UNIVERSITETET I AGDER

Forside

IS-304: 2018

Tittel:

Emnekode	IS-304
Emnenavn	Bacheloroppgave i informasjonssystemer
Emneansvarlig	Hallgeir Nilsen
Veileder	Even Åby Larsen
Oppdragsgiver	NorgesEnergi

Studenter:

Etternavn	**Fornavn**
Børresen	Benjamin
Fosseli	Marius
Slagnes	Eirik

Jeg/vi bekrefter at vi ikke siterer eller på annen måte brukes andres arbeid uten at dette er oppgitt, og at alle referanser er oppgitt i litteraturlisten.	**JA** **X**	**NEI** ___
Kan besvarelsen brukes til undervisningsformål?	**JA** **X**	**NEI** ___
Vi bekrefter at alle i gruppen har bidratt til besvarelsen	**JA** **X**	**NEI** ___

IS-304

Bacheloroppgave i informasjonssystemer

NorgesEnergi

https://github.com/Bunnymann/NorgesEnergi.git

Universitetet i Agder – 2018

Skrevet av:

Marius Fosseli

Eirik Slagnes

Benjamin G Børresen

Veileder

Even Åby Larsen

Innholdsfortegnelse

Figurliste

Tabelliste

Forord

Denne rapporten er skrevet for bacheloroppgaven våren 2018, hos Universitetet i Agder, under kurset IT og Informasjonssystemer.

Ved slutten av 2017 ble gruppen Telos dannet og vi begynte å søke etter potensielle oppdragsgivere for en bacheloroppgave. Vi var alle på RefreshIT på CoWorx for å høre på lokale næringsdrivende fremvise sine prosjekter. Alle i gruppen mente prosjektet til NorgesEnergi virket veldig interessant, og etter god dialog var vi heldige å få lov til å få jobbe med dette.

Gruppen Telos har gjennom semesteret utviklet et hjelpesystem som omhandler hjelpeinformasjon fra en database. Administratorer kan sette inn data via en nettapplikasjon, og hovedsystemet henter hjelpetekst fra databasen via et API-kall. Ideen er at ansatte kan få hjelp til utfylling av forskjellige felter når de opererer i systemet de bruker daglig.

Dialogen med NorgesEnergi har vært veldig bra. Vi har arbeidet på samme kontor som veileder og hele It-avdelingen deres, dette har gjort at vi har dannet et godt bilde på arbeidslivet.

Vi ønsker og takke NorgesEnergi for å ha gitt oss muligheten til å jobbe med et veldig interessant og utfordrende prosjekt, samt gitt oss kontorplass under hele bachelor perioden. Vi ønsker også å takke våre oppdragsgivere ved NorgesEnergi, Trond og Eyvind. Vi vil også takke veilederen vår fra UIA, Even Åby Larsen.

Sammendrag

Denne rapporten er utarbeidet av gruppen Telos i forbindelse med bacheloroppgaven vår i IT- og Informasjonssystemer ved Universitet i Agder, våren 2018. Den beskriver prosesser og arbeid med å utvikle et hjelpesystem for strømleverandøren NorgesEnergi som skal fungere som et hjelpeverktøy for deres ansatte. Dette skal brukes til å søke etter hjelp i form av vanlig tekst som beskriver eksempelvis hva ansatte skal fylle inn i systemet de bruker daglig.

Ettersom ingen på gruppen tidligere hadde arbeidet med å utvikle .NET applikasjoner, startet vi semesteret med et innføringsmøte sammen med oppdragsgivergiver. Her fikk vi en informasjon om hvilke teknologier og programmer som skulle brukes. Vi startet individuelt med å bygge opp vår egen kompetanse før prosjektet startet.

Hovedverktøy som ble tatt i bruk under utviklingen av hjelpesystemet var:

- Microsoft Visual Studio og Azure database (Programvare)
- .NET og SQL (programmering)
- GitHub (versjonskontroll)
- Google Drive (oppbevaring av prosjektrelaterte dokumenter)

Rapporten er strukturert etter arbeidsprosessen, hvor analysedelen etterfølges av design og prosjektgjennomføringen til slutt ligger kronologisk oppsatt etter sprintene. Inneholder gir en beskrivelse for hvordan vi har arbeidet oss frem til det resultatet vi har oppnådd. Det er omfattende figurer og tabeller gjennom hele rapporten samt vedlegg for den informasjon vi ikke har prioritert i selve rapporten.

1.0 Introduksjon

1.1 Prosjektgruppen

Marius Fosseli

Epost: mariusfosseli@hotmail.com

LinkedIn: https://www.linkedin.com/in/marius-fosseli-b8bba714b/

- 23 år, Bachelor i IT og informasjonssystemer. Vil ut å jobbe etter endt studie
- Rolle i prosjektet: Backend/Frontend ansvarlig

Benjamin G Børresen

Epost: Benjamin.g.borresen@uia.no

LinkedIn: www.linkedin.com/in/benjamin-gutierrez-b%C3%B8rresen-9b398811a/

- 23 år, Bachelor i IT og informasjonssystemer. Vil ut å jobbe etter endt studie.
- Rolle i prosjektet: Koding og rapport.

Eirik Emil Slagnes

Epost: Eirsla14@gmail.com

LinkedIn: www.linkedin.com/in/eirik-slagnes-572008150/

- 26 år, Bachelor i IT og informasjonssystemer, planlegger Master innenfor IT etter endt studie.
- Rolle i prosjektet: Rapport ansvarlig

1.2 Vår klient - NorgesEnergi

Vår klient er NorgesEnergi, en av Norges største strømleverandører. Oppgaven vi har blitt tildelt går ut på å opprette et hjelpesystem for deres ansatte, som skal brukes sammen med et nytt system som er under utvikling hos dem. NorgesEnergi har ønsket et slikt system lenge, men har aldri prioritert det. De håper å kunne ta systemet vårt i bruk i hele deres konsern, samt i søsterselskapene i Sverige og Finland.

1.3 Hvorfor dette prosjektet?

Vi ønsket dette prosjektet fordi det ville by på utfordringer, og dermed gi et stort læringsutbytte for oss. Vi skal ta i bruk programmer og språk som vi ikke har brukt før, blant annet Visual Studio og Azure, som er Microsoft sine egne programvarer. Disse programmene blir brukt av flere bedrifter og var med i begrunnelsen til hvorfor vi ønsket å jobbe med dette prosjektet. Som nevnt har produktet vært ettertraktet av NorgesEnergi lenge, og derfor mente vi at prosjektet hadde en viktig verdi.

1.4 Hva skal vi produsere?

Produktet vi skal lage er et API for systemet som ansatte ved NorgesEnergi bruker for og blant annet registrere nye kunder eller opprette fakturaer. Målet er et smart system som skal kunne vite hvor brukeren er i programmet, slik at hjelpeteksten som vises vil være relevant til de feltene som brukeren skal fylle inn.

2.0 Analyse

I analysedelen av denne rapporten vil du kunne lese om hvordan vi har gått frem for å finne og beskrive problemer og løsninger i prosjektet. Her finner du forskjellige modeller og diagrammer som beskriver de ulike kravene til produkteier. Dette har vi gjort ved å konstruere brukerhistorier basert og bygd opp på de forskjellige kravene. Ut i fra disse har vi også utviklet et klassediagram som viser hva de forskjellige klassene i systemet består av. De forskjellige modellene og diagrammene i denne delen av rapporten er laget for å danne et bilde av hvordan sluttproduktet skal se ut og hva det skal inneholde.

2.1 Rikt bilde

Vi har benyttet rike bilder for å utforske og definere en eller flere situasjoner ved bruk av en foreløpig mental modell. Dette er et kraftig verktøy i analysefasen ettersom det er enkelt å gjennomføre. Dette gjør at vi kan danne et bilde av hvordan systemet vi lager bygges opp, hvordan informasjonsflyten og systemarkitekturen vil se ut og hvordan systemet skal fungere i praksis.

Under et møte med oppdragsgiver fikk vi kartlagt hvordan en bruker skulle gå frem i systemet ved hjelp av stages, og hvilke brukerroller som skulle ha tilgang. Det kom også frem at vi ikke skulle fokusere noe på design delen da NorgesEnergi ville stå for dette selv, vi valgte derfor å ikke ta i bruk papirprototype. Det de ønsket vi skulle gjennomføre var selve funksjonene i systemet.

2.1.1 Rikt bilde - Stages

Vi har valgt å lage to rike bilder av essensielle deler av hjelpesystemet for å gi en introduksjon til hvordan vi har valgt å bygge prosjektet. Det første rike bildet viser de forskjellige klassene for “stages” og er en indikator på hvor du befinner deg i systemet. Grunnlaget for at vi valgte å bruke rike bilder av de forskjellige stegene er for å gi en grafisk framstilling av hvordan man tar seg fram i systemet og viser hvor man befinner seg etter hvert steg.

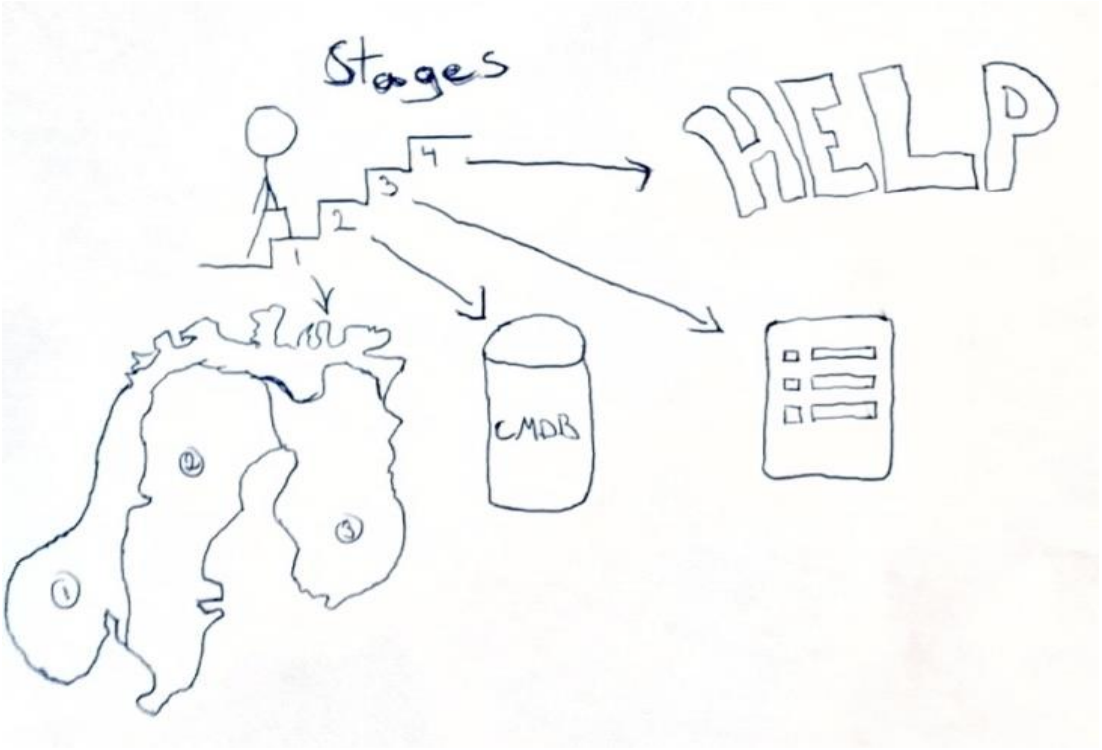
Under har vi beskrevet de fire stegene:

- Stage1- Norge/Sverige/Danmark - Hvilket land du befinner deg i
    - Dette steget viser systemet hvilket land du befinner deg i og gir brukeren tilgang til de hjelpetekstene fra tilhørende nasjon. Dette er fordi oppdragsgiver ønsker å fordele hjelpetekstene mellom søsterselskapene.

- Stage2- Systemet som bli brukt - CMDB (systemet brukt av Norges Energi)
    - Dette steget viser til hvilket system som blir tatt i bruk. Det er for øyeblikket bare CMDB som blir brukt i alle NorgesEnergi sine kontorer.

- Stage3- Kategori / hvor eksempel hjelpe kategorier legges til
    - Dette steget viser systemet hvor brukeren befinner deg innad i systemet (Privatkunde, bedriftskunde, hvor/hvilken side brukeren er inne på).

