

Informatikk Prosjekt 1

IT1901 - NTNU

Gruppe 1

Alm, Thomas
Aulie, Kristoffer
Egge, Iver
Liverød, Lars
Skuldstad, Vegard



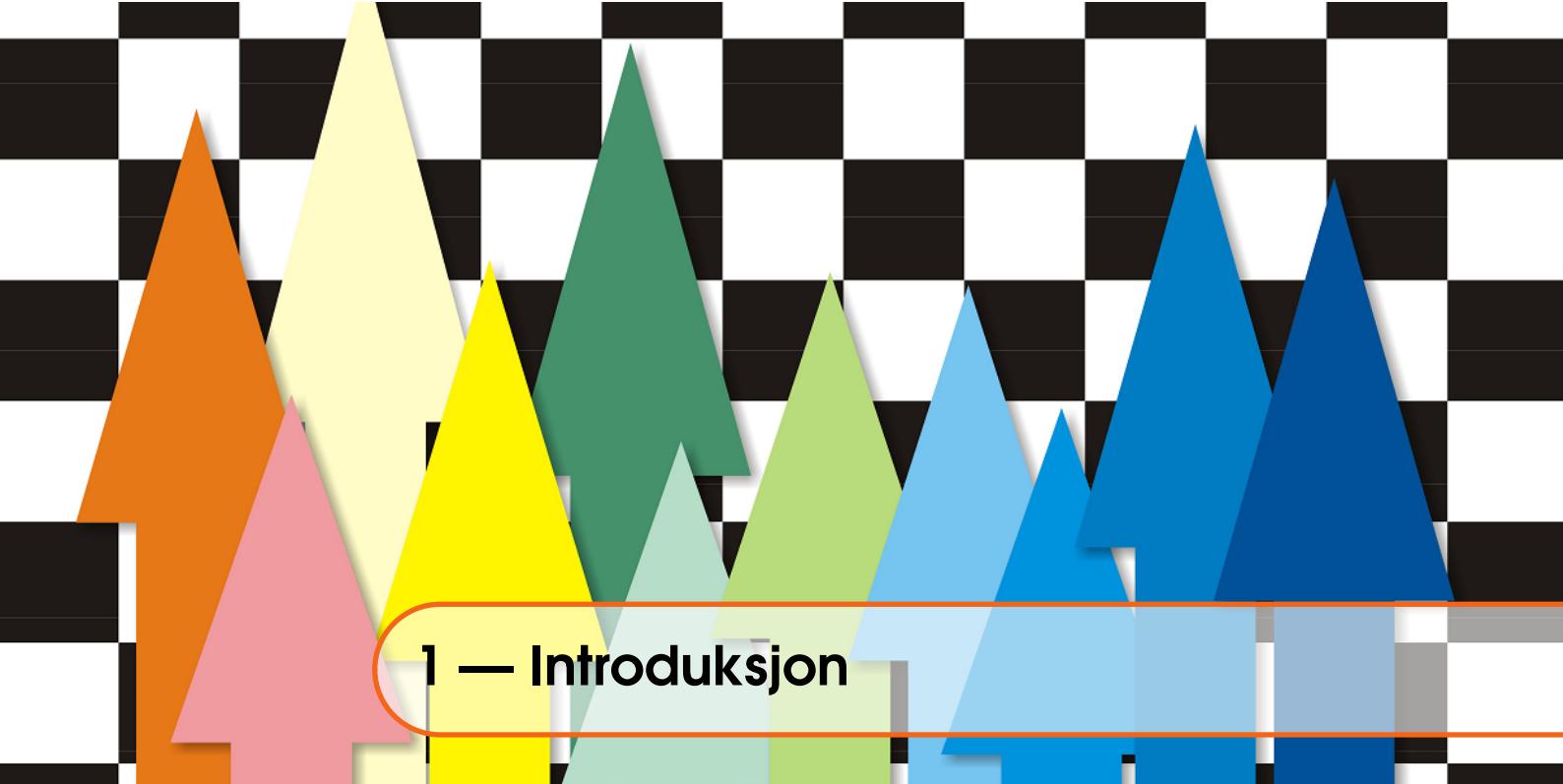


Innhold

1	Introduksjon	1
1.1	Faget	1
1.2	Prosjektet	1
1.3	Gruppen	1
1.4	Oppgaven	2
1.5	Prosessmetode	2
2	Prosjektorganisering	3
2.1	Ansvarsområder	3
2.2	Medlemmenes kunnskapsområder	4
3	Prosessbeskrivelse	6
3.1	Scrum	6
3.1.1	Beskrivelse	6
3.1.2	Roller	6
3.1.3	Arbeidsfordeling	7
3.1.4	Deling av kildekode	7
3.1.5	Sprinter	7
4	Tidsestimering	9
4.1	Estimering i Scrum	9
4.2	Arbeidstimer	10
5	Risiko i prosjektet	11

6	Arkitekturforklaring	12
6.1	Hovedkomponenter	12
6.2	NTNU Server - MySQL database	13
6.3	Klientapplikasjon (PC)	13
6.4	Simulator (Server)	13
6.5	JXMap og OpenStreetMap	13
6.6	Databaser	14
6.6.1	Databaseteknologi	14
6.6.2	Databasestruktur	14
7	Systemdesign	16
8	Testing	17
8.1	Papirprototypetest	18
8.2	Systemtest	18
8.3	Brukbarhetstest	18
9	Verktøy	19
9.1	Prosjekthåndtering	19
9.1.1	Google Drive	19
9.1.2	Facebook	19
9.1.3	ShareLaTeX	20
9.1.4	Rally dev	20
9.1.5	GitHub	20
9.2	Diagramverktøy	20
9.2.1	WBStool	20
9.2.2	GanttProject	21
9.2.3	Websequencediagrams	21
9.2.4	Gliffy	21
9.3	Implementering	21
9.3.1	NetBeans	21
9.3.2	MySQL	22
10	Vedleggsoversikt	23
10.1	Dokumentvedlegg	23
10.2	Programvedlegg	23
11	Vedlegg	24
11.1	Diagrammer	24
11.1.1	Work Breakdown Structure	24
11.1.2	Gantt-diagram	25
11.1.3	Timeliste	26

11.1.4 Risikomatrise	28
11.1.5 Deployment diagram	29
11.1.6 Entity Relationship-diagram	30
11.1.7 Sekvensdiagrammer	31
11.1.8 Rally-tasks eksempel	31
11.2 Kravspesifikasjon	35
11.2.1 User stories	35
11.2.2 Use case	39
11.3 Sprinter	47
11.3.1 Sprint 1	47
11.3.2 Sprint 2	49
11.3.3 Sprint 3	51
11.3.4 Sprint 4	53
11.3.5 Sprint 5	55
11.4 Testing	57
11.4.1 Papirprototypetest	57
11.4.2 Systemtest	60
11.4.3 Brukbarhetstest	66
11.5 Brukermanual	67
11.5.1 Teknisk brukermanual	67
11.5.2 Ikke-teknisk brukermanual	68



1 — Introduksjon

1.1 Faget

Beskrivelsen under er hentet fra emnesiden ¹.

I dette faget skal en gruppe på 4-6 studenter gjennomføre et mellomstort programmeringsprosjekt. Hovedmålet med faget er å gi studentene et innblikk i prosesser og produktorienterte utfordringer som kan oppstå i slike systemutviklingsprosjekter. I tillegg skal studentene lære seg å arbeide i team, noe som er meget relevant både for senere semestere og eventuelt arbeidsliv etter utdanningen. Studentene må arbeide med, reflektere og begrunne integreringen av ulike komponenter for å sette sammen et større produkt.

1.2 Prosjektet

Prosjektet innebærer et mellomstort programmeringsprosjekt som skal gjennomføres av en gruppe studenter i løpet av et semester. Det tar for seg en rekke utfordringer som gruppen må jobbe og reflektere over for å skape et fungerende system. Utviklingsprosessen står som et sentralt emne gjennom hele prosjektet, og det blir lagt stort fokus på dette i løpet av semesteret. Gruppen skal samarbeide som om det er et ekte utviklingsprosjekt i arbeidslivet, der man har en reell kunde som man skal lage et produkt for. Hele prosjektet vil være en læringsprosess der gruppa sammen må finne ut hvordan de skal løse utfordringene som oppstår utover i semesteret. Dette fører til at store deler av ansvaret for at produktet skal bli vellykket ligger hos gruppa, og gruppemedlemmernes evne til å samarbeide.

1.3 Gruppen

Gruppen består av totalt fem medlemmer med et variert kompetansenivå. Alle gruppemedlemme går Informatikk, men flere er på ulike årstrinn. Et av medlemmene kom fra en annen skole dette semesteret, vår SCRUM-master tar opp faget fra et tidligere semester, og tre av medlemmene har begynt sitt andre år på Informatikk. Dette gir gruppen et variert utgangspunkt for kompetanse,

¹<https://wiki.online.ntnu.no/projects/18/wiki/IT1901>

noe som åpner for at alle medlemmene kan lære av hverandre, siden de har forskjellige ting å bidra med. Gruppen består av individer som er interessert i å gjennomføre et godt prosjekt, noe som har ført til at motivasjonen hos alle gruppemedlemmene er meget god.

1.4 Oppgaven

Oppgaven vi har fått er å lage et system som sauebønder kan bruke til å administrere sauene sine. Bonden skal kunne spore posisjonen til sauene sine, samt bli varslet dersom en av dem blir angrepet. Han skal også kunne registrere en nødkontakt som også skal få alarm dersom en sau blir angrepet. Systemet skal loggføre sauenes lokaliserings- og angrepsrapporter, og han skal også kunne redigere informasjon om seg selv og sauene sine. De detaljerte kravene til systemet har vi fått utdelt i en kravspesifikasjon som vi har tolket og formulert på en måte som passer vår prosessmetode.

1.5 Prosessmetode

Valget av prosessmetode kan ha ekstremt mye å si for hvordan et utviklingsprosjekt gjennomføres. Her er det viktig å bruke riktige metoder til riktige prosjektyper. Dette prosjektet skal gjennomføres av et nokså lite team som befinner seg på samme geografiske sted. Disse omgivelsene legger mye til rette for at man skal kunne benytte seg av en smidig prosessmetode som Scrum. Siden Scrum passer godt for små til mellomstore prosjekter, i tillegg til at det er mye brukt i arbeidslivet og i senere semestre, falt valget vårt på denne prosessmetoden.

Det at Scrum er smidig medfører også den store fordelen av at vi jevnlig er i kontakt med kunden som dermed kan gi innspill underveis i utviklingsprosessen. Dette øker sjansene for at produktet vi ender opp med er det kunden faktisk ønsker seg, siden det ofte er store avvik mellom hva kunden beskriver i kravspesifikasjonen, og hva han egentlig er ute etter. Samtidig fører som regel høy kundeinvolvering til mer fornøyde kunder, siden de føler seg inkludert i prosessen. En bakdel med Scrum er at den høye graden av selvstyring i prosjektgruppa kan føles som unaturlig for enkelte, noe som kan føre til at ting ikke blir gjort. Dette krever at gruppa tar et kollektivt ansvar, noe Scrum-master må legge til rette for at skal kunne skje. Du kan lese mer om hvordan Scrum fungerer i praksis her: [Scrum 3.1](#)



2.1 Ansvarsområder

Siden vi benytter Scrum i dette prosjektet, vil vi ikke ha noen entydig ansvarsfordeling når det gjelder selve systemutviklingsoppgavene utover i semesteret. Scrum bygger på at alle gruppedelmedlemmer skal ha innvirkning på alle oppgavene, og man skal ikke ha én prosjektleader, én som gjør alt med database, én som gjør alt av GUI osv. Meningen er at man skal fordele oppgaver fortløpende i hver sprint, der alle gjør litt av hvert.

Allikevel ble vi under midtveispresentasjonen gjort oppmerksom på at selv om vi ikke fordeler alt arbeidet på forhånd i forskjellige kategorier, vil det være smart at alle gruppemedlemmene har et overordnet ansvar for de ulike delene av prosjektet, slik som database, GUI, kodeoppgaver osv. Disse ansvarspersonene har dermed i oppgave å sikre at oppgavene innenfor deres ansvarsområde blir utført, noe som unngår misforståelser i tillegg til at det sikrer skikkelig fremgang i prosjektet.

I Scrum er det viktig å skille mellom Scrum Master og prosjektleder. Scrum Masteren skal kun legge forholdene til rette for at Scrum kan følges av gruppa, han skal ikke ta noen sjefsavgjørelser; alle gruppemedlemmene har like stort ansvar.

Når vi har tatt hensyn til det vi har beskrevet over, har vi valgt å definere noen praktiske ansvarsområder i forbindelse med organisering og rapportskriving, slik som møtereferat, booking av møterom, logg, overordnet databaseansvar osv.

Kristoffer Aulie

- Scrum Master
- Scrum - sprinter
- Overvåke databaseoppgaver

Iver Egge

- Kodekonvensjoner
- LaTeX

Lars Liverød Andersen

- Møtereferat, logg
- Overvåke GUI-oppgaver

Vegard Skulstad

- Javadoc
- Overvåke kodeoppgaver

Thomas Alm

- Booking av møterom
- Overvåke rapporten (sikre at alt er med)

Grunnen til at vi har fordelt ansvarsområdene slik er stort sett basert på at gruppemedlemmet har tidligere erfaring innenfor området. Noen eksempler på det er at Kristoffer har erfaring med Scrum og databasehåndtering fra tidligere prosjekter, og kan bidra til at de andre får økt forståelse av hvordan en Scrum-Master rolle fungerer i praksis. I tillegg har Iver skrevet endel i LaTeX før, Lars har god innsikt i grafisk brukervennlighet, Vegard har kunnskap innen JavaDoc, og Thomas har et godt overblikk over hva en komplett rapport bør inneholde.

