - Design hvordan vi så for oss designet inne på en film/serie



Bilde 3 – Hvordan frontsiden endte opp med å se ut



Bilde 4 – Hvordan filmsiden endte opp med å se ut

Klientdelen

Pug

Vi har valgt å bruke Pug i stedet for *HTML* til å utvikle front-end til Filmatory. I bunn og grunn er *Pug* noe som gjør front-end programmering lettere ved at det er kortere og enklere å skrive. En trenger for eksempel ikke å benytte seg av så mange symboler, og når man indenter koden blir det automatisk en div, som også gjør at koden blir mer oversiktlig. *Pug* fungerer ved siden av *Node*, så man kan skrive *Javascript* rett i Pug-koden. Da kan man lett introdusere for eksempel sjekker (if's) i koden og vi kan enklere sette inn data fra API.

UiKit og CSS

Vi har valgt å bruke *UiKit* som rammeverket til å bygge det meste av front-end koden vi har på Filmatory. UiKit er et lettvektig og modulært rammeverk som gjør det enklere og raskere å gjennomføre front-end funksjoner som f.eks slidere, slideshows, modal og mye mer. UiKit har mange forhåndsdesignede funksjoner, men hvis vi ønsker å endre utseendet på disse front-end-funksjonene kan lett overstyres med CSS. Dette gjøres på samme måte som man ville brukt for å designe noe annet med CSS. UiKit sammen med Pug gjør ar front-end-delen på Filmatory beholder sitt design gjennom hele siden, samtidig som det gjør nettsiden både lett å lage samt lite tidskrevende sammenlignet med mer tradisjonelle metoder som kun HTML og CSS.

Javascript

Vi har brukt helt vanilla JavaScript på front-end. Dette er fordi at flere av gruppemedlemmene ikke hadde mye JavaScript-erfaring. Når vi først er bedre kjent med vanilla JavaScript-grunnlaget, kan vi enklere gå over til et rammeverk som React eller Angular senere.

Tjenerdelen

API

API-et vi bruker er fra TMDB, det gir oss muligheten til å få tak i all informasjon som ligger på TMDB. Vi henter informasjon fra API-et når det er informasjon som vi ikke har lagret/skal lagre i vår database. API-et tilbyr også ting som populære medialister og lignende, noe som vi bruker for eksempel på forsiden. Denne informasjon blir lagret i minne og blir hentet på nytt hver dag for å ha den oppdatert.

Logging

En av ekstrakravene var å logge ting som skjer på serveren. Loggesystemet vi har gjennomført benytter seg av en npm som heter «Winston». Dette er en pakke som gjør logging veldig lettvint og er konfigurerbar. Da vi skulle sette opp loggingen så fant vi først ut av hvilke nivåer vi trenger. Her finnes det selvfølgelig «beste praktiser» i kode-verden og vi prøvde å følge disse. Vi valgte å ha to forskjellige loggemetoder, det endte opp med «production» og «debug». «Production» er når applikasjonen er i drift og «debug» er under utvikling slik at utvikleren skal få så mye informasjon som mulig under arbeidet. Deretter er en logg delt inn i forskjellige nivåer som er «error», «warn», «info» og «debug». De tre første er informasjon som gå under «production», mens alle sammen blir loggført når «debug» modus er satt. Loggene blir logga til filer som er delt inn i de ulike nivåene, utenom info som bare blir logga i en kombinert fil som heter combined.log.

Språk

Det ene ekstrakravet var at applikasjonen skulle gjøres flerspråklig. Dette oppnådde vi med en npm-pakke ved navn «i18n». I18n er en enkel oversettelsesmodul med dynamisk JSON-lagring. I stedet for å ha ren tekst i HTML-filen, er denne erstattet med en string som samarbeider med disse JSON filene. Disse stringene er lagret i alle de forskjellige språkfilene, men med forskjellige verdier. Verdiene er den oversatte teksten til språket som filen hører til.

For å gi brukeren en mulighet til å endre på språket selv, måtte vi lage en funksjon som overskriver «Accept-Language» som er lagret i nettleseren. For en nordmann ville denne vært satt til norsk, og applikasjonen ville automatisk ha oversatt seg selv til det språket. Om språket ikke finnes til applikasjonen, ville den blitt oversatt til engelsk.

Det er også lagt til en mulighet for at administrator skal kunne legge til, slette og redigere språk. Dette kan være nye eller eksisterende språk. Disse funksjonene finnes i administrator-dashbordet på applikasjonen.

Det er ikke alt som var mulig å oversette. For når vi kan hente flere tusen filmer og serier via API-et, blir det vanskelig å oversette alt. En løsning vi kom frem til er at API-et kan sende over informasjon i flere språk. Dermed henter den informasjonen i det språket brukeren har valgt om det finnes, om det ikke finnes en oversettelse på klientens språk vil den automatisk hente engelsk. Slik får klienten i det minste tekst til filmen og det blir ikke en tom side.

Websocket

Vi bruker websocket for mye av kommunikasjonen mellom klient/tjener. For websocket så bruker vi biblioteket «socket.io». Dette gjør at kommunikasjon mellom klient og server blir i sanntid. Vi valgte socket.io siden vi ville lære oss å bruke det. Vi kunne ha brukt AJAX til mye av det vi bruker socket.io til, men for å oppnå det læringsutbyttet vi var ute etter, brukte vi hovedsakelig socket.io. Vi bruker denne form for kommunikasjon når vi ønsker at en forespørsel skal skje uten at siden trenger å lastes inn på nytt.

Kodeorganisering

Det å kunne organisere og skrive kode som er lett å forstå for andre er en utfordring i seg selv. Vi måtte da ha noen felles regler og måter som alle sammen forstod. Vi prøvde å holde oss til funksjonsbasert programmering og vi brukte bare klasser der vi følte det var en bedre løsning. Vi lagde også en klasse som gjorde at vi hadde en felles måte å returnere informasjon på. Denne kalte vi «ValidationHandler». Vi lagde denne fordi det ble veldig mange forskjellige måter vi kunne håndtere error på og returnere informasjon. Denne klassen inneholdt en «status» variabel som forteller om operasjonen var suksessfull eller ikke. Den andre variabelen «information» er en informasjonsvariabel og kan inneholde hva som helst. Dette gjorde at alle visste hva de fikk tilbake fra hver funksjon. Vi bruker også jsdoc for å dokumentere koden.

Databasen

Vi har bestemt oss for å bruke MongoDB som databasesystem i vårt prosjekt. Det viktigste argumentet for denne beslutningen er at MongoDB bruker JSon-lignende format som lagring av data, og at det er enkelt skalerbart. Hvis vi for eksempel hadde hatt MariaDB/MySQL som databasehåndteringssystem måtte vi ha gjort JSON om til SQL spørringer, så dette har spart oss for en del arbeid, både i kode og i planlegging av databasen.

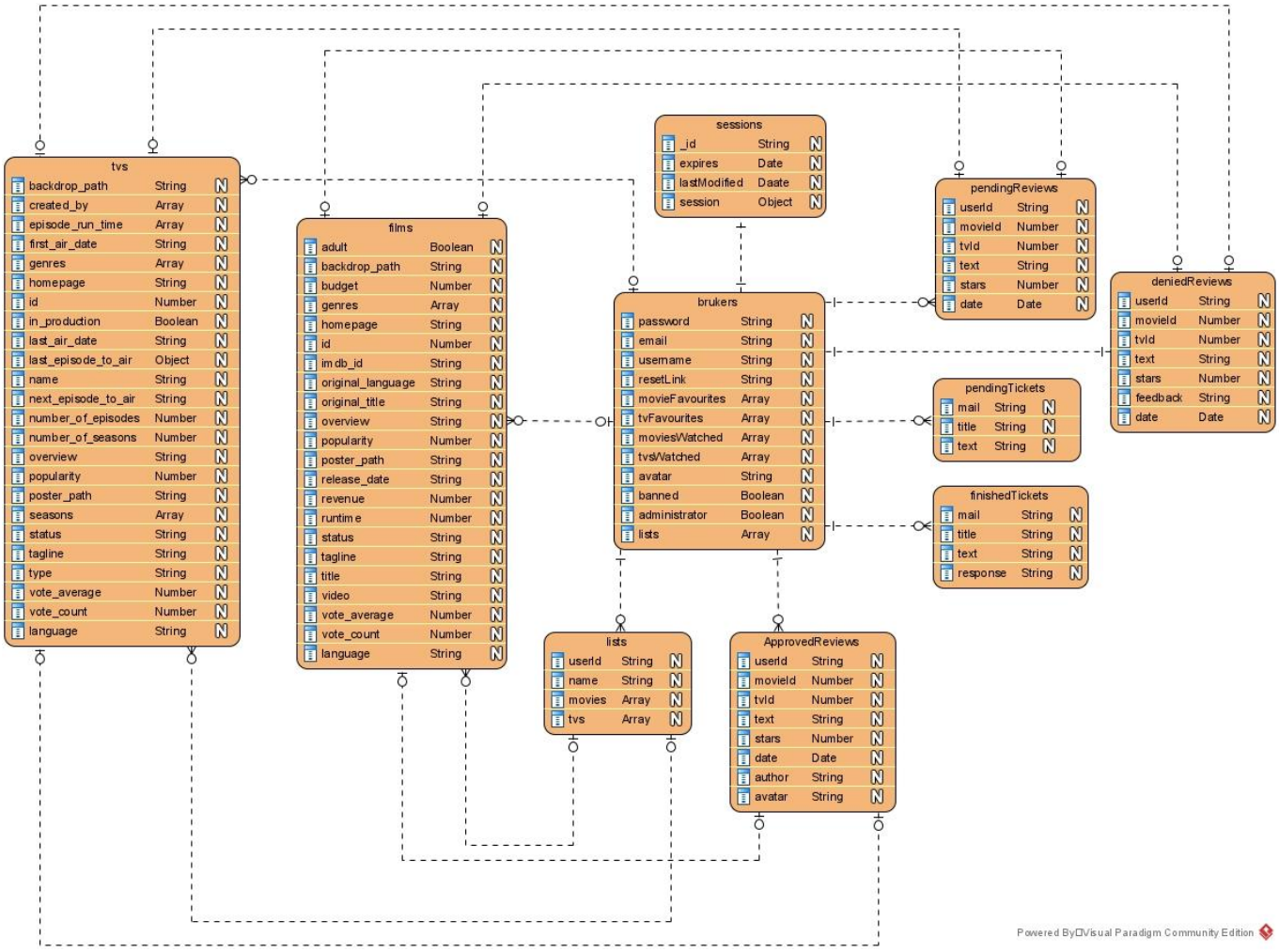
Når vi får en film/serie fra databasen, lagrer vi responsen fra API-et direkte inn i databasen. Modellen er da satt opp helt lik som svaret vi får fra API-et.

Modellen Bruker inneholder all informasjon som er knyttet til favoritter/watched og lister. Dette blir lagret som et array som inneholder Id'en til de andre dokumentene. Det gjør det veldig lett for oss å loope igjennom informasjonen når den trengs.

For anmeldelser valgte vi å lage tre forskjellige «collections» som det heter i mongoDB. Dette er da «pendingReview», «deniedReview» og «approvedReviews». Begrunnelsen for dette var at det er mye lettere å jobbe med, og det blir mye mer effektivt når vi skal hente anmeldelser for media og admin-funksjoner. Det samme gjelder tickets, men her har vi bare pendingTickets og finishedTickets.

For lister valgte vi å kjøre en collection, som kobles sammen til en bruker med en bruker-ID. Sessions inneholder informasjon om en session som cookies, når den utløper, sist modifisert osv.

Nedenfor kan du se en ER modell av databasen vår.

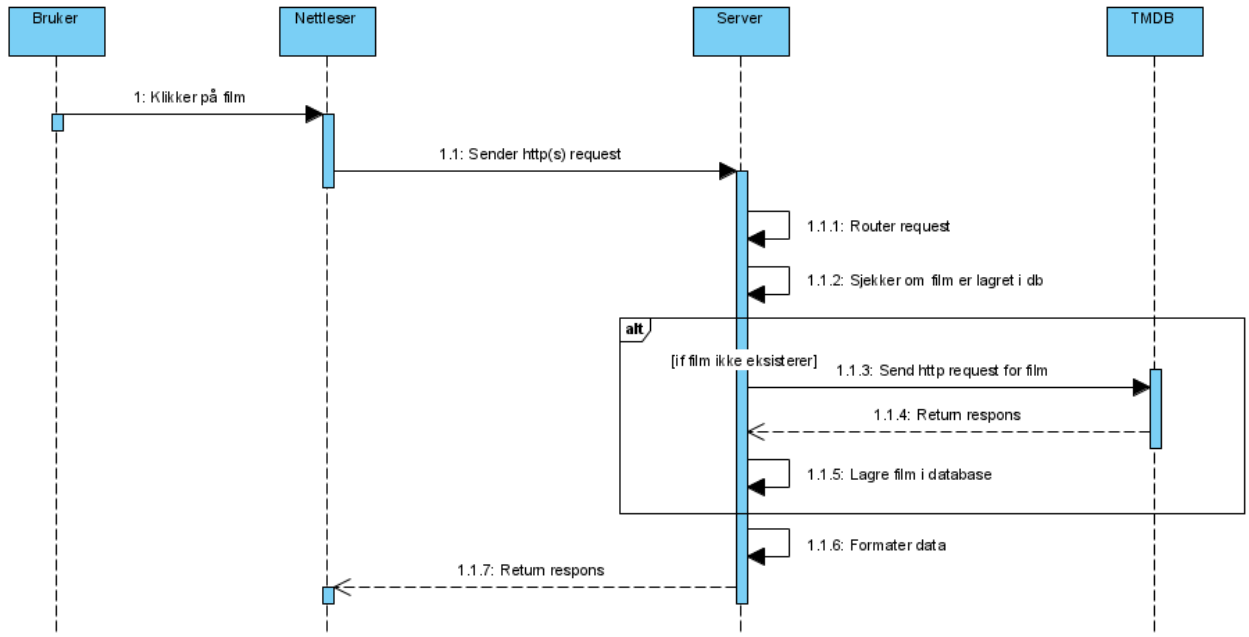


Bilde 5 – ER modell av databasen

Sekvensdiagram

Nedenfor har vi et bilde av et sekvensdiagram som illustrerer hva som skjer når en bruker går inn på en film/serie. Det første som skjer, er at det blir sendt en forespørsel til serveren.

Serveren router deretter forespørselen til riktig kontroller. Det blir kjørt en sjekk om filmen er lagret i databasen vår, dersom den ikke er det så vil informasjonen bli hentet fra API-et og lagret i databasen vår. Dataen blir da håndtert i kontrolleren og deretter sendt tilbake til brukeren der nettleseren viser nettsiden.



Bilde 6 – Sekvensdiagram av en film forespørsel

Sikkerhet

Da vi startet arbeidet med applikasjonen var det login- og registrerings-systemet som først ble gjennomført. Det er viktig at disse systemene overholder en viss standard på sikkerheten. Vi salter og krypterer passordet før vi lagrer det i databasen. Vi har også testet for No-SQL injeksjoner og ingen har fungert. Neste på listen etter et system for brukere, var å kunne huske at brukeren var logget inn. Her bruker vi session og cookies for å løse dette problemet. Her dukket det etter hvert opp et problem. Vi lagret bruker ID-en i cookies, så klienten har tilgang til bruker-ID-en som er koblet direkte til databasen. Det er denne ID-en som blir brukt under kommunikasjon mellom klient og server. Dersom klienten endrer denne bruker ID-en som ligger i cookies til en annen person sin ID, så vil denne personen kunne «hacke» andre. Heldigvis er jo disse ID-ene helt tilfeldig generert og ganske vanskelig å gjette seg fram til, men det er fortsatt et stort sikkerhetshull. Vi har dessverre ikke fått fikset opp i dette på grunn av tiden. Løsningen for dette problemet er å lagre en session-ID hos klienten istedenfor, og deretter gjøre et oppslag mot databasen som deretter skaffer bruker-ID.

Kravspesifikasjon

Endringer

Her er en liste over endringer av bruker historier og begrunnelse.

US.13 Som en gjest vil jeg kunne se en oversikt over de mest populære review

- Denne ble endret til «Som en gjest vil jeg kunne lese anmeldelser på filmer og serier» fordi vi ikke hadde noe planlagt vurderingssystem på anmeldelsene.

US.27 Som et medlem vil jeg kunne kommunisere med medlemmer og administratorer/support

- Denne ble endre til «Som et medlem vil jeg kunne kommunisere med administrator/support» fordi vi aldri implementerte kommunikasjon mellom medlemmer.

Uløste

Her er en liste over uløste brukerhistorier og begrunnelse.

US11. Som en gjest vil jeg kunne se en kalender over serier sine release-dates

- Vi rakk ikke å fullføre denne fordi vi ikke fant ut hvordan vi ville ha utseende på kalenderen. Selve koden for kalenderen hadde ikke vært noe stort problem, ettersom vi allerede har grunnmuren for det meste.

US12. Som en gjest vil jeg kunne se historiske filmer fra samme dato på hjemmesiden

- Ikke løst pga. tid.

US15. Som en gjest vil jeg kunne lese «quotes» fra filmen

- Denne User-storien ble faktisk ferdig, men vi var ikke fornøyde med hvordan det ble seende ut på siden. Koden ligger i mappen «Systems».

US16. Som en gjest vil jeg kunne se interessante og random fakta om filmen/produksjonen

- Ikke løst pga. tid.

US26. Som et medlem vil jeg kunne kommunisere med andre om film/serier på et forum.

- Forum var noe vi hadde lyst til å gjennomføre i starten og følte det ville gi nettsiden et stort løft. Vi fant fort ut at dette ble alt for mye jobb i forhold til hva vi hadde igjen etter hvert som vi kom ut i semesteret.

US28. Som et medlem vil jeg kunne legge inn interessante fakta om filmer/serier

- Ikke løst pga. tid.

US29. Som et medlem vil jeg kunne legge inn quotes fra filmene

- Denne User-storien er koblet til US15 og ble også ferdig. Men vi ble ikke fornøyd med hvordan det ble seende ut på siden. Koden ligger i mappen «Systems».

US35. Som en administrator vil jeg kunne slette/endre ting på forumet

- Forum rakk vi ikke å gjennomføre.

US36. Som en administrator vil jeg kunne godkjenne fakta skrevet av medlemmer på film/serier sider.

- Ikke løst pga. tid.

US37. Som en administrator vil jeg kunne godkjenne quotes skrevet av medlemmer på film/serier side.

- Ikke løst pga. tid.

Testplan

Bruker-testing fremgangsmåte

Vi delte inn bruker-testingen for nettsiden i to faser. Vi gjennomførte først en testing i midten av april og en ny test-gjennomføring mot slutten av mai. «Bruker-testing fase 1» vil vi kun forklare kort i det neste avsnittet, mens fase 2 vil vi presentere mer detaljer om. Dette er på grunn av at da vi gjennomførte den første fasen med bruker-testing var det mindre funksjonaliteter og enkelte problemer vi allerede var godt kjente med. I tillegg inneholdt fase-1-testingen hovedsakelig åpne spørsmål som for det meste gav oss tilbakemeldinger som var mer helhetlige og spesifikke. I fase 2 ble det spurt mer kvantitative spørsmål og brukerne måtte teste flere funksjonaliteter.

Bruker-testing fase 1

I bruker-testing fase 1 fikk vi gode tilbakemeldinger på design og mange forslag til videre utvikling og feil som må fikses. Blant annet fikk vi forslag om å kunne ha brukernavn og profilbilde for brukere, at brukeren skulle kunne fjerne media fra liste-sidene og at hjemmesiden viste populære/trending filmer. Vi var fornøyde med responsen vi fikk på det tidspunktet og bestemte oss for å gjennomføre en ny bruker-testing når flere funksjoner var lagt til og flere endringer var gjort.

Bruker-testing fase 2.

Brukertesting fase 2 er presentert nedenfor i flere punkter. Til denne brukertesten fikk vi tak i seks brukere til å teste siden. De mottok en link til et google-form skjema som inneholdt en instruks for gjennomføring av testingen. Etter alle stegene i instruksen var gjennomført skulle brukerne svare på spørsmål.

Bruker-testing Instruks

Her har vi listet instruksen for brukertestingen fase 2:

- Steg 1. Gå til https://filmatoryeksamen.herokuapp.com/
- Steg 2. Opprett en bruker og logg inn
- Steg 3. Søk på en film eller serie og legg den til som favoritt
- Steg 4. Se at filmen/serien havnet i favoritter
- Steg 5. Fjern en favoritt
- Steg 6. Prøv å lag en ny liste
- Steg 7. Legg til en film eller serie i den nye listen du lagde
- Steg 8. Gå inn på listen din
- Steg 9. Last opp et bilde til brukeren din
- Steg 10. Endre navnet til brukeren din
- Steg 11. Lag en Anmeldelse på en film eller serie
- Steg 12. Se lister som andre brukere har laget
- Steg 13. Se kommende filmer eller serier
- Steg 14. Prøv å endre språk på siden
- Steg 15. Se liste over filmer eller serier og prøv filter
- Steg 16. Logg ut
- Steg 17. Fyll ut tilbakemeldingsskjemaet

Tilbakemeldingsskjemaet

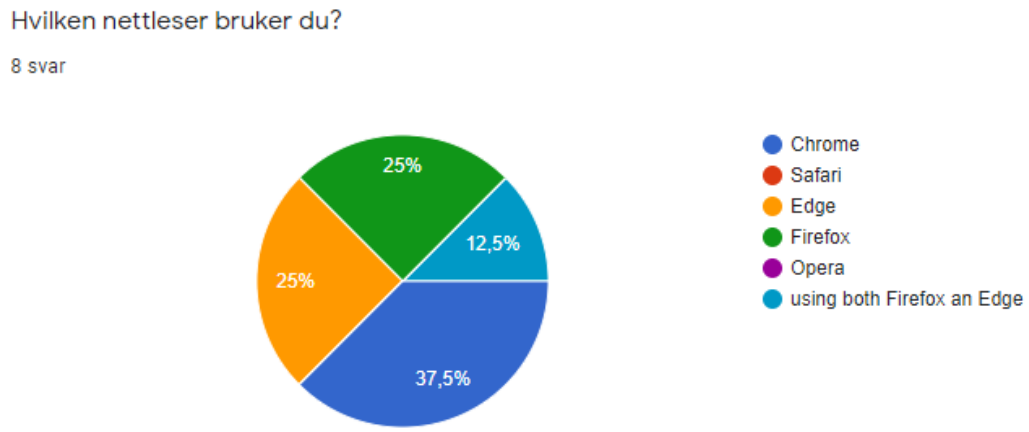
Her er spørsmålene spørreskjemaet inneholdt. I parentes er det skrevet hvilken form spørsmålene ble besvart:

- Hvilken nettleser bruker du? (Flervalg)
- Hvordan var førsteinntrykket ditt? (Skala 1-6)
- Hvor brukervennlig var nettsiden? (Skala 1-6)
- Hvordan opplevde du registreringen? (Skala 1-6)
- Hva synes du om favoritt-funksjonen? (Skala 1-6)
- Hva synes du om liste-funksjonen? (Skala 1-6)
- Hva synes du om profil-siden? (Skala 1-6)
- Hva synes du om review-funksjonen? (Skala 1-6)

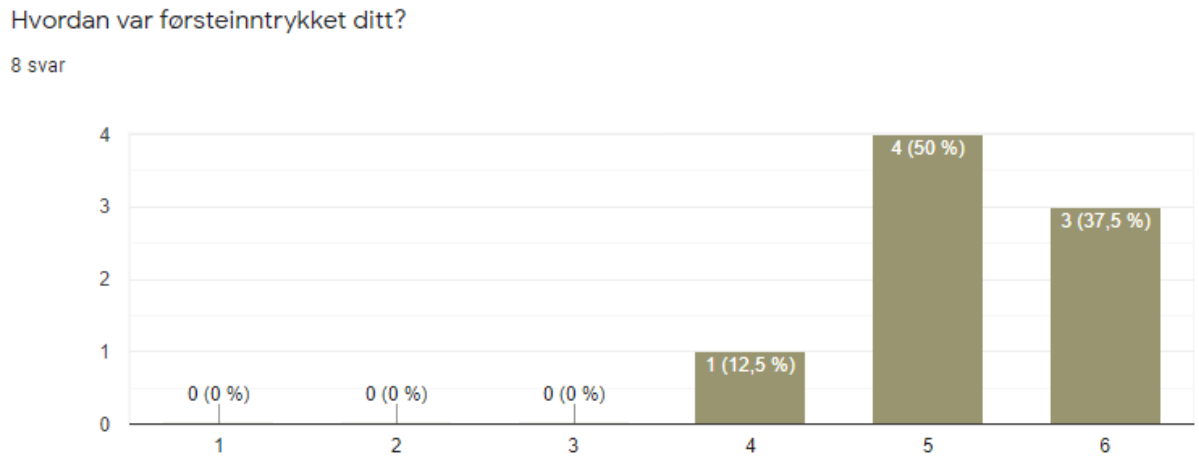
- Hva synes du om siden for kommende filmer? (Skala 1-6)
- Hva er din helhetsvurdering av nettsiden? (Skala 1-6)
- Kommenter gjerne hva du synes om nettsiden. (Avsnitt-svar)
- Har du forslag til forbedring av nettsiden? (Avsnitt-svar)

Resultater Bruker-testing fase 2

Nettleser



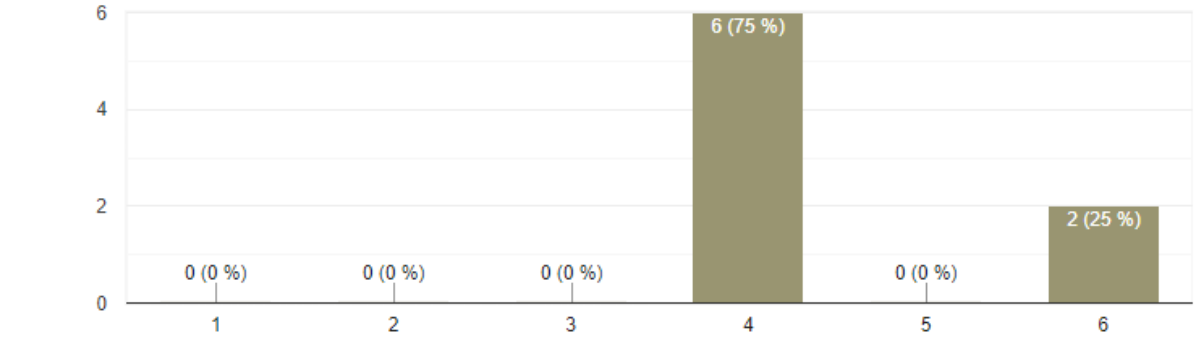
Førsteinntrykk



Brukervennlighet

Hvor brukervennlig var nettsiden?

