



Ikt201 prosjekt

Gruppe 1

av

Linor Ujkani, Sander Wesstøl, Habteab Teamé, Luka Vulovic og Erlend Tregde

i

IKT201

Internettjenester

Veiledet av Universitetslektor Christian Auby

Fakultet for teknologi og realfag

Universitetet i Agder

Grimstad, desember 2023

Sammendrag

YouWeMovie-prosjektet er utformet med målet om å levere en plattform som ikke bare er brukervennlig, men også effektiv for å muliggjøre en grundig brukeropplevelse. I dette prosjektet så Web utviklingsindustrien inkorporerer et vidt spekter innenfor programvareutvikling, og involverer ulike oppgaver som design, koding og optimalisering av online plattformer. Anvendelse av web-rammeverk er en konvensjonell praksis for å strukturere web-applikasjoner ved hjelp av ulike programmeringsspråk. I dette spesifikke tilfellet benyttes ASP.NET Core 7.0-rammeverket, utviklet for å muliggjøre mer dynamiske web-applikasjoner ved bruk av C#- programmeringsspråket. Prosjektet YouWeMovie befinner seg i et konkurransedrevet landskap blant film bloggplattformer som IMDB og Rotten Tomatoes. Når det gjelder kundebehov, har IMDB og Rotten Tomatoes i stor grad imøtekommel klientens forventninger, gitt dets relevans. Disse plattformene presenterer et bredt spekter av filmer og nyheter som appellerer til klienten. For å forbedre relevans og konkurransedyktighet, kreves det derfor tilsvarende eller forbedret funksjonalitet og responsivitet. Konkurrenter har tidligere valgt å differensiere seg gjennom unike funksjoner eller nisjer som bedre appellerer til klientens preferanser, for eksempel MALs tilnærming til IMDB og Rotten Tomatoes, der MAL har en mer demografisk orientering mot klientene og legger fokus i sjangeren "Anime". I dagens stand forventes film- og serie bloggplattformer å være responsive, informative om nye utgivelser, enkle å navigere og levere personaliserte visninger til brukeren, i tråd med generelle internett standarder. Derfor er det nødvendig å oppfylle disse forventningene samtidig som man søker å differensiere produktet for å gjøre det mer unikt og effektivt. Målet er å skape et produkt som ikke bare møter, men overgår klientenes forventninger.

Obligatorisk gruppeerklæring

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

1.	Vi erklærer herved at vår besvarelse er vårt eget arbeid, og at vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	Ja
2.	Vi erklærer videre at denne besvarelsen: <ul style="list-style-type: none">• Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands.• Ikke refererer til andres arbeid uten at det er oppgitt.• Ikke refererer til eget tidligere arbeid uten at det er oppgitt.• Har alle referansene oppgitt i litteraturlisten.• Ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	Ja
3.	Vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.	Ja
4.	Vi er kjent med at alle innleverte oppgaver kan bli plagiatskontrollert.	Ja
5.	Vi er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk.	Ja
6.	Vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider.	Ja
7.	Vi har i flertall blitt enige om at innsatsen innad i gruppen er merkbart forskjellig og ønsker dermed å vurderes individuelt. Ordinært vurderes alle deltakere i prosjektet samlet.	Ja

Publiseringssavtale

Fullmakt til elektronisk publisering av oppgaven Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven, §2).

Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

Vi gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	Ja
Er oppgaven båndlagt (konfidensiell)?	Nei
Er oppgaven unntatt offentlighet?	Nei

Forord

Dette prosjektet hører til faget IKT201-G 23H. Vi er gruppe 1. Målet med dette prosjektet er å lage en nettside ved bruk av C# som programmeringsspråk. Som rammeverk har vi valgt å bruke rider. Vår oppgave går ut på å lage et forum for filmer og serier der brukere kan lage reviews. Vi har fått et enormt læringsutbytte av dette prosjektet, og vi har virkelig fått kjenne hvordan det er å jobbe i en større gruppe.

Takk til universitetslektor Christian Auby som har veiledet oss igjennom dette prosjektet.

Grimstad

10. desember 2023

Linor Ujkani, Sander Wesstøl, Habteab Teame, Luka Vulovic og Erlend Tregde.

Innhold

1 Innledning	1
1.1 Bakgrunn	1
1.2 Produktvisjon	1
1.2.1 Hvem er det som kommer til å kjøpe vårt produkt?	2
1.2.2 Hvem er vår målgruppe?	2
1.2.3 Hvilke kundebehov skal vi presentere?	2
1.2.4 Hvilke produktegenskaper er avgjørende for å tilfredsstille behovene for å velge, og derfor skaper suksess for produktene?	2
1.2.5 Hva er tidsrammen og budsjettet for å utvikle og lansere produktet?	2
1.3 Lignende produkter	2
2 Metode	5
2.1 Vår utviklingsmetode	5
2.1.1 Requirements	5
2.1.2 Design	5
2.1.3 Implementation	5
2.1.4 Testing	6
2.1.5 Delivery	6
2.1.6 Gant Diagram	6
2.2 Prosess	7
2.2.1 Vår scrum agile prosess	7
2.2.2 Grupperoller	8
2.2.3 Arbeidsfordeling og ansvarsområder	8
2.2.4 Versjonskontroll	8
3 Krav	10
3.1 Funksjonelle krav	10
3.1.1 Må ha	10
3.1.2 Bør ha	11
3.1.3 Kan ha	11
3.1.4 Vil ikke ha	11
3.1.5 Use Case diagram	11
3.2 Tekniske krav	12
3.2.1 Må ha	12
3.2.2 Bør ha	12
3.2.3 Kan ha	12
4 Løsning	13
4.1 Produkt	13
4.1.1 Valg av farger	13
4.1.2 Hoved meny	13
4.1.3 Søkefeltet	14
4.1.4 Header	15
4.1.5 Movies og Series sidene	15
4.1.6 Content Info	16
4.1.7 Reviews page	17
4.1.8 Review	17
4.1.9 Login and register	18

4.1.10	Account page	19
4.1.11	Database modellen	19
4.2	Teknolgivalg	20
4.2.1	Rammeverk	20
4.2.2	MVC-pattern	21
4.2.3	Biblioteker	21
5	Implementasjon	23
5.1	Overordnet arkitektur	23
5.2	Oppsett	23
5.2.1	Test data fra api	23
5.2.2	Oppsett av database	25
5.2.3	Håndtering av bilder	25
5.3	Global funksjonalitet	26
5.3.1	Søkefelt	26
5.3.2	Meny knapp	28
5.4	De forskjellige sidene	28
5.4.1	Hjemmesiden	28
5.4.2	Content informasjon	29
5.4.3	Reviews	31
5.4.4	Filmer og serier	33
5.4.5	logging in og registrering	33
5.4.6	Profil siden	35
5.5	Interaction diagrams	36
5.5.1	logging in	36
5.5.2	Registrering	37
5.5.3	Søking	37
5.5.4	Legg til review	38
5.5.5	Fjern reveiw	38
5.5.6	Legg til film/serie i personlig liste	39
5.5.7	Fjern filmer/serier fra personlig liste	39
6	Testing og validering	40
6.1	Innledning	40
6.2	Hvordan vil testingen foregå?	40
6.3	Resultat av testing	40
6.3.1	Funksjonelle tester	40
6.3.2	Beskrivelse av hver funksjonell test	40
6.3.3	Samlet test resultater for funksjonelle tester	43
6.3.4	Tekniske krav	43
6.3.5	Beskrivelse av hver teknisk test	44
6.3.6	Samlet test resultater for tekniske tester	44
6.4	Konklusjon	44
7	Diskusjon	46
7.1	Prosess	46
7.1.1	Planlegging	46
7.1.2	Prioritering	46
7.1.3	Git	46
7.1.4	Hvordan har samarbeidet vært?	46

7.2	Produkt	46
7.3	Implementasjon	47
7.4	Utfordringer	47
7.5	Videre arbeid	47
8	Konklusjon	48
Tillegg A Appendix		50
A.1	Individuell rapport	50
A.1.1	Linor Ujkani:	50
A.1.2	Erlend Tregde:	50
A.1.3	Sander Wesstøl:	50
A.1.4	Habteab Teame:	50
A.1.5	Luka Vulovic:	50
A.2	Kanban	51
A.2.1	Done	51
A.2.2	Unresolved	51
A.3	Gruppe kontrakt og produkt visjon	52
A.4	Gitshortlog	53
A.5	Møtereferat	54
A.6	Presentasjonsvideo	54
A.7	Produkt info	55
A.8	Sprints	56
A.9	Timesheets	57

Figurer

1	Ting vi liker fra IMDB, hentet fra [7]	4
2	Use Case diagram av YouWeMovie	12
3	Våre valgte farger, samt deres fargekoder	13
4	Hoved menyen	14
5	Content siden	15
6	Content Info	16
7	Reviews page	17
8	Individuelt Review	18
9	Log in og register	18
10	Account page	19
11	Database modell, laget i mySQL	19
12	forholdet mellom MVC, hentet fra [17]	21
13	Prosjektets arkitektur	23
14	Funksjon som henter en film gjennom API grensesnitt.	24
15	interaksjoner mellom prosjektet og OMDb api	24
16	Våre database tablles med tilhørende relasjoner	25
17	Sjekking av filtype	26
18	Popup meny som dukker opp under søking	27
19	Meny knapp trykket inn	28
20	Henter informasjon fra databasen	28
21	Den implementerte hjemmesiden til You We Movie	29
22	benytter C# kode i HTML for å sjekke variabler	30
23	Content informasjon layout	30
24	Popup som dukker opp når du trykker på add review	31
25	Endelig utseende av review siden	32
26	Switch case med de forskjellige filter valgene	32
27	Side som viser filmer og serier, med average ymw blåbær score	33
28	Login siden	34
29	Tilpasset bildevalidering for sjekk av størrelse	34
30	Registreringssiden	35
31	profil siden	35
32	Bruker instillinger	36
33	Login siden	36
34	Registreringssiden	37
35	Søking	37
36	Legg til review	38
37	Fjern review	38
38	Legg til filmer/serier i personlig liste	39
39	Fjern filmer/serier fra personlig liste	39
40	Funksjonelle tester	41
41	Funksjonelle tester sammendrag	43
42	Teknisk tester	43
43	Teknisk test resultater	45

Tabeller

1	Tidsplan for IKT-prosjekt	7
---	-------------------------------------	---

Listinger

1 Innledning

YouWeMovie-prosjektet er konstruert med mål om å tilby en brukervennlig og effektiv plattform for en dypdykkende review opplevelse. Innenfor rammene av dette prosjektet har et dedikert rom blitt etablert for å fremme interaksjon og initiere en revolusjon i tilnærmingen til film opplevelser.

Bli med på en stimulerende reise gjennom YouWeMovie-prosjektet mens det søker å transformere tilnærmingen til film. Forpliktelsen ligger i å sikre at filmopplevelsene ikke bare er fornøyelige, men også minneverdige, oppnådd gjennom en plattform som oppfordrer til tilkobling, deling og en grundig utforskning av den filmatiske verdenen.

På YouWeMovie kan individer over 12 år registrere seg og artikulere tanker. Det går utover å være bare et verktøy for å nedtegne ideer; snarere fungerer det som en plattform der folk som løst kan bidra med betydningsfulle kommentarer.

YouWeMovie er essensielt på et oppdrag for å forenkle navigeringen gjennom det intrikate landskapet av kommende arrangementer, spesielt med fokus på filmer. Dette er ikke bare en database; snarere er det et dynamisk sted der tanker, filmanbefalinger og diskusjoner blir levende.

Forestil deg en verden der perspektiver kan deles, filmer kan anbefales, og en integrert del av et mangfoldig bilde av meninger kan utforskes. Dra nytte av et anbefalingssystem som knytter sammen nye filmer basert på jevnaldrendes preferanser. Med en tydelig og konsistent vurderingsmekanisme gir plattformen øyeblikkelig innsikt i fellesskapets meninger om en bestemt film.

Denne plattformen fungerer som en praktisk måte for enkeltpersoner å både motta og gi filmanbefalinger. Enten du er ivrig etter å dele tanker om filmer eller oppdage nye, er YouWeMovie det ideelle stedet.

YouWeMovie representerer ikke bare en nettside; det er det sentrale knutepunktet der filmverdenen systematisk utforskes, tanker kritisk analyseres, og aktiv deltagelse i utforskningen av filmens magi oppmuntres.

1.1 Bakgrunn

Filmer og serier er blitt en stor del av dagens underholdningskultur. Den stadig økende filmindustrien, mangfoldige utvalg av serier og filmer, og en økende popularitet av streamingstjenester lager et krevende marked for brukerne. Det finnes mye forskning når det kommer til å velge underholdning. Dette kan være alt fra utvikling av algoritmer som gir deg underholdningsforslag, eller reklamer.

Vi mener at den enorme veksten av serier og filmer som blir tilgjengelige i streamingsplattformer fører til et stort behov for meninger og vurderinger fra andre brukere. Dette betyr at brukere må ha en pålitelig platform der de kan dele deres meninger og vurderinger. Det allerede finnes lignende produkter som IMDB og Rotten Tomatoes, men vi mener disse plattformene mister fokus på vurderinger fra andre brukere. Dette blir diskutert nøyere i kap 1.3. Derimot vil vårt produkt ha fokus på at brukere kan legge inn vurderinger så lett og så raskt som mulig.

1.2 Produktvisjon

Vårt prosjekt har som mål å skape en forum, dedikert til film entusiaster, der de kan dele sine meninger om filmer. Hver bruker på plattformen har sin egen dedikert side for å liste

sine yndlings filmer. I tillegg skal nettsiden vår være bruker vennlig, sånn at alle kan bruke den for å dele sine meninger om filmer. Vi tok litt inspirasjon fra nettsider som IMDB og MAL. Ved å så integrere funksjoner fra begge, har vi skapt en unik og engasjerende opplevelse for våre brukere.

1.2.1 Hvem er det som kommer til å kjøpe vårt produkt?

Store selskaper innenfor samme bransje, sånt som IMDB, eller Rotten Tomatoes", kan være potensielle kjøpere, for å så bli kvitt med konkurransen.

1.2.2 Hvem er vår målgruppe?

Vår produkt setter mål på grupper sånt som, film entusiaster, filmkritikere, og andre som liker filmer og ønsker å dele og lese filmkritikk. Dette er blant annet folk som leter etter filmanbefalinger, har lyst å dele sin mening, eller å tillhøre en gruppe av folk som er på lik stilling med dem, som liker filmkritikk.

1.2.3 Hvilke kundebehov skal vi presentere?

Samfunnsengasjement: Filmentusiaster søker alltid etter et platform der de kan skape forbindleser med folk som er likesinnet, for å diskutere og debattere filmer. Forumet gir deg ett platform for slik diskusjon. Ved å tilby personlig tilpassede sider kan våre brukere få en skreddersydd opplevelse, vise frem sine favoritt filmer og anmeldelser, som gjør det lettere for andre brukere å bli kjent med andres filmsmak. Folk kan også få enkel tilgang til informasjon og gjennomgå filmene de har sett. Folk søker ofte anmeldelser og meninger før de ser en film. Denne platformen vil presentere ørlige og en vid bredde med ulike synspunkt på film. For å oppnå dette ønsker vi å gjøre platformen brukervennlig, slik at kundene forblir og bruker nettsiden i framtiden.

1.2.4 Hvilke produktekenskaper er avgjørende for å tilfredsstille behovene for å velge, og derfor skaper suksess for produktene?

Brukerprofiler: Personlige brukersider for å liste favorittfilmer og -anmeldelser. Diskusjonsfora: Dedikerte rom for ulike filmsjangre, regissører, skuespillere, etc., for å lette målrettede diskusjoner. Anmeldingssystem: Et omfattende system for å skrive, legge ut og lese anmeldelser. Søke- og filteralternativer: Finne enkelt filmer, anmeldelser og diskusjoner. Hvordan er produktet sammenlignet med eksisterende produkter, både fra konkurrenter og samme selskap? Hva er produktets unike salgsargumenter? IMDB, MAL, råtne tomater. Saken med disse nettstedene er at de er veldig rotete. Det er for mange ting du kan gjøre, og det er vanskelig å bruke. Det er derfor vi vill at vårt produkt skal være enkelt å bruke. Vi har som mål å fokusere på kundenes meninger, sånn at den skal ligne litt mere på et sosialt platform.

1.2.5 Hva er tidsrammen og budsjettet for å utvikle og lansere produktet?

Målet vårt er circa 9 uker for å fullføre prosjektet. Noe som betyr at vårt budsjett er på 12 timer i uken, per elev, for å fullføre prosjektet. I tillegg til dette har vi møter hver mandag og torsdag, for å følge med på framgangen til prosjektet.

1.3 Lignende produkter

I dette delkapittelet vil vi utforske produkter som likner på vårt produkt. Vi skal gjennomføre en litteraturstudie. I denne sammenheng er litteraturen forskjellige nettsider som

har samme funksjonaliteter som vårt produkt. Målet med vårt produkt er at brukere kan så lett som mulig legge til reviews av filmer og serier så raskt som mulig. Det finnes allerde nettsider som har denne funksjonaliteten. I dette kapittelet skal vi diskutere hvilke nettsider som gjør dette, hva vi synes er bra med dem og hva vi synes er dårlig. To nettsider vi har tatt mye insperasjon fra er rotten tomatoes og IMDB. Disse er to av de største og mest populære ressursene for filmkritikk og informasjon om filmer og serier.

Rotten Tomatoes, er en av de største nettsidene for filmkritikk. Her finner man anemledder som er gjort av både publikum som har sett filmene og serie, og eksperter i bransjen som har gitt sine tanker. Rotten Tomatoes presentere en rekke med informasjon om filmer og serier. Noe av dette er kommende serier og filmer, hvor man kan se dem, generell informasjon om filmer, og andre ting relatert med det som er nevnt.[19]

Rotten Tomatoes presenterer film kritikk på en god måte. Den gir oss informasjon om hva vanlige folk tenker om filmen, og hva velkjente kritikkere tenker om filmen. Måten det gjør dette på er ved bruk av tomater, og popcorn. Siden deres nettside har en ganske rotete hovedmeny som er litt vanskelig å navigere, har vi forkortet mengden av innhold. Dette er ikke en dårlig ting, siden vi har fokusert på det viktige, serier og film, kritikken, og ditt eget personlige sted på nettsiden. Vi har integrert deres måte å gjøre "watchlistpå og måten å gi vurdering på, altså med bruk av blåbær.[19]

IMDB er en stor database og platform som gir informasjon om filmer og serier, der brukere kan legge inn anmeldeser og rangere filmer og serier. IMDB funker dermed som et felleskap for film entusiaster. IMDB som en platform er designet slik at den er lett å bruke. En kan raskt finne relevant informasjon om det man er ute etter, om det gjelder filmer eller serier [7].