2.2 Medlemmenes kunnskapsområder

Her har vi valgt å kun liste opp kunnskapsområder som er relevante for dette prosjektet. Alle gruppemedlemmene er fra Informatikk, men har likevel ulike kunnskapsområder og erfaring. Det at medlemmene har ulike ting å bidra med, medfører at alle medlemmene kan lære mye av hverandre underveis i prosjektet.

Kristoffer Aulie

- Java
- Java Swing
- MySQL
- Scrum
- Diagrammer (ER, Gantt, WBS, Sekvensdiagrammer mm.)

Iver Egge:

- Java
- LaTeX

Lars Liverød Andersen:

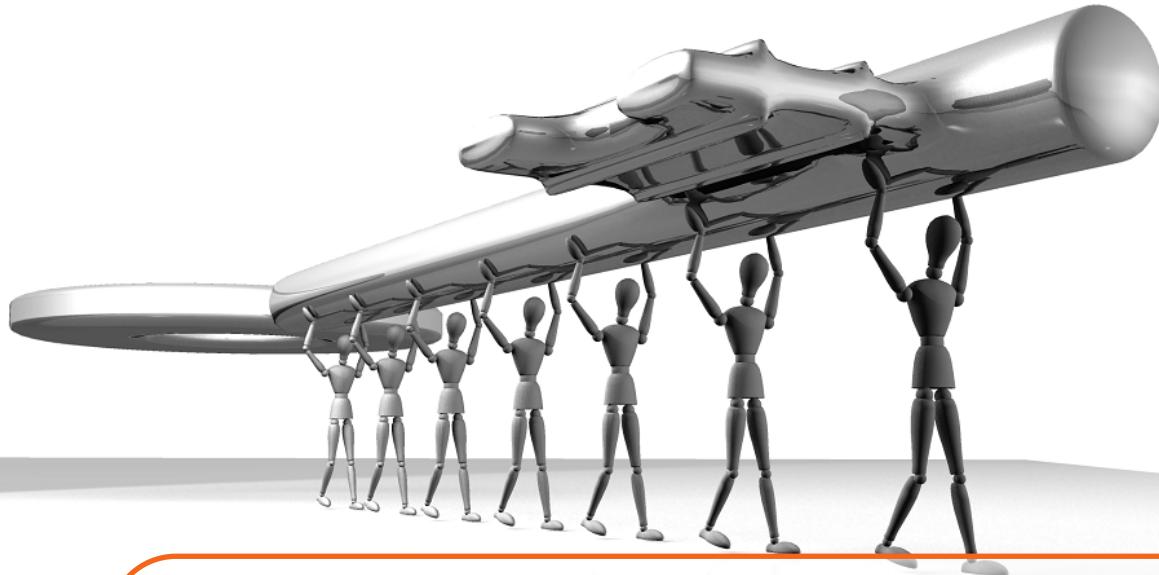
- Java
- Java Swing

Vegard Skulstad:

- Java
- Javadoc

Thomas Alm:

- Java
- Java Swing



3 — Prosessbeskrivelse

3.1 Scrum

3.1.1 Beskrivelse

Scrum er en smidig systemutviklingsmodell som er mye brukt i små til mellomstore programvareutviklingsprosjekter¹. Det at den er smidig vil si at den kan endre seg over tid. Det bygger på idéen om at man i utviklingsprosjekter sjeldent kan klare å vite alle kravene til en løsning før man begynner å implementere den. Dette er fordi man underveis får erfaringer og innblikk som endrer både utvikleren og kunden sin oppfatning av hvordan systemet bør være. Kunden klarer sjeldent å kommunisere nøyaktig hva han vil ha på forhånd, noe som fører til at kravene til systemet endrer seg underveis.

Scrum følger derfor en iterativ inkrementell prosess der man prøver å lage noe fungerende så raskt som mulig, slik at kunden kan se resultatet og raskt kan komme med eventuelle innvendinger som kan føre til at man må gjøre noe annerledes. Dette fører til at man raskere kan oppdage om man er på riktig spor i forhold til hva kunden faktisk ønsker.

Hver av disse inkrementene kalles for sprinter, der man i hver sprint viser noe nytt til kunden og får tilbakemeldinger på om man er på riktig spor eller ikke. Dette er forskjellig fra fossefall-metoden der alle krav defineres på forhånd før systemet designes og implementeres uten stor brukerinnvirkning i prosessen, noe som fører til at man ikke vet før på slutten av prosjektet om man har laget det kunden faktisk ønsket seg.

3.1.2 Roller

Scrum Master: Kristoffer Aulie

Scrum Team: Lars Liverød Andersen, Iver Egge, Vegard Skulstad, Thomas Alm

Kunde: Studass - Ørjan Breimo Helstrøm

¹[http://en.wikipedia.org/wiki/Scrum_\(software_development\)](http://en.wikipedia.org/wiki/Scrum_(software_development))

I dette prosjektet har kunden en litt annen rolle enn han har i scrumprosjekter i arbeidslivet. Studentassistenten vår fungerer som kunden, noe som gir han en mer hjelpende rolle enn en typisk kunderolle. Her har vi fått alle systemkravene utdelt på forhånd, vi skal altså ikke hente inn dette selv fra kunden, noe som betyr at kundens beskrivelse av hvordan systemet skal være, ikke vil endre seg på samme måte som det ville ellers.

Allikevel er det fortsatt åpent for hvordan vi som utviklere velger å tolke og løse kravene, noe som gjør at vi selv kan komme med innvendinger og forslag til endringer etter hver spint, ettersom vi får nye idéer til hvordan ting kan løses på bedre måter. Studass vil også kunne komme med innvendinger og forslag til bedre løsninger, noe som er grunnen til at Scrum vil kunne gi oss et godt utbytte; også i dette prosjektet.

3.1.3 Arbeidsfordeling

Arbeidsfordelingen i Scrum fungerer slik at alle teammedlemmene har ansvar for fremdriften i prosjektet. På hvert møte har vi arbeidet sammen, samt avtalt hva hvert medlem skal gjøre før neste møte. Vi har stort sett gitt hverandre varierte oppgaver som omhandler både rapportskriving og systemutvikling, noe som har ført til en jevn arbeidsfordeling der alle har fått prøvd seg på ulike oppgaver. Vi har også hatt noen spesifikke ansvarsområder i forbindelse med prosjektet, noe som er beskrevet i seksjonen Prosjektorganisering. For å sikre en jevn arbeidsfordeling, har vi konsekvent skrevet timelister på Google Drive der vi til enhver tid har oversikt over hvor mye alle teammedlemmene har jobbet.

3.1.4 Deling av kildekode

Underveis i prosjektet har vi brukt versjonskontrollen [GitHub](#) for å dele kildekoden mellom gruppemedlemmene, og samtidig sikre at det ikke oppstår konflikter i koden. Dette gir oss også en god oversikt over alle endringer vi gjør, samt hvem som gjør dem ettersom vi har en historie av commits der alle har passet på å beskrive hvilke endringer som er gjort hver gang vi pusher de. Dette gjør at alle kan følge med på hvordan de ulike taskene i user storyene i Spint Backlog ligger an, i tillegg til å se statusen i vårt digitale Scrum-verktøy; [Rally](#).

Som beskrevet under seksjonen [Verktøy](#) har NetBeans innebygget git-funksjonalitet, noe som gjør det enkelt og raskt å committe, pulle og pushe fra GitHub-serveren direkte i IDEen. Vi har passet spesielt godt på å committe og pushe endringer ofte, slik at alle til enhver tid vil ha tilgang til den nyeste versjonen av kildekoden.

3.1.5 Sprinter

Basert på våre møtetidspunkter som er tirsdager og fredager, samt møte med studass hver 14. dag på fredager, har vi bestemt oss for å bruke sprinter som er 13 dager lange (fredag til torsdag). På denne måten vil vi hver tirsdag og fredag kunne gjennomføre “stand ups” der vi forteller om hva vi har gjort siden sist, hva vi planlegger å gjøre til neste gang, samt hva som eventuelt stod i veien for at man fikk fullført oppgaven sin. Vanligvis kalles dette for “Daily Scrum meetings”,

men siden dette prosjektet ikke er på fulltid vil ikke dette foregå hver dag.

Hver sprint starter etter møtet med studass annenhver fredag. Da har vi et “Sprint planning meeting” der vi bestemmer oss for hva vi skal gjøre i den neste sprinten. Vi lager en Sprint Backlog som består av user stories hentet fra “Product Backlog” som beskriver hva slags funksjonalitet som skal implementeres. Vi deler hver user story inn i “Tasks” for å gi en klar oversikt over de spesifikke oppgavene som må gjøres for at et user story skal fullføres. Hver av disse taskene blir gitt ett tidsestimat slik at summen av disse estimerer hvor lang tid hvert user story vil ta. Et eksempel på en slik inndeling kan sees i vedlegget: [Tasks 11.15](#)

Siste frist for hver sprint er annenhver torsdag, mens vi fredagen etter hver sprint vil ha et møte med studass som fungerer som “Sprint review meeting” der vi gjennomgår arbeidet som ble gjennomført, samt det som ikke ble gjennomført og får tilbakemeldinger fra studass. Etter dette møtet går prosjektgruppa sammen og har et “Sprint Retrospective meeting”, der vi går gjennom hva som gikk bra og dårlig i forrige sprint, slik at vi kan gjøre forbedringer til neste sprint.

Siden prosjektet var såpass i oppstartsfasen de første par ukene, og ble mest brukt til gjennomføring av praktiske ting som måtte på plass, har vi valgt å begynne første sprint fra og med fredag 13. september.

Kravspesifikasjonen, som vi her benytter som produkt backlogg, kan finnes i vedlegget: [Kravspesifikasjon, \(11.2\)](#)

Alle sprintene, inkludert backlog, sprint møter og burndown charts, kan finnes i vedlegget: [Sprinter, \(11.3\)](#). Inkludert i disse sprintene er møtereferat fra sprint review møter og sprint retrospective møter. Derfor har vi ikke lagt ved noen andre møtereferater i vedlegget, selv om vi har skrevet logg for alle møtene vi har hatt; både med og uten studass.



4 — Tidsestimering

Når vi skal estimere tid i prosjektet tenkte vi i utgangspunktet å ikke ta i bruk en klassisk Work Breakdown Structure (WBS) eller et Ganttdiagram. Dette er fordi Scrum er laget slik at planene endrer seg over tid, noe som er hovedpoenget i smidige metoder. Vi tenkte at det ikke er mye poeng i å bryte ned alle arbeidsoppgavene og estimere når de skal gjøres og hvor lang tid de vil ta, siden det er meningen at planene og oppgavene endrer seg etter hver sprint.

Etter å ha pratet med studassen og undlassen valgte vi allikevel å lage en WBS ([figur 11.1](#)) for å få en overordnet oversikt over hva som skulle gjøres i prosjektet. Med et slikt diagram er det lettere å få en klar oversikt over hele prosjektet, enn kun ved å se på produkt backloggen. I tillegg gir WBS en oversikt over ting som ikke er med i produkt backloggen, slik som prosjektorganisering, læringsprosess og rapportskriving.

Vi valgte også å lage et ganttdiagram ([figur 11.2](#)) for å gi et overordnet estimat på hvor lang tid de ulike prosessene vil ta. Her har vi lagt inn de viktigste systemutviklingsoppgavene og gjort en foreløpig fordeling utover sprintene. Vi har ikke vist detaljerte oppgaver her siden sprintene i Scrum tar seg av det. I tillegg til sprintene har vi prosjektorganisering, læringsprosess, rapportskriving og resten av prosessene vi har med i vår WBS. WBS og ganttdiagrammet er med andre ord koblet sammen.

4.1 Estimering i Scrum

Når man bruker Scrum er det ikke vanlig å estimere hele prosjektet fra starten av; man estimerer heller hver enkelt sprint underveis. Det er heller ikke vanlig å estimere i antall timer. Man bruker heller “effort points” som et mål på hvor krevende en oppgave er, og gruppemedlemmene kommer til en enighet på hvor mye innsats som kreves for hver enkelt arbeidoppgave.¹

Allikevel er dette et nokså lite prosjekt, og etter samtale med faglærer, der vi fikk høre at man

¹<http://scrummethodology.com/scrum-effort-estimation-and-story-points/>

ikke måtte følge Scrum helt slavisk, valgte vi å estimere alle user stories i hver enkelt sprint i antall timer. Dette er fordi det gjør det lettere å sammenligne tidsestimatet med timelista vår, i tillegg til at antall timer det er forventet at hvert enkelt gruppemedlem bruker i uka er fastsatt (10 timer for 7,5 studiepoeng).

4.2 Arbeidstimer

Selv om det er beregnet 10 timer per person i uka, har vi ikke gitt hver enkelt person arbeidsoppgaver med større tidsestimat enn 7 timer i uka. Dette er for å ta hensyn til uforutsette hendelser, noe som er meget vanlig i programvareutviklingsprosjekter. De aller fleste slike prosjekter bruker mer tid, og medfører høyere kostnader enn beregnet på forhånd. Å estimere arbeidsoppgaver for maksimalt 7 timer i uka er dermed et tiltak for å unngå at vi overskridet tiden vi har til rådighet hver uke.

I tillegg fikk vi beskjed om at vi skulle ta hensyn til rapportskriving i tidsestimatet vårt, samt ta hensyn til møtevirksomhet og andre praktiske oppgaver som ikke omhandler direkte utvikling. Til dette har vi valgt å sette av to timer i uka per person. Vi har valgt å ikke ta med oppgaver som omhandler prosjektstyring og rapporter i vår Project Backlog i Scrum, der skal det kun være krav som går direkte på programvareutviklingen. Derfor vil rapportskriving og annet administrativt arbeid kun bli lagt til i beregningen under.

