Microsoft Teams-applikasjon for integrasjon mot fakturasystemet Semine

IS-304: 2021

Tittel:

Emnekode:	IS-304
Emnenavn:	Bacheloroppgave i informasjonssystemer
Emneansvarlig:	Hallgeir Nilsen
Veileder:	Hallgeir Nilsen
Oppdragsgiver:	Knowit Sør

Studenter:

Etternavn	**Fornavn**
Barbakken	Stian Eikvang Mørk
Gudevold	Ola Johansen
Lindbøl	Jørgen
Neergaard	Magnus Berg
Nesse	Kirsti Næsager
Sveggen	Markus

Vi bekrefter at vi ikke siterer eller på annen måte bruker andres arbeider uten at dette er oppgitt, og at alle referanser er oppgitt i litteraturlisten.	JA X	NEI
Kan besvarelsen brukes til undervisningsformål?	X	
Vi bekrefter at alle i gruppa har bidratt til besvarelsen	X	

Forord

Denne rapporten er skrevet som en del av emnet IS-304 Bacheloroppgave i informasjonssystemer i løpet av vårsemesteret 2021. Det har vært et spennende og lærerikt semester, hvor vi har møtt på ulike utfordringer underveis. Avslutningen på bachelorprogrammet har vært noe annerledes enn hva som først ble antatt. På grunn av en pågående pandemi har prosjektet og rapporten måtte ta hensyn til den medfølgende uforutsigbarheten. Tatt dette i betraktning sitter vi nå igjen med et vel gjennomført prosjekt, og for dette har vi flere å takke.

Takk til Tor Oskar Wilhelmsen, produkteier og veileder ved Knowit. Gjennom prosjektet har Tor Oskar vært med på å sikre klare tilbakemeldinger og forventninger til prosjektet og gruppen. Vi er takknemlige for god oppfølging og hjelp underveis i prosjektet.

Vi vil også rette en takk til James McRedmond, veileder ved Knowit som gjennom prosjektet har bistått med både tilbakemeldinger og hjelp med ulike problemer. Takk for deltagelse på møter og Sprint Reviews som har hjulpet med sikre fremgang og kvalitet i prosjektet og produktet.
Takk til Hallgeir Nilsen som gjennom prosjektet har fungert som vår veileder fra UiA. Gjennom prosjektet har Hallgeir sørget for tett oppfølging med faste møter, alltid vært åpen for å gi tilbakemeldinger og innspill, samt sørget for at gruppen har hatt mål og strekke seg etter.
Til slutt takk til Knowit Sør som har tatt seg tid til å samarbeide med studentgrupper ved UiA, samt sørget for god opplæring og oppfølging.

Kristiansand, Universitetet i Agder, 13. Mai 2021

Jørgen Lindbøl

Ola J. Gudevold

Kirsti Næsager Nesse

Stian E. M. Barbakken

Markus Sveggen

Magnus B. Neergaard

Sammendrag

Denne rapporten er skrevet av gruppen Transform IT som dokumentasjon på utført bachelorprosjekt i emnet IS-304 våren 2021. Oppgaven har gått ut på å utvikle en applikasjon i Microsoft Teams som integrerer fakturasystemet Semine. Produktet er ment for å benyttes internt hos vår oppdragsgiver Knowit Sør, og applikasjonen har som formål å effektivisere håndtering av fakturaer ved å integrere Semine i en Microsoft Teams app, slik at man ikke skal behøve å benytte en ekstern applikasjon for å behandle fakturaer.

For å utvikle løsningen har vi benyttet språkene C# for backend, ReactJS for frontend, og Scrum som arbeidsmetodikk. Videre har vi benyttet Microsoft Azure-miljøet for store deler av oppgaven, og en av de unike aspektene ved prosjektet har vært å få dette fungerende i Azure-plattformen. Dette er hurtig fremvoksende teknologi og har vært et ønsket fra både produkteier og gruppen å ta i bruk.

Løsningen vår er sentrert rundt to applikasjoner som kommuniserer: frontend og backend. Frontend består av brukergrensesnitt og interaksjon med sluttbruker, mens backend behandler all logikk og kommunikasjon mot databasen. På grunn av begrenset tilgang til Semine sitt eget API har vi simulert integrasjonen med Semine ved å utvikle vår egen backend-løsning.

Denne rapporten er en dokumentasjon av utviklingsprosessen og produktets status per 13. mai 2021, og produktutviklingen fortsetter frem mot muntlig eksaminering 2. juni 2021. Se Tillegg A for uttalelsen til Knowit Sør om prosjektets status per 12. mai 2021.

Innhold

Figurer

1 Introduksjon

I det siste semesteret som student på bachelorstudiet i IT og informasjonssystemer har vi, i anledning emnet IS-304 Bacheloroppgave i informasjonssystemer, fått i oppgave å gjennomføre et prosjekt i samarbeid med konsulentselskapet Knowit Sør. Prosjektet har som formål at vi skal bruke kunnskap og arbeidsmetodikk vi har opparbeidet oss i de foregående semestrene. Under arbeid med prosjektet skulle vi jobbe metodisk som om vi var systemutviklere for et IT-selskap.

På RefreshIT, som ble arrangert av UiA høsten 2020 med hensikt å koble grupper av studenter opp mot potensielle organisasjoner som ønsket å samarbeide i anledning bachelorprosjektet, presenterte Knowit Sør et prosjektforslag som vi valgte å arbeide med. Knowit Sør er en av landets ledende leverandører av digitaliseringsløsninger for privat og offentlig sektor. Prosjektet gikk ut på å ta i bruk Microsoft sitt rammeverk for utvikling av en Microsoft Teams-applikasjon. Applikasjonen skal effektivisere og forenkle håndtering av fakturaer for selskaper og organisasjoner ved integrering av fakturasystemet Semine i Microsoft Teams. Semine er en nyskapende AI-plattform som benytter maskinlæring til å forstå innholdet i en faktura. Basert på læring og gjenkjenning av mønster, kan Semine gjøre avanserte analyser av innholdet i fakturaen og korrekt foreslå regnskapskontoer, perioder og andre lignende spesifikasjoner ved en faktura (Semine, 2021). Per nå må bedrifter logge inn i et eksternt system for å behandle fakturaer prosessert av Semine. Ønsket var at denne prosessen skulle forenkles ved integrering i en Microsoft Teams app, slik at man kan autentiseres med sin eksisterende Microsoft konto og ikke lenger må forholde seg til en ekstern applikasjon.

Noe av det som gjorde dette prosjektet spennende og utfordrende var at konseptet med å ta i bruk en Microsoft Teams-applikasjon var relativt nytt, også for Knowit. Vi fikk dermed muligheten til å utforske ny teknologi, med en viss frihet under ansvar. En annen faktor som gjorde prosjektet spennende var ønsket om at systemet skulle være utarbeidet i Microsoft Azure, som står sentralt i utvikling av Microsoft sine tjenester. Som gruppe kom vi frem til at oppgaven og verktøyene som Knowit Sør presenterte er høyst aktuelle å lære seg for å bygge et godt grunnlag til overgang fra bachelorstudent til arbeidstaker eller student på masternivå. I tillegg til dette vurderte vi prosjektet passende da fordeling mellom database, backend og frontend var tydelig slik at vi raskt klarte å danne en idé om hvordan vi skulle jobbe med prosjektet, og ikke minst, hvordan resultatet kunne bli.

Demo av produktet

For å gi leseren en best mulig opplevelse av rapporten har vi valgt å legge ved en videodemonstrasjon av produktet. Videoen går igjennom det utviklede systemet, og har som formål og gi leseren et overordnet bilde av funksjoner i applikasjonen. Merk at demoen ble satt sammen før den muntlige fremføringen, og produktet vil derfor i noen grad bli utbedret i løpet av tiden frem til da. Man kan finne demoen her.

Oppsett for rapporten

Oppsettet for rapporten har vært et av de større temaene som har blitt diskutert innad i gruppen. Med et ønske om å klart få frem de viktigste aspektene for vårt prosjekt, og samtidig sørge for en rød tråd gjennom rapporten, endte vi derfor opp med et oppsett som beskriver dokumentasjonsprosessen ved å skille mellom valg og utførelsen av prosjektet. Gruppen starter først med å presentere sentrale valg som ble tatt for prosjektet. Dette gjelder sentrale valg innenfor prosjektstyring, teknologi samt kvalitetssikringstiltak. Her blir det også presentert begrunnelse for hvorfor vi vurderte disse valgene som gode valg. Etter de sentrale valgene begynner vi på selve utførelsen av prosjektet, hvor vi starter med å gå gjennom relevante erfaringer med prosjektstyring. Etter det tar vi for oss de mest sentrale aktivitetene vi gjennomførte i prosjektet som inkluderer initiell analyse, design av produkt og implementering. Til slutt går vi også igjennom sentrale utfordringer som vi har møtt på. Vi avslutter så rapporten med å oppsummere og reflektere over de aspektene som gruppen mener har vært viktig for prosjektet.

Vi gjør oppmerksom på at alle figurer, kapitler og vedlegg som refereres til er klikkbare lenker, slik at man enkelt kan navigere seg frem til dem.

Vi tar i denne rapporten forbehold om at leseren har generelle kunnskaper om Agil arbeidsmetodikk, samt enkelte Scrum-relaterte begreper. For ordens skyld vil det bli beskrevet hva de enkelte konseptene er for å sikre forståelse.

2 Valg og begrunnelser

Gjennom dette kapittelet vil vi gå igjennom sentrale valg vi har tatt for prosjektet og hvordan vi kom frem til dem. Målet med kapittelet er å forklare hvilke valg vi bevisst har tatt for å sikre kvalitet og fremgang for prosjektet. Kapittelet tar først for seg valg vedrørende prosjektstyring, deretter valg rundt teknologi og til slutt valg vedrørende ytterligere kvalitetssikring. Vi vil gjennom kapittelet supplere med argumenter for hvorfor vi mener de valgene vi har tatt er gode valg for prosjektet.

2.1 Prosjektstyring

Tidlig i prosjektet ble det nødvendig å ta stilling til hvordan prosjektstyring skulle organiseres og hvilken arbeidsmetodikk vi ønsket å følge for å sette et rammeverk på arbeidet. Dette er et viktig valg da det setter rammer for hvordan teamet jobber sammen og dikterer kommunikasjon med produkteier. Valget falt naturlig på Scrum, ettersom dette er en agil metodikk som vi alle kjente til fra før av og har erfaring med fra tidligere prosjekter. Videre passer Scrum godt for dette prosjektet ettersom vi hadde behov for å kunne tilpasse oss endringer underveis blant annet fordi deler av prosjektet ikke var helt avklart ved prosjektstart, og vi ønsket tilbakemelding jevnlig for å forsikre både kvalitet og retning på arbeidet. Scrum er godt tilpasset disse kravene. En agil arbeidsmetodikk, slik som Scrum, tillater hyppige endringer underveis i utviklingsprosessen. Dette er en utvikling fra mer tradisjonelle arbeidsmetodikker som fossefall der en følger en mer rigid rekkefølge som er lite tilpasset endringer (Sutherland & Schwaber, 2007). Denne arbeidsmetodikken ble også anbefalt av vår produkteier Knowit Sør, som tidligere har god erfaring med Scrum og benytter dette aktivt i sine prosjekter. Dette la et godt grunnlag for veiledning underveis dersom vi skulle ha behov for det, og gjorde det lettere å komme raskt i gang ettersom alle parter var kjent med rammeverket.

2.1.1 Oppdeling av arbeid

I henhold til Scrum ble arbeidet delt opp i Sprinter. En Sprint er en tidsperiode på maks én måned der teamet utfører bestemte oppgaver, som går etter hverandre uten opphold frem til prosjektets slutt (Sutherland & Schwaber, 2007). Etter samtaler med Knowit valgte vi å dele opp Sprintene i tre uker, da dette ga oss en god balanse mellom hyppig kommunikasjon og tilbakemelding fra produkteier samt nok tid til å ha litt større fremgang for hver Sprint. Vi kom frem til at en Sprint på fire uker kunne bli for lang, da det kunne vært fare for at vi jobbet lenge på oppgaver som viste seg å ikke stemme overens med visjonen eller kravet fra produkteier. En Sprint på fire uker ville typisk vært bedre egnet der man har større, veldefinerte oppgaver der det ligger mindre usikkerhet rundt utvikling og retning. Vi anså to uker som for kort tid til å fullføre betydelig arbeid før neste Sprint, noe som kan føre til at man har lite å vise frem på slutten av hver Sprint. Kortere Sprinter kan

være bedre tilpasset større team der man får utviklet betydelige mengder på kort tid, slik at man selv med en kortere Sprint har tydelig fremgang fra Sprint til Sprint.

2.1.2 Tidsestimering

Tidsestimering og planlegging av arbeidsmengde er essensielt i Scrum. Før man går i gang med prosjektet er det viktig å få laget et estimat på kompleksitet for å kunne beregne og prioritere bruk av tid og ressurser (Rubin, 2012).

For å gi oss et overblikk over relativ kompleksitet blant brukerhistoriene vi hadde opprettet, benyttet vi en metode ved navn *Planning Poker*. Planning Poker er en estimeringsmetode der hvert teammedlem selv estimerer en brukerhistories kompleksitet ved å velge tall fra en forhåndsbestemt skala, og man går igjennom brukerhistoriene én etter én (Mahnič & Hovelja, 2012). For hver brukerhistorie er det åpent for diskusjon, slik at man kan komme frem til enighet rundt endelig beregning.

I vårt prosjekt besluttet vi å bruke Fibonacci-tall, da dette ga oss et tydelig bilde på hvilke oppgaver som blir ansett som mer komplekse ettersom høyere tall har større sprik, og det er et av de mer populære tallsystemene for å gjennomføre Planning Poker i Scrum. Vi valgte å benytte oss av tjenesten til organisasjonen Planning Poker, som er tilgjengelig på deres nettsted (https://www.planningpoker.com), da denne tjenesten er utviklet for å gjennomføre Planning Poker virtuelt. Resultatet av gjennomført Planning Poker blir så lagt til i vår *Product Backlog* for å videre brytes ned i hver Sprint. Se resultatet fra vår Planning Poker i Figur 1.

Planning Poker - Product Backlog

Issue Key	Summary	Story Points
	Som en bruker vil jeg kunne se endringshistorikken på en faktura slik at jeg kan ha kontroll på hva som har blitt korrigert.	21
	Som en bruker vil jeg ha mulighet til å åpne en faktura for å kontrollere fakturadetaljer.	21
	Som en utvikler vil jeg at brukerne skal ha mulighet til å gi tilbakemelding slik at produktet kan forbedres over tid.	8
	Som en bruker vil jeg ha muligheten til å opprette nytt passord for å få tilgang til min konto dersom jeg glemmer passordet.	34
	Som en bruker vil jeg ha muligheten til å logge inn for å få tilgang til min bruker.	34
	Som en bruker vil jeg kunne godkjenne eller avslå en faktura slik at jeg kan ta en avgjørelse på om den er utfylt korrekt eller ikke.	8
	Som en bruker ønsker jeg å kunne kommentere på en faktura slik at det er enkelt å legge ved utfyllende informasjon.	55
	Som en bruker vil jeg kunne videresende en faktura til en annen bruker for annen nødvendig godkjenning.	21
	Som en bruker vil jeg kunne endre fakturaposteringer slik at jeg kan korrigere eventuelle feil på faktura.	89
	Som en bruker vil jeg ha en oversikt over tidligere fakturaer jeg har behandlet for å kunne kontrollere i ettertid.	21
	Som en bruker vil jeg få oversikt over fakturaer jeg må behandle for å ha kontroll over mine fakturaer.	13

Figur 1: Resultat fra Planning Poker

For å vite hva som skal gjøres i hver Sprint må brukerhistorier fra vår Product Backlog velges, legges til i Sprint Backlog og deretter deles opp i oppgaver. Dette blir gjort på et Sprint Planleggingsmøte.

Der Planning Poker tar for seg kompleksiteten i prosjektets hovedkomponenter, tar Sprint Planlegging kun for seg den kommende Sprinten og hva som skal gjøres i løpet av denne tidsperioden. Sprint Planlegging er gjerne det første møtet man har i begynnelsen av en Sprint, og har som oppgave å resultere i en plan for hva som skal utføres i løpet av Sprinten (Sutherland & Schwaber, 2007). Vi hadde dette møtet første dag i hver Sprint. I henhold til standard Scrum-regler satt vi en øvre tidsgrense på Sprint Planlegging møtet til to timer per uke, altså totalt seks timer. I Scrum kalles dette *timeboxing*.

På dette møtet velges det ut hvilke av brukerhistoriene man skal ta for seg den kommende Sprinten, og de brytes så ned i konkrete oppgaver, der brukerhistorier med høyere tall fra Planning Poker ansees som mer komplekse og derfor antas å kunne ta lengre tid. Disse oppgavene estimeres det så tid på, for å beregne hvor lang tid man antar at de ville kreve for å ferdigstilles, også kjent som *Estimated Time to Complete* (ETC). Tidsestimeringen på oppgavene ble gjennomført ved diskusjon rundt hver oppgave, og ble målt i faktiske timer.

2.1.3 Loggføring av tidsbruk

For å loggføre tidsbruk i prosjektet valgte vi å benytte et regneark der vi registrerte hvor mye tid man jobbet hver dag. Dette regnearket var overordnet for hele emnet IS-304 Bacheloroppgave i informasjonssystemer, og dekket både tidsbruk relatert til konkrete oppgaver, men også tid brukt på skriving av rapport, opplæring og lignende. Se et utklipp av regnearket i Figur 2.

Dato	Jørgen	Magnus	Markus	Kirsti	Ola	Stian
10.02.2021	6	6	6	6	6	6
11.02.2021	4	3	2	Utilgjengelig	4	3
12.02.2021	3	3	5	Utilgjengelig	5	2
13.02.2021						
14.02.2021						
15.02.2021	3	3	3	3	3	4
16.02.2021	3	2	2	4	4	4
17.02.2021	5	4	6	4	4	4
18.02.2021	2	3	6	6	4	3
19.02.2021	2	2	2	2	2	2
20.02.2021						
21.02.2021						
22.02.2021	3	3	3	3	Utilgjengelig	3
23.02.2021	2	5	2	4	Utilgjengelig	4
24.02.2021	4	4	3	4	Utilgjengelig	4
25.02.2021	3	2	5	5	Utilgjengelig	5
26.02.2021	4	2	3	4	Utilgjengelig	3
27.02.2021			3	3		

Figur 2: Utklipp fra tidslogg

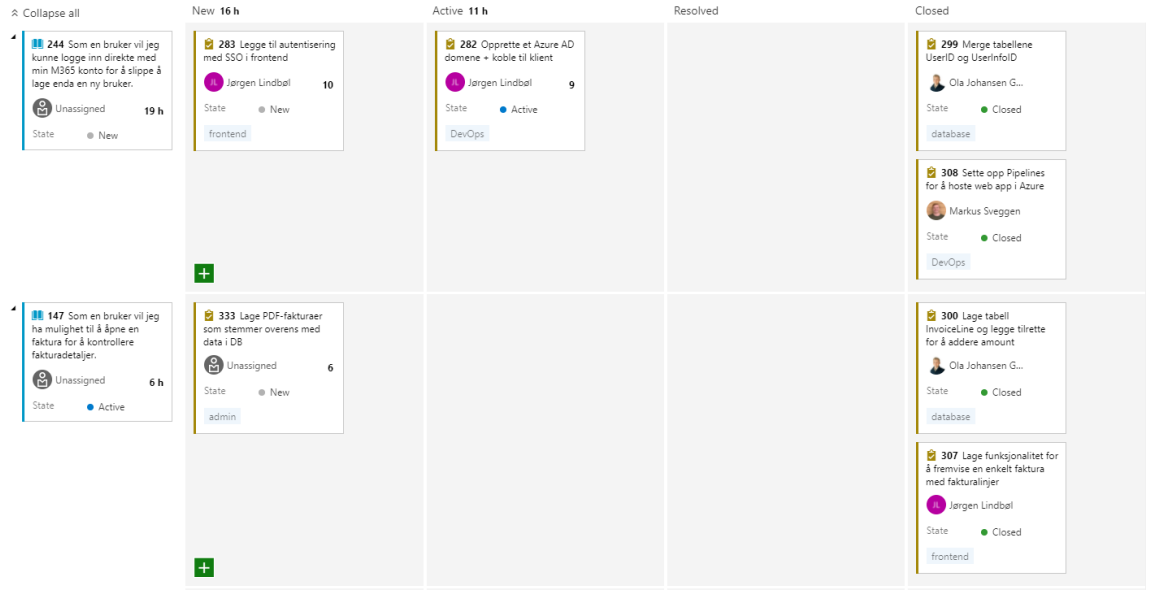
I tillegg til dette regnearket så benyttet vi også tidsestimering relatert til hver konkrete oppgave i prosjektstyringsverktøyet DevOps, se mer om dette i siste avsnitt i kapittel 2.1.4 Prosjektstyringsverktøy.

Vi valgte å holde oppgaver som ble lagt inn i DevOps kun relatert til konkret utvikling av produktet, mens skriveoppgaver, opplæring og annet ikke ble skrevet inn her. Dette nødvendiggjorde regnearket som et kontrollverktøy for total tidsbruk i emnet.

2.1.4 Prosjektstyringsverktøy

For å organisere og planlegge prosjektet i sin helhet samt hver Sprint Backlog, har vi besluttet å bruke et verktøy ved navn Azure DevOps. Dette er et prosjektstyringsverktøy som Knowit benytter i sine prosjekter, og som vi fikk tilgang til. Dette verktøyet er organisert slik at man legger inn brukerhistorier som øverste nivå, for så å tildele brukerhistorier til hver Sprint Backlog, ut i fra hvilke brukerhistorier som skal implementeres og ferdigstilles den Sprinten.

I hver Sprint Backlog så kan man lage mindre, konkrete oppgaver koblet mot hver brukerhistorie. Disse er sortert i kolonner etter status: "New", "Active", "Resolved" og "Closed". En oppgave som blir opprettet starter i kolonnen "New", deretter flyttes den til "Active" når man begynner på den. Dersom man har problemer med en oppgave kan den flyttes til "Resolved" når problemer er rettet opp, og til slutt flyttes oppgaven til "Closed" når den er ferdigstilt.



Figur 3: Utklipp fra Sprint Backlog i Sprint 4

Vi valgte å organisere arbeidet slik at hver oppgave eller "task" har sin egen branch i sitt tilhørende Git repository (se mer under 2.2.4 Versjonskontroll), der hensikten var å gjøre det lett å se arbeidet som ble gjort

knyttet til hver oppgave. Slik DevOps fungerer så ble en oppgave automatisk flyttet til "Closed" når arbeidet blir godkjent og slått sammen med brukerhistoriens "branch". På denne måten sørget vi for at ingen oppgaver kunne bli lukket før implementeringen var gått gjennom og godkjent, slik at vi kunne sikre kvalitet i arbeidet. Se mer om dette i kapittel 2.3.2 Gjennomgang av kode.

Som nevnt har DevOps også et system for loggføring av tidsestimering- og bruk per oppgave. Først legger man inn "Original Estimate" etter diskusjon av tidsestimering under Sprint Planlegging, deretter fyller man inn det samme under "Remaining" før man begynner på oppgaven. Feltet "Completed" står da som "0" ettersom man ikke har jobbet på oppgaven enda.

Underveis vil "Original Estimate" stå urørt som en logg på hvor lenge man originalt estimerte at oppgaven ville ta, også oppdaterer man antall timer under "Completed" ettersom man jobber. Dersom det viser seg at oppgaven tar mer eller mindre tid enn antatt kan man oppdatere feltet "Remaining" underveis, så ser man til slutt om det originale estimatet stemte overens med "Completed" hours. Dette er en ryddig og oversiktlig måte å holde kontroll på tidsbruk på hver oppgave i DevOps, og det gir en oversikt over hvor nøyaktige estimatene er ettersom vi kan sammenligne "Original Estimate" med "Completed" ettersom oppgavene blir fullført. I tillegg oppretter DevOps automatisk et Burndown-chart som oppdateres hver gang tidsbruk oppdateres på hver oppgave gjennom dette systemet, som bidrar til å visualisere fremgang i prosjektet.

2.1.5 Oppfølging og tilbakemelding

Daily Standup

For å sørge for kontinuerlig oppfølging underveis i en Sprint benyttet vi såkalte Daily Standups. Dette er korte møter timeboxet til maks 15 minutter, der hvert teammedlem svarer kort på tre spørsmål:

- Hva har du gjort siden forrige Daily Standup?
- Hva planlegger du å gjøre fram til neste Daily Standup?
- Hvilke eventuelle utfordringer har du møtt på i arbeidet?

Dette sørger for at alle på teamet er oppdatert på hverandres fremgang og utfordringer, og oppfordrer teamet til å hjelpe hverandre. Det sørger også for at problemer underveis blir oppdaget og løst raskt, slik at man sikrer fremgang (Sutherland & Schwaber, 2007).

Ettersom emnet IS-304 Bacheloroppgave i informasjonssystemer er et emne på 20 studiepoeng og vi har et annet kurs ved siden av, så ble det beregnet en total arbeidsmengde per teammedlem på tre og en halv dag i uken for bachelorprosjektet og én og en halv dag for resterende emne.

Derfor bestemte vi oss for å ha Daily Standup mandag, onsdag og fredag. Da hadde vi én dag i mellom til å jobbe på tildelte oppgaver, og gruppen fikk nok tid til å jobbe med oppgavene slik at vi kunne presentere potensielle utfordringer ved hver Daily Standup.

Sprint Review

Når Sprinten er over er det viktig å oppdatere produkteier på hva som er gjort, og gå igjennom progresjonen fra den siste Sprinten. Et Sprint Review er et møte mellom teamet og produkteier, der teamet kan oppdatere produkteier om hva som er status på prosjektet, og produkteier kan oppdatere teamet med produkteiers perspektiv av prosjektet. Det er en samtale der man kan diskutere hvordan ting har gått, og diskutere råd og anbefalinger om hvordan man skal gå videre fremover (Sutherland & Schwaber, 2007).

Ettersom Knowit er vant til å jobbe med Scrum i sine prosjekter så var de villige og ønsket gjerne å ha et Sprint Review med oss mot slutten av hver Sprint. Det er viktig å nevne her at selv om vi hadde en flytende samtale med produkteier hyppig gjennom Teams, så var Sprint Review et større møte der vi oppsummerte total progresjon og la fram status på prosjektets fremgang og de utfordringer vi hadde løst eller fortsatt jobbet med.