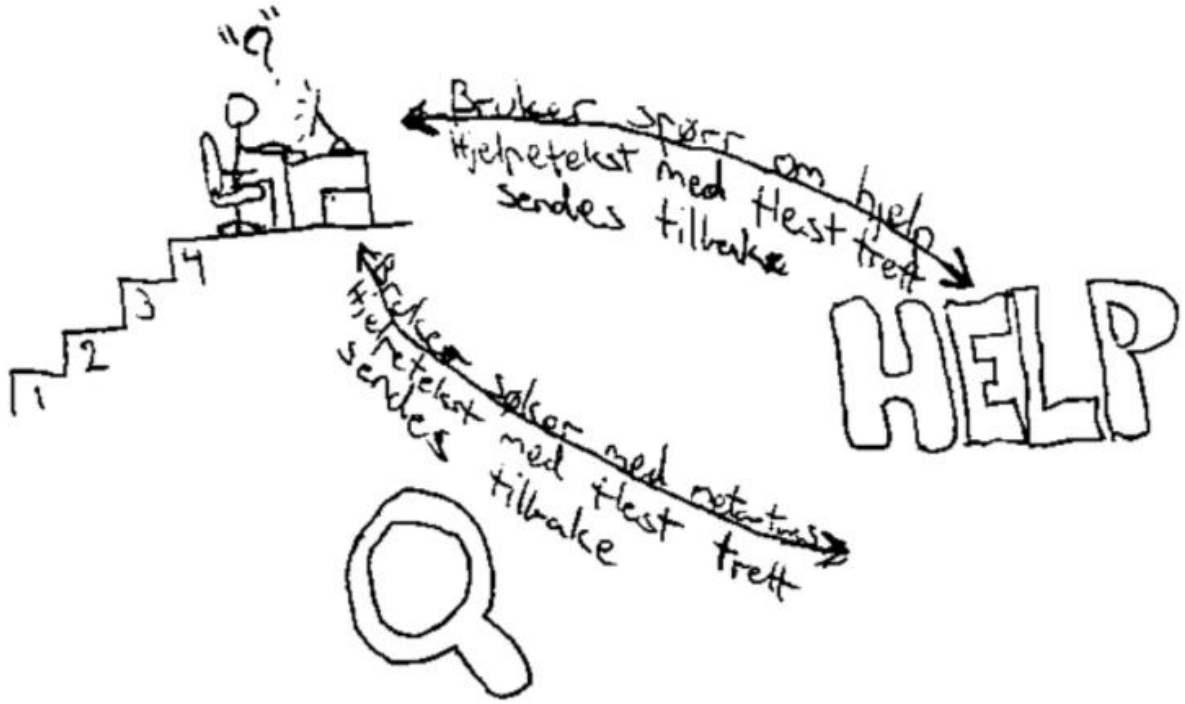
- Stage4- *Label* (felt)
    - Dette steget viser systemet hvilket felt brukeren ønsker å vite mer om.



Figur 1: Rikt bilde «Stages»

Bildet ovenfor beskriver hvordan brukeren tar seg gjennom hvert steg. Ønske fra NorgesEnergi var at systemet skulle være «smart» nok til å selv innhente data for å plassere brukeren i hvert steg. Dette lot seg derimot ikke gjøre, blant annet fordi systemet NorgesEnergi bruker ikke hadde noen indeks som indikerte hvilken stage3 eller stage4 brukeren etterspurte. Stage1- og 2 ble definert i SQL-spørringen slik at norske brukere fikk fremvist norske hjelpetekster, samt fra det systemet brukeren tilhørte (sistnevnte utgikk riktignok ettersom konsernet kun benytter et system).

2.1.2 Rikt bilde – Stage4 klassen



Figur 2: Rikt bilde «spørre om hjelp»

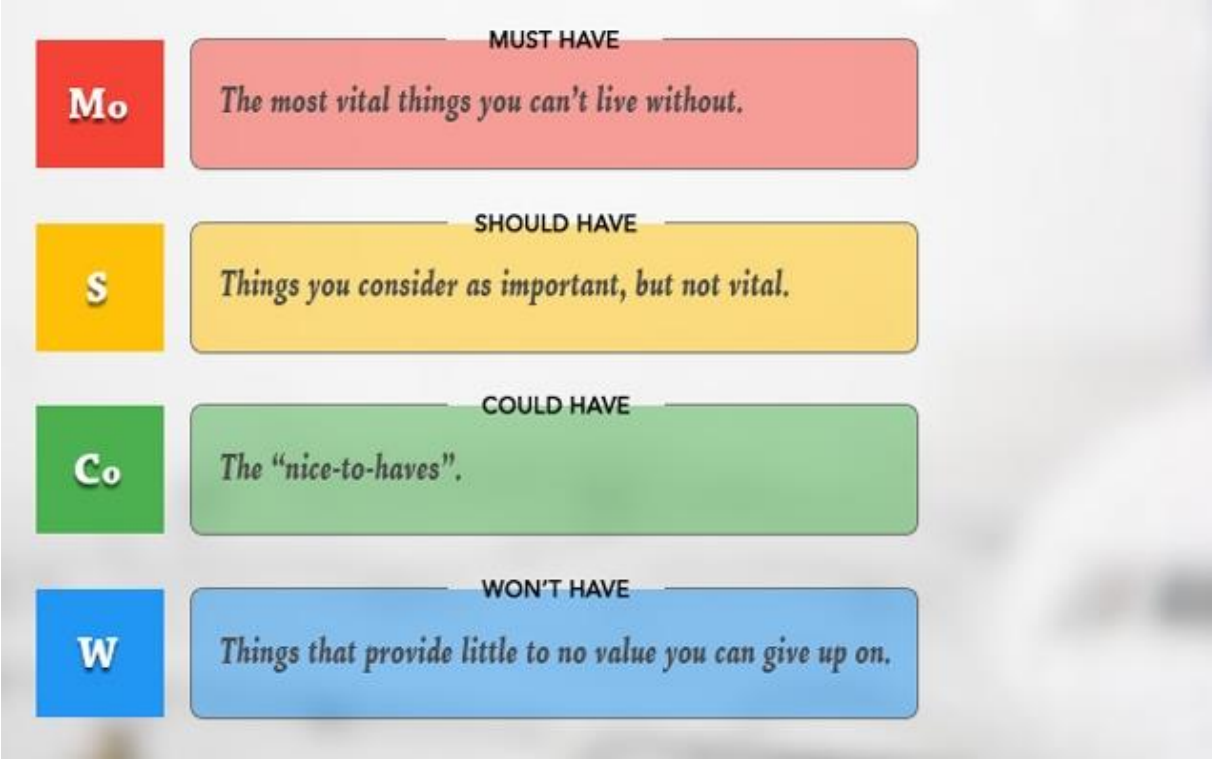
Ved opprettelse av en hjelpetekst legges teksten inn i en stage4 i databasen som en fremmednøkkel. Da vil en stage4 ha en tilhørende hjelpetekst, og hver hjelpetekst har en eller flere tilhørende metatags; det er metatags som er søkekriteriene i funksjonen.

Dersom en bruker er usikker på hva som skal fylles inn i et gitt felt (stage4), skal brukeren benytte vårt hjelpesystem for å søke om hjelp. Som nevnt var det ønsket at systemet selv skulle vite hvilket felt som brukeren etterspurte, men dette var noe vi ikke fikk til. Brukeren må nå søke etter metatags for å få hjelp. Vårt system returnerer de tre hjelpetekstene med flest treff på de metatags brukeren søker etter, sortert etter antall treff. Per nå vil systemet returnere tekst for riktig nasjon og system etter hvor brukeren er registrert, men stage3 kan være noe tilfeldig fordi det har vært komplikasjoner med vårt og NorgesEnergi sitt system.

2.3 Brukerhistorier

Vi har valgt å lage brukerhistorier for å komme frem til de forskjellige muligheter oppdragsgiver trenger i systemet. Dette gav oss et godt bilde på hvilke funksjoner vi skulle implementeres. Systemet vi skal lage har to brukertyper: Administrator og vanlig bruker.

For å vurdere brukerhistoriene og finne ut hvilke funksjoner som skulle prioriteres, benyttet vi MoSCoW metoden. Metoden brukes i ledelse, forretningsanalyse, og programvareutvikling for å ha en felles forståelse om viktigheten av alle krav.



Figur 3: MoSCoW tabell: (Agile Business Consortium, 2017)

De to siste kategoriene, "Could have" og "Won't have", hvor "Could have" er mindre viktige funksjoner enn de som burde inkluderes, men kan løfte prosjektet. Disse vil vi ikke prioritere med mindre tiden tillater det. Som vi ser av figur 3 er den siste kategorien "won't have". Disse funksjonene gir liten eller ingen verdi til prosjektet, og vil ikke bli implementert. «Must have» funksjonene er de essensielle funksjonene i systemet, hvor vi i vårt prosjekt har definert metoder for å opprette og redigere hjelpetekst samt å søke etter hjelpetekster som

førsteprioritet. «Should have» er funksjoner som burde være med, men som systemet kan fungere uten.

Ut fra arbeidsbeskrivelsen fikk vi forklart hvordan brukere skulle kunne bruke programmet og utfra dette har vi utformet brukerhistorier (Levy, 2014) for våre brukergrupper. Brukerhistoriene inneholder en kort beskrivelse av ønsket funksjonalitet, og for hvilken bruker hendelsen påvirker.

Nedenfor er brukerhistoriene vi har laget fra arbeidsbeskrivelsen vi fikk av NorgesEnergi:

Tabell 1: Brukerhistorier

NR.	Som en	Ønsker/Vil ha	Slik kan/For å/For at	MoSCOW	Rekkefølge
1	Admin	Legge til hjelpetekst	Brukere kan få hjelp	Må ha	1
2	Admin	Kunne redigere hjelpetekst	Endre eller fikse hjelpetekst	Må ha	2
3	Bruker	Søkefunskjon	Enklere finne hjelp	Må ha	3
4	Bruker	At relevant hjelpetekst vises per side	Enklere finne relevant hjelp der man er	Burde ha	4
5	Admin	Kunne slette hjelpetekst	Slette tekst som ikke brukes lengre	Burde ha	5
7	Admin	Legge til hjelpetekst i form av bilder	Legge til hjelpetekst i form av bilder	Kan ha	7
8	Admin	At brukere kan legge til tilbakemeldinger på hjelpetekst	Finne ut om hjelpeteksten er god og gir riktig hjelpe for brukeren	Kan ha	8

2.4 Hendelsestabell

Hendelsestabellen består av de forskjellige klassene og hendelsene mellom dem i systemet. Tabellen under viser hvilke klasser som er involvert i hvilke hendelser. Tabellen inkluderer de viktigste hendelsene basert på våre brukerhistorier. Brukerhistoriene vi har klassifisert som "must have" og "should have" i vår MoSCoW tabell er dem vi har hatt tatt med i denne tabellen.

Tabell 2: Hendelsestabell

Hendelser	Info	Stage1	Stage2	Stage3	Stage4	Helptext	Helptexttag	Metatag
Hjelpetekst opprettet	X	X	X	X	X	X	X	X
Hjelpetekst redigert	X					X		
Hjelpetekst slettet	X				X	X	X	
Søk etter hjelpetekster	X	X	X	X	X	X	X	X

Hendelsestabellen vår er litt spesiell, da flere klasser ofte brukes under de ulike hendelsene. Dette kommer av at informasjonen blir hentet/opprettet basert på *info_ID* verdien. Systemet går da gjennom alle tabellene før den henter det ønskede objektet.

2.4.1 Hendelser i hendelsestabell

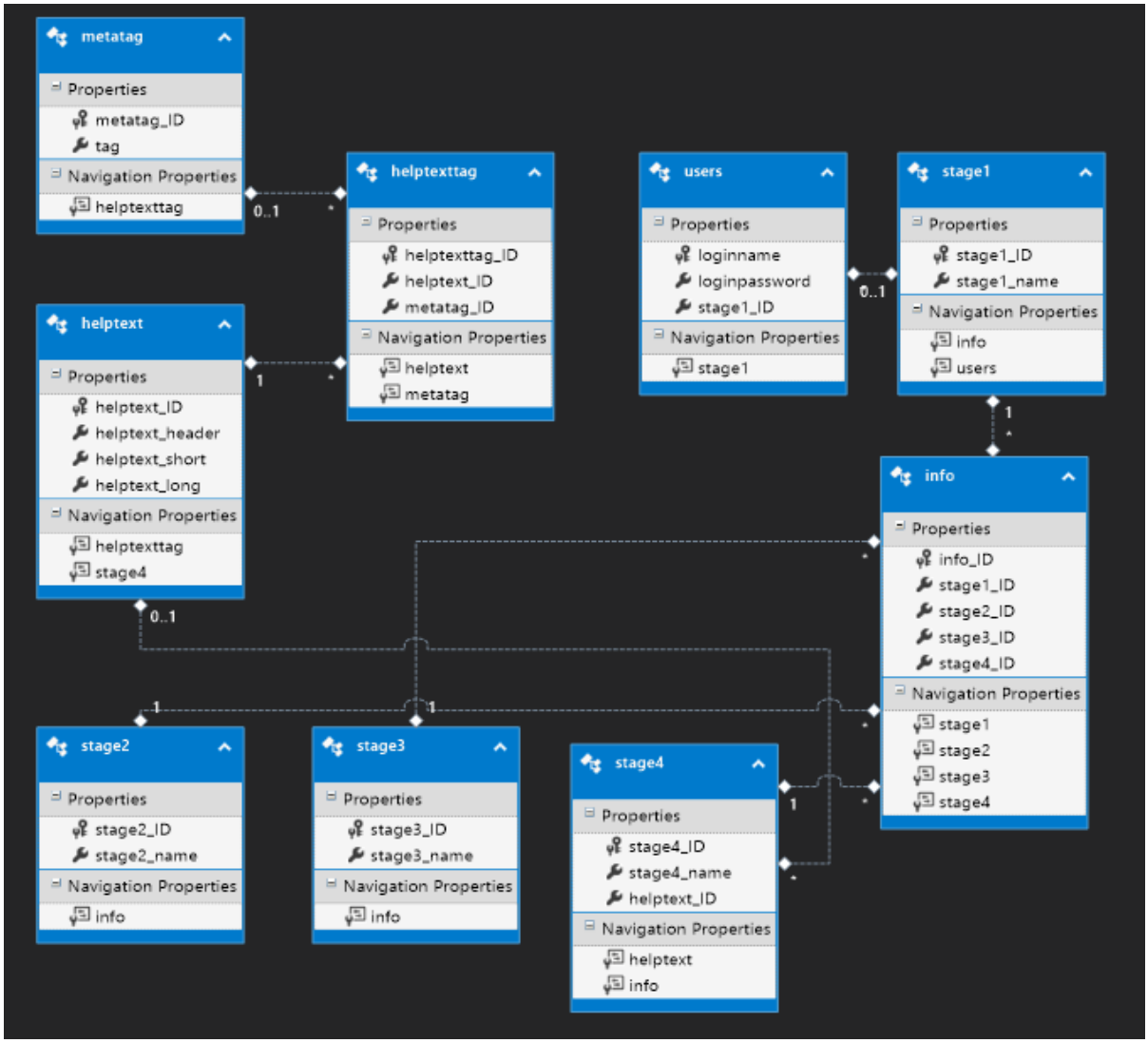
Her forklarer vi hva som skjer under de viktigste hendelsene fra hendelsestabellen i mer detalj. Vi legger det frem slik for at du ikke bare skal kunne se hvilke klasser som henger sammen under ulike hendelser, men også hva som skjer under interaksjonene mellom disse.