Etter vi har studert IMDB er det et par ting vi gjerne vil forbrede. Vi mener IMDB er rotete og det kan til tider være litt for mye informasjon for hver side. Vi synes også IMDB setter fokus vekk fra anmeldeser ved å ha mange forskjellige funksjonaliteter. Dermed vil vårt produkt fokusere mere på å gjøre det lett å lage reviews. Her er noen ting vi liker ved IMDB:

Med en gang man kommer inn på IMDB blir man møtt med en liste med filmer, og hvis man skroller litt lengre ned finner man en slideshow av de beste seriene og filmene, det er noe vi ønsker å intergtere i vår nettside og. På IMDB kan en også sortere filmer og serier etter ulike kriterier, som gjør at man lett kan finne noe man interesserer seg for. Når man trykker inn på en film eller serie kan man se user reviews. En ting vi ikke liker med IMDB er at disse ligger godt gjemt og man må bla et stykke for å finne dem. Det er nemlig mye som vi kan ta med oss fra IMDB. Den har gitt oss mange gode ideer til hva som kan implementeres, men også gitt oss et godt syn på ting vi ikke vil ha med. Det vi har fått mest ut av med å studere IMDB er at denne nettsiden er veldig rotete, noe vi helst vil unngå i vårt prosjekt.

Featured today

Upcoming Picks: 'May December,' 'Silent Night,' and More

[See the list](#)

see showtimes

+ Add to Watchlist Added by 173K users

1-50 of 399 324

Sort by: Popularity ▾

Rank	Title	Year	Length	Rating	Votes	Metascore	
1.	Napoleon	2023	2h 38m	15	★ 6.7 (39K)	☆ Rate	64 Metascore
2.	The Hunger Games: The Ballad of Songbirds & Snakes	2023	2h 37m	12	★ 7.2 (38K)	☆ Rate	54 Metascore
3.	Monarch: Legacy of Monsters	2023	-	12	★ 7.6 (7K)	☆ Rate	TV Series

Coriolanus Snow mentors and develops feelings for the female District 12 tribute during the 10th Hunger Games.

Votes 39,173

Votes 37,743

Votes 7,026

620 Reviews

Hide Spoilers Filter by Rating: Show All ▾ Sort by: Featured ▾

★★★ 4/10

My biggest disappointment this year

Ridley Scott's NAPOLEON feels like the highlight reel of a lengthy miniseries. Considering there's a 4-hour cut of this film, that explains it all.

NAPOLEON is certainly good spectacle. The battle scenes are breathtaking. Unfortunately, it's also shallow. I know Scott has sneered at viewers criticizing the historical inaccuracies in the film, but I'm more bothered by a total lack of interesting character psychology or even coherent storytelling. Characters pop in and out, leaving little impression in their brief scenes. Relationships between characters are barely fleshed out, including that of Napoleon and Josephine, which dominates the running time. Also, potentially unpopular opinion, I thought Joaquin Phoenix's performance was a one-note bore.

Figur 1: Ting vi liker fra IMDB, hentet fra [7]

2 Metode

2.1 Vår utviklingsmetode

En utviklingsmetode kan se annerledes ut utfra hvilken metode og prosess som brukes. I waterfall modellen er stadiene til software development: requirements, design, implementation, testing, Deployment and Maintenance [9, s.9]. I waterfall kan en bare gjøre en av stadiene om gangen. Derimot agile metoden som vi bruker kan vi jobbe med flere av stadiene samtidig [9]. På denne måten kan vi holde utviklingen vår smidig. Da dette er første gang vi utfører et prosjekt på denne størrelsen passer agile metoder ved hjelp av scrum prefekt for oss, da dette vil gi oss rom for prøving og feiling. For at prosjektet vårt skal bli en suksess har vi valgt å følge denne utviklingsmetoden når det lar seg gjøre, ved hjelp av agile metoder.

2.1.1 Requirements

Requirements er prosessen om å skaffe krav, både funksjonelle og ikke-funksjonelle krav. Men før vi kan skaffe oss kraven må vi finne ut av hva problemrådet vårt er. Vi må først finne problemrådet før vi kan finne kravene ifølge første lov om software engineering. Denne loven sier at en programvareutvikler alltid må finne ut av problemrådet [9]. Men før vi kan gjøre dette må vi finne ut av hva prosjektet vårt skal handle om. Derfor gikk det først møtet vårt ut på å idemyldre. Det er denne delen som gir grunnlaget for vårt prosjekt. I dette stadiet samlet vi alle ideene vi hadde. Ideemyldring er ikke bare en måte å skaffe ideer på men også en viktig prosess innenfor planlegging, teambygging og å finne et problemrådet. Nå som vi har funnet problemområdet ved hjelp av idemyldring vil det bli lett å lage kravene. Under dette støttet er det også viktig å diskutere hvordan typer krav som skal brukes. Krav blir diskutert nøyne i kapittel 3. Under requirements fasen lager vi også dokumentasjon på liknende løsninger. Vi tar med i dokumentasjonen det som er bra og det som ikke er så bra. Der skriver vi også ned hva vi kan gjøre annerledes.

2.1.2 Design

Etter at de fleste kravene er blitt diskutert kan vi gå videre til design fasen. I design fasen lager vi en plan for systemet vårt. Vi bruker kravene våre til å lage use case diagram. Dette er diskutert mere under kap 3.1.5. Under design fasen er målet å bruke kravene vi har laget til å designe hvordan systemet skal fungere. I tillegg til dette er det i design fasen vi diskuterer databasen, vi diskuterer hvilken database som skal brukes og hvordan databasediagramet skal se ut.(må skrive mer her eller referere til der vi snakker om database) En annen ting som gjøres under designfasen er å lage skisser av nettsiden ved hjelp av figma, dette er nøyere diskutert og forklart under kap 4.1. Poenget med dette er å ha en klar oppfatning av hvordan produktet skal se ut før vi begynner å implementere. Dette betyr at hvis vi har klart å planlegge kravene og designet godt nok, vil implementasjons delen gå raskt og enkelt. Heldigvis bruker vi simdige metoder og vi har rom for feil. I tillegg har vi muligheter til å gå tilbake å endre krav eller deler av designet underveis hvis det trengs. Dette er en av mange fordeler ved å bruke agile metoder.

2.1.3 Implementation

Etter vi har laget en god plan starter utviklingsdelen. Dette er delen da vi programmerer. Her er det viktig at alle i gruppen har god kjennskap til git og bit branching, da dette er

vikttige konsepter å mestre for at prosjektet skal bli velykket. Git vil sørge for å opprettholde en organisert og effektiv prosjektstruktur. Siden vi er såpass mange i gruppen er git essensielt for vårt samarbeid. Git vil dermed gi oss mulighet til å utvikle parallelt. En annen god grunn til å bruke git ofte er at endringer kan spores tilbake og vi kan reversere om det er nødvendig.

2.1.4 Testing

Etter vi er ferdig med å utvikle må produktet testes. Vi tester produktet for å forsikre oss at programmet gjør hva den skal gjøre og at den møter alle kravene. Gjennom testing fasen vil vi også forhåpentligvis finne uønskede oppførsel fra systemet. Under testing brukes kravene som vi lagde under design fasen til å finne ut om funksjonaliteten i programmet funker.

2.1.5 Delivery

For oss blir dette steget leveranse av oppgaven, siden vi ikke skal utgi produktet. I dette stadiet øker vi fokus på rapportskrivingen slik at vi er klar til å levere prosjektet. Hvis vi skulle ha lansert produktet vårt ville dette stadiet blitt kalt deployment og maintainence.

2.1.6 Gant Diagram

Hvert steg i vårt gant diagram er viktig for vår suksess. Vi har utviklet hvert trinn slik at vi får en trinnvis gjennomføring av vår plan. Her har vært trinn en spesifikk rolle som burde oppfylles for at prosjektet skal bli vellykket. Det er viktig å være oppmerksom på at vi jobber smidig og at selv om et steg er ferdig, kan vi gå tilbake hvis det trengs. Dette er en av mange fordeler med agile metoder.

Under første steg har vi 1. Requirements og design, dette er forklart i 2.1.2 og 2.1.1. Under disse har vi 1.1 Lage krav, dette er nøyde forklart her 2.1.1. I punkt 1.2 Lage skisser lager vi skisser i figma, slik at vi får et visuelt referansepunkt. I punkt 1.3 lage dokumentasjon henter vi inn informasjon om liknende løsninger. God dokumentasjon vil gjøre det tydeligere på hva prosjektet skal gå ut på, og hva vi ikke vil lage. Punkt 1.4 går ut på å lage use case diagram. Dette er gjort slik at det er klart for alle i gruppa hvordan programmet kan implementeres. Det siste punktet er 1.5 legge inn user stories. Dette er gjort for å bli vant med å bruke jira, og det er viktig å påpeke at vi kommer til å fortsette å bruke jira gjennom hele prosjektet.

Det neste steget er 2. Implementasjon som forklart i 2.1.3. Under dette steget er den første delen 2.1 prosjektstruktur i git. Før vi begynner å programere må vi ha en godt struktur i git. Dette er viktig og holde dette på en organisert måte slik at det styrker samarbeidet og at det er lett å håndtere kode endringer. Den neste delen er 2.2 Dele ut oppgaver. Dette steget handler om hvem som skal implementere hva. Vi setter fokus på å fordele oppgavene jevnt, slik at alle kan bidra. Etter dette kan vi starte å implementere koden, det er i dette trinnet koden blir skrevet, i samsvar med kravene og designet.

Etter dette har vi 3. testing og feilretting som er forklart under 2.1.4. Det siste steget er 4. leveranse. I dette stadiet øker vi fokus på rapport skriving, og innen uke 49 burde rapporten bli ferdig. I tillegg skal vi også lage en presentasjon av produktet. I uke 50 skal produktet leveres. Slik ser vår fulle gant-diagram ut:

IKT-prosjekt	Start	Slutt
1. Requirements/Design	42	43
1.1 Lage krav	42	43
1.2 Lage skisser	42	43
1.3 Lage dokumentasjon	42	43
1.4 Lage use case diagram	42	43
1.5 Legge inn user story i Jira	43	43
2. Implementasjon	44	47
2.1 Prosjektstruktur i Git, branching	44	44
2.2 Dele ut oppgaver	44	44
2.3 Utvikling	44	47
2.4 Ferdigstille produktet	47	47
3. Test og feilretting	48	48
4. Leveranse	48	50
4.1 Øke fokus på rapport	48	48
4.2 Ferdigstille rapport	49	49
4.3 Lage presentasjon av produkt	49	49
4.4 Finpuss rapport	49	49
4.5 Leveranseforberedelse	50	50

Tabell 1: Tidsplan for IKT-prosjekt

2.2 Prosess

I uke 42-43 var fokuset vårt på å etablere krav, lage skisser, utvikle dokumentasjon, konstruere use case-diagrammer og legge inn brukerhistorier i Jira, samt å bestemme produktnavnet 'You We Movie' og velge et fargetema. Vi la også vekt på å gjøre oss kjent med Scrum, Jira og Git, samtidig som vi startet med skisser og dokumentasjon.

Uke 44-47 var dedikert til utvikling. Denne perioden innebar å sette opp prosjektstrukturen i Git, deletere oppgaver og foreta forgreninger. Vi konsentrerte oss også om å ferdigstille databasestrukturen og utforske funksjoner som en diskusjonsfunksjon for å fullføre prosjektet.

Uke 48 var dedikert til bare testing og feilsøking, med fokus på å sikre at alle jobbens krav ble fullt ut oppfylt.

Fra uke 48 til 50 var oppmerksomheten vår rettet mot å ferdigstille produktet. Denne fasen inkluderte testing, retting av feil og videreutvikling av rapporten. Vi fullførte rapporten, utarbeidet en presentasjon av produktet og forberedte leveringen. Disse ukene var avgjørende for å sikre at presentasjonen og rapporten vår var klar til å deles.

Vi har faste møter to ganger i uken, på mandager eller tirsdag og på torsdager eller fredager. På torsdager og fredager konsentrerer vi oss om utviklingsarbeid, evaluering av fremdriften og justering av planer basert på prosjektets utvikling og teamets tilbakemeldinger. Hver mandag jobber vi med Git, hvor vi integrerer alt arbeidet som er utført i løpet av uken, inkludert helgene. Mandagene eller tirsdagene er dagene vi også har sprint møte. Her diskuterer vi hvordan uken kommer til å bli.

2.2.1 Vår scrum agile prosess

Siden vi bruker jira vil vært person som er med i prosjektet få et eget JIRA prospekt. Alle er dermed selv ansvarlige for å logge hvor mye og hva de har gjort i en task. Sprintene vår

varer i 1 uke hver, den starter på mandag og slutter på søndag kveld. Hver mandag elelr tirsdag har vi et sprint møte der vi planlegger oppgaven for uken, i tillegg går vi igjennom hva vi skal gjøre i denne sprinten. Når vi har avtalt hva som skal gjøres i sprinten, tsarter vi sprinten. Vi har ikke hatt noe daglige scrum møter, men vi har heller møtt så ofte som det passer, evt tas møtet gjennom meldinger på discord. Alle gruppemedlemmene velger seg issues som de kan gjøre, da skal man i tillegg flytte issuen til in progress. Etter en er ferdig med dagens arbeid logger man timene i jira. Når en er ferdig med oppgaven setter man den i DONE kolonnen.

2.2.2 Grupperoller

Deltakelse, positivt samarbeid og et godt arbeidsmiljø var nøkkelen til suksess i vårt prosjekt. Vi løste oppgavene effektivt, hvor hver person spilte en viktig rolle i koding og rapportskriving. Gruppen fungerte godt, og alle bidro aktivt med gode samarbeidsferdigheter. Erlend håndterte backend-arbeidet backend og påtok seg betydelige oppgaver som Scrum Master og gruppeleder. Linor, som utmerket seg i sin rolle, tok seg av detaljene i frontenden frontend av prosjektet, han var også ansvarlig for design. Sander, et hjelpsomt og aktivt teammedlem, jobbet både med front- og backend backend and frontend og administrerte databasen. Han hadde også ansvar for API-en. Luka fokuserte på de utfordrende sidene i backenden, mens Habteab demonstrerte spesielle ferdigheter i backend og behandlet identitetsrelaterte problemstillinger. I tillegg har alle gruppemedlemmer vært med å skrive denne rapporten.

Prosjektets suksess skyldtes at alle bidro, og vi hadde et positivt og hyggelig arbeidsmiljø. Vi delte ideer, samarbeidet om å løse problemer og satte pris på hverandres bidrag. Dette teamarbeidet og den gode atmosfæren påvirket prosjektets suksess i stor grad.

2.2.3 Arbeidsfordeling og ansvarsområder

Vi organiserte oppgavene slik at hvert teammedlem hadde tydelig definerte ansvarsområder, og samarbeidet effektivt gjennom hele prosessen. Arbeidsbyrden ble jevnt fordelt, og alle bidro positivt. Noen av oss hadde god ferdigheter innen Git og koding, og deres ekspertise var avgjørende når det kom til å løse eventuelle utfordringer knyttet til koden.

Denne sammearbeide av ferdigheter bidro til å skape et fantastisk arbeidsmiljø. Teammedlemmene som utmerket seg med Git var alltid tilgjengelige for å hjelpe når vi støtte på koderelaterte problemer, og deres innsats bidro til raske løsninger. Den overordnede atmosfæren i teamet var positiv, med aktiv deltagelse fra alle, noe som resulterte i en smidig og produktiv arbeidsprosess. Alt i alt var dette en svært positiv opplevelse preget av sømløst samarbeid og gjensidig støtte.

2.2.4 Versjonskontroll

Som versjonskontroll har vi valgt å bruke git. Git er et anerkjent versjonskontroll system som er ofte brukt i programmeringsverden. Git kan brukes av en bruker eller flere brukere i et prosjekt. En av de viktigste funksjonene i git er evnen til å håndtere branching og merging [3]. Ved bruk av git kan vi jobbe i separate deler av prosjektet samtidig. Git sikrer også historikken av koden, dette er en viktig faktor i store utviklingsprosjekter slik som vårt.

En av de viktigste konseptene innen git er branching. En kan se på branching som en gren eller et spor, denne grenen starter på et punkt kjent som commit. Standard grenen kalles for main, det er dette som er hovedgrenen. Ut ifra main går det ofte ut flere andre

branches. Ved å jobbe i separate branches kan en gruppe jobbe sammen på tvers av hverandre. Når arbeidet på en branch er ferdig kan man sammenflette den med main. Da brukes en prosess som kalles merge. For å merge kan en skrive: git merge -ff-only <feature>. Etter en har fullført merge er en standar praksis å slette den branchen man jobbet i, da skriver man: git branch -d <name>. Andre git kommandoer som ofte bruker er git checkout main, denne sjekker ut branchen som du vil jobbe ut fra (ofte main). Git pull, sørger for at du har de siste commitene på denne branchen. git checkout -b <feature-name>, lager en branch for dine endringer [3].

I vårt prosjekt spiller git en viktig rolle for vårt samarbeid. Som server bruker vi bitbucket, som er en av mange git-servere. Siden vi jobber med agile metoder/scrum passer jira perfekt for oss. Jira lar oss planlegge å utføre sprinter på en effektiv måte. Jira lar oss også prioritere og organisere oppgavene våre på en effektiv måte.

3 Krav

I dette prosjektet har vi valgt å lage funksjonelle krav og tekniske krav. Disse er utarbeidet slik at vårt produkt dekker behovende som trengs. Vi har også valgt å prioritere kravene våre. På denne måten har vi en pekepinne på hva som er viktig å gjøre i forhold til krav vi kan ta senere. Der har vi valgt å bruke funksjonelle krav og tekniske krav. Vi har valgt å bruke MoSCoW som metode for å skrive kravene våre fordi det er en enkel og rett frem måte å lage krav på. Ved bruk av MoSCoW er det lett for alle gruppemedlemmer og andre å vite nøyaktig hva de viktigste kravene som mås ha, og det er lett å vite hvilke krav som ikke er så viktige. I tillegg understreker MoSCoW hvilke krav vi ikke vil ha [14].

3.1 Funksjonelle krav

Funksjonelle krav er krav som beskriver mål eller oppgaver brukeren kan gjøre. Vi har valgt å skrive opp kravene våre som brukerhistorier. Grunnen til vi har valgt å skrive kravene som brukerhistorier er fordi da er det enklere å lage et use case diagram av hele problemstillingen vår. Det er viktig å påpeke at vi i vårt produkt skiller mellom registrerte brukere og ikke registrerte brukere. Dette kommer tydlig frem i våre funksjonelle krav, da registrerte brukere har flere funksjonaliteter og muligheter enn en bruker som ikke er registrert. De funksjonelle kravene våre er delt inn i må ha, bør ha, kan ha og vil ikke ha, ved hjelp av MoSCoW metoden [14]. Må ha, er de viktigste funksjonelle kravene i vårt produkt. Uten disse kravene har vi ikke noe produkt. Bør ha, er krav som er viktige som burde være med. Kan ha, er krav som vi har lyst til å legge til men ikke har tid til, eller krav som kanskje heller ikke er nødvendige. Vil ikke ha, er funksjoner som kan ha vært aktuelle men som ble valgt bort på grunn av tidspress eller andre årsaker som f.eks at det er unødvendig eller krav som ikke passer til vårt produkt mål.