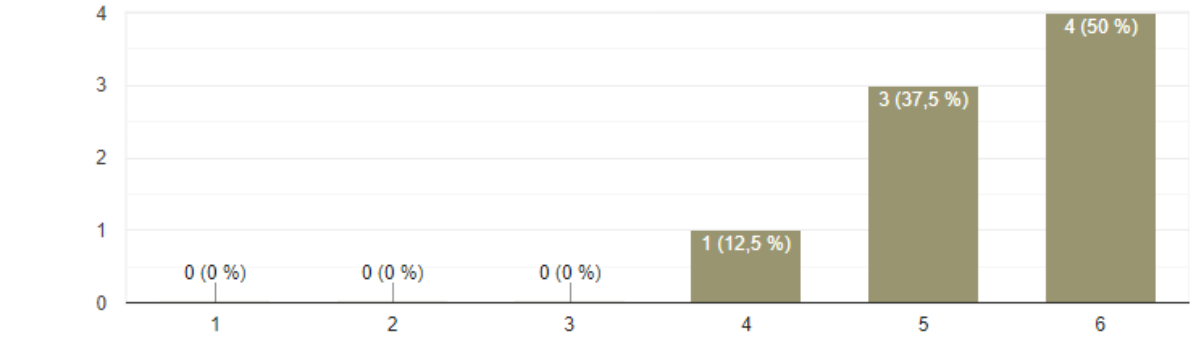
8 svar



Registrering

Hvordan opplevde du registreringen?

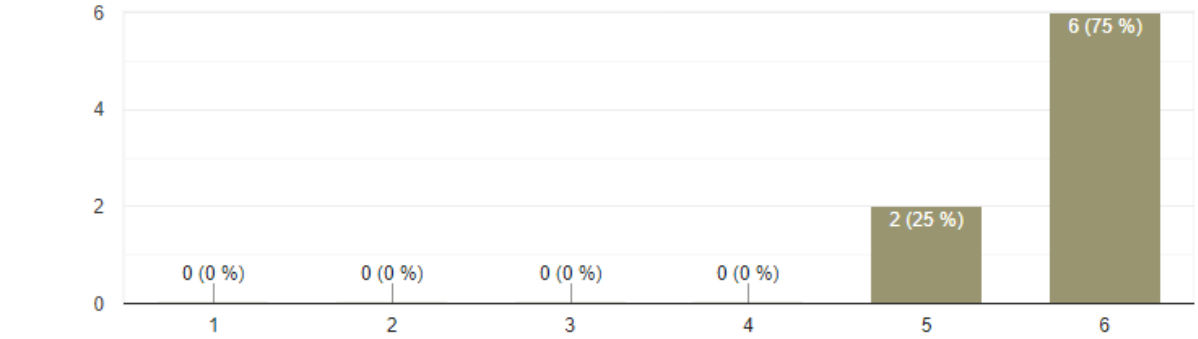
8 svar



Favoritt-funksjon

Hva synes du om favoritt-funksjonen?

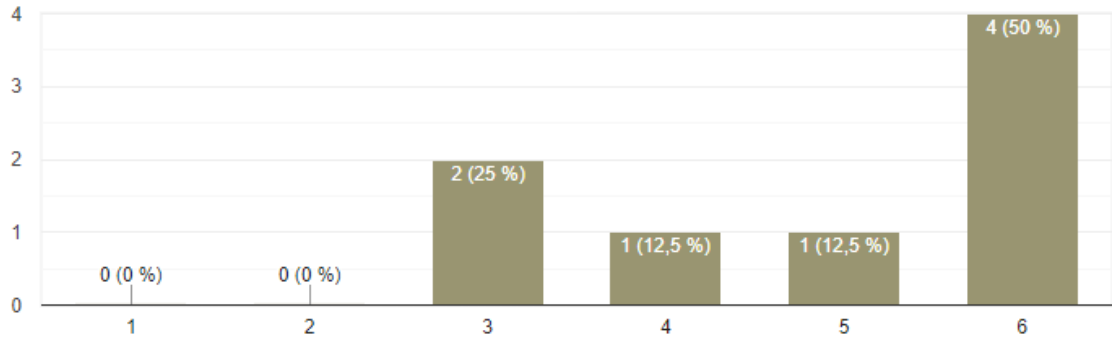
8 svar



Liste-funksjon

Hva synes du om liste-funksjonen?

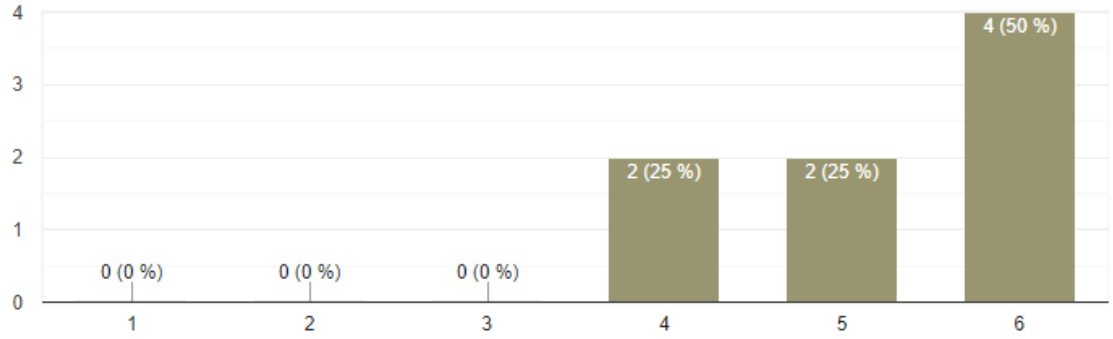
8 svar



Profil-siden

Hva synes du om profil-siden?

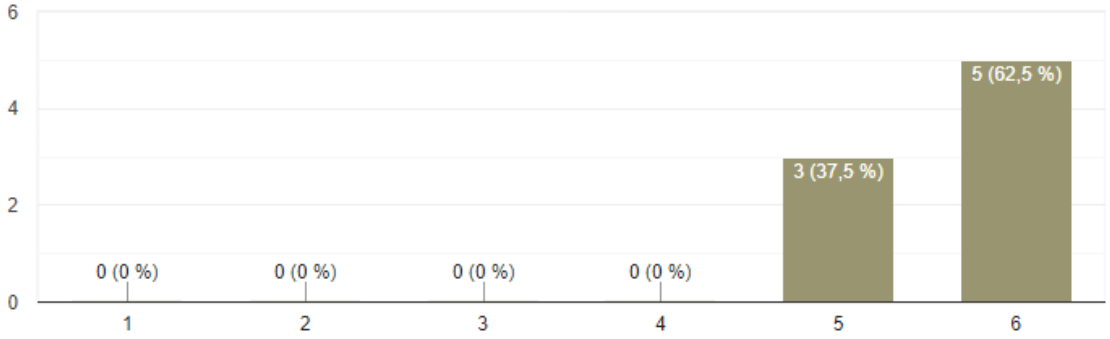
8 svar



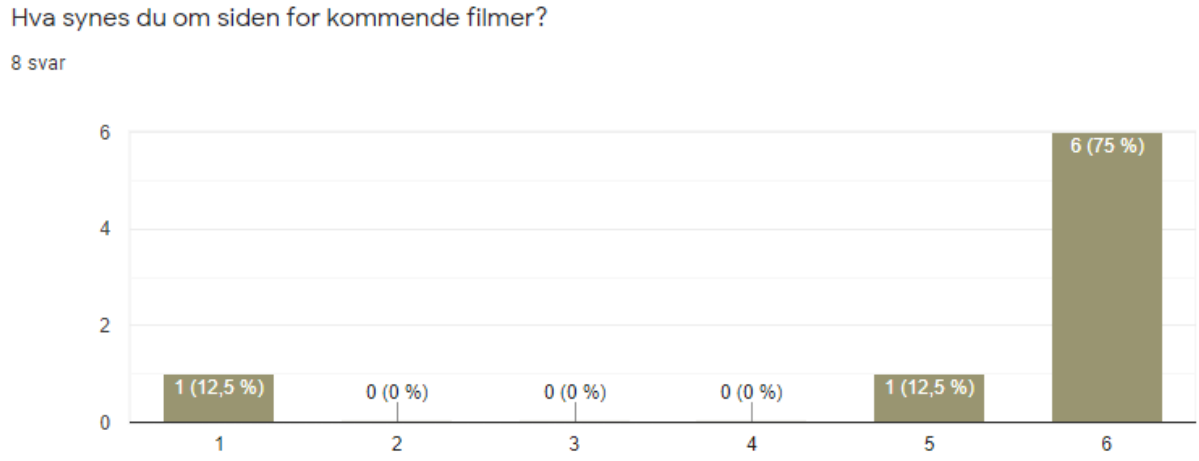
Review-funksjon

Hva synes du om review-funksjonen?

8 svar

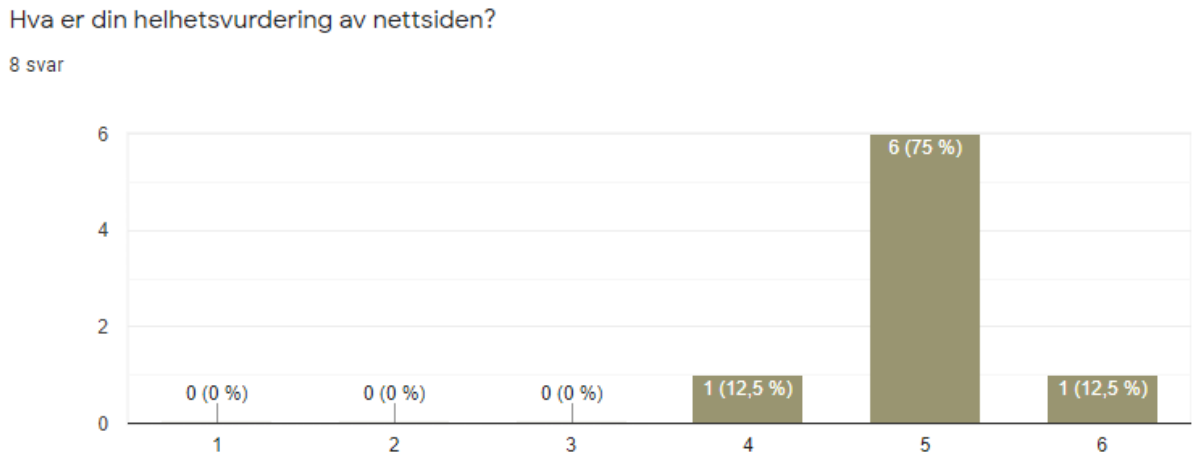


Kommende filmer



Kommentar: Vi fant ingen feil eller mangler på kommende filmer, så vi lurer på om testeren kan ha mislyktes i å finne siden eller lignende. Det ble ikke nevnt noe om kommende filmer i test-besvarelsene.

Helhetsvurdering



Helhetsvurdering med kommentarer

Kommenter gjerne hva du syntes om nettsiden.

6 svar

Litt enkelt design, men har alle funksjoner lett tilgjengelig. Morro at man kan lage egne lister, og bra at man kan skrive anmeldelser. Fint at man kan endre brukernavn og bilde.

Really super clean an fast loading site!

helt ok frem til jeg kræsjet den to ganger -christian

Små ting her og der som må fikses

Ikke rett fram at du må opprette liste inne på profilen,men fant ut av det

Det som trekker ned: favoritter knappen virker ikke. Jeg må gå på MinKonto først, dernest Favoritter. Noe plunder med Min Liste. Der er bare et tomt ikon. Men favoritter funker.

Har du forslag til forbedring av nettsiden?

6 svar

Når man lager bruker står det noen krav til brukeren. Over kravene burde det kanskje stå "Pass på at" eller lignende, evnt bytte ut "er" med "må være" i listen med krav. På de to siste punktene burde "er" byttes ut med "har", evnt "må ha". "Deler av teksten (f eks "Password")er engelsk selv om valgt språk er norsk.
Linken til favoritter funker ikke, man må gå via "Min bruker".
Evnt vurder å gjøre det mulig å fjerne en film fra favoritter direkte fra favoritter-siden, i stede for å måtte gå inn på filmsiden og fjerne den der.
Når man skriver en anmeldelse så kommer den ikke opp med en gang, man må gå ut og inn. Hadde vært kult om den kom opp med en gang. Profilbildet på anmeldelsen funker ikke.
Fikk ikke lagt til film i listen. Jeg valgte liste og trykket save, og det sto at filmen var blitt lagt til successfully. Pop-up'en forsvant ikke og eneste måte var å trykke cancel eller trykke utenfor bildet, og filmen havnet ikke i lista. I Pop-up'en var også mesteparten skrevet på engelsk, selv om jeg hadde norsk som valgt språk.
Generelt flere steder er det engelsk tekst selv om valgt språk er norsk.

would love to see option to set the background to black/darker color. some more details/information on some of the series/movies could be abit longer, fx Tom Clancy's Without Remorse,

Bildene til filmene som kommer opp i listene er litt for store, veldig hvit side. Når man skal lage ny liste er det litt forvirende om hvordan man oppretter liste, prøved å trykke på new men fant ut at man må lage navn også trykke new(kanskje flytte på knappen til høyre siden)

FIkse de knappene som nevnt over. Profilbildet forsvant. MIn liste lager et ikon, men filmen reistreres ikke. Siden er bra, men man må få dette nevnte til å funke. det er kun en film som registeres i favoritter. (har jeg gjort noe feil?)

Etter registrering bør du gå direkte til innloggingsiden, eller automatisk bli logget inn.

Funksjonstesting

I denne delen har vi gjennomført ulike tester for å oppdage feil med innlogging og registrering. I tabellen nedenfor har vi beskrevet forsøk og resultat av disse testene.

Forsøk	Resultat
Korrekt innlogging med en registrert bruker	Suksessfull innlogging
Prøvde å logge inn med ikke-registrerte opplysninger	Får feilen "User does not exist"
Registrering Forsøkte med ugyldig format under e-post	"ERROR: Email is not properly formatted"
Registrering Gyldig e-post, men ulike passord	ERROR: Password is not properly formatted
Registrering Gyldig e-post, men passord med kun tall	ERROR: Password is not properly formatted
Registrering Gyldig e-post, men for kort passord	ERROR: Password is not properly formatted
Innlogging Forsøk med denne NoSql-incection-en: "db.accounts.find({username: username, password: password});"	"ERROR: User does not exist" Så et enkelt hacking-forsøk ville ikke vært vellykket.

Kommentarer om testingen

Begge bruker-testingene viste seg å være nyttige for å forbedre siden vår, siden vi fikk gjennomganger av feilfiksing både før og etter testingene. I tillegg ble gruppen mer motivert for å lage et bra produkt siden venner og kjente fikk se den. I forkant av funksjonstestingen gjennomførte vi også forbedringer på error-systemet ved registrering og innlogging. Så både brukertestingene og funksjonstestingen har vist seg som nyttige verktøy for å oppdage problemer og løse dem.

Prosjektplan

Planleggingsprosessen

Planleggingsprosessen pågikk omtrent det første halve året av prosjektet. Den bestod av å planlegge gjennomføringen og å skaffe oss nødvendig kompetanse for gjennomføringsdelen.

Under det første møtet av prosjektet gjennomførte vi en brainstorming for hva vi ville at prosjektet skulle bestå av. Dette avgjorde vi tidlig og gikk over til å vurdere hvilke programmeringsspråk vi skulle bruke for å lage produktet. Vi bestemte oss også for å ha ukentlige møter i planleggingsprosessen og møtes to ganger i uken i resten av prosjektet. I tillegg gjorde vi flere praktiske ting som å sette opp og lære oss hvordan vi skulle samarbeide i Visual Studio Code. Vi satte opp arbeidsmiljøet med Node.js og Express.

Vi satte i gang med Trello, som er en nettbasert kanban-tavle. Der satte vi inn alle brukerhistoriene vi hadde planlagt og andre oppgaver. Med trello som vår kanban-tavle arbeidet vi med ukentlige Scrum-sprints. Vi fordelte oppgaver under møter og arbeidet med dem så godt vi kunne innen neste møte.

SWOT-analyse

Her har vi gjennomført en kort SWOT-analyse med fokus på hvordan gruppen som bedrift og prosjektet kan gjøre det i markedet. Selv om resultatet av denne analysen ble at konkurransen er høy og markedet mettet, så valgte vi allikevel å gjennomføre prosjektet. Vi ville heller prioritere et prosjekt som gav en stor grad av læringsutbytte og var praktisk for gruppen å samarbeide med.

Styrker: • Gode programmerere (backend og frontend) • Bra samarbeid i teamset	Svakheter: • Lav erfaring
Muligheter: • Kan være rom i det norske markedet.	Trusler: • Allerede godt etablerte konkurrenter • Vanskelig å skille seg ut

Risikovurdering

Vi har gjennomført en enkel risikovurdering av prosjektet med tanke på hva som kan påvirke prosjektets resultat. Under Risikofaktorer har vi beskrevet ulike hendelser og utregnet

risikofaktor. Risikofaktor er et resultat av sannsynlighet multiplisert med konsekvensen. Under Risikohåndtering har vi forklart vår vurdering og mottiltak for hver hendelse.

Risikofaktorer

Hendelse	Sannsynlighet	Konsekvens	Risikofaktor
API er nede	0,1%	100%	1%
Klarer ikke å lage sentrale løsninger for nettsiden	20%	50%	10%
Dårlig kommunikasjon	10%	30%	3%
Rekker ikke å fullføre prosjektet før fristen.	1%	100%	10%
Rekker ikke å utføre alle user-stories.	50%	10%	5%
Dataproblemer	10%	5%	0,5%

Risikohåndtering

Hendelse	Vurdering/Mottiltak
API er nede	Vi ser på sannsynligheten som lav, men siden konsekvensen er så høy er denne tatt med. Mottiltak vil være å lagre informasjonen på databasen vår for å erstatte API-et.
Klarer ikke å lage sentrale løsninger for nettsiden	For å forhindre at en ikke klarer å lage prosjektet bør gruppen forberede seg godt, finne gode arbeidsmetoder og kilder for nyttig informasjon.
Dårlig kommunikasjon	Gruppen er allerede godt kjente og samarbeider godt, men siden det kan oppstå problemer med alle grupper har vi tatt dette med. For å forhindre at dette skjer vil vi ha faste møter og sørge for tydelig kommunikasjon med hverandre.
Rekker ikke å fullføre prosjektet før fristen.	Sannsynligheten for å ikke fullføre prosjektet er satt som veldig lav siden gruppemedlemmene føler seg trygge på at prosjektet vil gjennomføres. Mottiltak vil være hyppige møter og godt samarbeid.

Rekker ikke å utføre alle user-stories.	Siden vi har satt opp ganske høye mål for prosjektet er vi ikke helt sikre på hvor mye vi kan gjennomføre, dermed er sannsynligheten høy. Våre mottiltak handler om å arbeide effektivt og prioritere de mest sentrale oppgavene først slik at konsekvensen blir så lav som mulig.
Dataproblemer	Konsekvensen med dataproblemer kan være at oppgaver tar lengre tid å gjennomføre siden en må vente på reparasjoner eller hvis en må installere programvare på nytt. Konsekvensen for denne risikoen vil generelt være akseptabel med unntak av perioden helt i slutten av prosjektet, da vil det være mer kritisk. Mottiltakene består av å minimere konsekvensen, ikke sannsynligheten for hendelsen. Vi må i så fall gjøre det vi kan for å løse problemene eller anskaffe ny pc så snart som mulig sånn at en kan komme tilbake til arbeidet. I perioder kan andre ta seg av nødvendige oppgaver for personen med dataproblemer.

Tidsbruk

Estimering

UseCase	Beskrivelse	Estimert tidsbruk	Tid brukt
US01	Som en gjest vil jeg kunne se informasjon om filmer	10	4
US02	Som en gjest vil jeg kunne se informasjon om serier	10	4
US03	Som en gjest vil jeg kunne se en “overview” slide	5	2,75
US04	Som en gjest vil jeg kunne se statistikker (Diagram)	10	2

US05	Som en gjest vil jeg kunne opprette meg en brukerkonto	5	0,5
US06	Som en gjest vil jeg kunne se trailere til filmer eller linker til YouTube	2	4
US07	Som en gjest vil jeg kunne få tilfeldig forslag innenfor ulike sjangrer	20	2
US08	Som en gjest vil jeg kunne se relaterte filmer/serier til den jeg ser på	15	2,5
US09	Som en gjest vil jeg kunne dele filmen/serien via sosiale medier	5	0,25
US10	Som en gjest vil jeg kunne se en kalender over kommende filmpremierer	10	1
US11	Som en gjest vil jeg kunne se en kalender over serier sine release-dates	20	0
US12	Som en gjest vil jeg kunne se historiske filmer fra samme dato på hjemmesiden	5	0
US13	Som en gjest vil jeg kunne lese anmeldelser på filmer og serier	5	4
US14	Som en gjest vil jeg kunne se ulike film/serie-lister av andre brukere	10	4,5
US15	Som en gjest vil jeg kunne lese quotes fra filmen.	5	0
US16	Som en gjest vil jeg kunne se interessante og random fakta om filmen/produksjonen	5	0
US17	Som et medlem vil jeg kunne logge inn	2	0,5

US18	Som et medlem vil jeg kunne logge av	1	0,6
US19	Som et medlem vil jeg kunne rate filmer/Serier	2	2,5
US20	Som et medlem vil jeg kunne omtale filmer/serier	5	9
US21	Som et medlem vil jeg kunne legge til en film i sett liste	5	2
US22	Som et medlem vil jeg kunne legge til en serie i sett liste	5	1
US23	Som et medlem vil jeg kunne se personlige statistikker (Grafer)	15	15,5
US24	Som et medlem vil jeg kunne kontakte support	5	4,5
US25	Som et medlem vil jeg kunne lage lister over filmer og dele listen i ulike medium.	15	12,5
US26	Som et medlem vil jeg kunne kommunisere med andre om film/serier på et forum.	25	0
US27	Som et medlem vil jeg kunne kommunisere med administrator/support	2	5
US28	Som et medlem vil jeg kunne legge inn interessante fakta om filmer/serier	3	0
US29	Som et medlem vil jeg kunne legge inn quotes fra filmene	3	0
US30	Som en administrator vil jeg kunne endre brukere sine reviews	4	1,5
US31	Som en administrator vil jeg kunne slette brukere sine reviews	2	0,6

US32	Som en administrator vil jeg kunne godkjenne brukere sine reviews	2	1,5
US33	Som en administrator vil jeg kunne avslå brukere sine reviews	2	1,5
US34	Som en administrator vil jeg kunne deaktivere brukere	1	0,5
US35	Som en administrator vil jeg kunne slette/endre ting på forumet	20	0
US36	Som en administrator vil jeg kunne godkjenne fakta skrevet av medlemmer på film/serier sider.	2	0
US37	Som en administrator vil jeg kunne godkjenne quotes skrevet av medlemmer på film/serier side.	2	0
US38	Som en bruker av nettsiden vil jeg kunne søke etter film/serie	4	3