Hjelpetekst opprettet/redigert/slettet: Denne hendelsen innebærer at du er logget inn som administrator (superbruker), og gir deg rettighetene til å gjøre ulike endringer. Hjelpeteksten du oppretter blir lagt til i en stage4 som bruker oppretter samtidig. Videre blir denne stage4 lagt til i *info*-tabellen, sammen med stage1-3 (valgt fra *dropDownList*). Hjelpeteksten får også tilhørende metatags som bruker oppretter, og disse blir knyttet sammen i *helptexttag*-tabellen. Ved redigering av hjelpetekst er det kun selve hjelpeteksten som blir redigert, NorgesEnergi mente det ikke var nødvendig å kunne redigere nasjon eller system.

Søk etter hjelpetekst: Denne hendelsen oppstår når en vanlig bruker trenger hjelp til noe i CMDB systemet. Bruker søker etter de ulike hjelpetekstene ved hjelp av metatags. Som vi har nevnt tidigere returnerer og sorterer systemet de hjelpetekstene med mest treff på metatags. Alle klasser er involvert i denne hendelsen da spørringen er bygd opp for å returnere en *info-ID* verdi, og å bygge objektene med denne verdien.

2.5 Klassediagram

Klassediagramet er tatt ut fra prosjektet vårt i Visual Studio ved å hente *Entity Models*. Diagrammet viser alle klassene som er involvert i prosjektet og hvordan de er knyttet sammen med fremmednøkler.



Figur 4: Klassediagram

Hensikten med klassediagramet er å vise de klassen i vårt system, samt sammenhengen mellom dem. Vi tar med klassediagramet for å gi leser en bedre forståelse av klasser, objekter og felter i systemet. Her er det viktig å få fram at klassediagramet viser en statisk versjon av systemet, og det forteller ingenting om hvordan det handler eller oppfører seg under bruk. Som nevnt tidligere går mange av spørringene våre gjennom info tabellen og

klassediagramet har vært et hjelpemiddel for å skrive SQL-spørringer. Ved å benytte klassediagramet har vi kunne sett hvordan vi skal koble sammen tabellene på en enkel og oversiktlig måte. Det har også gitt et bedre bilde på arv mellom klassene.

2.5.1 Klassebeskrivelser

Users

Denne klassen inneholder all informasjon om brukeren. Det er her det blir definert om det er vanlig bruker eller administrator som logger seg inn, med stage1 (nasjon) som fremmednøkkel. Det ønskelige var at administratorer ved NorgesEnergi skulle være logget inn sammen med innloggingen de benyttet i sitt eksisterende system. Vanlige brukere ville oppdragsgiver at skulle benytte *tokens* da dette ville gjort det mulig for ansatte og kun se informasjon skrevet på deres språk / hentet fra deres land.

Info

Info-tabellen fungerer som en "samlingstabell" for å knytte alle tabellene sammen. Tabellen inneholder ID-verdier fra Stage1-, -2, -3, og -4, hvor Stage4 videre har helptext_ID som fremmednøkkel. Dette gjør at hver hjelpetekst blir lagt i riktig nasjon, riktig system, riktig kategori i systemet og kun tilhører en label. Gjennom denne tabellen kan vi hente verdier fra alle tabellene ved hjelp av info_ID verdien.

Metatag

Metatag tabellen inneholder kun metatag og tilhørende ID verdi.

Helptexttag

Helptexttag er en mange-til-mangetabell. Dette gir oss muligheten for at en metatag er tilgjengelig til flere hjelpetekster, samtidig som flere hjelpetekst kan ha flere metatags.

Helptext

Dette er hovedtabellen i systemet - her oppretter vi selve hjelpeteksten som skal bistå de ansatte som tar i bruk applikasjonen vår. Overskrift, et kort sammendrag og en fullstendig

tekst er feltene som skal fylles inn i tabellen og som nevnt ovenfor legges det til metatags som bruker skriver for den gitte hjelpeteksten.

Stage 1

Stage1 tilsvarer nasjon og har kun en varchar input- og auto_increment ID verdi.

Stage 2

Stage2 tilsvarer system og har kun en varchar input- og auto_increment ID verdi.

Stage 3

Stage3 tilsvarer hvilen kategeri i systemet det er snakk om, privatkunde, bedriftskunde, faktura etc. og har kun en Varchar input- og auto_increment ID verdi.

Stage 4

Stage4 tilsvarer en label under stage3 og har kun en Varchar input- og auto_increment ID verdi. Stage4 har også som nevnt ovenfor heltpext_ID som fremmednøkkel.

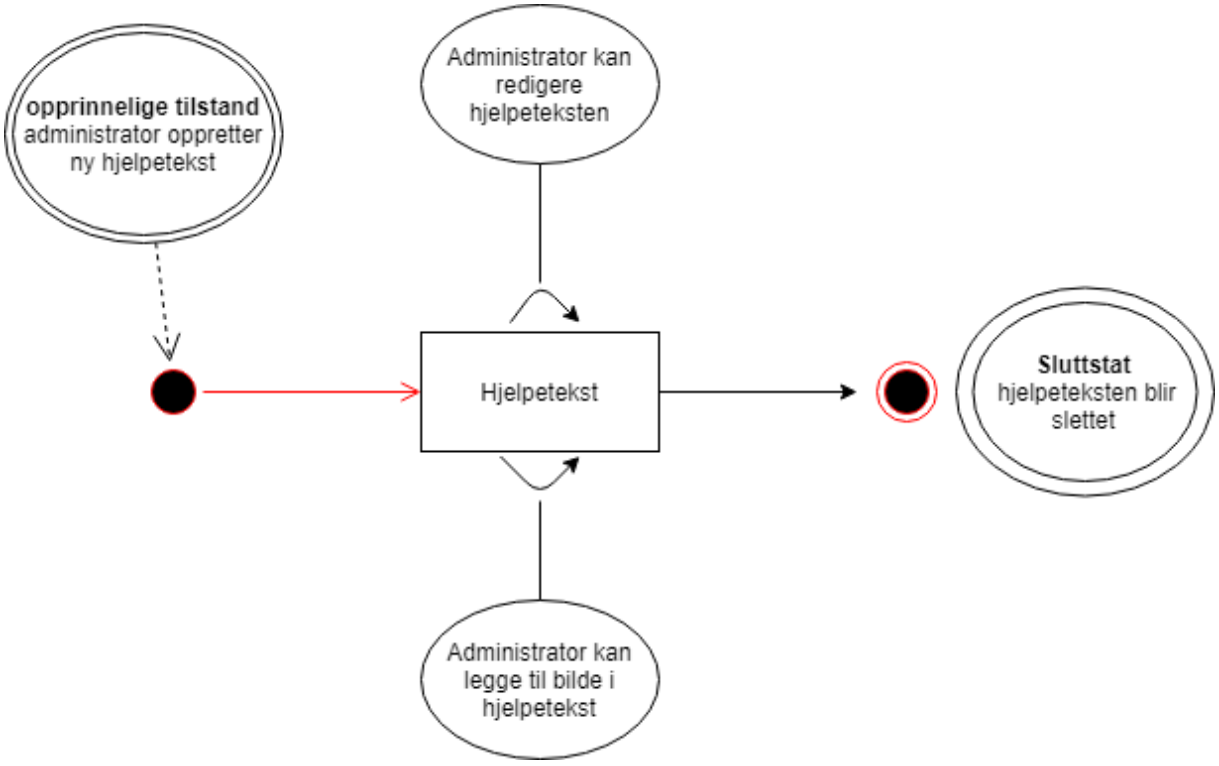
2.6 Tilstandsdiagram

Et tilstandsdiagram er et diagram som brukes i datavitenskap og relaterte felt for å beskrive oppførselen til forskjellige klasser i et system. Tilstandsdiagrammer krever at systemet/funksjonen beskrevet består av et begrenset antall tilstander.

Vi har i siste fase av analysedelen valgt å lage tilstandsdiagrammet for 'hjelpetekst' klassen for å bedre forklare hvordan den fungerer for administratorer og vanlig bruker.

2.6.1 Tilstandsdiagram for 'hjelpetekst' klassen

Vi har valgt å vise hvordan administrator og vanlig bruker påvirker tilstanden til "hjelpetekst" klassen og hvordan den oppfører seg i systemet vårt.

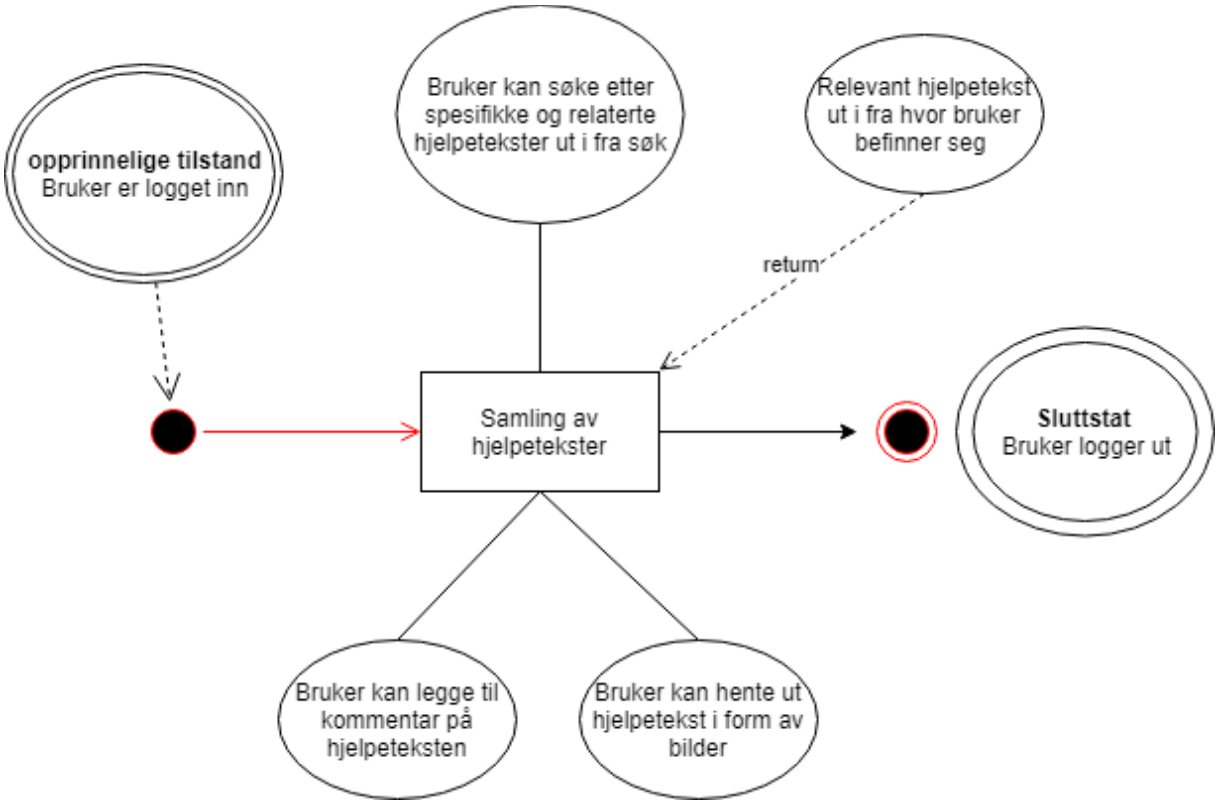


Figur 5: Tilstandsdiagram «Hjelpetekst bruker»

Objekter av hjelpetekst-klassen opprettes av administrator, som publiserer disse i systemet. Når en administrator oppretter en ny hjelpetekst legges den til i databasen med tilhørende nasjon og system administrator har spesifisert. Når en hjelpetekst blir opprettet vil den være tilgjengelig fram til den blir slettet. Ved endring vil den oppdaterte tilstanden være synlig

først etter endringen er lagret.