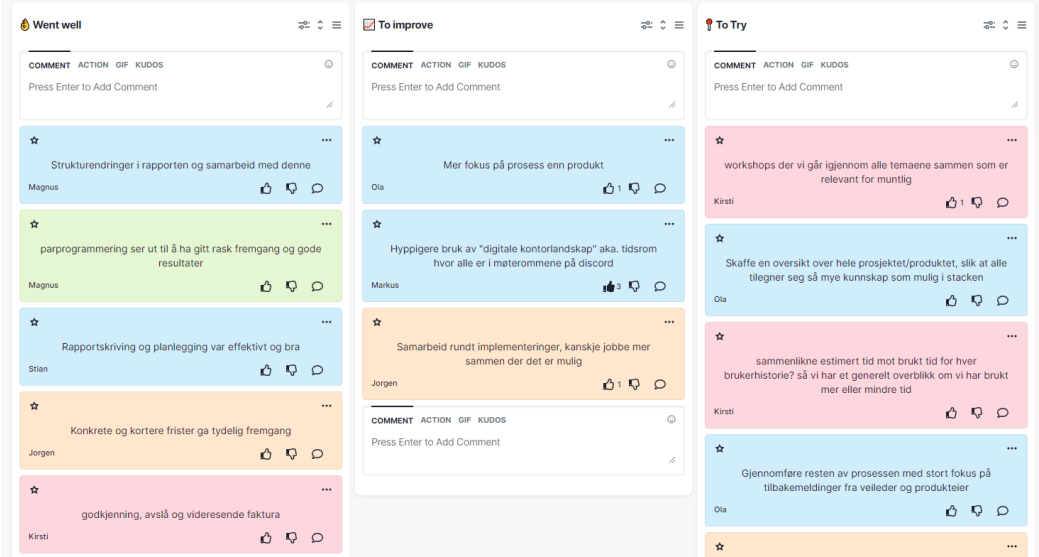
Sprint Retrospective

Etter gjennomføring av Sprint Review med produkteier er det viktig for teamet å ha et større møte der man går gjennom arbeidsprosessen den siste Sprinten. I Scrum kalles dette en “Sprint Retrospective”. Der Sprint Review fokuserer på produktet, fokuserer Sprint Retrospective på prosessen. Her kan teamet diskutere hva som gikk bra, hva som burde forbedres og konkret hva man ønsker å prøve i neste Sprint for å se forbedringer (Sutherland & Schwaber, 2007).

For å gjennomføre Sprint Retrospective valgte vi å bruke organisasjonen Reetro sitt nettsted (https://reetro.io/). Dette nettstedet gjør det mulig å invitere alle teammedlemmene inn på samme side, hvor det er lagt opp kolonner der man kan skrive inn hva som gikk bra, hva man ønsker å forbedre, og hva man ønsker å prøve. Dette gjorde det enkelt og effektivt å få samlet tilbakemelding fra alle i teamet, samt ga oss en god oversikt over det som ble skrevet. Et utklipp fra vår første Sprint Retrospective kan sees i Figur 4.

Det ble bestemt å gjennomføre Sprint Retrospective rett etter Sprint Review siste dag i hver Sprint, slik at alle tanker var ferskt i minne. Dette ville bidra til at møtet ble effektivt, og vi fikk samlet mye tilbakemelding hver Sprint.

Informasjonen hentet inn under Sprint Retrospective ble bestemt gjennomgått og vurdert ved førstkommende Sprint Planlegging, der vi kunne innføre tiltak basert på tilbakemeldingen for planlegging av neste Sprint.



Figur 4: Utklipp fra Sprint Retrospective

2.2 Teknologi

Det ble tidlig i prosjektet prioritert å ha et bevisst forhold til valgene rundt teknologi. Dette for å sørge for at prosjektet er utviklet med teknologi som kan bidra til kvalitet i prosjektet og gjøre utviklingsprosessen så smidig som mulig. Valgene knyttet til bruk av teknologi i prosjektet ble også tatt med fokus på oppgaven vi skulle utføre, som var å utvikle en applikasjon i Microsoft Teams. Dette domenet førte med seg noen begrensninger for hvilken teknologi vi kunne benytte oss av. Ettersom Microsoft har valgt å bygge en SDK (Software Development Kit) for JavaScript, og anbefaler dette språket for utvikling av Teams-applikasjoner, var utgangspunktet vårt at dette programmeringsspråket skulle benyttes på frontend (Microsoft, 2021b). Vi hadde også noen forutsetninger rundt valg av teknologi som Knowit hadde fastsatt. Blant annet benytter de seg av Azure-miljøet, herunder Azure DevOps, SQL Server og andre verktøy tilknyttet Azure. Disse forutsetningene gjorde at de resterende teknologi-valgene var enklere å ta, ettersom de naturlig hørte til i det samme Microsoft- og Azure-økosystemet.

En annen forutsetning som var med å prege blant annet valg av teknologi omhandlet det faktum at om systemet skulle bli tatt i bruk var det kun frontend-delen av prosjektet som skulle gjenbrukes, da backend sitt formål ville være å simulere Semine sitt eksisterende system. Semine sitt eksisterende system kan kommuniseres med via API-kall, og vi valgte dermed å simulere dette i høyest mulig grad. Dette er en praksis som Knowit også følger i sine prosjekt for å ha full kontroll over testmiljøet før det kobles på eksterne systemer. Vi besluttet da å separere backend og frontend som to individuelle applikasjoner; henholdsvis API og Teams-

applikasjon. En visuell representasjon av forholdet mellom disse applikasjonene er illustrert og beskrevet i kapittel 3.3.2 Systemarkitektur.

2.2.1 Programmeringsspråk og rammeverk

På bakgrunn av at vi besluttet å separere backend og frontend i ulike applikasjoner, fikk vi også mulighet til å benytte oss av ulike programmeringsspråk og rammeverk, best mulig tilpasset de forskjellige applikasjonene.

Frontend

Som nevnt i 2.2 Teknologi var vi avhengig av å benytte oss av JavaScript på frontend. For å undersøke hvilket JavaScript-rammeverk som passet best for vårt prosjekt og utvikling av applikasjoner i Microsoft Teams, undersøkte vi Microsoft sine egne manualer for Teams utvikling - her ble rammeverket React tatt i bruk. Dette rammeverket er komponentbasert som gjør det mulig å utvikle enkle individuelle komponenter som senere kan kobles sammen for å skape avanserte brukergrensesnitt (React, 2021). Dette er blant de mest brukte JavaScript-rammeverkene, og et av rammeverkene innenfor Knowit sin portefølje. Vi besluttet dermed å bruke JavaScript med rammeverket React for utvikling på frontend.

Backend

Ettersom vi besluttet å fordele frontend og backend med kommunikasjon via API kall sto vi fritt til å velge et annet programmeringsspråk for backend. Siden mesteparten av den teknologien vi allerede hadde besluttet å bruke var innenfor Microsoft og Azure-miljøet, så vi til et programmeringsspråk som egnet seg for backend-utvikling i denne stacken. Det er en viss trygghet i å velge teknologi som blir brukt med hverandre, da det mest sannsynlig vil tilsi at det finnes et utviklermiljø som benytter seg av disse teknologiene på tvers av hverandre. Programmeringsspråket C# var da et naturlig valg, da dette tilhører Microsoft-miljøet og egner seg godt for API-utvikling på backend ved hjelp av rammeverket *ASP.NET Core* sin modul for utvikling av API, kalt *ASP.NET Web Api* (Microsoft, 2021a).

Ved å utforske funksjonene i Azure, så vi at støtten for bruk av C# var tilstede i alle ledd. Vi besluttet da å bruke C# sammen med rammeverket *ASP.NET* og den nyeste versjonen kalt *.NET 5*. Vi ønsket også å benytte oss av en *Object Relational Mapper* (ORM) for å kunne kommunisere med databasen enkelt uten å skrive SQL-spørringer manuelt, samt koble klasser i koden mot tabeller i databasen, for å enklere hente ut og legge til ny data. Å skrive manuelle spørringer er tidkrevende, må testes og de innebygde spørringene i en ORM inneholder sannsynligvis færre sikkerhetsfeil og er mer optimaliserte enn de manuelle spørringene vi ville skrevet. Vi ble anbefalt å bruke *Entity Framework* som ORM av Knowit, og vi besluttet da å lytte til deres anbefaling og ta i bruk dette.

2.2.2 Database

I vårt prosjekt besluttet vi å bruke *Azure SQL Server* for å bygge en relasjonsdatabase. Grunnen til det er blant annet at tjenesten er skybasert, uten behov for å installere programvare (eller maskinvare) for å bruke tjenesten og dette gjorde det enklere for oss å kunne jobbe effektivt med databasen uten å bruke mye tid på å logge inn og koble opp til en server. En annen grunn til at vi valgte å ta i bruk Azure SQL Server var fordi Knowit anbefalte denne teknologien på bakgrunn av at Azure SQL Server kommuniserte veldig godt med resten av teknologiene forbundet med Microsoft. Azure SQL Server kjører *Microsoft SQL Server* (MSSQL) i skyen. MSSQL er utviklet av Microsoft og er en pakke med programvare for bruk av database. I vårt prosjekt inneholder MSSQL en såkalt relasjonsdatabase-motor som gjør at vi hadde det tekniske rammeverket vi trengte for å bygge en database som lagrer verdier i tabeller, rader og kolonner. For å skrive scripts og spørringer bruker MSSQL tradisjonell og kjent SQL-syntaks (Microsoft, 2021c).

2.2.3 Utviklingsmiljø

For å gjøre selve programmeringen mest mulig optimal og smertefritt, ønsket vi å ta i bruk utviklingsmiljø med funksjonalitet nødvendig for å gjøre dette mulig. Da tidligere valgt teknologi befinner seg i Microsoft stacken, vurderte vi utviklingsmiljø herfra, med ulike krav for frontend og backend. Vi besluttet å ta i bruk Visual Studio Code for frontend-utvikling ettersom dette miljøet tilbydde den nødvendige funksjonaliteten vi trengte, samt at dette miljøet var det eneste utviklingsmiljøet med en plugin utviklet for å bygge skallet til en Microsoft Teams-applikasjon (Microsoft, 2021b).

Vi besluttet å ta i bruk Visual Studio for backend-utvikling. Da dette har flere innebygde funksjoner enn Visual Studio Code, blant annet innebygde funksjoner for tilkobling til databasen i Azure og gode debugging og testverktøy, som vi vurderte som viktig på backend for å tilrettelegge for god kvalitetssikring.

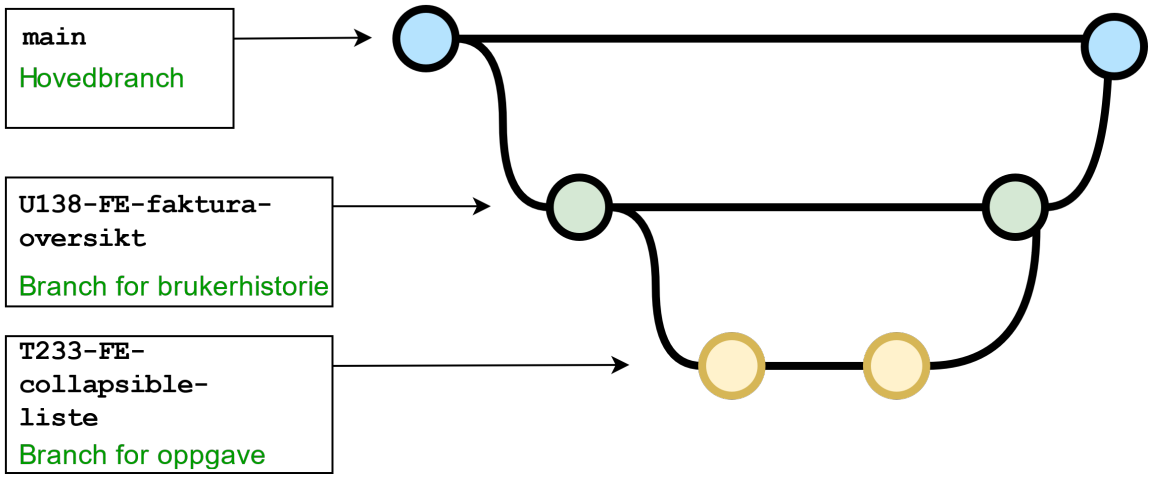
2.2.4 Versjonskontroll

I et prosjekt som blir skapt av flere utviklere er det til enhver tid viktig å holde orden på endringer og nye funksjoner i systemet. Ved å ta i bruk versjonskontroll får man mer naturlig oversikt og kontroll over ulike versjoner, og man kan samtidig sikre sømløse sammenslåinger av disse (Atlassian, 2021). Et tiltak vi benyttet oss av for å sikre trygg fremgang i prosjektet har vært bruken av Git i Azure Repos. Dette er et versjonskontrollsystem som er en del av DevOps, og ble satt opp for oss tidlig i prosjektperioden av Knowit.

Som nevnt tidligere i rapporten besluttet vi å separere prosjektet i to applikasjoner, frontend og backend. For å holde disse applikasjonene uavhengige av hverandre valgte vi å opprette et isolert repository for hver av de to applikasjonene. Dette sørger for at de to systemene ikke er direkte sammenkoblet, og gjør produktet mer

modulært og bedre tilpasset potensiell utbytting av for eksempel backend senere.

For å bruke Git på en ryddig måte, ble vi enige om en felles fremgangsmåte for å sikre høy kvalitet, samt forebygge konflikter ved merging. Fremgangsmåten går ut på å lage en brukerhistorie-branch for hver brukerhistorie i pågående Sprint. Dette gjøres i hvert av repositoriene. Videre skal det opprettes en egen branch for hver oppgave, basert på brukerhistorie-branchen den tilhører. Når oppgaven er ferdigstilt og testet, skal den bli merget inn med tilhørende brukerhistorie-branch. Først når alle oppgavene tilhørende en brukerhistorie-branch er blitt merget inn, kan denne brukerhistorie-branchen merges med vår hoved-branch, main. Dette er tiltak vi vil gjøre for å øke sikkerhet og forebygge feil, og for å få segregert oppgaver som fører til bedring av struktur i utviklingsprosessen. Vi utarbeidet denne prosessen gjennom samtaler med produkteier.



Figur 5: Git-branch diagram

For å sikre at gruppen hadde lik fremgangsmåte hva gjelder rutiner i repositoriet, utformet vi en «Git-guide» (se Tillegg C) som måtte følges. Denne ble satt sammen basert på prosessen beskrevet over og ved hjelp av Git sin offisielle dokumentasjon på deres nettsted (http://git-scm.com/). Hensikten med denne var å være et felles oppslagsverk for hvordan man jobber med Git i vårt prosjekt, og å gi føringer for konkrete scenarioer man kan møte på da Git kan være et komplisert verktøy. Denne Git-guiden inneholder blant annet fremgangsmåten for arbeid med branches i Git, som beskrevet over.

2.2.5 Automatisering og deployering

For spare tid og ressurser i det lange løp ønsket vi å ta i bruk automatiserte systemer for testing, bygging og deployering av kode som godkjennes inn i “main repository”. På denne måten sørger vi for at siste versjon av prosjektet alltid kjører i produksjon. I samarbeid med Knowit fikk vi innføring i et system som heter

Azure Pipelines, da det var et ønske både fra produkteier og teamet å ta dette i bruk. Pipelines baserer seg på begrepene «Continuous Integration» og «Continuous Deployment» og går ut på å ha et system som kan benyttes for å opprette automasjon av bygging og testing av kode, for deretter å sendes til plattformen man benytter for å kjøre koden i produksjon (Microsoft, 2019).

Dette blir satt opp slik at Azure automatisk kompilerer og bygger koden vår hver gang vi godkjenner endringer inn mot "main repository". Det fører til at det kun er ferdig kode som er gjennomgått av teamet (mer om dette under 2.3.2 Gjennomgang av kode) og er fungerende som til enhver tid blir sendt ut til produksjon , der koden kjøres i to Azure Cloud-applikasjoner, én for frontend og én for backend.

Dette valgte vi å bruke fordi det forenklet prosessen betydelig da vi slapp å selv kompilere kode som så ble lastet opp og startet i produksjon. Hele denne prosessen skjer automatisk og av seg selv uten at vi trenger å gripe inn. Ettersom denne prosessen kun settes opp én gang og deretter kjører automatisk, bidrar dette også til å forhindre eventuelle menneskelige feil som ellers kunne oppstått.

2.3 Kvalitetssikring

En stor del av det å bygge opp og sikre kvalitet i et prosjekt ligger i de sentrale avgjørelsene som blir tatt tidlig for prosjektet. Dette gjelder både for valg relatert til prosjektstyring, samt valg av teknologi, som ble presentert i delkapitlene ovenfor. Gruppen valgte imidlertid å gjennomført ytterligere tiltak for å kontinuerlig opprettholde kvaliteten gjennom prosjektet. De ytterligere tiltakene som vi tok i begynnelsen av prosjektet blir presentert som egne underkapitler nedenfor. Ytterligere kvalitetssikringsaktiviteter som ble gjennomført underveis i prosjektet vil bli presentert i kapittel 3 Utførelse og erfaringer.

2.3.1 Kodestandard og utviklingsprinsipper

En kodestandard er et sett med retningslinjer utviklere følger når man programmerer, eksempelvis Camelcase som tar for seg navngivning til variabler og klasser. Vi valgte å ha mye fokus på kodestandard for å ha et ensartet utseende på selve koden. Det øker kodens lesbarhet og vedlikeholdsevnen i fremtiden, og det økte effektiviteten til andre utviklere i teamet for videreutvikling og testing av koden. Hovedprinsippet vi valgte å ha i fokus var "Clean Code"-prinsippet, som har ren og lesbar kode i hovedfokus. For å få til dette benyttet vi oss av prinsippene *DRY*, *KISS*, og *SOLID*.

DRY (Don't repeat yourself) baserer seg på å unngå unødvendig repetisjon i koden. Dette kan utføres ved å alltid henvise tilbake til samme kilde, eller referansepunkt, i koden istedenfor å skrive den flere ganger. Hvis man trenger å endre på denne variabelen senere trenger man kun modifisering én gang.

Kode som har mange forskjellige komponenter og er skrevet med høy kompleksitet er ofte vanskeligere å forstå for andre utviklere, og har risikoen for flere bugs (Pyram, 2020). KISS-konseptet dreier seg om å holde koden simpel og lesbar, som ofte eliminerer komplekse systemer og gjør det enklere for vedlikehold og bug fixing.

SOLID er en kodestandard satt sammen av fem design prinsipper skapt av Robert C. Martin. Alle designprinsippene har uavhengighet og modularitet i fokus, for å skape et system som er enkelt å utvide uten å endre på eksisterende kode. Denne kodestandarden ble et naturlig valg for oss da vi ønsket å utvikle et system som er enkelt å forstå og utvide. Ved å følge SOLID prinsippene er det også færre bugs som oppstår på grunn av uavhengighet mellom komponenter.

Vi valgte å bruke Clean Code for å sørge for lav *coupling* og høy *cohesion*, som gir enklere og mer effektivt systemvedlikehold og redusert kompleksitet. Ved å ha en felles kodestandard og ved å følge samme prinsipper er det enklere for oss å hjelpe hverandre og å bygge på andre sin kode uten komplikasjoner. Prinsippene er ikke perfekte, og å følge prinsippene slavisk kan føre til problemer i koden. Programvare-ingeniøren Brian Geihsler er en av motstanderne til SOLID-prinsippet. Han mener SOLID fører til en kodebase med lav coupling, men den kan fort bli uleselig (Geihsler, 2014). Ved å være bevisst på både de positive og negative sidene valgte vi å bruke prinsippene der vi følte det var relevant, og å skape et system som vi mener er simpelt og forståelig med muligheter for utvidelse og vedlikehold.

2.3.2 Gjennomgang av kode

Som vi har nevnt tidligere i rapporten så har gjennomgang av kode fungert som et kvalitetssikringslag før kode blir vurdert som ferdig og blir pushet inn mot main branch. Som et sikkerhetstiltak har vi i DevOps lagt inn en begrensning på når en brukerhistorie branch kan merges med main branch. Dette ble vi enige om tidlig i prosjektet og har vært en standard praksis for hvert tillegg i repositoriet.

Begrensningen baserer seg på at koden som skal legges til må sendes gjennom en "pull request" der minst ett annet teammedlem må gå gjennom og godkjenne koden før den legges til. Dette er for å fange opp eventuelle feil eller mangler som kan forhindre applikasjonen i å kjøre slik det var ment. Under kodegjennomgangen må et annet gruppemedlem enn den som har opprettet pull-requesten se over koden som skal legges til og godkjenne at den følger kodestandarden vi har blitt enig om. Dersom det er avvik fra kodestandard eller funksjonalitetskrav kan dette noteres som kommentar på innsendt pull request, der den som åpnet pull request får notifikasjon om tilbakemelding og kan utbedre. Først når dette er gjennomført kan pull-requesten godkjennes og bli merget med main eller brukerhistorie branch. Ved sammenslåing av kode fra brukerhistorie branch til main branch må den som godkjenner gå igjennom akseptansekrav for å forsikre at disse er oppfylt

før godkjenning.

2.3.3 Testing

For å sikre at prosjektet skulle ha gjennomgående høy kvalitet prioriterte vi å utvikle kode som skulle ha god testdekning i alle ledd, og bli testet kontinuerlig. Testing er viktig da det uavhengig av implementering forsikrer at forventet funksjonalitet er på plass, sørger for at man tidlig kan oppdage eventuelle feil og mangler. Videre kan tester bidra til å sørge for at kvalitetskrav er oppfylt (Schaefer, 2014). Etter anbefalinger fra Knowit besluttet vi å skrive enhetstester på backend. For å fasilitere testkravene våre besluttet vi å ta i bruk det populære testrammeverket NUnit for enhetstesting på backend. Vi besluttet også å utforme testene etter prinsippet F.I.R.S.T. som tar for seg at tester skal være raske, selvstendige, repeterbare, selv-validerende og være skrevet i rett tid (Martin, 2008).

2.3.4 Kommunikasjon og samarbeid

Fra starten av prosjektet har gruppen strebet etter å holde god kommunikasjon. Dette gjelder både innad i gruppen men også utad med kontaktpersoner og Knowit, samt veileder ved UiA. Dette skyldes først og fremst fordi god kommunikasjon anses som et viktig prinsipp når man jobber på agile prosjekter, men også fordi Covid-19 har hindret gruppen å sitte i samme fysiske kontorlandskap. Teamet etablerte derfor tidlig kommunikasjonsverktøy for å sikre jevn og god kommunikasjon.

Microsoft Teams (for kommunikasjon)

Under pandemien har bruk av digitale møteplattformer økt i stor grad (Spataro, 2020). En en av disse plattformene er Microsoft Teams. Både gruppen og veiledere ved Knowit hadde tidligere erfaring med denne plattformen og det ble derfor vurdert som et godt verktøy. God digital kommunikasjon utad er kritisk for å kunne sikre tilbakemeldinger og veiledning i prosjektet. Microsoft Teams ble derfor brukt til generelle spørsmål, Sprint Reviews, samt styringsgruppemøter for å oppdatere prosjektstatus for interessenter både hos Knowit og ved UiA.

Discord

For å sikre gode rutiner på kommunikasjon innad i gruppen valgte vi å ta i bruk Discord. På samme måte som Microsoft Teams er Discord en møteplattform der man kan opprette egne kanaler eller bli med i andres eksisterende kanaler. Gruppen opprettet tidlig en egen Discord-kanal der det både finnes tale-rom og tekst-rom. Det ble opprettet såkalte “Grupperom” hvor ulike teams kan sitte å jobbe samtidig men da i forskjellige rom. Vi valgte å benytte oss av Discord som vår interne hovedkanal til Daily Standup og andre interne møter.

Muligheten for å dele skjerm i en Discord-samtale gjør det lettere å håndtere problemstillinger eller få innspill av andre.

Gruppekontrakt

Et annet tiltak gruppen gjorde for blant annet å sikre god kommunikasjon var å utarbeide en en gruppekontrakt. Dette dokumentet har gjennom prosjektet satt forventninger til hvordan gruppen skal forholde seg til prosjektet og hverandre. Her ble det blant annet satt faste dager hvor gruppen skulle jobbe samt hvilke kommunikasjonsverktøy som skulle bli brukt. Det ble også skrevet ned reprimander dersom noen skulle fravike i stor grad fra det som ble forventet. Gruppekontrakt er lagt til som Tillegg B.

2.3.5 Risikovurdering

Ved alle valg og beslutninger gruppen har tatt gjennom prosjektet har det vært en medhørende risiko, hvor noen hendelser og avgjørelser har blitt vurdert som mer kritisk enn andre. En god måte for å få oversikt over risiko som tilhører et prosjekt er å gjennomføre en risikovurdering. En risikovurdering gjør det mulig å få en oversikt over risikoer, samt kartlegge hvilke konsekvenser som tilhører risikoen. Man vil også kunne identifisere tiltak for å forebygge at risikable hendelser kan inntreffe, samt identifisere tiltak for å redusere omfang av skaden.
Gruppen valgte tidlig i prosjektet å gjennomføre en risikovurdering. Malen for risikovurderingen ble hentet fra Unit - Direktorat for IKT og fellestjenester i høyere utdanning og forsking (Unit, 2020). Den ble til en viss grad tilpasset etter hva som ble presentert av veileder Hallgeir i forelesning om risiko og risikovurdering.