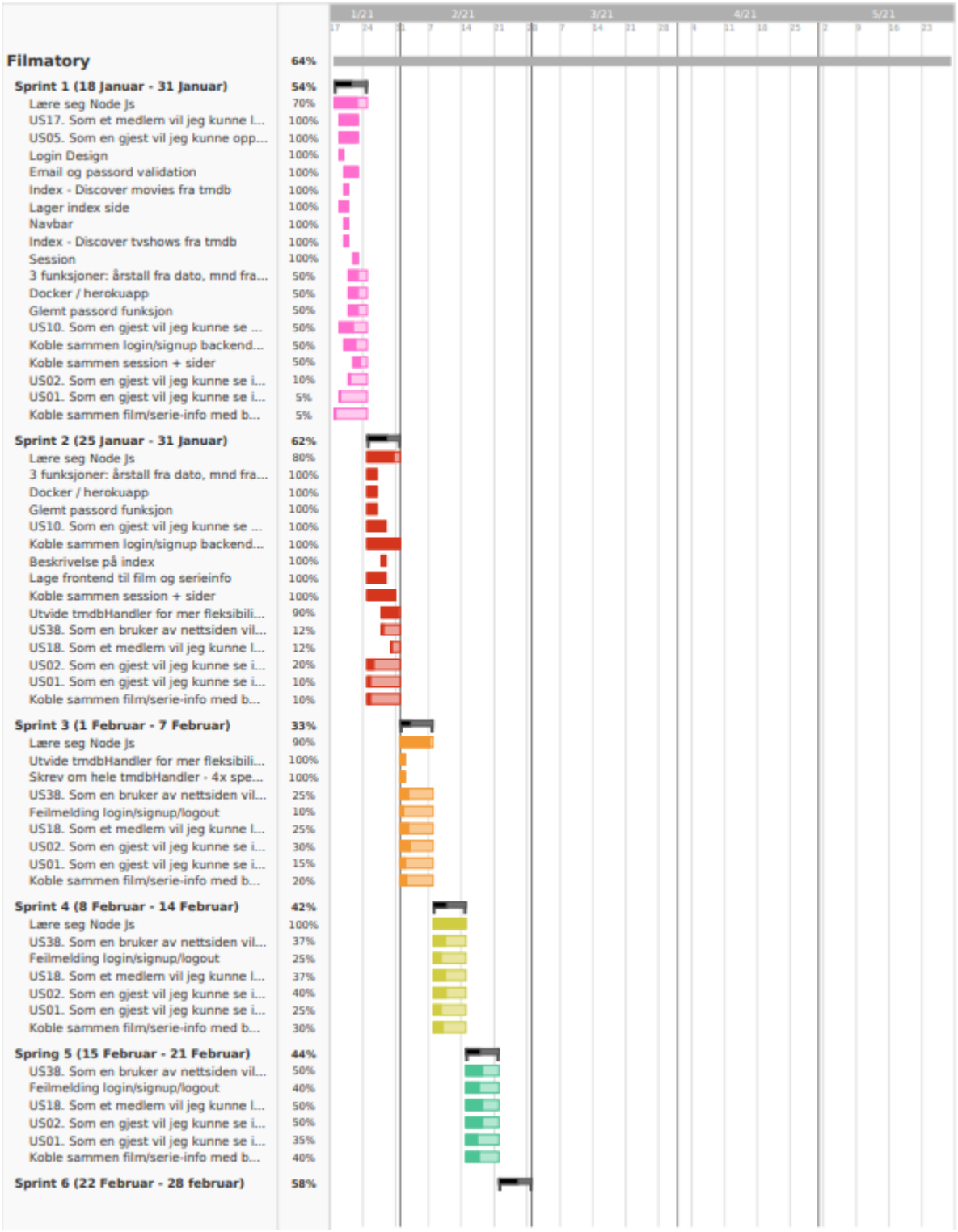
Totalt estimert for User storys: 264

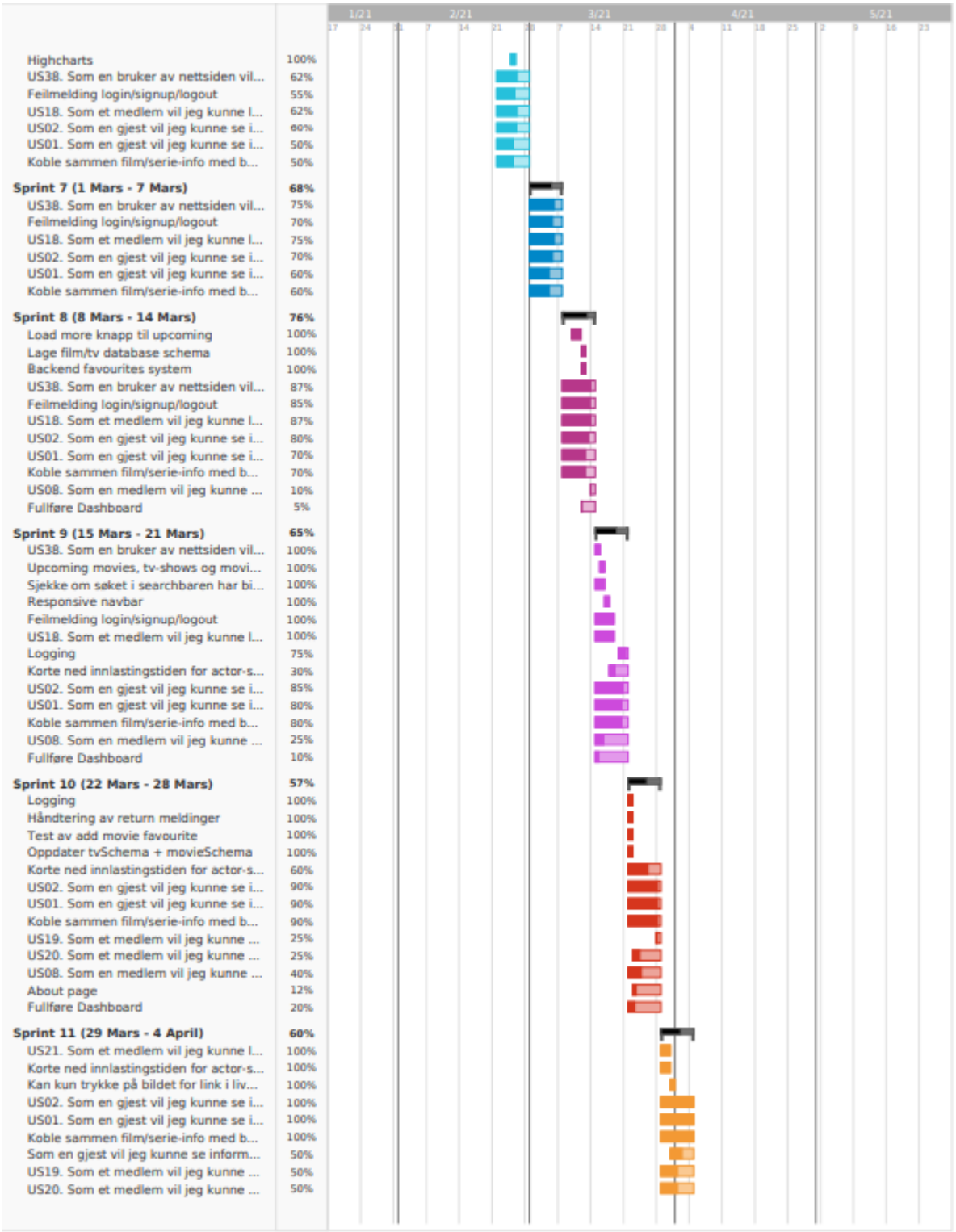
Totalt brukt for User storys: 93,2

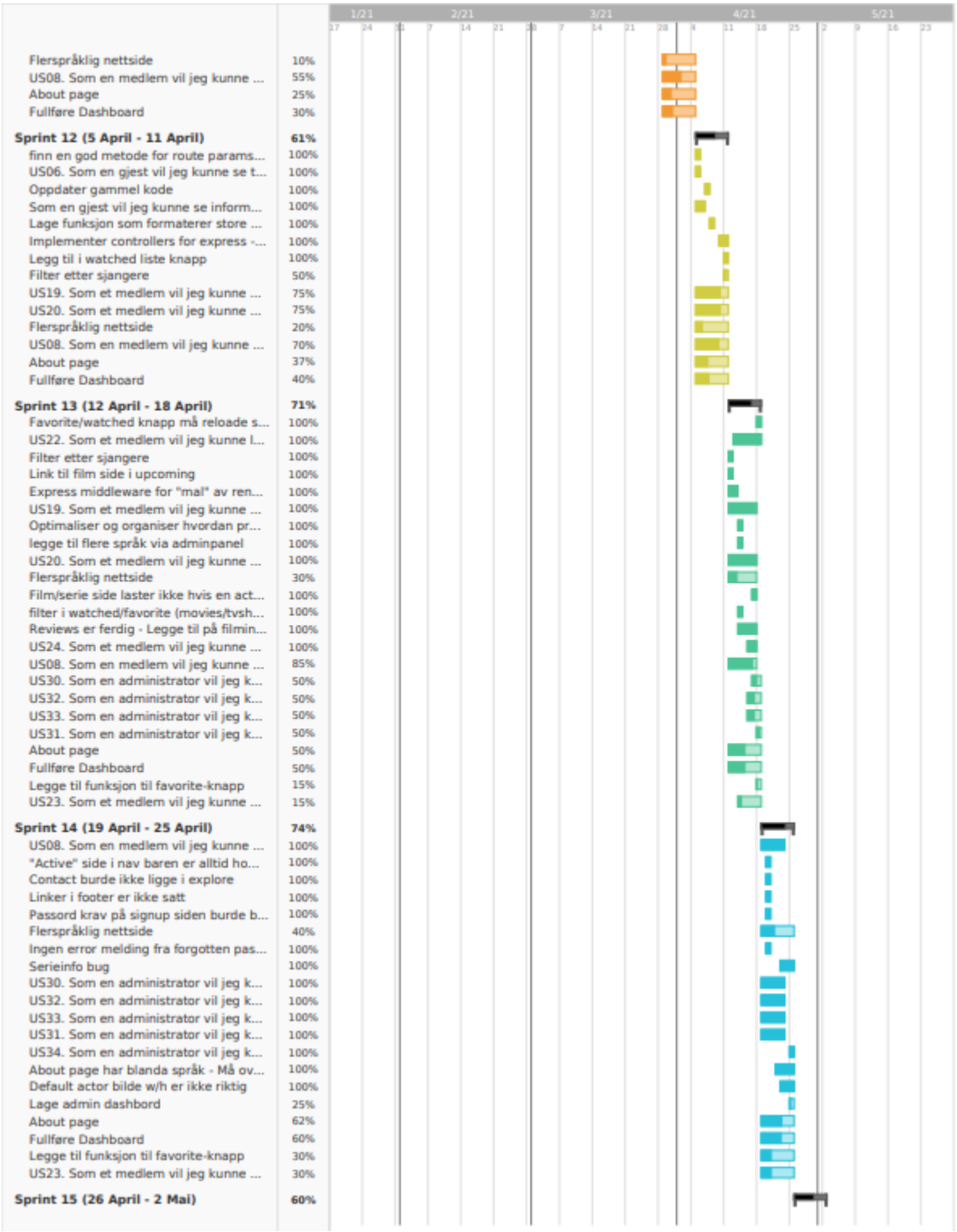
Totalt brukt tid på programmering: 469,5

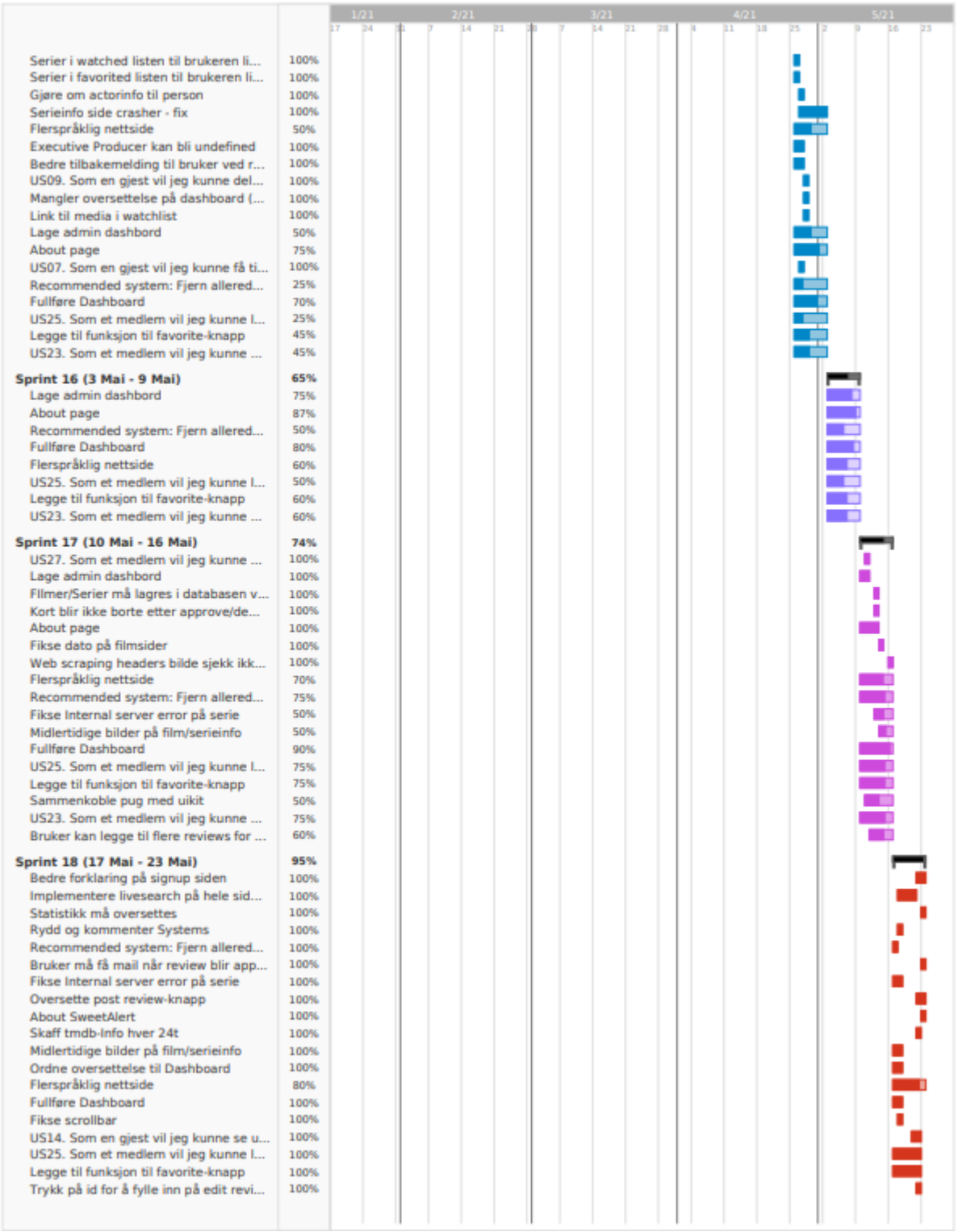
Gant-chart

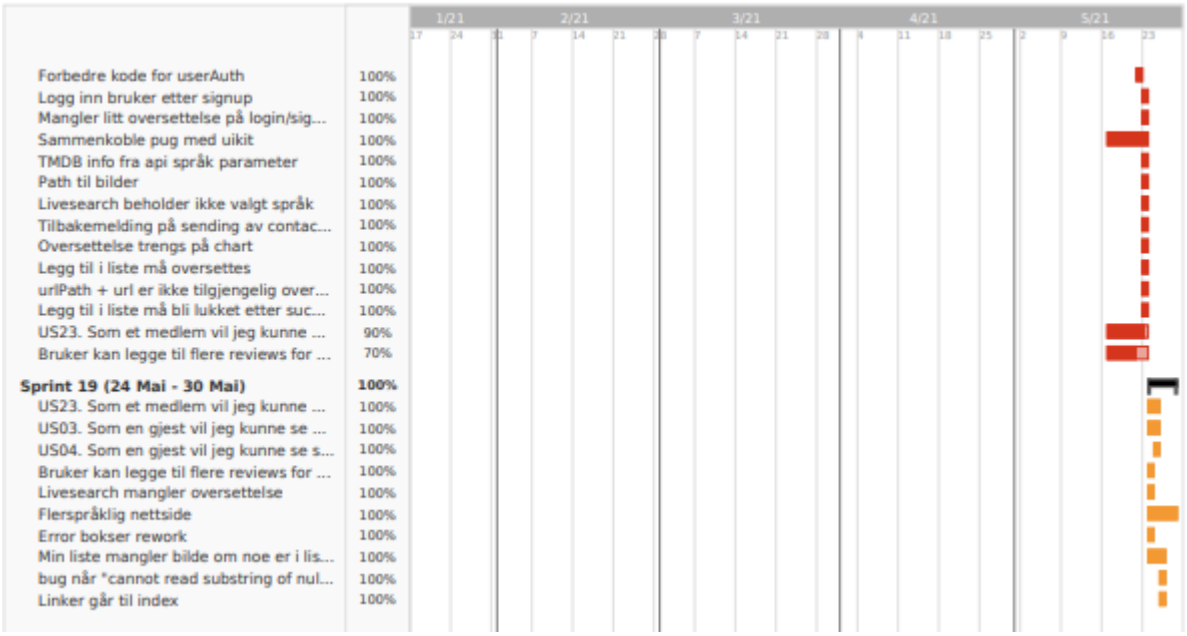
Gant-chartet viser gruppens fremgang og gjennomføring av oppgaver gjennom gjennomførings-delen. Prosenten vi har satt for hver av oppgavene i de ulike sprintene er for det meste satt av en gjennomsnittlig utregning av total tid brukt, siden vi ikke målte arbeidet i prosent. Gruppen har hatt ukentlige sprints med faste møter. Med ukentlige sprints ble det mange sprints og høyere antall oppgaver som ble jobbet med i flere sprints. Dette kunne vært unngått med å ha mer tid på hver sprint. Tross det, har gruppemedlemmene vært fornøyde med ukentlige sprints siden det medførte en stor grad av oversikt under hele prosessen.


1/21
2/21
3/21
4/21
5/21
17 24 31 7 14 21 28 7 14 21 28 4 11 18 25 2 9 16 23
Filmatory 64%
Sprint 1 (18 Januar - 31 Januar) 54%
Lære seg Node Js 70%
US17. Som et medlem vil jeg kunne l... 100%
US05. Som en gjest vil jeg kunne opp... 100%
Login Design 100%
Email og passord validation 100%
Index - Discover movies fra tmdb 100%
Lager index side 100%
Navbar 100%
Index - Discover tvshows fra tmdb 100%
Session 100%
3 funksjoner: årstall fra dato, mnd fra... 50%
Docker / herokuapp 50%
Glemt passord funksjon 50%
US10. Som en gjest vil jeg kunne se ... 50%
Koble sammen login/signup backend... 50%
Koble sammen session + sider 50%
US02. Som en gjest vil jeg kunne se i... 10%
US01. Som en gjest vil jeg kunne se i... 5%
Koble sammen film/serie-info med b... 5%
Sprint 2 (25 Januar - 31 Januar) 62%
Lære seg Node Js 80%
3 funksjoner: årstall fra dato, mnd fra... 100%
Docker / herokuapp 100%
Glemt passord funksjon 100%
US10. Som en gjest vil jeg kunne se ... 100%
Koble sammen login/signup backend... 100%
Beskrivelse på index 100%
Lage frontend til film og serieinfo 100%
Koble sammen session + sider 100%
Utvide tmdbHandler for mer fleksibili... 90%
US38. Som en bruker av nettsiden vil... 12%
US18. Som et medlem vil jeg kunne l... 12%
US02. Som en gjest vil jeg kunne se i... 20%
US01. Som en gjest vil jeg kunne se i... 10%
Koble sammen film/serie-info med b... 10%
Sprint 3 (1 Februar - 7 Februar) 33%
Lære seg Node Js 90%
Utvide tmdbHandler for mer fleksibili... 100%
Skrev om hele tmdbHandler - 4x spe... 100%
US38. Som en bruker av nettsiden vil... 25%
Feilmelding login/signup/logout 10%
US18. Som et medlem vil jeg kunne l... 25%
US02. Som en gjest vil jeg kunne se i... 30%
US01. Som en gjest vil jeg kunne se i... 15%
Koble sammen film/serie-info med b... 20%
Sprint 4 (8 Februar - 14 Februar) 42%
Lære seg Node Js 100%
US38. Som en bruker av nettsiden vil... 37%
Feilmelding login/signup/logout 25%
US18. Som et medlem vil jeg kunne l... 37%
US02. Som en gjest vil jeg kunne se i... 40%
US01. Som en gjest vil jeg kunne se i... 25%
Koble sammen film/serie-info med b... 30%
Spring 5 (15 Februar - 21 Februar) 44%
US38. Som en bruker av nettsiden vil... 50%
Feilmelding login/signup/logout 40%
US18. Som et medlem vil jeg kunne l... 50%
US02. Som en gjest vil jeg kunne se i... 50%
US01. Som en gjest vil jeg kunne se i... 35%
Koble sammen film/serie-info med b... 40%
Sprint 6 (22 Februar - 28 februar) 58%


1/21
2/21
3/21
4/21
5/21
17 24 31 7 14 21 28 7 14 21 28 4 11 18 25 2 9 16 23
Highcharts 100%
US38. Som en bruker av nettsiden vil... 62%
Feilmelding login/signup/logout 55%
US18. Som et medlem vil jeg kunne l... 62%
US02. Som en gjest vil jeg kunne se i... 60%
US01. Som en gjest vil jeg kunne se i... 50%
Koble sammen film/serie-info med b... 50%
Sprint 7 (1 Mars - 7 Mars) 68%
US38. Som en bruker av nettsiden vil... 75%
Feilmelding login/signup/logout 70%
US18. Som et medlem vil jeg kunne l... 75%
US02. Som en gjest vil jeg kunne se i... 70%
US01. Som en gjest vil jeg kunne se i... 60%
Koble sammen film/serie-info med b... 60%
Sprint 8 (8 Mars - 14 Mars) 76%
Load more knapp til upcoming 100%
Lage film/tv database schema 100%
Backend favourites system 100%
US38. Som en bruker av nettsiden vil... 87%
Feilmelding login/signup/logout 85%
US18. Som et medlem vil jeg kunne l... 87%
US02. Som en gjest vil jeg kunne se i... 80%
US01. Som en gjest vil jeg kunne se i... 70%
Koble sammen film/serie-info med b... 70%
US08. Som en medlem vil jeg kunne ... 10%
Fullføre Dashboard 5%
Sprint 9 (15 Mars - 21 Mars) 65%
US38. Som en bruker av nettsiden vil... 100%
Upcoming movies, tv-shows og movi... 100%
Sjekke om søket i searchbaren har bi... 100%
Responsive navbar 100%
Feilmelding login/signup/logout 100%
US18. Som et medlem vil jeg kunne l... 100%
Logging 75%
Korte ned innlastingstiden for actor-s... 30%
US02. Som en gjest vil jeg kunne se i... 85%
US01. Som en gjest vil jeg kunne se i... 80%
Koble sammen film/serie-info med b... 80%
US08. Som en medlem vil jeg kunne ... 25%
Fullføre Dashboard 10%
Sprint 10 (22 Mars - 28 Mars) 57%
Logging 100%
Håndtering av return meldinger 100%
Test av add movie favourite 100%
Oppdater tvSchema + movieSchema 100%
Korte ned innlastingstiden for actor-s... 60%
US02. Som en gjest vil jeg kunne se i... 90%
US01. Som en gjest vil jeg kunne se i... 90%
Koble sammen film/serie-info med b... 90%
US19. Som et medlem vil jeg kunne ... 25%
US20. Som et medlem vil jeg kunne ... 25%
US08. Som en medlem vil jeg kunne ... 40%
About page 12%
Fullføre Dashboard 20%
Sprint 11 (29 Mars - 4 April) 60%
US21. Som et medlem vil jeg kunne l... 100%
Korte ned innlastingstiden for actor-s... 100%
Kan kun trykke på bildet for link i liv... 100%
US02. Som en gjest vil jeg kunne se i... 100%
US01. Som en gjest vil jeg kunne se i... 100%
Koble sammen film/serie-info med b... 100%
Som en gjest vil jeg kunne se inform... 50%
US19. Som et medlem vil jeg kunne ... 50%
US20. Som et medlem vil jeg kunne ... 50%


1/21
2/21
3/21
4/21
5/21
17 24 31 7 14 21 28 7 14 21 28 4 11 18 25 2 9 16 23
Flerspråklig nettside 10%
US08. Som en medlem vil jeg kunne ... 55%
About page 25%
Fullføre Dashboard 30%
Sprint 12 (5 April - 11 April) 61%
finn en god metode for route params... 100%
US06. Som en gjest vil jeg kunne se t... 100%
Oppdater gammel kode 100%
Som en gjest vil jeg kunne se inform... 100%
Lage funksjon som formaterer store ... 100%
Implementer controllers for express -... 100%
Legg til i watched liste knapp 100%
Filter etter sjangere 50%
US19. Som et medlem vil jeg kunne ... 75%
US20. Som et medlem vil jeg kunne ... 75%
Flerspråklig nettside 20%
US08. Som en medlem vil jeg kunne ... 70%
About page 37%
Fullføre Dashboard 40%
Sprint 13 (12 April - 18 April) 71%
Favorite/watched knapp må reloade s... 100%
US22. Som et medlem vil jeg kunne l... 100%
Filter etter sjangere 100%
Link til film side i upcoming 100%
Express middleware for "mal" av ren... 100%
US19. Som et medlem vil jeg kunne ... 100%
Optimaliser og organiser hvordan pr... 100%
legge til flere språk via adminpanel 100%
US20. Som et medlem vil jeg kunne ... 100%
Flerspråklig nettside 30%
Film/serie side laster ikke hvis en act... 100%
filter i watched/favorite (movies/tvsh... 100%
Reviews er ferdig - Legge til på filmin... 100%
US24. Som et medlem vil jeg kunne ... 100%
US08. Som en medlem vil jeg kunne ... 85%
US30. Som en administrator vil jeg k... 50%
US32. Som en administrator vil jeg k... 50%
US33. Som en administrator vil jeg k... 50%
US31. Som en administrator vil jeg k... 50%
About page 50%
Fullføre Dashboard 50%
Legge til funksjon til favorite-knapp 15%
US23. Som et medlem vil jeg kunne ... 15%
Sprint 14 (19 April - 25 April) 74%
US08. Som en medlem vil jeg kunne ... 100%
"Active" side i nav baren er alltid ho... 100%
Contact burde ikke ligge i explore 100%
Linker i footer er ikke satt 100%
Passord krav på signup siden burde b... 100%
Flerspråklig nettside 40%
Ingen error melding fra forgotten pas... 100%
Serieinfo bug 100%
US30. Som en administrator vil jeg k... 100%
US32. Som en administrator vil jeg k... 100%
US33. Som en administrator vil jeg k... 100%
US31. Som en administrator vil jeg k... 100%
US34. Som en administrator vil jeg k... 100%
About page har blanda språk - Må ov... 100%
Default actor bilde w/h er ikke riktig 100%
Lage admin dashbord 25%
About page 62%
Fullføre Dashboard 60%
Legge til funksjon til favorite-knapp 30%
US23. Som et medlem vil jeg kunne ... 30%
Sprint 15 (26 April - 2 Mai) 60%


1/21
2/21
3/21
4/21
5/21
17 24 31 7 14 21 28 7 14 21 28 4 11 18 25 2 9 16 23
Serier i watched listen til brukeren li... 100%
Serier i favorited listen til brukeren li... 100%
Gjøre om actorinfo til person 100%
Serieinfo side crasher - fix 100%
Flerspråklig nettside 50%
Executive Producer kan bli undefined 100%
Bedre tilbakemelding til bruker ved r... 100%
US09. Som en gjest vil jeg kunne del... 100%
Mangler oversettelse på dashboard (... 100%
Link til media i watchlist 100%
Lage admin dashbord 50%
About page 75%
US07. Som en gjest vil jeg kunne få ti... 100%
Recommended system: Fjern allered... 25%
Fullføre Dashboard 70%
US25. Som et medlem vil jeg kunne l... 25%
Legge til funksjon til favorite-knapp 45%
US23. Som et medlem vil jeg kunne ... 45%
Sprint 16 (3 Mai - 9 Mai) 65%
Lage admin dashbord 75%
About page 87%
Recommended system: Fjern allered... 50%
Fullføre Dashboard 80%
Flerspråklig nettside 60%
US25. Som et medlem vil jeg kunne l... 50%
Legge til funksjon til favorite-knapp 60%
US23. Som et medlem vil jeg kunne ... 60%
Sprint 17 (10 Mai - 16 Mai) 74%
US27. Som et medlem vil jeg kunne ... 100%
Lage admin dashbord 100%
Filmer/Serier må lagres i databasen v... 100%
Kort blir ikke borte etter approve/de... 100%
About page 100%
Fikse dato på filmsider 100%
Web scraping headers bilde sjekk ikk... 100%
Flerspråklig nettside 70%
Recommended system: Fjern allered... 75%
Fikse Internal server error på serie 50%
Midlertidige bilder på film/serieinfo 50%
Fullføre Dashboard 90%
US25. Som et medlem vil jeg kunne l... 75%
Legge til funksjon til favorite-knapp 75%
Sammenkoble pug med uikit 50%
US23. Som et medlem vil jeg kunne ... 75%
Bruker kan legge til flere reviews for ... 60%
Sprint 18 (17 Mai - 23 Mai) 95%
Bedre forklaring på signup siden 100%
Implementere livesearch på hele sid... 100%
Statistikk må oversettes 100%
Rydd og kommenter Systems 100%
Recommended system: Fjern allered... 100%
Bruker må få mail når review blir app... 100%
Fikse Internal server error på serie 100%
Oversette post review-knapp 100%
About SweetAlert 100%
Skaff tmdb-Info hver 24t 100%
Midlertidige bilder på film/serieinfo 100%
Ordne oversettelse til Dashboard 100%
Flerspråklig nettside 80%
Fullføre Dashboard 100%
Fikse scrollbar 100%
US14. Som en gjest vil jeg kunne se u... 100%
US25. Som et medlem vil jeg kunne l... 100%
Legge til funksjon til favorite-knapp 100%
Trykk på id for å fylle inn på edit revi... 100%



Kommentar tidsbruk

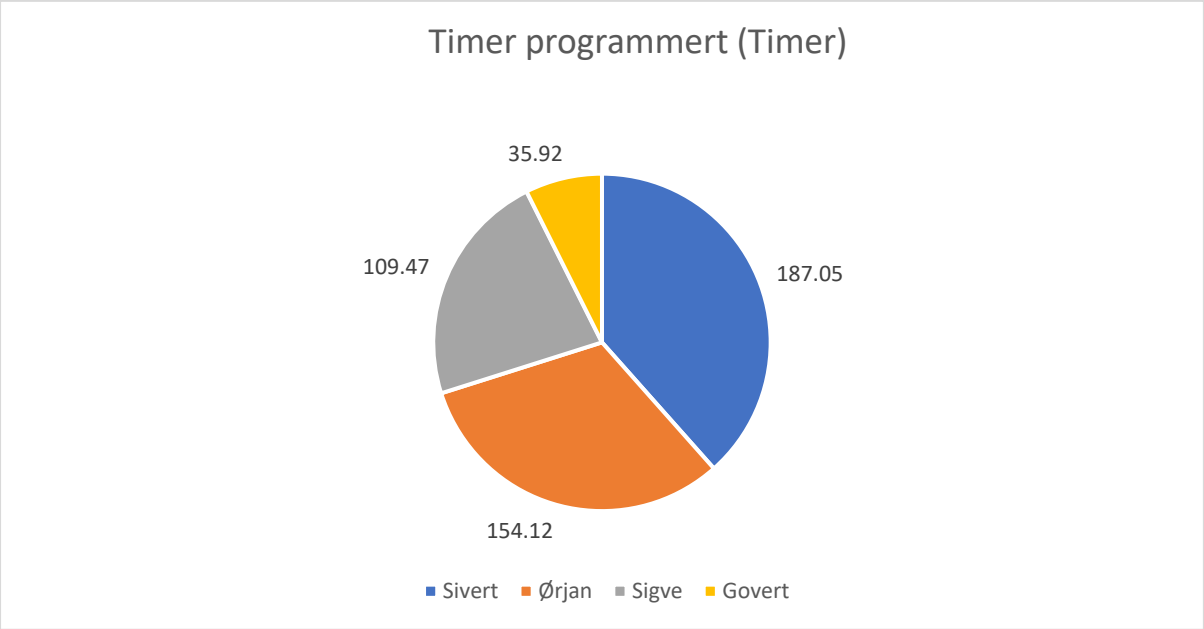
Tidlig i prosjektets planleggingsprosess så vi for oss å fullføre hoveddelene av prosjektet i god tid, slik at vi kunne ha mer ledig tid under eksamensperioden. I realiteten varte prosjektet hele semesteret, med forbedringer og problemløsning helt ut i uke 21. Vi har muligens undervurdert både arbeidsmengden prosjektet innebar, og hvor mye fokus de andre fagene krevde. Vi ble til tider nødt til å nedprioritere prosjektet for andre emners obligatoriske oppgaver og eksamener. Alt tatt i betraktning har vi oppnådd det vi ønsket. Med å sette av tid til arbeid når det var mulighet og være fleksible med tanke på tiltak for å løse utfordringer, klarte vi å produsere et produkt vi er stolte av.