Tidligere kopier vil altså ikke eksistere etter en hjelpetekst er endret, slik at ved redigering av en hjelpetekst vil tidligere versjoner forsvinne etter lagring.



Figur 6: klassediagram «hjelpetekst admin»

Når brukeren befinner seg i hjelpesystemet har de ett alternativ for å søke etter det de lurer på ved hjelp av et søkefelt. Søkefeltet er et enkelt input-felt som deler opp alle ordene som blir søkt etter (oppdeling skjer mellom ‘,’ , ‘;’ eller ‘:’) . Hjelpetekster med flest treff av ordene som blir søkt på vil vises for den ansatte i systemet. Bruker søker på «metatags» (søketags)» som administratorer legger til når en hjelpetekst blir opprettet.

Tilstandsdiagrammet er også utviklet basert på brukerhistoriene da disse beskriver i liten grad tilstandene til objektene. Vi har fokusert på å utdype tilstanden til hjelpetekst- og brukergruppeobjektene. Ved å kartlegge de hendelser som er mulig ved en gitt tilstand, har vi kunne fokusert på de funksjonene som bidrar til å oppnå disse tilstander og hendelser.

Gjennom analysedelen har vi kartlagt flere av de viktigste funksjonene NorgesEnergi trenger for å ta i bruk dette systemet. Brukerhistoriene og MoSCoW tabellen har gitt oss gode innblikk på funksjonalitet og fokusområder. Vi har også tatt i bruk og fått god nytte av klassediagramet og hendelsestabellen for å vite hvordan både klassene og iterasjoner samhandler, og bidra til å skrive SQL-spørringer. Ettersom prosjektet har vært relativt lite i forhold til bredde i funksjonalitet har vi gjort en tilstrekkelig analyse. Brukerhistoriene og MoSCoW tabellen er utviklet sammen med oppdragsgiver og bidratt til hvordan vi har gått frem i sprintene.

3.0 Design

Design-delen av prosjektet går ut på å finne og modellere konkrete løsninger for problemene som ble avdekket og beskrevet i analysedelen. Her har vi tatt for oss hvilke funksjonaliteter vi trenger å implementere i systemet. Vi viser til noe av tankegangen gjort rundt de enkelte oppgavene en administrator eller ansatt skal kunne utføre ved hjelp av diagrammer og lister, med fokus på våre «must have» funksjoner.

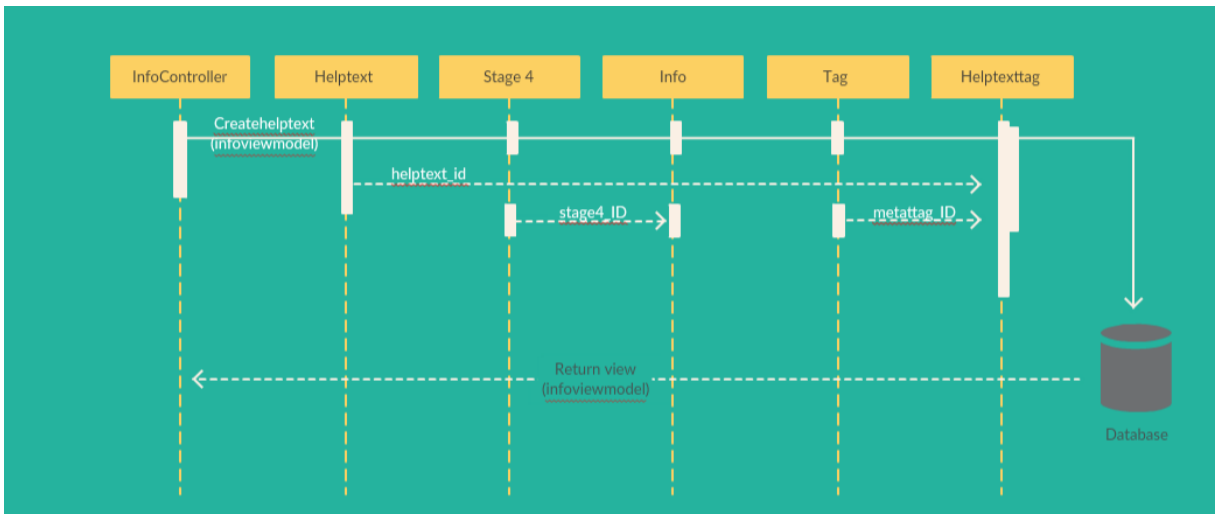
3.1 Brukergruppe

Brukergruppa for dette systemet er de ansatte hos NorgesEnergi og deres søsterselskaper. Enkelte ansatte vil blir tildelts administrative rettigheter med mulighet for å gjøre endringer innad i applikasjonen, men den vanlige brukeren vil kunne bruke det som et hjelpemiddel. I og med at dette er et API som legges oppå et allerede eksisterende system kan vi anta at de fleste ansatte er kjent med navigering i deres digitale grensesnitt. Om de ansatte behøver hjelp til hva som skal fylles inn i ulike felter, er det dette systemet vårt skal bistå med.

3.2 Sekvensdiagram

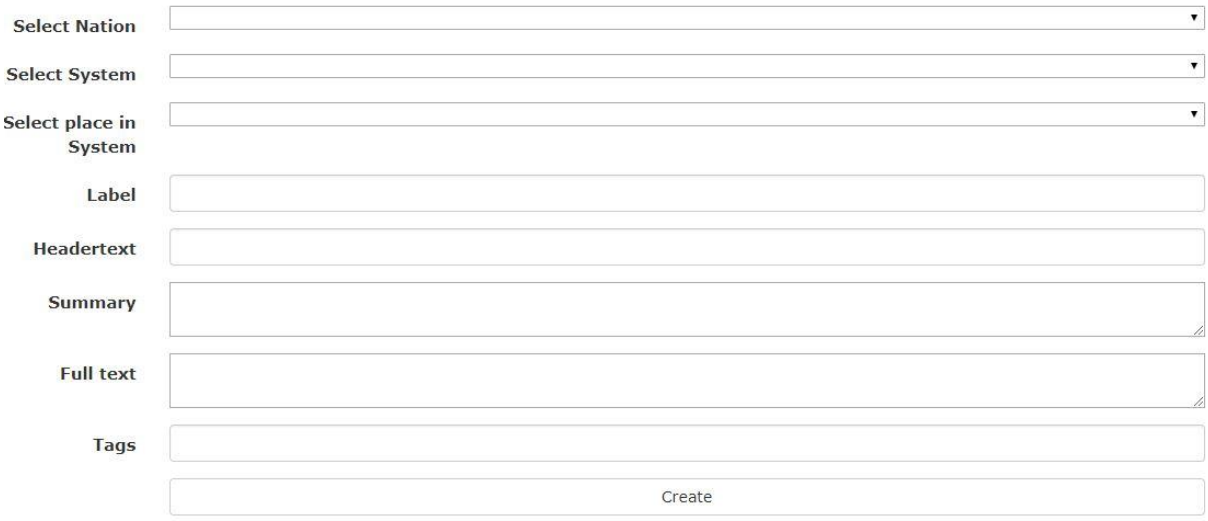
Vi fortsetter å bruke vårt systems viktigste brukerhistorier og har konstruert sekvensdiagrammer ut fra disse. Vi ser fra hendelsestabellen hvilke klasser det er som er involvert under de ulike sekvensene, men utdyper i mer detalj de ulike metodene og returtypene som blir sendt i systemet. Ved å holde fokus på hovedfunksjonene blir det lettere å forstå hvordan de fungerer i praksis og det lar oss som utviklere se hva som skjer bak de ulike sekvensene for å øke vår egen forståelse. Sekvensdiagrammene tas i bruk for å vise hvordan objekter jobber sammen når en oppgave blir utført og under forklarer vi brukerhistoriene og det tilhørende diagrammet, samt en beskrivelse av hendelsesforløpet.

"Som administrator i systemet, ønsker jeg å kunne legge til hjelpetekst."



Figur 7: Sekvensdiagram for «CreateHelptext»

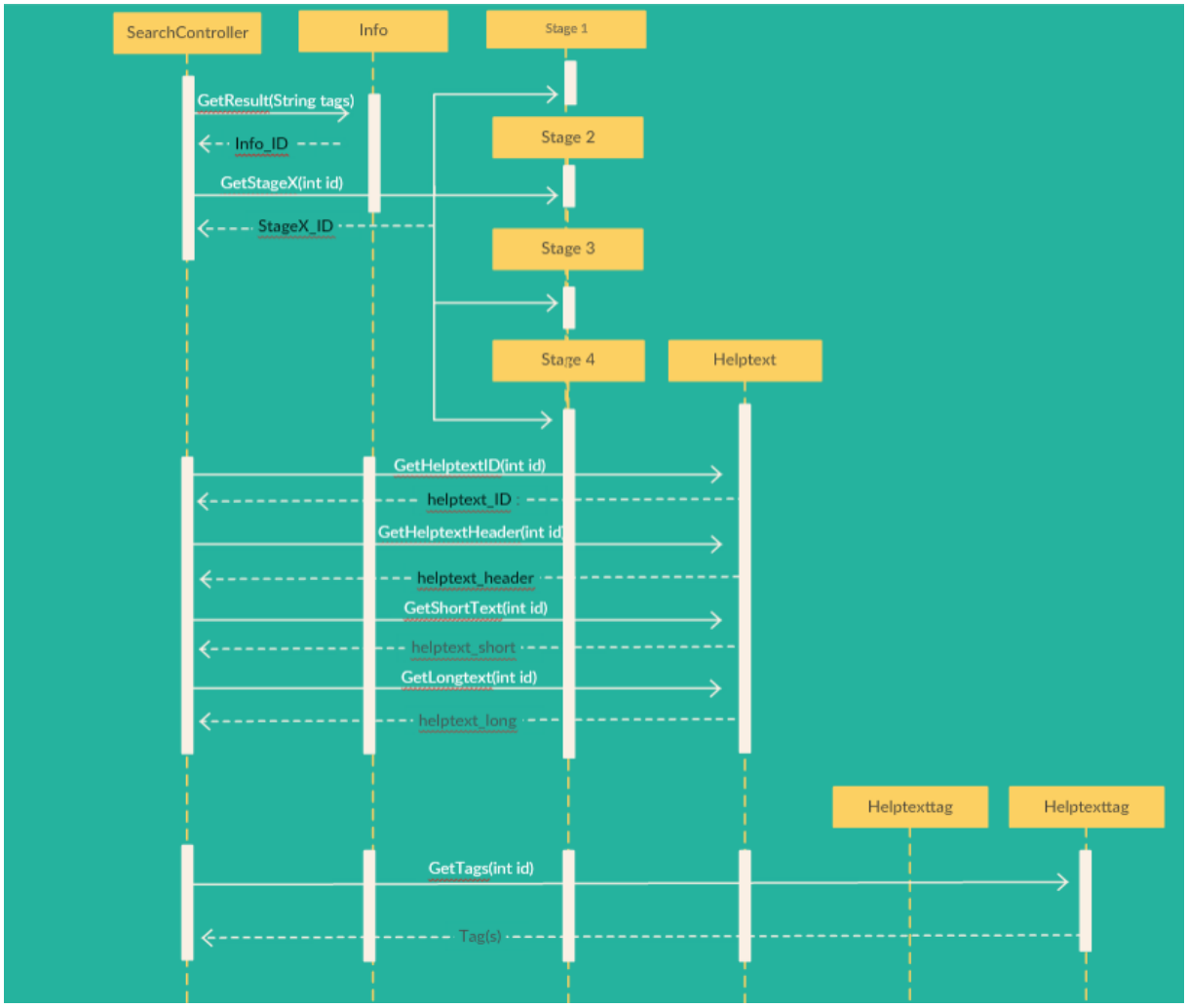
Metoden har et InfoViewModel som parameter, parametervariabelen blir definert som model. Datamodellen InfoViewModel har vi selv opprettet og inneholder de variablene vi trenger for å lagre verdi i alle databasens tabeller. I metodens tilhørende View kan vi derfor fylle inn verdi til Stage4_name og alle kolonnene til helptext-tabellen samt string for metatags (figur 8). Vi har i tillegg opprettet metoder som lar oss velge verdi for Stage1-, -2 og -3 fra dropDownList.



Figur 8: Utklipp av «CreateHelptext» metodens view

Fra bildet ovenfor ser vi hvordan metodens view blir seende ut i nettleser. Som det står nevnt ovenfor er Stage1-3 dropDownList, der bruker velger den valgte stage sin string verdi, og den tilhørende ID-verdien blir lagt i Info-tabellen. Stage4 (label) blir opprettet i View'et, og den opprettede stage4 sin ID verdi er det som blir lagt til i Info-tabellen. Videre i metoden legges helptext, summary og fulltext inn i helptext-tabellen, hvor ID-verdien blir lagt til i Stage4. For tags har vi opprettet en funksjon som splitter teksten i input-feltet ved komma, punktum, kolon etc., slik at vi kan opprette flere metatags per hjelpetekst. Den siste prosessen i metoden er da at hjelpetekstens ID-verdi blir tilegnet hver enkelt metatag ID-verdi - dette skjer i mange-til-mange tabellen helptexttag.