Med alt dette tatt med i beregningene blir regnestykket slik:

Antall timer per person i uka	10
Timer for uforutsette hendelser	-3
Timer for rapport, møter osv	-2
Antall timer til sprint backlog	5 timer per person i uka

Tabell 4.1: Tidsestimering

Vi er 5 personer, så dette gir oss 25 timer i uka, og 50 timer per sprint i rene systemutviklingsoppgaver. Dette vil være utgangspunktet for hvor mange timer summen av user storyene i hver Sprint Backlog vil utgjøre.

Underveis i prosjektet har vi ført en timeliste for å se hvor mye tid vi faktisk bruker på hver sprint slik at vi kan vurdere om vi beregnet riktig arbeidsmengde i sprintene. Timelisten hjelper oss også med å kontrollere at arbeidsfordelingen ikke blir ujevn. Timelisten kan sees i vedlegget: [Timeliste 11.1.3](#)



5 — Risiko i prosjektet

I et prosjekt som dette, er det stor risiko for at noe ikke går som planlagt. Som listet opp i risikomatrisen [Risikomatrisen](#), (figur 11.5), kan blant annet medlemmer være bortreist over lengre tid, vi kan feile på sprint-deadlines, databaseserveren kan være nede, tap av data kan skje, vi kan støte på problemer som ingen klarer å fikse, personer kan bli syke, eller kanskje enkelte personer ikke gjør sine tildelte oppgaver. De fleste av de nevnte situasjonene over kommer mest sannsynlig til å inntrefte. Derfor er det viktig å være klar over hvilke tiltak som bør bli gjort når de gjør det.

Ettersom nye utfordringer oppstår som vi muligens ikke hadde tenkt på tidligere, oppdaterte vi risikomatrisen kontinuerlig. Vi gjorde et forsøk på å dekke alle de relevante risikoene for denne typen gruppearbeid på forhånd, men i løpet av prosjektet måtte vi legge til en risiko: "Databaseserveren er nede". Siden vi brukte NTNU sine servere, tenkte vi ikke på forhånd at det ville være noe problem, men vi opplevde ved en anledning vi ikke fikk tilgang til databasen.



6 — Arkitekturforklaring

6.1 Hovedkomponenter

Arkitekturen til et system beskriver hvordan hele systemet henger sammen. Det sier hvilke ulike komponenter systemet består av, samt hvordan disse komponentene utveksler informasjon med hverandre. Vårt system består av flere ulike hovedkomponenter, der alle spiller en viktig rolle i å sørge for at hele systemet fungerer.

For å vise en modell av hvilke hovedkomponenter systemet består av, samt hvordan de kommuniserer, har vi valgt å lage et Deployment Diagram ([figur 11.6](#)). Slike diagrammer er laget for å vise hvilke enheter som kjører, i tillegg til hva som kjører på selve enhetene. Grunnen til at vi valgte å bruke et slikt diagram, er at vi mener det gir en god oversikt over hvordan hele systemet er bygget opp. Under følger en forklaring av de ulike komponentene i diagrammet.

For å vise programflyten i ulike scenarier, har vi også laget flere sekvensdiagrammer som kan sees i vedlegget: [Sekvensdiagrammer 11.1.7](#)

I “det virkelige liv” er det best å bruke et klient/tjener system der klienten kun kommuniserer med serveren, som igjen kommuniserer med databasen ¹. Etter at vi hørte fra fagstaben at vi ikke nødvendigvis måtte bruke klient/tjener arkitekturen, bestemte vi oss for å gjøre det slik at både simulatoren og klien ten kommuniserer med databasen. Dette gjorde vi fordi vi ikke hadde noe særlig erfaring med klient/tjener systemer, og vi valgte derfor å gjøre noe som vi hadde en viss anelse som hvordan vi ville implementere. Derfor kan vi ikke si at vi har en server i arkitekturen, selv om simulatoren er ment å kjøre på en pc hele tiden.

¹http://en.wikipedia.org/wiki/Client%E2%80%93server_model

6.2 NTNU Server - MySQL database

Det mest sentrale komponentet i systemet vårt er databasen vår som inneholder all informasjon om sau, bønder, nødkontakter og rapporter. Vi valgte å benytte oss av MySQL-databasen i hjemmeområdet vårt på NTNU sine servere, siden det er gratis, driftet godt av NTNU, samt meget enkelt å sette opp. Både simulatoren og klientapplikasjonen henter ut og legger inn informasjon i databasen via en JDBC tilkobling, som er et grensesnitt for databasetilkobling med Java.

6.3 Klientapplikasjon (PC)

Klientapplikasjonen er den applikasjonen som bonden bruker for å få tilgang til all informasjonen han er interessert i. Det er her selve det grafiske grensesnittet ligger, der bonden kan navigere rundt i applikasjonen, samt gjennomføre handlinger som opprettning eller redigering av informasjon om sau. Denne applikasjonen kan kjøre på en hvilken som helst PC som er tilkoblet internett og støtter JAR filer. Mange klienter kan i tillegg kjøre samtidig.

Alle klientene er koblet opp mot den samme databasen som nevnt over, og henter jevnlig den nyeste informasjonen fra databasen vha. en timer. I tillegg kan bonden selv velge å oppdatere informasjonen når han selv vil vha en oppdateringsknapp i GUI. Klientprogrammet benytter seg av et eksternt kart API, OpenStreetMaps, via JXMap.

6.4 Simulator (Server)

Simulatoren kjører hele tida på en server og genererer rapporter med lokasjonsdata og helsedata. Simulatorprogrammet har ingen interaksjon med brukeren, annet enn at den sender ut alarmer til bonden og nødkontakten dersom en sau er under angrep. Også denne henter og legger inn informasjon i databasen via JDBC.

Dersom dette systemet skulle ha blitt benyttet i virkeligheten, ville simulatoren ha vært erstattet med et program som lyttet på målere som fysisk var plassert på sau, slik at GPS og puls kunne rapporteres inn til serverapplikasjonen. Applikasjonen ville så generere rapporter og sende de inn til databasen, samt sende ut eventuelle alarmer ved angrep. I deployment-diagrammet vårt har vi valgt å ta utgangspunkt i det systemet vi i vårt tilfelle har, siden det bedre forklarer arkitekturen i systemet vi faktisk har laget.

6.5 JXMap og OpenStreetMap

Klientapplikasjonen benytter seg av et kartbibliotek som heter JXMap, som igjen benytter seg av OpenStreetMap. JXMap er en open source Swing-modul som kan brukes direkte i Java Swing og laster inn kart fra en bildeserver. Modulen har innebyggede funksjoner, noe som gjør det lett å arbeide med lokasjonsdata i kartet. OpenStreetMap er et open source kart over hele jorda som

enkelt kan lastes inn via JXMap-modulen.

Grunnen til at vi valgte å bruke JXMap med OpenStreetMap, var at det passet godt inn i en Java Swing applikasjon, var enkelt å bruke, i tillegg til at kartet er responsivt, detaljert og lett å navigere i. I begynnelsen tenkte vi på å bruke Google Maps, men vi fant ingen enkel og god løsning på dette for desktop-applikasjoner. Selv om vi synes OpenStreetMap er detaljert nok til vårt formål, er den eneste mulige svakheten vi kan tenke oss at det antagelig er andre kart som vil kunne være mer detaljerte på landsbygda i Norge. Dette vil kunne ha mye å si for en eventuell bonde som gjerne vil kunne se mer av hvordan terrenget ser ut i kartet. Da vil Statens Kartverk kunne være en bedre løsning, men for vår del følte vi altså at OpenStreetMap var godt nok.

6.6 Databaser

6.6.1 Databaseteknologi

I dette prosjektet har vi en sentral database som inneholder all informasjon som sendes til og fra klientene og simulatoren. Vi har valgt å bruke MySQL siden det var lett å sette opp via vårt studentområde på NTNU-serverene, i tillegg til at vi hadde endel erfaring med det fra tidligere. MySQL er også meget utbredt, så det er lett å finne dokumentasjon om det på nettet. Dette gjorde det lettere for oss å finne svar raskt dersom vi stod fast med noe relatert til databasen.

Ved å bruke InnoDB, som er en lagringsmotor for MySQL-databaser, støtter databasen flere restriksjoner i og mellom tabellene som har nytte egenskaper. Noen av dem er at man kan lage relasjonsdatabaser ved hjelp av fremmednøkler som kobler sammen flere tabeller. Dette gjør at vi kan normalisere databasen, slik at vi unngår dobbeltlagring og inkonsistens i dataene. I tillegg kan vi spesifisere hva som skal skje dersom en rad som har en fremmednøkkel mot en annen tabell slettes eller oppdateres. Et eksempel er at dersom en bonde endrer sin e-postadresse, som er primærnøkkelen i farmer-tabellen og fremmednøkkelen mot sheep-tabellen, så skal feltet i sheep med f_email også oppdateres slik at fremmednøkkelen opprettholdes.

6.6.2 Databasestruktur

For å vise vår databasestruktur har vi laget et ER-diagram som viser tabellene våre, samt relasjonene mellom de. Henviser til [figur 11.7](#). Som man kan se av diagrammet, har vi laget fire tabeller i databasen; sheep, farmer, emergencycontact og report. Vi har følgende forhold i tabellene for å oppnå høy normalisering:

- En bonde kan ha mange sauер
- En sau har nøyaktig én bonde
- En bonde har nøyaktig én nødkontakt
- En nødkontakt kan være nødkontakt for flere bønder.
- En sau kan ha null eller flere rapporter.
- En rapport har nøyaktig én sau.

I **sheep** har vi valgt å inkludere generell informasjon om sauene, samt siste koordinater og helse-informasjon. Denne tabellen har også en fremmednøkkelen “f_email” mot farmer-tabellen. Vi har valgt å bruke en *INT AUTO_INCREMENT* som primærnøkkelen i sheep siden dette gir en unik id for hver sau, noe som tillater at flere sauene kan ha samme navn. Vi bruker også restriksjone *ON DELETE CASCADE* som fører til at alle sauene til en bonde som slettes også blir slettet, og *ON UPDATE CASCADE* som oppdaterer “f_email” i sheep dersom bonden endrer sin e-post og “f_email” i farmer dermed endrer seg. Dette sikrer at relasjonen bevares.

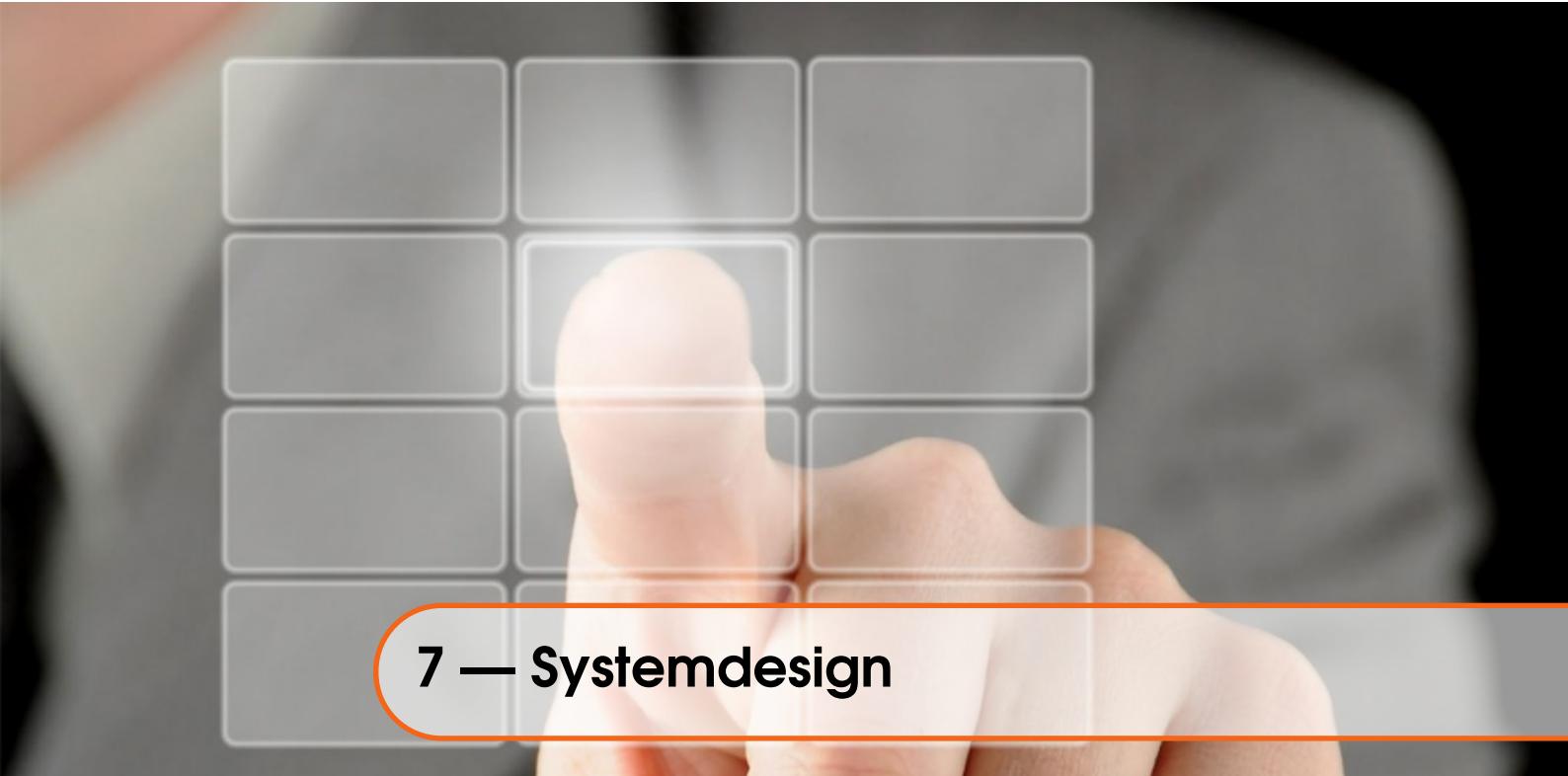
I **farmer** lagrer vi informasjon om bonden, samt GPS koordinatene til bondegårdens beliggenhet. Dette er for å holde styr på hvor gården ligger, slik at bonden kan se sin egen gård på kartet og sauenes posisjon i forhold til den. Dette lar også simulatoren generere posisjonene for bondens sauere slik at de kommer i nærheten av gården, noe som fører til at simuleringene virker mer realistiske. Her har vi en fremmednøkkelen “ec_email” mot emergencycontact-tabellen. Primærnøkkelen i farmer er “f_email” siden det ikke går annet å ha til like e-postadresser, og den egner seg dermed godt som unik identifikator. Også her sikrer vi at relasjonen bevares vha. *ON UPDATE CASCADE* som endrer “ec_email” i farmer dersom nødkontaktenes e-post endres, og “ec_email” i emergencycontact dermed endres.