Vi brukte kortere tid enn det vi hadde estimert på forhånd. Dette regner vi med skyldes en kombinasjon av effektivt arbeid, uferdige user storys og at mange user storys ble bearbeidet med tilleggskort. Totalen av tilleggskortene og user storyene tok samlet mer tid enn det vi estimerte at user storyene skulle ta. Det regner vi med at var på grunn av alle tilleggskortene og feilfiksing.

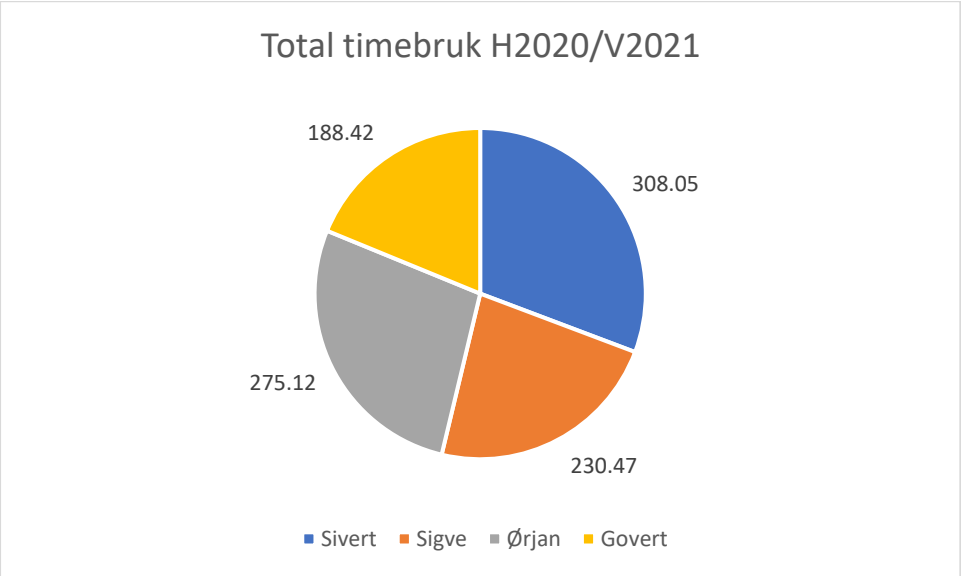
Timeforbruk

Vi har kategorisert timeforbruket vårt inn i 2 grupper. Den første er tid vi har brukt på ren programmering og ting som angår koden. Den andre delen er timer vi har brukt møter, dette er da samlet tid, og ikke delt inn per person ettersom at alle stort sett har vært på alle møtene.

Under ser du grafen for programmerte timer. (Se vedlagt fil «Timer» for detaljert oversikt over hver aktivitet)



Vi har også hatt en god del møter, rapportskriving og lignende arbeidsoppgaver. I møtene har vi for det meste gått igjennom status på prosjektet og sett på nye ting. I møtene har vi en total på 33 timer i høst og en total på 88 timer i vår (Se vedlagt fil «Timer» for detaljert oversikt).



Roller og arbeidsoppgaver

Roller

Nedenfor har vi skrevet navn på gruppens medlemmer og rollene de har hatt som hovedområde. Teamet har vært flinke til å holde god kontakt, samarbeide og gjøre ulike områder. Dette er altså bare hvor gruppemedlemmene har arbeidet mest, for vi har ikke hatt noen store begrensninger for arbeidsområder og ansvar. Det var viktig for oss at alle i gruppen fikk prøve seg på hvert område.

Gruppemedlem	Rolle(r)
Govert Dahl	Front-end, rapport
Sigve Eliassen	Front-end
Sivert Heisholt	Scrum Master, Back-end
Ørjan Dybevik	Full-stack

Arbeidsoppgaver

Front-end jobber med Pug-filer og Javascript på klient-siden. Back-end holder på med alt som skjer på server-siden, dette gjelder da alt fra databasen til API-et og sending av informasjon til front-end. Full-stack tar seg av både back-end og front-end. Scrum Master sørger for at alt er klart til møtene og vi vet hva vi skal gå igjennom. Scrum Master tar også av seg prosjektmetodikk og prosjektorganisering. Vi hadde også en som jobbet med rapporten, og fant ut hva vi trengte/manglet.

Dokumentasjon

Dokumentasjonsmetoden vi har brukt er JSDoc. Vi har også hatt møter 2 ganger i uken, der vi har gått over hva hver av oss har jobbet med. Da har vi også forklart hvordan andre kan bruke koden. Dette har fungert veldig bra, og vi har selvfølgelig chattemuligheter der vi kan spørre hverandre dersom ting er uklart. Vi har også prøvd å lage så modulært og forståelig kode som mulig, dette er jo en utfordring ettersom vi ikke er noen eksperter i Javascript. Vi lærer noe nytt hele tiden og finner ut av bedre metoder å skrive koden på, derfor ender vi opp med en ganske god miks av kodestrukturen. Dette håndterte vi ved å gå over nye metoder på møtene, og prøve å få en standard. Dersom en bedre metode ble lagt frem, så ville vi prøve å holde oss til den igjen. Dersom vi fikk tid, så gikk vi tilbake og skrev om gammel kode.

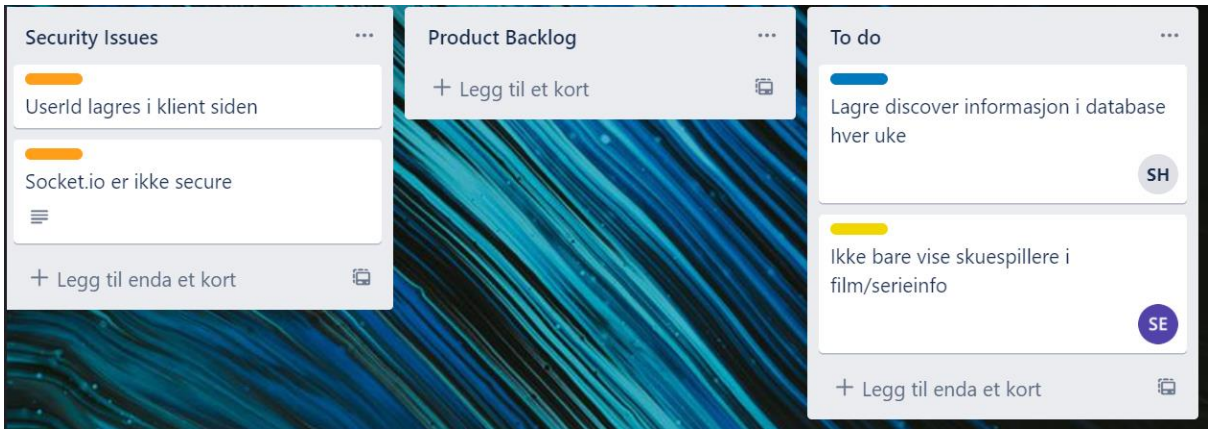
Fremgangsmåte

I arbeidet med dette prosjektet valgte vi å fordele arbeidsområdene nokså likt for at alle skulle lære mest mulig. Alle prøvde seg på de ulike feltene i starten, deretter så valgte hver enkelt person det de hadde lyst til å gjøre. Etter hvert som vi fikk mer erfaring og kunnskap, spesielt i slutten, så jobbet folk med det meste. Dette var for å fullføre prosjektet til tiden.

Diskusjon og refleksjon

Gjennomgang av forbedringspotensialet

Mangler og uløste problemer



I bildet ovenfor er det som gjenstår pr. 27.05.2021 på gruppens Trello-tavle. Som det fremstår på bildet lagres BrukerID i klienten, dette er ikke sikkert. Dette kan som nevnt tidligere løses med å lagre en SessionID i klienten. Det andre sikkerhetsproblemet er at Socket.oi ikke er sikker. Altså det er ikke gjort noen sjekk gjennom serveren om koblingene er verifiserte.

Under «To do» på trello har vi to uløste kort. Informasjonen fra «Discover Movies» på fremsiden kunne vi også ha lagret ukentlig på databasen. For hvis TMDB som vi henter informasjonen fra hadde vært nede nede, så kunne applikasjonen fortsatt vært kjørbar. Det kunne også blitt gjennomført et backup-system ved at informasjon blir hentet fra vår database istendefor API dersom det er nede. Kortet om film-/serieinfo omhandler informasjonssidene for filmer og serier ved at det kun viser navn på skuespillere, men ikke regissør, crew og

lignende. Alle disse uløste problemene ville vi gjennomført i forkant av en eventuell lansering.

Forbedringer før lansering

I tillegg til uløste brukerhistorier kunne vi hatt følgende forbedringer før lansering av en kommersiell versjon:

- En fullstendig oversettelse av informasjonen om filmer/serier fra API-et. På dette området har vi allerede oversatt noe, men ikke alt. Dersom Filmatory har oversatt mesteparten av nettsiden på flere språk ville vi hatt et konkurransefortrinn i markedet.
- Søking på personer. I dagens versjon kan en kun søke opp filmer og serier. Før en eventuell lansering ville det også vært mulig å søke på personer.
- Dark mode: Under testing fase 2 fikk vi et forslag om å legge til dark-mode på siden. Dette syntes vi er en god ide som ville gjort nettsiden attraktiv for mange brukere.
- Mer sosial: Vårt fokus i markedsføringen ved lansering ville vært at dette er en sosial og interaktiv nettside. Så i forkant av lansering ville vi lagt til funksjonalitet som blant annet gjør det mulig å se andre brukere sine lister og kommunikasjon mellom brukere.
- Vi burde også ha en form for antibot i kontakt-skjemaet for å forhindre at uvedkommende spammer innboksen vår eller krasjer siden.

Vurdering av prosessen

Samarbeidet i gruppen har fungert bra. Med faste møter, åpne diskusjoner og mye koding har vi klart å oppnå det meste vi hadde planlagt. Valget vårt om at prosjektet skulle bestå av å lage en filmdatabase var et godt valg. Gjennom dette prosjektet har vi kunnet arbeide med flere ulike områder og erfare mye innen denne typen programmering. Siden det var så mange ulike ting vi skulle lage for Filmatory så kunne vi i gruppen dele opp ulike områder som gjorde det enkelt å arbeide med flere ting samtidig. Når vi møttes startet vi med å fortelle hvordan det gikk, hva vi jobber med og hva vi skulle jobbe med. Deretter begynte vi å arbeide, så møtene våre var som regel «arbeidsmøter». De faste møtene har vist seg å være svært effektive for arbeidet. Vi møttes til faste tider og dager hver uke som passet for alle. Møtene var på ulik hyppighet avhengig av hvor langt vi var i prosessen og om andre emner

krevde mer fokus. Dette gjorde at vi fikk samkjørt mye av arbeidet og at alle hadde god oversikt over hvordan prosjektet gikk.

Tilpassing av kode

Vi har en kode som vi har hentet fra noen andre. Orginal koden ser da slik ut, den tilfeldig gjør et array.

```
function shuffleArray(arr) {
    for (let i = arr.length - 1; i > 0; i--) {
      const j = Math.floor(Math.random() * (i + 1));
      [arr[i], arr[j]] = [arr[j], arr[i]];
    }
  console.log(arr);
}
```

Kilde: https://www.w3docs.com/snippets/javascript/how-to-randomize-shuffle-a-javascript-array.html

Koden er tilpasset til at den lagrer funksjonen i en variabel og returnerer et array.

```
    /**
     * Randomizer et array
     * @param {String} array  Array som skal bli randomiza
     * @returns Et array som er shuffla
     * @author Internett
     * https://www.w3docs.com/snippets/javascript/how-to-randomize-shuffle-a-
javascript-array.html
     * Hentet dato: 23/04/2021
     */
    shuffleArray: function(array) {
        for (let i = array.length - 1; i > 0; i--) {
            const j = Math.floor(Math.random() * (i + 1));
            [array[i], array[j]] = [array[j], array[i]];
        }
        return array;
    },
```

Det ble bare brukt dokumentasjonen fra de forskjellige pakkene som vi brukte og de forskjellige teknologiene.

Bibliografi

BCrypt. (u.d.). Hentet fra npmjs: https://www.npmjs.com/package/bcrypt

Body-Parser. (u.d.). Hentet fra npmjs: https://www.npmjs.com/package/body-parser

Connect-mongo. (u.d.). Hentet fra npmjs: https://www.npmjs.com/package/connect-mongo/v/2.0.3

Dotenv. (u.d.). Hentet fra npmjs: https://www.npmjs.com/package/dotenv

Express. (u.d.). Hentet fra npmjs: https://www.npmjs.com/package/express

Express-session. (u.d.). Hentet fra npmjs: https://www.npmjs.com/package/express-session

Express-socket.io-session. (u.d.). Hentet fra npmjs: https://www.npmjs.com/package/express-socket.io-session

I18n. (u.d.). Hentet fra npmjs: https://www.npmjs.com/package/i18n

JSONWebToken. (u.d.). Hentet fra npmjs: https://www.npmjs.com/package/jsonwebtoken

Mongoose. (u.d.). Hentet fra npmjs: https://www.npmjs.com/package/mongoose

Multer. (u.d.). Hentet fra npmjs: https://www.npmjs.com/package/multer

Node-fetch. (u.d.). Hentet fra npmjs: https://www.npmjs.com/package/node-fetch

Nodemailer. (u.d.). Hentet fra npmjs: https://www.npmjs.com/package/nodemailer

Nodemailer-SMTP-Transport. (u.d.). Hentet fra npmjs: https://www.npmjs.com/package/nodemailer-smtp-transport

Pug. (u.d.). Hentet fra npmjs: https://www.npmjs.com/package/pug

Risikovurdering. (u.d.). Hentet fra Digitaliseringsdirektoratet: https://internkontroll-infosikkerhet.difi.no/risikostyring/risikovurdering

Socket.io. (u.d.). Hentet fra npmjs: https://www.npmjs.com/package/socket.io

TMDB. (u.d.). Hentet fra The Movie Database: https://www.themoviedb.org/

Winston. (u.d.). Hentet fra npmjs: https://www.npmjs.com/package/winston

Vedlegg A

Oppdatert kravspesifikasjon

Rader markert som grønne er fullførte kravspesifikasjoner og rader markert med rødt er kravspesifikasjoner som vi ikke fikk fullført.

UseCase	Beskrivelse	Forklaring
US01	Som en gjest vil jeg kunne se informasjon om filmer	Dette usecaset vil si at vi har en egen side til hver film eller serie. På denne siden kan vi se f.eks filmcover, bilder, trailer, tittel, beskrivelse, lanseringsdato, skuespillere og regissør til filmen. I tillegg vil vi kunne se f.eks IMDB score til filmen.
US02	Som en gjest vil jeg kunne se informasjon om serier	
US03	Som en gjest vil jeg kunne se en "overview" slide	Dette vil da si forsiden av Filmatory, her kan vi utforske filmer og serier, samt en oversikt over populære sjangere.
US04	Som en gjest vil jeg kunne se statistikker (Diagram)	På forsiden kan vi se populære sjangere
US05	Som en gjest vil jeg kunne opprette meg en brukerkonto	Som en gjest så vil det være mulig å bli medlem på siden for å benytte seg av flere funksjoner på nettsiden.
US06	Som en gjest vil jeg kunne se trailere til filmer eller linker til YouTube	På filmer og serie sider kan du se trailer til mediet det angår. (youtube)
US07	Som en gjest vil jeg kunne få tilfeldig forslag innenfor ulike sjangrer	For eksempel kan du få en oversikt over filmer som er tilfeldig utvalgt av f.eks action, eller komedie.
US08	Som en gjest vil jeg kunne se relaterte filmer/serier til den jeg ser på	Når du er på en film/serie-side, så kan du se filmer/serier som er lignende, eller relatert til filmen/serien du ser på.

US09	Som en gjest vil jeg kunne dele filmen/serien via sosiale medier	Du vil kunne trykke på en knapp for å dele den på f.eks veggen din på facebook.
US10	Som en gjest vil jeg kunne se en kalender over kommende filmpremierer	Du vil kunne se en liste med oppkommende filmer.
US11	Som en gjest vil jeg kunne se en kalender over serier sine release-dates	Du vil kunne se en liste over oppkommende serier
US12	Som en gjest vil jeg kunne se historiske filmer fra samme dato på hjemmesiden	Du vil kunne se filmer som kom ut samme tidsperiode på hjemmesiden.
US13	Som en gjest vil jeg kunne lese anmeldelser på filmer og serier	Kunne lese anmeldelser
US14	Som en gjest vil jeg kunne se ulike film/serie-lister av andre brukere	Du vil kunne gå inn på profilen til andre brukere, og se listene av filmene og seriene dems.
US15	Som en gjest vil jeg kunne lese quotes fra filmen.	Som f.eks “I’ll be back” - Terminator
US16	Som en gjest vil jeg kunne se interessante og random fakta om filmen/produksjonen	Som f.eks ekte hendelser filmen er bygget på, ting som skjedde under innspilling eller hvordan skuespillere fikk rollene.
US17	Som et medlem vil jeg kunne logge inn	Du vil kunne logge inn på din egen profil
US18	Som et medlem vil jeg kunne logge av	Du vil kunne logge ut av din profil
US19	Som et medlem vil jeg kunne rate filmer/Serier	

US20	Som et medlem vil jeg kunne omtale filmer/serier	Du vil kunne legge igjen en anmeldelse av en film, og samtidig kunne vurdere filmen fra en skala fra 1 til 5 stjerner.
US21	Som et medlem vil jeg kunne legge til en film i sett liste	Du vil kunne markere hvilke filmer og serier du har sett og legge dem til i en liste.
US22	Som et medlem vil jeg kunne legge til en serie i sett liste	
US23	Som et medlem vil jeg kunne se personlige statistikker (Grafer)	Du vil kunne se statistikker som er knyttet til bruker informasjonen (Favoritter, watched etc.)
US24	Som et medlem vil jeg kunne kontakte support	Du vil kunne kontakte dem som driver siden med f.eks forbedringer, eller rapporteringer.
US25	Som et medlem vil jeg kunne lage lister over filmer og dele listen i ulike medium.	Du vil kunne lage en liste av forskjellige filmer og serier
US26	Som et medlem vil jeg kunne kommunisere med andre om film/serier på et forum.	Forumsystem for medlemmer.
US27	Som et medlem vil jeg kunne kommunisere med administrator/support	Du vil kunne ha en dialog med administrator/support over epost.
US28	Som et medlem vil jeg kunne legge inn interessante fakta om filmer/serier	Ikke prioritert, men legg igjen beskrivelse om den blir lag til
US29	Som et medlem vil jeg kunne legge inn quotes fra filmene	Som f.eks “I’ll be back” - Terminator (Denne betyr at du kan legge inn quotes som medlem)

US30	Som en administrator vil jeg kunne endre brukere sine reviews	Som en administrator vil du kunne redigere en annen bruker sin anmeldelse av en film.
US31	Som en administrator vil jeg kunne slette brukere sine reviews	Som en administrator vil du kunne slette en annen sin bruker sin anmeldelse av film/serie
US32	Som en administrator vil jeg kunne godkjenne brukere sine reviews	Når brukeren skriver en anmeldelse av en film eller serie så kan administratorene få en oversikt over de nye, samt innholdet og hvem som har skrevet de, også kan de enten godkjenne, eller avslå anmeldelsen. Når anmeldelsen blir godkjent vil andre brukere kunne se anmeldelsen av filmen eller serien.
US33	Som en administrator vil jeg kunne avslå brukere sine reviews	
US34	Som en administrator vil jeg kunne deaktivere brukere	Som en administrator så vil du ha muligheten til å deaktivere brukere, slik at de ikke lenger kan logge på kontoen sin.
US35	Som en administrator vil jeg kunne slette/endre ting på forumet	Som en administrator så vil du kunne gjøre det du vil med det andre brukere har lagt ut på forumet.
US36	Som en administrator vil jeg kunne godkjenne fakta skrevet av medlemmer på film/serier sider.	Som en administrator vil du kunne faktasjekke og enten redigere, godkjenne, eller slette faktaene brukere har skrevet om film/serie.
US37	Som en administrator vil jeg kunne godkjenne quotes skrevet av medlemmer på film/serier side.	Som en administrator vil du kunne godkjenne om quotene er riktig eller ikke, og velge om du skal endre, godkjenne eller slette quoten.

US38	Som en bruker av nettsiden vil jeg kunne søke etter film/serie	Du vil kunne søke opp alle filmer og serier på headeren av alle sidene på filmatory.

Vedlegg B

1 Nettadresse til hovedside:

https://filmatoryeksamen.herokuapp.com/

OBS: Webserveren Heroku går i dvale etter en liten stund uten trafikk. Når du først laster inn, vil det ta opptil 1 minutt før den er i gang igjen. Det er bare å laste inn på nytt til du kommer inn på siden.

2 Brukerkontoer:

2.1 Brukerkontoer til bruk av sensor/faglærer.

Her har vi laget to kontoer, en administratorkonto, og en standardkonto.

Brukernavn	Passord	Administrator
sensurbruker@appeksamen.no	APP2000eksamen	Nei
sensurbrukeradmin@appeksamen.no	APP2000eksamen	Ja

2.2 Reservekontoer:

Her har vi laget to ekstra kontoer du kan velge å ta i bruk.

Brukernavn	Passord	Administrator
reservebruker@appeksamen.no	APP2000reserve	Nei
reservebrukeradmin@appeksamen.no	APP2000reserve	Ja

3 Veiledning til faglærer/sensor

Her finner du en veiledning til de forskjellige funksjonene på siden.

OBS: Vi regner med at du bruker norsk som valgt språk og har forklart og lagt til lenker deretter.

3.1 Hjem-side

3.1.1 Introduksjon

https://filmatoryeksamen.herokuapp.com/

Dette er forsiden til Filmatory.

3.1.2 Valg av språk

Språkvalget finnes oppe til høyre i navigasjonsbaren.

3.1.3 Innlogging

Dette gjør du ved å trykke på «Logg Inn» knappen øverst til høyre på siden vedsiden av språk knappen i navigasjonsbaren.

For å unngå å logge inn og ut, velg gjerne en administratorkonto.

3.1.4 Glemt passord

Dette er tilbake stilling av passord via epost-adresse. Siden vi kun har gjort klar testbrukere, har vi vedlagt en video som viser prosessen.

Filnavn: **PassordTilbakestillingViaMail.mp4**

3.2 Se informasjon om en media

https://filmatoryeksamen.herokuapp.com/no/mediainfo/filminfo/632357

Vi har lagt inn en lenke over til en film som har noen brukeranmeldelser slik at du får en realistisk opplevelse av en filmside.

På denne siden kan du se det meste av viktig informasjon om filmen eller serien.

3.3 Skrive eller se anmeldelser

Om du ønsker å prøve ut funksjonen for å skrive anmeldelse kan det gjøres på hvilken som helst film. Disse må godkjennes av administrator. Dette blir forklart på punkt 4.1.1

3.4 Se informasjon om en person

Om du har lyst til å se informasjon om en person kan du klikke deg inn på en person enten ved hjelp av slideren under film/serie informasjonen eller trykke på navnet/lenken til regissøren eller manusforfatteren på en medieside.

3.5 Utforsk

3.5.1 Kommende

3.5.1.1 Kommende filmer

Side over kommende filmer, disse er sortert etter dato.

3.5.1.2 Kommende TV-serier

Side over kommende serier, disse er sortert etter dato.

3.5.2 Lists

Side over lister lagd av andre brukere.

3.5.3 Filmer

Filmside som er sortert etter popularitet førte gang du kommer inn. Bruk knappene til å sortere etter preferanse.

3.5.4 Tv-serier

Serieside som er sortert etter popularitet førte gang du kommer inn. Bruk knappene til å sortere etter preferanse.

3.6 Om

På denne siden må du gjerne lese informasjonen vi har skrevet på denne siden. Hvis du blar litt ned på siden så kommer du til kontakt skjemaet.

Filmer/Serie statistikken som vises på denne siden representerer hva vi har lagret fra vår egen database, ikke hva vi tilbyr på siden.

3.6.1 Kontakt

For å komme direkte til kontakt-skjemaet kan du også trykke på «Kontakt oss» i footeren nederst på siden, eller gå via «Om».

3.7 Bruker dashbord

3.7.1 Bytt Passord

Her kan du bytte passord til brukeren. Det er samme krav og fremgangsmåte som om du skulle laget en bruker.

3.7.2 Favoritter

Her er oversikten over filmer og serier som er lagt til som favoritt. Det er mulig å slette filmer og serier fra denne listen ved å trykke på det lille krysset nede til venstre på hver media.

3.7.3 Sett liste

Helt lik som favoritter, men en annen liste. På engelsk er den kalt «Watchlist». Dette er en oversikt over filmer du gjerne allerede har sett.

3.7.4 Min liste

Her kan du opprette lister. Du skriver inn et navn du ønsker på listen, og går videre til en film eller serie, trykker legg til i liste, og velger listen fra menyen som kommer opp.

3.7.5 Statistikk

Her er statistikker til brukeren.

3.7.6 Profil

Her kan du lage et nytt brukernavn og laste opp profilbilde til brukeren.

4 Administrator

Når du logger inn på en administrator-konto vil du få et nytt valg under «konto» menyen din. Dette valget heter «Admin Dashbord» Her har du tilgang til flere ulike funksjoner som er listet nedover:

4.1 Anmeldelser

4.1.1 Godkjenning

Alle anmeldelser fra brukere vil gå gjennom godkjenning av administrator. Disse godkjennes her.

Om du skrev en anmeldelse med en av brukerne vil den komme opp her.

Du trykke enkelt på IDen (Anmeldelses ID), den vil da kopieres og limes inn automatisk i inputboksen. Herfra kan du velge å godkjenne, den vil da bli synlig inne på filmen det gjelder, eller du kan avslå. Om du avslår vil brukeren få mail med en begrunnelse om hvorfor den ble avslått.

4.1.2 Rediger

Vi har også lagt inn en mulighet for å redigere anmeldelser. Vi ser ikke for oss at den vil bli mye brukt, men tenkte det kunne være greit å ha med.

Du begynner med å velge i radioboksene om du skal redigere en anmeldelse fra en TV-serie eller film. Deretter må du skrive inn ID til filmen, denne får du tak i inne på URL-en til filmen, dette er de siste nummerene. For eksempel 460465.