"Som bruker av systemet, ønsker jeg å kunne søke etter hjelpetekst."



Figur 9:Sekvensdiagram for «GetSearchResult»

Ovenfor ser vi sekvensen ved metoden “GetSearchResult”. Denne metoden avhenger av ,flere andre metoder og tar i bruk Info_ID verdien som blir hentet ved hjelp av Dapper i en SQL-spørring. Bruker søker på x-antall metatags, metatags blir delt opp mellom samme tegn som ved input i CreateHelptext-metoden. Metoden returnerer de fire info_ID verdiene med flest treff på de søkte tags.

```
[HttpGet]
0 references | 0 changes | 0 authors, 0 changes | 1 request | 0 exceptions
public ActionResult GetSearchResult(string tags)
{
    var obj = conn.Query<InfoViewModel>(GetSearch(tags)).Take(4).ToList();

    List<InfoViewModel> result = new List<InfoViewModel>();
    if (obj != null)
    {
        foreach (var row in obj)
        {
            InfoViewModel model = new InfoViewModel
            {
                stage1_name = GetStage1(row.info_ID),
                stage2_name = GetStage2(row.info_ID),
                stage3_name = GetStage3(row.info_ID),
                stage4_name = GetStage4(row.info_ID),
                helptext_ID = GetHelptextID(row.info_ID),
                helptext_header = GetHelptextHeader(row.info_ID),
                helptext_short = GetShortText(row.info_ID),
                helptext_long = GetLongText(row.info_ID),
                tag = GetTags(row.info_ID),
            };
            result.Add(model);
        }
    }
    return View(result);
}
```

Figur 10: Kodeutklipp av «GetSearchResult» metoden

Som bildet illustrerer, bruker vi en metode kalt GetSearch med string parameter for å opprette selve SQL-spørringen. GetSearchResult metoden bruker også et string parameter, og det er denne verdien som blir brukt i GetSearch-metoden. Videre bygger vi en InfoViewModel model som blir lagt til i en liste (*result*). Hver datatype i InfoViewModel blir laget ved hjelp av metoder som returnerer en string verdi (foruten heltpext_ID som returnerer en integer). Alle metodene har int verdi som parameter, og returner den gitte verdi ut fra den Info_ID som vi henter i SQL-spørringen.

3.3 Funksjonsliste

Nedenfor har vi en funksjonsliste som viser funksjoner som inngår i våre brukerhistorier. Den forteller vår estimerte vanskelighetsgrad per funksjon, samt hvilken type funksjon dette er. Siden disse funksjonene er nødvendige for at brukerhistoriene skal bli oppfylt er det viktig at vi som utviklere får en oversikt over funksjonen som må implementeres og dens vanskelighetsgrad. Vi har delt inn funksjonene i tre vanskelighetsgrader; lett, moderat og vanskelig.

Alle oppgavene vi skal implementere krever at vi gjør en del forskning før vi begynner, da vi verken har brukt programvaren eller MVC rammeverket før. Derfor går vi ut fra disse vanskelighetsgradene etter vi har gjennomført kompetansebygging innenfor dette.

Lett: Vi har kunnskap nok til å kode med lite til ingen forskning.

Moderat: Kode vi har delvis kunnskap om, men trenger å søke opp for å utføre. Dette er kode vi har erfaring med, men ikke tilstrekkelig nok for å kunne ta i bruk uten forskning.

Vanskelig: Dette er kode vi har lite kunnskaper om eller vi ikke har vært borti før. Her kreves det mye forskning.

Tabell 3: Funksjonsliste

Funksjon	Vanskelighetsgrad	Type
Legge til hjelpetekst	Lett	Sende info
Kunne redigere hjelpetekst	Moderat	Hente info -> Sende info
Få opp relevant hjelpetekst	Vanskelig	Hente info
At relevant hjelpetekst vises per side	Vanskelig	Hente info
Kunne slette hjelpetekst	Lett	Hente info -> Sende info
Søkefunksjon for hjelpetekst	Vanskelig	Sende info -> Hente info
Legge til hjelpetekst i form av bilder	Vanskelig	Sender info
At brukere kan legge til tilbakemeldinger på hjelpetekst	Vanskelig	Sender info

3.4 Kvalitetssikring

Produktet blir testet i views ettersom vi bruker MVC i back-end. Hver *model* har en tilhørende *controller* som igjen gir *views* som benyttes til front-end. CRUD operasjoner (Create, Read, Update, Delete) blir testet ved å kjøre applikasjonen og gjennomføre alle operasjonene. Data blir også lagt til i databasen ved SQL-spørringer for å finne ut hvor fort og hvorvidt back-end funksjoner samsvarer med front-end display.

Unit testing har vi ikke benyttet i vårt produkt da mye av kvaliteten er sikret i databasen ved at eksempelvis alle primærnøklers ID har auto-increment (automatisk verdiøkning) for ny rad i tabellen. I tillegg har de nødvendige kolonnene i tabellen *not null* felt som gjør at data må fylles inn for at operasjonen blir verifisert.

I hver *model* blir også datatype definert. Dette gjør at kun int-verdier kan legges til i felt som tilhører int-verdier i databasen, og tilsvarende blir string-verdier knyttet opp mot varchar-verdier i databasen. At kun riktige datatyper blir godtatt i systemet testes som nevnt ovenfor ved å kjøre applikasjonen og å sette inn feil datatyper for å teste de feilmeldinger vi har definert i koden.

Til ettertanke kunne vi ha kvalitetssikret systemet ved å la ansatte ved NorgesEnergi test og komme med tilbakemeldinger. Dette ville også ha effektivisert den manuelle testingen vi har benyttet.

Vi kan selv ikke kvalitetssikre innholdet i systemet, men vi har utviklet systemet slik at datatyper hele veien blir riktig i henhold til databasen, samt at alle felter skal fylles inn ved hver ny opprettelse av hjelpeinformasjon (figur 8).

I hver model har vi lagt på *data annotation* som for eksempel *recuired*. Dette gjør at datafeltet må fylles inn, videre kunne vi benyttet *string lenght* som forteller noe om min/max lenge på feltet som kan fylles inn.

3.4.1 Testing

Vi har i vårt system kun brukt manuell testing. Ved å opprette CRUD-operasjoner til hver tabell i databasen har vi gått gjennom og testet at de ulike funksjonene opererer slik vi ønsker. Som nevnt ovenfor ved kvalitetssikring, inneholder modellene allerede definerte

datatyper som ikke gir oss mulighet til å legge inn tekst eller tegn hvor databasen skal ha tallverdier - dette har vi selvsagt testet og det har fungert som ønskelig. Det å teste et view sitt design er i og for seg en enkel oppgave da endringer i koden til individuelle views vises ved å oppdatere nettsiden. Testing av funksjoner i kontrollene er derimot noe mer tidkrevende da applikasjonen må lukkes og åpnes hver gang vi endrer serversidekoden.

Vi har ikke benyttet noen form for unit-testing. Det finnes ulike tester for å teste .NET applikasjoner, men det vi leste oss frem til på .NET sin dokumentasjon skrevet av Microsoft, var at ved unit-testing var det ikke anbefalt å teste infrastruktur som database (mairaw, et al., u.d.)

Ettersom vår applikasjon kun benytter, relativt, enkle CRUD-operasjoner, har vi heller ikke sett det nødvendig å teste i mye større grad. SQL spørringene er skrevet i azure sin portal og blitt testet der, for å så bli kopiert inn i vår kode. Vi har valgt å skrive spørringene i azure først da de i kontrollene blir skrevet som en string mellom to "apostrofer". Dette gjorde at vi ikke fikk feilmeldinger på SQLen før vi kjørte metoden med selve spørringen i applikasjonen - en svært tidkrevende fremgangsmåte.

Da vi senere i prosjektet opprettet koden som satt inn data i alle tabellene i databasen, den essensielle oppgaven ved systemet vårt, ble også denne funksjonen testet manuelt. Ved å kjøre metoden mange ganger, med forskjellig inputverdi, fikk vi verifisert at riktige datafelter fikk riktig verdi. I databasen er info-tabellene bestående av Stage1-, 2, 3 og 4-ID, alle med *not null* verifisering - alle stagesene måtte være fylt for at en info rad kunne bli opprettet, og også dette ble testet.

For å teste metodene med unit-test, kunne vi ha opprettet metoder som la til data i lister i stedet for direkte i databasen. Det ville gitt oss muligheten til å teste opprettelse av x-antall rader for å se at eksempelvis alle ID-verdiene holdt riktig verdistigning (vi har definert *auto_increment* i databasen på disse verdiene).

3.5 Teknologivalg

3.5.1 ASP.Net

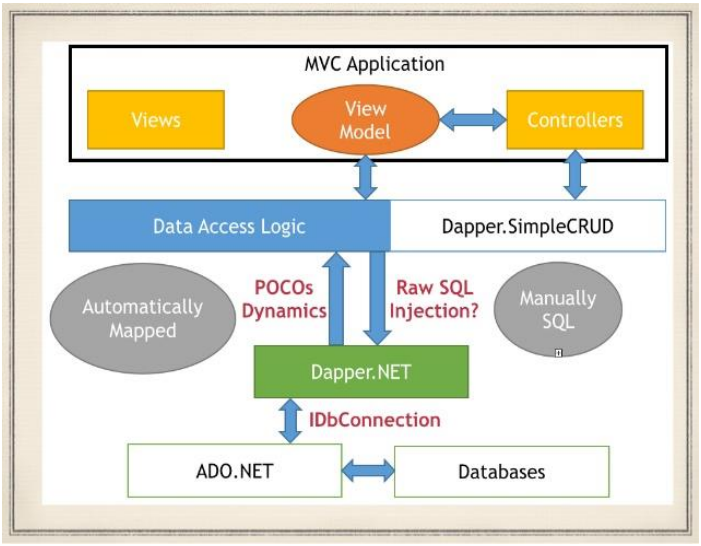
ASP.Net MVC rammeverket benytter en arkitektur som fordeler applikasjonen i tre deler, ***M**odel, **V**iew og **C**ontroller.* C# brukes for å skrive serverside-koden, CSHTML og CSS for klientside kode. Model-delen tar for seg logikken rundt henting og lagring til og fra databasen, samt oppsett og operasjoner av modellene. View-delen er hvor brukergrensesnittet i applikasjonen blir utformet. Med MVC blir Views skrevet i CSHTML og blir laget fra datamodellene. Det er i controllerne at funksjonene blir kjørt, brukerinteraksjoner og valg av Veiws blir definert og gjennomført her. Samspillet i MVC modellen er at controlleren kjører en SQL-spørring, verdiene som blir funnet blir hentet eller lagret i modellen som videre blir fremvist i et view (Dersom metoden i kontrolleren er en *getmetode*, ved *postmetode* vil SQL-spørringen gjerne være brukerinput, og modellen vil gjennomføre *post*-forespørsel (insert/delete/edit)) (Microsoft, u.d.)

ModelView kan også opprettes og benyttes. Dette er modeller som vi selv oppretter for å binde flere modeller sammen slik at vi kan gjennomføre flere operasjoner i flere tabeller i et kall. Vår applikasjon benytter i størst grad kun ett ViewModel, *InfoViewModel*, for å gjennomføre metodene. Dette kommer av at vi, sammen med oppdragsgiver, ønsker å legge til/hente data fra alle tabeller på likt i ett metodekall. Når en hjelpetekst opprettes legger systemet inn de metatags som blir lagt til av bruker → Hjelpeteksten legges inn i Stage4 (Label) som brukeren oppretter → Den nylig opprettede Stage4 blir lagt inn i Info-tabellen → Bruker velger så Stage1(Land), Stage2(System) og Stage3(Kategori) og alle disse blir lagt inn i samme rad som den nevnte Stage4 i Info-tabellen. Med vår InfoViewModel kan vi gjennomføre alle disse kallene i en metode.

Etter mye trøbbel med databasetilkobling sammen med Angular, fant vi sammen med oppdragsgiver ut at vi skulle gå vekk fra dette rammeverket. MVC var et annet forslag oppdragsgiver anbefalte for oss. Angular var i tillegg nytt for mange ansatte ved NorgesEnergi, hvorav MVC var et rammeverk som flere kunne bistå oss med, og dette gjorde

at vi byttet over til MVC. Dette ville på like linje med Angular kunne utføre de funksjonene vi hadde definert i MoSCoW-tabellen. Da vi gikk vekk fra Angular gikk vi også vekk fra Typescript og tilbake til C#.

C# er et objekt orientert programmeringsspråk med mange likheter til Java sin syntaks, og gjorde at gruppen hadde noe forhåndskunnskap fra tidligere emner om språket. C# er utviklet av Microsoft for å brukes sammen med .NET programmering (Wikipedia, 2018) som medførte at vi ikke kunne velge hvilket språk vi skulle programmere serverside-koden i.