I **emergencycontact** lagrer vi informasjonen om en nødkontakt. Primærnøkkelen her er “ec_email” av samme grunn som hvorfor vi brukte e-post som identifikator i farmer-tabellen, nemlig at den alltid er unik. Vi har laget programmet slik at en nødkontakt ikke registreres i databasen flere ganger, så dersom en bonde registrerer en nødkontakt som har vært registrert av en annen bonde fra før av, vil ikke noen ny nødkontakt bli opprettet. I stedet vil bonden referere til den allerede eksisterende nødkontakten i tabellen. Databasen ville uansett ikke lett oss opprette en ny rad med en allerede eksisterende “ec_email” siden primærnøkkelen må være unik i tabellen.

I **report** lagres saueraffører som simulatoren genererer. Både koordinater, puls og tidspunkt lagres her. Primærnøkkelen er en teller *INT AUTO_INCREMENT* som automatisk gir en unik id økende for hver rad, akkurat som i sheep-tabellen. Tabellen har en fremmednøkkel “s_id” som refererer til “s_id” i sheep-tabellen slik at vi vet hvilken sau rapporten omhandler. Siden “s_id” ikke kan endres, har vi ikke *ON UPDATE CASCADE* her. Vi går ut i fra at bonden ikke har bruk for rapportene til en sau som slettes, så vi har tatt i bruk *ON DELETE CASCADE*.

På grunn av måten vi har konstruert tabellene på, oppfyller relasjonene Boyce-Codd normalform, noe som er en sikring for å unngå dobbeltlagring og inkonsistens i databasen ².

²http://en.wikipedia.org/wiki/Boyce%E2%80%93Codd_normal_form



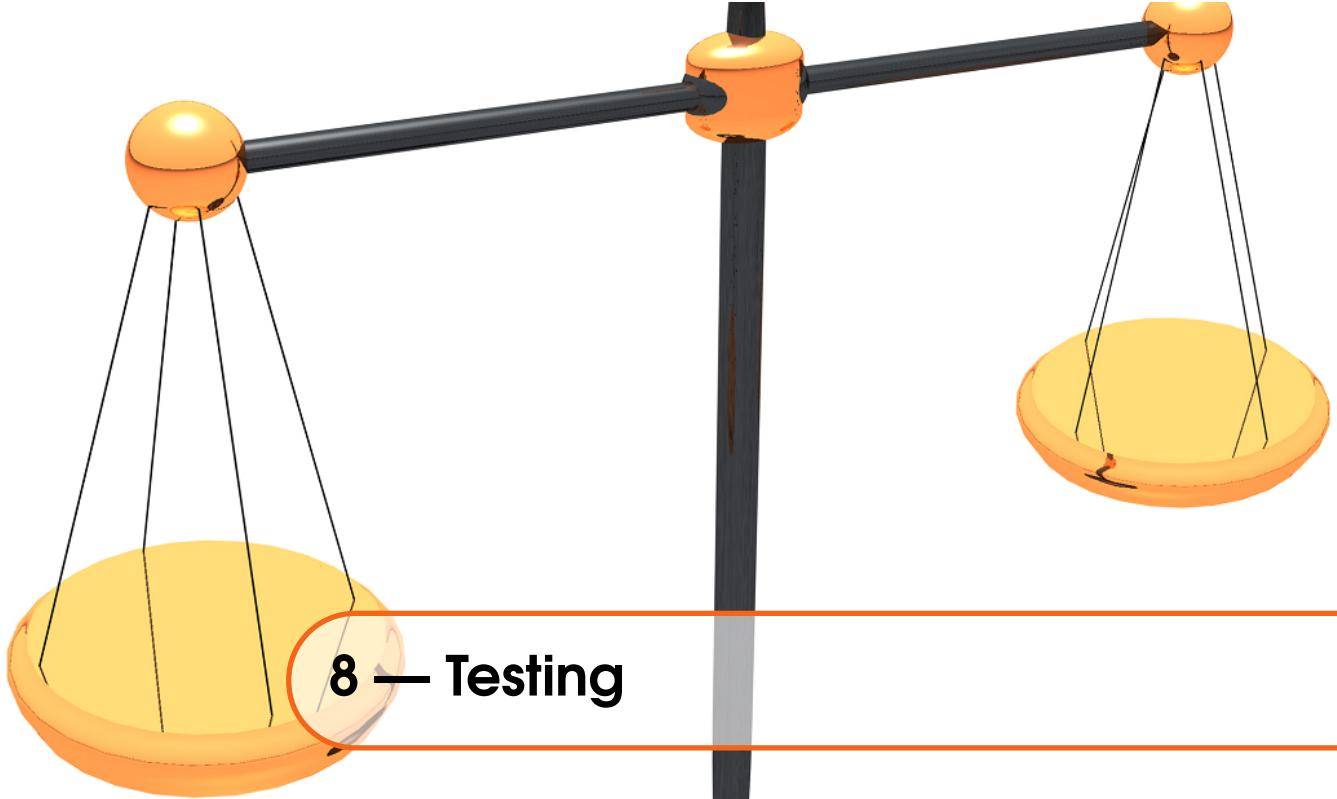
7 — Systemdesign

I et system på linje med det vi har utviklet her kommer det som en naturlig del å vise fram (mye) informasjon til brukeren. Vi valgte å formidle denne informasjonen gjennom et intuitivt og enkelt brukergrensesnitt. Vi bestemte oss for å ha en logisk oppdeling av brukergrensersnittet og benyttet oss av ulike faner/vinduer, hvor hver fane har en bestemt baktanke og funksjon. Hovedtanken var å fordele den nødvendig informasjonen utover et større omfang, som igjen vil være med på å skape et mer oversiktlig system. Hvis all informasjon skulle ha blitt vist i ett vindu ville det fort ha blitt mye rot, og dimensjonene på vinduet ville bli unødvendig store for å dekke alt av informasjon. Faner er også noe som er brukt veldig ofte i andre systemer, og er blitt en naturlig og intuitiv del av bl.a. webbaserte applikasjoner.

Det er mulig å navigere seg gjennom systemet ved hjelp av knapper. Hver knapp har sin beskrivelse som forteller om sin bestemte funksjon. Det første GUI-elementet som blir vist til brukere er Login-vinduet. Her blir brukeren bedt om å enten registrere seg, eller benytte en allerede registrert bruker til å logge seg inn. Videre blir brukeren tatt til "hovedvinduet". Her har vi valgt å presentere kartet og en liten logg-funksjon for hver sau. Under denne loggen har brukeren mulighet til å trykke på *Oppdater saueraapport* som oppdaterer loggen. Det er også mulig å navigere seg til en utvidet logg ved hjelp av en knapp kalt *Logg*. Denne loggen ligger i en egen fane. Kartet viser altså lokasjonene til saueraapportene til brukeren.

Videre har brukeren mulighet til å navigere seg til andre faner, blant annet *Legg til og rediger saueraapport* og *Rediger bonde og nødkontakt*. Her blir brukeren presentert for tekstmeldinger som gjør det mulig å redigere allerede registrert informasjon.

Fargevalget til systemet ble også bestemt med en baktanke. Vi ville framstille fargene så nøytrale som mulig slik at selve funksjonene til systemet ble i fokus, ikke designet i seg selv. Kartet introduserer også en rekke intuitive ikoner som hjelper brukeren å forstå hva som foregår. Når man har registrert en bondegård, vil det blant annet komme opp et ikon av en bondegård som viser hvor den ligger. Dette er et bevisst designvalg for å gjøre kartfunksjonen mer brukervennlig.



8 — Testing

For å hele tiden redusere sjansen for feil i systemet, har vi underveis i utviklingsprosessen gjennomført ulike type tester på programmet. Dette innebærer både testing av selve systemfunksjonene, samt testing av brukervennlighet. Kombinasjonen av slike tester øker sjansene for å ende opp med et system som fungerer som det skal, i tillegg til at det er enkelt og attraktivt for sluttbrukeren å faktisk ta det i bruk.

En viktig ting med testingen er at man ikke sparer alle testene til slutt. Dersom man kun tester når man er så å si ferdig med systemet, vil man risikere at feilene man oppdager vil kunne ha store følger for hele systemet, og det vil kunne kreve mye arbeid å rette de opp. Dette gjelder også for det grafiske designet av programmet. Derfor er det viktig å jevnlig få innspill fra kunden når det gjelder brukergrensesnittet, slik at vi underveis vet at vi er på riktig spor.

Av denne grunn har vi brukt sprint review-møtene til å få tilbakemeldinger på brukergrensesnittet i programmet, selv om dette ikke direkte kan regnes som "formelle tester". For å kontrollere at vi laget et fornuftig brukergrensesnitt fra begynnelsen av, gjennomførte vi en papirprototypetest¹ før vi begynte å implementere GUI i programmet. For å sikre at vi endte opp med et godt brukergrensesnitt til slutt, gjennomførte vi en brukbarhetstest med SUS-skjema (System Usability Scale²) på personer som ikke hadde sett noe av systemet fra før. Det er kun på denne måten vi virkelig får kontrollert at systemet er enkelt å bruke for sluttbrukeren, siden kunden har opparbeidet seg en forståelse av systemet underveis i prosessen vha. Scrum-møtene, og er derfor ikke representativ for "den vanlige brukeren". Vi hadde også en omfattende systemtest til slutt for å sikre at alle systemfunksjonene i kravspesifikasjonen fungerte som det skulle, selv om mye av dette ble testet underveis i sprintene.

¹<https://blogs.atlassian.com/2011/11/usability-testing-with-paper-prototyping/>

²<http://www.measuringusability.com/sus.php>

8.1 Papirprototypetest

For å gjennomføre papirprototypetesten, tok vi for oss noen sentrale funksjoner systemet skulle ha i form av user stories, og skisserte opp skjermbildene disse user storyene ville innebære. Vi lot så venner som ikke hadde tidligere kjennskap til systemet få gjennomføre de ulike user storyene. Når testpersonen trykker på en “knapp” på arket, tar vi frem den nye skissen av neste skjermbilde som skal dukke opp. Slik fortsetter det frem til caset er gjennomført. Testpersonen blir bedt om å tenke høyt underveis, slik at vi får et innblikk i om brukeren føler at systemet er intuitivt nok. I et GUI med optimal brukervennlighet, vil testpersonen utføre caset uten opplæring på forhånd eller forklaring underveis. Et eksempel på en slik test kan sees i vedlegget: [Papirprototypetest](#), (11.4.1)

Noen av tilbakemeldingene fra papirprototypetesten var:

1. Bør ha en “Avbryt registrering” knapp i registreringsskjema for bonde og reservekontakt.
2. Bør ha en “Logg ut” knapp i alle fanene, slik at man ikke må gå tilbake til “Kartoversikt” for å logge ut.
3. Endre fanenavn “Registrer og rediger sauer” til “Legg til og rediger sauer” så det blir lettere å lese (registrer og rediger flyter i hverandre..).
4. Bør bytte om på rekkefølgen av fanene slik at de som brukes oftest er til venstre, samtidig som at “like funksjonaliteter” er ved siden av hverandre (redigering av bonde/sau):
Bytte fra: Kartoversikt - Rediger bonde og nødkontakt - Logg - Legg til og rediger sauer
Til: Kartoversikt - Legg til og rediger sauer - Rediger bonde og nødkontakt - Logg

8.2 Systemtest

Når systemet ble sett på som ferdig, valgte vi å gjennomføre en systemtest for å sikre at alle kravene er oppfylt og at funksjonaliteten rundt kravene fungerer som det skal. Derfor opprettet vi flere test caser som vi gjennomførte og noterte oss resultatene på. Siden vi hadde testet nokså mye underveis i Scrum-prosessen, forventet vi ikke å finne mange feil. Allikevel er det viktig å sikre at sluttproduktet fungerer 100 prosent. Systemtesten kan finnes i vedlegget: [Systemtest](#), (11.4.2)

8.3 Brukbarhetstest

Etter vi hadde gjennomført systemtesten, gjennomførte vi en brukbarhetstest for å vurdere hvor godt brukergrensesnittet til systemet er. Som forklart over har vi valgt å gjennomføre denne testen ved hjelp av et SUS-skjema. Grunnen til dette er at det er en enkel og nokså komplett testmetode som dekker store deler av hovedspørsmålene ved brukbarhet.