Nr.	Uønsket hendelse (risikoelement)	Årsak	Sårbarhet	Risikonivå		
				S	K	Risiko
	Hva kan skje? Hva er den uønskede hendelsen? Hvilke tap oppstår?	**Hvorfor vil det kunne skje?** Hvem eller hva initerer hendelsen? For overlagte hendelser: Hvilken kapasitet og motiv har trusselaktøren?	**Hvilken svakhet/feil kan utnyttes her?**	**Sansynlighet og konsekvens på en skala fra 1 til 4.** 1-5 = Lav 6-10 = Moderat 11-13 = Over middels 14-16 = Høy		
NR1	Dårlig eller svekket kommunikasjon og samarbeid innad i gruppen	Hvis vi må jobbe mye digitalt kan avstand fra hverandre påvikre kommunikasjonen, tekniske problemer kan påvikre. f.eks. kan uengiheter i gruppen, missforståelser eller andre privte hendelser som gjør at kommunikasjon svekkes.	Det vil være en sårbarhet ved at samarbeid i forhold til gruppearbeid blir svekket. Kan føre til at oppgaver ikke blir løst eller at de ikke blir løst på en riktig måte. Kan også påvirke samholdet i gruppen.	3	4	12

Figur 6: Eksempel på risikoer

Utklippet i Figur 6 presenterer eksempler på risikoer som gruppen kan stå overfor gjennom prosjektet. Her knyttes årsak opp mot uønsket hendelse, og det spesifiseres hvilken sårbarhet som utnyttes i hvert tilfelle. Risikoen blir så tildelt et risikonivå. Risikonivået blir regnet ut ved å gange sannsynlighet med konsekvens,

som måles på en skala fra 1-5. Gruppen ble så enig om hva som vurderes som akseptabel risiko, hva som krever visse tiltak, og til slutt hva som skal vurderes som uakseptabel risiko som krever tiltak:

- Hendelser i røde/mørk oransje felt: Tiltak nødvendig
- Hendelser i gule felt: Tiltak vurderes ut fra ressurs og tidsbruk i fht. nytte
- Hendelser i grønne felt: Risikoer i disse feltene regnes som akseptabel risk. ”Billige” tiltak gjennomføres

Risikonivå			Tiltak for at det ikke skal skje	Tiltak redusere skade	Kommentar
S	K	Risiko			
Sansynlighet og konsekvens på en skala fra 1 til 4. 1-5 = Lav 6-10 = Moderat 11-13 = Over middels 14-16 = Høy			Beskriv forslag til nye tiltak. De kan deles opp i organisatoriske, menneskelige og teknologiske sikringstiltak.	Beskrive forslag til tiltak som vil redusere skaden	Erfaringer, vinkling til rapport
3	4	12	Ta i bruk tiltak for å opprettholde kommunikasjon når vi må jobbe digitalt (se eget punkt). Sette av tidspunkter for å kunne diskutere og ta opp nødvendige temaer. Ha på forhånd klar hva som skal tas opp og diskuteres så alle har mulighet til å forberede seg.	Avklar tidlig i prosessen at det skal være et åpent miljø hvor folk kan ta opp det de vil, og at alle skal bli hørt. Eksterne tiltak kan være å involvere veileder på UIA.	Skulle dette oppstå er dette noe som må settes av tid til. Dette er en essensiell funksjon som må være på plass for å gjennomføre prosjektet på best mulig måte. Settt heller av utviklingstid til dette og refleker over det i rapporten.

Figur 7: Risikotiltak

For den vurderte risikoen blir det identifisert tiltak for at den gitte risikable hendelsen ikke skal inntreffe, samt tiltak som beskriver hvordan vi kan redusere skade. Det ble også vedlagt en kommentar for hver risiko som ga rom for eventuelle andre tanker.

Mange av tiltakene som ble skrevet ned ble basert på valg vedrørende prosjektstyring og generell kvalitetssikring, som for eksempel Daily Standups og hyppig kommunikasjon. Risikomatrisen har i prosjektet blitt brukt som en enkel visualisering av risikovurdering gruppen har utført, som gjennom prosjektet ble oppdatert når gruppen møtte på nye risikoer. Hele risikovurderingen med tilhørende matrise er lagt til som Tillegg D.

3 Utførelse og erfaringer

Som vi presenterte i kapittel 2.3 Kvalitetssikring, så har vi i tillegg til tiltak som ble definert i starten av prosjektet, gjennomført aktiviteter underveis for å blant annet sikre fremgang og kvalitet for produktet. Dette kapittelet vil gjennomgå utførelsen av prosjektet; hvordan vi har jobbet for å oppfylle krav til sluttproduktet, utfordringer som har dukket opp underveis og våre erfaringer rundt prosessen.

3.1 Prosjektstyring og prosjektgjennomførelse

I dette underkapittelet vil vi beskrive vår prosess rundt administreringen av prosjektet og diskutere utførelsen og erfaringer, samt hva slags resultater vi oppnådde. Seks personer med ulike personlige egenskaper og bakgrunner åpner opp for muligheten til å være strategisk med oppdeling av ansvarsområder, slik at man har mulighet til å spesialisere seg innen et fagområde. Vi delte opp ansvarsområdene slik at tre jobbet primært med frontend, to jobbet primært med backend og en hadde hovedfokus på arbeid med database. Allikevel jobbet vi dynamisk slik at det ofte var mulig for noen å bidra utenfor sitt ansvarsområde, dersom det var behov. I tillegg til å dele inn i ansvarsområder fordelte vi også enkelte roller for noen av gruppemedlemmene. Scrum master ble valgt for å lede det agile utviklingsteamet. Gruppen har generelt hatt gode erfaringer med å jobbe agilt og det har gitt muligheter for å tilpasse og endre på elementer underveis. Selv om rammeverket har vært fleksibelt har det vært fint å ha en Scrum Master som har holdt oss innenfor de nødvendige rammene for å sikre fremgang. I tillegg til Scrum master valgte gruppen å ta i bruk en designert Git-ansvarlig, og testansvarlig.

Daily Standup og Sprint planlegging

Som vi beskrev i kapittel 2.1.5 Oppfølging og tilbakemelding, ble de faste felles arbeidsdagene bestemt til å være mandag, onsdag og fredag. Disse dagene fungerte som minstekrav for antall tid den enkelte skulle sette av og, i tråd med agil arbeidsmetodikk, var det alltid muligheter for å endre på tidspunktene for å tilpasse for ytre faktorer, f.eks. møte med Knowit eller personlige hendelser som kan oppstå. Det var også disse dagene vi gjennomførte Daily Standup, som skulle vise seg å være et viktig Scrum-rituale for oss av flere grunner; vi ble mer motiverte til å vise frem progresjon, det var lavere terskel for å be om hjelp og det var lettere å ha kontroll på oppgavene som lå i vår Sprint Backlog. Ved å ta i bruk Daily Standup la vi også opp for at alle teammedlemmene holdt hverandre oppdatert på tvers av ansvarsområder, som da også la grunnlag for diskusjon.

Sprint planlegging har vært grunnsteinen i organisering og suksessfull gjennomføring av sprintene. Vi opplevde dette som en sentral aktivitet for at vi skulle ha en god oversikt over arbeidet som skulle utføres i hver Sprint, og det var her vi fikk valgt ut funksjonalitet som skulle implementeres i hver sprint og brutt de ned i

oppgaver. Som nevnt tidligere satt vi en øvre tidsgrense på seks timer for dette møtet, to timer for hver uke i sprinten. Vi opplevde at sprint planlegging ble mer effektivt etterhvert som vi fikk mer erfaring og møtene ble derfor kortere.

Sprint Review og Sprint Retrospective

Sprint Review vært utrolig verdifullt for teamet, da vi opplevde å få gode, konstruktive tilbakemeldinger fra produkteier. Vi fikk også mulighet til å diskutere utfordringene vi har hatt i løpet av sprinten, og fikk presentert mulige løsninger fra produkteier og veileder. Ved flere anledninger ble det poengtert ut synspunkter på produktet vi som utviklere ikke alltid kunne se selv, selv om det kunne virke åpenbart sett fra utsiden.

Sprint Retrospective har gjennom prosjektet også vært et av de elementene som har sørget for jevn og god samhandling innad i gruppen. Hvor nyttig Sprint Retrospective var, er noe omdiskutert i teamet. På en side var det en god arena for å diskutere hvordan ting hadde gått, noe som absolutt var en fordel. Men samtidig tok vi også dette opp underveis på hvert Daily Standup-møte, og det var derfor tidvis begrenset hvor mye nytt som ble tatt opp under Sprint Retrospective, da vi ofte jobbet med å løse de største utfordringene underveis i arbeidet.

Tidsestimering

Vi forutså at det kom til å bli vanskelig å estimere hvor lang tid de ulike funksjonalitetene ville ta å gjennomføre. Dette blir grundigere diskutert under delkapittel 3.5.2. Verdien ved å tidsestimere var å skaffe en meget god oversikt over de ulike oppgavenes kompleksitet. Ved å komme med et forslag til tidsestimat, ble vi også tvunget til å vurdere oppgavene mot hverandre og dermed danne oss referansepunkter. Prosessen gjennomførte vi i plenum og hver eneste oppgave ble diskutert, slik at alle skaffet seg en dypere forståelse for arbeidet som ligger bak de ulike oppgavene.

Siden vi er forskjellige, er det også naturlig vi har ulike talenter og interesser innen systemutvikling. Noen foretrekker design og oppgaver relatert til frontend, andre foretrekker mer teknisk programmering og oppgaver som er mer relatert til backend. Siden vi individuelt rangerte kompleksiteten til en brukerhistorie gjennom Planning Poker, fikk vi en indikator på hvem som synes hvilke oppgaver relatert til en brukerhistorie var mest overkommelig.

Resultatet av tidsestimeringsprosessen var ikke bare at vi fikk rangert hvilke oppgaver som kom til å være mest tidkrevende, men vi fikk alle mer innsikt i hva slags arbeid som krevdes for å komme i mål.

3.1.1 Sprintene

Vår erfaring med valget på tre uker lange Sprinter er i stor grad positive. Vi opplevde det som en passende lengde på tidsperioden mellom Sprint Review og mellom Sprint Planlegging uten at det ble for lenge slik at vi kunne risikert å glemme viktige punkter fra f.eks. Sprint Planlegging. Til sammen har gruppen dette semesteret gjennomført fem Sprinter, hvor siste Sprint vil bli gjennomført etter innleveringen av skriftlig rapport. Kort oppsummert så handler de ulike Sprintene om disse temaene:

Sprint 1

Vår første Sprint bestod av en del individuelle arbeidsoppgaver innen *research*. Etter valgene rundt teknologier og rammeverk ble tatt, gikk oppgavene ut på at den enkelte skulle lese seg opp på teknologier og rammeverk som var relevant for den enkeltes ansvarsområde. Dette valgte vi å sette av tid til å gjennomføre fordi vi mente den strategien ville være givende når vi skulle starte å jobbe med tekniske utfordringer og oppgaver. I den første Sprinten ble det også blant annet avholdt et faglig «kick-off» i regi av Knowit. Der fikk vi en innføring i hva Knowit forventet av oss, hvordan et sluttprodukt kunne være, og hva slags funksjonaliteter de mente var hensiktsmessig å inkludere. Denne dagen var ikke bare ment for å gå igjennom tekniske krav, men også ment å være sosial slik at vi kunne bli bedre kjent med Knowit som et IT-selskap og lære oss mer om deres arbeidskultur. Dette var nyttig for oss, og vi fikk et tydeligere bilde på hvordan vi kunne forvente å arbeide med Knowit. Etter kick-offen fikk vi all informasjon vi trengte for å kunne starte på analyse og design. Denne prosessen vil bli nærmere forklart i kapittel 3.2.

Sprint 2

Da Sprint 2 startet hadde vi blitt ferdige med analyse og design, og vi kunne starte med de første oppgavene innen teknisk utvikling. Vi startet med å bygge opp sentrale deler i databasen for å kunne legge inn testdata, som vi senere skulle bruke for å visualisere vår progresjon i tillegg til å ha verdier som kan hentes fra backend og frontend for å teste funksjonalitet. Vi jobbet med å tilrettelegge for at backend skulle kunne hente ut data fra databasen og sende dette videre til frontend som en faktura. Utover dette ferdigstilte vi en standardisert guide for hvordan vi skulle bruke Git-funksjoner. Hensikten med dette var å forebygge konflikter mellom endringer i prosjektet dersom flere jobbet på samme branch.

Sprint 3

Sprint 3 bestod i stor grad av videreutvikling av sentrale funksjonaliteter. Blant oppgavene ble det lagt mer søkelys på å jobbe med testdekning i backend. Dette var relativt nytt for de fleste, men vi forstod formålet og logikken ved å skrive tester, og så dermed nytteverdien ved å bruke tid på dette.

Sprint 4

I Sprint 4 brukte vi tid og ressurser på bl.a. å opprette fungerende Azure Pipelines for både frontend og backend. Dette var utfordrende og nytt for oss. Knowit hadde heller ikke så mye erfaringer med å bruke Azure Pipelines, men vi fikk allikevel noen tips som vi kunne bruke for etter hvert å komme i mål med implementering av Azure Pipelines. Azure Pipelines blir nærmere forklart i underkapittel 2.2.5 og 3.4.3.

Mål for Sprint 5

I skrivende stund har vi nettopp påbegynt Sprint 5 som blir den siste Sprinten i prosjektet. Fokuset vil herfra bli å forsøke å ferdigstille kobling mot Single Sign On og notifikasjoner i Teams.

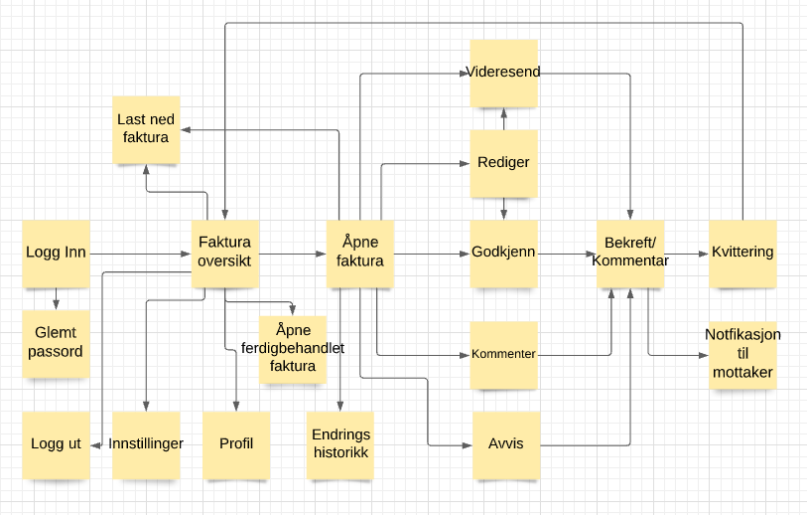
3.2 Initiell analyse

En av de viktige aktivitetene vi hadde fokus på i første Sprint var å få gjennomført en initiell analyse av produktet, slik at vi fikk en oversikt over de viktigste aspektene ved produktet som skulle utvikles. Derfor velger vi å skrive mer utfyllende om dette her. Som nevnt tidligere i rapporten har gruppen gjennomført ulike aktiviteter for å blant annet sikre og opprettholde kvaliteten på produktet. Flere av disse aktivitetene ble gjennomført i den initielle analysen i samarbeid med Knowit for å kartlegge og få en overordnet forståelse av oppgaven.
Planlegging og analyse i et prosjekt blir ofte gjennomført for å sikre at et team har en god forståelse for arbeidet som kreves for å gjennomføre prosjektet (Digitaliseringsdirektoratet, 2019). Av Knowit ble det lagt stor vekt på hvor viktig det var med god planlegging og analyse tidlig i et prosjekt. For å sitere produkteier Tor Oskar: “For hver time spesifikasjon sparer dere to timer utvikling”. Den initielle analysen har med andre ord vært en essensiell del av prosjektet.
Etter samtale med veiledere hos Knowit ble vi enige om at gruppen skulle gjennomføre ulike analyse-workshops. Slike workshops ble sett på som standard praksis hos Knowit når de startet på nye prosjekter. Det var derfor naturlig for oss og gjennomføre det samme. Vi tok for oss blant annet forretningsprosess, interessentanalyse, langsiktige mål, flowchart, og definerte til slutt en prosjektramme som munnet ut i konkrete brukerhistorier.

3.2.1 Forretningsprosess

Da oppgaven ble presentert av Knowit ble det lagt frem hvordan forretningsprosessen så ut per dags dato, hvor de benyttet Semine sin egen eksterne app for å behandle fakturaer. Hovedmålet med å sette opp en ny forretningsprosess går ut på å kartlegge hvordan en virksomhet eller en prosess kan effektiviseres (Wikipedia, 2019).
Målet var derfor å utforske mulighetene med å forbedre prosessen ved å ta i bruk en Microsoft Teams-

applikasjon. Basert på hvordan oppgaven ble presentert begynte gruppen å kartlegge alle mulighetene og prosessene. Vi satt opp en forretningsprosess som visualiserte all potensiell funksjonalitet produktet kan ha, dersom man ikke setter noen begrensninger på utviklingsressurser. Dette ga gruppen en tidlig oversikt over potensialet og størrelsen på prosjektet.

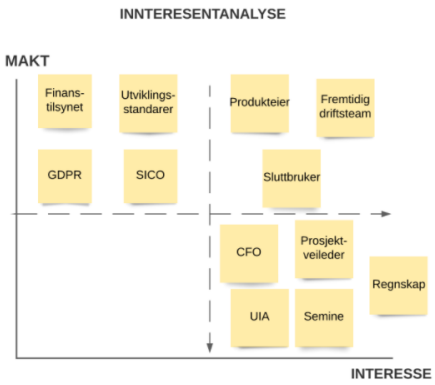


Figur 8: Utarbeidet forretningsprosess

3.2.2 Interessentanalyse

Etter å ha utarbeidet en forretningsprosess ønsket vi å kartlegge hvilke interessenter som kan ha påvirkning på prosjektet. Denne aktiviteten kalles interessentanalyse. Vi satt opp interessentanalysen ved å undersøke hvor stor makt og hvor stor interesse de ulike partene hadde i prosjektet. En part med stor makt og liten interesse for prosjektet ville for eksempel vært GDPR retningslinjer. En part med stor interesse og liten makt kunne for eksempel være vår veileder ved UiA. Ved å gjennomføre denne analysen ble det i større grad klart hvilke potensielle aspekter ved prosjektet en burde ta hensyn til. Vi fikk også en større forståelse for hvem som var prosjektets målgruppe.

Som et resultat av Sprint Review 1 og diskusjon rundt interessentanalysen ble det klart at målgruppen for produktet i all hovedsak var Knowit sine egne ansatte, der vi skal bidra til å strømlinjeforme deres interaksjon med fakturaer som må godkjennes i Semine.



Figur 9: Utarbeidet interessentanalyse

3.2.3 Langsiktige mål

Som en del av kartleggingsprosessen ble vi også anbefalt å komme frem til noen konkrete langsiktige mål for produktet. Dette bidrar til å gi perspektiv på hvilke kriterier som må oppnås for at produktet skal ansees som en suksess. Det er viktig at disse målene er målbare, slik at det er enkelt å følge de opp før og etter implementering. Det er også viktig å notere her at selv om dette antageligvis ikke er noe vi vil være i posisjon til å følge opp etter implementering, så gir det fortsatt verdifullt perspektiv på målene vi ønsker å oppnå. Basert på disse kriteriene kom vi frem til følgende forslag:

- Lavere behandlingstid per faktura

Dette kan måles ved tidtaking av behandlingstid per faktura før produktet implementeres, ved bruk av Semines eksisterende løsning i nettleseren, og deretter sammenligne tidsbruk med applikasjonen vi utvikler i Microsoft Teams. Dersom resultatet er en reduksjon i behandlingstid er dette målet realisert.

- 9/10 Brukere fornøyd

Her kan man implementere et enkelt system for å hente tilbakemelding fra brukere, ved for eksempel å gi brukere, som ferdigbehandler en faktura, mulighet til å velge tommel opp eller tommel ned for å gi en generell tilbakemelding på opplevelsen med produktet. Dersom 9 av 10 tilbakemeldinger er positive, er denne å anse som oppfylt.

- Forenklet behandlingsløp

Denne er noe vanskeligere å måle, men en mulighet kan være intervjuer eller spørreundersøkelser blant ansatte før og etter implementering av systemet. Dette punktet skiller seg fra lavere behandlingstid ved at det her ikke

er fokus på tidsaspektet, men heller om brukere opplever den fornyede prosessen som forenklet, forverret eller lik. Dersom tilbakemelding fra brukere viser at de opplever den fornyede prosessen som forenklet så er dette målet oppnådd.

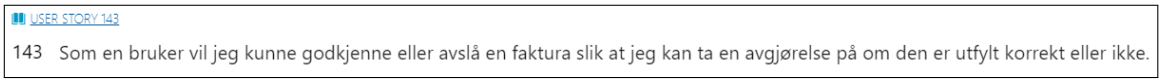
3.2.4 Scope

Etter å ha skaffet en oversikt over den totale kompleksiteten og mulig funksjonalitet i prosjektet, ble det viktig å begrense dette ned til en realistisk arbeidsmengde basert på tid og ressurser vi hadde tilgjengelig. Prosjektet vårt hadde potensiale for mye funksjonalitet. Selv om veiledere hos Knowit oppfordret til å vurdere annen funksjonalitet enn hva som ble spesifisert av dem i oppgaven, la de også vekt på at dette ikke var et krav. Av forretningsprosessen presentert ovenfor var det de seks punktene vist i Figur 10 som ble tatt med i prosjektet vårt, og la grunnlaget for hvilke brukerhistorier vi kunne definere.

Figur 10: Utarbeidet scope

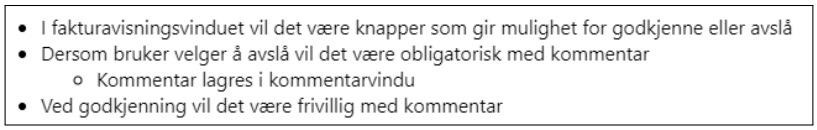
3.2.5 Brukerhistorier med akseptansekrav

Basert på informasjonen som var hentet inn fra den initielle analysen ble det utarbeidet brukerhistorier. Gruppen ble enige om å ta bruk et format for brukerhistoriene som så slik ut: “Som en X, ønsker jeg å kunne Y, slik at Z”. Slik som vist i Figur 11.



Figur 11: Eksempel på brukerhistorie

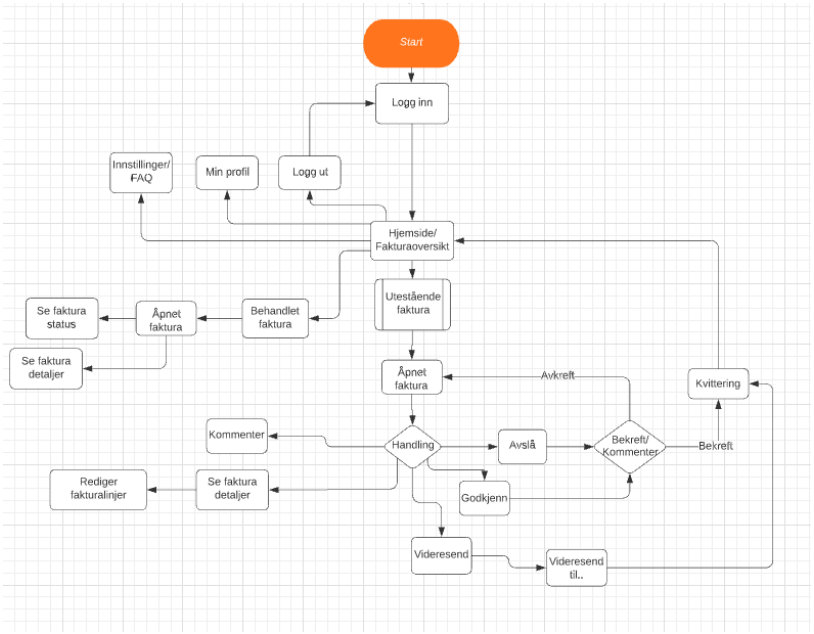
I samarbeid med Knowit ble brukerhistoriene rangert etter MoSCoW metoden. Funksjonalitet som spesifikt hadde blitt etterspurt av produkteier ble vurdert som must-have. Annen funksjonalitet som gruppen valgte å ta med i scopet ble vurdert som should-have, could-have eller would-have, fra høyest til lavest prioritet. For hver brukerhistorie ble det også utarbeidet tilhørende akseptansekrav. Akseptansekravet ga gruppen mer detaljerte beskrivelser på hva som måtte være på plass for at brukerhistoriene til slutt kunne anses som ferdigstilt. For å sikre at akseptansekravene ble oppfylt ble disse gått gjennom før endelig godkjenning av en brukerhistorie i DevOps, i forbindelse med kodegjennomgang, som skrevet om i kapittel 2.3.2 Gjennomgang av kode. Et eksempel på akseptansekrav for en brukerhistorie kan sees i Figur 12.



Figur 12: Eksempel på akseptansekrav for en brukerhistorie

3.2.6 Flowchart

Flowchart ses ofte på som steget mellom en analysedel og en designdel i et prosjekt, hvor målet er å vise frem et diagram som representerer en arbeidsflyt eller prosess (Vihovde, 2020). Gruppen erfarte dette som et godt visualiseringsverktøy som la grunnlag for videre utvikling. Det ga også muligheten for mer konkret tilbakemelding fra produkteier da den tenkte prosessen innenfor vårt scope ble mer synlig. Etter Sprint Review hvor vårt flowchart ble presentert, fikk gruppen tilbakemelding som var med å sikre et godt grunnlag for videre design av produktet. Figur 13 viser siste utkast av flowchart som gruppen tok i bruk:



Figur 13: Utarbeidet flowchart

3.3 Design av produktet

I dette kapittelet tar vi for oss hvordan designet av selve produktet ble utarbeidet. Vi vil ta for oss ulike aktiviteter vi gjorde for å utarbeide et design for frontend med brukergrensesnitt, samt hvordan vi utarbeidet systemarkitekturen i prosjektet. Målet med kapittelet er å vise kompleksiteten for de ulike komponentene (frontend, backend og database) samt vise hvordan de henger sammen.

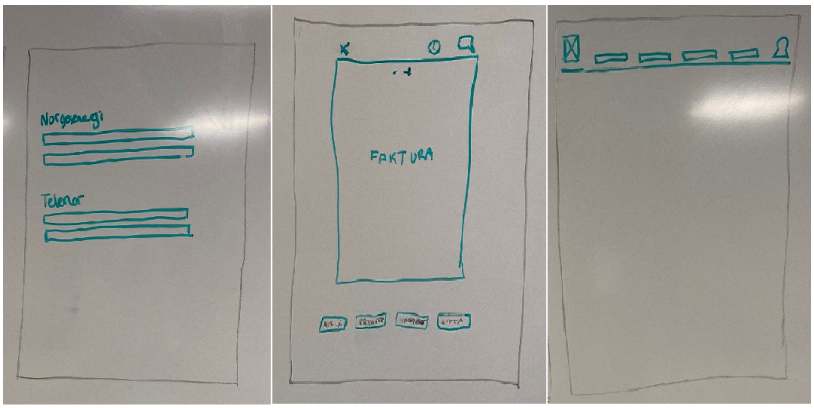
3.3.1 Brukergrensesnitt

Gjennom prosjektet har det kommet tydelig frem at hovedfokuset i stor grad har basert seg på kompleksitet rundt teknisk implementering og systemløsninger. Etter et møte med Knowit i Sprint 1 kom det derimot frem et ønske om å få testet UI design for Microsoft Teams-applikasjonen. Ønsket var å få testet designet på aktuelle sluttbrukere, for å sikre at det som ble implementert senere var av kvalitet i forhold til designprinsipper, samt at det sto opp til produkteiers forventninger. For å utforme et design for applikasjonen valgte gruppen å ta i bruk sketsjer, wireframes, mockups og til slutt en klikkbar prototype.