Deretter trykker du på knappen «Hent» og alle anmeldelser knyttet din filmen vil komme opp.

Fra her er det samme prosedyre som godkjenning for å hente ID til anmeldelsen. Deretter kan du skrive ny tekst og gi ny vurdering.

4.1.3 Slett

For å slette anmeldelser er det akkurat den samme prosedyre som når du skal redigere. Etter at du har hentet trykker du på IDen til anmeldelsen og trykker slett. Denne slettes da fra filmen og vil ikke vises lenger på filmsiden.

4.2 Språk

4.2.1 Legg til

Om du velger å legge til et nytt språk finnes det instrukser inne på siden. Etter at du trykker lagre vil siden lastes inn på nytt og du vil se under språkmenyen oppe til høyre at du har et nytt valg.

4.2.2 Rediger

Om du vil prøve å redigere språket du nettopp la til flytter vi oss til «Rediger». Her velger du språket fra listen og det vil komme opp et tekstområde du kan gjøre endringer i.

Det som er viktig er at du ikke sletter selve språkvariablene, disse er lett gjenkjennelige da de alltid er skrevet i store bokstaver, og oftest har en forklarende beskrivelse. Teksten til venstre er variabler og til høyre er teksten som vises for brukere.

Om du redigerer det språket du nettopp lagde vil det automatisk være engelsk.

Du kan gi ett forsøk på å redigere

"NAV_HOME": "Home",

til

"NAV_HOME": "DIN TEKST HER".

Det viktigste for at det skal fungere er at du kun redigerer teksten mellom anførselstegnene. Om jeg redigerer til svensk vil det da se slik ut:

"NAV_HOME": "Hem",

Det skal alltid være kommategn etter det siste anførselstegnet, bortsett fra den siste linjen på hele dokumentet.

4.2.3 Slett

For å slette språket du nettopp lagde må du skrive det engelske navnet. Så om du lagde for eksempel svensk, vil det da være «swedish».

Husk å bruke kun små bokstaver når du sletter.

4.3 Brukere

4.3.1 Ban/Unban(utestengelse)

Utestengelse fungerer slik at du skriver inn e-postadressen til brukeren som skal utestenges. Når en bruker utestenges, vil ikke brukeren kunne logge inn igjen. Du kan prøve dette med en av de andre brukerne du har fått tilgang til, eller du kan prøve å logge inn med denne brukeren under. Denne brukeren er en testbruker og er allerede utestengt. Dermed kan du også prøve å fjerne utestengelsen.

Brukernavn	Passord
utestengtbruker@filmatoryeksamen.no	APP2000eksamen

4.4 Support

4.4.1 Billetter/Tickets

Her vises support tickets sendt inn fra brukere. Disse sendes inn i kontaktskjema inne på «Om».

For å svare på disse limer du inn billettID i inputfeltet. Deretter en respons/svar og trykker send. Dette svaret sendes direkte til brukermailen som har sendt inn denne ticketen.

5 Mappestruktur

Vår kode finner du i alle mappene, vi har dokumentert det som trengs å dokumentere. Alle filene har vi laget med unntak av: «package.json» og «package-lock.json»

Vedlegg C

Gjesteforelesninger

Visma

GDPR

Visma tok for seg sikkerhet og GDRP under gjesteforelesningen. For applikasjonen vår, så er ikke GDRP så relevant. Vi lagrer ingen personinfo om brukerne eller identifiserende informasjon. Vi benytter heller ingen hemmelig innsamlingsmetode og er helt åpen om hva som blir samlet inn av data. Det som vi kanskje skulle hatt er en cookie melding, som brukeren må akseptere. Det eneste vi bruker cookies til er å huske session til brukeren.

Det som kan skje, som Visma nevner er ustrukturert data. Vi har tekstfelt for anmeldelser som brukeren sender inn. Denne informasjonen blir tilgjengelig for alle med administrator tilgang og dersom denne inneholder sensitiv informasjon, så vil ikke de være riktig håndtert.

Visma bruker mange verktøy for å opprettholde en standard og god håndtering av persondata. De har også egne penetrasjonstester med bruk av ulike metoder og verktøy. I vår applikasjon så har vi bare testet de mest vanlige sikkerhetshullene, penetrasjonstester kunne vært en ide å teste på vår applikasjon. Spesielt når nye funksjoner ble gjennomført.

Ibexa og Aplia

Ibexa tok for seg datasikkerhet i webapplikasjoner og sikkerhetsløsninger i Symfony og Github. De snakket for eksempel om passord og hvordan dette burde håndteres, og snakket om at det er viktig å lese manualen. Vår applikasjon skal også håndtere passord for brukeren. De snakket mer rettet mot PHP, og der finnes det forskjellige funksjoner og måter for å hashe passord. Vi bruker npm pakken «bcrypt» til denne operasjonen.

Det er et open source bibliotek som hjelper til med hashing av passord. Ettersom det er open source så betyr det at alle kan se koden og fikse hull dersom noe skulle dukket opp. Dette gir økt sikkerhet.

De gikk også igjennom andre typer sikkerhetsrisikoer som er ganske vanlige feil/angrep. Injections er en veldig populær angrepsmetode hvor de snakket om de ulike injections angrepene som kan bli gjort, for eksempel SQL-injection. I vår applikasjon så har vi veldig god inndata kontroll og et vanlig NoSQL-injection angrep fungerte ikke på applikasjonen vår.

Applikasjonen godtar heller ikke svake passord, vi har en egen regex for å validere nettopp dette. Når du forsøker å lage ett passord kreves det at det er minimum 8 tegn, og maks 20. Det kreves også at du har minst 1 tall, 1 stor bokstav, og en liten bokstav. Det er ikke krav til å bruke tegn, men valideringen godtar også dette. Vi har også feilmeldinger om passordet ikke tilfredsstiller krav, om mailen som blir brukt allerede finnes i databasen, og om passord ikke matcher.

MVC-konseptet som Aplia fortalte om har vi implementert i NodeJS.

Model – Handling

Delen av applikasjonen som skal håndtere endringer i databasen skjer her.

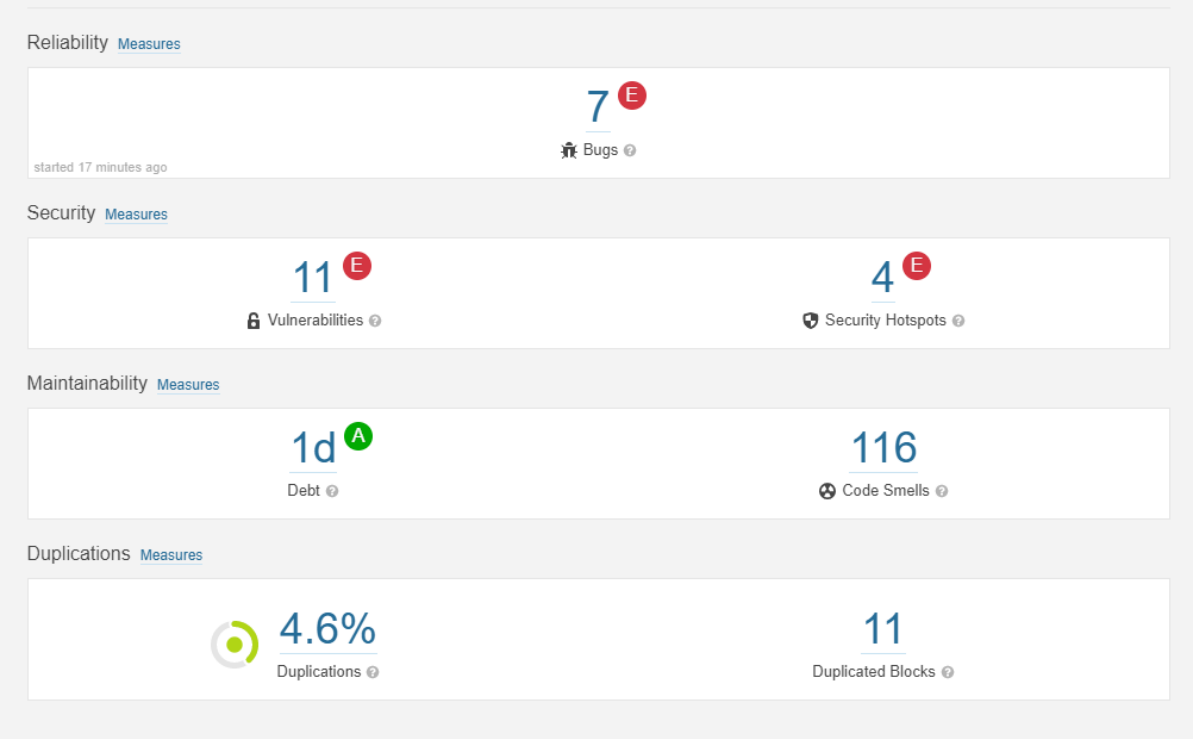
View – Views(Pug/Html)

Det brukeren ser ligger her, altså sidene som sendes til klienten.

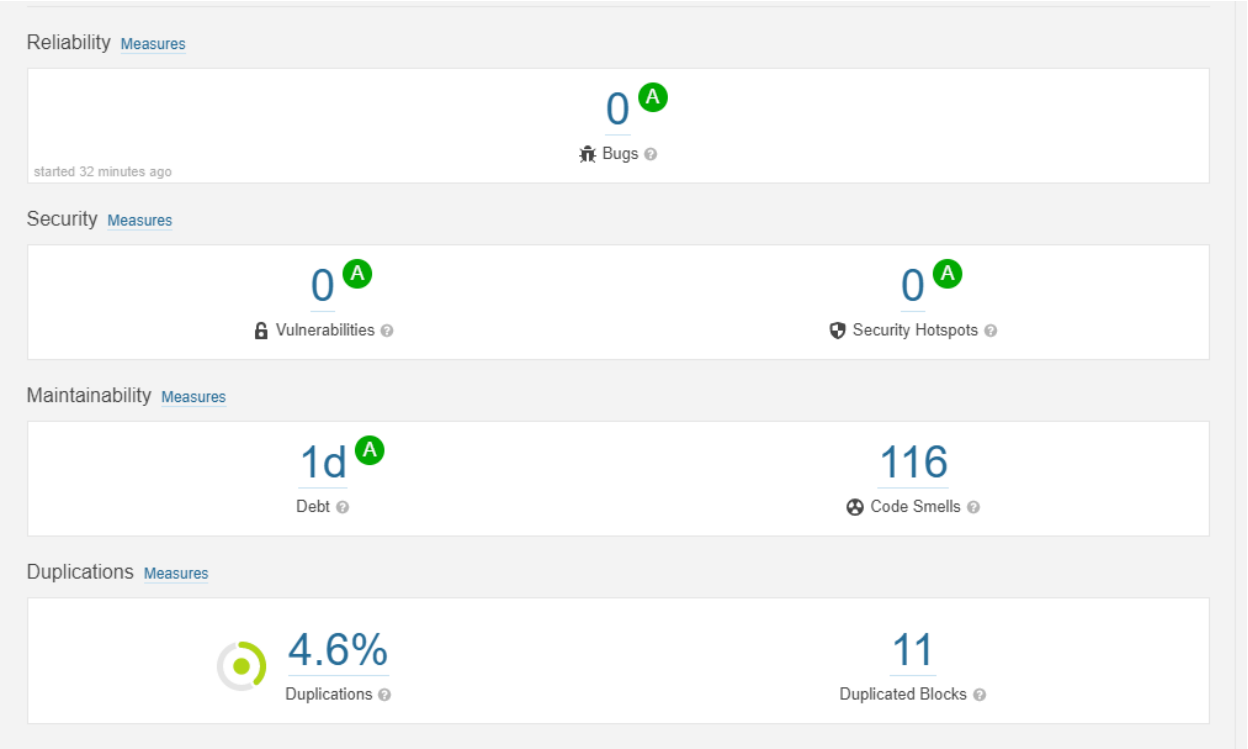
Controller & Routes – Routing/Controller

Logikken til nettstedet. Denne delen knytter models og views. Vi kaller modellene for å hente data, og viderefører den dataen til views som blir sendt til klienter.

Vi brukte tipset til Ibexa og kjørte en analyse av prosjektet på Sonarcloud. Som du ser i bildet under så fikk vi en del feil, disse feilene var enkle å fikse.



Vi gikk igjennom alle bugs og sikkerhetshull og fikk fikset de aller fleste, noen var false positive som vil si at analyse programmet ikke visste riktig.



Vedlegg D

Sivert – 233518

Høst 2020

Vi hadde bare møter som var planlegging og arbeid med obligatoriske oppgaver i høst. Dette ble gjort sammen så denne delen er lik på alle sin timeliste.

Uke 35

Dato	Oppgave	Tidsbruk (timer)
24/08/202	Brainstorming	2

Uke 36

Dato	Oppgave	Tidsbruk (timer)
03/09/2020	Sett opp arbeidsmiljø	2

Uke 37

Dato	Oppgave	Tidsbruk (timer)
07/09/2020	Designe prototype	1
08/09/2020	Videre design av prototype	1
10/09/2020	Planlegging av grafer/data/innhold	2

Uke 38

Dato	Oppgave	Tidsbruk (timer)
15/09/2020	Videre planlegging	2

Uke 39

Dato	Oppgave	Tidsbruk (timer)
22/09/2020	Planlegging av applikasjonens struktur	2

Uke 40

Dato	Oppgave	Tidsbruk (timer)
01/10/2020	Arbeid på obligatorisk oppgave 1	4

Uke 41

Dato	Oppgave	Tidsbruk (timer)
08/10/2020	Arbeid på obligatorisk oppgave 1	2

Uke 42

Dato	Oppgave	Tidsbruk (timer)
15/10/2020	Generell planlegging av applikasjonen	4

Uke 43

Ingenting ble jobbet med i denne uken på grunn av andre fag.

Uke 44

Ingenting ble jobbet med i denne uken på grunn av andre fag.

Uke 45

Dato	Oppgave	Tidsbruk (timer)
05/11/2020	Arbeid på obligatorisk oppgave 2	4

Uke 46

Dato	Oppgave	Tidsbruk (timer)
10/11/2020	Arbeid på obligatorisk oppgave 2	2

Uke 47

Dato	Oppgave	Tidsbruk (timer)
17/11/2020	Arbeid på obligatorisk oppgave 2	2

Uke 48

Dato	Oppgave	Tidsbruk (timer)
24/11/2020	Arbeid på obligatorisk oppgave 2	2

Samlet tidsbruk

Møte timer: 33

Vår 2021

Uke 01

Programmering

Ingen programmering ble utført denne uken

Møter

Dato	Tidsbruk (timer)
05/01/2021	2
08/01/2021	2

Uke 02

Programmering

Ingen programmering ble utført denne uken

Møter

Dato	Tidsbruk (timer)
12/01/2021	2
14/01/2021	2

Uke 03

Programmering

Oppgave	Tidsbruk (timer)
US17. Som et medlem vil jeg kunne logge inn	0.5
US05. Som en gjest vil jeg kunne opprette meg en brukerkonto	0.5
Index - Discover movies fra tmdb	0.5
Index - Discover tvshows fra tmdb	0.25
Session	0.5
Satt opp server	5

Møter

Dato	Tidsbruk (timer)
19/01/2021	2
21/01/2021	2

Uke 04

Programmering

Oppgave	Tidsbruk (timer)
Docker / herokuapp	1
Koble sammen login/signup backend med frontend	0.25
Koble sammen session + sider	2

Møter

Dato	Tidsbruk (timer)
26/01/2021	4
28/01/2021	2

Uke 05

Programmering

Oppgave	Tidsbruk (timer)
Utvide tmdbHandler for mer fleksibilitet	1
Skrev om hele tmdbHandler - 4x speed	6

Møter

Dato	Tidsbruk (timer)
02/02/2021	2

Uke 06

Ingenting ble jobbet med i denne uken på grunn av andre fag.

Uke 07

Ingenting ble jobbet med i denne uken på grunn av andre fag.

Uke 08

Programmering

Oppgave	Tidsbruk (timer)
Highcharts	2

Møter

Dato	Tidsbruk (timer)
23/02/2021	2
25/02/2021	2

Uke 09

Møter

Dato	Tidsbruk (timer)
02/03/2021	1
04/03/2021	2

Uke 10

Programmering

Oppgave	Tidsbruk (timer)
Lage film/tv database schema	1
Backend favourites system	1

Møter

Dato	Tidsbruk (timer)
11/03/2021	3

Uke 11

Programmering

Oppgave	Tidsbruk (timer)
US38. Som en bruker av nettsiden vil jeg kunne søke etter film/serie	1
Sjekke om søket i searchbaren har bilde	0.16
Feilmelding login/signup/logout	3
US18. Som et medlem vil jeg kunne logge av	0.5

Møter

Dato	Tidsbruk (timer)
16/03/2021	2
18/03/2021	2

Uke 12

Programmering

Oppgave	Tidsbruk (Timer)
Logging	5
Håndtering av return meldinger	1
Test av add movie favourite	1
Oppdater tvSchema + movieSchema	0.16

Møter

Dato	Tidsbruk (timer)
23/03/2021	2
25/03/2021	2

Uke 13

Programmering

Oppgave	Tidsbruk (timer)
US21. Som et medlem vil jeg kunne legge til en film i sett liste	2

Møter

Dato	Tidsbruk (timer)
30/03/2021	1
01/04/2021	1

Uke 14

Programmering

Oppgave	Tidsbruk (Timer)
Flerspråklig	1
Rydd opp kode - Oppdater gammel kode	1
Implementer controllers for express - Gjør koden mer modulær	5
finn en god metode for route params til language	1

Møter

Dato	Tidsbruk (timer)
05/04/2021	4
06/04/2021	1.5
08/04/2021	2

Uke 15

Programmering

Oppgave	Tidsbruk (Timer)
Favorite/watched knapp må reloade siden for at bilde skal oppdatere	0.5
US22. Som et medlem vil jeg kunne legge til en serie i sett liste	1
Express middleware for "mal" av render og language middleware istedenfor router.all	3
US19. Som et medlem vil jeg kunne rate filmer/Serier	1
Optimaliser og organiser hvordan prosjektet starter opp	6
US20. Som et medlem vil jeg kunne omtale filmer/serier	9

Møter

Dato	Tidsbruk (timer)
13/04/2021	3
14/04/2021	2
17/04/2021	2

Uke 16

Programmering

Oppgave	Tidsbruk (Timer)
US08. Som et medlem vil jeg kunne se relaterte filmer/serier til de jeg ser på	2
US31. Som en administrator vil jeg kunne slette brukere sine reviews	0.16

Møter

Dato	Tidsbruk (timer)
22/04/2021	1.5
23/04/2021	1

Uke 17

Programmering

Oppgave	Tidsbruk (Timer)
US07. Som en gjest vil jeg kunne få tilfeldig forslag innenfor ulike sjangrer	1

Møter

Dato	Tidsbruk (timer)
27/04/2021	2

Uke 18

Ingenting ble jobbet med i denne uken på grunn av andre fag.

Uke 19

Programmering

Oppgave	Tidsbruk (Timer)
FIlmer/Serier må lagres i databasen vår Kort blir ikke borte etter approve/deny review	1 0.5
Web scraping headers bilde sjekk ikke funke	1

Møter

Dato	Tidsbruk (timer)
11/05/2021	1
14/05/2021	1.5

Uke 20

Programmering

Oppgave	Tidsbruk (Timer)
Bedre forklaring på signup siden	0.05
Livesearch funker bare på index	0.16
Statistikk må oversettes	1
Recommended system: Fjern allerede sett media fra lista	1
Rydda og kommenterte Systems	2
Bruker må få mail når review blir approved og denied	0.16
About SweetAlert	2
Skaff tmdbInfo hver 24t	0.16
US14. Som en gjest vil jeg kunne se ulike film/serie-lister lagd av andre brukere.	2
US25. Som et medlem vil jeg kunne lage lister over filmer og dele listen i ulike medium	10
Trykk på id for å fylle inn på edit review	0.25
Forbedre kode for userAuth	2
Logg inn bruker etter signup	0.25
TMDB info fra api språk parameter	1
Livesearch beholder ikke valgt språk	0.33
Tilbakemelding på sending av contact form på about	1
Oversettelse trengs på chart	0.5
urlPath + url er ikke tilgjengelig overalt	0.5

Møter

Dato	Tidsbruk (timer)
18/05/2021	1
20/05/2021	2.5
21/05/2021	2
22/05/2021	2
23/05/2021	4

Uke 21

Programmering

Oppgave	Tidsbruk (Timer)
Kommentert kode	4
Server crasher om du trykker på listen fort etter den blir lagd	1
Sjekk om bruker ikke har watch/favoritt i statistikk	0.14
US23. Som et medlem vil jeg kunne se personlige statistikker (Grafer)	14
US03. Som en gjest vil jeg kunne se en "overview" slide	2
US04. Som en gjest vil jeg kunne se statistikker (Diagram)	2
Bruker kan legge til flere reviews for samme media	1
Livesearch mangler oversettelse	0.41
Min liste mangler bilde om noe er i listen	0.5

Møter

Dato	Tidsbruk (timer)
24/05/2021	4
25/05/2021	4
26/05/2021	4
27/05/2021	4
28/05/2021	2
29/05/2021	2

Ekstra

Dette er tidsbruk for aktiviteter som har gått over alle ukene.

Oppgave	Tidsbruk (Timer)
Organisering og strukturering av applikasjonen	15
Testing av forskjellige kode strukturer og måter websiden skulle fungere	18
Fikse småbugs	18
Forbedring av gammel kode	12
Testing under programmering	5

Samlet tidsbruk

Programmert timer: 187.05

Møte timer: 88

Totalt: 275.05

Sigve – 233511

Høst 2020

Vi hadde bare møter som var planlegging og arbeid med obligatoriske oppgaver i høst. Dette ble gjort sammen så denne delen er lik på alle sin timeliste.

Uke 35

Dato	Oppgave	Tidsbruk (timer)
24/08/202	Brainstorming	2

Uke 36

Dato	Oppgave	Tidsbruk (timer)
03/09/2020	Sett opp arbeidsmiljø	2

Uke 37

Dato	Oppgave	Tidsbruk (timer)
07/09/2020	Designe prototype	1
08/09/2020	Videre design av prototype	1
10/09/2020	Planlegging av grafer/data/innhold	2

Uke 38

Dato	Oppgave	Tidsbruk (timer)
15/09/2020	Videre planlegging	2

Uke 39

Dato	Oppgave	Tidsbruk (timer)
22/09/2020	Planlegging av applikasjonens struktur	2

Uke 40

Dato	Oppgave	Tidsbruk (timer)
01/10/2020	Arbeid på obligatorisk oppgave 1	4

Uke 41

Dato	Oppgave	Tidsbruk (timer)
08/10/2020	Arbeid på obligatorisk oppgave 1	2

Uke 42

Dato	Oppgave	Tidsbruk (timer)
15/10/2020	Generell planlegging av applikasjonen	4

Uke 43

Ingenting ble jobbet med i denne uken på grunn av andre fag.

Uke 44

Ingenting ble jobbet med i denne uken på grunn av andre fag.

Uke 45

Dato	Oppgave	Tidsbruk (timer)
05/11/2020	Arbeid på obligatorisk oppgave 2	4

Uke 46

Dato	Oppgave	Tidsbruk (timer)
10/11/2020	Arbeid på obligatorisk oppgave 2	2

Uke 47

Dato	Oppgave	Tidsbruk (timer)
17/11/2020	Arbeid på obligatorisk oppgave 2	2

Uke 48

Dato	Oppgave	Tidsbruk (timer)
24/11/2020	Arbeid på obligatorisk oppgave 2	2

Samlet tidsbruk

Møte timer: 33

Vår 2021

Uke 01

Programmering

Ingen programmering ble utført denne uken

Møter

Dato	Tidsbruk (timer)
05/01/2021	2
08/01/2021	2

Uke 02

Programmering

Ingen programmering ble utført denne uken

Møter

Dato	Tidsbruk (timer)
12/01/2021	2
14/01/2021	2

Uke 03

Programmering

Ingen programmering ble utført denne uken.

Møter

Dato	Tidsbruk (timer)
19/01/2021	2
21/01/2021	2

Uke 04

Programmering

Oppgave	Tidsbruk (timer)
Lage frontend til film og serieinfo	18
Koble sammen session + sider	2

Møter

Dato	Tidsbruk (timer)
26/01/2021	4
28/01/2021	2

Uke 05

Programmering

Ingen programmering ble utført denne uken.

Møter

Dato	Tidsbruk (timer)
02/02/2021	2

Uke 06

Ingenting ble jobbet med i denne uken på grunn av andre fag.

Uke 07

Ingenting ble jobbet med i denne uken på grunn av andre fag.

Uke 08

Programmering

Ingen programmering ble utført denne uken.

Møter

Dato	Tidsbruk (timer)
23/02/2021	2
25/02/2021	2

Uke 09

Programmering

Ingen programmering ble utført denne uken

Møter

Dato	Tidsbruk (timer)
02/03/2021	1
04/03/2021	2

Uke 10

Programmering

Ingen programmering ble utført denne uken

Møter

Dato	Tidsbruk (timer)
11/03/2021	3

Uke 11

Programmering

Oppgave	Tidsbruk (timer)
Sjekke om søket i searchbaren har bilde	0.16

Møter

Dato	Tidsbruk (timer)
16/03/2021	2
18/03/2021	2

Uke 12

Programmering

Ingen programmering ble utført denne uken.

Møter

Dato	Tidsbruk (timer)
23/03/2021	2
25/03/2021	2

Uke 13

Programmering

Oppgave	Tidsbruk (timer)
på film og serier bruker den for langt tid til å laste actors	4
Kan kun trykke på bildet for link i livesearch	0.16
US02. Som en gjest vil jeg kunne se informasjon om serier	2
US01. Som en gjest vil jeg kunne se informasjon om filmer	2
Koble sammen serieinfo med backend	4
Koble sammen filminfo med backend	3

Møter

Dato	Tidsbruk (timer)
30/03/2021	1
01/04/2021	1

Uke 14

Programmering

Oppgave	Tidsbruk (Timer)
finn en god metode for route params til language	1
US06. Som en gjest vil jeg kunne se trailere til filmer eller linker til yt	3.5
Som en gjest vil jeg kunne se informasjon om cast/crew	7

Møter

Dato	Tidsbruk (timer)
05/04/2021	4
06/04/2021	1.5
08/04/2021	2

Uke 15

Programmering

Oppgave	Tidsbruk (Timer)
US20. Som et medlem vil jeg kunne omtale filmer/serier	4
US19. Som et medlem vil jeg kunne rate filmer/Serier	2.5
Film/serie side laster ikke hvis en actor ikke har imdb id.	2
Reviews er ferdig - Legge til på filminfo/serieinfo	4

Møter

Dato	Tidsbruk (timer)
13/04/2021	3
14/04/2021	2
17/04/2021	2

Uke 16

Programmering

Oppgave	Tidsbruk (Timer)
Serieinfo bug	0.75
Default actor bilde w/h er ikke riktig	0.75

Møter

Dato	Tidsbruk (timer)
22/04/2021	1.5
23/04/2021	1

Uke 17

Programmering

Oppgave	Tidsbruk (Timer)
Gjøre om actorinfo til person	0.25
Serieinfo side crasher	1
Executive Producer kan bli undefined	0.25
Bedre tilbakemelding til bruker ved review	1.5

Møter

Dato	Tidsbruk (timer)
27/04/2021	2

Uke 18

Ingenting ble jobbet med i denne uken på grunn av andre fag.