Vi ba testdeltakerene tenke seg at de faktisk var sauebønder da de gjennomførte testen, slik at vi i størst mulig grad kunne få en test som er representativ for den faktiske sluttbrukeren. Siden vi ikke har noen ekte sauebønder å teste på, vil nok ikke testen være optimal. I tillegg har vi brukt nokså få testpersoner, og et lite utvalg er sjeldent representativt for hele brukermassen. Allikevel tror vi at vi i alle fall får et visst innblikk i systemets brukbarhet, noe som kan hjelpe oss med å bestemme om brukergrensenittet vårt bør forandres før endelig levering.

SUS-skjemaet kan sees i vedlegget: [Brukbarhetstest](#), (11.4.3)



9 — Verktøy

Her skal vi diskutere de valgene vi tok når vi benyttet oss av verktøy. Videre skal vi begrunne hvorfor vi valget akkurat disse programmene, samt fordeler og ulemper med dem. Generelt sett kan vi si at en stor faktor når vi skulle velge verktøy for prosjektet var prisen. Alle programmene vi valgte å ta i bruk var gratis, men samtidig produserte de et produkt som vi var fornøyd med. Programmene var også som regel lette å ta i bruk for flere brukere samtidig, noe som gav gode muligheter for fildeiling og samarbeid.

9.1 Prosjekthåndtering

9.1.1 Google Drive

<https://drive.google.com/>

Google Drive ble brukt som en plattform for å lagre og dele dokumenter og andre filer. En stor fordel med Google Drive er at alle medlemmene av gruppen har mulighet til å arbeide på det samme dokumentet samtidig, i tillegg til at det er en lagringsplattform som er lett tilgjengelig for alle gruppemedlemmene. Det er viktig å nevne at det kan oppstå konflikter og problemer hvis flere jobber på samme dokument, men med god planlegging ble dette en enkel og effektiv løsning.

9.1.2 Facebook

<https://www.facebook.com/>

Tidlig i oppstartfasen brukte vi email-funksjonen på «It's Learning» for å opprettholde kommunikasjonen mellom gruppemedlemmene. Vi bestemte oss deretter for å opprette en Facebookgruppe istedenfor, siden ble lettere å opprettholde jevnlig og fortløpende kontakt med hverandre. Facebook ble dermed vårt hovedverktøy for kommunikasjon og benyttet det til blant annet diskusjon, avtale møter og informasjonsdeling. Gruppemedlemmene var også jevnlig brukere av Facebook og det ble derfor et naturlig steg å benytte Facebook som kommunikasjonsplattform.

9.1.3 ShareLaTeX

<https://www.sharelatex.com/>

LaTeX er et verktøy som lar brukeren fokusere på innholdet i dokumentet og ikke designet. Tanken bak LaTeX er at forfatteren av dokumentet skal slippe å bruke mye tid på designet, noe som forhindrer dårlig design og gir mer tid til selve innholdet i rapporten. Gruppen bestemte seg for å ta i bruk dette verktøyet for å både spare tid og få en bedre sluttrapport.

Vi har valgt å benytte oss av det nettbaserte verktøyet <https://www.sharelatex.com/> til å skrive rapporten vår. Dette er fordi flere kan arbeide på samme dokument fra hvilket som helst maskin, noe som gjør rapportskrivingen mye enklere og mer effektiv.

En bakdel med LaTeX er at det ofte genereres mange blanke felter i rapporten slik at formateringen blir riktig. Dette fører til at rapporten tar flere sider enn den ellers ville ha gjort. Av denne grunn har rapporten vår litt for mange sider enn det som ble satt som maksimalt. Siden vi fikk beskjed om at dette var greit dersom vi begrunnet det, har vi ikke tatt noen grep for å fikse dette.

9.1.4 Rally dev

<https://rally1.rallydev.com/>

Rally dev var et program som ble brukt til å holde oversikt over produktbackloggen, sprintbackloggene og progresjonen på arbeidsoppgavene som ble delegert til de forskjellige gruppemedlemmene. Man kunne også dele user storyene i backloggen inn i mindre oppgaver, noe som gjorde det lettere å estimere tid, samt holde styr på nøyaktig hva som måtte gjøres (se vedlegg 11.15). Programmet satte opp en oversiktlig grafisk framstilling av sprintene og backloggen, samt muligheten til å delegere oppgaver til forskjellige personene. Deretter kunne man rapportere inn progresjonen på de forskjellige arbeidsoppgavene og markere dem som enten ferdig, under arbeid eller uferdig. På grunn av alle disse mulighetene, valgte vi å benytte oss av netttopp dette verktøyet. I tillegg var det gratis for inntil 10 samarbeidspartnere, så det passet godt for oss.

9.1.5 GitHub

<https://github.com/>

GitHub er en versjonskontroll for softwareutvikling. GitHub ble brukt som et verktøy for å dele programmeringskoden mellom gruppemedlenne. Hvert gruppemedlem kunne logge seg inn for å hente eller laste opp filer, som igjen kunne ble benyttet av andre gruppemedlemmer. Grunnen til at vi brukte GitHub som versjonskontroll, var at vi hadde erfaring med det fra før, samt at det er lett å koble opp mot GitHub via NetBeans.

9.2 Diagramverktøy

9.2.1 WBStool

<http://www.wbstool.com/>

WBStool er et nettbasert program for å designe WBS (Work Breakdown Structures), WBS dia-

grammer og andre typer for hierarki. Programmet ble hovedsaklig brukt til å lage et rammeverk for prosjektet ved hjelp av et organogram konstruksjon. Vi valgte å bruke dette verktøyet, siden det var gratis, enkelt å bruke, samt lett tilgjengelig.

9.2.2 GanttProject

<http://www.ganttproject.biz/>

GanttProject er et desktop program for prosjekt administrering og planlegging. I programmet kan man opprette et Gantt-diagramm som tar for seg organogrammet brukt i WBStools. Forskjellen mellom WBS diagrammet og Gantt diagrammet er at Gantt tar for seg hvordan arbeidet for WBS organogrammet er delt opp og viser når oppgavene skal gjøres. Vi valgte å benytte GanttProject siden det var enkelt å legge inn alle elementene fra WBS-en, samt oversiktlig når det gjaldt å se fremgangen i arbeidet.

9.2.3 Websequencediagrams

<https://www.websequencediagrams.com/>

For å lage sekvensdiagrammer valgte vi benytte <https://www.websequencediagrams.com/>. Grunnen var at dette verktøyet genererer all grafikken for deg. Alt du trenger å gjøre er å skrive inn enkle setninger for å vise programflyten i diagrammet. I tillegg er den webbasert, så man kan arbeide på diagrammene fra hvor som helst.

9.2.4 Gliffy

<http://www.gliffy.com/>

Gliffy er en webbasert applikasjonsside for å opprette diagrammer. Programmet tar i bruk ferdig-stilte figurer som blir dratt til en arbeidsskerm og deretter fylt ut med informasjon. Hovedsaklig ble programmet brukt for å planlegge strukturen i programmet (variabler, metoder, etc.) for de forskjellige metodene/funksjonene. En stor fordel med programmet var at det var mulig å opprette en gruppe og deretter opprette filer som var delt mellom gruppemedlemmene, noe som var mye av grunnen til at vi valgte å bruke Gliffy.

9.3 Implementering

9.3.1 NetBeans

<https://netbeans.org/>

På tross av at de fleste i gruppen hadde mest erfaring med Eclipse¹ som utviklingsverktøy valgte vi å benytte NetBeans. Dette var hovedsakelig på grunn av GUI-egenskapene som følger med NetBeans. I motsetning til Eclipse har NetBeans mulighet til å plassere GUI elementer i arbeidsområdet ved hjelp av en «drag and drop» funksjon, noe som ville ha vært mer tidkrevende i et program som Eclipse. Det skal likevel nevnes at dette også er mulig i Eclipse ved hjelp av plugins som kan lastes ned og installeres, men NetBeans sin løsning liker vi bedre. NetBeans

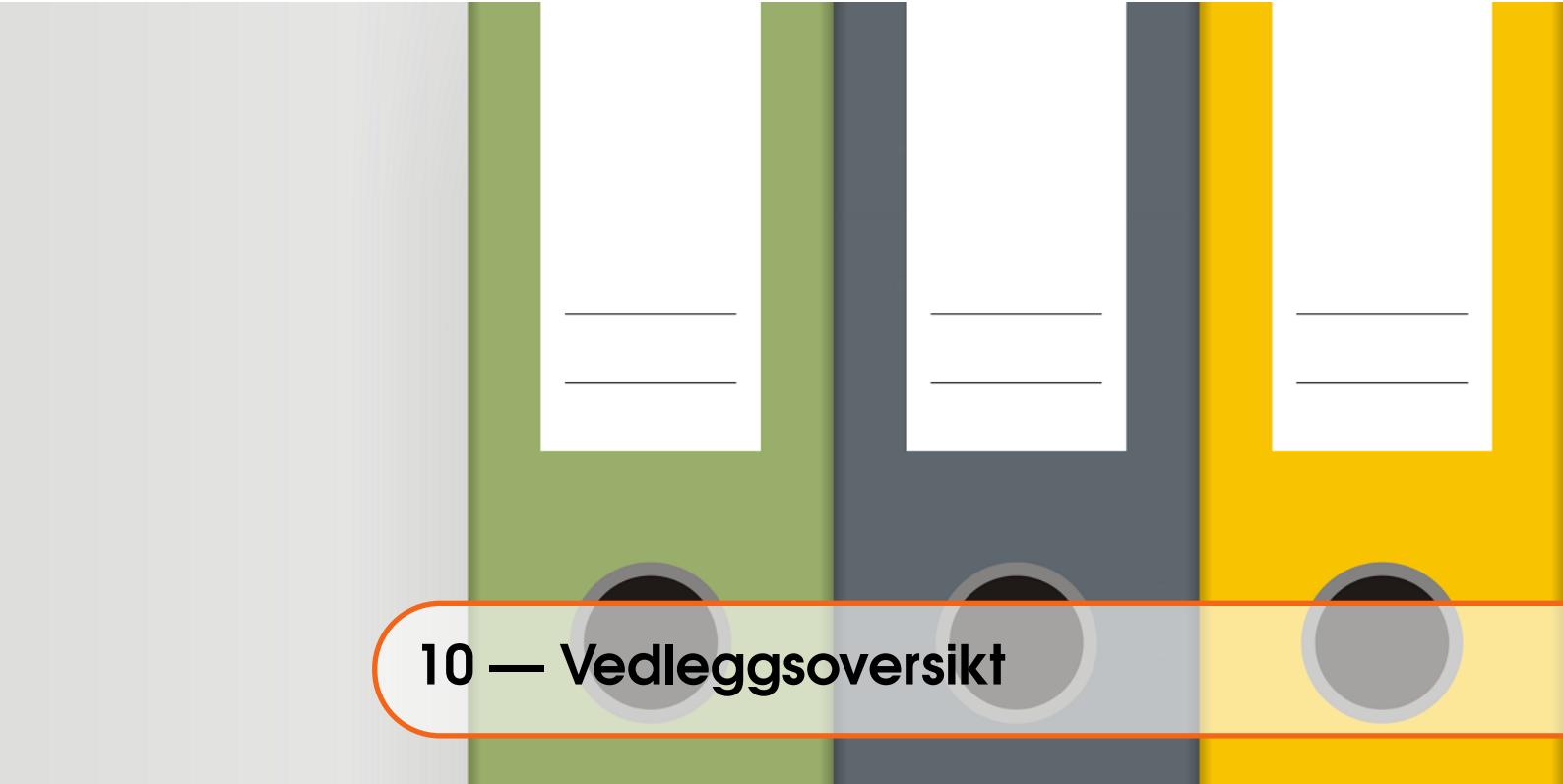
¹<http://www.eclipse.org/>

har også på mange måter de samme funksjonene som Eclipse, så overgangen skapte ikke noen problemer.

9.3.2 MySQL

<http://www.mysql.com/>

MySQL er et meget utbredt databasesystem som det eksisterer mye god dokumentasjon på. Dette, i tillegg til at det er lett å sette opp mot Java-applikasjoner, er de viktigste grunnene til at vi valgte å bruke dette systemet. Vi har skrevet mer om MySQL under seksjonen [Databaseteknologi](#).



10 — Vedleggsoversikt

Under er det listet opp en oversikt over alle vedleggene, samt alle programfilene som følger med prosjektoppgaven.