Skisser

For å utforme de grunnleggende konseptene for brukergrensesnittet begynte gruppen med skisser. *Skisser* ses ofte på som et av de første stegene for å forstå og løse en utfordring (Justmind, 2019). Etter Sprint Review på

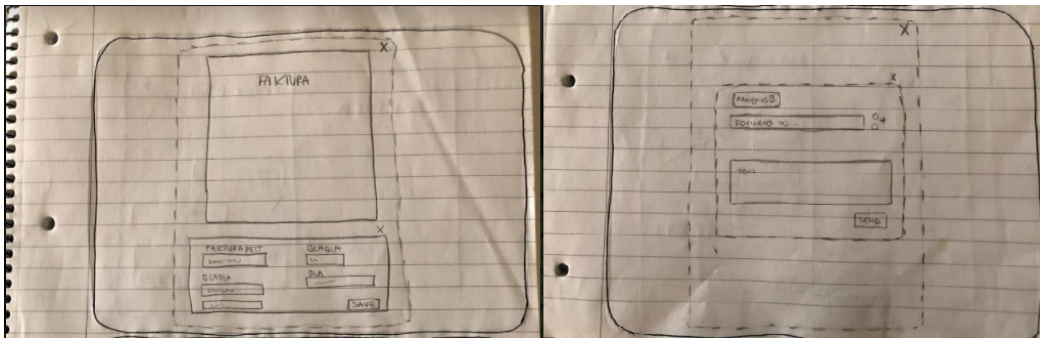
slutten av Sprint 1 sammen med produkteier og veileder hos Knowit, besluttet gruppen at grunnlaget var på plass for å begynne å skisse idéer. Gruppen tok utgangspunkt i brukerhistoriene med tilhørende akseptanse-krav, samt flowchartet som ble utarbeidet i analysedelen. Det ble gjennomført workshops med brainstorming hvor ideene ble tegnet opp på whiteboard og diskutert i plenum. Gruppen erfarte dette som en god måte å få samlet alles tanker og ideer rundt prosjektet. Figur 14 under illustrerer prosessen.



Figur 14: Eksempel på skisser illustrert på tavle

Wireframes

Gruppen valgte å ta i bruk wireframes for å bedre visualisere de tenkte design ideene rundt applikasjonen. Wireframes er i utgangspunktet en Lo Fi måte å visualisere de viktigste komponentene for en applikasjon (Planet, 2018). Gruppen valgte for dette prosjektet å lage wireframes for hånd, som ga muligheten for å raskt tegne opp ulike versjoner av en tenkt komponent. Ved å lage flere versjoner av en komponent så tidlig i designprosessen fikk vi muligheten til å kjapt få tilbakemelding fra interessenter på hvilken versjon som så best ut, samt hva som virket mest intuitivt. Se figur Figur 15. Resterende wireframes kan sees i Tillegg E.



Figur 15: To wireframes som ble tegnet for hånd.

Mockups

Mockups ble i dette prosjektet brukt som et middel til å vise frem hvordan designet faktisk ville se ut når det ble ferdig implementert. Gruppen hadde noen ferdiglagde designelementer å ta utgangspunkt i når både wireframes og mockups ble utarbeidet. En av veilederne fra Knowit som tidligere hadde arbeidet med Semine ga oss tilgang til bilder fra Semine sin mobilapplikasjon. I tillegg til bildene fant gruppen et innlegg på Behance som var laget for Semine. Behance er en sosial medieplattform som benyttes til å vise frem og oppdage kreativt arbeid (Benediktsson, 2019). Her fant gruppen ulike fargepaletter, ikoner, skriftfont og andre design ideer.
Mockupsene ble satt sammen i Adobe XD som er Adobes prototype verktøy for brukeropplevelse og interaksjonsdesignere (Valishvili, 2018). Noen av gruppemedlemmene hadde tidligere erfaring med dette programmet, og ble derfor vurdert som et veldig nyttig verktøy. Et utvalgt av mockupsene kan sees i Tillegg F.

Prototype og design-testing

Etter å ha utarbeidet de nødvendige mockupsene for prosjektet ble de satt sammen til en flow. En flow i Adobe XD tilsvarer en klikkbar prototype som gjør det mulig å navigere seg mellom komponenter ved å klikke på dem. Prototypen kan ses her.

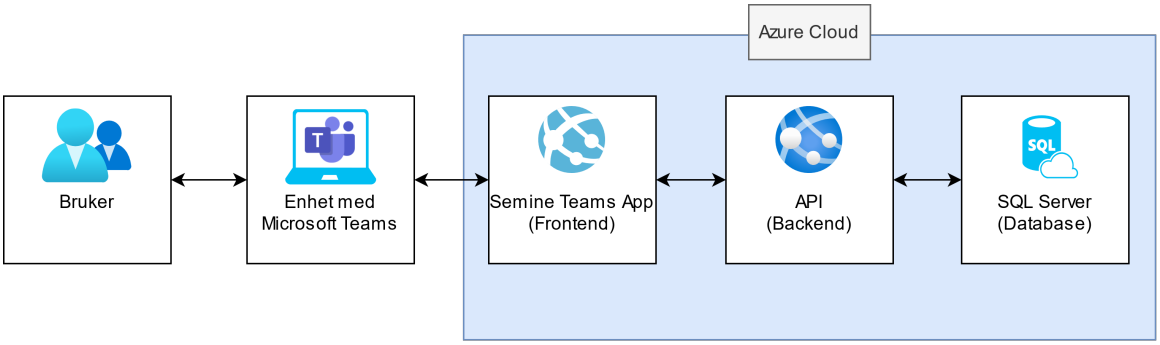
Gruppen valgte at den utarbeidede prototypen skulle bli brukt for å gjennomføre design-tester. Etter et møte med veiledere på Knowit ble vi enige om at design-testene skulle bestå av ulike caser som skulle løses. Dette ga muligheter for å teste både UI, altså hvordan nettsiden ser ut, samt UX, hvordan nettsiden faktisk fungerer. Produkteier og veileder i Knowit var begge med på å teste designet for sikre kvaliteten i prosjektet.
Prototypen med tilhørende caser ble sendt ut for design-testing i forkant av Sprint 2. Det ble lagt opp til å ha en gjennomgang av tilbakemeldingene i Sprint Review, noe som ga gruppen muligheter til å komme med eventuelle oppfølgingsspørsmål eller refleksjoner rundt endringer. Gruppen følte dette ga et godt utgangspunkt for utviklingsprosessen. Det ble i senere Sprint Review kun gitt tilbakemelding om små endringer på design detaljer, som for eksempel var å endre farge på en knapp.

3.3.2 Systemarkitektur

Tidlig i prosjektet fant vi ut at det ville være hensiktsmessig å definere en systemarkitektur for prosjektet med tilhørende visuell representasjon for både den overordnede systemarkitekturen, samt systemarkitekturen for backend-delen av prosjektet. Dette prioriterte vi ettersom vi basert på tidligere erfaringer så verdien i å ha konkrete arkitekturløsninger for å få en oversikt over et eller flere system. Vi ønsket dermed tidlig å kunne definere noen sentrale designvalg og skape en abstraksjon av systemet for å enklere kunne forstå og videreutvikle systemet ved hjelp av en systemarkitektur.

Overordnet systemarkitektur

For å danne oss en oversikt over de ulike delene av prosjektet dannet vi en overordnet systemarkitektur for å få et visuelt bilde av hvordan henholdsvis frontend, backend og databasen er koblet sammen. Dette kan ses på Figur 16. Her illustreres det hvordan en bruker igjennom Microsoft Teams på sin enhet, bruker vår Microsoft Teams applikasjon (frontend). Denne applikasjonen kommuniserer så med vårt API (backend) via HTTP-forespørsler, og backend tar seg av kommunikasjonen med databasen. Dette førte til at systemet var bestående av tre hovedkomponenter hostet i Azure skyen - frontend, backend og databasen.



Figur 16: Overordnet systemarkitektur

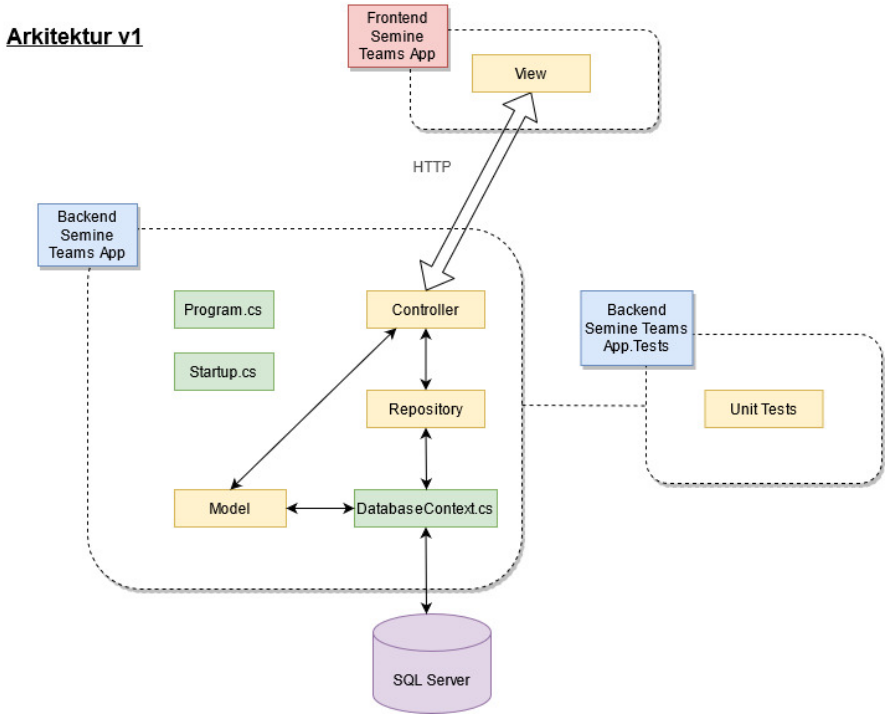
Systemarkitektur for backend

Ettersom vi har brukt det agile rammeverket Scrum, har vårt hovedfokus rundt systemarkitektur for backend vært å beholde agilitet i prosjektet ved å tilrettelegge for et dynamisk design, med fokus på at hver komponent (henholdsvis klasse) skal ha sitt eget ansvarsområde og ha liten grad av avhengighet til andre komponenter. Dette gjør det enklere å videreutvikle systemet, da man ikke skal behøve å modifisere allerede eksisterende kode for å legge til nye komponenter. Vi har dermed valgt å ikke foreta en *Big Design Up Front*, som handler om å perfeksjonere arkitekturen på forhånd, en metodikk som ofte er tilknyttet fossefallsmetoden, men heller legge til rette for en agil prosess med utgangspunkt i en enkel arkitektur med lav coupling (Martin, 2008).

Definering av arkitektur for backend

For å imøtekomme de kravene vi satt oss definerte vi en *n*-lags arkitektur, som følger Model-View-Controller prinsippet hvor hvert lag har sitt eget definerte ansvarsområde (Gamma et al., 1994, 14-16). Vi tok også i bruk et designmønster kalt *repository pattern* hvor det laveste nivået i arkitekturen inneholder all kode for interaksjon med databasen. Over Repository-laget ligger "Controller"-laget hvor kommunikasjon mellom applikasjonen i frontend og API-et i backend foregår via HTTP-forespørsler. For å representere tabellene i databasen benyttet vi oss av domene-modeller, som hver for seg representerte et aggregat i databasen. Arbeidet med

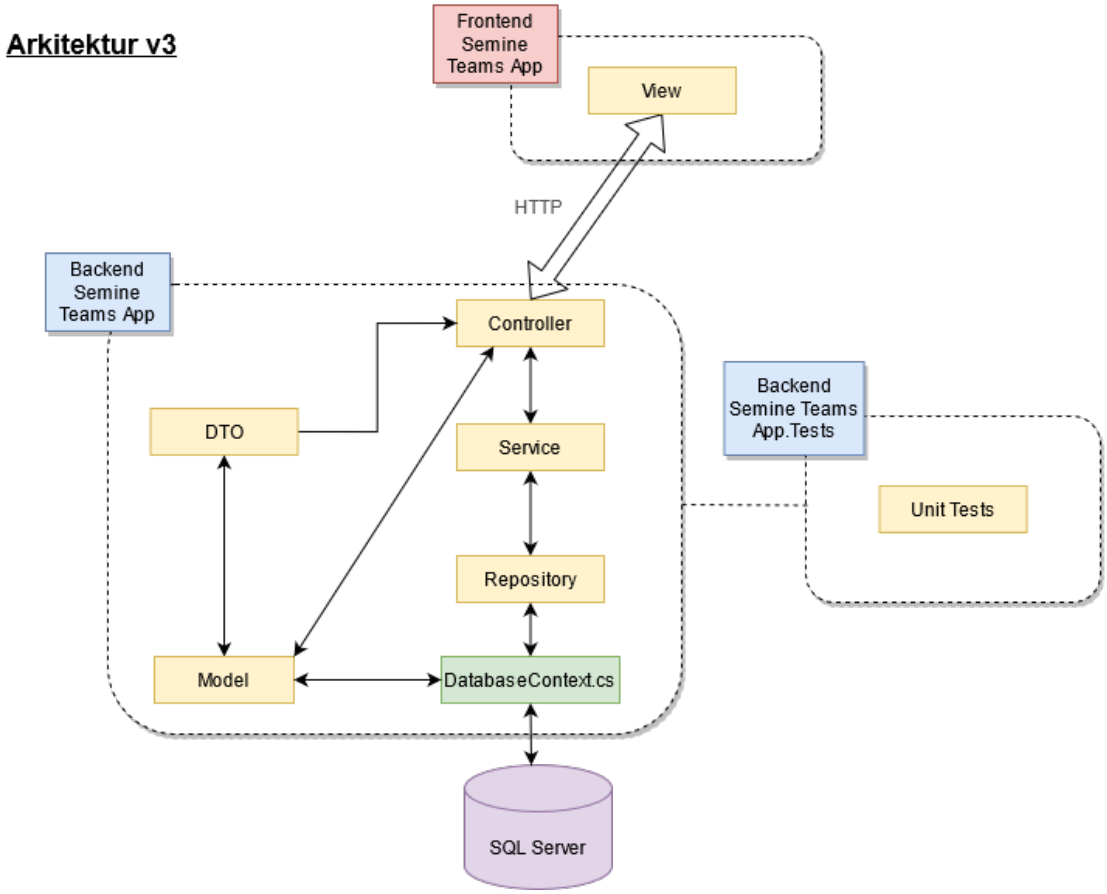
å definere systemarktiekturen har blitt gjennomført kontinuerlig i prosjektet, og den første designskissen av systemarkitekturen kan ses på Figur 17.



Figur 17: Utgangspunkt for systemarkitektur

Endringer i arkitektur for backend

Ved vår andre Scrum Review fikk vi tilbakemelding fra Knowit om å gjøre noen endringer i systemarkitekturen. Dette innbefattet å inkludere et lag i arkitekturen mellom "Controller" og "Repository"-laget for å imøtekomme Robert. C. Martins Single-responsibility principle (Martin, 2008). Dette gjorde vi fordi Controller-laget tidligere hadde hatt ansvar for både logikk og kommunikasjon med frontend. Vi inkluderte dermed et "Service"-lag i arkitekturen hvor all business-logikken skulle være plassert, som f.eks. kall på tvers av "repositories". Ettersom vi hadde dannet en arkitektur med utgangspunkt i et agilt og dynamisk design, var det enkelt å legge til et ekstra lag i arkitekturen. Knowit ønsket også at vi skulle ta i bruk Data Transfer Objects (DTO) for å skille mellom data vi forholder oss til i domenet, og data som blir transportert til og fra klient via API-kall. Oppdatert systemarkitektur kan ses på Figur 18.



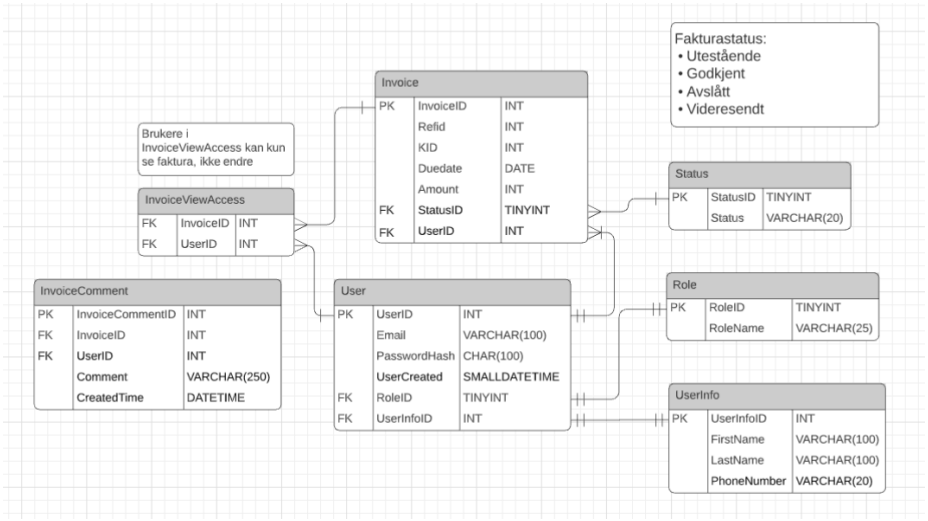
Figur 18: Systemarkitektur etter endringer

Konsekvenser av økt fokus på systemarkitektur

Valgene vi har tatt relatert til systemarkitektur har ført til at vi som utviklere enkelt har kunnet danne oss en oversikt over hvor hvilke komponenter hører til i systemet. Dette har ført til at tiden mellom definering av oppgave og implementering av kode har blitt kort. Siden vi har delt opp arkitekturen i flere lag basert på ansvarsområde har det vært mulig å jobbe agilt med stadig nye oppgaver og endringer, ettersom det har vært enkelt å legge til og fjerne komponenter. Enhetstesting av kode har også vært betraktelig enklere enn hva det ville vært om alt av kode lå i samme lag, da vi har hatt mulighet til å enkelt skrive isolerte tester til hver individuelle komponent, se kapittel 3.4.2 Testing for flere detaljer rundt dette.

3.3.3 Database

Databasen ble utviklet gradvis ettersom vi så behovet for å legge til tabeller og relasjoner mellom tabellene. I starten av arbeidet med prosjektet designet vi et ER-diagram (Entity Relationship) med formål å kartlegge og visualisere alle de nødvendige tabellene, og hvordan disse tabellene skulle forholde seg til hverandre. I tråd med vår agile tilnærming, var vi klare over at vårt ER-diagram sannsynligvis ikke vil samsvare helt med hvordan databasen kom til å bli ved prosjektets slutt. Allikevel var det nyttig for oss å ha et utgangspunkt å bygge på, og underveis i prosjektet har vi ikke opplevd at designet av databasen har avviket stort fra vårt opprinnelige design. Noen strukturelle endringer har blitt gjennomført underveis, f.eks. at vi har lagt til og slettet noen tabeller, men databasens opprinnelige designformål har forblitt ivaretatt.



Figur 19: ER-diagram

Et av hovedfokusene, ved siden av å være funksjonell i systemet, var å sørge for at databasen ville fungere godt med testing og eksperimentering. Dette la vi til rette for bl.a. ved å legge til realistiske verdier og navn på all testdata, slik at spørringene hentet ut data som er tydelig og beskrivende. I tillegg opprettet vi to identiske databaser, der formålet var at den ene skulle fungere som en database der man kan kunne teste ut scripts eller spørringer med høy grad av usikkerhet, slik at man ikke kom til skade for å ødelegge funksjonalitet mot resten av systemet. Den andre databasen hadde som formål å være vår primærdatabase som skulle være koblet opp mot resten av systemet.

3.4 Implementering

Gruppen har gjennom hele prosjektet hatt mye fokus på selve implementeringen og programmering i prosjektet. I dette kapittelet vil vi ta for oss gjennomførelse og erfaring med selve programmeringen, samt også testing og deployering.

3.4.1 Programmering

Ettersom teknologien og utviklingsspråkene vi valgte for dette prosjektet var relativt nye for alle på gruppen var det naturlig å sette av tid til opplæring. Frontend-utviklerne kunne da velge å ta kurs innen JavaScript og React, og backend-utviklerne kunne ta kurs innenfor C#. Selv om det var delt inn på denne måten var omstendighetene relativ åpne og medlemmene kunne velge selv å ta et eller flere av kursene. Det ble også lagt opp for generelle gjennomganger av de ulike temaene som gjorde at det var mer naturlig for alle medlemmene å delta i diskusjoner som omhandlet disse temaene.

Oppgavefordeling

Som nevnt tidligere i rapporten ble det laget oppgaver for hver av brukerhistoriene. Erfaringen vi har med programmering er at man blir mye mer motivert hvis man jobber med noe man synes er gøy og utfordrende, og derfor ble oppgavene ofte valgt ut ifra interesse. Vi erfarte at dette var en veldig god måte å dele inn oppgaver på for å sikre både læring og motivasjon innad i prosjektet. Den økte motivasjonen førte derfor også til mer effektivt arbeid hos de enkelte gruppemedlemmene, da de fikk jobbe med oppgaver de var interessert i og følte de mestret.

Parprogrammering

For å dele kunnskap og for å hjelpe hverandre valgte vi å prøve ut parprogrammering. Vi delte dette inn i backend og frontend for å maksimere effektivitet. Dette ble både gjort over Discord og ved fysiske møter. Ved å ta i bruk Discord kunne vi dele oss inn i «grupperom» som ofte førte til at backend teamet satt i det ene rommet, mens frontend satt i et annet. Det var på denne måten enkelt og hoppe over i en annen samtale for å diskutere og spør om hjelp på tvers av teamene. Fysiske møter var derimot ofte å foretrekke da det i noen tilfeller økte motivasjon og effektiviteten mer enn digitale møter, da samarbeidet ble mer flytende og effektivt.

En liten utfordring vi møtte på i løpet av prosjektet var at flere av oppgavene i Sprint Backlog var tett relaterte, og dette førte til at man kunne bli sittende i en situasjon der man måtte vente på en annen utvikler. Parprogrammering løste denne utfordringen ved at vi kunne programmere sammen ved å bruke LiveShare i Visual Studio Code. Dette er en løsning som lar flere utviklere jobbe på samme fil virtuelt. I noen tilfeller var

en eller flere oppgaver avhengig at en annen oppgave var ferdigstilt og ble pushet til brukerhistorie branchen i Git Repo. Ved å benytte oss av parprogrammering kunne vi ferdigstille disse oppgavene kjappere for å sikre fremgang i prosjektet. Videre sørget vi også for å tildele tett sammenkoblede oppgaver til samme utvikler i de kommende Sprintene, slik at man ikke var direkte avhengig av andre for å fullføre sine oppgaver.

Mot slutten av prosjektet har gruppens mål vært å ferdigstille alle must-have brukerhistoriene fra vår Product Backlog, og etter å ha gjennomført Sprint Review 4 nærmer vi oss i stor grad.

3.4.2 Testing

Testing har vært et sentral tema i arbeidet for å sikre høy kvalitet på implementeringen i prosjektet, men har også vært et av temaene som har vært utfordrende i den grad at vi tidligere har hatt lite erfaring med testing av kode.

Enhetstesting

For å kunne isolere komponenter i koden i den grad at det er mulig å skrive enhetstester for selvstendige komponenter, er "mocking", eller imitering av komponenters avhengigheter et sentralt tema i enhetstesting (Hamill, 2004). Ettersom vi har delt opp backend i flere lag og brukt repository pattern, som forklart i kapittel 3.3.2 Systemarkitektur, har det vært enkelt å erstatte avhengigheter med imiterte testklasser. Dette har gjort arbeidet med å skrive tester etter F.I.R.S.T. prinsippet (se kapittel 2.3.3 Testing), enklere i den grad at vi har kunnet isolerer komponenter tilstrekkelig for å skrive gode enhetstester.

Vi startet med å skrive flere enhetstester for hver metode i Controller-laget, og imiterte alle avhengighetene til metodene her. Da vi endret systemarkitekturen og inkluderte et ekstra lag i systemarkitekturen som nevnt i kapittel 3.3.2 Systemarkitektur endret vi testene slik at de imiterte det nye underliggende Service-laget også. Dette medførte til at enhetstestene ikke bare testet en metode i en klasse, men også metodens avhengigheter. Dermed var ikke lenger enhetstestene våre enhetstester i ordets rette betydning, da de testet mer enn bare én enhet.

Etter anbefaling fra Knowit, endret vi test-strukturen i den grad at vi laget individuelle tester for hver metode i både Controller-laget, og Service-laget, hvorav hvert av lagene imiterte alle underliggende lag. Dette sørget for at enhetstestene igjen kun testet én enkelt enhet, og gjorde det enklere å både forstå hva som ble testet, og å avdekke hvor eventuelle feil lå dersom testene feilet.

3.4.3 Automatisering og Deployering

Som nevnt i 2.2.5 så valgte vi å bruke Azure Pipelines for å automatisere bygging og deployering av koden. På denne måten sparte vi både tid og ressurser på å få ferdig kode opp til kjørende produksjonsmiljø. Dette er en nyere funksjonalitet i Microsoft Azure-miljøet, og det er derfor begrenset dokumentasjon på hvordan man setter dette opp. Vi hadde et kort opplæringsmøte med Knowit, men selv etter dette møtet tok det mye tid å få dette til å fungere for både frontend og backend, da det er mange komponenter som må kobles sammen og tilpasses hvert unike prosjekt.

Vi fikk dette til slutt til å fungere, da med to pipelines for hvert repository: én "build pipeline" og én "release pipeline". Build pipeline kjøres automatisk idet en ny kode aksepteres inn i "main repository", slik at vi vet at den nye koden er gått igjennom og overholder kvalitetskrav. Deretter kjøres release pipeline automatisk når en ny "build" fra build pipeline er tilgjengelig. Denne kjører da koden opp til Azure Cloud, der begge kjøres i produksjon slik at de kjører kontinuerlig og er tilgjengelig fra en offentlig URL.
Selv om dette var komplisert å få til å fungere, så var det utrolig nyttig og spennende å lære. Dette er kompetanse som er høyt ettertraktet i arbeidsmarkedet, og har gitt oss en god forståelse for hvordan Microsoft Azure Pipelines fungerer i praksis.