3.1.1 Må ha

1. Som en registrert bruker ønsker jeg å kunne lage review av filmer og serier
2. Som en bruker ønsker jeg å kunne se reviews og rating av andre brukere
3. Som en registrert bruker ønsker jeg å kunne rate filmer og serier.
4. Som en bruker ønsker jeg å kunne registrere meg
5. Som en bruker ønsker jeg å kunne logge inn.
6. Som en registrert bruker ønsker jeg å kunne logge ut
7. Som en bruker ønsker jeg å kunne bruke search baren for å søke etter filmer og serier
8. Som en registrert bruker ønsker jeg å kunne legge til filmer i min liste
9. Som en registrert bruker ønsker jeg å kunne legge til serier i min liste
10. Som en registrert bruker ønsker jeg å kunne fjerne filmer i min liste
11. Som en registrert bruker ønsker jeg å kunne fjerne serier i min liste
12. Som en registrert bruker ønsker jeg å kunne slette mine reviews
13. Som en registrert bruker ønsker jeg å kunne se min liste over filmer og serier
14. Som en bruker ønsker jeg å kunne få informasjon om alle filmer/serier
15. Som en bruker ønsker jeg å kunne få informasjon om hver enkel film og serie

16. Som en bruker ønsker jeg å kunne få informasjon om hver enkel review
17. Som en registrert bruker ønsker jeg å kunne endre passord
18. Som en registrert bruker ønsker jeg å kunne legge til mobilnummer
19. Som en bruker ønsker jeg å kunne navigere enkelt rundt i nettsiden

3.1.2 Bør ha

1. Som en bruker ønsker jeg å kunne sortere filmer og serier etter ulike kriterier for å finne filmer/serier etter mine interesser
2. Som en bruker ønsker jeg å kunne sortere reviews etter ulike kriterier for å finne reviews etter mine interesser

3.1.3 Kan ha

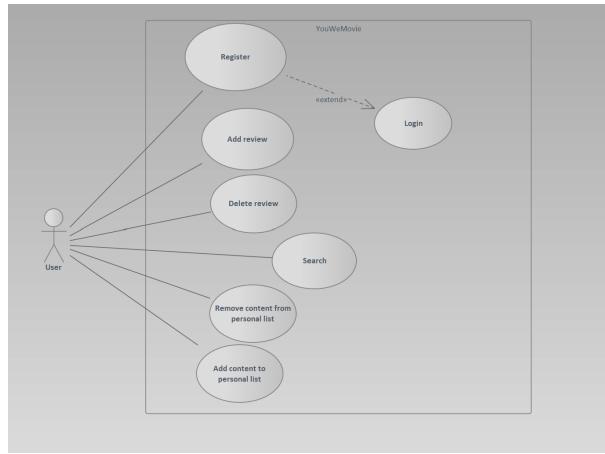
1. Som en bruker ønsker jeg å kunne se på andre brukeres profiler.
2. Som en registrert bruker ønsker jeg å kunne legge til filmer og serier som ikke er i databasen
3. Som en bruker ønsker jeg å kunne gi tilbakemelding og rapportere andre brukere for å beholde et godt felleskap.
4. Som en bruker ønsker jeg å få forslag til filmer og serier basert på filmer/serier jeg har i listen og filmer/serier jeg har ratet.
5. Som en bruker ønsker jeg å få nyheter om kommende filmer og serier
6. Som en bruker ønsker jeg å få informasjon om når nye filmer kommer på lokale kinoer.
7. Som en registrert bruker ønsker jeg å kunne legge til venner

3.1.4 Vil ikke ha

- Som en bruker ønsker jeg ikke å kunne se en trailer av filmer og serier

3.1.5 Use Case diagram

Under tidlig utvikling av vårt produkt i uke 42 lagde vi et use case diagram for YouWeMovie systemet. Dette use case diagrammet er veldig overfladisk, men viser godt de viktigste use casene og hensikten med systemer, selv om vi har lagt til flere use cases underveis dekker dette use case diagrammet de viktigste funksjonalitetene. Det er dette som kommer til å bli vårt produkts grunnlag. Dette use case diagrammet viser godt de store forskjellene mellom brukere som ikke er logget inn, og registrerte brukere. Use case diagrammet vår refererer til Må ha: 1, 2, 3, 4, 5, 7, 13, 10, 8, 9, 12, 11



Figur 2: Use Case diagram av YouWeMovie

3.2 Tekniske krav

Alle nødvendige krav som handler om hastighet, personvern, sikkerhet kan en kalle tekniske krav. Dette inkluderer alle spesifikasjoner som ikke direkte angår funksjonaliteter eller oppgavener brukere skal kunne utføre [9, side 75].

3.2.1 Må ha

1. Passord må sikres i databasen ved bruk av passord hasing i asp.net sin standar
2. Nettsiden må kunne tilpasse seg etter normale skjermstørrelser på pc inkludert 1366x768, 1920x1080, og større skjermer.
3. Nettsiden skal funke på vanlige nettlesere som Chrome og FireFox
4. Nettsiden må funke for linux og windows brukere
5. Koden skal bli skrevet på en god og oversiktlig måte
6. Koden skal skrives med hjelp av kodespråkene javascript og cSharp og skal styles med HTML og css.
7. Jetbrains rider skal brukes under utviklingen
8. en bruker skal ikke kunne ha større profilbilde enn 250 kb

3.2.2 Bør ha

1. Nettsiden bør ha tofaktot autitisering for å gjør innlogging sikrere

3.2.3 Kan ha

1. Nettsiden kan ha tilpassende funksjoner slik at den funker på mobil og nettbrett
2. Persondata må slettes når brukerprofilen slettes

4 Løsning

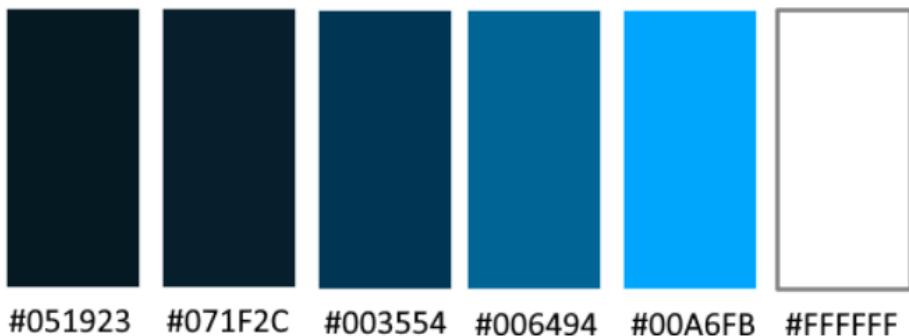
4.1 Produkt

Før vi startet ved å implementere programmet, lagde vi et forslag av nettsiden i figma. Dette var bestående av alle sidene vi ønsket å lage. Sidene i figma ble laget ganske tidlig under prosessen. Ved å følge gant planen vår ble vi helt ferdig med å lage figma skissene våre i uke 43. I tillegg hadde vi laget mesteparten av use case-ene, samt use case diagrammet var også ferdig. Dette gjorde at vi hadde en god oversikt når utviklingsfasen startet i uke 44. Målet med skissene våre var at de skulle oppfylle alle våre funksjonelle krav.

Som diskutert under 1.3 Lignende produkter har vi gjennomført en grundig litteraturstudie av eksisterende løsninger. Dette har vi gjort for å finne inspirasjon, men også svakheter og ting som kan forbredes ved de eksisterende løsningene. Dette kommer tydlig frem under våre skisser i figma.

4.1.1 Valg av farger

Selv om dette ikke er et webdesign fokusert fag har vi tatt noen bevisste valg når det kommer til farger. Valg av farger spiller en viktig rolle i hvordan bruker opplevelsen er. Ifølge [5] er blått en anerkjent farge for nettsider på grunn av dens assosiasjoner. Blått er et symbol på mange forskjellige positive egenskaper. Fargen blå er tilknyttet egenskaper som pålitelighet og intelligens, noe som passer perfekt til vårt produkt som setter fokus på et trykt felleskap. Fordi brukere har ofte en tendens til å få en sikker og en trygg følelse av fargen blå. I tillegg formidler fargen blå om effektiv kommunikasjon [5], noe som passer perfekt til vårt forum der man deler meninger om filmer med hverandre. Dette er fargene vi har endt opp med å bruke:



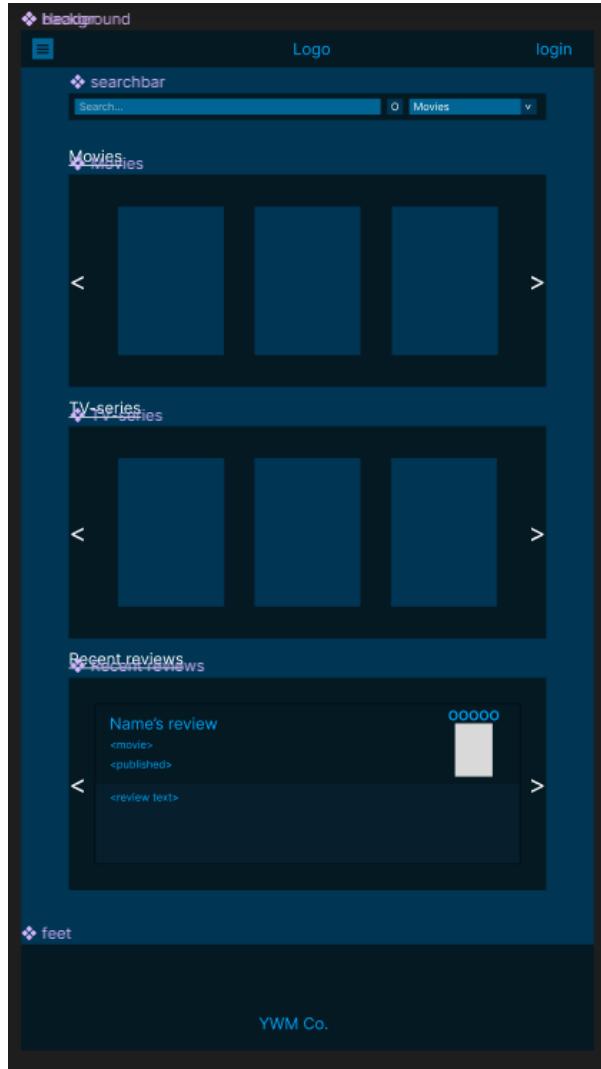
Figur 3: Våre valgte farger, samt deres fargekoder

For å konkludere med dette er ikke valget av blå som hovedfarge tilfeldig. Det er et strategisk valg etter å ha lest seg opp på vitenskaplige artikler om farger, og dens rolle i nettsider. Ved å velge fargen blå ser vil nettsiden se bra ut, og den vil formidle positive assosiasjoner som trygghet, pålitelighet og logikk, som vil forbrede brukeropplevelsen. For høpentlig vis vil dette gjøre det lettere for bruker å legge ut reviews og fortsett å bruke nettsiden.

4.1.2 Hoved meny

Som nevnt over har skissene av nettsiden en blå-mørk aktig fargepalett. Siden produktet vårt har fokus på filmer, vil den mørke-blå tonen lage en kino-lignende atmosfære. Vår

planlagte hovedmeny ser slik ut:



Figur 4: Hoved menyen

Hensikten med hoved menyen er å gi brukeren en umiddelbar oversikt over hovedfunksjonene i vår nettide. I hovedmenyen kan du enkelt å raskt se alle filmene og alle seriene som er i vår database. Vi bruker en scroll funksjon slik at man kan skrolle gjennom filmene i horisontal retning. Vi har også valgt å skille nøye mellom filmer og serier. Under dette har vi lagt til plass for reviews. Denne delen fremhever budskapet i YouWeMovie, å dele meninger om filmer og serier gjennom reviews. På denne måten kan brukere raskt lese reviews fra andre brukere. I review delen står det også hvem som har laget reviewen, hvilken film det gjelder, når den ble publisert og selve reviewen. I tillegg inkluderer review en rating fra 1-5, dette blir et raskt og tydlig visuelt intrykk av ratingen. At man kan se reviews svarer til use casen: må ha 2

4.1.3 Søkefeltet

Felles for alle sidene i vår nettside har vi valgt å plassere et søkefelt. Vi har valgt å plassere søkefeltet sentralt helt på toppen av siden. Dette gjør det lett tilgjengelig og veldig tydelig for brukeren. Hvis brukeren ønsker å finne noe raskt kan en søke. Vi har også valgt at man kan filtrere søker etter filmer, serier eller brukere. Siden søkerfeltet er felles for alle

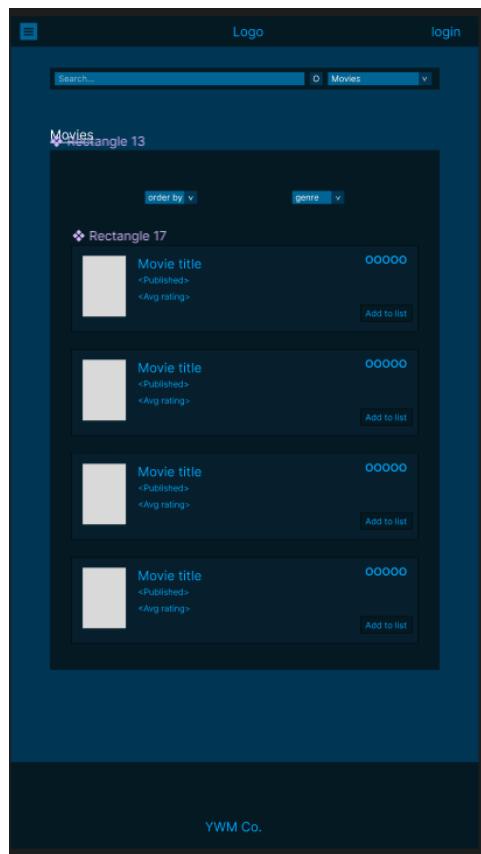
sidene kan et partial view være aktuelt for dens implemtering. Søkefeltet svarer til use case må ha 7.

4.1.4 Header

Over search baren har vi headeren til nettsiden. Headeren er også felles for alle sidene, og hinter også som en partial view. Vi har valgt å sette logoen vår helt på toppen i midten, denne vil også fungere som en måte å komme tilbake til hoved menyen i enhver tid. Til høyre for logen har vi login, som sender deg videre til login siden. Helt til venstre av headeren har vi valgt å ha en dropdown meny. Fra denne dropdown menyen vil du kunne komme deg til de fleste andre sidene. Dette vil føre til en enkel passage mellom de forskjellige sidene våre, som vil forbrede brukeropplevelsen. Dette refererer til use case må ha 19.

4.1.5 Movies og Series sidene

Ved å trykke inn på Movies eller Series vil du komme inn på en side som viser informasjon om alle filmene eller alle seriene, denne siden refererer til use case Må ha 14 . For å komme til disse sidene må du enten trykke på Movies eller series i home pagen, eller bruke dropdown menyen som er tilgjengelig i alle sider. Som et resultat av dette vil du bli møtt med følgende side:



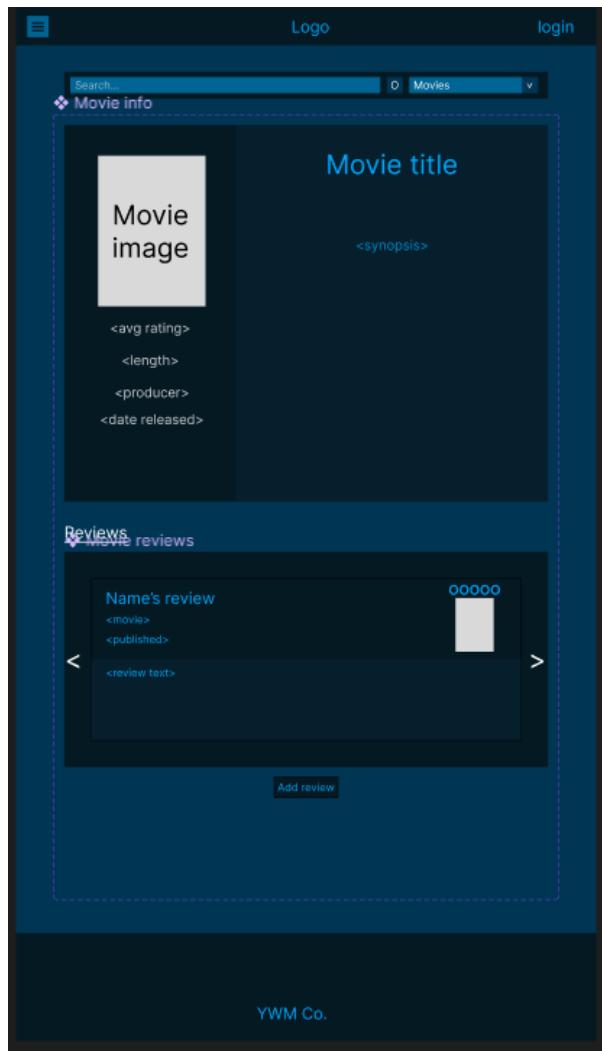
Figur 5: Content siden

Siden vi har planlagt å ha mange filmer og serier i databasen har vi valgt å legge til sorteringsfunksjoner. Brukeren skal kunne sortere filmer og serier etter ulike kriterier. Dette refereres til use caseen Bør ha 1. I hver konteiner er hver film/serie listet ut med bilde av filmen, tittelen, publisering dato, rating og en add to list/remove from list

knapp. Dette gir et standar format på hvordan filmene skal bli listet ut. Ved siden av hver film er det en add to list/remove from list knapp, basert på om den ligger i listen din eller ikke. Her kan brukere legge til filmene/seriene i sin liste raskt. Dette refererer til use case må ha nr 8-11. Rett over add/remove har vi valgt å plassere rating, på denne måten kan brukere raskt rate en film/serie om det ønskes. Dette oppfyller use case må ha 3.

4.1.6 Content Info

Hvis du synes en film eller serie virker interessant kan du klikke deg inn på den, enten ved å trykke på bildet av filmen eller tittel. Da kommer du til content Info, som inneholder spesifikk informasjon om en serie eller en film. Denne siden ser slik ut:



Figur 6: Content Info

Det sentrale i denne siden er informasjons boksen som viser detaljert informasjon om den filmen/serien du klikket inn på. Dette samsvarer til use case må ha 15. Det inkluderer gjennomsnittlig rating, lengde, produsent, utgivelses dato, og handlingsreferat. En slik detaljert beskrivelse kan hjelpe brukeren i å ta beslutningen om filmen/serien er verdt å se eller ikke.

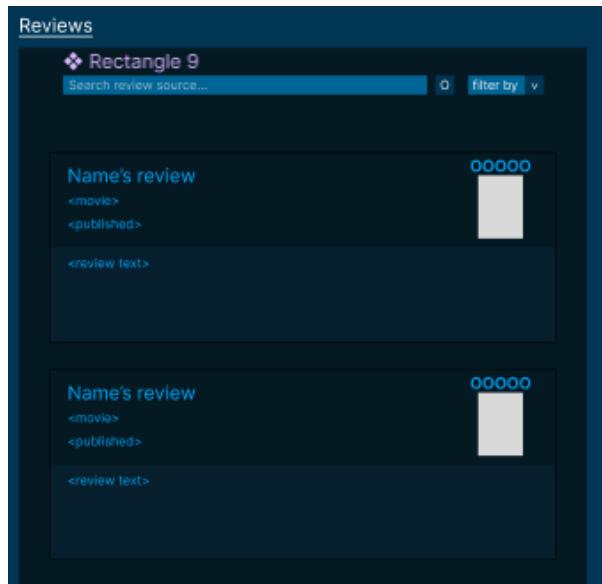
Under informasjonsboksen har vi valgt å plassere reviews. Her har du muligheten til å se

reviews om valgt film/serie fra andre brukere. Refererer til use case må ha 2. Vi har valgt å kun vise et review om gangen, og du kan bla frem og tilbake mellom reviews. Dette har vi gjort slik at skjermen ikke blir overbelastet med informasjon. Hvis ikke informasjonen om filmen/serien har overtalt deg til å se den, kanskje et godt review kan. I review-en kan du tydlig se hva den har blitt rated og hva brukeren synes om filmen.