Figur 11:MVC med Dapper

3.5.2 Dapper Framework

Dapper er en NuGet[1] bibliotek for utvidelse av databasekall funksjoner og var også noe som ble anbefalt å bruke av vår oppdragsgiver. Med Dapper skriver vi SQL-koden vi ønsker å utføre, og bygger modellene vi tar i bruk ut fra de verdiene vi vil hente.

[1] *"NuGet is the package manager for .NET. The NuGet client tools provide the ability to produce and consume packages. The NuGet Gallery is the central package repository used by all package authors and consumers."* (Nuget.org, u.d.)

```
[HttpGet]
public ActionResult Details(int id)
{
    var obj = conn.Query<helptext>("SELECT * FROM helptext WHERE helptext_ID =  @text_ID", new { text_ID = id });

    if (obj != null)
    {
        helptext model = new helptext();
        model.helptext_ID = obj.FirstOrDefault().helptext_ID;
        model.helptext_header = obj.FirstOrDefault().helptext_header;
        model.helptext_short = obj.FirstOrDefault().helptext_short;
        model.helptext_long = obj.FirstOrDefault().helptext_long;
        return View(model);
    }
    return View();
}
```

Figur 12: Utklipp fra en «Details»-metode, hvor vi bruker Dapper til å hente alt (*) fra tabellen helptext, og videre bygge opp en helptext-model basert på de verdiene som blir hentet med den ID som blir definert i parameteret.

3.5.3 TypeScript

Typescript ble i startfasen benyttet i vår applikasjon, sammen med AngularJS. TypeScript er et superset av JavaScript , som hjelper med å unngå flere bugs JavaScript ofte har, ved å type-sjekke koden for deg (Lasn, 2018). TypeScript som JavaScript kan kjøres på alle nettlesere. TypeScript har dynamiske variabler som gjør at de får en datatype etter at de har blitt deklarert. Eks: (var y = 5;) eller (var y = ‘’skole’’;). Vi fant ut at å bruke TypeScript i vår applikasjon var lurt for å løse noen av de tekniske utfordringene for å endre feltverdier på nettapplikasjonen uten at en «*post*» forespørsel ble sendt til serveren.

Angular fungerer som en *utvidelse* av HTML rammeverket ved å gjøre mulighetene for å opprette dynamiske *views* i webapplikasjoner lettere. Ved et angular rammeverket sortes alle modeller i egne, godt organiserte mapper med egne views og kontrollere. Det var etter anbefaling fra oppdragsgiver at gruppen tok i bruk denne fremgangsmåten da de mente dette ville være en god erfaring for oss å ta med videre. Angular fungerte veldig effektivt til å opprette enkle modeller og tilhørende views, men det var da vi skulle opprette databasetilkobling mot våre kontrollere at det oppstod komplikasjoner. Etter mye prøvelser kom vi ikke frem til hvordan koblingen til databasen skulle settes opp, og vi gikk derfor vekk fra dette rammeverket og i stedet benyttet fremgangsmåter vi hadde bedre kjennskap til.

Etter et møte med oppdragsgiver kom vi frem til at angular ikke var en nødvendighet, og kom til enighet om at MVC var bedre iht. våre ferdigheter.

3.5.4 Visual Studio

Visual Studio fra Microsoft er utviklingsprogrammet vi har valgt å programmere i. Dette var et enkelt valg, da NorgesEnergi bruker dette programmet hos dem. Vi valgte å bruke det samme da det var enklere å få hjelp hvis vi trengte dette. ASP.NET er utviklet av Microsoft, som også er en grunn til å bruke Microsoft sitt Visual Studio program, da disse samarbeider bra med hverandre. I Visual Studio har vi også brukt SQL Server Object Explorer, som er et verktøy som lar oss enkelt legge til/slette og/eller endre tabeller i databasen.

Det er viktig å legge til at SQL Server Object Explorer ikke finnes til Mac per dags dato. Da vi prøvde å installere dette på Mac fikk vi ikke muligheten til å bruke denne utvidelsen. Vi gikk derfor over til at alle i grupper brukte Windows maskiner under hele prosjektet.

3.5.5 Azure

For databasen vår valgte vi også et Microsoft produkt, **Azure**. Azure har et gratis alternativ for studenter ved UiA som fungerte tilstrekkelig for oss under dette prosjektet, i tillegg finnes det også ekstra utvidelser for oss IT-studenter. Azure var enkelt og koble sammen med Visual Studio, da dette kunne gjøres med et enkelt klikk fra Azure sin egen nettportal. Fordelen med Azure er at databasen alltid er online, og vi kunne jobbe med applikasjonen uansett hvor vi var, så lenge vi hadde internett. Dette gjorde oss mer fleksible, slik at vi ikke trengte å jobbe på kontoret hver dag, men kunne jobbe på skolen etter forelesninger eller hjemmefra. Samtidig var det anbefalt fra oppdragsgiver da de også brukte Azure i sine eksisterende systemer og dermed kunne bistå med eventuell hjelp.

3.5.6 Google Disk

For å enklere holde kontroll på alt av dokumenter, backlogs, rapporter m.m, valgte vi å bruke Google Disk. Her har vi laget en egen mappe for vår gruppe hvor alle kan legge til å redigere dokumentene vi jobber med. Det er viktig å ha kontroll på dokumenter i et stort prosjekt, sånn at man ikke mister eller glemmer hva som skal gjøres. Google disk er fint å bruke da det

støtter Google.docs og .sheets, som igjen tilrettelegger for at alle kan jobbe i samme dokument samtidig. Dette gjør også, som Azure, at vi kan jobbe fra hvor som helst ettersom alle dokumenter ligger i "skyen".

3.5.7 SQL

Da vi valgte Azure som vår skyløsning for databasen, ble det en SQL database som var det eneste alternativet vårt innenfor Azure. Dette mente vi også var det beste valget, da vi alle har hatt om SQL i databasefaget vi hadde i 3.Semester, IS-202 videregående databasesystemer.

Alle produktene vi har valgt å bruke er Microsoft produkter, disse produktene gjør koblinger og arbeid mye enklere. Vi har også gjennom studiet fått våre valgte Microsoft produkter gratis gjennom Onthehub.com

3.6 Kommunikasjon og verktøy

I store prosjekter er kommunikasjon en viktig del av arbeidet, derfor var det viktig for oss å bruke et felles verktøy der alle medlemmer kunne kommunisere med hverandre. Facebook var førstevalget siden alle er vant med å bruke det. Det var også viktig for oss å kunne kommunisere med veilederne våre fra NorgesEnergi og UIA, for dette formålet ble mail brukt som kommunikasjonsverktøy, der alle medlemmer i gruppen ble lagt til som Cc. På denne måten fikk alle tilgang på epostene som gikk frem og tilbake mellom gruppen og veilederne.

Under selve designdelen har spesielt funksjonslisten vært til nytte for videre implementasjon. Her har vi selv bestemte vanskelighetsgrad på gitte funksjoner og estimert tidsbruken i sprintene basert på dette. Vi har også forsøkt å definere hvorvidt en funksjon er Post- eller Get-metode, da dette definerer returtype. Sekvensdiagramet har vi, i samarbeid med klassediagramet, brukt for å skrive SQL-spørringene. Sekvensdiagramet har vært til stor nytte når vi ser hvilke tabeller et objekt går gjennom i en metode. Fremgangsmåten ved bruk av Facebook har som forventet; Det er både effektivt, raskt og et godt verktøy.

4.0 Prosjektgjennomføring

I denne seksjonen av rapporten tar vi for oss hvordan vi har jobbet med prosjektet. Vi har valgt å bruke Scrum metoden når vi har erfaring med den fra tidligere prosjekter og gruppemedlemmene er positive til en mer agil arbeidsform. Denne metoden inneholder blant annet en del planlegging og en del møter forbundet med metoden, for å sørge for at alle i gruppen er samkjørte. Vi sitter ofte sammen på NorgesEnergi og jobber tett flere dager i uken, dette sørger også for at alle er oppdatert med hva alle jobber med og hva som er neste steg for progresjon i prosjektet. En annen viktig del av Scrum metoden er at prosjektet blir delt opp i flere runder eller "sprinter", hvor vi i hver sprint ser på tidligere arbeid og hva som skal gjøres i neste runde.

4.1 Metodikk

Vi har brukt en agil fremgangsmåte i Scrum. Scrum metoden går ut på å ta for seg små deler av prosjektet og få disse til å fungere før man bygger videre på det som er fullført. I tillegg er det en del planlegging underveis og flere møter per runde som sørger for at alle medlemmene i gruppen skal være samkjørte.

4.1.1 Prosjektstyring

Vi har benyttet oss av burndown-chart til å holde kontroll på oppgave- og tids administrering. Oppfølging av estimert tidsforbruk og faktisk tidsforbruk per oppgave blir registrert her, samt at hver oppgave blir lagt til. Oppgavene blir videre delegert innad i gruppen i et eget dokument for å vise status og kunne bidra med oppfølging. For arbeid registreres også timer, oppmøte og eventuell frafallsårsak i et eget dokument for å opprettholde oversikt og dokumentasjon av arbeid.

I forhold til tidsestimering av oppgavene har vi i gruppen lite forkunnskaper om oppgavene som skal utføres. Mye av tidsforbruket er bestående av forskning for å bygge kompetanse for å kunne gjennomføre oppgavene til vårt prosjekt. Dette medfører også at den estimerte tiden på enkelte oppgaver kan være langt unna den faktiske tidsbruken.

Oppgavene blir prioritert ut fra gruppens MoSCoW tabell, de viktigste oppgavene blir forsøkt utført så tidlig som mulig. De viktigste operasjonene blir definert i MoSCoW-tabellen i samsvar med brukerhistoriene.

Planleggingen skjer hovedsakelig ved hver ny sprint. Da blir nye oppgaver definert, samt at oppgaver som ikke ble fullført ved forrige sprint blir lagt til på ny. Synspunkter rundt forrige sprint blir tatt opp og diskutert dersom endringer er ønskelig/nødvendige. Ettersom gruppen er bestående av et oddetalls medlemmer, *kan* alle avgjørelser bestemmes ved *flertallet-bestemmer* tankegangen. Vi stiller relativt likt i forhold til forkunnskaper slik at avgjørelser er mest rettferdig når flertallet er enig. Dersom det blir diskutert sterkt for eller imot en avgjørelse basert på kunnskap, vil dette bli tatt opp med arbeidsgiver som vil den endelige avgjørelsen.

4.1.2 Styringsgruppe

I løp av bachelor perioden vil det bli gjennomført tre styringsgruppemøter, disse møtene er for at prosjekteier og veileder blir oppdatert underveis i prosjektet. Styringsgruppemøte skal være akkurat som et styringsgruppemøte i et vanlig prosjekt, der utviklingsteamet(vi) diskuterer utfordringer med prosjekteieren og veilederen for rådgivning, samt sikring av kvalitet i prosjektet.

4.2 Bruken av Scrum

Agile er en generell filosofi knyttet til programvare produksjon og Scrum er en implementering av den filosofien som er spesielt knyttet til prosjektledelse. Scrum er det vi kaller et bestemt rammeverk for planlegging, administrasjon og kontroll av et prosjekt i fleksibel programvareutvikling. Ved å holde daglige møter, vet hele gruppen hele tiden hvem som gjør hva på prosjektet, som igjen sørger for mindre forvirring og misforståelser.
Gruppen vår har ikke satt en spesifikk Scrum Master siden vi fra start av har jobbet så tett at det ikke har vært behov for dette. Dette er en funksjon som ofte kreves i større prosjekter med flere team medlemmer, men med tanke på at vi bare er 3 i teamet og forholder oss til kun en produkteier og en fra ledelsen har vi ikke sett behovet for en overordnet prosessbehandler.

Siden vår prosjektbeskrivelse er tydelig vet vi hva som må gjøres og prøver å planlegge rundene tilsvarende hvor vi kommer tilbake til planleggingsfasen etter hver runde, eventuelt flere ganger i en runde, da vi er en liten gruppe på tre personer hvor diskusjonen i de fleste tilfeller fører til progresjon. Ved å arbeide på denne måten kunne vi enkelt tilpasse oss uforutsette problemer og er tilpasset GitHub-arbeidsprosessen, da vi kan gå tilbake til tidligere versjoner hvis noe skulle skje.

Før vi iverksatte bruken av Scrum ble det også vurdert om vi skulle ta i bruk Waterfall. Her deles arbeidet inn i åtte ulike steg hvor vi tar oss stegvis gjennom prosjektet. Vi vet blant annet at dette innebærer en mye lengre planleggingsfase hvor kodingen ikke starter før to til fire faser allerede er gjennomført. Vi så raskt at prosjektet vårt krevde en del kompetansebygging og muligheten for å gå tilbake til tidligere versjoner er viktig da vi prøver oss mye fram, så denne arbeidsmetoden egnet seg dårlig for vårt prosjekt.
Vi vurderte også å gå for en annen agile form for rammeverk med Kanban, men siden gruppen har brukt Scrum tidligere og har gode erfaringer med dette føltes det naturlig å ta dette i bruk.