10.1 Dokumentvedlegg

1. Diagrammer
2. Kravspesifikasjon
3. Sprinter
4. Testing
5. Brukermanual

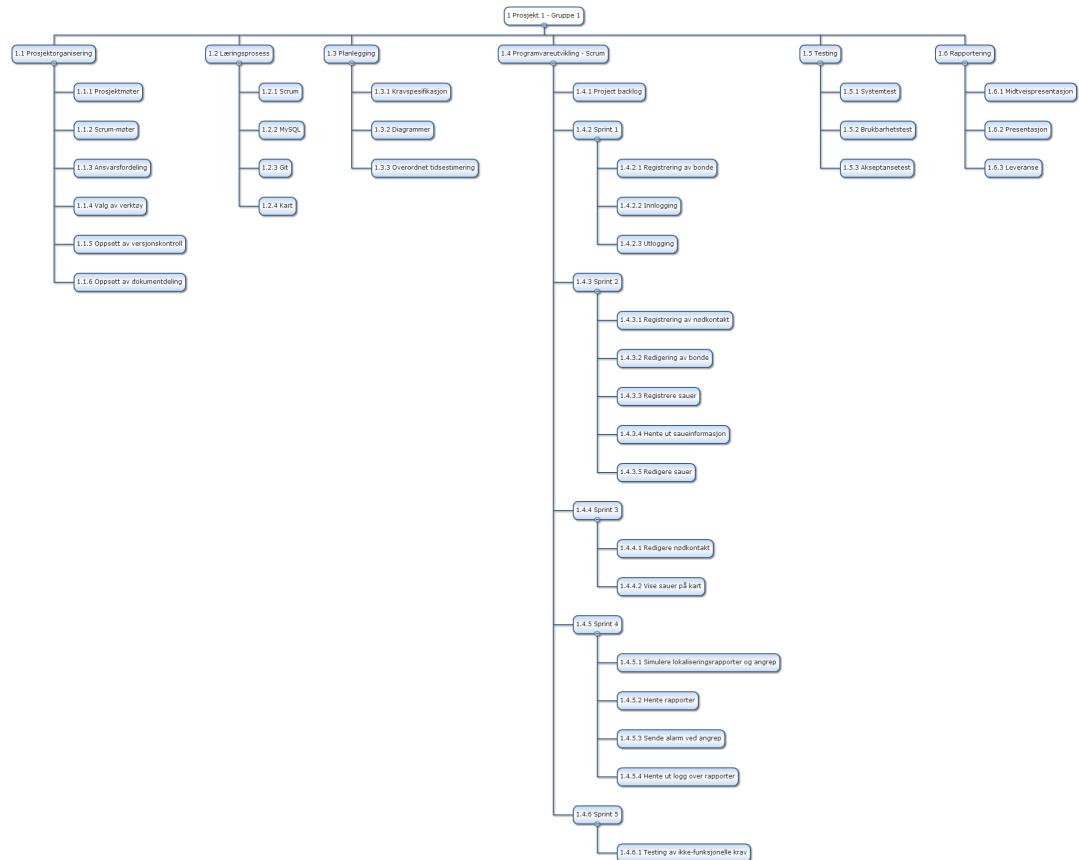
10.2 Programvedlegg

- 'Gruppe1_Kildekode'
- 'Gruppe1_JavaDoc'
- 'Gruppe1_JAR_Filer' (inkluderer mappe med SheepWatchUI.jar og Simulator.jar)

11 – Vedlegg

11.1 Diagrammer

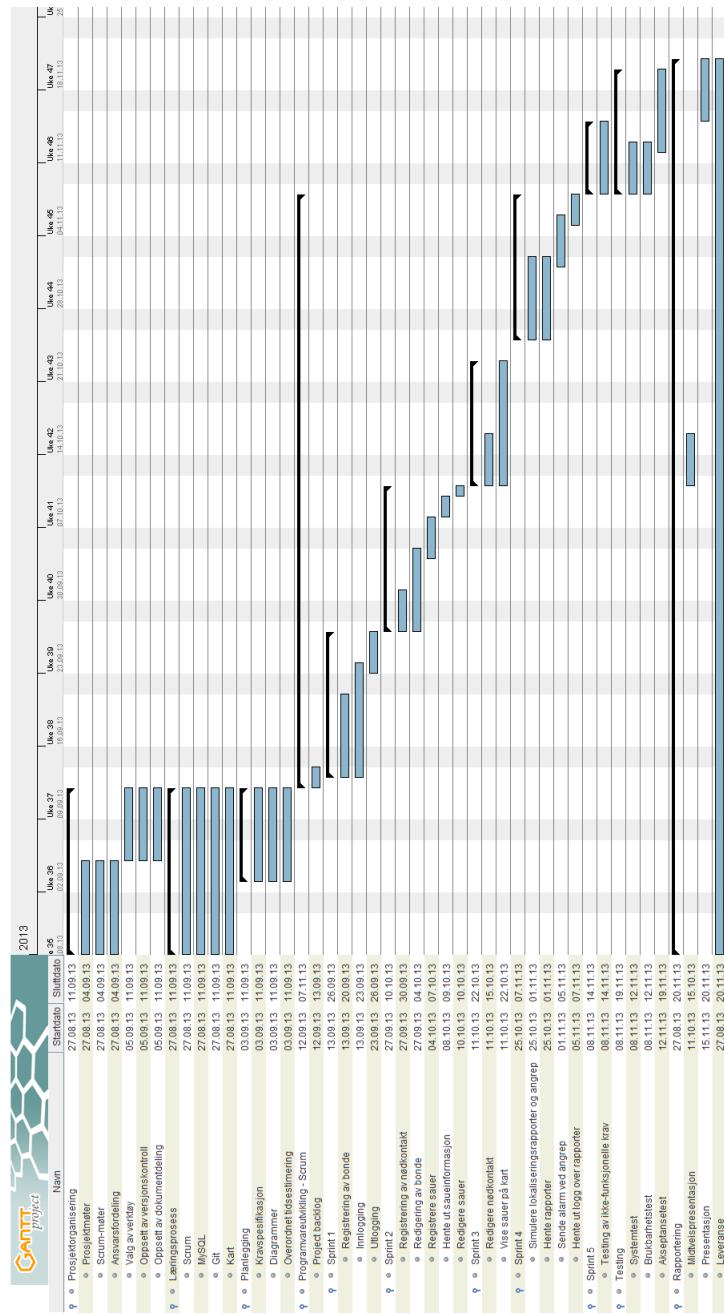
11.1.1 Work Breakdown Structure



www.ritetot.com

Figur 11.1: WBS-diagram

11.1.2 Gantt-diagram



Figur 11.2: Gantt-diagram

11.1.3 Timeliste

	Dato	Kristoffer	Thomas	Iver	Vegard	Lars	SUM
Sprint 1	13/09/13	2	2	2	2	2	10
	14/09/13						0
	15/09/13						0
	16/09/13	2					2
	17/09/13	2	2	2	2	2	10
	18/09/13	1			1	1	3
	19/09/13			4			4
	20/09/13	2	2	2	2	2	10
	21/09/13						0
	22/09/13	2			3	3	8
	23/09/13	1	2	3	1		7
	24/09/13	2	2	2	2	2	10
	25/09/13	2	4	2	2	2	12
	26/09/13	3	2	2	2	2	11
Sum sprint 1		19	16	19	17	16	87
Sprint 2	27/09/13	3	2	2	3	2	12
	28/09/13						0
	29/09/13						0
	30/09/13	2	2	2	2	2	10
	01/10/13	5	3	3	4	3	18
	02/10/13						0
	03/10/13						0
	04/10/13	2	2	2	2	2	10
	05/10/13						0
	06/10/13						0
	07/10/13	2	2	2	3	3	12
	08/10/13	4	2	2	2	2	12
	09/10/13		3	3			6
	10/10/13	2				3	5
Sum sprint 2		20	16	16	16	17	85

Figur 11.3: Timeliste - sprint 1-2

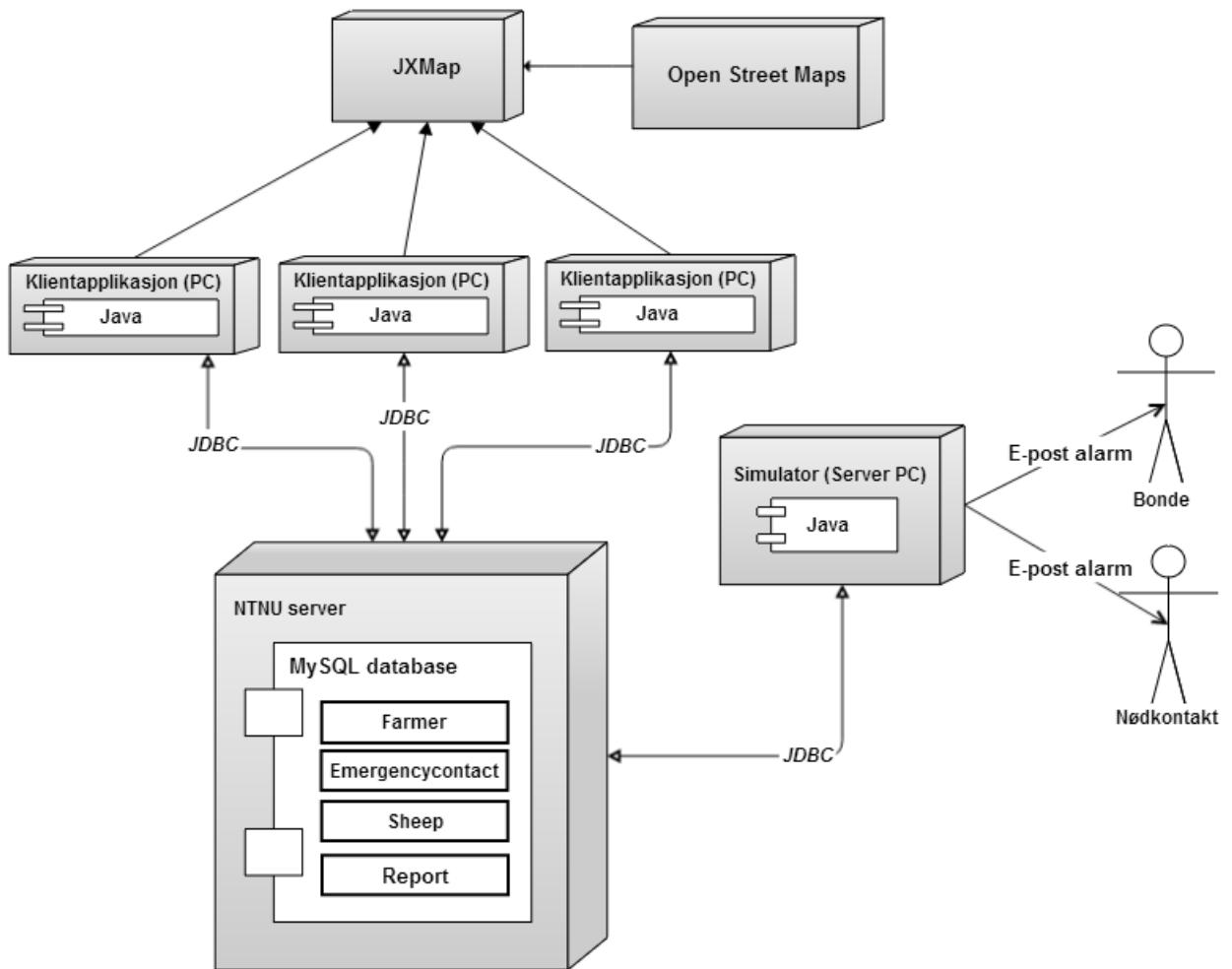
Sprint 3	11/10/13	3	2	2	2	2	11
	12/10/13						0
	13/10/13						0
	14/10/13	3	1	4	2	5	15
	15/10/13	2	2	2	2	2	10
	16/10/13						0
	17/10/13						0
	18/10/13	3	3	3	3	3	15
	19/10/13						0
	20/10/13						0
	21/10/13	2	2	2	2	2	10
	22/10/13	2	2	2	2	2	10
	23/10/13						0
	24/10/13	3	5	3	4	2	17
Sum sprint 3		18	17	18	17	18	88
Sprint 4	25/10/13	2	2	2	2	2	10
	26/10/13						0
	27/10/13	1	2	1	1	2	7
	28/10/13						0
	29/10/13	3	2	4	2	2	13
	30/10/13	2	2	2	2	2	10
	31/10/13	3	1	1	2	1	8
	01/11/13	3	3	3	3	3	15
	02/11/13						0
	03/11/13						0
	04/11/13	1	2	1	1	1	6
	05/11/13	2	2	2	2	2	10
	06/11/13	2	3	1	2	3	11
	07/11/13	3	3	3	3	3	15
Sum sprint 4		22	22	20	20	21	105
Sprint 5	08/11/13	2	2	2	2	2	10
	09/11/13						0
	10/11/13	1	2	1	1	2	7
	11/11/13	3	3	3	4	3	16
	12/11/13	2	2	1	1	1	7
	13/11/13	1		2	1	1	5
	14/11/13	1		1			2
Sum sprint 5		10	9	10	9	9	47
	TOTAL SUM	89	80	83	79	81	824