Hvis du selv er interessert i å legge til en review kan du gjøre det ved å trykke på Add review knappen, som er plassert rett under reviews. Denne knappen samsvarer til use case må ha 1.

4.1.7 Reviews page

Hvis du er interessert i å se alle reviews kan du dra til reviews siden. Du kan komme hit ved å gå inn i dropdown menyen og klikke på reviews. Her vil alle reviews være tilgjengelige. Refererer til use case må ha 2. Dette er gjort for å bedre brukeropplevelsen. Ved å plassere alle reviews i en side gjør vi det veldig enkelt for brukeren å finne reviews. Ved å plassere reviews inn i dropdown menyen er det lett å navigere seg til reviews siden.



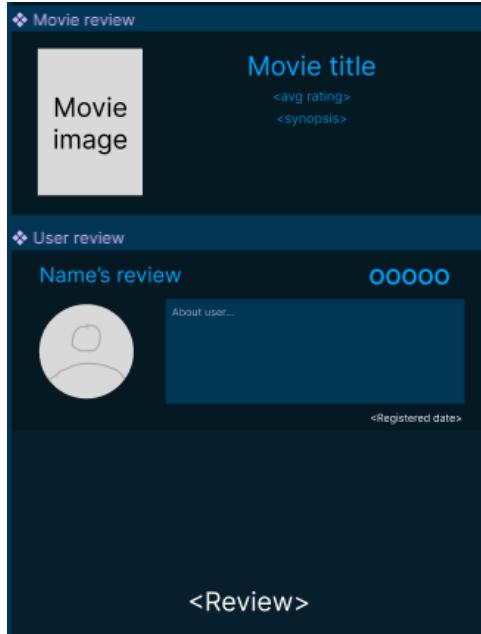
Figur 7: Reviews page

I reviews siden vil det være mulig å kunne sortere reviews etter filmer/serier, rating osv. refererer til use case bør ha 2. Hvis brukeren har enkelte filmer, serier eller sjangere de er innterset i vil det å tilby en sorteringsfunksjon bety mye. På denne måten kan brukere lett finne reviews de er interessert i.

4.1.8 Review

Ved å klikke seg inn på et review fra reviews siden kommer du inn til et inviduelt review. refererer til use case må ha 16. Et inviduelt review vil inneholde mere detaljert informasjon om hva reviewen handler om. Det vil også gjøre navigasjon i nettsiden mye enklere da brukere enkelt kan finne seg frem til en spesifik review uten å søke gjennom hele forumet.

Når du kommer inn til et inviduelt review vil du først bli møtt med innhold om hvilken film/serie reviewen handler om. Dette har vi valgt å gjøre fordi da vil brukeren få informasjon om hvilken film eller serie reviewen refererer til, noe som gir klarhet og kontekst.



Figur 8: Individuelt Review

Under informasjonen om content kommer du til reviewen. Her får du litt informasjon om brukeren, samt hva brukeren har skrevet om filmen. Å inkludere informasjon om brukeren kan gi leserene en forståelse fra brukerens prspektiv. I tillegg vil dette føre til å bygge tillitt til leseren, noe som kan gjøre reviewen mere troverdig,

4.1.9 Login and register

Ut ifra use casene våre har vi en stor forskjell mellom registrerte brukere og vanlige brukere. Derfor har vi valgt å gjøre dett lett for brukere å registrere seg og logge inn. Dette kan gjøres ved å trykke på register eller log in, som er plassert i headeren på alle sider. Å kunne registrere seg og logge inn refererer til use case må ha 4 og 5. Ved å trykke på enten login eller register vil du bli møtt med følgende:

(a) Logg Inn
(b) Registrer

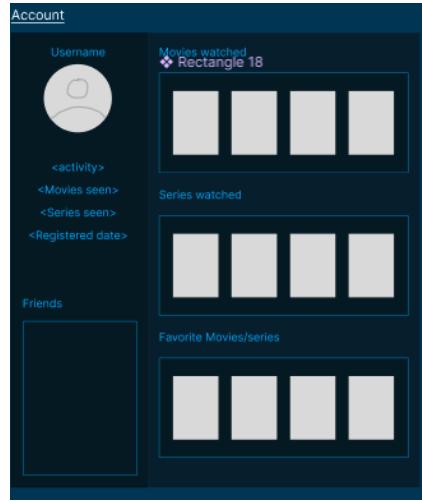
Figur 9: Log in og register

Valget ved å ha registrering i headeren gjør at nye brukere enkelt kan registrere seg. I tillegg kan det hende at brukere vil utforske nettsiden før de velger å registrere seg. Mye av det samme gjelder for login. Disse er viktige funksjoner i vår nettside, og vi har dermed

valgt plasseringen av dette slik at det er lett tilgjengelig for brukere.

4.1.10 Account page

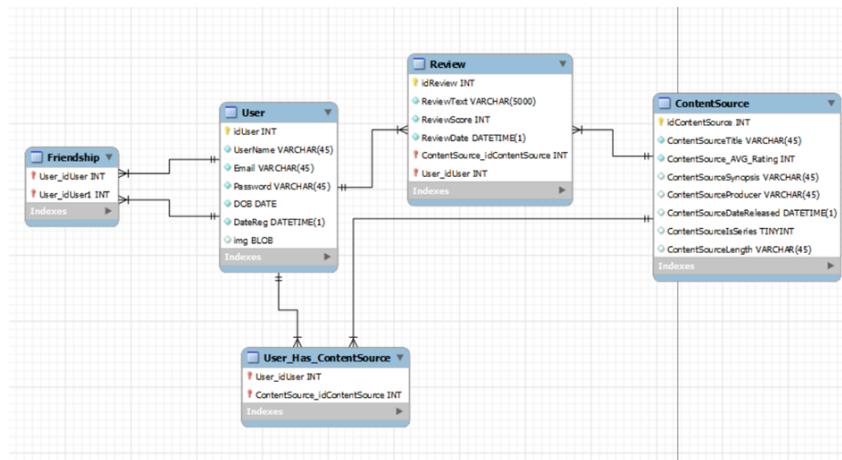
For å oppfylle use case må ha 13, har vi valgt å ha en account page. I denne siden har vi informasjon om brukeren, samt lister vi ut alle filmene og seriene som er i brukerens liste. Dette gjør det lett for brukeren å vite hvilke filmer og serier som er i ens liste. I Account siden har vi også tenkt å kunne endre passord og kunne legge til mobilnummer. Dette refererer til use case må ha 18 og 17.



Figur 10: Account page

4.1.11 Database modellen

Før vi kan gå videre til implementasjon stadiet må vi moddelere databasen etter kravene. Ved å ta de viktigste klassene våre kan vi lage en database modell. For å lage en god database har vi lest oss opp hva som kreves for å oppnå dette. Databasen modellen kobler sammen brukeren og content med reviews som en mange til mange tabell. Modellen går hovedsaklig ut på at en bruker kan legge ut mange reviews, og at en film eller serie kan ha mange reviews. Databasen er mere diskutert i kapittel 5.2.2. I figur 11 kan en se hvordan databasen endte opp under løsningen.



Figur 11: Database modell, laget i mySQL

4.2 Teknologivalg

4.2.1 Rammeverk

For å klare å løse vårt problem vil et passende rammeverk være viktig for vår suksess. Vi trenger et rammeverk der vi kan bruke entity og identity framework. Med dette mener vi at rammeverket asp .net er det perfekte rammeverket for vårt prosjekt.

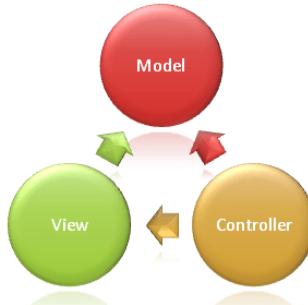
I vårt prosjekt spiller databasen en viktig rolle. Vi må ha koblinger mellom bruker og reviews, og koblinger mellom Content og reviews som disutert under 4.1.11. Disse koblingene vil sørge for at en bruker kan legge til reviews som refererer til use case å ha 1. Dette kan vi gjøre med hjelp av entity framework. Dette er ORM som vil si object-relational-mapping [1]. Det er en viktig del av .net, og er en av mange grunner til at vi har valgt dette rammeverket. Entity framework lar oss også bruke databasen ved å bruke høy nivå objektmodeller. Dette trenger vi for å hente ut og bruke databasen, men også for å kunne sortere ut filmer eller serier etter behov. Vi tenker å bruke kode først tilnærming der vi definerer forholdene mellom entitetene ved hjelp av attributtene. Etter vi implementerer dette vil entity framework lage databasen for oss automatisk [1].

Som beskrevet under use casene er det en stor forskjell mellom brukere og ikke registrerte brukere. Derfor trenger vi å bruke identity framework i vårt prosjekt slik at brukere kan logge inn. Identity framework gir oss innebygd funksjonaliteter som bruker registrering og login [2]. På denne måten kan vi raskt sette opp en trykt og sikkert innloggingssystem. I følge use caseene våre er det bare registrerte brukere som har lov til å gjøre visse ting. Dette kan være å legge til reviews. Dette har bare registrerte brukere lov til å gjøre. Ved å bruke identity framework for rollehåndtering og autorisering kan vi kontrollere hvem som har tilgang til å legge til reviews.

Valget av asp.net som vårt rammeverk er viktig for vår suksess. Vi synes at asp.net skiller seg ut fra andre rammeverk som node.js. Dette er fordi asp.net er bra for å håndtere brukerinteraksjoner samt databehandling som er to viktige deler innen vårt prosjekt. Når vi integrerer dette sammen med entity framework vil dette forenkle interaksjoner med databasen betydelig, fordi det tillater oss å moddelere databasen ved hjelp av C sharp. Identity framework tilbyr gode løsninger på registrering og login, samt rollehåndtering [2]. Dette er viktig for å sikre at bare autoriserte brukere kan få tilgang til forskjellige funksjoner som å legge inn reviews. For å konkludere vil asp.net sammen med entity og identity framework bli en god trio som passer perfekt til vårt prosjekt.

4.2.2 MVC-pattern

Siden vi bruker .net er det naturlig å bruke MVC-pattern. Dette er en god måte å organisere kode på. Ved bruk av MVC i .net kan vi dele koden i tre hovedkomponenter. De er modeller, views og kontrollere [17]. Modeller er ansvarlig for data og kodespråket C# er ofte brukt her. Det er i kontrolleren vi også vil initialisere databasen. Views skal vise dataen fra modellen. Views er ofte skrevet i razor-pages, der kodespråket er en blanding av HTML of C#. Til slutt har vi kontrollere som fungerer som et bindeledd mellom modellene og views. Ofte motar kontrollerene HTTP forspørslar som gettere og setttere [17]. Vi bruker dermed kontrollere med actions, og disse kontrollerene skal returnere views. Figur 12 gir en god forklaring på dette. Den viser forholdet mellom modeller, views og kontrollere.



Figur 12: forholdet mellom MVC, hentet fra [17]

Vi har valgt MVC-arkitektur i vårt prosjekt fordi MVC gir tydelige separasjon mellom modeller, views og kontrollere. Dette vil bidra til en organisert og kode som er lett å vedlikeholde. Siden hver komponent kan kjøres uavhengig av hverandre gjør det at koden blir lett å teste. MVC lar oss også gjennomføre autorisering som nevnt under identiy framework på en lettere måte.

4.2.3 Biblioteker

I utviklingen av nettstedet benyttet vi flere biblioteker for å håndtere spesifikke funksjoner og forenkle utviklingsprosessen. Nedenfor er en liste over de viktigste NuGet-pakkene som er inkludert i vårt prosjekt:

- Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore - versjon 7.0.11
Dette gir støtte for håndtering av feil. også for unntak i tilkobling med Entity Framework Core-operasjoner. [2]
- Microsoft.AspNetCore.Identity.EntityFrameworkCore - versjon 7.0.11
dette brukes for identitets håndtering i forindelse med Entity Framework Core. [2]
- Microsoft.AspNetCore.Identity.UI - versjon 7.0.11
dette hjelper for inkluderer brukergrensesnittkomponenter for identitets håndtering i ASP.NET Core. [2]
- Microsoft.EntityFrameworkCore.Design - versjon 7.0.11
dette gir designrelaterte verktøy for Entity Framework Cor. [2]
- Microsoft.EntityFrameworkCore.Sqlite - versjon 7.0.11
Legger til støtte for SQL lite - database i Entity Framework Core. [2]

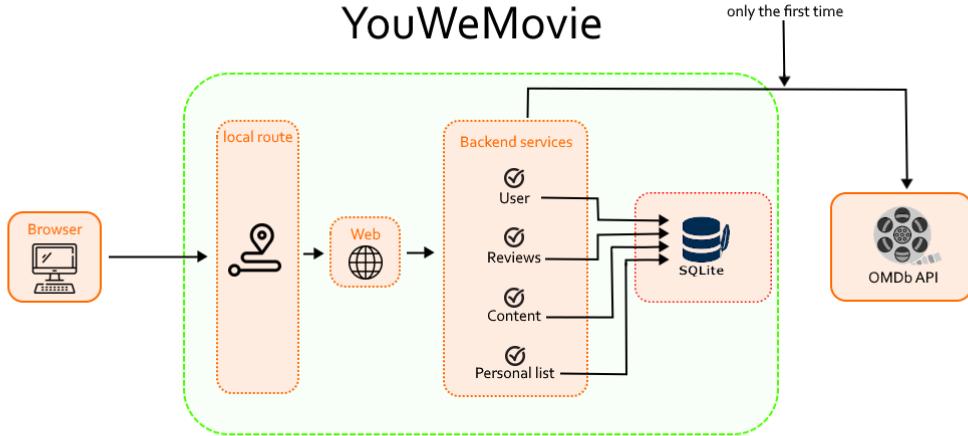
- Microsoft.EntityFrameworkCore.SqlServer - versjon 7.0.11
dette gir støtte for Microsoft SQL Server i Entity framework Core. [2]
- Microsoft.EntityFrameworkCore.Tools - versjon 7.0.11
dette inkludere verktøy for Entity framework core-migrasjoner og databaseadministrasjon.
- Microsoft.Extensions.Http - versjon 7.0.0
Dett gir utvidelser for HTTp-relatert funksjonalitet. [2]
- Microsoft.jQuery.Unobtrusive.Ajax - versjon 3.2.6
Dette brukes å legge til AJaX-støtte klientens side med jQuery. [2]
- Microsoft.VisualStudio.Web.CodeGeneration.Design - versjon 7.0.10
dette includerer disignverktøy for webkodingsOppgaver i rider [2]
- Npgsql - versjon 7.0.4
Dette gir støtte for PostgresSql-databaseforbindeler. [2]

Disse bibliotekene gir funksjonalitet for alt fra database kobling , identitets håndtering, til klient-siden dynamikk ved hjelp av AJAX.

5 Implementasjon

5.1 Overordnet arkitektur

For å vise logikken og infrastrukturen til vårt prosjekt har vi valgt å bruke denne endelige arkitekturen:



Figur 13: Prosjektets arkitektur

I denne arkitekturen er "browser" det brukeren benytter for å aksessere vårt produkt. Denne bruker den lokale ip adressen til en localhost server (ettersom vi ikke har lansert dette prosjektet). Denne local routen viser vår nettside ("web") som brukeren interacter med. Denne nettsiden i frontend kaller, og tar i bruk backend service, og alle komponentene inni den. Disse komponentene tar i bruk database, som er der vi lagrer all informasjon om brukere, filmer/serier, reviews og liknende. Denne informasjonen om filmer/serier henter vi fra en extern tredjeparts api, hvor denne hentinga av informasjon bare skjer første gang, under første oppstart av programmet. Dersom vi hadde lansert dette prosjektet, hadde local routeblitt gjort om til public route", og brukeren hadde fått tilgang til vår side igjennom en DNS server. Kommunikasjonen med ekstern api hadde blitt gjort flere ganger for å hente inn nye filmer vi ikke hadde.

5.2 Oppsett

5.2.1 Test data fra api

Får å forsikre en suksess i vårt produkt var det essensielt at vi anskaffet test data så tidlig som mulig. For å oppnå dette valgte vi å kommunisere med en tredjeparts aktør gjennom et API grensesnitt. Ved å benytte oss av en tredjeparts API, vil vi redusere utviklingstiden, og kostnadene som ville oppstått dersom vi skulle re-implementere dette selv. Vi valgte å ta i bruk The Open Movie Database (OMDb) sin API, ettersom de var villige til å gi oss tusen forespørsler per dag, uten kostnad [15]. For å få tilgang til å bruke denne api-en fikk vi tilgitt en personlig api-key, fra deres tjeneste, som lot oss aksessere informasjonen deres fra vårt prosjekt. I tillegg hadde denne API-en all informasjonen vi trengte om hver film, noe en del andre API-er ikke hadde. Dermed opprettet vi en liste med navn på filmer vi ønsket å lagre i databasen, for å så kjøre en HTTP GET forespørsel

for hver film. Dette gjør vi slik:

```
private static async Task<string> GetInfoFromApi(HttpClient client, string requestUrl)
{
    // the json received will be a part of the response body, it will be stored in this variable
    string? responseBody;

    // http get request
    var httpRequest = new HttpRequestMessage
    {
        Method = HttpMethod.Get,
        RequestUri = new Uri(requestUrl) // URL: https://www.omdbapi.com/?apikey=APIKEY&plot=full&t= + movie name
    };

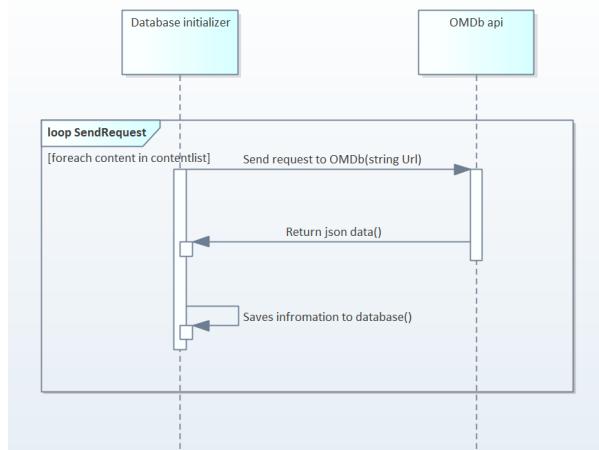
    // the response from the website
    var httpResponse = await client.SendAsync(httpRequest);
    // makes sure the response is ok (code 200)
    if (httpResponse.IsSuccessStatusCode)
        responseBody = await httpResponse.Content.ReadAsStringAsync();
    else
        return "";

    return responseBody;
}
```

Figur 14: Funksjon som henter en film gjennom API grensesnitt.