4.2.1 Sprint Oversikt

Tabell 4: Sprint oversikt

Sprint Oversikt	***Tidsløp***	***Hovedpunkter***
Pre-sprint	*15.01 - 28.01*	Valg av teknologi og etablering i NorgesEnergi sitt lokale
Sprint 1	*Sent januar + Februar*	Etablere en database med server forbindelse og kunne hente informasjon fra databasen ved bruk av Angular.
Sprint 2	*Mars*	Etablere en ny database med serverforbindelse og kunne hente informasjon fra databasen, samt påbegynne ny struktur av API, MVC og opprettelse av CRUD operasjoner.
Sprint 3	April	Kompetansebygging og oppsett av API. Funksjon for å sette inn data i alle tabeller i et metodekall.
Sprint 4	*Mai*	Implementere en søkefunksjon

4.2.2 Pre-Sprint: Planleggingsfase og analyse av systemer

Periode - 15.01.2018 - 28.1.2018

Mål for sprinten: Valg av teknologi og etablering i NorgesEnergi sitt lokale

Denne fasen er for å bygge kompetanse før vi kommer i gang med prosjektet. Pre-sprinten forekommer bare en gang før første sprint og er en innledende planleggingsfase. Vi laster ned Visual Studio Enterprise gjennom onthehub, som fakultetet har tilgang til. Vi måtte også sette oss inn i C#, HTML og CSS som vi har benyttet i forskjellige grad gjennom prosjektet. Under pre-sprinten gikk det meste av tiden ut på å bli bedre kjent med oppdragsgiver og andre ansatte. Som avtalt i planleggingsmøte gjorde vi oss kjent med programvarene og språkene vi skulle ta i bruk. Oppdragsgiveren vår er Trond Arild Wahlstrøm som er IT-manager hos NorgesEnergi, sammen med front-end utvikler Eyvind er de kontaktpersonene våre for prosjektet.

Det NorgesEnergi ville ha var en læringsportal for sine ansatte, hvor vår oppgave var å konstruere et hjelpesystem som kunne gi svar på ulike spørsmål ansatte måtte ha ut i fra hvor de befant seg (Norge, Sverige, Finland). Systemet skulle også vite hvilket system den ansatte tilhørte og hva det var den ansatte ønsket hjelp til. Mye mer enn dette fikk vi ikke vite før det første planleggingsmøte med oppdragsgiver like før vi startet første sprint.

4.2.3 Sprint - WinForms

Periode - Sent Jan + Februar

Mål for sprinten: *Etablere en database med server forbindelse og kunne hente informasjon fra databasen ved bruk av Angular.*

Når vi gikk inn i første sprint gikk vi fra planleggingsfasen over til utviklingsfasen. Vi skrev ned definisjoner og kravspesifikasjoner for systemet og utarbeidet brukerhistorier ved hjelp av disse. Mye av tiden gikk også til kompetansebygging rundt de forskjellige programmeringsspråkene, hvor målet for sprinten var å skissere tabeller og forskjellige funksjoner på en virtuell side knyttet til vår database. Dette gjaldt også oppsett av datasett og tanker rundt API-design.

Før planleggingsmøte for andre sprint hadde vi kommet fram til at det var Visual Studio og SQL som skulle bli brukt til programmering, samt at vi brukte Azure som serverløsning. Det som derimot ikke var avklart før neste møte, var hvordan prosjektet skulle bygges opp og vises fram. Vi hadde begynt å lage en Windows Form løsning med en gang vi begynte med utviklingen av programmet. Dette var det vi hadde brukt da vi lagde et system tidligere i studiet under emnene IS-200, 201 og 202. Dette endte med at vi måtte skrape den første Windows Form løsningen vi hadde påbegynt ettersom dette ikke egnet seg for dem, men førte til at vi kom raskere i gang med neste og siste løsningen av prosjektet.

4.2.4 Sprint - .NET MVC

Periode - Mars
Mål for sprinten: Etablere en ny database med serverforbindelse og kunne hente informasjon fra databasen, samt påbegynne ny struktur av API, MVC og opprettelse av CRUD operasjoner.

Under første planleggingsmøte for andre sprint kom det fram at Windows Forms ikke var en god nok løsning for prosjektet. Vi kom her fram til mye av den nye strukturen og oppsettet for neste løsning, men før selve programmeringen for denne sprinten kom i gang måtte vi sette oss inn i bruk av Angular. Dette var litt tidkrevende, men vi lærte å ta i bruk nye verktøy og metoder for prosjektbygging i VSE som kom godt til nytte. I sluttfasen av denne perioden ble også oppbyggingen av de ulike stegene inne i prosjektet vårt definert. Etter endt møte med Trond og Eyvind kom vi fram til at den beste måten å finne ut hvor man var og hvor ting skulle legges til i prosjektet skulle stegene deles inn i fire nivåer.

Dette var vårt første review meeting hvor vi den faktiske oppbyggingen av prosjektet hadde påbegynt. Angular var tidkrevende å sette seg inn i, men det var også spennende å se på den nye funksjonaliteten som kom frem. Senere det ble dette rammeverket skrapet da det ble vanskelig å få koblet til databasen ved bruk av Angular. Det var her vi fant ut at MVC var det beste rammeverket å bruke for å få til våre oppgaver.

Når vi skulle endre alle våre controllere til MVC, i stedet for Angular som vi hadde, hadde vi laget et ganske stort tidsestimat. Dette estimatet var veldig feil, da endringer gikk mye

enklere enn vi selv trodde. Vi hadde estimert en hel uke på å få endret alle controllerne med i virkeligheten så brukte vi under en dag på denne endringen. Dette viser at vi ikke har tidsestimatet på hvor store oppgaver egentlig er, men det er noe som har blitt bedre og bedre for hver sprint som har vært og ettersom vi har fått satt oss mer inn i teknologien og språkene vi bruker.

4.2.5 Sprint - Hjelpetekst

Periode - April
Mål for sprinten: Kompetansebygging og oppsett av API. Funksjon for å sette inn data i alle tabeller i et metodekall.

Når vi satt i gang med tredje sprint var vi endelig innforstått med hvilken løsning prosjektet skulle ta i bruk. Dette ledet til mer strukturert arbeid som igjen sørget for høyere effektivitet. Samtidig som vi nå var på rett vei, var det mye som måtte forbedres. En del av tiden gikk til å passe på at mapper, filer og at selve koden så ryddig og fin ut. Nå som prosjektet begynner å ta form har vi også påbegynt implementeringen av de ulike funksjonene hentet ut fra brukerhistoriene, som det å kunne legge til, redigere og slette hjelpetekster. Vi lagde CRUD funksjoner til de forskjellige stadiene og jobbet med å kunne søke, legge til og hente ut disse korrekt. Vi måtte også oppdatere alle MCV stages og endre databasen i henhold til segmentet.

Det var samtidig i denne sprinten at kompleksiteten i prosjektet økte. Å implementere enkle, selvstendige CRUD-operasjoner gjøres ved hjelp av lette veiledninger på internett, men gruppen fant store utfordringer når data skulle settes inn i flere tabeller samtidig i et metodekall. Dette var en veldig tidkrevende oppgave, og mye av tiden gikk med på å løse nettopp dette. Det påvirket en del av arbeidsfordelingen i prosjektet, hvor det kom tydeligere frem hvem som ville fokusere på programmering og hvem som ville fokusere på rapport. Vi tok også kontakt med oppdragsgiver i denne fasen og fikk input fra dem rundt hva vi kunne gjøre for å løse opp i knuten.

Under denne sprinten hadde gruppen eksamen i IS-305, forvaltning av IT-ressurser, dette tok selvsagt opp en del av gruppens tid.

4.2.6 Sprint - Søkefunksjon

Periode - Mai

Mål for sprinten: Implementere en søkefunksjon

Før vi gikk inn i fjerde og siste sprint for prosjektet vårt hadde vi et grundig planleggingsmøte hvor vi kom fram til at vi hadde fullført to av våre 'Must have' funksjoner i prosjektet. Det som gjenstod var å implementere 'Must have' funksjonen for en søkefunksjon. Vi hadde også mye igjen å gjøre på rapporten før den tilfredsstilte kvaliteten vi selv ønsket. Dermed gikk mye av tiden til å ferdigstille denne. Metoden for søk etter metatags ble gjort ferdig og funker nå som det skal.

Det siste vi har å gjøre etter at rapporten blir levert er å sette opp et API prosjekt i vår løsning. Dette gjør at NorgesEnergi kan sende sine spørringer som en string til vår metode, for så å få tilbake relevant informasjon, i henhold til deres søk, returnert fra databasen. Denne oppgaven planlegger vi, og regner med, at skal bli ferdig før vi leverer prosjektet til arbeidsgiver. Dette er den siste oppgaven som vil gjøre at prosjektet vil fungere i sin helhet, som et "beta" produkt, til det nye systemet som oppdragsgiver holder på å utvikle.

4.2.7 Sprint retrospekt

I etterpåklokskap skulle vi tidligere avtalt møte med oppdragsgiver hvor det ble stadfestet hvilke programmer som skulle brukes. Likevel fungerte Windows Form som en prøveversjon for oss før vi gikk løs på den riktige oppbyggingen som gjorde at vi var bedre kjent med VSE, SQL og Azure når vi påbegynte korrekt løsning. Vi kom raskt i gang når semesteret startet og ser ikke på dette som noe som preget prosjektet negativt.
Videre ser vi blant annet at vi kunne fulgt Scrum metodikken ivrigere, da en del av tiden gikk til kompetansebygging underveis i programmeringen. Vi ble mot slutten av prosjektet litt dårlige på å føre inn oppgaver med estimert og anvendt tidsbruk. Dette var noe vi var dyktige på i starten, men når vi kom ut i tredje og fjerde sprint merket vi at det ble vanskeligere å holde styr på alle oppgavene i backloggen. Flere av funksjonene var komplekse og krevde mye tid både rundt kompetansebygging og selve kodingen. Selv om

oppgavene var veldig komplekse, klarte vi ikke på en god måte å fordele dem slik at vi kunne utvikle dem på likt. Vi endte derfor opp med «Pair programming» i store deler av de siste sprintene.

Burndown chart og backlog er nyttige verktøy som kunne blitt mer iherdig brukt, men dette var allikevel ikke noe som påvirket arbeidet underveis. Vi ser i ettertid at det kunne ført til en mer ryddig og effektivisert gjennomgang av de forskjellige arbeidsoppgavene.

5 Refleksjon

5.1 Arbeidsfordeling

Tidlig i prosjektfasen hadde vi en veldig jevn arbeidsfordeling. Alle i gruppen deltok jevnt og trutt både på programmering og på rapportskriving. Etterhvert som programmeringsfasen ble mer avansert og tidkrevende, begynte vi på en naturlig måte å fordele oppgaver. Eirik tok i større grad over ved rapportskriving og Marius tok ansvar for programmering, hvorav Benjamin hadde innspill i begge prosesser. Denne fordelingen kommer også av våre egne interesser, Marius ønsker eksempelvis å arbeide videre med programmering, mens Benjamin og Eirik ønsker å gå en annen retning.

Vi har heller ikke hatt en klar SCRUM-master i arbeidet vårt. Som vi tidligere har nevnt har vi arbeidet ved NorgesEnergi's kontorlandskap. Dette har hele veien skapt god kommunikasjon og mye rom for diskusjon, og vi har ikke følt det nødvendig å benytte en slik leder. I tillegg har ingen av oss verken mer eller mindre forhåndskunnskaper rundt et slikt prosjekt, så det ville for oss vært unaturlig at noen skulle kunne bestemme mer enn andre. Derimot, om noen av oss har hatt formeninger om noe kode har dette blitt lagt frem på en god måte for å gjøres forståelig for hele gruppen. Viktige avgjørelser har vi tatt opp med oppdragsgiver før noe har blitt vedtatt.

Selv føler vi arbeidsfordelingen har vært i god iht. til prosjektstørrelse og forhåndskunnskaper. Selv om det har vært utfordringer både ved programmering og rapportskriving, er det ingen i gruppen som angrer på at en selv eller andre tok mer ansvar for ledelse.

5.2 Erfaringer

Gruppen sitter igjen med mange nye ferdigheter. Det har vært enormt lærerikt å kunne arbeide med et prosjekt som gir nytte for en oppdragsgiver, samt å arbeide sammen med andre fagpersoner. Selv om vi har hatt en arbeidsfordeling, da spesielt rundt

programmering, har allikevel alle på gruppen fått økt sine kunnskaper i flere programmeringsspråk.

I tillegg har vi fått ny erfaring innen det å starte *fra scratch* i et prosjekt. Oppdragsgiver hadde sine tanker og ønsker, og vi har selv fra bunnen produsert noe av verdi. Det å jobbe i et team vil være en god og meget verdifull erfaring å ta med videre i arbeidslivet.

Det har også vært en erfaring å skrive en slik rapport basert på prosjektet som utføres, og ikke kun etter en gitt mal slik det ble gjennomført i IS-200, tredje semester 2016. Som vi fikk tilbakemelding på av veileder liknet rapporten vår tidligere på noe som skulle leveres i nettopp IS-200, da vi hadde skrevet mye teori og tabeller rundt mange temaer som ikke var relevante for vårt prosjekt. Det å selv komme frem til hvilke tabeller og metoder som er nødvendige for et gitt prosjekt vil være tidsbesparende og samt gi ansiennitet ved andre, liknende prosjekter.