3.5 Utfordringer

Når man gjennomfører et stort prosjekt er det naturlig at det følger med ulike utfordringer. Selv om gruppen hadde gode forutsetninger for prosjektet var det ikke til å unngå at både forutsette og uforutsette utfordringer ville oppstå. Gjennom dette vårsemesteret har gruppen møtt på flere utfordringer, hvor de mest sentrale blir beskrevet i delkapitlene nedenfor.

3.5.1 Tekniske Utfordring

Det å utvikle et system i nye programmeringsspråk og miljøer kan være utfordrende. Vi har vært nødt til å sette oss inn i blant annet C# og JavaScript, samt hvordan å utvikle en applikasjon som skal kjøres i Teams med tester og automatisert deployering i Azure, noe ingen av oss i stor grad har jobbet med tidligere. Applikasjoner som skal utvikles for Microsoft Teams er relativt lite utbredt, og det var en utfordring å finne dokumentasjon på hvordan dette skulle gjennomføres. Vi møtte på utfordringer her da dette heller ikke er et kjent område for Knowit.

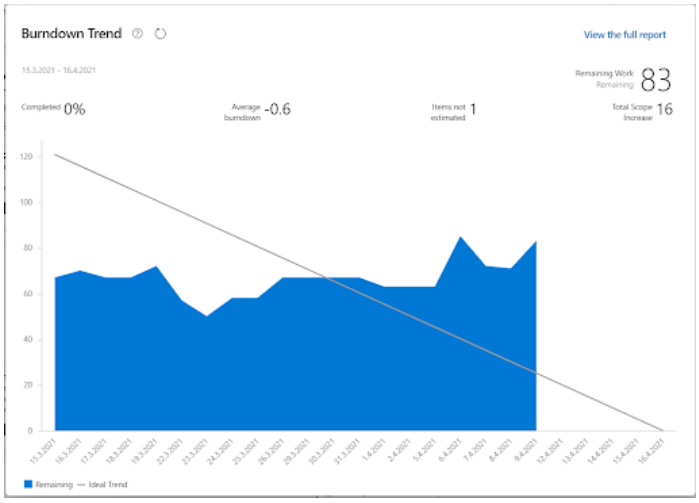
Muligheten for "Single Sign-on" var en av de viktigere funksjonalitetene oppgitt av produkteier tidlig i prosjektet og baserer seg på sømløs integrering med Teams, hvor brukeren kun må oppgi innloggingsdetaljer én gang; når de logger inn i Teams. Dette har vist seg å være relativt komplisert, og fordi Knowit ikke selv har

erfaring med produksjon av Teams-applikasjoner har vi brukt mye tid på å utforske dette området. En av de større utfordringene her har vært relatert til rettigheter i Azure-miljøet der dette er sentrert. På tross av dette har vi som mål å få dette på plass før utgangen av siste Sprint.

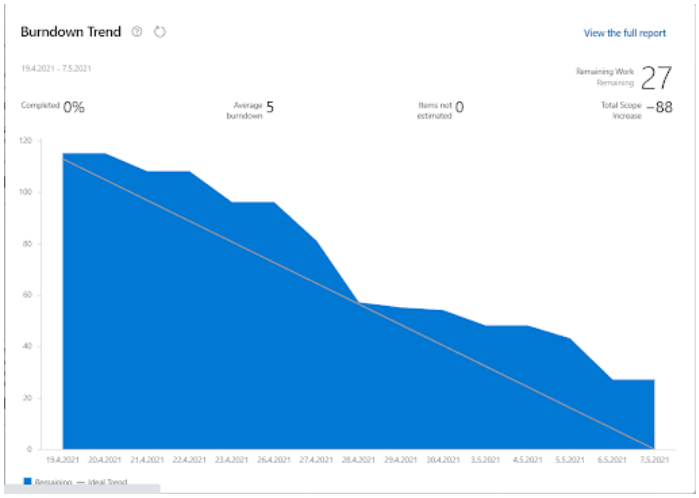
3.5.2 Tidsestimering

En av de tingene som har vist seg å være utfordrende både i starten og gjennom prosjektet har vært tidsestimering. En av de største utfordringene tidlig i prosjektet var å estimere vanskelighetsgrad og tid for en brukerhistorie. Utfordringen lå mye i at gruppen ikke har mye erfaring med utvikling. Når man ikke har mye erfaring er det vanskelig å kunne estimere hvor lang tid ting vil ta, da det er mye uventet som kan skje og vanskelighetsgraden på noe kan være større en først antatt.

I tillegg til estimering av brukerhistoriene har gruppen hatt litt utfordringer med estimering av enkelte oppgaver som er knyttet mot brukerhistoriene. Igjen ligger mye av utfordringen i at vi har lite erfaring som derfor gjør det vanskelig å estimere.
En av utfordringene vi møtte på i starten av prosjektet var at vi lagde oppgaver som var større enn ett dagsverk på åtte timer. Oppgavene vi først delte opp i var relativt store og tilhørende estimert tid ble derfor mye mer enn bare et dagsverk. Dette hang mye sammen med at brukerhistorien ikke ble ordentlig avdekket i starten av Sprinten, som igjen førte med seg at vi ofte oppdaget nye oppgaver underveis i Sprinten som vi måtte legge til. Det var derfor vanskelig å se fremgang i prosjektet ved å studere Burndown Chart. Det var ikke før senere i prosjektet at vi i en forelesning med veileder Hallgeir ble klar over konseptet med å ikke estimere mer enn et dagsverk. Dette krevde at vi måtte nedjustere arbeidsmengden slik at oppgaven oppfylte kravet om å være mindre enn ett dagsverk. Dette førte til at mer av en brukerhistorie ble dekket tidlig i Sprinten, og vi slapp å legge til oppgaver underveis. Vårt Burndown Chart viste nå mye bedre flyt og fremgang i prosjektet. Figur 20 viser Burndown Chart før vi gjennomførte tiltak. Figur 21 viser Burndown Chart etter tiltak.



Figur 20: Burndown Chart før oppdeling av oppgaver



Figur 21: Burndown Chart etter oppdeling av oppgaver

Selv om vi mot slutten av prosjektet har begynt å se litt fremgang er det ikke til å legge skjul på at estimering har vært utfordrende del av prosjektet.

3.5.3 Covid-19

Covid-19 og de medfølgende restriksjonene var en av de utfordringene gruppen til en viss grad var klar over. Semesteret startet forholdsvis lovende med at gruppen kunne møtes fysisk og sitte sammen og jobbe, da med 2-meters avstand. I løpet av Sprint 1 oppsto det derimot en ny bølge som førte til at Universitetet i Agder

stengte og kontorene hos Knowit ble lukket. Det ble nå innført hjemmekontor noe som gruppen til en viss grad var kjent med. Selv om overgangen til å ha hjemmekontor forholdsvis gikk greit da vi tidligere hadde erfaring med det, var motivasjonen til gruppen det som var mest utsatt. Vi har generelt merket konsekvensene av pandemien og vi begynte å merke på ulempene med å sitte alene hjemme store deler av dagen. Det har derfor vært utfordrende i den forstand, men gruppen har jobbet med å motvirke dette ved å gjennomføre ulike aktiviteter og tiltak.

Når man jobber på agile prosjekter og tar i bruk Scrum-rammeverket er det mye som legges opp til at gruppen skal sitte samlokalisert og arbeide. Derfor har det vært kritisk å etablere gode måter for å jobbe sammen digitalt. Mange av disse tiltakene har blitt nevnt i prosjektet som for eksempel å ta i bruk grupperom på Discord og Liveshare i Visual Studio og Visual Studio Code. Det å ha gode kommunikasjonsverktøy hvor man lett kan interageren med hverandre har vært med å holde motivasjonen oppe og man føler ikke lengre at man sitter alene og jobber.

Daily Standups og Sprint Reviews har motivert gruppen til og møtes og jobbe kontinuerlig digitalt. Ved Sprint Planlegging har gruppen hatt mulighet til å planlegge en Sprint og på denne måten satt oss mål som vi har lyst til å oppnå. Sprinter med satte tidsfrister har også være med å motivere gruppen til å jobbe kontinuerlig. For å oppsummere så har faste rammer å forholde seg til vært med å motivere i en ellers krevende periode hvor man fort kan ende opp umotivert og tafatt.

På den sosiale fronten har gruppen gjennomført såkalte teambuildings, innenfor de satte restriksjonene, for å holde motivasjonen oppe. Når det har vist seg passende har vi som nevnt tidligere i rapporten møttes for å jobbe fysisk, enten med å gjennomføre workshops eller parprogrammering. Det og av og til møtes fysisk har i stor grad vist seg å være motiverende.

4 Refleksjon og oppsummering

Avslutningsvis vil vi i dette kapittelet presentere en oppsummerende refleksjon knyttet til prosessen rundt bachelorprosjektet. Her nevner vi temaene vi har jobbet med, og reflekterer hvordan vi, underveis i prosessen, har tatt til oss nye læringsmetoder, kunnskap og erfaringer. Bachelorprosjektet har vært annerledes enn noe vi har gjennomført tidligere på studiet, og med dette oppsto det nye læringsrike utfordringer som har ført til at vi har utviklet vår kompetanse og allsidighet som studenter innen IT og informasjonssystemer.

4.1 Prosjektstyring

Med tanke på at bachelorprosjektet har vært et prosjekt som har krevd en stor grad av samarbeid, har det vært en høy prioritering å legge til rette for at samarbeidet skal enkelt la seg gjennomføre. Ved hjelp av Scrum som arbeidsmetodikk, og det faktum at gruppemedlemmene er godt kjent med hverandre fra tidligere samarbeid på IT-studiet, sitter alle igjen med et inntrykk av at samarbeidet har fungert godt. Ved hjelp av fundamentale Scrum-tiltak, som f.eks. Daily Scrum og Sprint Review, har vi sikret god kommunikasjon og godt planlagte Sprinter. Dette har også lagt til rette for at vi raskt har hatt mulighet til å hjelpe hverandre der vi har sittet fast, og vi har også opplevd at dette hadde en positivt effekt på motivasjon. Videre har dette bidratt til å innføre rutiner og sikre fremgang i en ellers uforutsigbar arbeidshverdag under den pågående pandemien.

4.2 Kvalitetssikring

Et stort fokus gjennom hele prosjektet har vært å etterstrebe og sikre kvalitet i alle ledd. Derfor har vi hatt et bevisst forhold til dette siden prosjektets begynnelse, og implementert en god del tiltak for å bevare dette under hele prosessen.

Dette har gitt oss god og verdifull erfaring med både arbeidsmetodikker og verktøy som vedgår kvalitetssikring, og det har blitt høyt verdsatt av både produkteier og teamet. Blant annet har vi sørget for å ta valg basert på tidligere erfaringer fra produkteier, bransjestandarder og tilgjengelig dokumentasjon. Videre har vi jobbet med kodestandarder, utviklingsrammeverk og testing for å etablere en gjennomgående kvalitetssikring i både arbeidsprosessen og det endelige produktet. Selv om vi har hatt flere teknologiske utfordringer underveis, har dette lagt et grunnlag for hvordan vi har gått frem for å løse disse utfordringene. Dette er verdifull erfaring å ta med videre da det har lært oss mye om hvordan man kan sikre fremgang i et utviklingsprosjekt på tross av utfordringer man møter underveis.

4.3 Kompleksitet

Oppgaven vår har vært preget av nye teknologier med begrenset tilgjengelig dokumentasjon. Vår samarbeidspartner Knowit har ikke mye erfaring med utvikling av en Teams applikasjon, så dette krevde at gruppen måtte ha en mer selvstendig tilnærming til prosjektet. Dette har ført til at mye tid og ressurser har gått til opplæring og innføring i nye aspekter. Selv om dette har vært en utfordring, har motivasjonen vært høy da teknologien vi har benyttet er fremvoksende og kompetansen er høyt etterspurt. Ettersom teknologien er relativt ny og det derfor er begrenset med tidligere prosjekter å basere vårt produkt på, så har vi fått muligheten til å stå relativt fritt ved valg av løsning. Dette har vi opplevd at har ført til et større læringsutbytte enn hvis vi hadde basert oss på en allerede eksisterende mal.

Generelt for dette prosjektet sitter vi igjen med godt læringsutbytte og gode erfaringer som vi kommer til å ta med oss videre i arbeidslivet.

5 Selvevaluering

Jørgen

I dette prosjektet har jeg først og fremst vært dedikert frontend-programmerer. I den forbindelse har jeg jobbet mest med ReactJS og implementering av brukergrensesnittet til applikasjonen. Videre har jeg vært Scrum master i prosjektet og har stått ansvarlig for å følge opp Scrum-ritualer slik som Daily Standup, Sprint Planlegging og Retrospective.

Kirsti

I dette prosjektet har jeg i stor grad hatt fokus på utvikling og utbedring av design for applikasjonen. I denne sammenheng hadde jeg også ansvar for design testing. Jeg har gjennom semesteret bidratt med enkel implementering på frontend, samt også bidratt med å skrive enhetstester for backend. Mye av mitt fokus har også gått til administrering og prosjektstyring rundt prosjektet og rapporten.

Magnus

I dette prosjektet har jeg i all hovedsak vært medansvarlig i utvikling av vårt backend-system. Det har innebært utvikling av API-endepunkter og generell systemarkitektur med C# som kodespråk i Visual Studio. Jeg har også vært en bidragsyter i administrative oppgaver, utarbeidelse av den endelige rapporten samt fremføringer i forbindelse med Sprint Review.

Markus

Jeg har vært ansvarlig for defineringen av systemarkitekturen i prosjektet, samt arbeidet med systemutvikling på backend. Jeg har også vært involvert i konfigurasjonen for automatisk deployering til Azure Portal. I tillegg til dette har jeg vært ansvarlig for bruken av Git som versjonskontroll, og har utarbeidet og vedlikeholdt en manual for hvordan vi skal bruke Git i vårt prosjekt.

Ola

Mitt hovedansvar i prosjektet har vært å bygge og drifte databasen. Dette betyr at jeg har kontinuerlig jobbet med SQL for å bygge tabellene i databasen, og angi de nødvendige relasjonene mellom tabellene. I tillegg til dette har jeg bistått der det har vært behov, som f.eks. på Sprint Review og ved dokumentasjon av Scrum-prosesser.

Stian

I dette prosjektet har jeg vært en dedikert frontend utvikler og mitt hovedansvar har da vært frontend. Jeg har jobbet med JSX og ReactJS, og har ved bruk at disse verktøyene implementert brukergrensesnittet i vår applikasjon. I tillegg har jeg bistått med de administrative oppgavene, den initielle analysen, fremføringer i både emnet og Sprint Review, samt utarbeidelse av rapporten.

Referanser

Atlassian. (2021, 11. mai). *What is version control?* https://www.atlassian.com/git/tutorials/what-is-version-control

Benediktsson, A. (2019, 21. januar). *Adobes plattform for deling av kreative prosjekter*. http://www.dataporten.net/adobe-behance/

Digitaliseringsdirektoratet. (2019, 3. mai). *Planlegge*. https://www.prosjektveiviseren.no/hva-er-prosjektveiviseren/planlegge

Geihsler, B. (2014, 22. august). *Why I Don't Teach SOLID*. https://qualityisspeed.blogspot.com/2014/08/why-i-dont-teach-solid.html

Hamill, P. (2004). *Unit test frameworks: tools for high-quality software development*. "O'Reilly Media, Inc."

Justmind. (2019, 9. juli). *Beginner's guide to UI sketching. Justmind*. https://www.justinmind.com/blog/ui-sketching/

Mahnič, V. & Hovelja, T. (2012). On using planning poker for estimating user stories. *Journal of Systems and Software*, *85*(9), 2086–2095.

Martin, R. C. (2008). *Clean Code: A handbook of Agile Software Craftsmanship*. Pearson Addison-Wesley.

Microsoft. (2019, 31. mai). *What is Azure Pipelines? - Azure Pipelines*. https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops

Microsoft. (2021a). *Build secure REST APIs on any platform with C#.* Hentet 8. mai 2021, fra https://dotnet.microsoft.com/apps/aspnet/apis

Microsoft. (2021b, 18. mars). *Get started with Microsoft Teams app development*. Hentet 8. mai 2021, fra https://docs.microsoft.com/en-us/microsoftteams/platform/build-your-first-app/build-first-app-overview

Microsoft. (2021c, 21. september). *What is the Azure SQL Database service?* https://docs.microsoft.com/en-us/azure/azure-sql/database/sql-database-paas-overview

Planet, U. (2018, 26. juni). *Wireframe, Mockup, Prototype: What is What? uxplanet*. https://uxplanet.org/wireframe-mockup-prototype-what-is-what-8cf2966e5a8b

Pyram, J. (2020, 14. september). *6 Principles Of Software Engineering That Every Engineer Should Know*. https://medium.com/swlh/6-principles-of-software-engineering-that-every-developer-should-know-7868f362b633

React. (2021, 8. mai). *A JavaScript library for building user interfaces*. Hentet 13. mai 2021, fra https://reactjs.org/

Rubin, K. S. (2012). *Essential Scrum: A practical guide to the most popular Agile process*. Addison-Wesley.

Schaefer, H. (2014). Systematisk Testing av Software. Hentet 23. april 2021, fra https://www.sans.org/reading-room/whitepapers/sysadmin/port-knocking-basics-1634

Semine. (2021). *The first mover in providing full automation for accounting*. https://www.semine.com/

Spataro, J. (2020, 5. mars). *Vi ønsker å hjelpe kundene under COVID-19*. https://www.microsoft.com/nb-no/microsoft-365/blog/2020/03/05/our-commitment-to-customers-during-covid-19/

Sutherland, J. & Schwaber, K. (2007). The scrum papers. *Nuts, Bolts and Origins of an Agile Process*.

Unit. (2020, 19. november). *Pådriver for digital utvikling*. https://www.unit.no/om-unit

Valishvili, L. (2018, 20. desember). *Design with Precision – An Adobe XD Review*. https://www.toptal.com/designers/adobe/adobe-xd-review

Vihovde, E. H. (2020, 22. januar). *Flytskjema*. https://snl.no/flytskjema_-_IT

Wikipedia. (2019, 22. februar). *Forretningsprosess*. https://no.wikipedia.org/wiki/Forretningsprosess

A Uttalelse fra oppdragsgiver

knowit

Uttalelse fra Knowit Sør

Knowit Sør har samarbeidet med Semine AS over flere år. Etter suksess med utvikling av en ny mobil-app for forenklet fakturabehandling, ble muligheten for å lage en tilsvarende app som kjører i Microsoft Teams diskutert. En slik form for forenklet behandling av fakturaer på en plattform i vekst ble vurdert som en fornuftig oppgave for et bachelorprosjekt.

Knowit beskrev oppgaven som følger: Lag en applikasjon for å godkjenne fakturaer fra Semine i Microsoft Teams.

Prosjektet

Ved prosjektstart avklarte vi forventingene våre til studentene. Studentene skulle følge en god utviklingsprosess og lage en applikasjon med en ren kildekoderepository og god testdekning. Knowit legger i sine oppgaver større vekt på prosessen for å komme frem til et resultat enn selve resultatet. Det er viktig studentene skjønner hva som skal til for å lage et solid og ordentlig produkt.

I tillegg etterspurte vi en "utredning" som beskriver best practice / howto for å lage en Teams applikasjon da dette er forholdvis ukjent for mange.

Vi vurderer at studentene har utmerket seg med følgende:

- De organisert seg selv og viste god fremdrift gjennom hele prosjektet. Gruppen har tatt ansvar og det virker som de har hatt godt samarbeid og god fremdrift gjennom hele prosjektet.
- De fulgte en ordentlig SCRUM-prosess og brukte en god branching-strategi med merging via pull requester. Azure og Git ble brukt med god effekt.
- Flere teknologier benyttet i prosjektet var nye for enten samtlige eller flere av studentene, bl.a. React, .Net Core, Azure pipelines samt SSO løsninger. De lærte samtidig mye på egenhånd og på selvstendig vis, og denne utfordringen hindret ikke prosjektets fremdrift. Vi ser det som veldig positivt at gruppen har håndtert utfordringer med oppsett av CI/CD miljø, oppsette av Azure samt Single Sign On mellom Teams og Semine på en god måte.
- Et kjørende prosjekt ble lagde med rene kildekode-repositories. Backend koden har god testdekning. Appen inkluderer også alle etterspurt funksjonalitet fra oppgaven.
- På første sprint ble det gjennomført en god analyse/planleggingsprosess og en imponerende proof-of-concept ble utarbeidet.
- Gruppen har vært nysgjerrige gjennom prosjektet og stilt gode spørsmål rundt arkitektur og design av softwareløsninger

knowit

Vi har ingen negative bemerkninger utover to liten forslag til forbedringer

- Frontend-repository kunne ha vært bedre organisert. Dersom frontend ble utvidet, ville dagens mappestruktur vært vanskelig å vedlikeholde.
- Det er ikke kjent hvor mye peer reviewing som ble utført eller hvorvidt PR-er ble godkjent uten kommententarer. Ut ifra observasjoner i repositoriet ser det ut som det er laget gode og ryddige pullrequester som har blitt godkjent uten særlige kommentarer (noe som for så vidt er gjennomgående for ferske utviklere).

Oppsummering

Teamet har laget en app av god standard og samarbeidet i teamet har vært imponerende. Det gjenstår litt før app-en er produksjonsklar, men de har fått alt rundt rammeverket på plass, noe som var en stor del av prosjektet. Studentenes programmeringsferdigheter og forståelse for utviklings- og testmetodikk har utviklet seg godt og spørsmålene rundt slik metodikk og programmeringsstil viser at studentene har god innsikt.

Vi vurderer prosjektet som en suksess, og studentene har i høyeste grad møtt våre forventninger både når det gjelder prosess, kommunikasjon og teknisk gjennomføring.

James McRedmond
Veileder

Tor Oskar Wilhelmsen
Fagleder

B Gruppekontrakt

Gruppekontrakt

Generelt

Oppmøte

- Gruppen vil ha daglige møter (stand-ups) på mandag, onsdag og fredag for å sørge for jevn progresjon og fremgang i prosjektet
    - Disse møtene vil være hovedarena for spørsmål og utfordringer underveis og være åpen for diskusjon
- Møtet starter 10:15 de satte dagene og har forventet varighet til 11:15
- Det vil tilstrebes at i hvert fall et av disse møtene hver uke gjøres fysisk dersom det lar seg gjøre
    - Forslag: fysisk møte hver mandag
- Gruppen har som mål å fylle de forventede arbeidstimene iløpet av en uke. Dvs. at vi individuelt jobber minst. 26 timer med Bachelorprosjektet IS-304. Gruppemedlemmene har selv ansvar for å fylle de forventede timene med arbeid i løpet av en uke, men det holdes oversikt over i Scrum-boardet i Devops.

Ambisjonsnivå

- Vi er en gruppe med høyt motiverte medlemmer som streber etter å oppnå det beste mulige resultatet gjennom arbeid med svært høy kvalitet.
- Karaktermessig er målsettingen å oppnå A for alle gruppemedlemmene.

Kvalitetssikring

Kode

- For å sikre høy kvalitet og pålitelighet i kode vil gruppen jobbe testdrevet
    - Det vil før enhver implementering skrives unit-tester som skal bestås før koden godkjennes
- Kode skal følge kvalitetsmål som DRY (Don’t Repeat Yourself) samt andre “clean code”-prinsipper som KISS (Keep It Simple Stupid) hvor unødvendig kompleksitet skal unngås…
- Før kode blir pushet til main skal det ha vært godkjent av to andre medlemmer på gruppen.

Samhandling

- Det vil holdes jevn dialog med produkteier for å kartlegge og sikre andre kvalitative aspekter slik som brukervennlighet, funksjonalitet og lignende.
- Det vil også holdes jevn dialog med tildelte veiledere fra bedriften og universitetet. Det vil opprettes kommunikasjonskanaler som vil sikre denne samhandlingen om restriksjoner hindrer oss fra å møtes.
- Gruppen vil til enhver tid strebe etter å holde god og åpen kommunikasjon innad i gruppen. Terskelen for å spørre om hjelp og gi hverandre tilbakemeldinger vil holdes lav for å sikre best mulig samarbeid.
- For å sikre best mulig samhandling rundt prosjektet vil vi aktivt ta i bruk valgte scrum-elementer (vil bli spesifisert senere i prosjektet) og andre verktøy som Devops og trello.

Rapport

- Plagiat-unngåelse er basert på tillit til at alle gruppemedlemmene følger UiA sine retningslinjer for kildehenvisning og skriver med sine egne ord. Hvis bevisst plagiat blir oppdaget vil tiltak beskrevet under "reprimander" bli utført.

Reprimander

- Dersom tildelte oppgaver ikke blir ferdigstilt til forventet tidspunkt, og det ikke gis god forklaring og begrunnelse på dette, kan det gis opptil 3 advarsler i løpet av semesteret.
    - Det er forventet at uforutsette problemer og utfordringer kommuniseres til gruppen ved første anledning etter man blir oppmerksom på de eller de oppstår
- Gruppen tillater 5 minutter forsentkomming til møter. Skulle forsentkomming over 5 minutter gjenta seg vil det bli tatt opp av gruppen, og passende reprimande vil bli presentert for vedkommende. Skulle dette ikke hjelpe vil det også her kunne bli gitt ut 3 advarsler iløpet av semesteret.
- I ytterste konsekvens vil et medlem kunne bli fjernet fra gruppen, dersom det er enstemmig enighet for dette i gruppen. Dette vil kun være aktuelt dersom medlemmet allerede har fått 3 advarsler og andre tiltak ikke har vært effektive.
- Hvis et gruppemedlem bevisst begår plagiat vil det bli gitt ut 1 advarsel. Hvis dette ikke er effektivt kan gruppemedlemmet risikere å bli fjernet fra gruppen, så lenge dette er enstemmig.

Scrum

Sprints

- Hver sprint skal vare i 3 uker hvor det gjennomføres en demo på slutten av hver sprint hvor vi presenterer ferdig funksjonalitet for veileder i Knowit.