En velykkt respons fra deres API, med retur kode 200, returnerete informasjonen i JavaScript Object Notation (JSON) format, inni kroppen på responsen. Dette er også det mest vanlige formatet returnert av API-er. For å bruke denne informasjonen som foreløpig bare var av typen string, parset vi JSON stringen for å få ønsket datatyper, samt koblet denne opp mot klassen "Content" som vi bruker for filmer og serier. Disse to prosessene kan også bli kalt for å deserialize JSON. Til dette bruker vi en innebygd funksjon, hvor vi identifiserer eventuelle variabler som ikke samstemmer med JSON navnene.

Ettersom vi bruker denne apien til å bare hente testdata, har vi valgt å gjøre dette bare en gang i første startup av prosjektet. Da blir hele denne prosessen gjennomført i initialiseringen av database, og vi kan kommentere ut initializer funksjonen i program.cs etter første startup. Nå som vi har lagret all denne informasjonen i databasen, bruker vi bare denne informasjonen videre til alle funksjonene vi har i programmet. Derfor valgte vi å inkludere databasen i bitbucket også, slik at all dataen vi hentet inn ble lagret, og kunne brukes videre. Dette kan se noe slikt ut:



Figur 15: interaksjoner mellom prosjektet og OMDb api

Vi valgte å lagre dataen i egen database, fordi vi ønsker å ha mer kontroll over dataen vi bruker, samt gjøre denne dataen lettere aksesserbar når programmet vårt faktisk kjører. Dette vil si at ettersom vi ikke lagrer alle filmene/seriene, vil ikke search eller andre funksjoner kunne vise alle eksisterende filmer/serier. Her kunne vi valgt å lage en funksjon som tilater brukere selv å legge inn filmer gjennom denne apien, som beskrevet

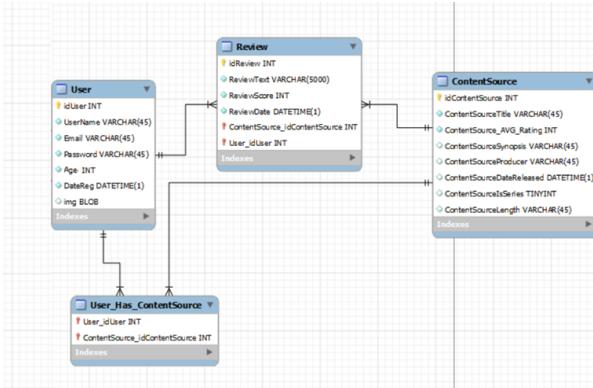
i funksjonell krav "kan ha 2". Men siden vi brukte dataen bare som test data, var vi sikre på at dette produktet også ville fungere på de manglende filmene/seriene, og vi valgte å ikke legge dette til i denne omgang.

5.2.2 Oppsett av database

I prosjektet vårt har vi valgt å benytte oss av en code-first approach til å sette opp databasen. På denne måten kan vi sette opp databasen ved bruk av modellene vi lager i selve koden. Dette gjør databasen enklere å modifisere etter kodeendringer i stedet for å ha et forhåndsbestemt database oppsett. I tillegg tar vi bruk av en sqlite database. Når denne sqlite databasen utfører spørninger bruker den forskjellige låser som opererer på tilkoblingsnivået til databasen [4]. Dette vil si at når en lås blir brukt i database, vil denne låsen bli brukt på hele databasen. Dermed vil hele databasen bli låst dersom en del av databasen blir låst. Heldigvis, kan databasen bruke flere "shared locks" samtidig, og vi har dermed mulighet til å lese fra databasen flere plasser samtidig [6]. Derimot vil hele databasen bli låst når vi skal oppdatere eller sette noe inn i databasen, som blir gjort med en "exclusive lock" [6].

Selv om disse databasene ikke er veldig optimaliserte til mange oppdateringer i databasen, funker de mer enn godt nok til vårt nåværende produkt. Vi har også prøvd å holde våre sql spørninger ganske korte, for å miske tiden vi låser databasen på. I tillegg er dette en veldig simpel og lett database å opprettholde, som ikke trenger noen forhånds konfigurasjoner. Noe vi tar godt nytte av på grunn av tidbegrensninger.

Vår valgte databasestruktur ble litt annerledes enn hva vi planlag i 4.1.11, ettersom vi valgte å ikke ta med et vennesystem på grunn av tidsbegrensninger. Dermed ble vår databasestruktur slik:



Figur 16: Våre database tabller med tilhørende relasjoner

5.2.3 Håndtering av bilder

I vårt produkt lar vi brukerne laste opp bildet til sin profil under registreringen. Dette bildet har vi valgt å lagre i en SQLite database, i stedet for å ta i bruk et filsystem som håndterer lagringen av bildene for oss. Grunnen til dette er, ifølge [21] og [20], vil det være mer effektivt for oss å oppbevare bildene i sqlite databasen dersom bildene er mindre enn 250kb. Denne størrelsen er nok til at en bruker skal kunne laste opp et frivillig profilbilde

uten problemer.

Ettersom sqlite databaser ikke har egen bilde datatype vi kan ta i bruk for lagring av bilder, benytter vi oss av en liste med bytes [8]. På denne måten vil vi kunne konvertere, og lagre hvert enkelt bilde som en lang liste med bytes. Dette kalles for en BLOB, og står for Binary large object[8]. For at brukeren ikke skal laste opp for store bilder, eller laste opp andre ting enn bilder, har vi satt en maks størrelse på bildet på 250kb som bedt om i tekniske krav ”må ha 8”. Samt vi sjekker hvilke filtype som blir lastet opp. Dette blir alt håndtert under registreringen av en bruker.

Ved å lagre alle bilder vi mottar som en BLOB, skaper dette andre problemer. Disse problemene oppstår når vi faktisk skal vise bildet på nettsiden. Ettersom vi er interreserte i å vise et faktisk bilde og ikke en liste med bytes, må vi konvertere denne listen tilbake til en bildefil. I og med at alle de forskjellige bildetypene blir lagret som bytes, vil vi ikke, uten tiltak, vite hvilke filtype bildet hadde før vi lagret det i databasen. For å oppnå riktig konvertering av byte listen, er det essensielt for oss å vite hvilke filtype vi mottok. Dette er hvor filsigraturer også kalt ”the magic number” blir relevant. En filsigratur er de første bytene i en fil, og de forteller hvilke filtype den etterfulgte dataen er lagret i. På denne måten kan vi sjekke filtypen til bildene, med å se på de første bytene i listen. Dette blir gjort slik:

```
//checks file signature also known as magic number
@Usage @SanderWestad"
private static string CheckSignature(Span<byte> pic) // 'span' is used to get only the first few bytes
{
    // makes sure file is big enough to be checked
    if (pic.Length != 5)
        return string.Empty;

    // actual checking of the first bytes:
    return pic[0] switch
    {
        0xFF when pic[1] == 0x00 => "jpeg",                                     //jpeg and jpg
        0x89 when pic[1] == 0x50 && pic[2] == 0x4E && pic[3] == 0x47 => "png", //png
        0x47 when pic[1] == 0x47 && pic[2] == 0x44 => "gif",                //gif
        _ => ""
    };
}
```

Figur 17: Sjekking av filtype

Vi har valgt å implementere egen funksjon til å sjekke filtypen til bildet, ettersom noen av de innebygde funksjonene bare støttet windows operativsystemet, men ikke linux. Dette skapte problemer for linux brukere, og førte til at bildet ikke ble konvertert riktig. Dersom et bilde ikke blir konvertert riktig, eller til en ukjent filtype, har vi valgt å vise et standard profilbilde. Dette vises også dersom en bruker har valgt å ikke laste opp ett bilde.

På grunn av tidsbegrensninger, har vi ikke implementert noen form for bildemoderering. Dette vil si at foreløpig har brukeren mulighet til å laste opp hva enn de måtte ønske at bildet viser. Dette er noe vi ville implementert en stopper for dersom vi hadde hatt mer tid.

5.3 Global funksjonalitet

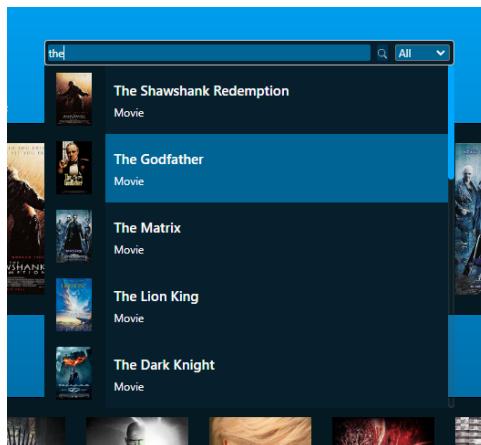
5.3.1 Søkefelt

Som nevnt tidligere i 4.1.3, sørger søkefeltet for at brukere kan lettere navigere til deres ønskede destinasjon i vårt produkt. For å vise dette søkerfeltet på hver side, har vi valgt å implementere den i et partial view som vi kalte `_SearchPartial`. Metoden vi brukte for

å implementere dette søkefeltet var ved å feste en eventlistener til selve søkefeletet som blir aktivert hver gang en bruker skriver i søkerfeltet, samt en til kategori knappen, som blir aktivert når en endring forekommer. Disse to eventlistenerene kaller på to funksjoner som begge befinner seg i samme javascript klasse. Funksjonen for kategoriene omgjør den valgte kategorien til et tall mellom 0-3 for hvert filter alternativ, som søkerfunksjonen kan benytte seg av ettersom den befinner seg i den samme klassen. Funksjonen for selve søkerfeltet tar i bruk JQuery for å utnytte AJAX til å sende ansykrone HTTP GET forespørsler til en hjelpe kontroller, som inkluderer hva brukeren skrev inn og hvilke kategori som er valgt. Denne kontrolleren utfører en databaseforespørsel basert på de dataene den fikk inn fra forespørselen. Utifra valgte kategori, vil den søke igjennom tilsvarende tabeller i databasen før den returnerer resultatene i JSON format. Vi har satt søkerfunksjonen i javascripten til å sende en request for hvert annenhvert tastetrykk. Dette gjør vi ettersom, hvis en bruker skal søke opp en ønsket film/serie som beskrevet i funksjonelle krav ”må ha 7”, trenger brukeren mer enn 1 bokstav uansett. Dette sparar også databasen for mange unødvendige spørninger.

Den returnerte JSON verdiene blir håndtert av javascript før den blir sendt til en funksjon i kontrolleren som returnerer en partial view med denne informasjonen. Til dette bruker vi en HTTP POST request som inneholder denne informasjonen i kroppen på forespørslen, slik at vi ikke får en veldig lang HTTP GET request. Hvis vi hadde hatt mer tid på prosjektet, hadde vi latt kontrolleren returnere en liste med partial views med en gang, i stedet for en og en. Ettersom vi da slipper å sende informasjonen frem og tilbake mellom kontrolleren og javascriptet, for å vise den til brukeren. Vi brukte den metoden vi har nå fordi den var rasktest å implementere, noe som vår nyttig for vår begrensete tid.

Ved å ta i bruk AJAX har vi mulighet til å sende GET requests uten å måtte laste inn siden på nytt [18]. Dette passer perfekt ettersom vi ikke ønsker å laste inn siden for hver gang brukeren skriver noe i søkerfeltet. Når vi skal vise resultatene for brukeren, har vi valgt å bare gjøre dette i form av en popup boks under søkerfeltet. Dette er i en skjult div konteiner som er tom. For hvert element som matcher med det brukeren har skrevet inn blir dette lagt til ett partial view i div konteineren, og den blir gjort synlig for brukeren. Dette ser slik ut:

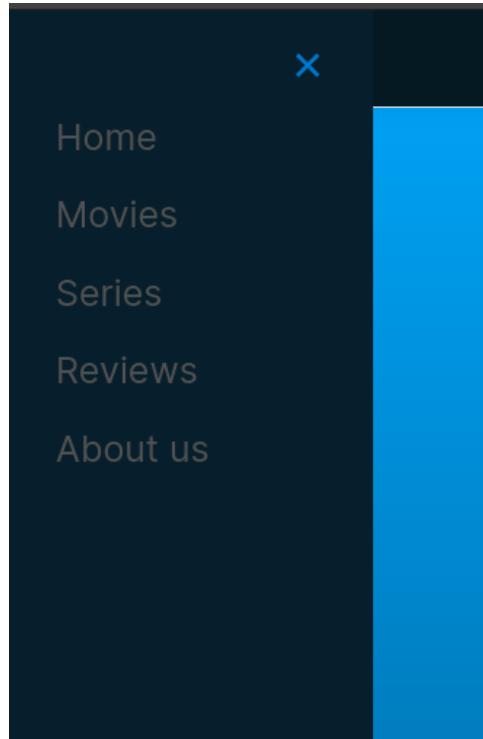


Figur 18: Popup meny som dukker opp under søking

Denne kommunikasjonen blir vist i 35.

5.3.2 Meny knapp

Denne knappen gjør det lettere for brukere å navigere seg rundt på nettisden vår som ønsket i ”må ha 19”. Ettersom den viser en liste med hvilke hovedsider vi har, som man en bruker kan enkelt trykke på for å umidelbart bli redirected til den siden. Måten denne knappen fungerer på er at i har lagt på en eventlistener som blir aktivert når brukeren trykker på knappen. Da endrer vi stylingen på konteneeren som inneholder denne informasjonen. Vi øker størrelsen på den så den blir synlig for brukeren, samt vi har lagt in en transisjons hastighet, som får denne til å ”skli” elegant ut når brukeren trykker på meny knappen. Denne ser slik ut:



Figur 19: Meny knapp trykket inn

5.4 De foskjellige sidene

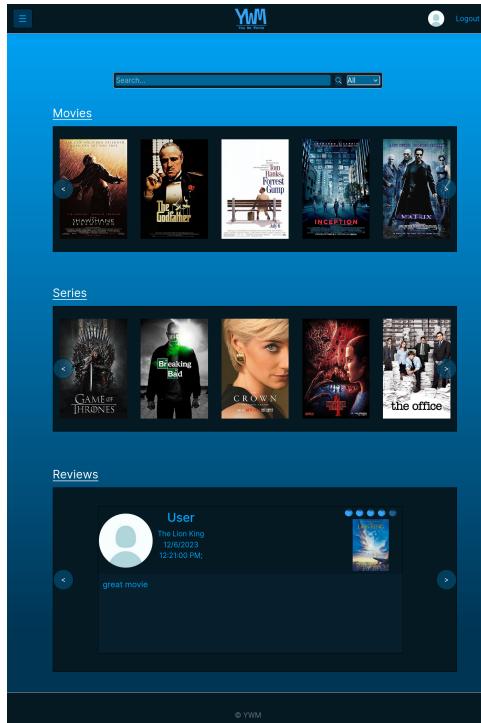
5.4.1 Hjemmesiden

På hjemmesiden 4.1.2 planlag vi å vise alle filmene i database, men vi valgte i implementasjonen å bare vise et par av filmene/seriene som eksisterer i databasen. Dette er bare for å få et overblikk over hvilke filmer/serier vi har. Dermed er det ikke nødvendig for oss å liste ut alle filmene/seriene vi har i databasen på denne siden, ettersom vi allerede har en side for akkurat det. På hjemmesiden derimot velger vi 25 filmer og 25 serier fra databasen, samt 10 reviews. Dette blir hentet fra databasen i home kontrolleren slik:

```
//Retrieve information from database:  
var movieContents = _dbContext.Contents.Where(c => c.IsSeries == false).Take(25).ToList();  
var seriesContents = _dbContext.Contents.Where(c => c.IsSeries == true).Take(25).ToList();  
var reviews = _dbContext.Reviews.Include(r => r.Content).Include(r => r ApplicationUser).Take(10).ToList();
```

Figur 20: Henter informasjon fra databasen

Når siden mottar disse listene med modeller, blir det loopet igjennom dem, og vist bildet av hver film/serie, samt hver review. Det blir vist 5 bilder om gangen, eller 1 review, og brukeren har mulighet til å trykke på piler til høyre eller venstre for å scrollle videre i listen. For å gjøre dette har vi festet en eventlistener til hver av knappene som endrer på transformasjonsegenskapene i stylingen til kontaineren som inneholder bildene og reviewene. Slik at posisjonen i listen blir forflyttet til høyre, eller venstre, med en fancy animasjon som indikerer dette for brukeren. Dette gjøres med en javascript funksjon som tar inn lengden på listen som skal vises, og utfører denne transformasjonen for hvert trykk. Hjemmsiden ble seendes slik ut:



Figur 21: Den implementerte hjemmesiden til You We Movie

I tillegg la vi inn redirects i tittelene over hver liste, slik at når brukeren trykker på disse, blir de sendt til de forventede sidene.

5.4.2 Content informasjon

Når en bruker trykker inn på en film/serie, blir de møtt med en side som viser all informasjon om denne filmen/serien. Denne siden har vi valgt å beholde ganske lik det vi tenkte fra 4.1.6. På denne siden kan de se imdb sin rating, lengden på filmen/serien, våre brukere sin rating, film direktør(er), utgivelsesdato, synopsis og tittel på filmen. Dette utfyller funksjonell krav ”må ha 15”. Siden all denne informasjonen blir hentet eksternt, er vi ikke alltid sikre på lengden til informasjonen. For eksempel, kan en film/serie ha flere enn bare en direktør, eller en veldig lang tittel og synopsis. For at dette ikke skal se rart ut på vår side, har vi valgt å legge på en automatisk customized scrollbar, hvis teksten begynner å overstige høyden på kontaineren de står i. For firefox brukere vil denne scrollbaren ikke vises på samme måte som i chrome, ettersom firefox har store begrensninger til modifisering av scrollbar.

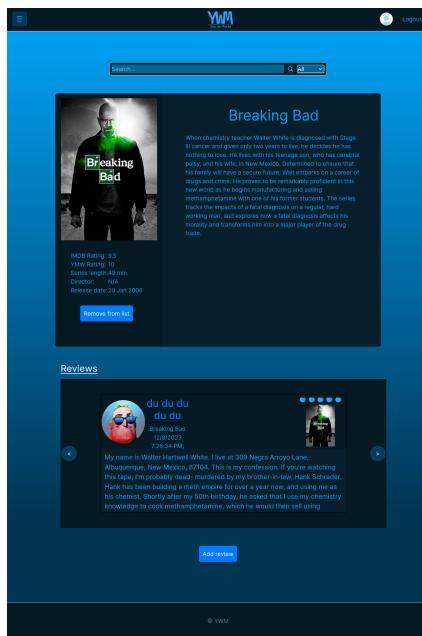
Størrelsen på bildene vi får fra OMDb har alltid samme størrelse, så vi har valgt å gjøre dette til en fixed size. I tillegg kan enkelte av verdiene være null verdier. Dette skaper

problemer når vi skal vise disse for brukeren. Dermed er vi nødt til å sjekke om variablene er av nullverdi før vi prøver å vise dem til brukeren. Dette gjør vi i HTML filen ved hjelp av razor syntax. Dette har vi gjort slik:

```
<td>@(Model.Director ?? "N/A")</td>
```

Figur 22: benytter C# kode i HTML for å sjekke variabler

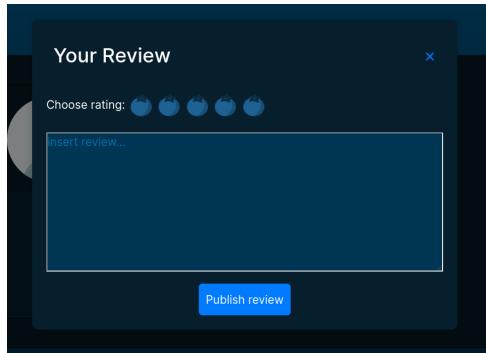
Dermed viser produktet vårt ”N/A” (not available), dersom en variabel er null. Dette er også gjort på de andre variablene i koden. Selve utseende på denne siden har vi lagd til å bli følgende:



Figur 23: Content informasjon layout

Rett under informasjonen har vi valgt å legge til en ”Add to list/Remove from list” knapp. For å vise denne knappen sjekker vi om brukeren er autentisert. Dersom de er det vises denne knappen med tilsvarende tekst basert på om brukeren har filmen/serien allerede i sin personlige liste eller ikke. Denne knappen kaller en funksjon som sjekker om filmen/serien er i nåværende innlogget bruker sin listen i database. Hvis den ikke er i listen blir den lagt til, og hvis den er i listen blir den fjernet. I tillegg bruker vi javascript til å endre på teksten i knappen, slik at vi ikke trenger å laste inn siden på nytt for å se endringene. Dette gjøres ved å bytte mellom de to tekstene for hvert trykk. Teksten vil alltid samsvarer med logikken, ettersom siden blir lastet inn med riktig tekst allerede i knappen. Kommunikasjonen til denne knappen vises i 38 og 39.