Figur 11.4: Timeliste - sprint 3-5

11.1.4 Risikomatrise

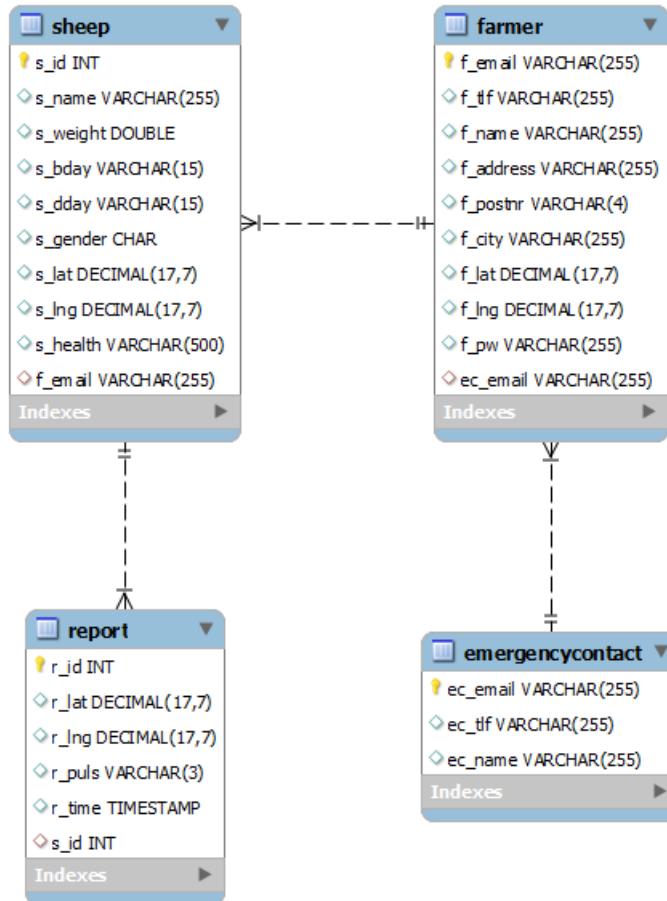
Beskrivelse	Sannsynlighet (1-9)	Konsekvens (1-9)	Viktighet (Sannsynlighet * Konsekvens)	Preventive tiltak	Utbedringstiltak
Et gruppedellem er borte i en lengre tidsperiode	4	8	32	Si i fra i god tid dersom man planlegger å være borte i en periode, som f.eks. på ferie eller hjemmesøk.	Tildel oppgaver som det aktuelle gruppedellemmet skal gjøre mens han er borte.
En sprint deadline er ikke møtt	4	6	24	Planlegg realiserbare sprinter, og beregn ekstra tid til eventuelle problemer som kan oppstå.	Jobb ekstra timer for å ta igjen det tapte.
Databaseserveren er nede	3	8	24	Bruk en databaseserver som er stabil.	Kontakt serveransvarlig for å få serveren online.
Tap av data	3	7	21	Bruk versjonskontroll. Lagre også kopier av koden lokalt.	Spør de andre gruppedellemmene om de har dataene lagret lokalt. Hvis ikke må antall arbeidstimer økes for å ta igjen det tapte.
Gruppa står fast med et problem som er vanskelig å fikse	4	5	20	Prøv å finne løsninger som er gjennomførbare.	Spør om hjelp fra studass, eventuelt prøv en alternativ løsning.
Et gruppedellem er syk	8	2	16	Alle skal gjøre sitt beste for å holde seg friske!	Dersom gruppedellemmet ikke er i stand til å jobbe hjemme, må vi fordele hans arbeidsoppgaver på resten av gruppa slik at vi ikke blir stående fast.
Ujevn arbeidsfordeling	4	4	16	Prøv å fordele omrent like mange og like store oppgaver til alle.	Om noen får mer å gjøre enn andre, tar de andre det igjen senere.
Ett eller flere gruppedellemmer kommer for sent til et møte	7	2	14	Alle skal ha respekt for avtalte møtetider. Bruk faste tidspunkter hver uke, slik at man ikke glemmer det så lett.	Gruppedellemmet må sitte lenger for å jobbe igjen tida han kom for sent.
Ett eller flere gruppedellemmer gjør ikke sine tildelte arbeidsoppgaver	2	5	10	Alltid ha en åpen dialog og bruk et verktøy som gir en oversikt over hva alle gjør til en hver tid.	Gruppedellemmet må jobbe ekstra for å ta igjen det tapte. Andre gruppedellemmer hjelper til dersom forsinkelsen sørger for at andre arbeidsoppgaver må vente pga. avhengigheter.

Figur 11.5: Risikomatrise

11.1.5 Deployment diagram**Deployment Diagram**

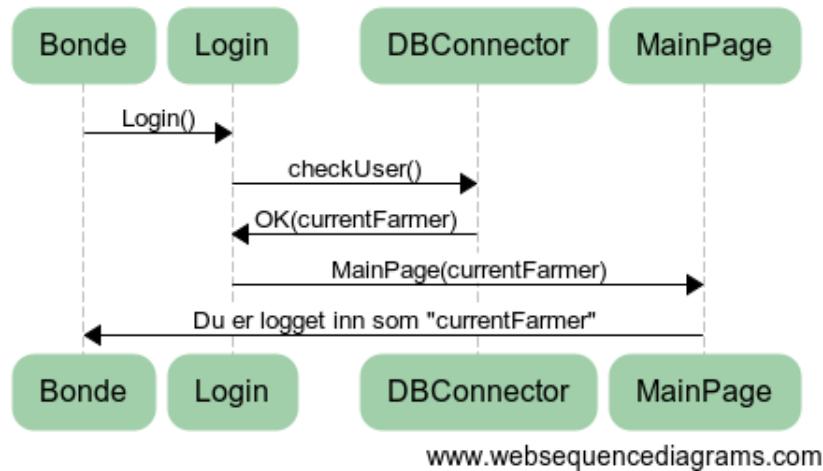
Figur 11.6: Deployment diagram

11.1.6 Entity Relationship-diagram

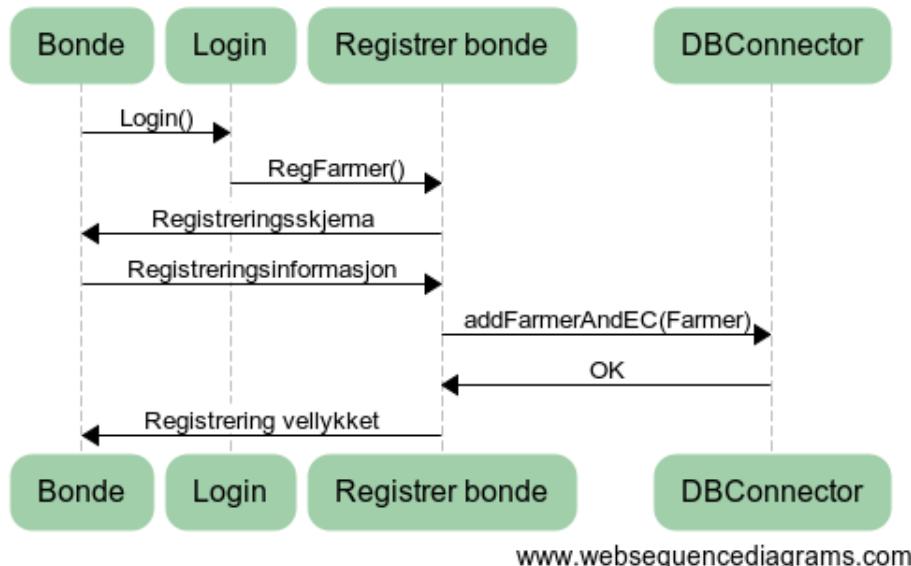


Figur 11.7: ER-diagram

11.1.7 Sekvensdiagrammer

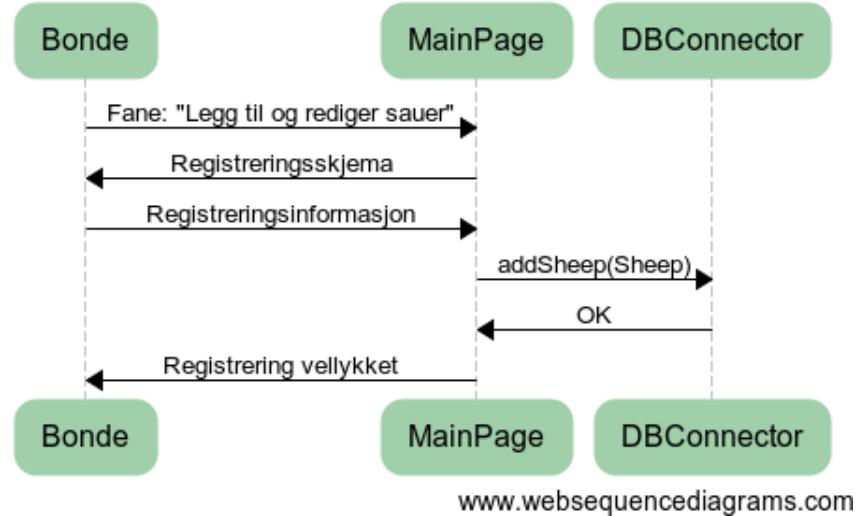


Figur 11.8: Login

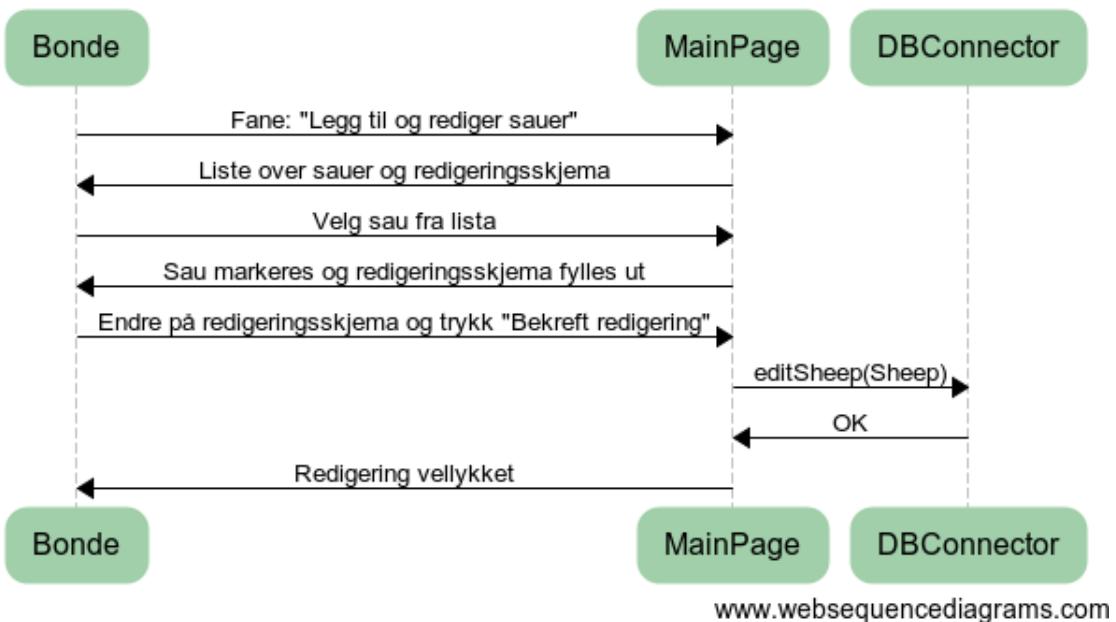


Figur 11.9: Registrer bonde

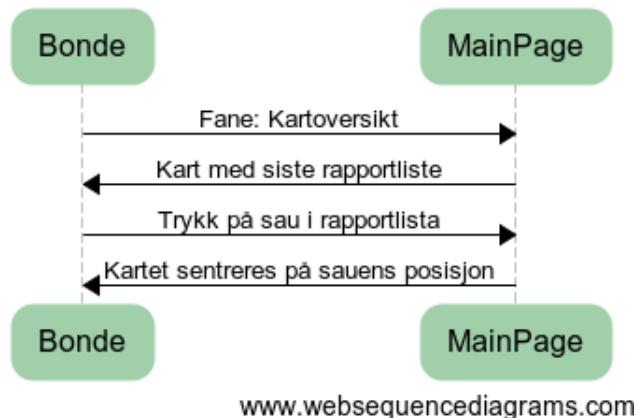
11.1.8 Rally-tasks eksempel



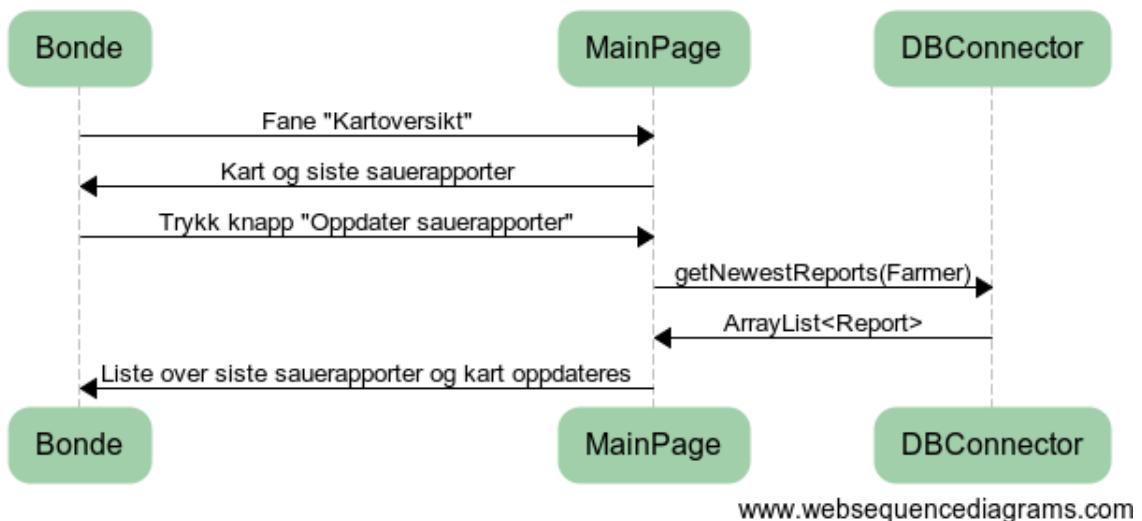
Figur 11.10: Registrer sau



Figur 11.11: Rediger sau



Figur 11.12: Sjekk saueposisjon



Figur 11.13: Oppdater rapporter



Figur 11.14: Sjekk logg

The screenshot displays two main sections: a 'Tasks' table on the left and a list of 'Accepted' user stories on the right.

Tasks:

ID	NAME	OWNER	STATE	ESTIMATE (H)	TO DO (H)	BLOCKED
TA1	Opprett tabell for bønder i databasen.		Completed	5.00	0.00	0
TA2	Lag objektklasse Bonde i systemet.		Completed	5.00	0.00	0
TA3	Lag metode i databaseklassen for å registrere en bonde.		Completed	5.00	0.00	0
TA4	Lag GUI skjema for å registrere informasjon om en bonde.		Completed	5.00	0.00	0

1 - 4 of 4

Accepted User Stories:

- US3** kristoffer.aulie Som en bonde vil jeg kunne registrere meg som bonde i systemet. 53 days 9
- US5** kristoffer.aulie Som en bonde vil jeg kunne logge meg inn i systemet med brukernavn og passord. 53 days 8
- US6** kristoffer.aulie Som en bonde vil jeg kunne logge meg ut av systemet. 53 days 8

Figur 11.15: Tasks-inndeling for User Story 1

11.2 Kravspesifikasjon

11.2.1 User stories

I dette prosjektet har vi valgt å benytte oss av User Stories for å beskrive de funksjonelle kravene til systemet. Dette er fordi det gir en god indikasjon på hva slags funksjonalitet brukeren faktisk ønsker, siden det er formulert fra brukerens synsvinkel. Dette gjør det også lettere for stakeholders å forstå hva kravene sier.