Roller

- Scrum Master: Jørgen Lindbøl
- Git Master: Markus Sveggen

Tidsestimering

- Det vil avholdes en såkalt "planning poker" for å estimere arbeidsmengde og beregne tidsbruk for hver oppgave i backloggen
    - Dersom backloggen revideres underveis vil det avholdes nye estimerings-møter for å beregne tidsbruk
- I begynnelsen av hver sprint avholdes det et avklaringsmøte der vi går igjennom tidligere beregnet tidsestimater og oppdaterer om nødvendig
- Tidsestimater diskuteres på hver stand-up etterhvert som vi arbeider på oppgavene, slik at man får et mer realistisk estimat over tid.

Arbeidsfordeling

- I begynnelsen av hver sprint vil det avholdes et møte for å velge oppgaver fra backloggen som fordeles til hver Sprint og derifra

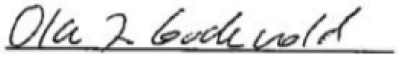
Plattformer

- **Microsoft Teams**
  - Offisielle møter
  - Kommunikasjon med veileder i Knowit
- **Google Docs**
  - Prosjektdokumenter
- **Azure DevOps**
  - Scrum Board
- **Azure Git**
  - Verktøy for versjonskontroll
- **Discord og Messenger**
  - For kommunikasjon/møter innad i gruppen
- **Overleaf**
  - Rapportskriving

Dato og sted: Kristiansand, 15.01.2021

Navn og signatur

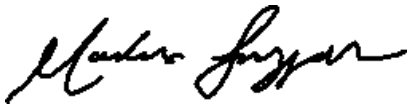
Ola Johansen Gudevold



Kirsti Nesse

Magnus Neergaard

Markus Sveggen



Jørgen Lindbøl

Stian Eikvang Mørk Barbakken

C Git Guide

Transform IT - Github Guide

Markus Sveggen

Sist oppdatert: Mai 2021

Innhold

1 Hvordan jobbe med Git

1.1 Første gang man skal jobbe på et repository

- Det første som må gjøres for å jobbe med git er å laste ned den nyeste versjonen av git her.
- Om man ikke har repositoriet lokalt må man hente dette ned. Sørg da for at man har navigert seg til den mappen man ønsker ha repositoriet i og kjør følgende kommando:

```
git clone [link til repo] [lokalt navn til repo]
```

- For å bruke Microsoft kontoen med versjonskontrollverktøyet Git må man gjøre noen justeringer. Først må man kjøre Git credential manager som anvist her.
- Videre må man sette opp en git bruker for hvert repository vi jobber med, man må altså gjøre dette for både backend og frontend repoet. Dette gjøres ved å navigere til repositoriet man har klonet ned, og kjøre følgende kommandoer byttet ut med ditt navn og din uia-epost. Det er viktig at man har med student.uia i UiA-mailen, ellers vil det ikke funke.

```
git config user.name "Fornavn Etternavn"
git config user.email "din_uia_mail@student.uia.no"
```

1.2 Opprette en ny branch

Vi har valgt et format på branching hvor hver brukerhistorie har sin egen branch som er branchet utifra main. Videre skal alle taskene som tilhører brukerhistorien branches utifra brukerhistorie-branchen. Når man er ferdig med alt arbeidet i en task og alt av kode er testet, oppretter man en pull request for å merge innholdet i task-branchen med brukerhistorie-branchen. Når alle taskene i en brukerhistore er ferdig og testet så merger man brukerhistorie-branchen med main. Vi seperarer også mellom frontend og backend som er i hvert sitt repository.

For å opprette en ny branch må man gjøre følgende:

- Opprett en branch utifra en brukerhistorie i Sprint-boardet i DevOps. Dette gjør man ved å trykke på de tre dottene oppe til høyre på en brukerhis-

torie. OBS: Dette trenger man ikke gjøre om branchen allerede eksisterer.

- Navngi den nye branchen etter følgende format: U[nr-på-user-story]-[FE/BE]-[navn-på-brukerhistorie]
  Eksempel: U140-FE-Bruker-logginn
- Opprett en ny branch for oppgaven man skal arbeide på, fra den brukerhistorie-branchen den tilhører. Dette gjøres ved å trykke på de tre dottene på en task og velg New branch, og opprett en branch utifra brukerhistorien den tilhører.
- Bruk følgende format for oppretting av branch navn til å være:
  T[nr-på-task]-[FE/BE]-[passende-navn-til-task-branch]
  Eksempel: T148-BE-Endringshistorikk

1.3 Hente ned endringer fra repositoriet

- Hent ned alle remote endringer med å kjøre følgende kommandoer:

```
git checkout main
git pull origin main
```

- Endre til din task/brukerhistorie-branch med følgende kommando:

```
git checkout [branch-navn]
```

- Hent deretter ned alle endringer fra remote repo:

```
git pull origin [branch navn]
```

1.4 Når man er ferdig med arbeidet

- Når man ønsker å lagre endringer man har gjort i git, bruker man følgende kommando for å legge til ALLE endringer man har gjort.

```
git add .
```

- Når man har lagt til flere endringer eller ønsker å legge til disse endringene på den remote branchen må man først commite endringene:

```
git commit -m"[commit melding]"
```

  Commit meldingen skal si noe om hva som har blitt gjort. F.eks. La til tester for glemt passord funksjonen.

- **OPTIONAL:** Om man har lagt til flere commits som man ønsker å squashe sammen. F.eks. om man har mange nesten like commits kan man gjøre dette med rebase kommandoen. Mer info om dette kan også finnes her.

```
git rebase -l HEAD~[antall commits man vil hoppe tilbake]
```

Dette kan også gjøres på en enklere måte ved å skrive inn kommandoen under. Da må man også spefisere ny commit-message i kommandoen:

```
git reset --soft HEAD~[antall commits man vil hoppe tilbake]
   && git commit -m"[ny commit message]"
```

- Push endringer opp til remote branch

```
git push origin [branch-navn]
```

1.5 Merge task-branch med brukerhistorie-branch

- Når alt av tester kjører og er godkjent kan man merge branchen sin med den tilhørende brukerhistorie-branchen.
- Trykk på pull-requests knappen under Repository-fanen venstre sidemeny i DevOps og åpne en pull request fra den branchen man ønsker å merge mot en brukerhistorie-branch.
- Når man har åpnet en pull request må koden reviews av minimum et annet gruppemedlem før den kan bli godkjent og merget mot brukerhistorie-branch.

1.6 Merge brukerhistorie-branch mot main

Her er prosedyren nesten helt lik som for merging mot brukerhistorie-branch, se Underkapittel 1.5. Det som er forskjellig er som følger:

- Alle task-branches tilhørende brukerhistorie-branchen som skal merges, må være merget inn i brukerhistorie-branchen.
- I tillegg må man velge å merge brukerhistorie-branchen inn mot main-branchen.

D Risikomatrise

Nr.	Uønsket hendelse (risikoelement)	Årsak	Sårbarhet	Risikonivå			Tiltak for at det ikke skal skje	Tiltak redusere skade	Kommentar
				S	K	Risiko			
	Hva kan skje? Hva er den uønskede hendelsen? Hvilke tap oppstår?	**Hvorfor vil det kunne skje?** Hvem eller hva initerer hendelsen? For overlagte hendelser: Hvilken kapasitet og motiv har trusselaktøren?	**Hvilken svakhet/feil kan utnyttes her?**	**Sansynlighet og konsekvens på en skala fra 1 til 4.** 1-5 = Lav 6-10 = Moderat 11-13 = Over middels 14-16 = Høy			**Beskriv forslag til nye tiltak.** De kan deles opp i organisatoriske, menneskelige og teknologiske sikringstiltak.	**Beskrive forslag til tiltak som vil redusere skaden**	Erfaringer, vinkling til rapport
NR1	Dårlig eller svekket kommunikasjon og samarbeid innad i gruppen	Hvis vi må jobbe mye digitalt kan avstand fra hverandre påvikre kommunikasjonen, tekniske problemer kan påvikre. f.eks. kan uenigheter i gruppen, missforståelser eller andre privte hendelser som gjør at kommunikasjon svekkes.	Det vil være en sårbarhet ved at samarbeid i forhold til gruppearbeid blir svekket. Kan føre til at oppgaver ikke blir løst eller at de ikke blir løst på en riktig måte. Kan også påvirke samholdet i gruppen.	3	4	12	Ta i bruk tiltak for å opprettholde kommunikasjon når vi må jobbe digitalt (se eget punkt). Sette av tidspunkter for å kunne diskutere og ta opp nødvendige temaer. Ha på forhånd klar hva som skal tas opp og diskuteres så alle har mulighet til å forberede seg.	Avklar tidlig i prosessen at det skal være et åpent miljø hvor folk kan ta opp det de vil, og at alle skal bli hørt. Eksterne tiltak kan være å involvere veileder på UIA.	Skulle dette oppstå er dette noe som må settes av tid til. Dette er en essensiell funksjon som må være på plass for å gjennomføre prosjektet på best mulig måte. Settt heller av utviklingstid til dette og refleker over det i rapporten.
NR2	Dårlig eller svekket kommunikasjon eller samarbeid med bedriften	Digitalt arbeid setter høyere terskel for å kontakte bedriften. Bedriften har liten interesse av å følge opp prosessen. Bedriften har for liten tid til å kunne følge opp prosessen. Vi som gruppe tar av setter terskelen for høy til å kontakte bedriften.	Vi sitter igjen med for lite informasjon til å kunne gjennomføre prosjektet på best mulig måte. Vi tar beslutninger vi egentlig ikke er sikre på. Arbeid stopper opp fordi vi mangler informasjon	3	4	12	Skape klare mål og en klar plan på starten av prosjektet. Vær tidlig ute med å etablere god kontakt med bedriften. Vær tydelig om forventinger, og la dem være det samme tilbake. Sett opp gode kommunikasjonskanaler hvis alt blir digitalt.	Holde lavterskel og faglig kommunikasjon med Knowit jevnlig. Inviter til møte, sett opp agenda så alle er forberedet på hva som skal bli tatt opp på møte. Hold god kontakt med UIA veilder og spør om råd.	Hvis bedriften er tidlig ut med å si at deres rolle vil være veldig liten fra begynnelsen av, så er dette noe vi må jobbe rundt og reflektere over i rapporten.
NR3	Arbeidsoppgaver tar mye lengre tid enn forventet	Høyere vanskelighetsgrad på oppgaven enn først tenkt. Gruppemedlemmer jobber ikke når de skal. For dårlig vurdering av oppgaven, og dermed dårlig estimering.	Sårbarhet her blir at vi ikke rekker å bli ferdig med de oppgavene vi ønsker, og som Knowit forventer som produkteiere.	4	3	12	Oppgaver må diskuteres grundig for å best mulig kunne scope og tidsestimere. Jo mer vi diskuterer jo mer avdekker vi av ting vi må ta hensyn til, og på den måten kan vi estimere bedre. Ta bruk tiltak som gjør at vi opprettholder forventet abeid, som f.eks. logging av timer, daily standups osv.	Lær av tidligere estimeringer, analyser tidsbruk iforhold til oppgavene, og reestimer der du må.	Det er selvfølgelig ikke forventet at vi må bli ferdig med alle oppgavene som både veileder og knowit har presisert. Viktigste e at vi har god kontroll på srbeidsprosess og hvilke beslutninger vi tar som har ført til resultatet.
NR4	Sosial loffing	Det er et kjent konsept at gruppemedlemmer yter mindre i grupper enn det de ville gjort individualt. sjangsen for sosial loffing blir større, jo større gruppen blir.	Med tanke på at vi er max antall på vår bachelor gruppe, altså 6 personer, vil også sjangsen øke for at det oppstår sosial loffing.	2	4	8	Ved å tidlig i prosessen sette krav og forventinger. Samt sette sammen en klar og definitiv gruppekontrakt. Ta i bruk tiltak som opprettholder forventet arbeid, eks. logging av timer og daily standups.	God og klar kommunikasjon innad i gruppa. Ta i bruk gruppekontrakt når det skjer hendelser. Konsekvensene skal være like for alle.	Vi har den fordelen med at vi har jobbet som en gruppe siden første året (med Ola som unntak). Vi har til nå ikke opplevd dette fenomenet, men det er viktig at vi er klar over det og tar i bruk tiltak for å motvirke det.
NR5	Total lockdown over en lengre periode	Coronatiltak kan føre til lockdown hvor universitetet stenger og det ikke er lov til å møtes fysisk i en gitt gruppestørrelse	Vi har lite erfaring med å bare jobbe digitalt. Produktivitet og motivasjon kan synke når man ikke møtes fysisk. Det at man bare sitter inne kan påvirke mental helse.	3	2	6	Sette opp gode kommunikasjonskanaler, faste møter, gode tidsloggingssytemer, daily standup, hyppig rapportering av arbeid gjennomført.	Være flinke til å inkludere alle og passe på at alle har det bra.	Med mulig lockdown så kommer omstendigehetene til å endre seg en del, det blir viktig forr oss å prøve å holde tiltakene, men også reflektere over hvordn disse endringene har påvirket oss og om vi har måtte endre/legge til flere tiltak.
NR6	Nødvendige ressurser som vi tar i bruk og trenger for prosjektet blir utilgjengelige	Vi bruker flere forskjellige verktøy og produkter for å få gjennomført prosjektet. Verktøy og produkter kan gå ut av produksjon, eller bli til betalte løsninger som ikke vi har tilgang til. Det kan komme nye oppdaterte versjoner som gjør at vi ikke lengere får kjørt prosjektet. Budsjett setter begreninger for hva vi kan ta i bruk	Vi må finne nye løsninger, eventuelt flytte vår løsning til en annen versjon. Det vil være tidkrevende og kan i mange tilfeller være veldig komplisert. Det kan gjøre at prosjektarbeid stopper opp, eller at arbeid vi allerede har gjort bar må legges bort.	2	4	8	Velge verktøy som det er liten sannsynlighet vil bli endret på, eller hindre oss i arbeid hvis det blir endret på. Finne tidlig ut av budsjetter som er satt hos knowit (hvis det er relevant). Høre med knowit hvilke anbefalinger de har.	Se etter løsninger som ligner løsning vi har. Velge verktøy og programmer som kan gjøre det lett for oss å bytte om vi må.	Det trenger ikkenødvendigvis være at et prgram blir utilgjengelig, men det kan også være at vi finner ut av at det ikke fungerer som vi ville, eller at vi senere har oppdaget noe bedre. Det må uansett da vurderes og reflekteres osver i rapport.
NR7	En risiko som kan oppstå er at det blir skjev arbeidsfordeling	Med lite erfaring innen scoping og estimering kan det hende at noen sitter igjen med mye mer arbeid en andre. En annen årsak kan være at noen på gruppen ikke engasjerer seg nok, mens andre kanskje engasjerer seg for mye.	Gruppemedlemmer føler at de må gjøre alt av arbeidet og at de ikke får noe hjelp. Kan bli utarbeidet. Andre kan føle at de ikke kommer til og ikke får mulighet til å prøve seg før oppgaver allerede er tatt.	2	3	6	Revurder hele tiden etdsestimeringen, og analyser timer brukt. Gå grundig igjennom oppgaver som blir delt ut.	Ha åpen kommunikasjon innad i gruppen og spør om folk føler de har for mye eller for lite å gjøre.	Folk er forskjellige, og kan ha forskjellige meninger rundt oppgaver. Vi må ha åpent sin og legge til rette for endringer der det trengs.
NR8	Et eller flere gruppemedlemmer kan ha harware eller software problemer som hindrer arbeid	Man vet aldri tid det kan oppstå skader på både hardware, software. Internett er ikke heller alltid like stabilt så det kan fungere som en risiko.	Med tanke på at vårt prosjekt er et utvilingsprosjekt er vi avhengige av at både hardware og software skal være på plass. Hvis det blir til at vi må jobbe digitalt kan det være svært sårbart om noen av delene ikke skulle funke, samt om internett ikke er oppe. Det vil hindre gruppearbeid, utvikling og fremgang i prosjektet	2	4	8	Før oppstart burde vi sjekke og sikre så godt vi kan at alt det nødvendige vi trenger for å gjennomføre prosjektet er på plass.	Hvis det skulle oppstå problemer må det etter omstendighetene bli fikset så fort som mulig, Om det skulle vise seg og ikke være mulig må vi få ordnet en kommunikasjonskanal slik at vedkommne kan være med på gruppearbeid til problemet er løst.	Skulle dette oppstå må vi bare skrive om det i rapporten og reflektere over måten vi valgte å løe det på.
NR9	Vi holder oss ikke innenfor reglene når det kommer til behandling av data (GDPR) eller andre rettningslinjer og regler.	Vi jobber med data, og programmet vi utvikler skal kunne håndtere dat i herhold til GDPR. Hvis vi ikke tar hensyn til dette risikrer vi at data kan misbrukes, og det må betales bøter	Med tanke på at det er lovpålagt vil det være svært dumt om vi skulle overtre noen regler. Det kan bli dyrt, tidkrevende og ressurskrevende. Sårbart for dem dataen omhandler	2	4	8	Være klar over reglene som eksiterer. Prate med knowit og semine om hvilken data som kan være sensitv og om de har råd til hvordan vi skal forholde oss til det. Vurdere om vi vil ha det med, eller om det er noe som kan bli utelatt.	Dette skal bare ikke skje.	Det kan også hende at vi ikke står overfor denne problemstillingen. Men om vi gjør det må vi sette av god tid til å få nok kunnskap om hva som er riktig.
NR10	Vi mister kode som er essensiell	Software eller hardwareproblemer som gjør at vi mister kode. Andre åsaker kan være at man både bevisst og ubevisst sletter kode som det viser seg er viktig.	Det kan sette oss mye tilbake i arbeid hvis vi mister essensiell kode. Det er både ressurskrevende og tidkrevende å få rettet opp i. Om man ikke har backusp kan det potensielt være mye arbeid som går tapt.	2	3	6	Ha oppdatert software og hardware. Sett retningslinjer for hvordan å håndtere kode. Pushe og merge ofte slik at flere har tilgang på koden. Få det tydelig frem i gruppekontrakt at man ikke bevisst og forsåvidt onsindet skal slette kode.	Passe på at man har lagret backups på flere maskiner om det er mulig.	Om alle er oppdatert og tar i bruk github aktivt burde dette ikke bli et stor problem. Vi kan reflektere over hvordan vi har kvalitetikting gjennom versjonskontroll.
NR11	Koden/Testene ikke vil kjøre og vi finner ikke ut årsaken	Årsakene kan skyldes synstaks errorer, runtime error, software, IDE bugs og andre uforutsette hindringer.	Ressurskrevende og tidkrevende. Manglende erfaring kan gjøre at vi står fast på et problem lenge.	2	4	8	Ha god og åpen kommunikasjon med veildere både på knowit og ved UIA. Ha lav terskel for å spør om hjelp der man kan finne det.	Innse at den eneste man kanskje trenger er noen med mer erfaring som har vært borti problemet før.	Vi er heldige med at vi har knowit som bedrift som har et gjennomsnitt på 25 år med erfaring
NR12	Veiledere James og Tor Oscar blir utilgjengelige over en lengre perode	Kan skyldes private hendelser, sykdom, andre jobb hendelser, karantene osv.	Vi sitter igjen med lite informasjon, eller at vi ikke får svar på ting vi lurer på. Kan førte til at vi må ta beslutninger på grunnlag av feil inormasjon eller ingen informasjon i det hele tatt.	2	3	6	Ha god kommunikasjon med veildere, planlegge i god tid om det er mulig å planlegge. Vanskelig å skulle forutse. Lurt å kunne ha flere å henvende seg til.	Om det ikke er planlagt kan man forhøre seg om det er noen andre som kunne ha vært veileder midlertidig. Lurt å henvende seg til veileder på UIA for rådgivning.	Key her er å ha god og åpen kommunikasjon.
NR13	Veileder ved UIA Hallgeir blir utilgjengelig over en lengre periode	Kan skyldes private hendelser, sykdom, andre jobb hendelser, karantene osv.	Vi sitter igjen med lite informasjon, eller at vi ikke får svar på ting vi lurer på. Kan førte til at vi må ta beslutninger på grunnlag av feil inormasjon eller ingen informasjon i det hele tatt.	2	2	4	Ha god kommunikasjon med veildere, planlegge i god tid om det er mulig å planlegge. Vanskelig å skulle forutse. Lurt å kunne ha flere å henvende seg til.	Om det ikke er planlagt kan man forhøre seg om det er noen andre som kunne ha vært veileder midlertidig.	Key her er å ha god og åpen kommunikasjon
NR14	En eller flere gruppemedlemmer blir utilgjengig og ikke i stand til å jobbe med prosjektet over en merkbar periode	Kan skyldes private hendelser, sykdom, andre jobb hendelser, karantene osv.	Gruppemedlemmet risikerer å gå glipp av avgjørelser og eventuelle andre hendlser som kan være viktig for prosjektet. Det vil være mindre arbeidkraft, og mer arbeid på de andre i gruppen.	1	4	4	Ha flere muligheter for møter og samarbeid (både fysisk og digitalt). Generelt ha god og åpen kommunikasjon. Planlegge i god tid om det er mulig å planlegge. vanskelig å forutse. Del all informasjon så ikke en på gruppen sitter alene med viktig informasjon som gruppen trenger.	Alle er bevisst på å holde seg friske, respektere hverandre. Sette opp referater fra det som blir gått igjennom av gruppen slik at vedkommne kan holde seg oppdatert med hvilke beslutninger som er blitt tatt.	Vi må ta hensyn til hverandre med tanke på pandemien. Hvis det er noe som skulle oppstå kan dette tas opp så vil vi finn en løsning.
NR15	Scrum-master blir utilgjengig og ikke i stand til å jobbe med prosjektet over en merkbar periode	Kan skyldes private hendelser, sykdom, andre jobb hendelser, karantene osv.	Gruppen vil mangle en viktig rolle for prosjektet. Kan føre til at prosjektet stopper opp, beslutninger vil ikke bli tatt, mister struktur og oversikt. Vil også være mindre arbeidskraft for resten av gruppen.	1	4	4	Gjennomfør prosjektet på en måte hvor alle har kontroll, og vet dette scenarioet så kunne hvem som helst steppet inn som scrum vara mens denne personen var borte. Del all informasjon så ikke en på gruppen sitter alene med viktig informasjon som gruppen trenger.	God kommunikasjon er vitktig. Ikke la all struktur og organiserin henge på en person, fordel på resten av gruppen og deleger roller der det er nødvendig.	Viktig at når vi deler inn i roller så er vi klar over at dette ikke betyr at kun denne personen skal ha kontroll på denne tingen. Reflkleter over hvordan vi har løst dette i rapporten.
NR16	Problemer med konflikter i Git	Det at vi er relativt nye på git, og ikke har tatt det så mye i bruk utgjør derfor en risk for oss. Uvitenhet og lite erfaring. Dårlig planlegging og kommunikasjon innad i gruppen.	Hvis det ikke blir gjennomført tiltak kan det hende at konflikter i git kan stanse oss i samarbeide på et prosjekt. Det vil gå mye tid til å rette opp eventuelle konflikter.	3	1	3	Utarbeide tydelig guide for hvordan vi jobber med git. Ha felles gjenomganger av hvordan ting skal gjøres. Sett standaren lav iforhold til å spørre om hjelp før man gjør noe.	Flere som må godkjenne pull og merge. Ha en dedikert Git master.	Git guiden er utarbeidet på en måte som gjør at folk som er helt nye med git burde klare å gjennomføre det som står i guiden.

NR17	En risiko kan være at de målene vi har satt oss er for store	På grunn av lite erfaring med både scoping og gjennomføring av prosjekter kan det være en risiko at vi setter for store mål.	Det kan både være en stressfaktor for gruppen samtidig som at det kan påvirke kvaliteten på arbeidet vi gjør. Istedenfor å ta oss tid til å gjennomføre noe ordentlig, så stresser vi oss gjennom for å gjennomføre målene.	2	2	4	Vi må være klar over scopecreep når vi sitter å diskuterer oppgaver. Merker vi at vi har gått for langt med en oppgave må vi ta et steg tilbake og sammenlikne med med hva kravet fra produkt eier var.	Vi kan få tips fra knowit om hvordan å best mulig scope, samt få avklart tidlig hva det er de ønsker og hva som ikke er like viktig.	Av tidligere erfaring så vet vi at vi er glad i å se muligheter ved et prosjekt, men vi har tidligere vist at vi klarer å holde oss innenfor scope. Vi må huske å reflektere over dette i rapport og få ned viktige punkter med tiltak vi har gjort.
NR18	Vi får ikke noe utbytte av forelesningene som omhandler bachelorprosjektet	Foreleser har ikke god nok kvalitet på forelesningene. Vi lærer ikke det vi egentlig skal lære.	Grunnlaget vårt for når vi skl begynne på bacheloren er for dårlig. Vi vet ikke hvor vi skal begynne eller hva som er viktig å fokusere på i et slitk projekt.	1	2	2	Møte opp i alle forelesninger, ta notater så vi har noe å se tilbake på hvis vi er usikkre på noe. Ta i bruk Veilder på UIA for å videre utdyepe eller få forklaring på noe.	Vitkig at vi diskuterer sammen i gruppa om det er noe vi er i tvil på. Ta opp med foreleser hvis det blir et problem at vi ikke får noe ut av forelesningene.	Er Hallgeir som har en del av forelesningene, burde gå fint. Hvis det skulle bli et problem er ihvertfall terskelen lav for å si ifra.
NR19	En risk er at gruppekontrakten brytes men det blir ingen konsekvenser	Gruppemedlemmene tar ikke kontrakten serriøst. Kan være gruppemedlemmer som bryter kontrakten både bevisst og ubevisst. Gruppeleder og/eller gruppemedlemmer syns det er ukomfortabelt å ta opp når det skjer noe.	Reglene vi er blitt enige om ikke blir respektert, og problemet fortsetter å skje fordi det ikke er konsekvenser. Kan føre til at det oppstår hendelser som både er tidkrevende og ressurskrevende.	1	3	3	Når først gruppekontrakten utarbeides er det viktgi at alle medlemmene deltar og er enige i punktene som blir skrevet ned. Alle skal også være enige om reprimendene som blir hvis en bryter gruppekontrakten.	skulle det skje overtredelser er det fint å kunne gå igjennom kontrakten og få en oppfriskning på punktene og reprimendene. Skulle det fortsette burde det tas tak og om nødvendig invlovere UIA veileder.	Vi har fordelen med at vi har vært en gruppe lenge, så vi kjenner hverandre godt og terskelen er lav for å ta opp ting.
NR20	Koden vi skriver er uoversiktig og uforståelig for andre	Lite erfaring og lite foreredelser. Gruppen skriver kode på ulik måte.	Koden som skrives blir uforståelig for både andre på gruppa men også eksterne folk som har interesse i prosjektet. Kan være både ressurskrevende og tidkrevende å få rettet opp i.	2	2	4	Bli tidlig enig om en kodestandard som alle skal bruke. Ta i bruk standarer som allerede finnes på nett. Ha god kommunikasjon i gruppen og bli enige om ting sammen.	Spør om tips og råd fra de som jobber på knowit som har mere erfaring med dette.	Burde utarbeides en standard for både frontend og backend. Reflektere over i rapporten hvordan vi har valgt å løse det, og om det funket.
NR21	Databasen blir hacket	Enkel tilgang, og dårlig valg av brukernavn og passord. Brukernavn og passord blir bevisst eller ubevisst delt.	Scriptene som er lagret kan bli slettet. Verdier lagret i databasen kan bli missbrukt. Vil være tidkrevende og ressurskrevende om det må fikses, eller i værste fall gjøres på nytt.	2	2	4	Ta i bruk databaseløsninger som vi har lest har gode sikkerhetsløsninger. Ta i brukt sterke passord. Sett tydelige regler om at passord ikke skal deles til andre som ikke skal ha eller trenger tilgang.	Ha alltid backups av scrips slik at databasen lett kan sette opp igjen. Pass på å ikke lagre sensitiv data som kan bli missbrukt på et lett tilgjengelig sted.	Vi har tidligere erfart at databasen vår har blitt hacket, så disse tiltakene har blitt en selvfølge for oss.
NR 22	Vi må endre på lengden av sprintene	Hvis vi skulle stå ovenfor ulike problemer som gjør at vi ikke har noe nytt å vise på sprinten. Hvis produkteier eller gruppen ikke har mulighet på dette tidspunktet.	Vi må vente lengre på bekrefeltsen/tilbakemeldingene vi vanligvis får på en sprint review. Fremstår uproffesjonelt for produkteier og UIA veilder.	2	2	4	Planlegge godt og vurdere alle aspekter ved prosjektet og konsekvenser med å utsette en sprint. Vei opp fordeler og ulemper. Ha alltid oppdatert devops og trello board så vi har en god oversikt over opgpaver som skal gjøres og estimert tid til dette. Ved å ha dette på kontroll kan vi argumentere for og imot og skulle utsette eller ikke.	Ha god kommunikasjon med både veileder på UIA og produkteier. Få bekreftelse fra begge parter og søg for å holde gjevn og god arbeidsflyt slik at ikke ting bare blir utsatt.	Igjen, om dette skulle oppstå må vi bare dokumentere alle argumenter for og imot og begrunne vårt valg. Dette vil uansett ikke bli feil og vi klarer å begrunne valget vårt godt.