På bunnen viser vi reviews på samme måte som på hovedsiden. Under her har vi valgt å legge ”add review” knappen. Når du trykker på denne vises et popup vindu som lar deg skrive inn hva du ønsker å si om denne filmen, samt hvilke score du vil gi den. Denne kommunikasjonen vises i 36 og utfyller funksjonell krav ”må ha 1”.



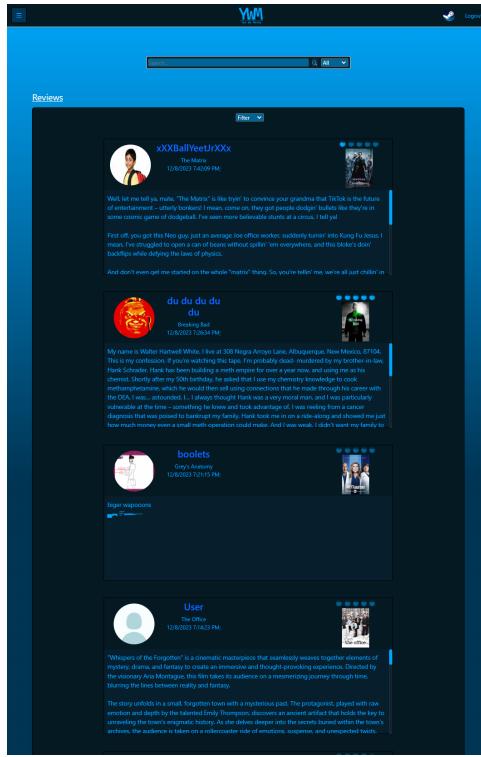
Figur 24: Popup som dukker opp når du trykker på add review

For å velge en score, har vi valgt å bruke blåbær til å visualisere scoren du gir en film/serie. Dette passer til det blå temaet til nettsiden vår, og brukere kan velge scoren med å trykke på ønsket blåbær. De har mulighet til å velge mellom 0-5 blåbær. Dersom brukeren ikke er autentisert vil ”add review” knappen vises som en ”Login to review” kapp. Når brukeren trykker på denne redirecter vi dem til login skjermen, der de må logge inn før de kan legge til en review. På grunn av tidsbegrensninger har vi, akkurat som med bildene, ikke noe moderering av reviews, annet enn en maks lengde på 5000 karakterer. Dette er noe vi ville implementert dersom vi hadde hatt mer tid.

5.4.3 Reviews

Når en bruker legger til en review gjennom add review knappen, blir det sendt en post request til review kontrolleren. Denne kontrolleren sjekker om brukeren allerede har en review registrert til samme film. Hvis brukeren allerede har en eksisterende review til den valgte filmen, vil den gamle reviewen bli oppdatert med den nye angitte informasjonen. Dersom en review ikke allerede eksisterer vil den bli lagt til i databasen. Når vi legger til en review må vi også oppdatere scoren til filmen som nettop ble rated av brukeren. Dette ta den samme kontrolleren seg av, ved å kjøre en sqlspørring som spør etter average rating score på alle reviewene som tilhører denne filmen. I tillegg bruker vi en rating skala fra 0-5. Hadde vi vist 5 (høyeste score), for brukeren ville dette sett lite ut når vi også viser imdb sin score, med skala 0-10. Dermed har vi valgt å gange vår score med 2, slik at vi får samme skala som imdb, og det er lettere å tolke for brukerene.

Når en bruker legger til en review blir de redirected til review siden, der de kan se sin egen review øverst på siden. På denne siden vil de også kunne se alle reviewene som ligger i databasen vår som planlagt i 4.1.7, og utfyller funksjonell krav ”må ha 2”. Vi valgte å bare ta med filtrering av reviews basert på opplastningsdato. Ettersom vi ikke hadde mer tid til å legge til flere filtre. Her vises alle reviewene til alle brukerne og filmene. I hver review kan man også se hvilke film det er til, hvem som har lagt den ut, når den er blir laget, scoren brukeren har gitt, visuelt bildet av hvilke film som blir reviewet og selve innholdet i reviewen. Ettersom alle Reviewene som ligger der skal være av samme format, bruker vi en partial view til å vise disse reviewene. Dersom en Review skulle bli for lang, har vi akkurat som i content info siden, lagt inn en customized scroll bar, som lar brukerne scrolle igjennom en review.



Figur 25: Endelig utseende av review siden

På denne siden har brukere mulighet til å slette enhver review de selv har lagt ut, kommunikasjonen til dette vises i 37. I tillegg har de mulighet til å filtrere reviewene etter nyeste eller eldste. Får å gjøre dette sender vi en HTTP GET request til review kontrolleren hver gang filteret blir endret på. Kontrolleren vil så utføre en spørring mot databasen får å få en sortert liste etter opplastningsdato, som enten vil være i stigende eller synkende rekkefølge.

Vi valgte også ikke implementere egen review side som planlagt i 4.1.8, ettersom vi begynte å få liten tid, samt brukere kan allerede se all informasjonen i siden som viser alle reviewene.

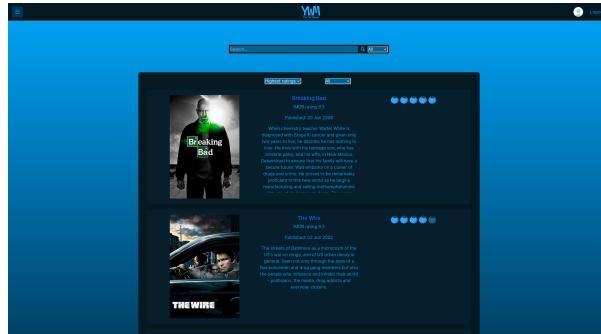
```
switch (filter)
{
    case "newest":
        reviewsQuery = reviewsQuery.OrderByDescending(r => r.Time);
        break;
    case "oldest":
        reviewsQuery = reviewsQuery.OrderBy(r => r.Time);
        break;
}
```

Figur 26: Switch case med de forskjellige filter valgene

Her har vi valgt å ikke ta i bruk JQuery, ettersom det ikke har noen betydelige fordeler i denne sammenhengen.

5.4.4 Filmer og serier

En annen side som vi også har brukt filtrering på, er film/serie siden. Denne siden har vi valgt å implementere slik:



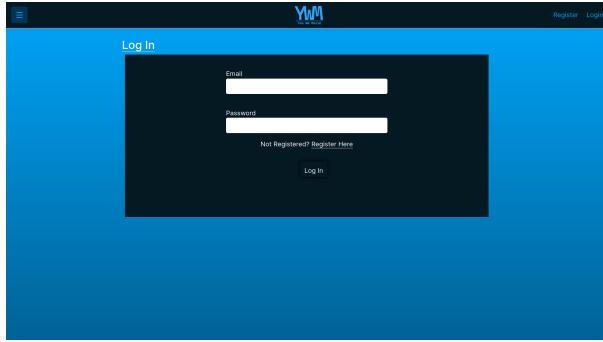
Figur 27: Side som viser filmer og serier, med average ymw blåbær score

Denne siden bruker vi for å vise alle filmene eller seriene vi oppbevarer i databasen som vi tenkte i 4.1.5, og utfyller funksjonell krav ”må ha 14”. Akkurat som med reviews blir hver film/serie vist ved bruk av et partial view, ettersom alle skal ha samme layout. På denne siden benytter vi oss av JQuery til å kunne sortere filmene/seriene som ønsket uten å laste inn siden på nytt [18]. Grunnen til at vi bruker JQuery i dette tilfelle, er at vi valgte å ha to filtreringsknapper. Uten JQuery vil filtrering med den ene knappen resette den andre. Derfor benytter vi JQuery til å utføre asynkrone HTTP GET request fra javascripten, slik at vi beholder de forrige valgene til brukeren. Dette tillater at brukeren kan sortere på sjangeren samtidig som de sorterer på utgivelsesdato. forespørselen blir sendt til en egen ”content” kontroller som tar seg av disse database spørringene basert på brukeren sitt filtrerings valg, og returnerer de filtrerte listene av typen: List<Content>. På denne siden skulle vi også inkludere en ”add to list” knapp. Dette valgte vi å ikke legge til, ettersom vi ikke hadde tid igjen, samt vi hadde allerede den samme knappen på selve informasjons siden.

Da hver film eller seire på denne siden er et eget partial view har vi tatt i bruk en partial view to string funksjon. Denne funksjonen konverterer partial viewn om til en string. Når vi deretter bruker filter funksjonen som filtrerer filmer og serier etter gitte valg vil da hver film eller serie bli gjort om til en string og bli sendt videre til vår javascript funksjon. På denne måten unngår vi at siden refresher etter hver querie. For å klare å lage partial view to string funksjonen har vi lest oss opp på microsoft dokumentasjonen om partial views [13] samt blitt inspirert av stack overflow [16].

5.4.5 logging in og registrering

Når vi implementerte login og register sidene, ble disse ganske lik det vi planlag i 4.1.9, og utfyller funksjonell krav ”må ha 5”. På login siden, var det ikke mye som trengte å endre på i backend delen, hvor kommunikasjonen som skjer under login er vist i 33. Her valgte vi å beholde den standard login funksjonen som fulgte med asp.net [2]. Derimot valgte vi bare å style denne i frontend delen, for å forsikre oss at den passet til vårt ønsket fargetema. Dermed ble login siden slik:



Figur 28: Login siden

Hadde vi hatt mer tid, ville vi implementert login funksjoner med eksterne autentisering systemer, men dette var noe vi ikke prioriterte i denne omgang [11]. For å kunne gjøre endringer på denne siden, samt registreringssiden måtte vi autogenerere disse filene ved hjelp av ”aspnet-codegenerator” [2]. Dette gjorde vi med kommandoene:

- Log in

```
dotnet aspnet-codegenerator identity -dc YouWeMovie -files
"Areas/Identity/Pages/Account/Login.cshtml"
```

- Register

```
dotnet aspnet-codegenerator identity -dc YouWeMovie -files
"Areas/Identity/Pages/Account/Register.cshtml"
```

I selve registreringssiden brukte vi også den standarde registreringsiden til asp.net, men her måtte vi legge til to nye felter som skulle bli lagret om brukeren. Disse to feltene var brukernavnet og profilbildet. Til bildet feltet implementerte vi en tilpasset valideringsfunksjon, som sjekket størrelsen på bildet. På den måten ville ikke brukere kunne laste opp uendelig mye data, og krasje databasen i samme prosess. I tillegg måtte vi være sikre på at filen brukeren lastet opp, faktisk var en bildefil. Til dette bruker vi javascript til å sjekke filtypen som blir lastet opp.

```
//custom function to validate the maximum size of the entered image
1 usage  A Sander Westøl
public static ValidationResult MaxFileSize(IFormFile file, ValidationContext context)
{
    return file is { Length: > 256_000 } ? // 250KB
        new ValidationResult("The file size must be less than 250KB.") : ValidationResult.Success;
}
```

Figur 29: Tilpasset bildevalidering for sjekk av størrelse

På lengden til brukernavnet satt vi på en begrensning på 50 karakterer, slik at brukere ikke skal kunne laste opp uendelige lange brukernavn, og igjen krasjer databasen. Dette brukernavnet er ikke unikt for en bruker. Ettersom emailen til en bruker allerede er unikt, har vi valgt å tillate flere brukere å benytte seg av det samme brukernavnet. Kommunikasjonen til denne siden er vist i 34 Denne siden stylet vi også på samme måte som login siden, slik at den passet til vårt produkt. Dermed fikk den slik utseende:

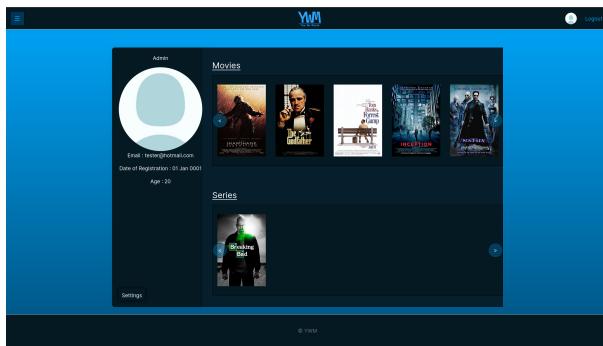


Figur 30: Registreringssiden

Som sagt ville vi gitt brukere mulighet til å registrere seg og logge inn gjennom eksterne autentiserings systemer [11]. På denne måten kan vi overlate lagringen av kritisk bruker-informasjon til de store eksterne systemene, og dermed slipper vi å tenke på dette. Vi har valgt å ikke prioritere dette begrunnet tidsbegrensninger, også på grunn av at asp.net bra standarder for kryptering av passord, som passer godt nok for vårt prosjekt [12]. Dette inkluderer både hashing og salting av passordene som kommer inn [10].

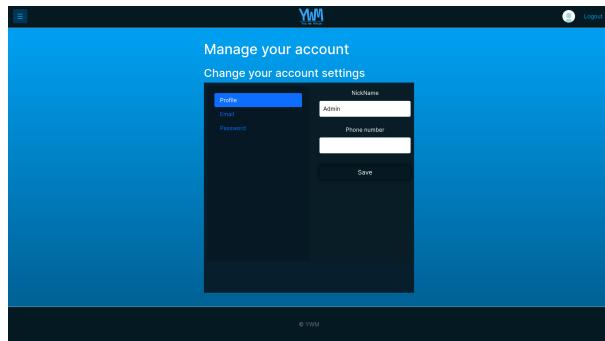
5.4.6 Profil siden

Når en bruker trykker på profilbildet sitt i toppen på høyre side, har vi satt dette til å redirecte dem til sin egen brukersiden, basert på id-en de har. I tillegg har de mulighet til trykke inn på andre sine profil sider, som ønsket i funksjonell krav ”kan ha 1”. Får å utføre dette har vi en egen kontroller for brukere, som tar inn bruker iden som skal sjekkes. Denne kontrolleren foretar en databasespørring mot vår database får å hente ut all informasjonen om brukere. Denne spørringen inkluderer også hver film/serie de har lagt til i sin personlige liste, som utfyller ”må ha 13”. Denne informasjonen legger vi i en viewbag om vi aksesserer fra viewen. Denne bruker siden ser slik ut:



Figur 31: profil siden

På denne siden bruker vi samme javascript funksjon som på hjemmesiden til å scroll gjennom filmene. På denne siden planlag vi egentlig å ha med venner og favoritt serier/filmer til brukeren som i 4.1.10. Dette valgte vi å ikke implementere ettersom vi hadde begrenset mengde med tid. Her har vi også valgt å legge til en settings knapp i venstre nedre hjørnet, som redirecter brukeren til en egen side der de kan legge til telefonnummer, eller endre brukernavn og passord. Den ser slik ut:



Figur 32: Bruker instillinger

Dette er en stylet versjon av den standard manage user siden som fulgte med i asp.net. Her måtte vi, akkurat som med login og register, også generere filene vi ikke hadde [2]. Dette gjorde vi slik:

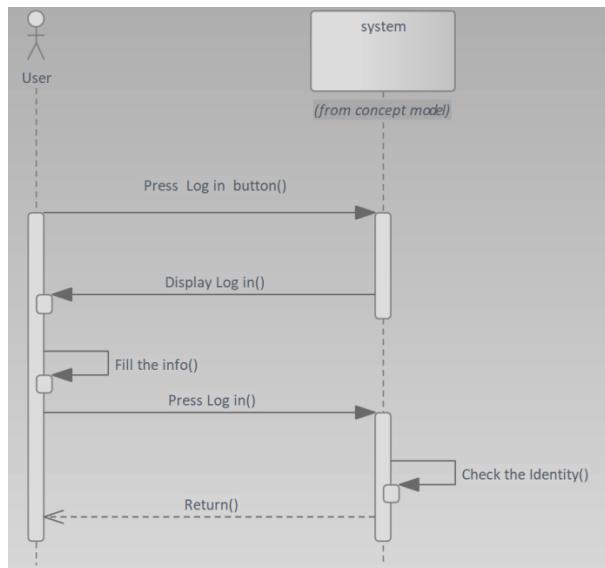
- Manage


```
dotnet aspnet-codegenerator identity -dc YouWeMovie -files
"Areas/Identity/Pages/Account/Manage/Index.cshtml"
```

5.5 Interaction diagrams

5.5.1 logging in

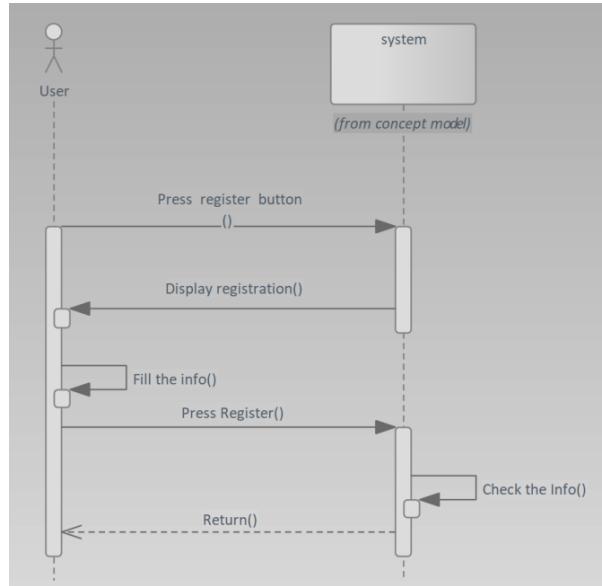
Dette interaksjons diagrammet representerer kommunikasjonen mellom en bruker og systemet vårt når brukeren skal logge inn, som oppfyller ”må ha 5”. For at brukeren skal kunne gjøre dette kommunikasjonen, må brukeren allerede ha en registrert bruker.