Uke 19

Programmering

Oppgave	Tidsbruk (Timer)
Dato trenger fiks	2.2

Møter

Dato	Tidsbruk (timer)
11/05/2021	1
14/05/2021	1.5

Uke 20

Programmering

Oppgave	Tidsbruk (Timer)
Internal server error på serie	1

Post review knapp må oversettes	1
Midlertidige bilder på film/serieinfo	2
US14. Som en gjest vil jeg kunne se ulike film/serie-lister lagd av andre brukere.	4.8

Møter

Dato	Tidsbruk (timer)
18/05/2021	1
20/05/2021	2.5
21/05/2021	2
22/05/2021	2
23/05/2021	4

Uke 21

Progammering

Oppgave	Tidsbruk (Timer)
Kommentert kode	6
Bruker kan legge til flere reviews for samme media	4

Møter

Dato	Tidsbruk (timer)
24/05/2021	4
25/05/2021	4
26/05/2021	4
27/05/2021	4
28/05/2021	2
29/05/2021	2

Ekstra

Dette er tidsbruk for aktiviteter som har gått over alle ukene.

Oppgave	Tidsbruk (Timer)
Fikse småbugs	5
Oversetting	2
Finpussing av front-end	10
Testing under programmering	5

Samlet tidsbruk

Programmert timer: 109.48

Møte timer: 88

Totalt: 197.48

Govert – 233565

Høst 2020

Vi hadde bare møter som var planlegging og arbeid med obligatoriske oppgaver i høst. Dette ble gjort sammen så denne delen er lik på alle sin timeliste. Tid brukt på studie/kursing av JavaScript er ikke regnet med her.

Uke 35

Dato	Oppgave	Tidsbruk (timer)
24/08/202	Brainstorming	2

Uke 36

Dato	Oppgave	Tidsbruk (timer)
03/09/2020	Sett opp arbeidsmiljø	2

Uke 37

Dato	Oppgave	Tidsbruk (timer)
07/09/2020	Designe prototype	1
08/09/2020	Videre design av prototype	1
10/09/2020	Planlegging av grafer/data/innhold	2

Uke 38

Dato	Oppgave	Tidsbruk (timer)
15/09/2020	Videre planlegging	2

Uke 39

Dato	Oppgave	Tidsbruk (timer)
22/09/2020	Planlegging av applikasjonens struktur	2

Uke 40

Dato	Oppgave	Tidsbruk (timer)
01/10/2020	Arbeid på obligatorisk oppgave 1	4

Uke 41

Dato	Oppgave	Tidsbruk (timer)
08/10/2020	Arbeid på obligatorisk oppgave 1	2

Uke 42

Dato	Oppgave	Tidsbruk (timer)
15/10/2020	Generell planlegging av applikasjonen	4

Uke 43

Ingenting ble jobbet med i denne uken på grunn av andre fag.

Uke 44

Ingenting ble jobbet med i denne uken på grunn av andre fag.

Uke 45

Dato	Oppgave	Tidsbruk (timer)
05/11/2020	Arbeid på obligatorisk oppgave 2	4

Uke 46

Dato	Oppgave	Tidsbruk (timer)
10/11/2020	Arbeid på obligatorisk oppgave 2	2

Uke 47

Dato	Oppgave	Tidsbruk (timer)
17/11/2020	Arbeid på obligatorisk oppgave 2	2

Uke 48

Dato	Oppgave	Tidsbruk (timer)
24/11/2020	Arbeid på obligatorisk oppgave 2	2

Samlet tidsbruk

Møte timer: 33

Vår 2021

Uke 01

Programmering

Ingen programmering ble utført denne uken

Møter

Dato	Tidsbruk (timer)
05/01/2021	2
08/01/2021	2

Uke 02

Programmering

Ingen programmering ble utført denne uken

Møter

Dato	Tidsbruk (timer)
12/01/2021	2
14/01/2021	2

Uke 03

Programmering

Oppgave	Tidsbruk (timer)
Login design	2
Satt opp server	5

Møter

Dato	Tidsbruk (timer)
19/01/2021	2
21/01/2021	2

Uke 04

Programmering

Oppgave	Tidsbruk (timer)
3 funksjoner: årstall fra dato, mnd fra dato og dagnr fra dato.	2
Beskrivelse på index	0.25
Koble sammen session + sider	2

Møter

Dato	Tidsbruk (timer)
26/01/2021	4
28/01/2021	2

Uke 05

Programmering

Ingen programmering ble utført denne uken.

Møter

Dato	Tidsbruk (timer)
02/02/2021	2

Uke 06

Ingenting ble jobbet med i denne uken på grunn av andre fag.

Uke 07

Ingenting ble jobbet med i denne uken på grunn av andre fag.

Uke 08

Programmering

Ingen programmering ble utført denne uken.

Møter

Dato	Tidsbruk (timer)
23/02/2021	2
25/02/2021	2

Uke 09

Møter

Dato	Tidsbruk (timer)
02/03/2021	1
04/03/2021	2

Uke 10

Programmering

Ingen programmering ble utført denne uken.

Møter

Dato	Tidsbruk (timer)
11/03/2021	3

Uke 11

Programmering

Oppgave	Tidsbruk (timer)
Sjekke om søket i searchbaren har bilde	0.16

Møter

Dato	Tidsbruk (timer)
16/03/2021	2
18/03/2021	2

Uke 12

Programmering

Ingen programmering ble utført denne uken.

Møter

Dato	Tidsbruk (timer)
23/03/2021	2
25/03/2021	2

Uke 13

Programmering

Ingen programmering ble utført denne uken.

Møter

Dato	Tidsbruk (timer)
30/03/2021	1
01/04/2021	1

Uke 14

Programmering

Oppgave	Tidsbruk (Timer)
finn en god metode for route params til language	1
Lage funksjon som formaterer store tall (M/K/B)	0.5

Møter

Dato	Tidsbruk (timer)
05/04/2021	4
06/04/2021	1.5
08/04/2021	2

Uke 15

Programmering

Oppgave	Tidsbruk (Timer)
US24. Som et medlem vil jeg kunne kontakte support	1.5

Møter

Dato	Tidsbruk (timer)
13/04/2021	3
14/04/2021	2
17/04/2021	2

Rapportskriving

Oppgave	Tidsbruk (Timer)
Testing	2

Uke 16

Programmering

Oppgave	Tidsbruk (Timer)
About page har blanda språk - Må oversettes	1.5

Møter

Dato	Tidsbruk (timer)
22/04/2021	1.5
23/04/2021	1

Uke 17

Programmering

Ingen programmering ble utført denne uken.

Møter

Dato	Tidsbruk (timer)
27/04/2021	2

Uke 18

Ingenting ble jobbet med i denne uken på grunn av andre fag.

Uke 19

Programmering

Oppgave	Tidsbruk (Timer)
US27. Som et medlem vil jeg kunne kommunisere med medlemmer og administratorer/support	3
About page	11

Møter

Dato	Tidsbruk (timer)
11/05/2021	1
14/05/2021	1.5

Uke 20

Programmering

Ingen programmering ble utført denne uken

Møter

Dato	Tidsbruk (timer)
18/05/2021	1
20/05/2021	2.5
21/05/2021	2
22/05/2021	2
23/05/2021	4

Rapportskriving

Oppgave	Tidsbruk (Timer)
Lagde ny test for fase 2	2

Uke 21

Programmering

Ingen programmering ble utført denne uken

Møter

Dato	Tidsbruk (timer)
24/05/2021	4
25/05/2021	4
26/05/2021	4
27/05/2021	4
28/05/2021	2
29/05/2021	2

Rapportskriving

Oppgave	Tidsbruk (Timer)
Fullførte testing	4

Ekstra

Dette er tidsbruk for aktiviteter som har gått over alle ukene.

Dato	Tidsbruk (timer)
Loggføring møter	3
Rapportskriving underveis	7
Gantt chart	11.25
Omskrive info på about page	1
Testing og forbedring av about pagen	5

Samlet tidsbruk

Programmert timer: 35.92

Møte timer (vår): 88

Møte timer (høst): 33

Annet: 31,5

Totalt: 188,42

Ørjan – 233530

Høst 2020

Vi hadde bare møter som var planlegging og arbeid med obligatoriske oppgaver i høst. Dette ble gjort sammen så denne delen er lik på alle sin timeliste.

Uke 35

Dato	Oppgave	Tidsbruk (timer)
24/08/202	Brainstorming	2

Uke 36

Dato	Oppgave	Tidsbruk (timer)
03/09/2020	Sett opp arbeidsmiljø	2

Uke 37

Dato	Oppgave	Tidsbruk (timer)
07/09/2020	Designe prototype	1
08/09/2020	Videre design av prototype	1
10/09/2020	Planlegging av grafer/data/innhold	2

Uke 38

Dato	Oppgave	Tidsbruk (timer)
15/09/2020	Videre planlegging	2

Uke 39

Dato	Oppgave	Tidsbruk (timer)
22/09/2020	Planlegging av applikasjonens struktur	2

Uke 40

Dato	Oppgave	Tidsbruk (timer)
01/10/2020	Arbeid på obligatorisk oppgave 1	4

Uke 41

Dato	Oppgave	Tidsbruk (timer)
08/10/2020	Arbeid på obligatorisk oppgave 1	2

Uke 42

Dato	Oppgave	Tidsbruk (timer)
15/10/2020	Generell planlegging av applikasjonen	4

Uke 43

Ingenting ble jobbet med i denne uken på grunn av andre fag.

Uke 44

Ingenting ble jobbet med i denne uken på grunn av andre fag.

Uke 45

Dato	Oppgave	Tidsbruk (timer)
05/11/2020	Arbeid på obligatorisk oppgave 2	4

Uke 46

Dato	Oppgave	Tidsbruk (timer)
10/11/2020	Arbeid på obligatorisk oppgave 2	2

Uke 47

Dato	Oppgave	Tidsbruk (timer)
17/11/2020	Arbeid på obligatorisk oppgave 2	2

Uke 48

Dato	Oppgave	Tidsbruk (timer)
24/11/2020	Arbeid på obligatorisk oppgave 2	2

Samlet tidsbruk

Møte timer: 33

Vår 2021

Uke 01

Programmering

Ingen programmering ble utført denne uken

Møter

Dato	Tidsbruk (timer)
05/01/2021	2
08/01/2021	2

Uke 02

Programmering

Ingen programmering ble utført denne uken

Møter

Dato	Tidsbruk (timer)
12/01/2021	2
14/01/2021	2

Uke 03

Programmering

Oppgave	Tidsbruk (timer)
email og passord validation	1
Lager index side	2
navbar	1
Satt opp server	5

Møter

Dato	Tidsbruk (timer)
19/01/2021	2
21/01/2021	2

Uke 04

Programmering

Oppgave	Tidsbruk (timer)
glemt passord funksjon	12
US10. Som en gjest vil jeg kunne se en kalender over kommende filmpremierer	1
Koble sammen session + sider	2

Møter

Dato	Tidsbruk (timer)
26/01/2021	4
28/01/2021	2

Uke 05

Programmering

Ingen programmering ble utført denne uken.

Møter

Dato	Tidsbruk (timer)
02/02/2021	2

Uke 06

Ingenting ble jobbet med i denne uken på grunn av andre fag.

Uke 07

Ingenting ble jobbet med i denne uken på grunn av andre fag.

Uke 08

Programmering

Ingen programmering ble utført denne uken.

Møter

Dato	Tidsbruk (timer)
23/02/2021	2
25/02/2021	2

Uke 09

Programmering

Ingen programmering ble utført denne uken

Møter

Dato	Tidsbruk (timer)
02/03/2021	1
04/03/2021	2

Uke 10

Programmering

Oppgave	Tidsbruk (timer)
Load more knapp til upcoming	2.5

Møter

Dato	Tidsbruk (timer)
11/03/2021	3

Uke 11

Programmering

Oppgave	Tidsbruk (timer)
US38. Som en bruker av nettsiden vil jeg kunne søke etter film/serie	2
Upcoming movies, tv-shows og movies.	6
Sjekke om søket i searchbaren har bilde	0.16
Responsive navbar	1.5
Feilmelding login/signup/logout	0.5
US18. Som et medlem vil jeg kunne logge av	0.08

Møter

Dato	Tidsbruk (timer)
16/03/2021	2
18/03/2021	2

Uke 12

Programmering

Ingen programmering ble utført denne uken.

Møter

Dato	Tidsbruk (timer)
23/03/2021	2
25/03/2021	2

Uke 13

Programmering

Oppgave	Tidsbruk (timer)
US02. Som en gjest vil jeg kunne se informasjon om serier	2
US01. Som en gjest vil jeg kunne se informasjon om filmer	2

Møter

Dato	Tidsbruk (timer)
30/03/2021	1
01/04/2021	1

Uke 14

Programmering

Oppgave	Tidsbruk (Timer)
Flerspråklig	20
Finn en god metode for route params til language	1
US06. Som en gjest vil jeg kunne se trailere til filmer eller linker til youtube	0.5

Møter

Dato	Tidsbruk (timer)
05/04/2021	4
06/04/2021	1.5
08/04/2021	2

Uke 15

Programmering

Oppgave	Tidsbruk (Timer)
Legg til i watched listeknapp	0.5
Filter	5.5
Link til film side i upcoming	0.5
Express middleware for "mal" av render og language middleware istedenfor router.all	0.25
legge til flere språk via adminpanel	8
filter i watched/favorite (movies/tvshows/all)	0.25
US24. Som et medlem vil jeg kunne kontakte support	3

Møter

Dato	Tidsbruk (timer)
13/04/2021	3
14/04/2021	2
17/04/2021	2

Uke 16

Programmering

Oppgave	Tidsbruk (Timer)
US08. Som et medlem vil jeg kunne se relaterte filmer/serier til de jeg ser på	0.5
Active side i nav baren er alltid home	0.016
Contact burde ikke ligge i explore	0.016
Linker i footer er ikke satt	0.08
Passord krav på signup siden burde bli vist	0.16
Ingen error melding fra forgotten password	1
US30. Som en administrator vil jeg kunne endre brukere sine reviews	1.5
US32. Som en administrator vil jeg kunne godkjenne brukere sine reviews	1.5
US33. Som en administrator vil jeg kunne avslå brukere sine reviews	1.5
US31. Som en administrator vil jeg kunne slette brukere sine reviews	0.5
US34. Som en administrator vil jeg kunne deaktivere brukere	0.5

Møter

Dato	Tidsbruk (timer)
22/04/2021	1.5
23/04/2021	1

Uke 17

Programmering

Oppgave	Tidsbruk (Timer)
Serier i watched listen til brukeren linker til filminfo ikke serieinfo	0.25
Serier i favorited listen til brukeren linker til filminfo ikke serieinfo.	0.25
US07. Som en gjest vil jeg kunne få tilfeldig forslag innenfor ulike sjangrer	1
US09. Som en gjest vil jeg kunne dele filmen/serien via sosiale medier	0.25
Link til media i watchlist	0.25

Møter

Dato	Tidsbruk (timer)
27/04/2021	2

Uke 18

Ingenting ble jobbet med i denne uken på grunn av andre fag.

Uke 19

Programmering

Oppgave	Tidsbruk (Timer)
US27. Som et medlem vil jeg kunne kommunisere med medlemmer og administratorer/support	2

Møter

Dato	Tidsbruk (timer)
11/05/2021	1
14/05/2021	1.5

Uke 20

Programmering

Oppgave	Tidsbruk (Timer)
Livesearch funker bare på index	0.16
Dashboard	5.5
Scroll bar blir ikke fjerna når det ikke er noe resultat på film/serie search	0.16
US25. Som et medlem vil jeg kunne lage lister over filmer og dele listen i ulike medium	2.5
Favorites knappen gjør ingenting	1
Tilbakemelding på sending av contact form på about	1
Legg til i liste må bli lukket etter success	0.16

Møter

Dato	Tidsbruk (timer)
18/05/2021	1
20/05/2021	2.5
21/05/2021	2
22/05/2021	2
23/05/2021	4

Uke 21

Progammering

Oppgave	Tidsbruk (Timer)
Kommentert kode	4
US23. Som et medlem vil jeg kunne se personlige statistikker (Grafer)	1.5
US03. Som en gjest vil jeg kunne se en "overview" slide	0.75
Error bokser reworked	0.25
Min liste mangler bilde om noe er i listen	0.25
Linker går til index	0.25
Bug når "cannot read substring of null"	0.25

Møter

Dato	Tidsbruk (timer)
24/05/2021	4
25/05/2021	4
26/05/2021	4
27/05/2021	4
28/05/2021	2
29/05/2021	2

Ekstra

Dette er tidsbruk for aktiviteter som har gått over alle ukene.

Oppgave	Tidsbruk (Timer)
Oversetting	11
Små bugfiksing	10
Designing (animasjoner, rework og forbedringer)	12
Testing og debugging	12

Samlet tidsbruk

Programmert timer: 154.12

Møte timer: 88

Totalt: 242.12

Vedlegg E

Sivert – 233518

Favoritt system

Jeg implementerte favoritt systemet. Systemet skulle gjøre det mulig å legge til og fjerne serier/filmer fra en favoritt liste.

«addFavourite» funksjonen legger til en film som favoritt for en bruker. Det eneste du skal trenge å gi denne funksjonen er en film ID og en bruker ID. Den gjør deretter nødvendige sjekker og returnerer et svar om operasjonen var vellykket eller ikke. «addFavourite» for serie er ganske lik, bare at vi jobber med serie databasen istedenfor.

```
/**
 * Legger til film i database
 * @param {Number} movieId
 * @param {String} userId
 * @returns ValidationHandler
 * @author Sivert - 233518
 */
async function addFavourite(movieId, userId) {
    logger.log({level: 'debug', message: `Adding movie with id ${movieId} to ${userId}'s favourite list`});

    //Skaffer bruker
    const user = await userHandler.getUserFromId(userId);
    if(!user.status) return user;

    //Sjekker om bruker allerede har filmen som favoritt
    const isFavorited = await checkIfFavorited(movieId, user.information);
    if(isFavorited.status) return isFavorited;

    //Prøver å oppdatere bruker
    const updateUserResult = await userHandler.updateUser(user.information, {$push: {movieFavourites: movieId}});
    if(!updateUserResult.status) return updateUserResult;
```

```
    //Sjekker om film er lagret i database
    const isSaved = await movieHandler.checkIfSaved(movieId);
    if(isSaved.status) return isSaved;

    //Skaffer film informasjon
    const movieInfo = await tmdb.data.getMovieInfoByID(movieId);
    if(!movieInfo) {
        logger.log('error', `Could not retrieve information for movie with id
${movieId}`)
        return new ValidationHandler(false, 'Could not retrieve movie informat
ion');
    }

    //Legger til film i database
    const addToDatabaseResult = await movieHandler.addToDatabase(movieInfo);
    if(!addToDatabaseResult.status) return addToDatabaseResult;

    //Suksess
    logger.log({level: 'debug', message: `Successfully added movie with id ${m
ovieId} to ${userId}'s favourite list`});
    return new ValidationHandler(true, `Favourite successfully added`);
}
```

Liste system

Jeg implementerte liste systemet. Systemet skulle gjøre det mulig å opprette liste, slette liste, legge til film/serie i liste og fjerne film/serie fra liste.

«createList» oppretter en liste og tar inn en bruker og navn. Den oppretter objektet og blir lagret av funksjonen «saveToDatabase». Deretter blir brukeren oppdatert i databasen og det blir returnert et svar om operasjonen var vellykket eller ikke.

```
/**
 * Lager en liste for bruker
 * @param {Object} user
 * @param {String} name
 * @returns ValidationHandler
 * @author Sivert - 233518
 */
exports.createList = async function(user, name) {
    logger.log({level: 'debug', message: `Creating list for user with id ${use
r._id}`});
```

```
    //Lagrer liste til database
    const resultList = await saveToDatabase({
        userId: user._id,
        name: name
    });
    if(!resultList.status) return resultList;

    //Oppdaterer bruker
    const userResult = await userHandler.updateUser(user, {$push: {lists: resu
ltList.information._id.toString()}})
    if(!userResult.status) return userResult;

    //Suksess
    logger.log({level: 'debug', message: `Successfully created list with id ${
resultList.information._id} for user with id ${user._id}`});
    return new ValidationHandler(true, 'Successfully created list');
}
```

«saveToDatabase» funksjonen lagrer til databasen. Den blir kjørt av funksjonen «createList».

Den oppretter en ny model med indataen den får (Et objekt som har samme oppsett som listSchema). Deretter blir dette lagret i databasen.

```
/**
 * Lagrer til databasen
 * @param {Object} listObject
 * @returns ValidationHandler
 * @author Sivert - 233518
 */
function saveToDatabase(listObject) {
    const list = new ListModel(listObject);
    return list.save().then((doc, err) => {
        if(err){
            logger.log({level: 'error', message: `There was an error adding th
e list to the database! ${err}`});
            return new ValidationHandler(false, 'Could not add list to the dat
abase!');
        }
        logger.log({level: 'info', message: `List with id ${doc._id} was saved
 to the database`});
        return new ValidationHandler(true, doc);
    })
}
```

Review system

Jeg implementerte review systemet. Systemet skulle gjøre det mulig å opprette, endre, slette, akseptere og avslå anmeldelse.

Jeg har holdt meg til funksjonell programmering, men for å teste ut klasser så tenkte det ville gjøre jobben lettere her. Jeg ville lage en universal kode som fungerte for både film og serie. Det ble da opprettet 3 klasser. Hovedklassen ble Review og to underklasser ReviewTv og ReviewMovie. Det ga meg muligheten til å utvikle koden slik at den fungerte for både film og serie.

```
/**
 * Hovedklassen for reviews
 * @author Sivert - 233518
 */
class Review {
    constructor(userId, text, stars) {
        this.userId = userId;
        this.stars = stars;
        this.text = text;
    }
}

/**
 * Klasse for review av tv
 * @author Sivert - 233518
 */
class ReviewTv extends Review {
    constructor(userId, tvId, text, stars) {
        super(userId, text, stars)
        this.tvId = tvId;
    }
}

/**
 * Klasse for review av movie
 * @author Sivert - 233518
 */
class ReviewMovie extends Review {
    constructor(userId, movieId, text, stars) {
        super(userId, text, stars)
        this.movieId = movieId;
    }
}
```

«makeReview» funksjonen sjekker inndataen og om brukeren allerede har en eksisterende review (Denne sjekke blir også gjort på klient siden, men den kan hackere gå rundt så det blir også gjort i backend). Deretter lagrer den informasjon til databasen og returnerer om operasjonen var vellykket eller ikke.

```
/**
 * Lager review og lagrer i databasen
 * @param {ReviewTv|ReviewMovie} review En av underklassene til Review
 * @author Sivert - 233518, Sigve - 233511
 */
async function makeReview(review) {
    logger.log({level:'debug', message: `Creating new review`});

    //Sjekker tekst
    if (review.text === "")
        return new ValidationHandler(false, 'Your review needs some content!')
;

    //Sjekker stars
    if (review.stars == undefined)
        return new ValidationHandler(false, 'You need to select atleast 1 star
 to post a review!');

    //Sjekker om approved review allerede eksisterer
    if(await (await reviewGetter.getApprovedReviewUser(review.userId, review.m
ovieId == null ? review.tvId : review.movieId, review.movieId == null ? 'tv' :
 'movie')).status)
        return new ValidationHandler(false, 'User already made review for this
 media');

    //Sjekker om pending review allerede eksisterer
    if(await (await reviewGetter.getPendingReviewUser(review.userId, review.mo
vieId == null ? review.tvId : review.movieId, review.movieId == null ? 'tv' :
'movie')).status)
        return new ValidationHandler(false, 'User already have a pending revie
w for this media');

    //legger til i database
    const databaseReturn = await addToDatabase(review);
    if(!databaseReturn.status) return databaseReturn;

    //Skaffer bruker
    const userResult = await userHandler.getUserFromId(review.userId);
    if(!userResult.status) return userResult;
```

```
    //Suksess
    logger.log({level:'debug', message: `Review was successfully created for u
ser ${review.userId}`});
    return new ValidationHandler(true, 'Review was successfully created for us
er');
}
```

Watched system

Jeg implementerte watched systemet. Det er en sett liste for brukeren. Systemet skulle gjøre det mulig å legge til og fjerne serier/filmer fra en sett liste. Hvorfor er ikke dette samme kode som favoritt? Den gjør det samme, men favoritt systemet er delt opp i film/serie. Jeg ville lage en universell kode som på review systemet, men ville holde meg unna klasser her. Det endte opp med at jeg utviklet koden med helt vanlig switch statements. Jeg var mer fornøyd med denne måten å utvikle koden på, og det var mye mindre jobb med testing her.