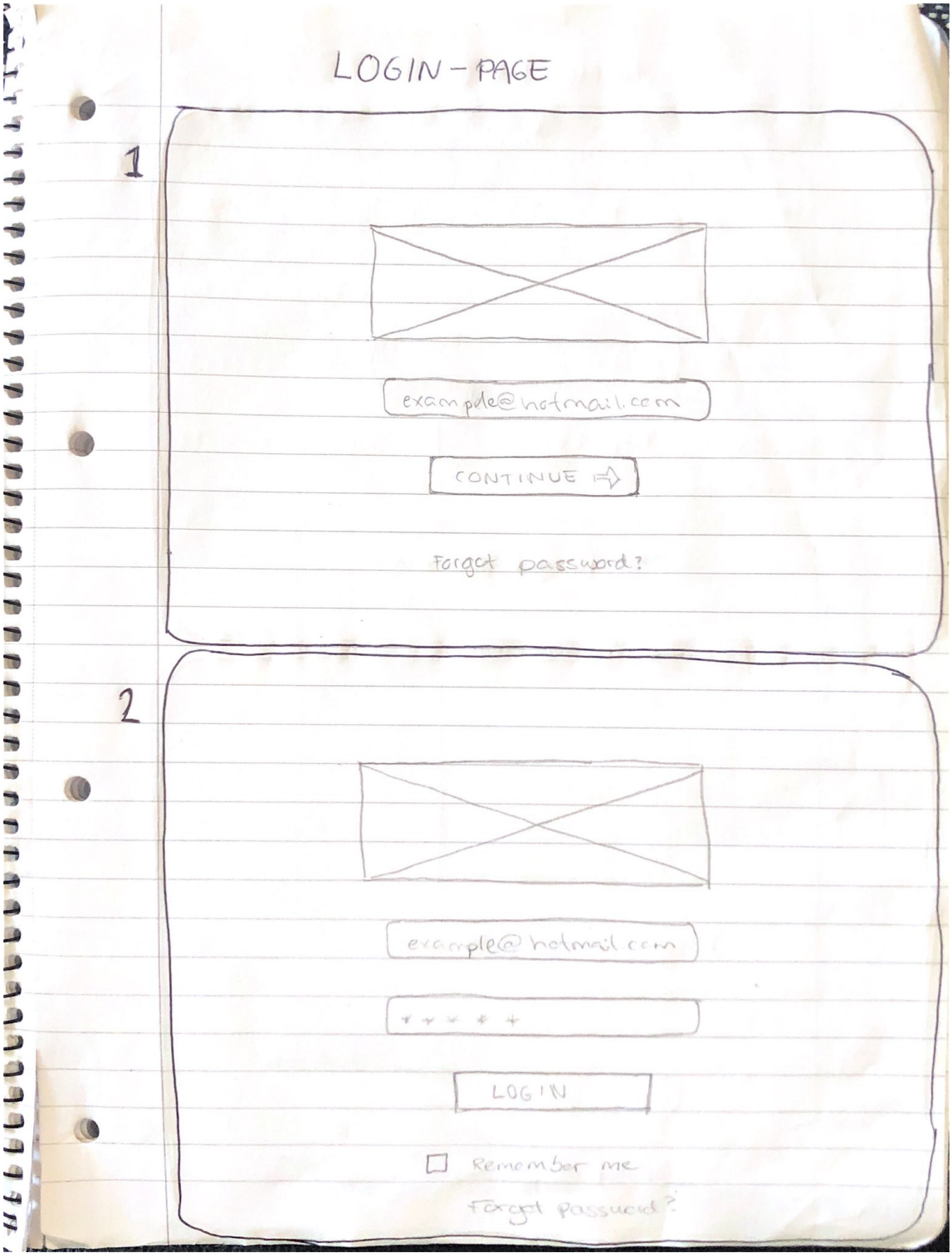
Kilder

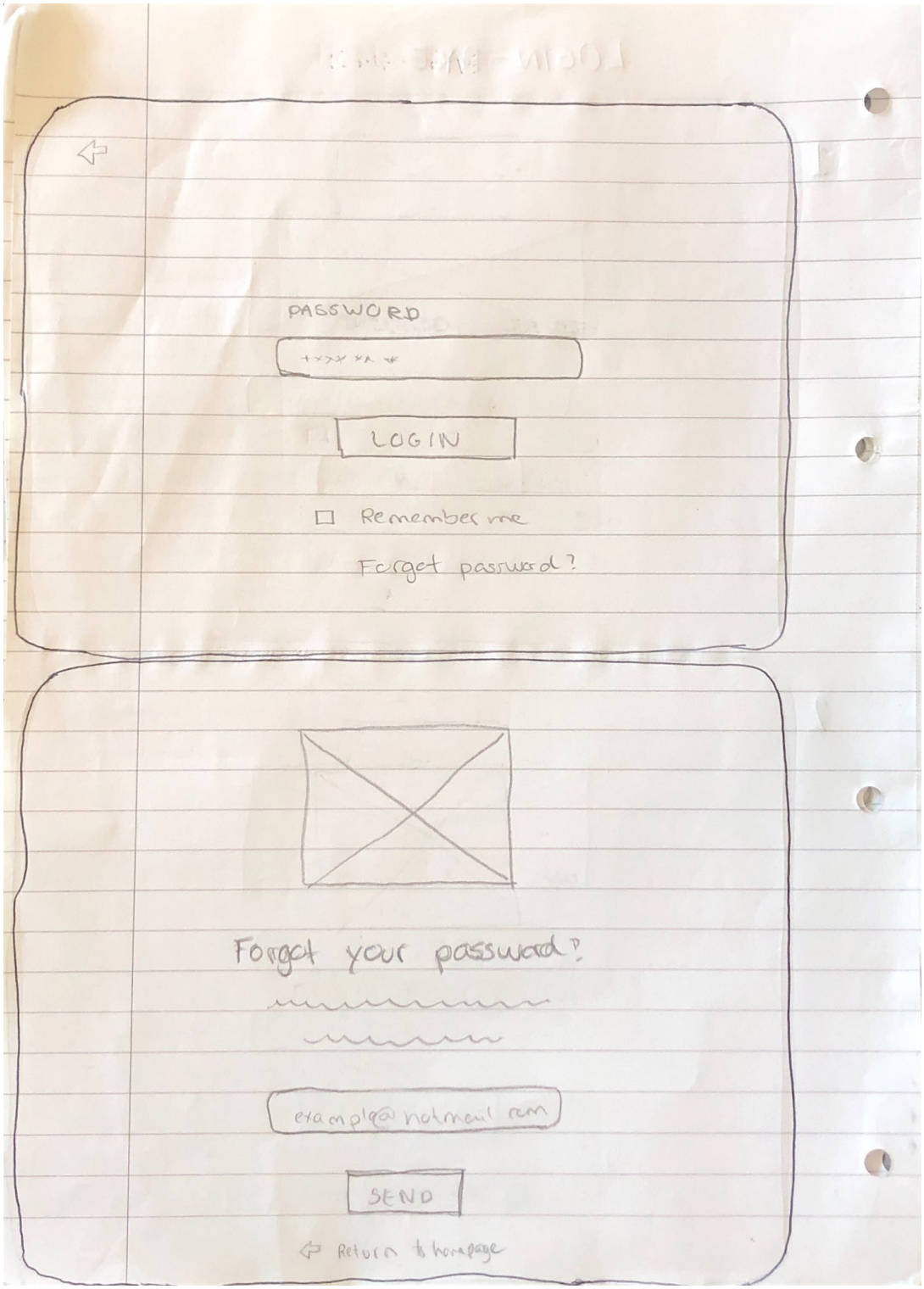
Mal for ROS analyse - Bærum kommune

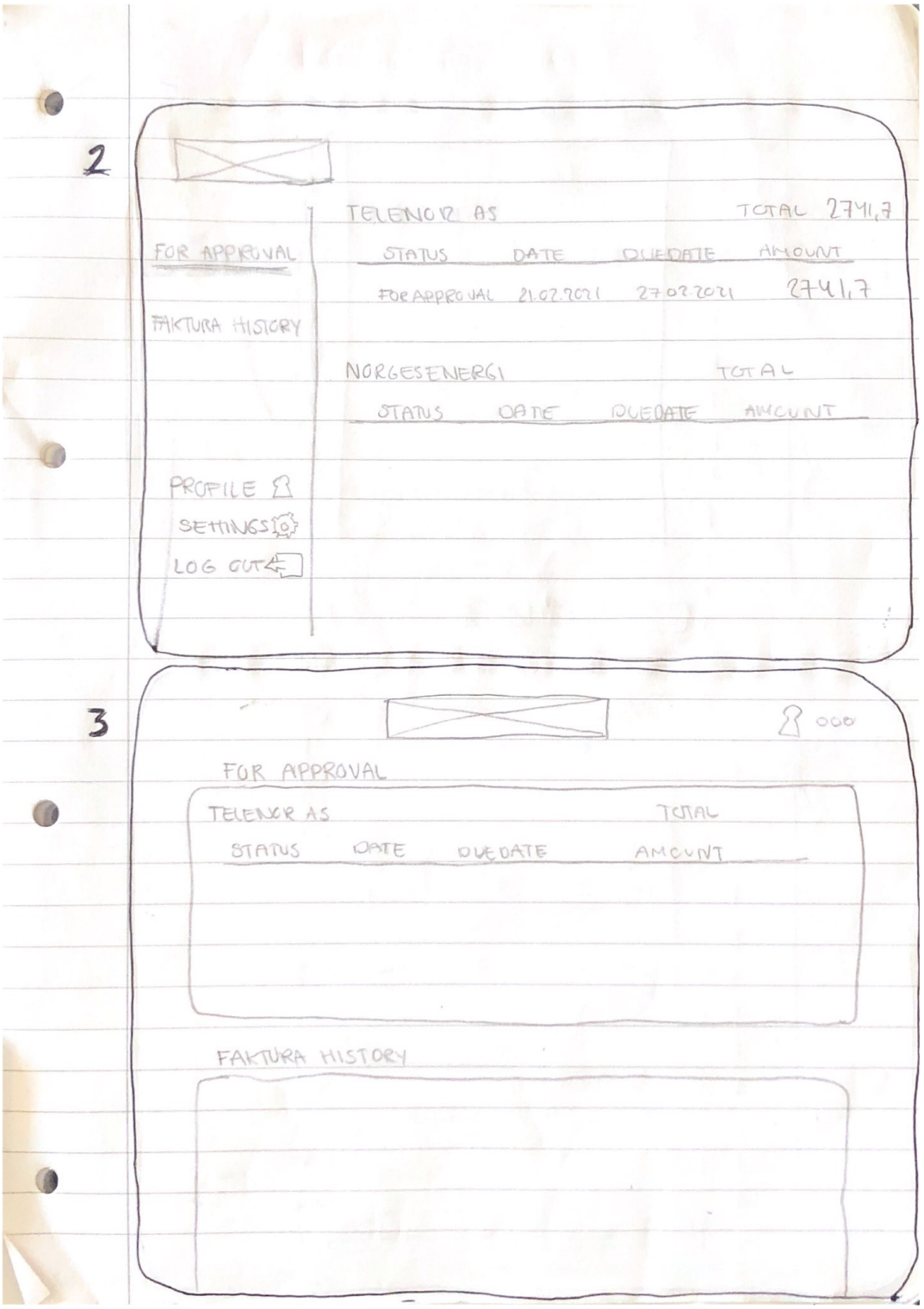
Powerpoint om risikoanlayse - Hallgeir Canvas

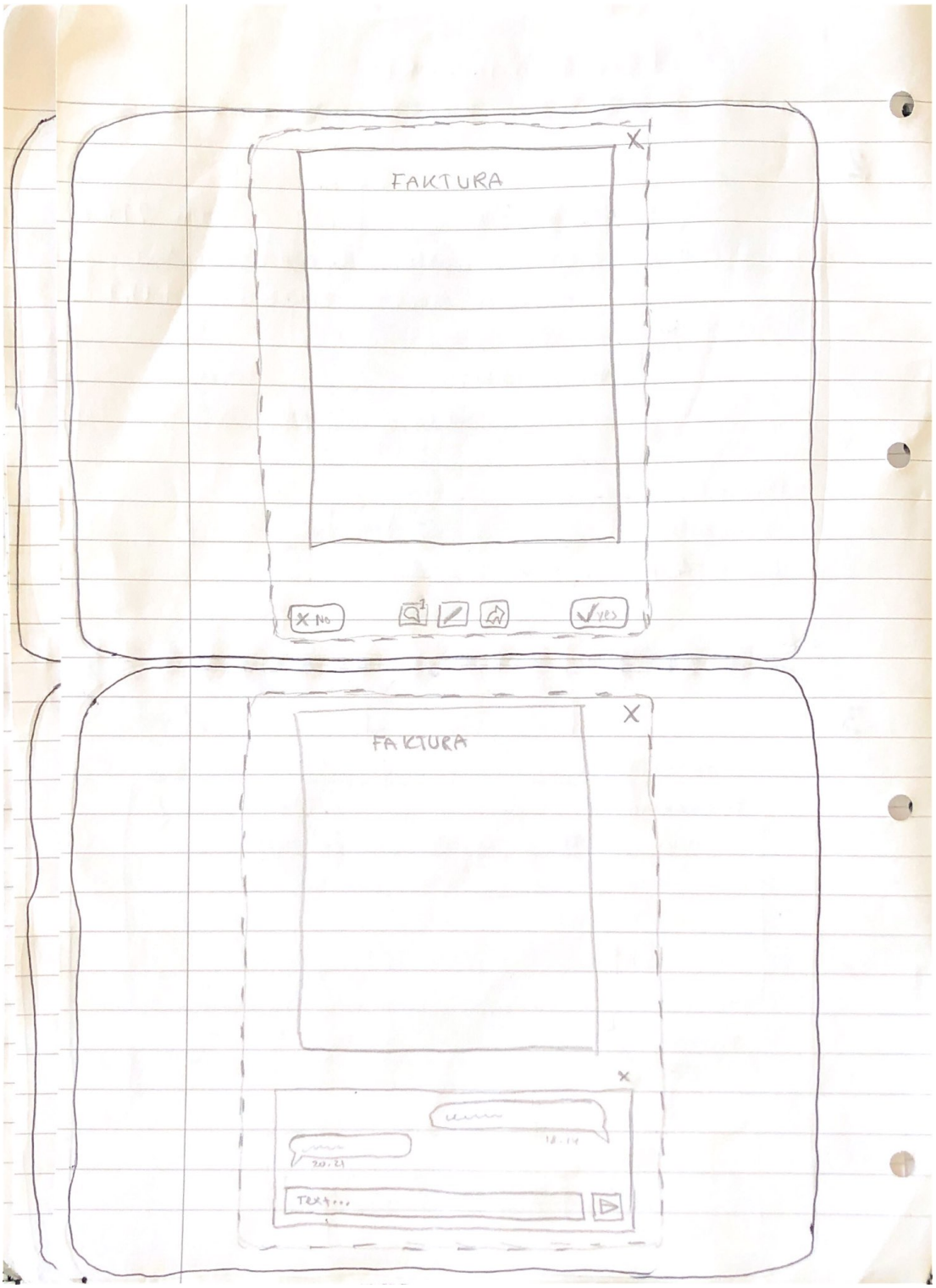
Mal hentet fra unit.no direktorar for IKT og fellestjenester i høyere utdanning og forsking.
her er lenken til filen: https://www.unit.no/sites/default/files/media/filer/2019/02/Mal-for-risikovurderinger-v7.xlsx

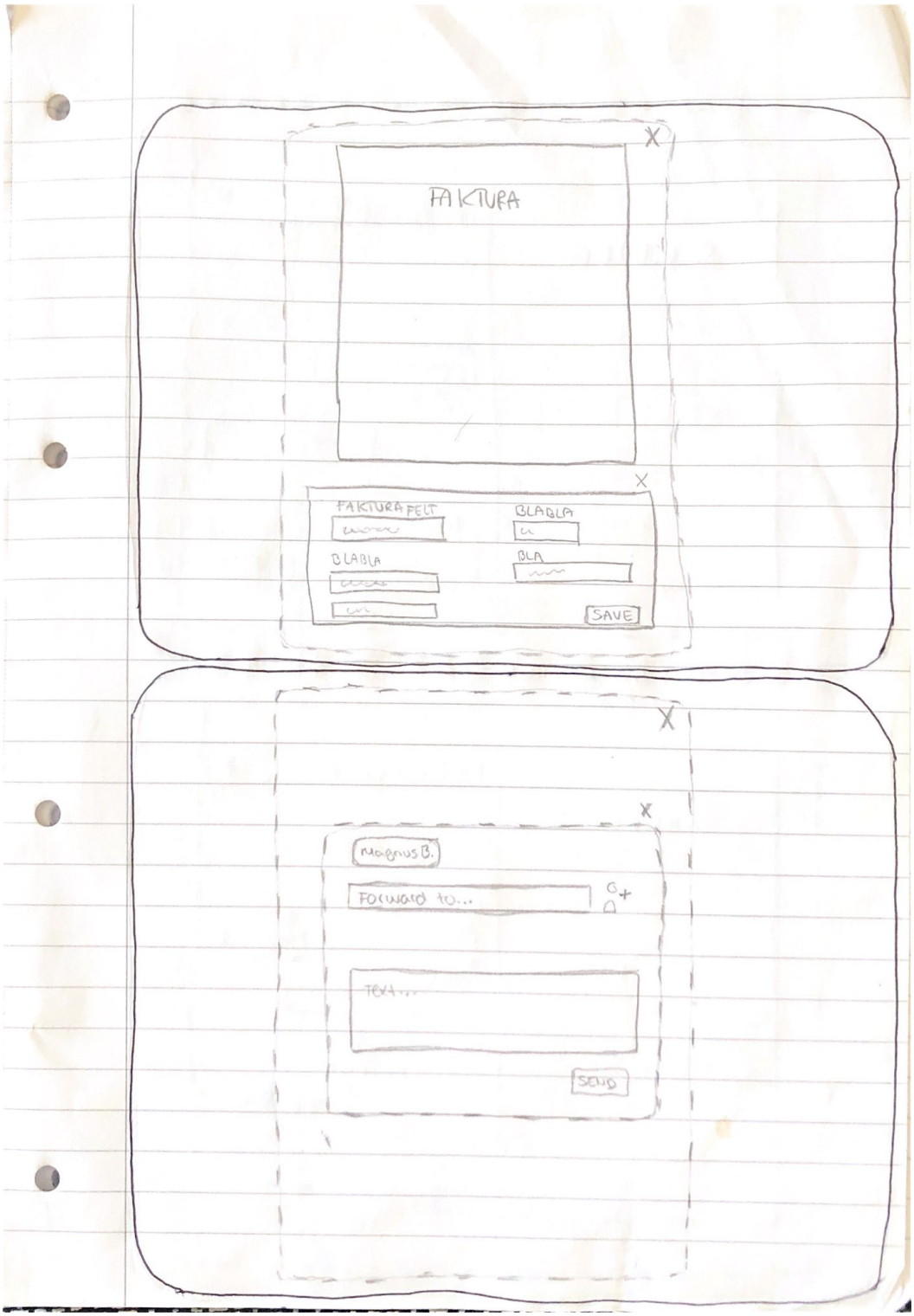
E Wireframes

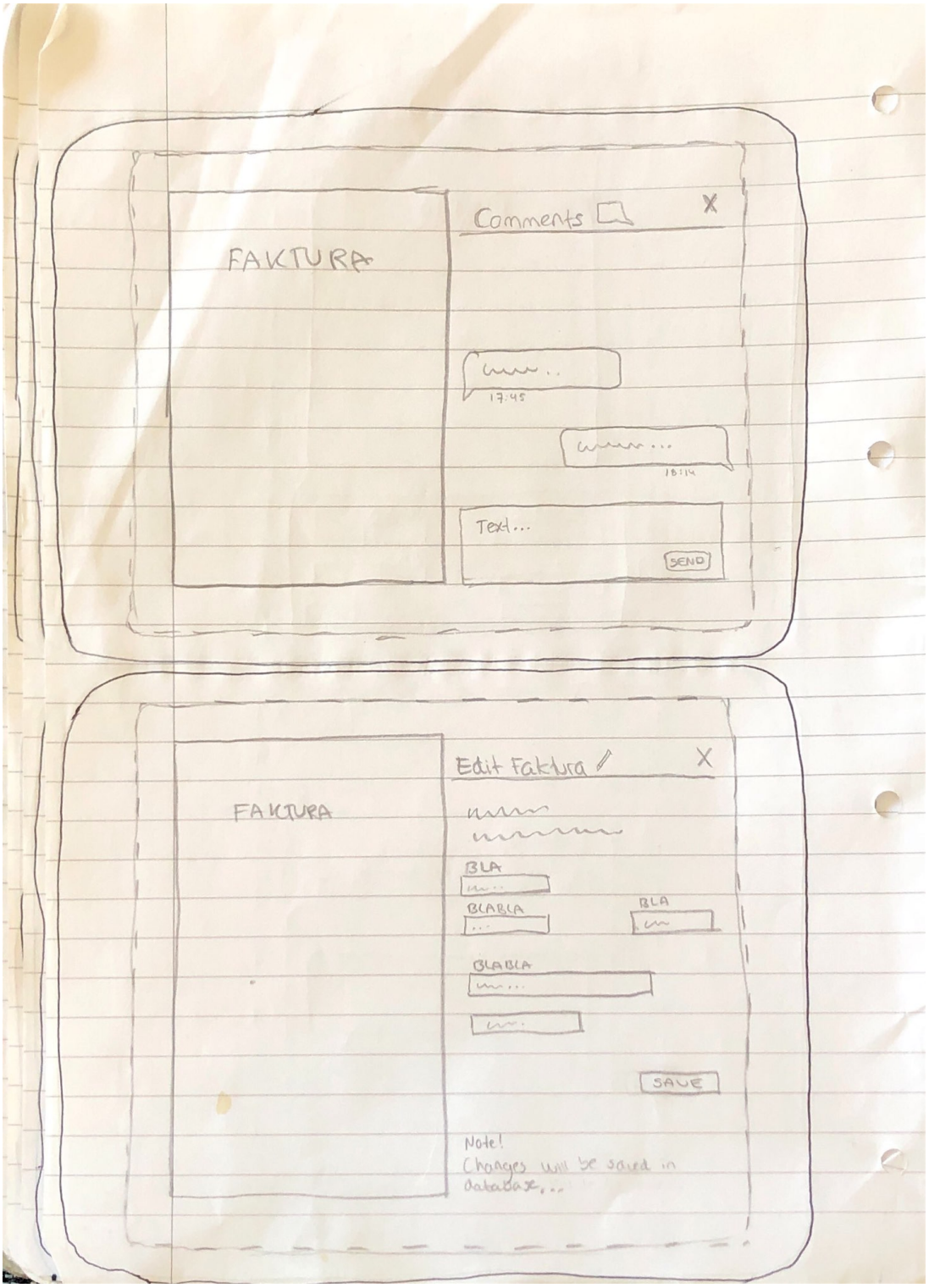


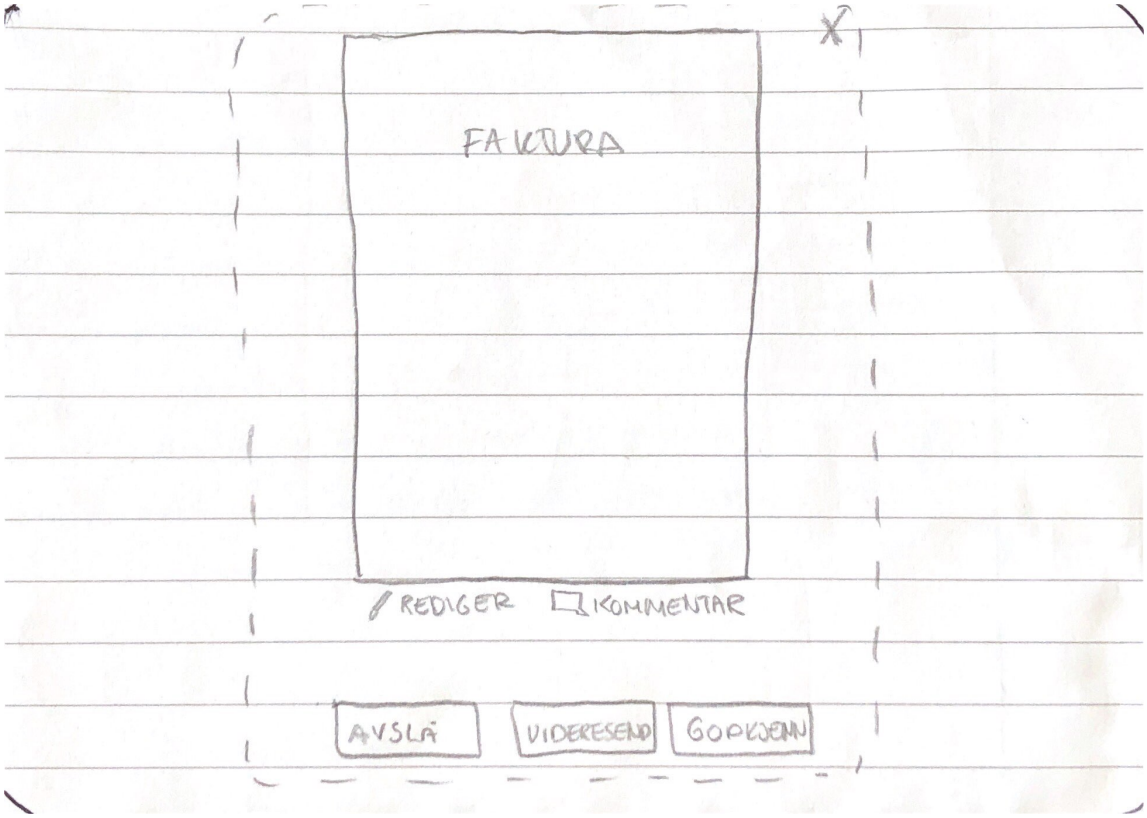

PASSWORD
LOGIN
Remember me
Forget password?
Forgot your password?
example@hotmail.com
SEND
Return to homepage