Figur 33: Login siden

5.5.2 Registrering

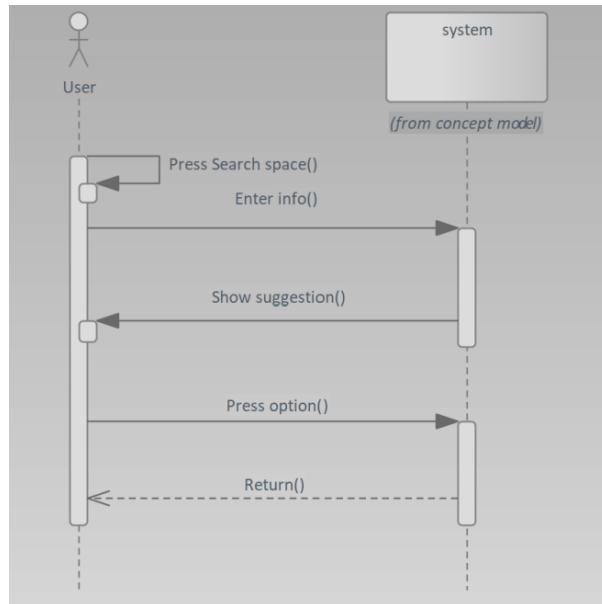
Dette interaksjons diagrammet representerer kommunikasjonen mellom en bruker og systemet vårt når brukeren skal registrere seg, som oppfyller ”må ha 4”.



Figur 34: Registreringssiden

5.5.3 Søking

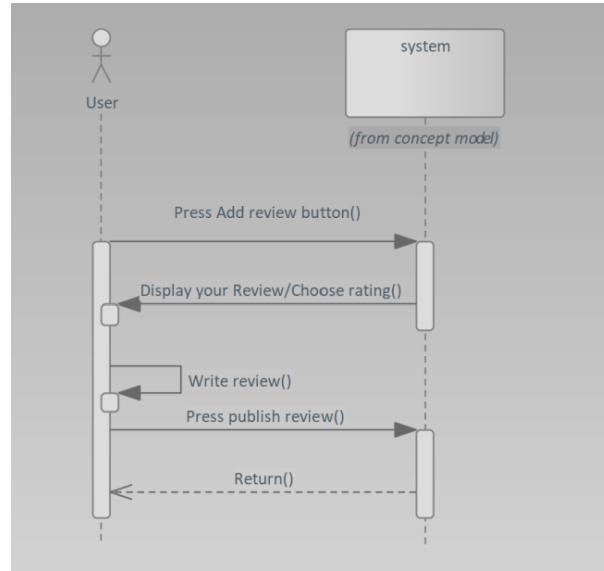
Dette interaksjons diagrammet representerer kommunikasjonen mellom en bruker og systemet vårt når brukeren skal søke etter filmer eller serier, som oppfyller ”må ha 7”.



Figur 35: Søking

5.5.4 Legg til review

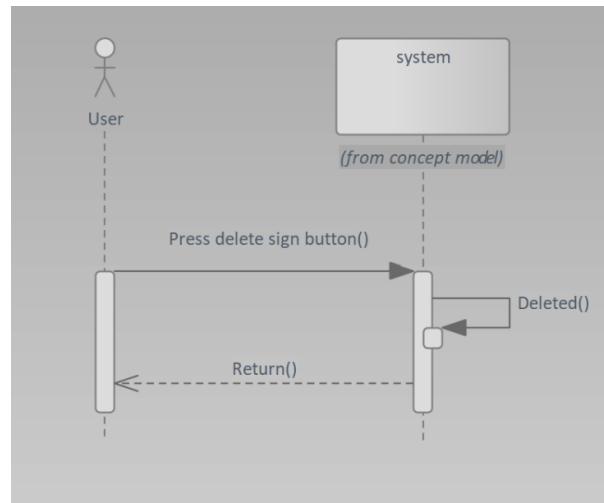
Dette interaksjons diagrammet representerer kommunikasjonen mellom en bruker og systemet vårt når brukeren velger å legge til en review om en film, som oppfyller "må ha 1".



Figur 36: Legg til review

5.5.5 Fjern review

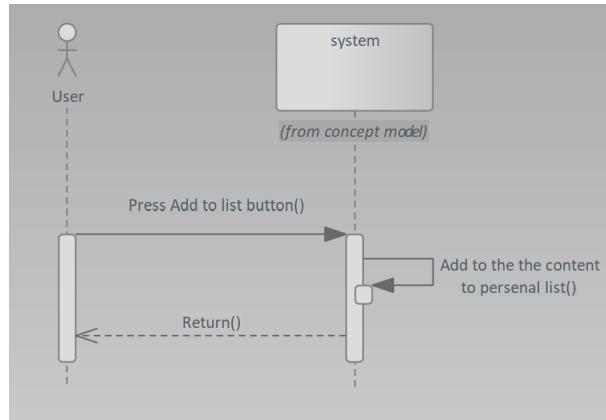
Dette interaksjons diagrammet representerer kommunikasjonen mellom en bruker og systemet vårt når brukeren ønsker å slette en av sine reviews, som oppfyller "må ha 12".



Figur 37: Fjern review

5.5.6 Legg til film/serie i personlig liste

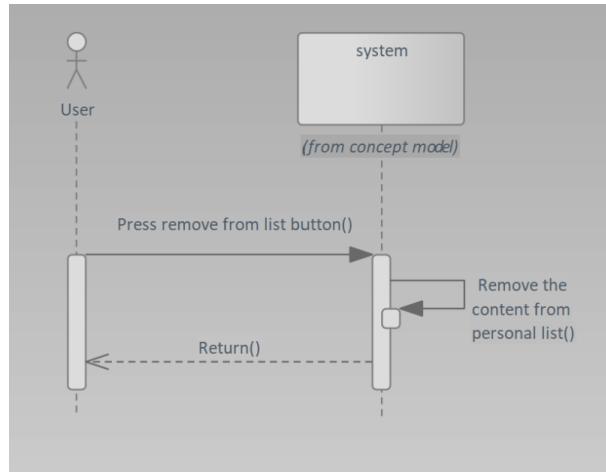
Dette interaksjons diagrammet representerer kommunikasjonen mellom en bruker og systemet vårt når brukeren ønsker å legge til en type content i sin personlige liste, som oppfyller ”må ha 8 og 9”.



Figur 38: Legg til filmer/serier i personlig liste

5.5.7 Fjern filmer/serier fra personlig liste

Dette interaksjons diagrammet representerer kommunikasjonen mellom en bruker og systemet vårt når brukeren ønsker å fjerne en type content fra sin personlige liste, som oppfyller ”må ha 10 og 11”.



Figur 39: Fjern filmer/serier fra personlig liste

6 Testing og validering

6.1 Innledning

Dette kapittelet handler om å teste vår løsning og vise om løsningen vår faktisk løser problemet vi har tatt oss for. Vi vil diskutere om hvordan vår foreslalte løsning har blitt testet for å sikre at kravene som vi lagde i kap 3 er nådd. Målet ved dette kapittelet er å vise at løsningen vår tar for seg hele eller det meste av problemet vårt. Vi har også prøvd å presentere en så objektiv vurdering av våre resultater og vårt arbeid som mulig.

6.2 Hvordan vil testingen foregå?

Testing av produkter kan gjøres på mange måter. En kan bruke kravtester, disse testene sjekker om funksjonene fungerer. Vi fokuserer da på å sjekke om produktet tilfredsstiller kravene som vi har laget i kapittel 3. Dette kan være funksjonelle krav, som er hva produktet skal gjøre og dens funksjoner, eller tekniske krav, som fokuserer på sikkerhet og ytelse. En kan også gjøre manuelle funksjonstester. Disse testene innebærer hva som blir testes, hvordan vi skal utføre testene og hva resultatet er. Manuelle funksjonstester vil bli utført manuelt av personer. Da sjekker en om funksjonaliteten funker og hvordan brukeropplevelsen er. En kan også inkludere testing for å finne uforutsette problemer og feil. Det finnes også automatisk testing der en lager kode som tester produktet vårt automatisk. Dette har vi ikke valgt å gjøre fordi...

Vi har heller valgt å gjøre en blanding av kravtester og manuelle funksjonstester. Dette er hvordan testingen vil foregå. Vi vil starte med det første kravet, sjekke om koden gjør det den skal. Videre starter vi programmet for å teste om koden gjør det den skal når det utføres i programmet. Etter dette skriver vi ned resultatet. Deretter går vi videre til neste krav, og fortsetter til vi er ferdig med alle kravene.

6.3 Resultat av testing

Under implementeringen har vi vært gode til å legge inn masse test data. Dette har gjort testfasen lettere for oss, da vi enkelt og greit kan teste produktet uten å måtte tenke på koden. Vi har prøvd å sammensveise kravene vi lagde i kap 4 med testene våres. På denne måten kan vi få en klar oversikt over hvilke krav som vi har nådd.

6.3.1 Funksjonelle tester

For å få en god oversikt over testene og resultatene har vi lagd all informasjonen i en tabell. Denne tabellen inneholder ID til testen, beskrivelse av testen, hvilken use case den tilhører, resultatet, dato testen foregikk, hvem som skrev koden, hvem som testet og til slutt en liten kommentar.

6.3.2 Beskrivelse av hver funksjonell test

I den første testen med test ID 1 har vi testet om brukere kan legge til reviews av både filmer og serier. Denne testen samsvarer til use case må ha 1. Denne testen ble godkjent, som betyr at registrerte brukere kan legge til reviews. Vi testet også hva som skjedde om en bruker ikke er registret. Da byttet knappen som sier add review om til log in to add review. Dette er akkurat den oppførselen vi hadde forventet, slik som diskutert i kapittel 5.4.2.

A	B	C	D	E	F	G	H
1	Funksjonelle tester						
2	Test case ID	Beskrivelse	Use case nr	Resultat	Test dato	Ansvarsfulle utvikler	Ansvarsfulle tester
3		1 Lage review	må ha 1	Pass	08.10.2023	Linor Ujkani	Erlend Tregde
4		2 Se review	må ha 2	Pass	08.10.2023	Linor Ujkani	Erlend Tregde
5		3 Rate content	må ha 3	Delvis riktig	08.10.2023	Linor Ujkani	Erlend Tregde
6		4 Registrere bruker	må ha 4	Pass	08.10.2023	Habteab Teame	Erlend Tregde
7		5 Login/logout	må ha 5-6	Pass	08.10.2023	Habteab Teame	Erlend Tregde
8		6 Bruke søkefeltet	må ha 7	Pass	08.10.2023	Sander Wesstøl	Erlend Tregde
9		7 Legg til filmer/serier i min liste	må ha 8-9	Pass	08.10.2023	Sander Wesstøl	Erlend Tregde
10		8 Fjerne filmer/serier fra min liste	må ha 10-11	Pass	08.10.2023	Sander Wesstøl	Erlend Tregde
11		9 Slette review	må ha 12	Pass	08.10.2023	Linor Ujkani	Erlend Tregde
12		10 Se liste over dine filmer/serier	må ha 13	Pass	08.10.2023	Luka Vulovic	Erlend Tregde
13		11 Få informasjon om alle filmer/serier	må ha 14	Pass	08.10.2023	Erlend Tregde	Erlend Tregde
14		12 Få informasjon om en enkelt film/serie	må ha 15	Pass	08.10.2023	Sander Wesstøl	Erlend Tregde
15		13 Få informasjon om en enkel review	må ha 16	Failed	08.10.2023	Linor Ujkani	Erlend Tregde
16		14 Endre passord	må ha 17	Pass	08.10.2023	Habteab Teame	Erlend Tregde
17		15 Knytte mobilnummer til bruker	må ha 18	Pass	08.10.2023	Habteab Teame	Erlend Tregde
18		16 Sortere filmer/serier etter kriterier	bør ha 1	Pass	08.10.2023	Erlend Tregde	Erlend Tregde
19		17 Sortere reviews etter kriterier	bør ha 2	Pass	08.10.2023	Linor Ujkani	Erlend Tregde
20		18 Se på andres profiler	kan ha 1	Pass	08.10.2023	Sander Wesstøl	Erlend Tregde
21		19 Legge til nye filmer i databasen som en bruker	kan ha 2	Failed	08.10.2023		Erlend Tregde
22		20 Gi tilbakemelding og rapportere brukere	kan ha 3	Failed	08.10.2023		Erlend Tregde
23		21 Få forslag på content basert på en algoritme	kan ha 4	Failed	08.10.2023		Erlend Tregde
24		22 Få nyheter om content	kan ha 5	Failed	08.10.2023		Erlend Tregde
25		23 Få informasjon om når filmer kommer på kino	kan ha 6	Failed	08.10.2023		Erlend Tregde
26		24 Legge til venner	kan ha 7	Failed	08.10.2023		Erlend Tregde

Figur 40: Funksjonelle tester

I den andre testen med ID 2 har vi testet om brukere kan se andre brukeres sine reviews. Denne testen samsvarer til use case må ha 2. Denne testen ble godkjent som betyr at en kan se andre brukeres sine reviews. Reviews kan bli sett i hjemmesiden og i Reviews siden. Dette er akkuratt den oppførselen vi forventet, slik som diskutert under kapittel 5.4.3

I den tredje testen med ID 3 har vi testet om en bruker kan legge til rating av filmer og serier. Denne testen samsvarer til use case må ha 3. Denne testen ble delvis godkjent. Dette er fordi vi bare kan legge til rating ved å legge inn et review. Vi hadde planlagt under designfasen å kunne legge til review i filmer og serie sidene som diskutert i kapittel 4.1.5, men på grunn av tidspress fikk vi ikke tid til dette.

I den fjerde testen med ID 4 har testet om en bruker kan registrere seg. Denne testen refererer til use case må ha 4. Når man registerer seg skriver en inn personlige opplysninger og man kan legge til et profilbilde. Denne testen ble godkjent som betyr at brukere kan registrere seg. Dette er den oppførselen vi forventet siden vi brukte den standariserte registreringsiden til asp.net som diskutert under kapittel 5.4.5

I den femte testen med ID 5 har vi testet om en bruker kan logge seg inn og ut. Denne testen refererer til use case må ha 5 og 6. For å logge inn trenger en bare skrive inn email og passord. Denne testen ble godkjent som betyr at brukere kan logge seg inn og ut av nettsiden. Som forventet kan en også se profilbildet sitt i headeren. Implementasjonen av dette er diskutert under kapittel 5.4.5

I den sjette testen med ID 6 har vi testet om en bruker kan søke i søkerfeltet. Denne testen refererer til use case må ha 7. Denne testen ble godkjent som vil si at brukere kan søke opp filmer, serier og andre brukere. En ting å merke seg er at søkerfeltet bare oppdateres hver andre karakter, men det er slik oppførsel vi forventet ifra implementasjonen som ble diskutert i kapittel 5.3.1

I den syvende testen med ID 7 har vi testet om en bruker kan legge til filmer og serier i sin liste. Denne testen refererer til use case må ha 8 og 9. Denne testen ble godkjent som betyr at registrerte brukere kan legge til filmer og serier i sin liste. Dette kan du gjøre ved å trykke inn på en spesifikk film eller serie, så trykke på add to list knappen. I denne testen sjekket vi også om brukere som ikke er registrerte kunne bruke add to list knappen. Det kunne de ikke. Dette er forventet oppførsel fra programmet som diskutert

under implementasjonen i kapittel 5.4.2

I den åttende testen med ID 8 har vi testet om en bruker kan fjerne filmer og serier fra sin liste. Denne testen refererer til use case må ha 10 og 11. Denne testen ble godkjent som vil si at registrerte brukere kan fjerne filmer og serier fra sin liste. Brukere som ikke er registrerte kan naturligvis ikke se denne knappen, da de ikke har mulighet til å legge til filmer og serier i sin liste. Dette er forventet oppførsel fra systemet som er diskutert i kapittel 5.4.2.

I den niende testen med ID 9 har vi testet om en bruker kan slette sine reviews. Denne testen refererer til use case må ha 12. Denne testen ble godkjent som vil si at registrere brukere som har lagt ut et review kan slette sine egne reviews. Testen gikk også ut på å teste om andre brukere kunne slette andre sine reviews, dette gikk ikke. Dette er forventet oppførsel som er nøyde diskutert i kapittel 5.4.3.

I den tiende testen med ID 10 har vi testet om en bruker kan se en liste over sine likte filmer og serier. Dette henviser til use case må ha 13. Denne testen ble godkjent som vil si at en registrert bruker kan se sine filmer og serier i sin profil side. Testen gikk også ut på å legge til en film eller serie, for så å gå tilbake til profil siden for å se om listen ble oppdatert. Det ble den. I tillegg sjekket vi om ikke registrerte brukere hadde tilgang til en slik side, det hadde de ikke. Dette er forventet oppførsel fra implementasjonen som diskutert i kapittel 5.4.6

I den ellevte testen med ID 11 sjekker vi om brukeren kan få informasjon om alle filmer og om alle serier. Denne testen refererer til use case må ha 14. Denne testen ble godkjent som vil si at vi har en side som viser alle filmene, og en side som viser alle seriene. Implementasjonen av disse sidene er diskutert i kapittel 5.4.4

I den tolvte testen med ID 12 sjekker vi om brukeren kan få informasjon om en enkelt film eller serie. Denne testen refererer til use case må ha 15. Denne testen ble godkjent som vil si at hver film og serie har sin egen side med informasjon. Dette er forventet ifra implementerings stadiet som diskutert i kapittel 5.4.2

I den trettende testen med ID 13 sjekker vi om brukeren kan få informasjon om en enkel review. Denne testen refererer til use case må ha 16. Denne testen ble ikke godkjent. Dette er fordi vi ikke fikk tid til å implementere dette.

I den fjortende testen med ID 14 sjekker vi om en registrert bruker kan endre passord. Denne testen henviser til use case må ha 17. Denne testen ble godkjent som vil si at registrerte brukere kan endre passord. Dette er forventet oppførsel som diskutert i kapittel 5.4.6.

I den femtende testen med ID 15 sjekker vi om registrerte brukere kan legge til mobilnummer til sin bruker. Dette henviser til use case må ha 18. Denne testen ble godkjent som vil si at en bruker kan legge til mobilnummeret sitt til brukeren sin. Dette testet vi med å legge til mobilnummer i profil innstillinger, så sjekket vi databasen om mobilnummer hadde blitt lagt inn. Det ble den. Implementasjonen av dette er diskutert under kapittel 5.4.6.

I den seistene testen med Id 16 sjekker vi om filmer og serier kan bli sortert etter ulike kriterier. Dette henviser til use case bør ha 1. Denne testen ble godkjent som vil si at vi kan sortere filmer og serier etter ulike kriterier. I tillegg refresher ikke siden seg når vi sorterer. Dette er akuratt det vi forventet. Implementasjonen av dette er diskutert i kapittel 5.4.4

I den syttende testen med ID 17 sjekker vi om reviews kan bli sortert etter ulike kriterie. Dette henviser til use case bør ha 2. Denne testen ble godkjent som vil si at reviews kan bli sortert etter ulike kriterier. Noe vi merket oss under testing var at siden refresher seg etter vi sorterer, dette er noe vi kunne forbedret til senere. Implementasjonen av dette er nøyere diskutert under kapittel 5.4.3.

I den attende testen med ID 18 sjekker vi om brukere kan se andres profiler. Dette henviser til use case kan ha 1. Denne testen ble godkjent som betyr at brukere kan se andres profiler. Det funker bra, men et problem er at en må søke opp brukeren for besøke profilen, istendet for å kunne trykke inn på brukernavnet. Dette er forventet oppførsel og nøyere diskutert under kap 5.4.6

Resten av testene det vil si test ID 19-24, er tester som ikke er blitt godkjent. Dette er fordi vi ikke har fått tid til å implementere dette. Noen av testene inneholder funksjoner som vi gjerne ville hatt med, slik som ett vennesystem dvs testen med ID 24. Denne testen refererer til use case kan ha 7.