5.3 Samarbeid med oppdragsgiver

Samarbeidet med arbeidsgiver har vært veldig bra og de har hjulpet med alt vi har lurt på. Siden vi har hatt kontorplass på deres IT-avdeling, har de bare vært et lite skritt unna hvis det var noe vi lurte på. Vi har kunne spurt om alle spørsmål, alt fra database, front end, back end og generelt spørsmål som omhandlet prosjektet.
Det er heller vi som har vært for dårlige til å spørre om hjelp når vi har stått fast med noe. Vi som gruppe skulle ha vært flinkere til å spørre når vi har stått fast, vi har alltid prøvd å finne løsning på problemene våre (og lykkes) istedenfor å spørre. Dette har nok gjort at vi har brukt en del lengre tid på enkelte oppgaver enn det vi egentlig hadde trengt å gjøre.
Vi har allikevel hatt god dialog med oppdragsgiver, blant annet ved at vi har oppdatert dem jevnlig, gjerne annenhver uke.

5.4 Det vi kunne gjort annerledes

Vi sitter igjen med en del tanker over hva vi kunne ha gjort annerledes i prosjektet. Det første vi ser er at vi skulle hatt et bedre møte med arbeidsgiver med en gang prosjektet startet. På denne måten kunne vi ha unngått de problemene vi støtte på i sprint WinForms, og ha begynt å lage en webapplikasjon med en gang i stedet for en Windows Form løsning. Kontakten med veileder kunne også ha vært bedre i starten av prosjektet, vi tok seint kontakt for veiledning og kunne nok fått mer hjelp i starten av selve prosjektet. Dette gjelder både med oppsetting av styringsgruppemøter og hvordan vi burde gått frem med oppgave og rapport.

Hvis vi skulle gjennomført prosjektet en gang til, skulle vi gjerne vært bedre til å forhøre oss med oppdragsgiver når vi trengte hjelp. Selv om den arbeidsfordelingen vi har benyttet har fungert for oss i vårt prosjekt, kunne vi sannsynligvis vært bedre på at alle hadde fått bidratt likt på alle oppgavene i prosjektet.

Vi skulle også gjerne ha gjennomført det første styringsgruppe- og veiledningsmøte tidligere i prosessen. Dette ville gjort at veileder tidligere fikk innblikk i hva vi skulle utvikle og dermed kunne bidra til hvilke fremgangsmåter og tabeller/diagrammer som kunne komme til nytte. I tidligere evalueringer av rapporten kom det som nevnt frem at vi hadde mye unødvendigheter med, som vi hadde lært om i tidligere emner. I selve programmeringsfasen kunne vi gjerne benyttet veileder i større grad.

5.5 Det som gjenstår

Tabell 5: Utklipp av backend

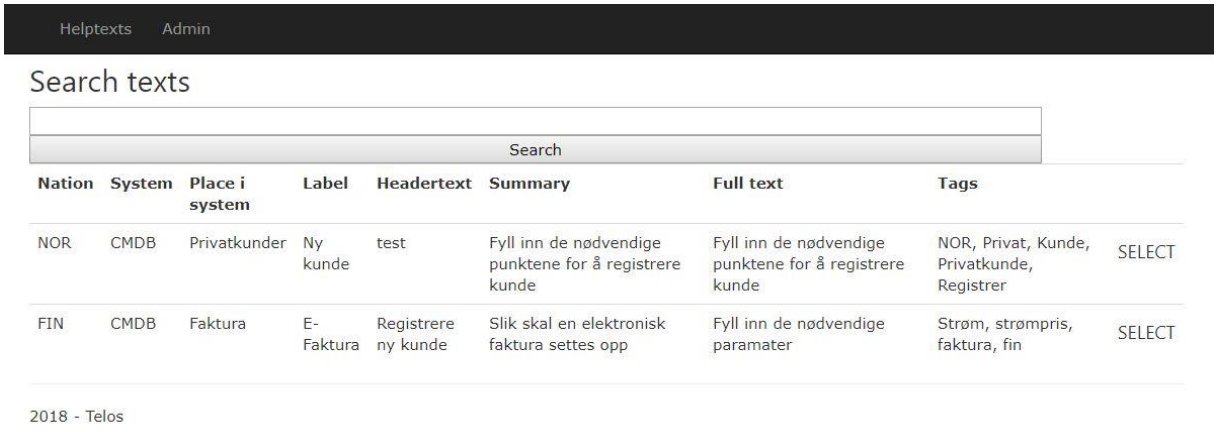
Oppgaver	Deadline	Utførende	Ansvarlig	Dato fullført
Det som gjenstår				
Funksjon for å ta i bruk user data (Token)				
API				
Smart funksjon, applikasjonen vet hvor brukeren befinner seg i systemet når brukeren spør om hjelp				
Redigere metatags i "edit"-metode og view				
Bilder i databasen				

Som vi ser av Tabell 5, er det enda en del oppgaver som gjenstår:

- **Funksjon for å ta i bruk user data (Token)**
    - Ønsket var et smart system, som automatisk skulle vite hvor brukeren befant seg. Vi fant sammen med oppdragsgiver at dette ble for komplekst. I stedet skulle vi ta i bruk Token for å verifisere at brukerne som spurte om hjelp, faktisk var en bruker fra NorgesEnergi.
- **API**

    - Dette er oppgaven som vi foreløpig arbeider med i den siste sprinten. Funksjonen er per dags dato ikke helt fungerende, men planlegger å ferdigstille den før vi leverer fra oss prosjektet til oppdragsgiver.
- **Smart funksjon**
    - Som det står nevn ovenfor ble dette en for kompleks oppgave for oss å implementere. Derimot kan det være mulig for erfarne utviklere å implementere en slik funksjon.
- **Redigere metatags i «edit»-metoden og view**
    - Dette er egentlig ikke en kompleks oppgave, men heller ikke en «Must have» oppgave. Derfor ønsker vi å ferdigstille våre «Must have» funksjoner før vi går videre med dette. Hvis API går raskere enn antatt, vil denne metoden bli prøvd utført før vi leverer fra oss prosjektet.
- **Bilder i databasen**
    - Dette er heller ingen «Must have» funksjon, og blir derfor prioritert deretter. Tanken var at administrator kunne legge opp hjelpetekster i form av bilder, men tiden har ikke strukket til for at dette vil bli implementert.

6 Resultat



Figur 13: View for søkeresultat.

Designet i våre views er ikke fokusert på, dette er en oppgave vi fikk beskjed om at vi ikke skulle bruke tid på. Vår oppdragsgiver har egne UX designere som skal designe dette i likhet med deres system.

Det er fremdeles en del funksjonalitet som kan utarbeides, som nevnt i refleksjonen, blant annet ved at norske brukere kun skal få opp norske hjelpetekster. Som eksempelet i Figur 13 viser, returnerer nå søkefunksjonen vår hjelpetekster fra flere nasjoner. Allikevel har vi opprettet et view med mulighet for søk som returnerer objektene fra søket. Vi har også utviklet et view for det å opprette hjelpetekst i de riktige stegene (Figur 8).
Dette er de essensielle funksjonene i vårt system («Must have»). Ettersom disse er fullført mener vi at resultatet i sin helhet er tilfredsstillende i henhold til de forventinger og forkunnskaper vi hadde.

Mye av grunnen til at vi føler samarbeid og resultat har vært bra, kommer av at vi har fått arbeid sammen med NorgesEnergi i deres lokale. Noe så enkelt som det å ha en arbeidsplass å gå til påvirker positivt på motivasjon og effektivitet.

Som nevnt i refleksjonen er vi fornøyd med hvordan vi har arbeidet sammen som gruppe gjennom prosjektet. Det å arbeide i gruppe med et slikt prosjekt er også noe vi hadde lite forkunnskaper om, men tross utfordringer tar vi med oss gode erfaringer videre. Vi har

oppnådd vårt resultat gjennom iherdig jobbing, vi har flere ganger nevnt at våre forkunnskaper har vært få med tanke på et slikt prosjekt.

Vi har mer erfaring fra emner som tar for seg analyse- og designprosesser. Derfor var vi kjappe med å sette opp våre brukerhistorier og vi har bygd flere av diagrammene rundt disse. Programmeringsfasen har hatt en bratt læringskurve og det at vi har spesifisert våre arbeidsoppgaver til rapport og utvikling har gitt et godt resultat for sluttproduktet, men det har også gitt fordelt kunnskap mellom gruppemedlemmene.

7 Referanser

Agile Business Consortium. (2017, Juni 15). *Agile Business Consortium*. Hentet Februar 2018 fra The DSDM Agile Project Framework (2014 Onwards): https://www.agilebusiness.org/content/moscow-prioritisation

Lasn, I. (2018, Februar 28). *freecodecamp*. Hentet Mai 2018 fra TypeScript - JavaScript with superpowers – freeCodeCamp: https://medium.freecodecamp.org/typescript-javascript-with-super-powers-a333b0fcabc9

Levy, D. (2014, Oktober 14). *Gatherspace*. Hentet Mars 2018 fra Software Requirements Specification, what you NEED to know: http://www.gatherspace.com/static/use_case_example.html

mairaw, guardrex, rprouse, BillWagner, tdykstra, cartermp, . . . svick. (u.d.). *docs.microsoft*. Hentet Mai 2018 fra Unit Testing in .NET Core and .NET Standard: https://docs.microsoft.com/en-us/dotnet/core/testing/

Microsoft. (u.d.). *Msdn.Microsoft.net*. Hentet Mai 2018 fra About Processes and Threads (Windows): https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx

Nuget.org. (u.d.). Hentet Mai 2018 fra NuGet Gallery | Calcium 1.0.26: https://www.nuget.org/

Wikipedia. (2018, Mai 14). ikipedia. *C Sharp (programming language)*. Hentet fra https://en.wikipedia.org/wiki/C_Sharp_(programming_language)

8 Vedlegg

8.1 Uttalelse fra oppdragsgiver

Dette ble dessverre ikke levet til avtalt til av oppdragsgiver og er derfor heller ikke med i denne rapporten.

8.2 Backlog

Oppgaver	Deadline	Utførende	Ansvarlig	Dato fullført
Installere Visual Studio på alle pc-ene (kontoret, privat)	20.01.2018	Alle	Alle	Endt pre-sprint
Godkjenning av prosjektet	20.01.2018	Alle	Eirik	Endt pre-sprint
Kompetansebygging C#	01.02.2018	Alle	Benjamin/Marius	Endt pre-sprint
Sette opp database i Azure	01.02.2018	Benjamin/Marius	Benjamin	Endt pre-sprint
Rapportskriving	09.05.2018	Alle	Benjamin/Eirik	Endt pre-sprint
Connect VS Angular project to Azure Database				
Create ConfigurationManager File	09.02.2018	Marius	Marius	Utgikk 10.02.2018
Kompetansebygging for Dapper	12.02.2018	Alle	Alle	Endt sprint1

Create controller for each Model files	15.02.2018	Alle	Marius	25.01.2018
Create file for SQL Queries	16.02.2018	Benjamin/Marius	Marius	16.01.2018
Create test data	16.02.2018	Marius	Marius	Utgikk 17.01.2018
View/Edit tables	23.02.2018	Alle	Marius	30.01.2018
Create models for tables	23.02.2018	Benjamin	Benjamin	13.01.2018
Endringer iht. styringsgruppemøte				
Endre databasen iht styringsgruppemøte	28.02.2018	Marius	Marius	27.02.2018
Endre data-mappen til model-mappe, fikse refaransefeil	28.02.2018	Eirik	Eirik	27.02.2018
Kompetansebygging for API	05.03.2018	Alle	Benjamin	Endt sprint 3
Lage nytt prosjekt under samme solution	12.03.2018	Marius	Marius	27.02.2018
Endre dappertestcontroller-filnavn til categorycontroller	16.03.2018	Benjamin	Benjamin	27.02.2018
1. Oppdatere MVC iht sgm	23.03.2018	Alle	Alle	27.02.2018

1.1 Oppdatere stage 1 MVC	23.03.2018	Eirik	Eirik	27.02.2018
1.2 Oppdatere stage 2 MVC	23.03.2018	Eirik	Eirik	27.02.2018
1.3 Oppdatere stage 3 MVC	23.03.2018	Eirik	Eirik	27.02.2018
1.4 Oppdatere stage 4 MVC	23.03.2018	Eirik	Eirik	27.02.2018
1.5 Oppdatere info MVC	23.03.2018	Benjamin	Benjamin	27.02.2018
1.6 Oppdatere helptext MVC	23.03.2018	Marius	Marius	27.02.2018
1.7 Oppdatere metatag MVC	23.03.2018	Marius	Marius	27.02.2018
1.8 Oppdatere helptextTag MVC	23.03.2018	Marius	Marius	27.02.2018
Opprette hjelpetekst				
Opprette metode for å gjøre flere å sette inn i flere tabeller i databasen i et kjøring	31.04.2018	Marius/Benjamin	Marius	27.04.2018
Opprette tilhørende view for nevnt metode	31.04.2018	Marius/Benjamin	Marius	28.04.2018
Opprette søkefunksjon				

Opprette metode for å kunne søke etter hjelpetekst	25.05.2018	Marius	Marius	
Opprette tilhørende View for nevnt metode	25.05.2018	Marius	Marius	