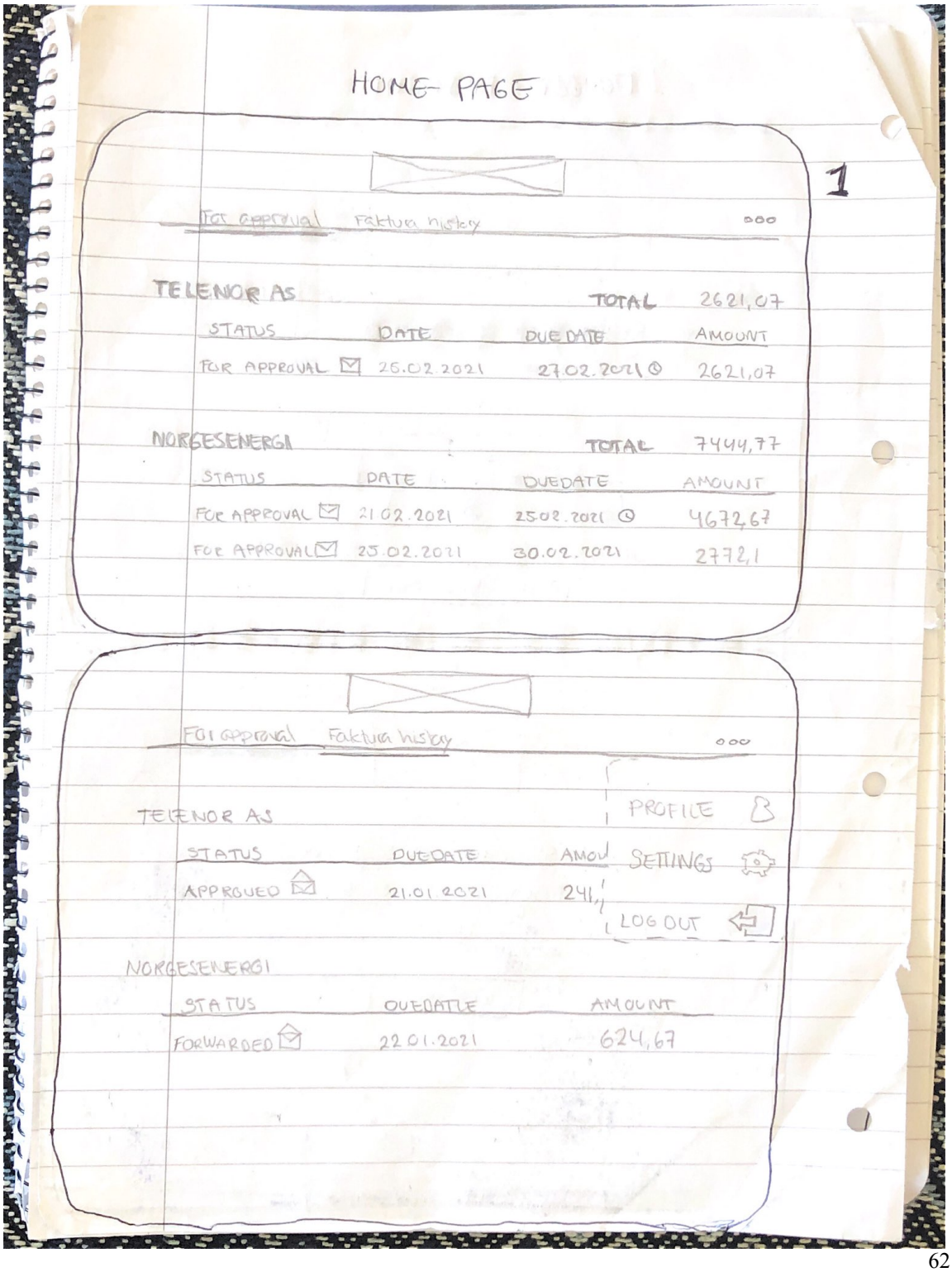

2
TELENOR AS
TOTAL 2741,7
FOR APPROVAL
STATUS DATE DUEDATE AMOUNT
FOR APPROVAL 21.02.2021 27.02.2021 2741,7
FAKTURA HISTORY
NORGESENERGI
TOTAL
STATUS DATE DUEDATE AMOUNT
PROFILE
SETTINGS
LOG OUT
3
FOR APPROVAL
TELENOR AS
TOTAL
STATUS DATE DUEDATE AMOUNT
FAKTURA HISTORY


FAKTURA
FAKTURA

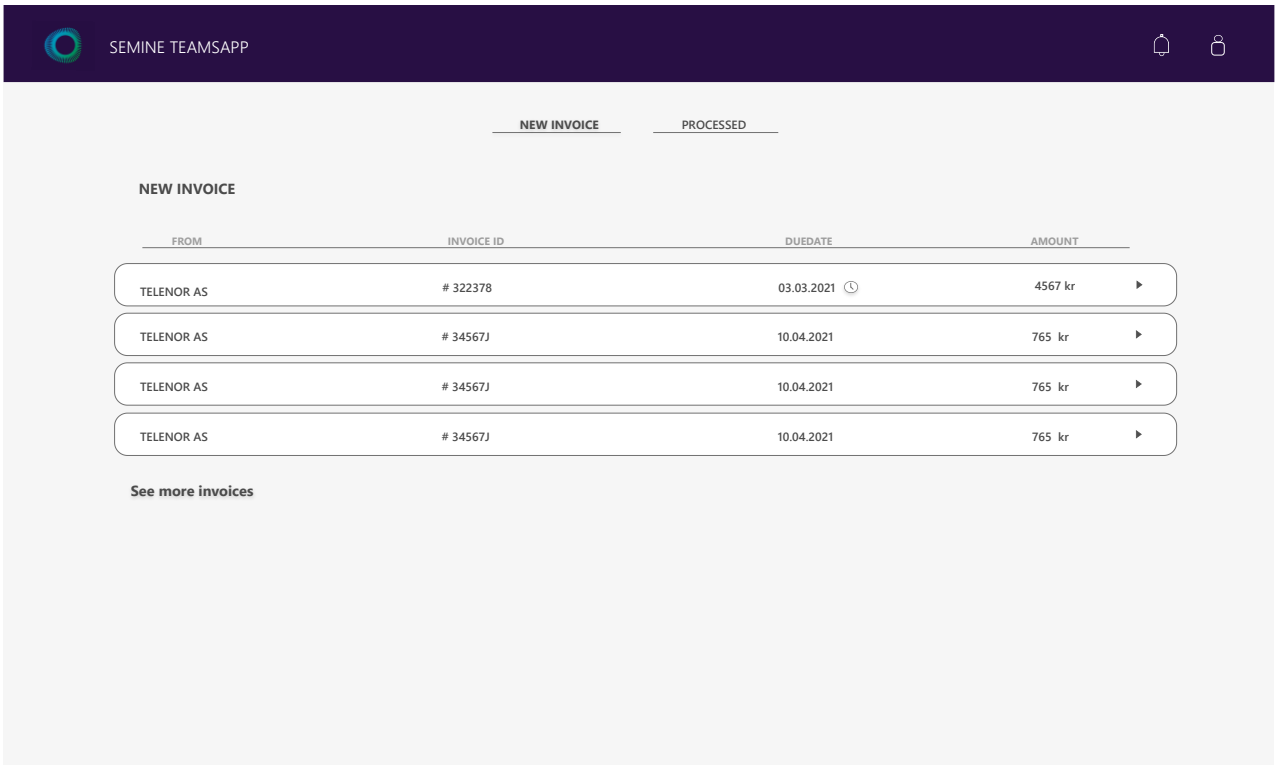

FAKTURA
FAKTURA FELT
BLABLA
BLABLA
BLA
SAVE
Magnus B.
Forward to...
Text...
SEND

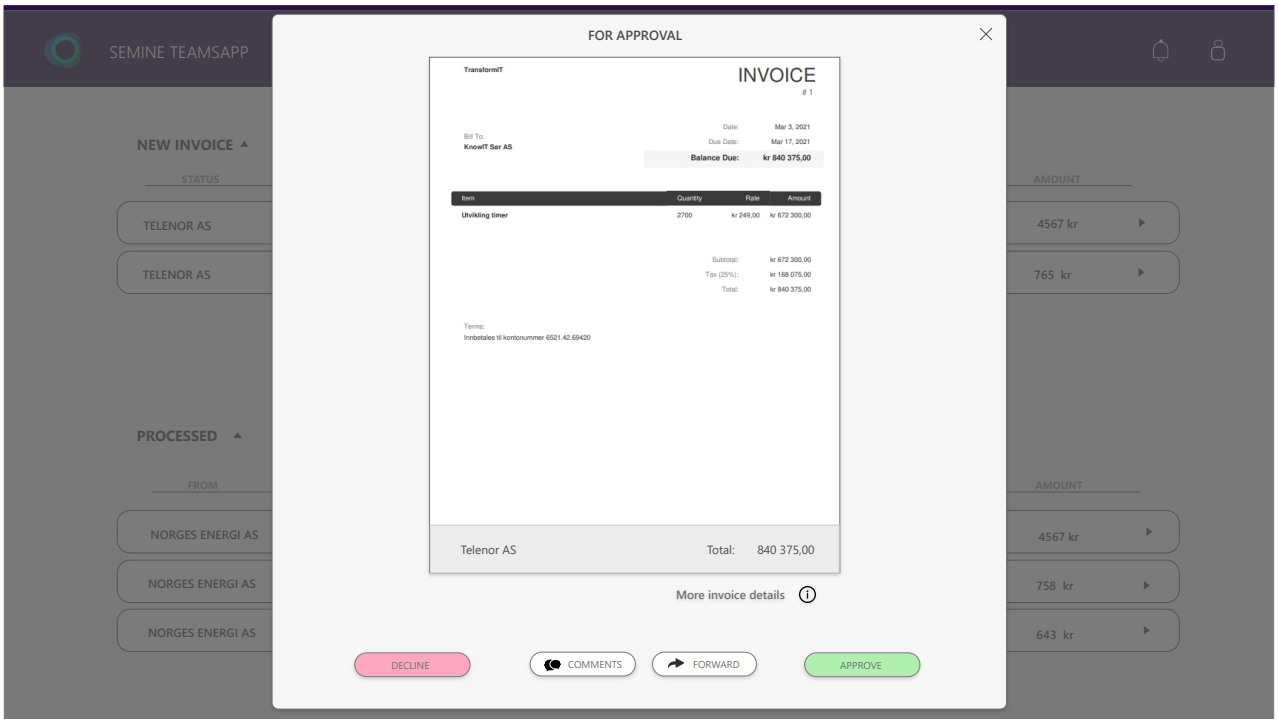

FAKTURA
Comments
X
17.45
18.14
Text...
SEND
FAKTURA
Edit Faktura
X
BLA
BLABLA
BLA
BLABLA
SAVE
Note!
Changes will be saved in
database...

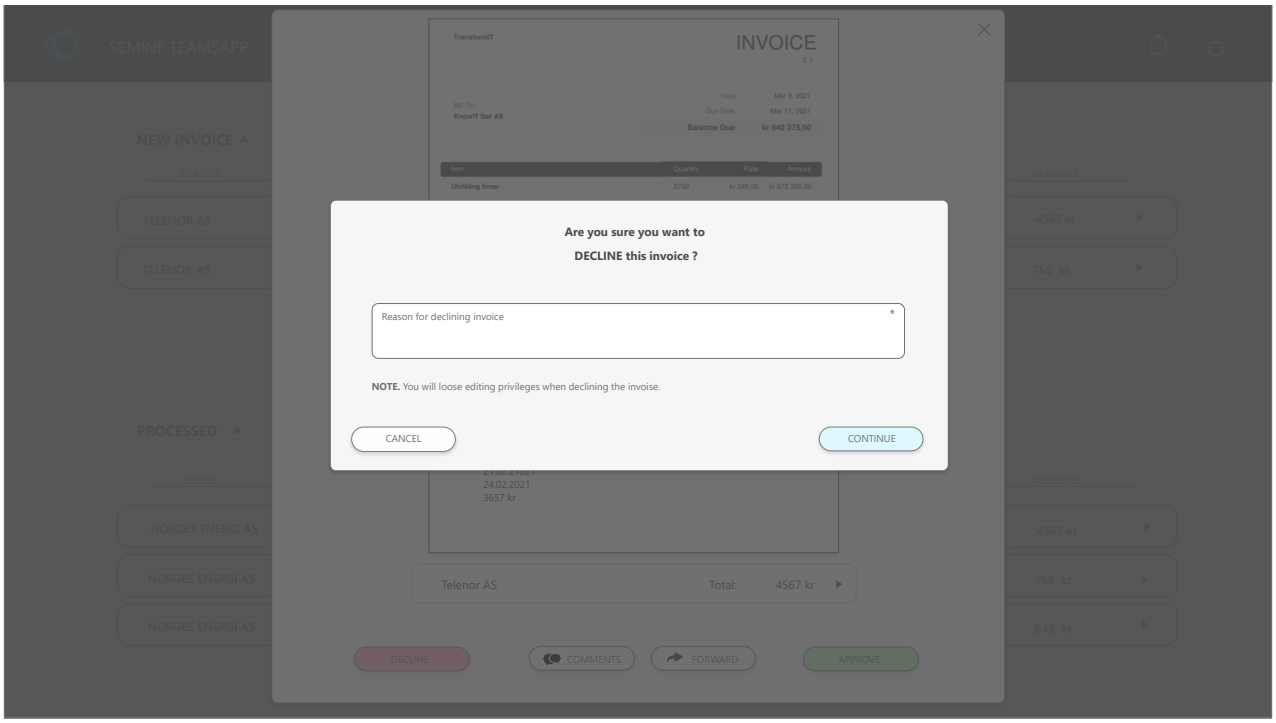

FAKTURA
REDIGER
KOMMENTAR
AVSLÅ
VIDERESEND
GODKJENN

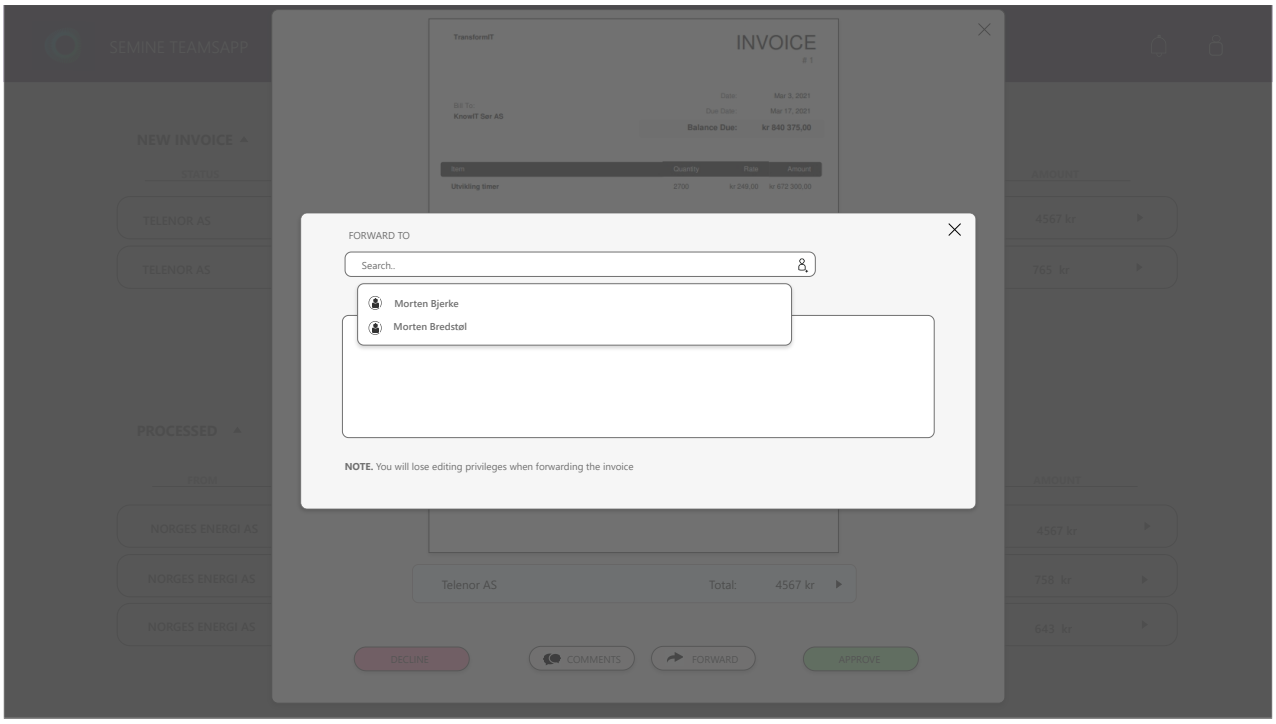

HOME PAGE
1
For approval
Faktura history
TELENOR AS
TOTAL 2621,07
STATUS DATE DUE DATE AMOUNT
FOR APPROVAL 26.02.2021 27.02.2021 2621,07
NORGESENERGI
TOTAL 7444,77
STATUS DATE DUEDATE AMOUNT
FOR APPROVAL 21.02.2021 25.02.2021 4672,67
FOR APPROVAL 25.02.2021 30.02.2021 2772,1
For approval
Faktura history
TELENOR AS
STATUS DUEDATE AMOU
APPROVED 21.01.2021 241,
NORGESENERGI
STATUS DUEDATTE AMOUNT
FORWARDED 22.01.2021 624,67
PROFILE
SETTINGS
LOG OUT

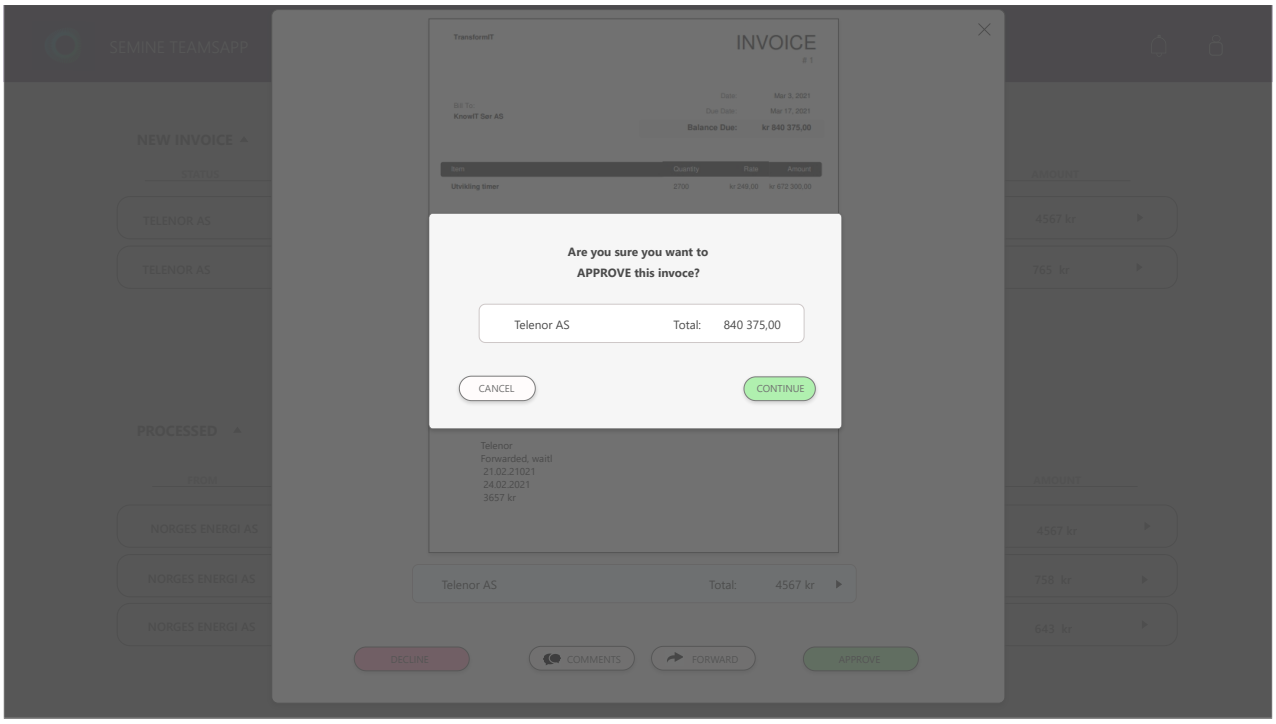
F Mockups


SEMINE TEAMSAPP
NEW INVOICE
PROCESSED
NEW INVOICE
FROM
INVOICE ID
DUEDATE
AMOUNT
TELENOR AS
322378
03.03.2021
4567 kr
TELENOR AS
34567J
10.04.2021
765 kr
TELENOR AS
34567J
10.04.2021
765 kr
TELENOR AS
34567J
10.04.2021
765 kr
See more invoices


SEMINE TEAMSAPP
NEW INVOICE
STATUS
AMOUNT
TELENOR AS
4567 kr
TELENOR AS
765 kr
PROCESSED
FROM
AMOUNT
NORGES ENERGI AS
4567 kr
NORGES ENERGI AS
758 kr
NORGES ENERGI AS
643 kr
FOR APPROVAL
TransformIT
INVOICE
1
Bill To:
KnowIT Sør AS
Date: Mar 3, 2021
Due Date: Mar 17, 2021
Balance Due: kr 840 375,00
Item
Quantity
Rate
Amount
Utvikling timer
2700
kr 249,00
kr 672 300,00
Subtotal: kr 672 300,00
Tax (25%): kr 168 075,00
Total: kr 840 375,00
Terms:
Innbetales til kontonummer 6521.42.69420
Telenor AS
Total: 840 375,00
More invoice details
DECLINE
COMMENTS
FORWARD
APPROVE


SEMINE TEAMSAPP
NEW INVOICE
STATUS
AMOUNT
TELENOR AS
4567 kr
TELENOR AS
765 kr
PROCESSED
FROM
AMOUNT
NORGES ENERGI AS
4567 kr
NORGES ENERGI AS
758 kr
NORGES ENERGI AS
643 kr
TransformIT
INVOICE
1
Bill To
KnowIT Sør AS
Date: Mar 3, 2021
Due Date: Mar 17, 2021
Balance Due: kr 840 375,00
Item
Quantity
Rate
Amount
Utvikling timer
2700
kr 249.00
kr 672 300,00
24.02.2021
3657 kr
Telenor AS
Total: 4567 kr
DECLINE
COMMENTS
FORWARD
APPROVE
Are you sure you want to
DECLINE this invoice ?
Reason for declining invoice
*
NOTE. You will loose editing privileges when declining the invoise.
CANCEL
CONTINUE


SEMINE TEAMSAPP
NEW INVOICE
STATUS
AMOUNT
TELENOR AS
4567 kr
TELENOR AS
765 kr
PROCESSED
FROM
AMOUNT
NORGES ENERGI AS
4567 kr
NORGES ENERGI AS
758 kr
NORGES ENERGI AS
643 kr
TransformIT
INVOICE
1
Bill To
KnowIT Sør AS
Date: Mar 3, 2021
Due Date: Mar 17, 2021
Balance Due: kr 840 375,00
Item
Quantity
Rate
Amount
Utvikling timer
2700
kr 249,00
kr 672 300,00
Telenor AS
Total: 4567 kr
DECLINE
COMMENTS
FORWARD
APPROVE
FORWARD TO
Search..
Morten Bjerke
Morten Bredstøl
NOTE. You will lose editing privileges when forwarding the invoice


SEMINE TEAMSAPP
NEW INVOICE
STATUS
AMOUNT
TELENOR AS
4567 kr
TELENOR AS
765 kr
PROCESSED
FROM
AMOUNT
NORGES ENERGI AS
4567 kr
NORGES ENERGI AS
758 kr
NORGES ENERGI AS
643 kr
TransformIT
INVOICE
1
Bill To
KnowIT Sør AS
Date: Mar 3, 2021
Due Date: Mar 17, 2021
Balance Due: kr 840 375,00
Item
Quantity
Rate
Amount
Utvikling timer
2700
kr 249,00
kr 672 300,00
Are you sure you want to
APPROVE this invoce?
Telenor AS
Total: 840 375,00
CANCEL
CONTINUE
Telenor
Forwarded, waitl
21.02.21021
24.02.2021
3657 kr
Telenor AS
Total: 4567 kr
DECLINE
COMMENTS
FORWARD
APPROVE