6.3.3 Samlet test resultater for funksjonelle tester

Til sammen har vi kjørt alle våre funksjonelle tester. Dette var total 24 tester, der 16 av dem ble godkjent. 7 av testene ble ikke godkjent og 1 ble delvis godkjent. Ut ifra use casene våre har vi fått ett godt resultat. Det er bare en av use casene som vi må ha som vi ikke har klart å fullføre. Dette er nemlig at en bruker skal kunne få informasjon om hver enkelt review, use case må ha 16. Resten av testene som ikke har blitt godkjent er fordi vi ikke har hatt tid til å implementere det, dette er mer diskutert under kapittel 5

A	B
1	Funksjonelle tester sammendrag
2	Totale tester
3	24
4	Kjørte tester
5	24
6	Passed
7	16
8	Failed
9	7
10	Delvis
11	1

Figur 41: Funksjonelle tester sammendrag

6.3.4 Tekniske krav

For å teste de tekniske kravene har vi brukt samme fremgangsmåte som ved de funksjonelle kravene. Det vil si at vi bruker den samme tabellen som sist som er bestående av ID til testen, beskrivelse av testen, hvilken use case det gjelder, dato, hvem som skrev koden, testeren og en kommentar.

A	B	C	D	E	F	G	H
1	Tekniske tester						
2	Test case ID	Beskrivelse	Use case nr	Resultat	Test dato	Ansvarsfulle utvikler	Ansvarsfulle tester
3	1	Passord må sikres i databasen	må ha 1	Pass	09.12.2023	Habteab Team	Erlend Tregde
4	2	Nettsiden må fungere for normale skjermer	må ha 2	Pass	09.12.2023	Samtlige	Erlend Tregde
5	3	Skal fungere for chrome og firefox	må ha 3	Pass	09.12.2023	Samtlige	Erlend Tregde
6	4	Skal fungere for linux og windows	må ha 4	Pass	09.12.2023	Samtlige	Erlend Tregde
7	5	Kode skal skrives oversiktlig	må ha 5	Pass	09.12.2023	Samtlige	Erlend Tregde
8	6	Koden skal skrives ved hjelp av gitt kodespår	må ha 6	Pass	09.12.2023	Samtlige	Erlend Tregde
9	7	Jetbrains rider skal brukes under utvikling	må ha 7	Pass	09.12.2023	Samtlige	Erlend Tregde
10	8	Tofaktor autitisering	bør ha 1	Failed	09.12.2023	Habteab Team	Erlend Tregde
11	9	Nettsiden skal fungere for nettrett og mobil	kan ha 1	Ikke testet	09.12.2023		Erlend Tregde
12	10	Persondata må slettes når profilen slettes	kan ha 2	Ikke testet	09.12.2023		Erlend Tregde

Figur 42: Teknisk tester

6.3.5 Beskrivelse av hver teknisk test

I den første testen med test ID 1 har vi testet om passordene blir sikret i databasen ved bruk av hasing. Denne testen refererer til use case må ha 1. Denne testen ble godkjent. Vi testet dette ved å registrere en ny bruker, så sjekke databasen om hvor passordet er lagret. Der så vi at passordet hadde blitt omgjort ved hasing.

I den andre testen med test ID 2 har vi testet om nettsiden funker for normale skjermstørrelser. Denne testen refererer til use case må ha 2. Denne testen ble godkjent. Vi testet ved normale skjermopløsninger som 1920x1080 og 1266x768. I tillegg brukte jeg ctrl + og - for å sjekke med ulike skaleringer. Dette funget bra.

I den tredje testen med test ID 3 har vi testet om nettsiden funker både i chrome og firefox. Denne testen refererer til use case må ha 3. Denne testen passerte. Men en ting vi merker i firefox var at scrollbaren sin css ikke funget i firefox. Dette er noe vi hadde fixet hvis vi hadde hatt mere tid i prosjektet.

I den fjerde tesren med test ID 4 har vi testen om nettsiden funker for windowsbrukere og linux brukere. Denne testen referer til use case må ha 4. Denne testen passerte. Vi utførte testen ved å starte programmet på et linux operativsystem så sjekket om funksjonalitetene funget, samme gjorde vi på et windows operativsystem.

I den femte, sjette og syvende testen med ID 5, 6 og 7 har vi testet ulike ting om koden. Dette refererer til use case må ha 5, 6, 7. Disse testene gikk ut på at koden skal skrives oversiktlig, språkene som brukes er C sharp, javascript, html og css. I tillegg skal rider brukes under utviklingene. Alle disse testene ble godkjent. Disse testene ble gjort ved observeringer i programmet.

I den åttende testen med test ID 8 har vi testet om nettsiden har tofaktor autitisering. Denne refererer til use case bør ha 1. Denne teste ble ikke godkjent, dette er fordi vi ikke har fått tid til å teste dette.

I den niende testen med test ID 9 skulle vi ha testet om nettsiden funker for nettbrett og mobil. Denne testen refererer til use case kan ha 1 Denne testen ble ikke gjennomført, da dette ikke har vært en prioritert i vårt produkt.

I den tiende testen med test ID 10 skulle vi ha testen om persondata slettes når brukeren slettes. Denne testen refererer til use case kan ha 2. Denne testen ble ikke gjennomført da vi ikke har noen måte at brukeren kan slette sin profil.

6.3.6 Samlet test resultater for tekniske tester

Til sammen har vi kjørt 8 av 10 tekniske tester. Vi endte opp med at 7 av dem ble godkjente. 1 av testene feilet. Da dette tilhører en use case vi burde ha med i programmet gjør det ikke så mye at denne ikke ble godkjent. Til slutt har vi 2 tester vi ikke har kjørt. Dette er fordi de ikke har hatt høy prioritet igjennom vårt produkt og har derfor ikke blitt implementert.

6.4 Konklusjon

Ut ifra alle testen vi har gjort har vi passert 23 av 34 tester. Dette er et resultat vi er fornøyde med. Vi har klart å få godkjent alle må ha use casene borsett fra en. Samtidig er det en del tester vi ikke har fått godkjent som vi gjerne ville ha klart. For videre utvikling hadde vi hatt lyst til å klare å få disse testene godkjent og. Utenom dette er tesingen

	A	B	C
1	Funksjonelle tester sammendrag		
2	Totale tester	10	
3	Kjørte tester	8	
4	Passed	7	
5	Failed	1	
6	Ikke testet	2	
7			

Figur 43: Teknisk test resultater

velykket og vi har fått et godt overblikk over hvilke use cases som ble laget i kapittel 3 som er fullført.

7 Diskusjon

7.1 Prosess

7.1.1 Planlegging

Helt i begynnelsen av planleggingen så brukte vi et Gantt diagram til å legge opp ulike prosesser som skulle bli fullført iløpet av prosjektet, dette gjorde det mer konkret og enklere for oss å holde styr på når ting skulle bli ferdig og når vi skulle begynne med nye faser. Etter en stund med diskusjon i starten fant vi ut av produktet vi skulle produsere og dens premisser, samt diskuterte produktet sine hovedfunksjoner og layout. For layout brukte vi Figma til å skissere hvordan alle de ulike sidene skulle se ut, det gjorde slik at vi hadde et bedre bilde av hvordan nettsiden skulle se ut.

7.1.2 Prioritering

I starten så var vi veldig dårlig til å skrive timer inn på Jira, men begynte å bli enklere etterhvert når vi hadde egen dedikerte git branches på prosjektet (under utviklingsfasen), det ble mye mer oversiktlig med timeskriving og hva hver av gruppemedlemmene holdt på med. Vår prioritering gjennom hele prosjektet vårt lå i at vi skulle ligge godt an i følge Gantt diagrammet vårt, nokså ofte lå vi ganske godt an andre ganger ikke så godt gitt andre prioriteter i studiet.

7.1.3 Git

Git hadde vi tidligere lært å bruke i veldig spesifikke tilfeller i emnene IKT-101 og IKT-103, men hadde ikke fullstendig forståelse av operasjonene vi tilførte i git bash. Så vi la det som mål å lære oss Git branching gjennom <https://learngitbranching.js.org/> samt ressurser på canvas. Når det gjaldt praktisk bruk av Git så hadde vi en del problemer med merge konflikter, som var veldig nytt for oss, men etterhvert lærte vi at så lenge vi delte prosjektet i mindre deler og oppdaterte branchen ofte nok så var det enklere å ta seg av.

7.1.4 Hvordan har samarbeidet vært?

Dette prosjektet er det første prosjektet i denne størrelsen som de fleste av oss har samarbeidet på, så vi forventet utfordringer gjennom hele prosjektet. Som av og til hendte gjennom uenigheter, men dette er gitt når vi er så mange på en gruppe og alle har ulike fokus på hva som er viktig. Problemer med kommunikasjon skjedde for det meste når vi var for fokuserte på eget arbeid og la ikke nok oppmerksomhet til andre sitt arbeid, uenigheter oppstod av og til når vi planla på implementasjonen, men som oftest ble diskutert i møtene og til slutt ble vi enige og hadde et fellesyn for hva vi trengte. Igjen er det nokså mange ting som vi kunne ønske vi fikk til, men er fortsatt fornøyd med sluttproduktet.

7.2 Produkt

Måloppnåelse. Feil og mangler. I sluttproduktet har vi oppnådd de viktigste funksjonene som det å kunne se gjennom film/serie, legge til en anmeldelse, ha en personalisert bruker som andre kan se og noen mindre relevante funksjoner. Selv med alle disse hovedfunksjonene så har vi også en god del funksjoner vi hadde ønsket å ha med, som for eksempel: en diskusjons page, funksjonalitet til å legge til venner og muligheten til å chatte med hverandre. Disse funksjonene er ikke hovedfunksjoner, men er funksjoner som hadde gjort produktet mer interaktivt og interessant.

7.3 Implementasjon

I dette delkapittelet diskuterer vi hva som er bra og ikke så bra med vår implementasjon. Vi diskuterer om det er noen ting som bør skrives om. Vi kommer også innom hva som vi er fordøyde med og hvorfor vi er fornøyde med dette.

Implementasjonen synes vi er ganske bra, den har de fleste viktige elementene som vi hadde diskutert, men manglet fortsatt de frivillige implementasjonene som nevnt tidligere. Det som er gjort bra i implementasjonen er hvor brukervennlig og minimalistisk programmet. Det er enkelt å navigere med hjelp av searchbar og vår navigasjons bar. Vi som gruppe synes samarbeidet har vært veldig bra, at vi klarte å sette oss sammen og lage noe som vi ble fornøyde med var en ganske god følelse å dele.

7.4 Utfordringer

Vi hadde en del Problemer med egne oppgaver, men som oftest lå det nokså mange nettressurser som fungerte som en løsning for ting vi aldri hadde lært å implementere før, ofte ble også det vi jobbet med for spesifikt til å kunne finne inspirasjon til, dette kunne eventuelt ha forårsaket mangel på noen av de implementasjonene som ikke ble noe av.

I begynnelsen av prosjektet var det nokså vanskelig å komme på en unik Ide som vi kunne utføre som vi samt var interessert i, vi hadde veldig mange forslag, samtidig også uenigheter om disse forslagene enten om de ikke var interessante eller litt for krevende for oss å få til, som ledet oss til denne ideen som vi var veldig fornøyd med siden vi synes den var interessant og noe krevende som vi fortsatt kan få til.

i starten av implementasjonen hadde vi diskusjoner om hvordan vi skulle legge til filer med all informasjon i vår applikasjon. Første løsningen vår var å bruke API fra en tredjeparts nettside til å liste opp informasjonen til filmene, men vi fant senere ut at det skulle være vanskelig å implementere add to list funksjonen og review funksjonen, etterhvert fant vi ut at det hadde vært bedre å ha alt i en database slik at vi kan bruke filmens/seriens ID til å kunne enklere implementere add to list funksjonen og review funksjonen ved å bruke dens id som en fremmednøkkel til å identifisere. Denne løsningen syns vi var optimalt gitt våre forhold, andre film bloggsider som IMDb bruker sin egen database mest sannsynlig for samme begrunnelser som oss.

7.5 Videre arbeid

Som begrunnet i tidligere kapittel, så har vi ikke klart å implementere et vennskaps system, et kommunikasjons system eller som nevnt testings kapittelet så har vi funksjonell mangel på: bruker skal kunne legge til nye filer, få en algoritme med forslag for brukeren, få nyheter og info om når filer blir vist på kino. Disse funksjonene er ikke påbegynt fordi vi ikke så det som mulig gitt tiden og ressursene vi hadde, alt her ble diskutert, men vi satset på konsise hovedfunksjoner istedenfor.

Implementering av disse tjenestene vil ikke hjelpe til med hoved premisset til You We Movie, men det hadde gjort det til et mer flytende og interaktivt system for brukeren. Å ha lite funksjonalitet kan uinteressere brukeren av programmet og vil lede til å ha færre tilhengere, med andre mer funksjonelle film forumsider som ligger der ute. Hadde det vært mer dedikert tid for kun dette prosjektet så skulle det å implementere alle ideene våre vært mer aktuelt.

8 Konklusjon

Vårt prosjekt definerer vi som veldig godt, siden vi har kommet frem til å ha implementert alt av viktige funksjoner, men hadde vi hatt mer dedikert tid for dette arbeidet så hadde vi fått til et veldig flytende program med flere funksjoner. Løsningen vår var var for oss å holde på med det viktige og ikke bli for fokusert på ting som ikke hadde hatt nytte i programmet. Vår applikasjon i hensyn til andre film forumsider kan ikke anses som rivaler, gitt at de har flere tiår med utvikling, forbedring og et veldig stort utvikler team. En bruker av programmet vårt vil få ganske overordnet informasjon om filmer, serier og anmeldelser, samtidig vil den kunne personalisere siden sin med ulik film-/serie liste og profil info. Til neste gang vil vi ha en bedre forståelse av hvordan et slikt prosjekt må bli prioritert og tidsbruk av ulike faser i utviklingen. Til neste gang hadde det vært en god ide å se på eksisterene løsninger til problemer og ikke bruke lang tid på noe som allerede eksisterer.

Referanser

- [1] Christian Auby. *07. Entity Framework basics*. Forelesning i canvas, IKT201. Institutt for Teknologi og bærekraft, Universitetet i Agder]. Grimstad. 2023.
- [2] Christian Auby. *09. Identity Framework*. Forelesning i canvas, IKT201. Institutt for Teknologi og bærekraft, Universitetet i Agder]. Grimstad. 2023.
- [3] Christian Auby. *Git in a team environment*. Forelesning i canvas, IKT201. Institutt for Teknologi og bærekraft, Universitetet i Agder]. Grimstad. 2023.
- [4] Ray Ferrell. *SQLite Locks: Types and How to Manage Them*. <https://sqldocs.org/sqlite/sqlite-locks/>. Accessed: 2023-12-10. 2023.
- [5] Kevin Ferris og Sonya Zhang. «A Framework for Selecting and Optimizing Color Scheme in Web Design». I: *2016 49th Hawaii International Conference on System Sciences (HICSS)*. 2016, s. 532–541. doi: 10.1109/HICSS.2016.73.
- [6] Sibsankar Haldar. *Inside SQLite*. Sebastopol, Calif.: O'Reilly, 2007.
- [7] *Internet Movie Database*. Accessed on 1. desember 2024. URL: <https://www.imdb.com/>.
- [8] Jay A. Kreibich. *Using SQLite*. O'Reilly Media, Inc., 2010. ISBN: 9780596521189.
- [9] Ivan Marsic. *Software Engineering*. Rutgers University, Department of Electrical and Computer Engineering. 2012.
- [10] Microsoft. *Hash passwords in ASP.NET Core*. <https://learn.microsoft.com/en-us/aspnet/core/security/data-protection/consumer-apis/password-hashing?view=aspnetcore-8.0>. Accessed: 2023-12-10. 2022.
- [11] Microsoft. *Microsoft Account external login setup with ASP.NET Core*. <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/social/microsoft-logins?view=aspnetcore-8.0>. Accessed: 2023-12-10. 2022.
- [12] Microsoft. *Overview of encryption, digital signatures, and hash algorithms in .NET*. <https://learn.microsoft.com/en-us/dotnet/standard/security/cryptographic-services>. Accessed: 2023-12-10. 2022.
- [13] Microsoft. *Partial views in ASP.NET Core*. <https://learn.microsoft.com/en-us/aspnet/core/mvc/views/partial?view=aspnetcore-8.0>. Accessed: 2023-12-10. 2023.
- [14] *Moscow Prioritization - ProductPlan Glossary*. <https://www.productplan.com/glossary/moscow-prioritization/>. Accessed last: 1. desember 2024.
- [15] *OMDb API - The Open Movie Database*. <http://www.omdbapi.com/>. Accessed: 10. november 2023.
- [16] Stack overflow. *Return View as String in .NET Core*. <https://stackoverflow.com/questions/40912375/return-view-as-string-in-net-core>. Last Accessed: 2023-12-10. 2017.
- [17] *Overviwe of ASP.NET core MVC*. <https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-7.0>. Accessed: 1. desember 2024.
- [18] *Post Data without Whole Postback*. <https://www.c-sharpcorner.com/UploadFile/0c1bb2/post-data-without-whole-postback/>. Accessed: 9. desember 2023. 2015.
- [19] *Rotten Tomatoes*. URL: <https://wwwrottentomatoes.com/>.
- [20] Russell Sears, Catharine van Ingen og Jim Gray. *To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem*. Tekn. rapp. MSR-TR-2006-45. Microsoft Research, 2006.
- [21] *SQLite Is Faster Than The Filesystem*. <https://www.sqlite.org/fasterthanfs.html>. Accessed: 9. desember 2023.

A Appendix

A.1 Individuell rapport

A.1.1 Linor Ujkani:

Linor_arbeid.pdf

A.1.2 Erlend Tregde:

Erlend_arbeid.pdf

A.1.3 Sander Wesstøl:

Sander_arbeid.pdf

A.1.4 Habteab Teame:

Habteab_arbeid.pdf

A.1.5 Luka Vulovic:

Luka_arbeid.pdf

A.2 Kanban

A.2.1 Done

Kanban_Done.pdf

A.2.2 Unresolved

Kanban_Unresolved.pdf

A.3 Gruppe kontrakt og produkt visjon

groupandprod.pdf

A.4 Gitshortlog

git.log

A.5 Møtereferat

Uke_42-49.pdf

A.6 Presentasjonsvideo

IKT201_PresentasjonsVideo.mp4

A.7 Produkt info

ProductInfo.pdf

A.8 Sprints

Jira1spring.pdf
Jira2spring.pdf
Jira3spring.pdf
Jira4spring.pdf
Jira5spring.pdf
Jira6spring.pdf
Jira7spring.pdf
Jira8spring.pdf
Jira9spring.pdf
Jira2springNotCompleted.pdf
Jira3springNotCompleted.pdf
Jira4springNotCompleted.pdf
Jira5springNotCompleted.pdf
Jira6springNotCompleted.pdf
Jira7springNotCompleted.pdf
Jira8springNotCompleted.pdf

A.9 Timesheets

Tempo _- _Jira.pdf