Funksjone som legger til i sett listen er ganske lik som favoritt systemet, bare med ekstra parameter. Koden gjør nødvendige sjekker, oppdaterer databasen og returnerer om operasjonen var vellykket eller ikke. Det er i funksjonen som oppdaterer databasen der switch statementsa kommer inn. Den finner ut av hvilken database som skal oppdateres og hvilken felt hos brukeren.

```
/**
 * Oppdaterer databasen
 * @param {Object} user
 * @param {Number} mediaId
 * @param {'movie'|'tv'} mediaType
 * @returns ValidationHandler
 * @author Sivert - 233518
 */
function updateDatabase(user, mediaId, mediaType) {
    switch(mediaType) {
        case 'movie':
            return userHandler.updateUser(user, {$push: {moviesWatched: mediaI
d}});
        case 'tv':
            return userHandler.updateUser(user, {$push: {tvsWatched: mediaId}}
);
    }
}
```

Logging

Logging av et av ekstra kravene for prosjektet. Jeg implementerte logging systemet ved hjelp av npm pakken «winston». Denne pakken gjør det lett å konfigurere din egen logger. «Const logger» oppretter en ny Winston objekt konfigurert med 3 transports(filer) og forskjellige nivåer (level). Nivåene er da info, error og warn. Filene er error, warn og combined.log. Deretter sier vi at dersom environment er production, så legger vi til en tom transports.console som sier at du skal logge alt som er i objektet. Dersom environmentet er i debug mode, så legger vi til en ny transports.file som heter debug, og en transports.console som sier at vi skal logge alle nivåene fra debug og under til konsollen.

```
const winston = require('winston');

/**
 * Lager customFormat
 * @author Sivert - 233518
 */
const customFormat = winston.format.printf(({level, message, timestamp}) => {
    return `[${level.toUpperCase()}][${timestamp}]: ${message}`
})

/**
 * Lager en ny logger med vår egne instillinger
 * @author Sivert - 233518
 */
const logger = winston.createLogger({
    level: 'info',
    format: winston.format.combine(
        winston.format.timestamp(),
        customFormat
    ),
    transports: [
        new winston.transports.File({ filename: './logging/logs/error.log', le
vel: 'error' }),
        new winston.transports.File({ filename: './logging/logs/warn.log', lev
el: 'warn' }),
        new winston.transports.File({ filename: './logging/logs/combined.log'
})]
});

//Logger om production environment
if (process.env.NODE_ENV == 'production') {
    logger.add(new winston.transports.Console({}));
}
```

```
//Logger om debug environment
if(process.env.NODE_ENV == 'debug') {
    logger.add(new winston.transports.File({ filename: './logging/logs/debug.l
og', level: 'debug'}))
    logger.add(new winston.transports.Console({level: 'debug'}))
}

module.exports = logger;
```

API

«hentTmdbInformasjon» funksjonen er ansvarlig for å hente statisk informasjon. Dette gjelder da ting som trending filmer/serier. Denne informasjon blir bare endret en gang i uken, og det er veldig lite effektivt å hente denne informasjonen hver gang noen går inn på forsiden. Istedenfor så blir denne informasjonen lagret i minne istedenfor. Den henter også inn kommende filmer/serier. Jeg implementerte denne funksjonen så effektivt som mulig, ved å ta i bruk Promise.all istedenfor å vente på hver enkelt forespørsel til API'et. Dette betyr at alle requestene blir prosessert på samme tid. Denne funksjonen kjører hver gang server starter og blir oppdatert hver 24t om serveren står på.

```
    /**
     * Henter informasjon fra The Movie Database API'en
     * Legger informasjonen inn i variabelen tmdbInformasjonKlar
     * @author Sivert - 233518
     */
    hentTmdbInformasjon: async function () {
        try {
            logger.log({level: 'info', message: 'Starting collection of tmdb i
nformation...'});
            let currentDate = new Date();
            let currentDateFormated = `${currentDate.getFullYear()}-
${(currentDate.getMonth()+1).toString().padStart(2, "0")}-
${currentDate.getDate().toString().padStart(2,"0")}`
            const antallPages = 10; //Antall sider som skal bli hentet
            let tmdbInformasjon = {
                discoverMoviesUpcoming: [],
                discoverMoviesPopular: [],
                discoverTvshowsUpcoming: [],
                discoverTvshowsPopular: [],
            };
```

```
            const [discoverMoviesUpcoming, discoverMoviesPopular, discoverTvsh
owsUpcoming, discoverTvshowsPopular] = await Promise.all([
                Promise.all(getDiscoverMovie(antallPages, `primary_release_dat
e.gte=${currentDateFormated}`)
                    .map(promise => promise
                    .then(res => res.results))),
                Promise.all(getDiscoverMovie(antallPages, `primary_release_dat
e.lte=${currentDateFormated}`)
                    .map(promise => promise
                    .then(res => res.results))),
                Promise.all(getDiscoverTvshow(antallPages, `first_air_date.gte
=${currentDateFormated}`)
                    .map(promise => promise
                    .then(res => res.results))),
                Promise.all(getDiscoverTvshow(antallPages, `first_air_date.gte
.lte=${currentDateFormated}`)
                    .map(promise => promise
                    .then(res => res.results))),
            ])

            tmdbInformasjon.discoverMoviesUpcoming = discoverMoviesUpcoming.fl
at();
            tmdbInformasjon.discoverMoviesPopular = discoverMoviesPopular.flat
();
            tmdbInformasjon.discoverTvshowsUpcoming = discoverTvshowsUpcoming.
flat();
            tmdbInformasjon.discoverTvshowsPopular = discoverTvshowsPopular.fl
at();

            //Sorterer movies upcoming etter dato
            tmdbInformasjon.discoverMoviesUpcoming.sort((a, b) => {
                return new Date(a.release_date).getTime() - new Date(b.release
_date).getTime();
            })
            //Sorterer tvshows upcoming etter dato
            tmdbInformasjon.discoverTvshowsUpcoming.sort((a, b)  => {
                return new Date(a.first_air_date).getTime() - new Date(b.first
_air_date).getTime();
            })
            tmdbInformasjonKlar = tmdbInformasjon;
            logger.log({level: 'info',message: 'All information is sucessfully
 collected!'});
            return new ValidationHandler(true, 'All information is sucessfully
 collected!');
        } catch(err) {
            logger.log({level: 'error' ,message: `Something unexpected happen
while collecting tmdb information! Error: ${err}`});
```

```
            return new ValidationHandler(false, 'Couldnt get start information
 from API');
        }
    },
```

"getMovieResults" søkter etter en film fra tittel og språkkode. Her ser du eksempelet på en url som brukes til å sende forespørsel til API ' et. Når jeg implementerte denne funksjonen, så ville jeg lage noe som var modulær, ettersom vi trengte mange av disse. Jeg lagde da også funksjonene makeLog(Som bare logger forespørselene) og resultHandler. Funksjonen «resultHandler» håndterer da svaret fra forespørselen. Denne blir returnert som en Promise med en resolve og reject.

```
/**
 * Søker etter film fra tittel
 * @param {String} movieTitle tittel på filmen
 * @param {String} languageCode språkkoden
 * @returns JSON med film info
 * @author Sivert - 233518
 */
Tmdb.prototype.getMovieResults = function getMovieResults(movieTitle, language
Code) {
    var url = `https://api.themoviedb.org/3/search/movie?api_key=${this.token}
&query=${movieTitle.replace(/ /g, "+")}&language=${languageCode}`;
    return fetch(url).then(res => {
        makeLog(res, url);
        return resultHandler(res);
    })
}
/**
 * Håndterer svaret på requesten
 * @param {Object} res Respons fra API
 * @returns Promise
 * @author Sivert - 233518
 */
function resultHandler(res) {
    return new Promise((resolve, reject) => {
        if(res.ok) {
            resolve(res.json());
        } else {
            reject(false)
        }
    })
}
```

Livesearch

Vi ville gjerne ha en livesearch på nettsiden. Altså at et resultat kommer opp når bruker søker etter en film i sanntid. Jeg implementerte da funksjonen searchForMedia. Den tar inn en tittel og språkkode. Den vil alltid ha maks 10 resultat, 5 filmer og 5 serier. Den prøver å søke etter språket som brukeren har valgt, men har en fallback på engelsk dersom infoen ikke er oversatt. Den looper så imellom hvert resultat og pusher i return arrayet.

```
/**
 * Søker etter media ved hjelp av tittel
 * Sender tilbake 5 av hver
 * @param {String} title
 * @returns ValidationHandler
 * @author Sivert - 233518
 */
async function searchForMedia(title, languageCode) {
    let tenResult = [];
    const resultMovie = await tmdbHandler.data.getMovieInfo(title, languageCod
e); //Henter info fra api
    const resultTv = await tmdbHandler.data.getSerieInfo(title, languageCode);
 //Henter info fra api

    if(resultMovie.length == 0 && resultTv.length == 0) return new ValidationH
andler(false, 'No results');

    let counter = 0;
    for(let movie of resultMovie.results) {
        if(counter == 5) break;
        if(movie.overview == "" || !movie.overview) {
            movie = await tmdbHandler.data.getMovieInfoByID(movie.id, 'en');
        }
        tenResult.push({
            id: movie.id,
            poster_path: movie.poster_path,
            title: movie.title,
            overview: movie.overview,
            type: 'movie'
        });
        counter++;
    }
    counter = 0;
    for(let tv of resultTv.results) {
        if(counter == 5) break;
        if(tv.overview == "" || !tv.overview) {
            tv = await tmdbHandler.data.getSerieInfoByID(tv.id, 'en');
        }
```

```
        tenResult.push({
            id: tv.id,
            poster_path: tv.poster_path,
            title: tv.name,
            overview: tv.overview,
            type: 'tv'
        });
        counter++;
    }
    return new ValidationHandler(true, tenResult); //Returnerer resultat
}
```

Sigve – 233511

Det er for det meste meg som har hatt hovedansvaret for film/serie/person informasjons sidene i Filmatory prosjektet.

tvController.js

Under ser vi ett utdrag fra tvController.js som er en get for seriesiden. Denne henter serieinformasjonen, anmeldelser til serien også videre.

Det de første linjene i koden handler for det meste om å initialere variabler som kan hende at ikke blir initialisert. Dette kan for eksempel skje med isReviewed, hasPendingReview eller hasAnyReview hvis en bruker ikke er logget inn, på grunn av at de trenger variabler fra brukeren som har logget inn, hvis noen er logget inn. Det er også inkludert logging, som forteller deg I konsollvinduet hva siden holder på med akkurat nå. Til slutt har vi variable som blir hentet direkte fra API fra TMDB og variablene hentet fra Filmatory's database uavhengig om en bruker har logget inn eller ikke. «castinfolet» henter for eksempel alle personene som har jobbet på serien ved hjelp av ID'en til filmen og språkkoden. Til slutt har vi let serie, som tar de fleste variablene som tilhører informasjonen om serien og de som har jobbet med den, og pakker det inn i et objekt.

```
exports.tv_get_info = async function(req, res) {
    let isReviewed = false;
    let hasPendingReview = false;
    let hasAnyReview = false;
    logger.log({level: 'debug', message: 'Getting castinfo..'});
    let castinfolet = await tmdb.data.getSerieCastByID(req.params.id, req.rend
erObject.urlPath);
    logger.log({level: 'debug', message: 'Getting reviews..'});
    let reviews = await reviewGetter.getApprovedReviews(req.params.id, 'tv');
```

```
    let pendingReviews = await reviewGetter.getPendingReviews(req.params.id, '
tv');
    let isTvFav = new ValidationHandler(false, "");
    let isTvWatched = new ValidationHandler(false, "");
    logger.log({level: 'debug', message: 'Getting serieinfo, tailers, lists of
 persons & making object..'});
    let userMediaList = [];
    if(req.renderObject.user){
        for(let i = 0; i < req.renderObject.user.lists.length; i++){
            let userLists = await listGetter.getListFromId(req.renderObject.us
er.lists[i]);
            let tempObj = {
                id: userLists.information.id,
                name: userLists.information.name
            }
            userMediaList.push(tempObj);
        }
    }
    let serie = {
        serieinfo: res.locals.tvInfo,
        shortBio: await hjelpeMetoder.data.maxText(res.locals.tvInfo.overview,
 500),
        castinfo: castinfolet,
        videos: await tmdb.data.getSerieVideosByID(req.params.id, req.renderOb
ject.urlPath),
        listOfPersons: await Promise.all(getPersons(castinfolet.cast, req.rend
erObject.urlPath)),
        reviews: dateFixer(reviews.information)
    }
```

Etter serie objektet har blitt satt sammen vil den gå igjennom en serie sjekker og få litt mer informasjon lagt til. For eksempel i denne koden under så henter den mer informasjon om personer utifra ID som tilhører en person som har vært med på å lage serien. Denne informasjonen blir så lagt til i serie objektet.

```
    logger.log({level: 'debug', message: 'Getting list of persons'});
    for(const item of serie.castinfo.cast){
        serie.listOfPersons.push(await tmdb.data.getPersonByID(item.id, req.re
nderObject.urlPath));
    }
```

Disse variablene blir brukt for å finne ut om brukeren har skrevet en anmeldelse av serien, og om de venter på bekreftelse eller om de er godkjent. I Pug koden kommer dette til nytte da personen som er logget inn får informasjon om dette på anmeldelse feltet på serieinformasjons siden.

```
logger.log({level: 'debug', message: 'Checking movie is reviewed..'});
    if(req.renderObject.session){
        isReviewed = checkIfReviewed(await (req.session.userId), reviews.infor
mation);
        hasPendingReview = checkIfPendingReview(await (req.session.userId), pe
ndingReviews.information);
        hasAnyReview = checkIfAnyReview(isReviewed, hasPendingReview);
    }
```

Denne metoden tar inn bruker-id'en til personen som er logget inn, og anmeldelsene som tilhører serie siden, og sjekker om brukeren sin id samsvarer med id'ene til anmeldelsene til siden. Hvis den gjør dette så returnerer den «true».

```
/**
 * Metode for å sjekke om personen som er logget inn har en godkjent anmeldels
e av mediet.
 * @param {Number} thisUserId
 * @param {Object} thisReviews
 * @returns {Boolean} True eller False, avhengig på om det er en godkjent anme
ldelse eller ikke.
 * @author Sigve E. Eliassen - 233511.
 */
function checkIfReviewed(thisUserId, thisReviews) {
    for (const item of thisReviews) {
        if(item.userId == thisUserId) {
            return true;
        }
    }
    return false;
}
```

Denne metoden tar inn og sjekker pending reviews og godkjente reviews til brukeren, og hvis noen av de er «true» vil den returnere true. Under i Pug koden skal jeg vise hvorfor denne er nyttig.

```
/**
 * Sjekker om bruker har en ubehandlet, eller godkjent anmeldelse av mediet.
 * @param {Object} reviews1
 * @param {Object} reviews2
 * @returns {Boolean} True = har review, False = har ikke review.
 * @author Sigve E. Eliassen - 233511.
 */
function checkIfAnyReview(reviews1, reviews2) {
    if (reviews1 == true || reviews2 == true) {
        return true;
    }
    else return false;
}
```

Enkelt sagt så sørger denne koden for at all data som blir skaffet eller behandlet i javascript filen sendt over for å kunne bli brukt eller vist frem i serieinfo.pug filen.

```
logger.log({level: 'debug', message: 'Rendering page..'});
    req.renderObject.serie = serie;
    if (req.renderObject.user != undefined){
        req.renderObject.userId = JSON.stringify(req.renderObject.user._id)
    }
    req.renderObject.isLoggedIn = req.renderObject.session;
    req.renderObject.userMediaList = userMediaList;
    req.renderObject.tvId = JSON.stringify(req.params.id);
    req.renderObject.isTvFav = JSON.stringify(isTvFav.status);
    req.renderObject.isTvWatched = JSON.stringify(isTvWatched.status);
    req.renderObject.isReviewed = isReviewed;
    req.renderObject.hasPendingReview = hasPendingReview;
    req.renderObject.hasAnyReview = hasAnyReview;
    res.render("mediainfo/serieinfo", req.renderObject);
}
```

serieinfo.pug

Her har vi et godt eksempel over der vi har data fra API fra tvController.js som har blitt «sendt» til serieinfo.pug filen. Vi kan for eksempel se her at vi har skaffet informasjon fra API som forteller noe om når serien først ble sendt, og når den siste episoden ble sendt. Man setter da dette for eksempel i en «h3» overskrift som blir vist fram på nettsiden.

```
- if(serie.serieinfo.first_air_date != undefined || serie.serieinfo.last_air_d
ate != undefined || serie.serieinfo.first_air_date != null || serie.serieinfo.
last_air_date != null)
                h3= `${serie.serieinfo.first_air_date} - ${serie.serieinfo.las
t_air_date} ${genres}`
              .close
                h3(style='float: center;')=`${serie.serieinfo.number_of_season
s}`
                -  if (serie.serieinfo.number_of_seasons == 1)
                  h3(style='float: center;') #{__("TVINFO_SEASON")}
                - else
                  h3(style='float: center;') #{__("TVINFO_SEASONS")}
                br
              h1= `${serie.serieinfo.vote_average*10}%`
              h2 #{__("MOVIEINFO_OVERVIEW")}
```

Her kan vi se pug-koden til anmeldelse-posteren på serieinfo-siden. Først så sjekker den om «session» som gir true eller false om det er en bruker logget inn. Er det ikke en bruker logget inn vil den hoppe rett til «p #{__("REVIEW_LOG_IN_TO_POST")}» i koden. Denne paragrafen er skrevet på denne måten fordi den er oversatt til mange forskjellige språk, så den er på en måte bare en peker på en paragraf den kan si på flere språk, som er lagret i JSON-filer.

Hvis en bruker er logget inn vil den så sjekke om brukeren har allerede skrevet en anmeldelse, og hvis en bruker har det vil den sjekke om den er godkjent eller til vurdering av administratorer, den vil så vise frem et svar avhengig av dette.

Hvis brukeren aldri har skrevet en anmeldelse eller ikke fått godkjent anmeldelsen sin vil brukeren da kunne velge hvor mange stjerner brukeren skal gi serien og skrive hva brukeren mener om serien.

```
            - if(session)
              - if(!hasAnyReview)
                #serieinfo-review-form
                  .uk-text-left(class='uk-child-width-expand@s' uk-grid='')
                    div
                      .stars(id="stars")
                        input#five(type='radio' name='rate' value='5')
                        label(for='five')
                        input#four(type='radio' name='rate' value='4')
                        label(for='four')
                        input#three(type='radio' name='rate' value='3')
                        label(for='three')
                        input#two(type='radio' name='rate' value='2')
                        label(for='two')
                        input#one(type='radio' name='rate' value='1')
                        label(for='one')
                        span.result
                      textarea.uk-textarea.uk-form-large(id="reviewText")
                      a.uk-button.uk-button-primary.uk-button-
small(id="reviewPoster")
                        p #{__("REVIEW_POST_COMMENT")}
                    div
                    div
                #serieinfo-review-result
              - else if (isReviewed)
                p #{__("REVIEW_TV_ALREADY_REVIEWED")}
              - else if (hasPendingReview)
                p #{__("REVIEW_PENDING")}
            - else
              p #{__("REVIEW_LOG_IN_TO_POST")}
```

Originalt hadde vi et problem med at oversikten over filmen, serien eller personen, eller teksten som forklarer den eller personen kunne bli temmelig lang, som gjorde at siden ikke så veldig fin ut lenger. Dette ble løst med at det ble lagt til en «les mer» knapp på slutten av teksten hvis den oversteg en begrensning på 500 ord. Hvis brukeren trykket på dette så ville en «modal» dukke opp (en type tekstboks) i forgrunnen på siden som gir den fulle beskrivelsen. Her ser vi også at UiKit blir benyttet som er et rammeverk for å kunne gjøre ting som f.eks «modal» på en rask og enkel måte.

```
h2 #{__("MOVIEINFO_OVERVIEW")}
p= `${serie.shortBio}`
  - if(serie.serieinfo.overview.length > 500)
    a.uk-text-bold(href='#modal-fullbio' uk-
toggle='')  #{__("MEDIAINFO_MODAL_READ_MORE")}
#modal-fullbio.uk-flex-top(uk-modal='')
  .uk-modal-dialog.uk-modal-body.uk-margin-auto-vertical(class='uk-width-1-
2@s')
    button.uk-modal-close-default(type='button' uk-close='')
    p= `${serie.serieinfo.overview}`
```

personController.js

Dette er controlleren til person informasjons siden. Den henter altså data fra API, setter det sammen og sender det over slik at PUG filen får informasjonen tilgjengelig.

Denne controlleren henter all slags informasjon som tilhører personen ved hjelp av ID. Dette kan være informasjon som blir brukt til å vise navn, biografi, fødselssted, filmer eller serier personen har vært med i, eller med på å skape og lenker til f.eks Instagram, facebook eller imdb.

```
/**
 * GET for person side.
 * @param {Object} req En forespørsel fra klienten
 * @param {Object} res En respons fra server
 * @author Sigve E. ELiassen - 233511
 */
exports.personInfo_get = async function (req, res) {
    logger.log({level: 'debug' ,message:'Finding session for user'})
    const personId = req.url.slice(1);
    let personInfo = await tmdb.data.getPersonByID(personId, req.renderObject.
urlPath);
    //Lager person objekt
    let person = {
      personinfo: personInfo,
      links: await tmdb.data.getPersonLinksByID(personId, req.renderObject.url
Path),
      shortBio: await hjelpeMetoder.data.maxText(personInfo.biography,500)
    }
    if(person.personinfo.biography == "" || !person.personinfo.biography) {
      person.personinfo = await tmdb.data.getPersonByID(personId, 'en')
      person.shortBio = await hjelpeMetoder.data.maxText(person.personinfo.bio
graphy,500)
    }
```

```
    req.renderObject.credits = await tmdb.data.getPersonCombinedCreditsByID(pe
rsonId, req.renderObject.urlPath);
    req.renderObject.person = person;
    res.render("person/personinfo", req.renderObject);
}
```

Lists.pug

Denne siden er ansvarlig for å vise fram alle listene med filer som brukere av siden har laget. I denne filen kan man se at UiKit har blitt tatt i bruk mye. Hver liste blir lagt i sin egen «accordion» som er en klikkbar post i en liste. Når en klikker på denne så utvider den seg og kan vise mer informasjon. I denne er det et slideshow som viser alle filmene og seriene sine plakater som er i listen. I tillegg til dette viser den navnet på listen, hvem som har laget den og antall filmer og serier. Det er også noen If statements som blir benyttet her som sjekker om filmen har en plakat eller ikke, om den har det så blir den lagt til i slideshowet, og hvis ikke blir det lagt inn en standard plakat i stedet.

```
//-
    Pug/Html for liste siden. Her er en liste over alle tilgjenglige lister
    Author: Sigve E. Eliassen - 233511, Ørjan Dybevik - 233530
body
    div#searchDiv
    .container
        h2 #{__("LIST_LISTS")}
        ul(uk-accordion='collapsible: true')
            each item in listene
                li
                    a.uk-accordion-title(href='')=`${item.listName}`
                    .uk-accordion-content
                        a(href=`/${urlPath}/list/lists/${item.listId}` class='
anchor-remove-text-decoration')
                            .uk-card.uk-card-default.uk-grid.uk-grid-
collapse(class='uk-width-1-1@s')
                                .uk-card-media-left.uk-cover-container.uk-
width-auto
                                    .uk-position-relative.uk-visible-
toggle.uk-light(tabindex='-1' uk-slideshow='autoplay: true')
                                        .uk-slideshow-items(class='list-image-
style')
                                            - if (item.posters.length != 0)
                                                each poster in item.posters
                                                    - if (poster != null || po
ster != undefined)
```

```
                                                    img(data-
src=`https://www.themoviedb.org/t/p/w600_and_h900_bestv2/${poster}` alt='' uk-
img='' class='list-image-style')
                                                - else
                                                    img(data-
src=`/images/filmatory_default_poster.png` alt='' uk-img='' class='list-image-
style')
                                            - else
                                                img(data-
src=`/images/filmatory_default_poster.png` alt='' uk-img='' class='list-image-
style')
                                        .uk-position-center-left.uk-position-
small.uk-hidden-hover(href='#' uk-slidenav-previous='' uk-slideshow-
item='previous')
                                        .uk-position-center-right.uk-position-
small.uk-hidden-hover(href='#' uk-slidenav-next='' uk-slideshow-item='next')
                                    .uk-width-expand.uk-align-center
                                        .uk-card-body
                                            div.uk-grid(class='uk-width-1-1@s')
                                                div.uk-text-center(class='uk-
width-1-4@s')
                                                    h3.uk-card-
title=`${item.listName}`
                                                div.uk-text-center(class='uk-
width-1-4@s')
                                                    p.uk-text-
bold #{__("LIST_CREATED_BY")}
                                                    p.uk-text-
meta=`${item.userName}`
                                                div.uk-text-center(class='uk-
width-1-4@s')
                                                    p.uk-text-
bold #{__("LIST_NUMBER_OF_MOVIES")}
                                                    p.uk-text-
meta=`${item.numberOfMovies}`
                                                div.uk-text-center(class='uk-
width-1-4@s')
                                                    p.uk-text-
bold #{__("LIST_NUMBER_OF_TVS")}
                                                    p.uk-text-
meta=`${item.numberOfTvShows}`
```

listController.js

listController.js fungerer som en backend til lists.pug. Denne henter hovedsakelig listene fra API'et og legger dem inn i et objekt som forskjellige variabler som numberOfMovies, numberOfTvShows og plakatene en ser.

```
/**
 * GET for liste side
 * @param {Object} req En forespørsel fra klienten
 * @param {Object} res En respons fra server
 * @author Sigve - 233511, Sivert - 233518
 */
exports.list_get = async function(req, res) {
    let lister = await listGetter.getAllLists();
    let listene = [];
    for (const info of lister.information) {
        listene.push({
            listId: info._id,
            numberOfMovies: getNumberOfMovies(info),
            numberOfTvShows: getNumberOfTvs(info),
            posters: await getPosterUrls(await getMoviePosterUrls(info.movies,
 req.renderObject.urlPath), await getTvPosterUrls(info.tvs, req.renderObject.u
rlPath)),
            userName: await (await userHandler.getUserFromId(info.userId)).inf
ormation.username,
            listName: info.name
        })
    }
    req.renderObject.listene = listene;
    res.render("list/lists", req.renderObject);
}
```

Metodene eller funksjonene under er relativt enkle, men nødvendige for at siden skal fungere ordentlig. Her har vi metoder som tar inn to arrays, setter de sammen til ett array og blander dem i tilfeldig rekkefølge. Det er også metoder som helt enkelt bare forteller størrelsen på arrayet og metoder som henter alle URL'ene til plakatene fra databasen.

```
/**
 * Metode for å skaffe URL til plakatene til seriene
 * @param {Objekt} array Filmer
 * @param {String} languageCode Språket brukeren har valgt
 * @returns {Array} Array med URL til plakatene av seriene.
 * @author Sigve E. Eliassen - 233511
 */
async function getTvPosterUrls(array, languageCode){
    let posters = [];
    for (const tv of array) {
        let tvs = await tvHandler.getShowById(tv, languageCode);
        posters.push(tvs.information.poster_path);
    }
    return posters;
}

/**
 * Metode for å blande sammen alle film og TV posterne i tilfeldig rekkefølge.
 * @param {Array} array1 film/serie urler
 * @param {Array} array2 film/serie urler
 * @returns {Array} movieAndTvPosters Blandede postere.
 * @author Sigve E. Eliassen - 233511.
 */
async function getPosterUrls(array1, array2) {
    let movieAndTvPosters = array1.concat(array2);
    hjelpeMetoder.data.shuffleArray(movieAndTvPosters);
    return movieAndTvPosters;
}
```

Govert – 233565

about.pug

Dette er koden i pug-filen for about-pagen. Her har jeg prøvd å gjøre siden så nyttig og informativ som mulig. Med statistikker om siden på toppen, informasjon om filmatory i midten og et kontaktskjema på bunnen av siden.

```
//-

    Pug/Html for "Om oss" siden. Her er også kontakt skjema og statistikker
    Author: Ørjan Dybevik - 233530, Govert - 233565

include ../includes/head.pug
include ../includes/nav.pug
- var totalMoviesResult = totalMoviesResult;
- var totalTvResult = totalTvResult;
- var totalUserResult = totalUserResult;
- var totalReviewResult = totalReviewResult;
body
    div#searchDiv
    .container
        .uk-heading-medium.uk-text-center.about-page-
margin #{__("ABOUT_TOP_HEADING")}
        .uk-width-1-1
            .uk-column-1-4.uk-text-center.uk-align-center
                div
                    p.uk-text-large= `${totalUserResult}`
                    p #{__("ABOUT_STATISTIC_ONE")}
                div
                    p.uk-text-large= `${totalReviewResult}`
                    p #{__("ABOUT_STATISTIC_TWO")}
                div
                    p.uk-text-large= `${totalTvResult}`
                    p #{__("ABOUT_STATISTIC_THREE")}
                div
                    p.uk-text-large= `${totalMoviesResult}`
                    p #{__("ABOUT_STATISTIC_FOUR")}

        .uk-card.uk-card-default.uk-grid-collapse.uk-margin(class='uk-child-
width-1-2' uk-grid='')
            .uk-card-media-left.uk-cover-container
                img(src='/images/about-
movies.jpg' alt='' style='height:380px;')
                canvas(width='600' height='0')
            div
                    .uk-card-body
                        h3.uk-card-title Filmatory Group
                        p #{__("ABOUT_CARD_ONE_TEXT")}
```

```
        .uk-card.uk-card-default.uk-grid-collapse.uk-margin(class='uk-child-
width-1-2@s' uk-grid='')
            .uk-card-media-right.uk-cover-container(class='uk-flex-last@s')
                img(src='/images/about-
10101.jpg' alt='' style='height:380px;')
                canvas(width='600' height='0')
            div
                .uk-card-body
                    h3.uk-card-title #{__("ABOUT_CARD_TWO_HEADING")}
                    p #{__("ABOUT_CARD_TWO_TEXT")}
        #contact-form
        .uk-heading-small.uk-text-center #{__("ABOUT_CONTACT_HEADING")}
        form(action=`/${urlPath}/infosider/contactform` method='post' id='cont
actForm')
            fieldset.uk-fieldset
                legend.uk-legend.uk-width-1-3.uk-margin-auto.uk-card.uk-card-
body
                    .uk-margin
                        label.uk-text-medium #{__("ABOUT_CONTACT_TITLE")}
                        input.uk-
input(type='text' name='contacttitle' id='contactTitle')
                    .uk-margin
                        label.uk-text-medium #{__("ABOUT_CONTACT_EMAIL")}
                        input.uk-
input(type='text' name='contactmail' id='contactMail')
                    .uk-margin
                        label.uk-text-medium #{__("ABOUT_CONTACT_MESSAGE")}
                        textarea.uk-
textarea(rows='5' name='contacttext' id='contactText')
                    .uk-margin
                        button.uk-button.uk-button-default.uk-.uk-width-
medium.uk-margin(type='submit' id='contactPostBtn') #{__("BUTTON_ENTER")}
                    #about-contact-error(uk-icon="icon: warning")
include ../includes/footer.pug
script(src="/javascript/infosider/about.js")
```

en.json:

For å vise hvordan teksten på about-pagen fungerer så har jeg bare tatt med relevant kode med teksten for engelsk språk-alternativ. Så i about page så vil f.eks språkvariabelen "#{__("ABOUT_STATISTIC_FOUR")}" bli til «Movies» når brukeren har engelsk språk-valg.

```
    "ABOUT_TOP_HEADING": "About",
    "ABOUT_STATISTIC_ONE": "Registered users",
    "ABOUT_STATISTIC_TWO": "Reviews",
    "ABOUT_STATISTIC_THREE": "TV-shows",
    "ABOUT_STATISTIC_FOUR": "Movies",
    "ABOUT_CARD_ONE_TEXT": "The group was founded in 2020 with a goal of makin
g movie-
watching more social. The website was launched in 2021 while Filmatory was sti
ll under development. Today we're fully functional and with more changes comin
g we hope to be the most popular place for information about media entertainme
nt.  We welcome you to join and share with your friends.",
    "ABOUT_CARD_TWO_HEADING": "Our work",
    "ABOUT_CARD_TWO_TEXT": "This website uses API's from TMDI to gather inform
ation about any tv-
show, movie and actor. It uses MongoDB as the database, node.js for the web se
rver the frontend is built using pug and UiKit",
    "ABOUT_CONTACT_HEADING": "Contact",
    "ABOUT_CONTACT_TITLE": "Title",
    "ABOUT_CONTACT_EMAIL": "E-mail",
    "ABOUT_CONTACT_MESSAGE": "Message",
```

Index.css

Tilhørende kode i about page:

```
/* About page
/* @Author Govert Dahl - 233565
*/

.about-page-margin {
  margin-top: 36;
  margin-bottom: 40;
  }

.about-page-center-image {
  display: block;
  margin-left: auto;
  margin-right: auto;
  width: 50%;
}
```

tvHandler.js

`addToDatabase` sletter en attributt i inndata-objektet. Så oppretter den en ny modell for serie utfra en mal (tvschema). Så kjører den en save funksjon som da lagrer i databasen, deretter har den en «.then» som beskriver hva som skjer etter funksjonen. Så kjøres det en returnhandler som utfører operasjoner basert på hva resultat ble.

`getShowById` søker etter en serie og kjører returnhandler som returnerer resultatet utfra hvilken serie en søker på.

```
/**
 * Skaffer serie fra ID
 * @param {Number} tvId ID til serie
 * @returns ValidationHandler
 * @author Govert - 233565
 */
function getShowById(tvId, languageCode) {
    logger.log({level:'debug', message: `Getting tv-
show from database with id ${tvId}`});
    return Tv.findOne({id: tvId, language: languageCode}).then((doc,err) => re
turnHandler(doc,err));
}

/**
 * Legger til serie i databasen
 * @param {Object} serie Serien som skal legges til
 * @returns ValidationHandler
 * @author Govert - 233565
 */
function addToDatabase(serie) {
    logger.log({level: 'debug', message: `Adding tv-
show to database with id: ${serie.id}...`});
    delete serie.next_episode_to_air
    const tv = new Tv(serie);
    return tv.save().then((doc, err) => returnHandler(doc, err));
}
```

aboutController

`exports.about_info`: Henter inn statistikker til about-pagen.

`exports.about_post_contact`: Her har vi Ticket, som er et objekt som tar vare på innsendt informasjon fra brukerne. Videre har vi lagt til error-meldinger som brukes dersom brukeren glemmer å skrive inn informasjon eller har feil format under mail-delen. Når meldingen er

riktig utfylt blir den lagt til i databasen med addTicket, og brukeren får en tilbakemelding om at meldingen er sendt til support.

```
/**
 * Get for "om oss" siden. Henter også statistikker
 * @param {Object} req Forespørsel fra klient
 * @param {Object} res Respons fra server
 * @author Govert - 233565, Ørjan Dybevik - 233530
 */
exports.about_info = async function(req, res) {
  let totalMoviesResult = await aboutStats.totalMovies();
  let totalTvResult = await aboutStats.totalTvs();
  let totalUserResult = await aboutStats.totalUsers();
  let totalReviewResult = await aboutStats.totalReviews();

  req.renderObject.totalMoviesResult = totalMoviesResult;
  req.renderObject.totalTvResult = totalTvResult;
  req.renderObject.totalUserResult = totalUserResult;
  req.renderObject.totalReviewResult = totalReviewResult;

  res.render("infosider/about", req.renderObject);
}

/**
 *  Post for å sende ticket til support, gjør sjekker om felt er tomme
 * @param {Object} req Forespørsel fra klient
 * @param {Object} res Respons fra server
 * @returns Message
 * @author Ørjan Dybevik - 233530, Govert - 233565
 */
exports.about_post_contact = function(req, res) {
  const body = req.body.ticket; //Skaffer body fra form
  let ticket = {
    title: body.title,
    mail: body.mail,
    text: body.text
  };
  if (ticket.title === ""){
    logger.log({level: 'debug', message: `Title is missing`});
    return res.status(400).send({error: req.__('ABOUT_BACKEND_TITLE_MISSING')}
);
  }
  if (ticket.mail === ""){
    logger.log({level: 'error', message: `Mail is missing`});
    return res.status(400).send({error: req.__('ABOUT_BACKEND_MAIL_MISSING')})
;
  }
```

```
  if (ticket.text === ""){
    logger.log({level: 'error', message: `Text is missing`});
    return res.status(400).send({error: req.__('ABOUT_BACKEND_TEXT_MISSING')})
;
  }
  if (!hjelpeMetoder.data.validateEmail(ticket.mail)) {
    logger.log({level: 'error', message: `Mail is not properly formatted`});
    return res.status(400).send({error: req.__('ABOUT_BACKEND_EMAIL_FORMAT')})
;
  }
  ticketCreator.addTicket(ticket);
  res.status(200).send({message: req.__('ABOUT_BACKEND_TICKET_SUCCESS')});
}
```

ticketCreator

addTicket logger at tickets er lagt til i databasen dersom meldingen, og bruker updateDatabase for å legge meldingen til i databasen.

```
const ticketHandler = require("../../handling/ticketHandler");
const ValidationHandler = require("../../handling/ValidationHandler");
const logger = require("../../logging/logger");

/**
 * Legger ticket til i databasen
 * @param {Object} ticket Objektet med en ticket
 * @returns ValidationHandler
 * @author Ørjan Dybevik 233530, Govert - 233565
 */
async function addTicket(ticket){
    const result = await updateDatabase(ticket);
    if(!result.status){
        return result;
    }
    logger.log({level: "debug", message: "Ticket successfully added to databas
e"});
    return new ValidationHandler(true, "Successfully added ticket");
}

/**
 *  Oppdaterer databasen med den nye ticketen
 * @param {Object} ticket Objektet med en ticket
 * @returns ValidationHandler
 * @author Ørjan Dybevik 233530, Govert - 233565
 */
```

```
async function updateDatabase(ticket) {
    return await ticketHandler.addToDatabase(ticket);
}

module.exports = {addTicket}
```

hjelpemetoder.js

lagFinDato brukte vi for å få rett dato-format til front-end. Og det samme gjorde jeg med lagfinDag, lagfinÅrstall og lagfinMåned-funksjonen under hjelpemetoder.js

```
    /**
     * lagFinDato metoden gjør en dato lettere å lese
     * @param {Date} datoInn Dato som skal brukes
     * @param {String} stringTilSplitting String som kobler sammen dato
     * @returns String av dato
     * @author Govert - 233565
     */
    lagFinDato: function(datoInn, stringTilSplitting) {
        try {
            let splitDato = datoInn.split(stringTilSplitting);
            const dato = new Date(splitDato[0], splitDato[1]-1, splitDato[2])
            return dato.toLocaleString('default', { day: 'numeric', month: 'lo
ng', year: 'numeric' });
        } catch(err) {
            logger.log({level: 'error', message: `Could not format date from $
{datoInn} with string ${stringTilSplitting}! Error: ${err}`});
        }
    },
```

Tallformatering lagde jeg for å korte ned store verdier. Som f.eks filmers budsjett ol.

```
    /**
     * Formaterer tall til hele tusen/millioner/milliarder - Gov
     * @param {Number} int Nummer som skal formateres
     * @returns tall i hele tusen/millioner/milliarder
     * @author Govert - 233565
     */
    tallFormatering: function(int) {
        if (int < 999999)
            return `${Math.round(int / 1000)}K`
        if (int < 999999999)
            return `${Math.round(int / 1000000)}M`
        return `${Math.round(int / 1000000000)}B`
    },
```

Ørjan – 233530

Flerspråklighet

Dette er konfigurasjonen til språkpakken vi valgte å bruke. Språkpakken er kalt i18n og er en npm pakke. I18n er en oversettelsesmodul med dynamisk JSON-lagring.

```
const hjelpeMetoder = require('../handling/hjelpeMetoder');
const i18n = require('i18n');

/**
 *
 * @param {object} req En forespørsel fra klienten
 * @param {object} res En respons fra server
 * @param {callback} next Neste
 * @returns et objekt for å initialisere språkkoden(flerspråklighet)
 * @author Ørjan Dybevik - 233530
 */
exports.configure_language = async function(req, res, next) {
    //Konfigurerer språk
    i18n.configure({
        locales: await hjelpeMetoder.data.getAllLangCodes(),
        directory: './lang',
        defaultLocale: 'en'
      });
    return i18n.init(req, res, next);
}
```

Slik ser filene som inneholder de forskjellige språkene ut

langList er en oversikt av de språkene som er tilgjengelig på siden. Denne inneholder JSON objekter med navn på språket på engelsk, navn på språket i oversatt versjon og landskode.

```
{"availableLanguage":[{"name":"norwegian","id":"no","originalname":"Norsk"},{"
name":"english","id":"en","originalname":"English"},{"name":"german","id":"de"
```

```
,"originalname":"Deutche"},{"name":"french","id":"fr","originalname":"Français
"},{"name":"russian","id":"ru","originalname":"русский"},{"name":"chinese","id
":"zh","originalname":"中国人"}]}
```

Slik ser det ut inne hos 2 av språkfilene. Systemet vil bruke den filen til språket som er valgt. Disse blir variablene på venstre siden blir brukt direkte inni pug/html filen.

Engelsk – en.json

```
    "NAV_HOME": "Home",
    "NAV_ABOUT": "About",
    "NAV_EXPLORE": "Explore",
    "NAV_MOVIES": "Movies",
    "NAV_TVSHOWS": "Tv-Shows",
    "NAV_MORE": "More",
```

Tysk – de.json

```
    "NAV_HOME": "Zuhause",
    "NAV_ABOUT": "Über",
    "NAV_EXPLORE": "Erkunden",
    "NAV_MOVIES": "Filme",
    "NAV_TVSHOWS": "Fernsehshows",
    "NAV_MORE": "Mehr",
```

Slik blir disse variablene brukt inne i nav.pug. Om engelsk er valgt vil det da stå «Home» og «About» på disse to linjene.

```
            li
              a(href=`/${urlPath}/homepage`) #{__("NAV_HOME")}
            li
              a(href=`/${urlPath}/infosider/about`) #{__("NAV_ABOUT")}
```

Legge til nytt språk

Jeg implementerte også 3 viktige funksjoner for språk, legge til språk, redigere språk og slette språk. Dette gjøres gjennom administrator-dashbordet. Den tar inn de nye parameterne som trengs for å lage språk(Språknavn på engelsk, originalt navn på språk og språkkode) og lagres i pugBody. Denne variablen sendes til validering(validateNewLang). Denne sjekken sjekker bare at det kun er brukt tekst, at språkkoden er 2 tegn og gjør de om til liten tekst om store bokstaver er brukt. Dermed legger fs.wrifeFileSync til den nye informasjonen i «available-languages» og det lages en ny JSON fil som språkkoden som navn. Denne er en kopi av den engelske språkfilen, dermed kan den som legger til språk redigere fra engelsk til det nye språket.

```
/**
 * Post for å legge til nytt språk til nettsiden, den lager en ny JSON fil med
 språkkoden som er skrevet inn.
 * @param {Object} req Forespørsel fra klient
 * @param {Object} res Respons fra server
 * @returns
 * @author Ørjan Dybevik - 233530
 */
exports.admin_post_addlanguage = async function(req, res) {
  const pugBody = req.body.admin_add_lang_details; //Skaffer body fra form
  let languageJson = await hjelpemetoder.data.lesFil("./lang/langList.json");
  let langs = JSON.parse(languageJson.information);
  let langObj = hjelpemetoder.data.validateNewLang(pugBody, req);

  if(!langObj.status){
    logger.log({level: 'debug', message: 'Error when validating language to ad
d'});
    return res.status(400).send({error: `${langObj.information}`});
  }
  langs.availableLanguage.push(langObj.information);
  languageJson = JSON.stringify(langs);
  fs.writeFileSync("./lang/langList.json", languageJson, "utf-8");
  fs.copyFileSync("./lang/en.json", `./lang/${langObj.information.id}.json`);

  logger.log({level: 'debug', message: `Successfully added new language`});
  res.status(200).send({message: req.__('SUCCESS_LANGUAGE_ADDED')});
}
```

Redigere språk

Redigere et språk gjøres på samme side som å legge til språk.

Det begynner med en dropdown liste over de tilgjengelige språkene som er. Når bruker velger et språk vil det komme opp et stort dokument i sanntid. Dette er siden vi har brukt socket.io til dette. Koden nedenfor er funksjonen for å hente valgt språk i listen. De forskjellige valgene inneholder en ID som er unik til hvert språk, socket sender da denne ID til server for å hente fila som er valgt.

```
function getLang(){
    adminLangOptionDesc.style.display = 'none';
    let selectedLang = adminLangSelectbox.options[adminLangSelectbox.selectedI
ndex].getAttribute('data-lang-id');
    if(selectedLang !== null){
        socket.emit('getLanguage', selectedLang);
        languageOutput.innerHTML = '';
    } else {
        languageOutput.innerHTML = 'Select a language';
    }
}
```

Denne koden er svaret fra socket.io etter brukeren har valgt hvilket språk han eller hun vil hente.

```
 * @Author Ørjan Dybevik - 233530
 */
socket.on('displayLang', function(args){
    langTextareaContent.style.display = 'block';
    langTextareaContent.value = '';
    langTextareaContent.value += args;
});
```

Etter at brukeren har gjort de endringene i språkfilen som skal bli gjort og trykker på lagre vil denne eventlisteneren kjøres. Den sender språkendringer sammen med ID til språket via socket.io til serveren.

```
 * @Author Ørjan Dybevik - 233530
 */
saveLangBtn.addEventListener("click", function(e) {
    e.preventDefault();
    let selectedLang = adminLangSelectbox.options[adminLangSelectbox.selectedI
ndex].getAttribute('data-lang-id');
    if(langTextareaContent.value !== ''){
        socket.emit('saveLanguage', {langContent: langTextareaContent.value, l
angId: selectedLang})
        languageOutput.innerHTML = '';
    } else {
        languageOutput.innerHTML = 'Choose a language';
    }
});
```

Serveren oppdaterer da filen som er lagd. «args» inneholder 2 verdier. Språkkoden og de nye endringene. Fs.writefile skriver over den eksisterende filen med de nye endringene. Deretter sendes det en tilbakemelding til brukeren at endringene er godkjent.

```
 * @param {Object} socket
 * @param {Object} args språkId og nye språkendringer
 */
async function saveLanguage(socket, args){
    fs.writeFile(`./lang/${args.langId}.json`, args.langContent,(err) =>{
        if(err){
            logger.log({level: 'error', message: `There was an error when savi
ng language file: ${err}`});
        }
        logger.log({level: 'debug', message: 'File has been saved..'});
    });
    socket.emit('savedLanguage');
}
```

Slett språk

Post for å slette språk. Variabelen "langToDelete" inneholder en tekst av hvilket språk som skal slettes. Grunnen til at det er en litt tungvint måte å slette språk på er fordi jeg ville unngå at folk skal slette språk med uhell. For å slette språket må du skrive det engelske navnet på språket, og det er case sensitive. Om du ønsker å slette norsk kan du ikke skrive «noRweGian». Dette sjekker validatoren validateDeleteLang(). Dermed vil fs.unlinkSync slette språkfilen fra systemet og det vil slettes fra «available-languages».

```
exports.admin_post_deletelanguage = async function(req, res) {
  const langToDelete = req.body.admin_delete_lang_details;
  let langObj = await hjelpemetoder.data.validateDeleteLang(langToDelete.admin
dashlangdelete, req);
  if(!langObj.status){
    logger.log({level: 'error', message: 'Error when validating language to de
lete'});
    return res.status(400).send({error: `${langObj.information}`});
  }
  fs.unlinkSync(`./lang/${langObj.information.id}.json`, (err) => {
    if(err){
      logger.log({level: 'debug', message: 'Could not delete old language'});
      return res.status(400).send({error: req.__('ERROR_DELETE_LANGUAGE')});
    }
  });
  logger.log({level: 'debug', message: `Successfully deleted the language`});
  res.status(200).send({message: req.__('SUCCESS_LANGUAGE_DELETED')});
}
```

Her er eventlistener for å slette språk. Posten blir sendt via Ajax(var jqxhr). Den sender et objekt med informasjon som trengs for å fullføre posten(admin_delete_lang_details). Etter posten er fullført vil den enten gå tilbake til jqxhr.done eller jqxhr.fail. Hvilken den går til er basert på om informasjonen som ble sendt er gyldig eller ikke. Ajax blir brukt i de fleste posts på applikasjonen og de aller fleste ser like ut som denne under:

```
adminDeleteLangBtn.addEventListener("click", function(e){
    e.preventDefault();
    adminLangDeleteError.style.display = 'none';
    adminLangDeleteError.innerHTML = "";
    //Lager info
    admin_delete_lang_details = {
        admindashlangdelete: adminLangDeleteInput.value
    }
```

```
    //Sender post
    var jqxhr = $.post(`/${urlPath}/admin/deletelanguage`, {admin_delete_lang_
details: admin_delete_lang_details});

    //Om suksess
    jqxhr.done(async function(result) {
        adminLangDeleteInput.value = '';
        swal("Success!", result.message, "success",{
            buttons: false,
        });
        await new Promise(r => setTimeout(r, 3000));
        window.location.href = `/${urlPath}/admin/admindashboard`;
    });

    //Om failure
    jqxhr.fail(function(result) {
        adminLangDeleteError.style.display = 'block';
        adminLangDeleteError.innerHTML = result.responseJSON.error;
    });
});
```

Mailsystem

Jeg valgte å implementere et automatisk mailsystem for å kunne tilbakestille passord. Selve mailsystemet er implementert med hjelp av en pakke kalt «nodemailer» og «nodemailer-smtp-transport»

Variablen transporter setter opp selve mailsystemet. Her deklareres hvilken mail-host som brukes, innlogging og port. Den logger på en måte inn på gmail kontoen og gjør seg klar til å sende mail.

Transport.verify kjøres når serveren startes på nytt og gir tilbakemelding om alt er oppe å går.

Sendmail funksjonen kombinerer selve transporteren sammen med den mailen du skal sende.

```
const nodemailer = require('nodemailer');
const smtpTransport = require('nodemailer-smtp-transport');
const logger = require('../logging/logger');
const ValidationHandler = require('../handling/ValidationHandler');

/**
 * lager et gjenbrukbart transportørobjekt ved hjelp av standard SMTP-
transport
 * @author Ørjan Dybevik - 233530
 */
```

```
let transporter = nodemailer.createTransport(smtpTransport({
  service: 'gmail',
  host: 'smtp.gmail.com',
  secure: false,
  port: 465,
  auth: {
    user: process.env.EMAIL,
    pass: process.env.PASSWORD
  },
  tls: {
    rejectUnauthorized: false
  }
}));

/**
 * Sjekker at transporteren er klar til bruk og funker.
 * @author Ørjan Dybevik - 233530
 */
transporter.verify(function(err, success) {
  if (err) {
    logger.log({level: 'error', message: `Could not initialize email transport
er! ${err}`});
  } else {
      logger.log({level: 'debug', message: 'Email transporter is ready and fun
ctional'});
  }
});

/**
 * Lagrer mail funksjonen i en variabel slik den kan brukes flere steder
 * @param {Til, Fra, Emne, Text} mailOptions
 * @author Ørjan Dybevik - 233530
 */
let sendMail = (mailOptions) => {
  transporter.sendMail(mailOptions,(err, info) => {
    if(err){
      logger.log({level: 'error', message: `Could not send email! Error: ${err
}`});
      return new ValidationHandler(false, 'Could not send response mail');
    } else {
      logger.log({level: 'debug', message: `Email sent: ${info.response}`});
      return new ValidationHandler(true, info.response);
    }
  })
}

module.exports = sendMail;
```

Om du skulle sendt en mail ville den sett slik ut:

Dette eksempelet er fra passord tilbakestillingsskjema.

Den tar da inn hvem som sender mail, som er oss, denne er lagret i environment filen. Den neste er mottakeren av mailen. Det neste punktet er emne på mailen, og den siste er innholdet i mailen, dette er ren HTML for å få mailen til å se litt bedre ut enn plain tekst. Det er mulig å bruke en egen HTML som mail-template.

```
let mailer = require('../handling/mailer');

    //Sender mail
    mailer({
        from: process.env.EMAIL,
        to: userResult.information.email,
        subject: 'Password Reset Link',
        html: `
        <h2>Please click on the link below reset your password</h2>
        <h3>This link will expire in 60 minutes</h3>
        <a href='${link}'>Click here to reset your password</a>
        `
    });
```

Passordtilbakestilling

Filmatory har også mulighet for å få tilsendt mail med et skjema for tilbake stilling av passord.

Brukeren trykker da på «glemt passord» og skriver inn sin epost-adresse. Om epost-adresse blir godkjent og den finnes i vår database vil brukeren få en epost med en lenke. Denne lenken inneholder en unik «token» som består av mange forskjellige bokstaver, tall og tegn. Tokenen er generert med en pakke kalt «jsonwebtoken» eller jwt. Variabelen token genererer denne tokenen. Tokenen er en kryptert og hashet autentiseringskode(HMAC). updateUser vil da oppdatere databasen til brukeren med denne token og brukeren vil motta en mail(mail koden ovenfor) med en lenke knyttet til denne token.

```
    //Lager token og oppdaterer bruker
    const token = jwt.sign({_id: userResult.information._id}, process.env.RESE
T_PASSWORD_KEY, {expiresIn:'60m'});
    const updateUser = await userHandler.updateUser(userResult.information, {r
esetLink: token});
    if(!updateUser.status) {
        logger.log({level: 'debug', message: `Reset password link error`});
```

```
        return res.status(400).send({error: req.__('ERROR_RESET_PASSWORD_LINK')});
    }

    //Lager link
    let link = `${req.renderObject.url}/${req.renderObject.urlPath}/auth/resetpassword/${token}`
    logger.log({level: 'debug', message: `Link ${link} sent`});
```

Lenken er knyttet til denne getten. Den sender her den unike tokenen til siden

```
exports.userAuth_get_resetpassword = async function(req, res) {
    logger.log({level: 'debug', message: `Request received for /resetpassword/:token`});
    let token = req.params.token;
    req.renderObject.token = token;
    res.render("auth/resetpassword", req.renderObject);
}
```

Når brukeren trykke på bytt passord vil denne posten kjøres. Det første den sjekker er om token faktisk eksisterer. Tokenen har en timer på seg og vil slutte å fungere etter 1 time. Dette sjekker jwt.verify. Dermed gjør den de passordsjekkene som er nødvendig. Om alt stemmer vil brukeren knyttet til tokenen oppdateres med det nye passordet.

```
 * @param {Object} req Forespørsel fra klient
 * @param {Object} res Respons fra server
 * @returns Message
 * @author Ørjan Dybevik - 233530, Sivert - 233518
 */
exports.userAuth_post_resetpassword = async function(req, res) {
    logger.log({level: 'debug', message: `Request received for /resetPassword/:token`});

    //Skaffer link og info
    const pugBody = req.body.reset_pw_details;
    const resetLink = pugBody.token;

    //Sjekker om link eksisterer
    if(!resetLink){
        logger.log({level: 'error', message: `Authentication error`});
        return res.status(400).send({error: req.__('ERROR_AUTHENTICATION')});
    }
```

```
    //Sjekker info
    jwt.verify(resetLink, process.env.RESET_PASSWORD_KEY, async function(err,
decodedData) {
        if(err) {
            logger.log({level: 'debug', message: `Incorrect token or it has ex
pired`});
            return res.status(400).send({error: req.__('ERROR_EXPIRED_TOKEN')}
);
        }
        let resetLinkConfirmed = pugBody.token;

        //Skaffer bruker
        const userResult = await userHandler.getUser({resetLink: resetLinkConf
irmed})
        if(!userResult.status) {
            logger.log({level: 'debug', message: `User with this token doesn't
 exist`});
            return res.status(400).send({error: req.__('ERROR_USER_TOKEN_DOESN
T_EXIST')});
        }

        //Sjekker at passord tilfredstiller krav
        if(!(hjelpeMetoder.data.validatePassword(pugBody.newPassword))){
            logger.log({level: 'debug', message: `Password is not properly for
matted!`});
            return res.status(400).send({error: req.__('ERROR_PASSWORD_NOT_PRO
PERLY_FORMATTED')});

        }

        //Sjekker om felt er like
        if(!(pugBody.newPassword == pugBody.newPasswordRepeat)) {
            logger.log({level: 'debug', message: `Passwords do not match`});
            return res.status(400).send({error: req.__('ERROR_PASSWORD_NOT_MAT
CH')});
        }

        //Nå må vi lage ny salt for å hashe passord
        const salt = await bcrypt.genSalt(10);

        //Nå setter vi passord til det hasha passordet
        const password = await bcrypt.hash(pugBody.newPassword, salt);

        //Lagrer bruker
        const updateUser = await userHandler.updateUser(userResult.information
, {password: password, resetLink: ''});
```

```
        if(!updateUser.status) {
            logger.log({level: 'debug', message: `Could not change password`})
;
            return res.status(400).send({error: req.__('ERROR_COULD_NOT_CHANGE
_PASSWORD')});
        }

        //Suksess
        logger.log({level: 'debug', message: `Password successfully changed fo
r user`});
        res.status(200).send({message: req.__('SUCCESS_PASSWORD_CHANGE')});
    })
}
```