I tillegg er User Stories ofte brukt som en naturlig del av Scrum, der de utgjør elementene i Product Backlog, samt Sprint Backlog. De funksjonelle og ikke-funksjonelle kravene under er derfor også en oversikt over vår Product Backlog.

Underveis i prosjektet endret vi litt på noen av de originale kravene fra oppgaveteksten etter å ha pratet med studass. For eksempel kravene om at sauene skal rapportere kun tre ganger i døgnet, og at bonden kun skal hente nye rapporter to ganger i døgnet. Vi ble enige om at dette var litt for sjeldent, så vi endret på de kravene slik at oppdateringene kan skje litt oftere. Slik vi har tolket det sammen med studass oppdaterer sauene sin posisjon og helsetilstand hvert halvtime, mens bonden henter nye sauerapporter automatisk hver time. Under oversikten over de funksjonelle og ikke-funksjonelle kravene, har vi laget en [oversikt](#) over hvordan vi har tolket hvert enkelt krav fra oppgaveteksten.

User Stories følger et bestemt setningsformat, og i vårt tilfelle benytter vi oss av følgende format: "Som en <rolle> vil jeg <gjøremål>." Eks: "Som en bruker vil jeg kunne registrere meg i systemet."

Funksjonelle krav

De funksjonelle kravene sier noe om hva systemet skal gjøre. Under har vi sortert de funksjonelle kravene etter synkende prioritet. De kravene som vi ser på som mest essensielle for systemet er prioritert øverst.

Tabell 11.1: Funksjonelle krav

Nr	User Story	Prioritet (1-9)
1	Som en bonde vil jeg kunne registrere meg som bonde i systemet.	9
2	Som en bonde vil jeg kunne registrere sauene i systemet.	9
3	Som en bonde vil jeg kunne logge inn i systemet med brukernavn og passord.	8
4	Som en bonde vil jeg kunne logge ut av systemet.	8
5	Som en bonde vil jeg kunne redigere en registrert bonde.	8
6	Som en bonde vil jeg kunne redigere registrerte sauene.	8
7	Som en bonde vil jeg kunne registrere en “reserve-kontaktperson” i systemet.	7
8	Som en bonde vil jeg kunne redigere en registrert “reserve-kontaktperson”.	6
9	Som en bonde vil jeg kunne hente ut informasjon om sauene fra systemet.	6
10	Som en bonde vil jeg kunne sjekke på kart hvor en sau sist var registrert.	6
11	Som en bonde vil jeg kunne få alarm til meg selv og “reserve-kontaktperson” via e-post, sms eller telefon om en sau blir angrepet.	6
12	Som en bonde vil jeg kunne få lokaliseringsdata rapportert fra hver sau hver halvtime.	5
13	Som en bonde vil jeg kunne forespørre lokaliseringsinformasjon om mine sauene hver time.	5
14	Som en bonde vil jeg kunne hente ut en logg med informasjon om tidligere angrep og lokasjoner.	4

Ikke-funksjonelle krav

Ikke-funksjonelle krav sier noe om hvordan systemet skal være. Disse har vi valgt å skrive som normale krav, siden brukerne ikke er direkte involvert i disse kravene.

Etter videre informasjon fra fagstaben har vi sett bort i fra enkelte krav, slik som “Systemet skal implementeres som et klient/tjenersystem”. Siden vi ønsker at kravspesifikasjonen kun skal inneholde de kravene som faktisk skal implementeres, har vi ikke inkludert dette kravet i oversikten under.

Under har vi sortert de ikke-funksjonelle kravene etter synkende prioritet. De kravene som vi ser på som mest essensielle for systemet er prioritert øverst.

Tabell 11.2: Ikke-funksjonelle krav

Nr	Ikke-funksjonelt krav	Prioritet (1-9)
15	Systemet skal være operativt døgnkontinuelig.	9
16	Systemet skal være tilgjengelig minimum 95% av tiden.	8
17	Systemet skal kunne håndtere 200 bønder samtidig.	7
18	Systemet skal kunne håndtere 10 000 sauar.	7
19	Innkommande alarm fra sau, skal gis prioritet.	7
20	Ved systemfeil kan systemet være utilgjengelig i maksimalt 3 timer.	6
21	Responstid på forespørslar fra bønder skal være maksimalt 2 sekunder.	5
22	Innleggelse av lokaliseringinformasjon om en sau skal ta maksimalt 0.5 sekunder.	4

Tolkningsoversikt

Her har vi laget en oversikt over de ulike kravene fra oppgaveteksten og en link til hvordan vi har tolket hvert av kravene. Flere av kravene fra oppgaveteksten har vi bakt inn i samme user story, i tillegg til at vi har utvidet enkelte krav ut fra det som var naturlig å gjøre. Et eksempel er at vi ut av kravet 'Bonden logger seg på systemet med brukernavn og passord' har laget krav om at man skal kunne registrere og redigere både bonde og nødkontakt, samt logge ut av systemet.

Tabell 11.3: Tolkning av funksjonelle krav

Krav fra oppgavetekst	Tolkning
Bonden logger seg på systemet med brukernavn og passord.	1, 3, 4, 5, 7, 8
Det skal lagres helseinformasjon, identifikasjon til sauene (navn, alder o.l.), geografisk lokasjon o.s.v.	2
En bonde skal kunne registrere sine sauene med id, vekt o.l. Dersom en sau er registrert fra før skal det kunne gå an å slå opp i systemet og redigere informasjonen.	2, 6
Bonden skal kunne sjekke på et kart hvor sauene sist var registrert.	9, 10
Alarmmeldinger sendes bonden og en han har valgt ut (reserve) i tillegg.	11
Alarmmelding sendes ut som epost, SMS, og mobiltelefon forsøkes ringt opp.	11
Bonden skal få beskjed om en sau er under angrep.	11
Input om lokalisering av de enkelte sauene kommer inn normalfordelt over et døgn.	12
Hver sau rapporterer sin lokalisering 3 ganger per døgn.	12
Hver bonde forespør lokaliseringinformasjonen om sine sauene 2 ganger i døgnet.	13
Bonden skal kunne gå tilbake i tid og se på logger.	14

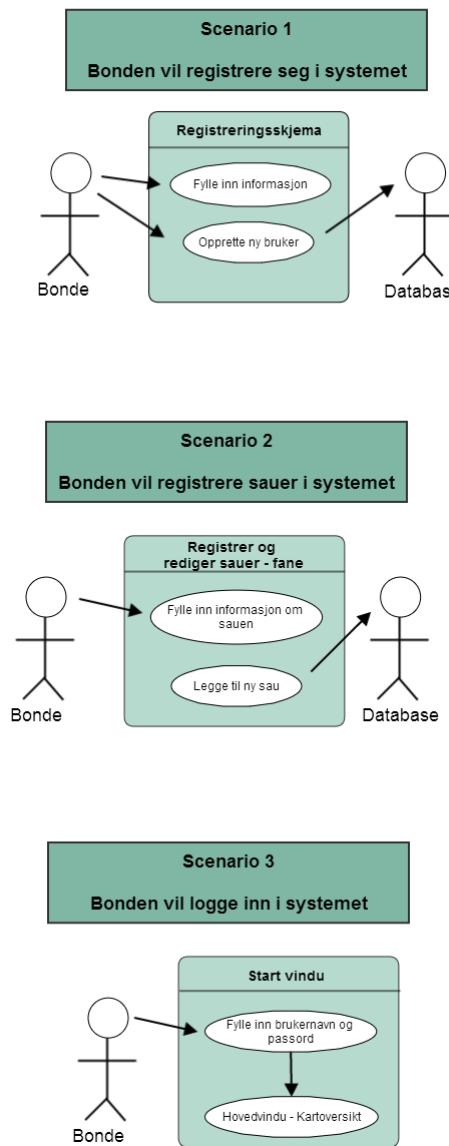
Tabell 11.4: Tolkning av ikke-funksjonelle krav

Krav fra oppgavetekst	Tolkning
Systemet skal være operativt døgnkontinuerlig.	15
Systemet skal ha en tilgjengelighet på minimum 95 % av tiden.	16
Maksimalt 200 bønder skal samtidig kunne bruke systemet.	17
Maksimalt 10000 sauene kan håndteres av systemet.	18
Innkommende alarm fra en sau gis prioritet.	19
Ved systemfeil tillates systemet å være utilgjengelig i inntil 3 timer.	20
Responstid på forespørsler fra bønder skal maksimalt ta 2 sekunder.	21
Responstid på innleggelse av lokaliseringinformasjonen om en sau skal maksimalt ta 0,5 sekund.	22
Systemet implementeres som klient / tjenersystem.	Krav utgår (begrunnelse)

11.2.2 Use case

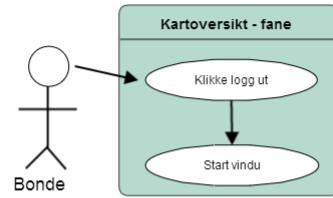
Basert på de funksjonelle kravene har vi laget use case diagrammer, samt tekstlige use cases for flere ulike scenarier. Scenariene er rettet mot de user storyene med samme nummer, men enkelte diagrammer overlapper flere user stories. Et eksempel på det er Scenario 6 som også omhandler User Story 9.

Use case diagrammer

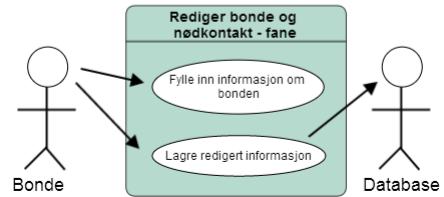


Figur 11.16: Use Case Diagram

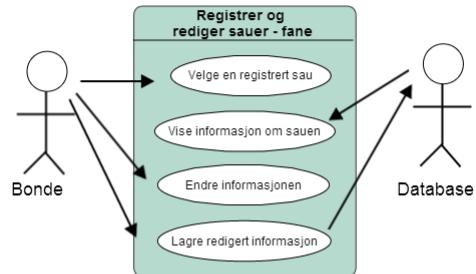
Scenario 4
Bonden vil logge ut av systemet



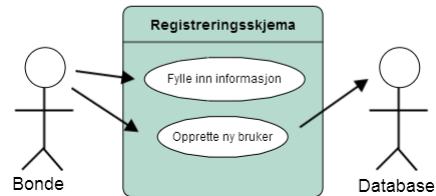
Scenario 5
Bonden vil redigere en registrert bonde



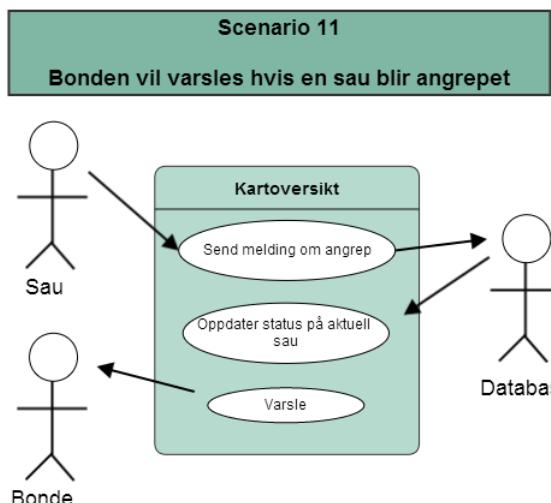
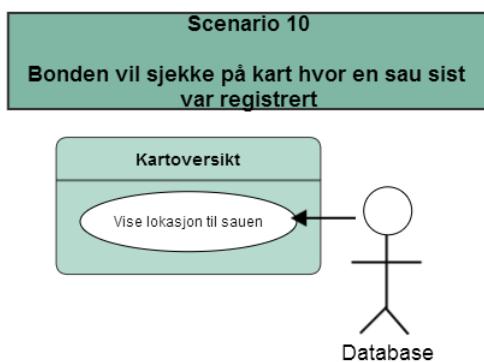
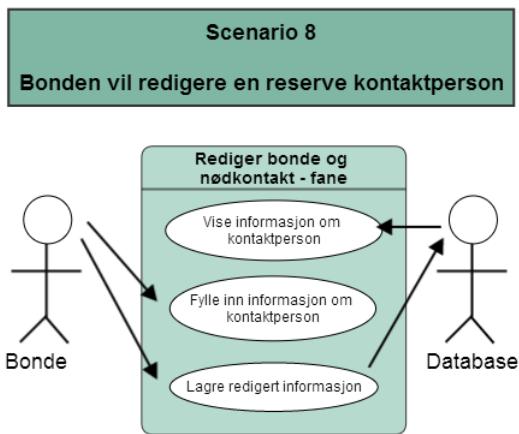
Scenario 6
Bonden vil redigere en registrert sau



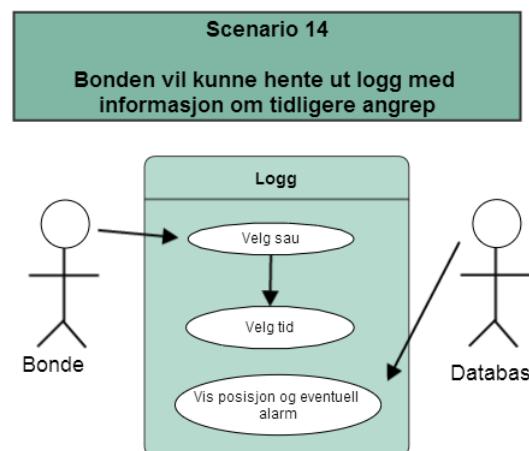
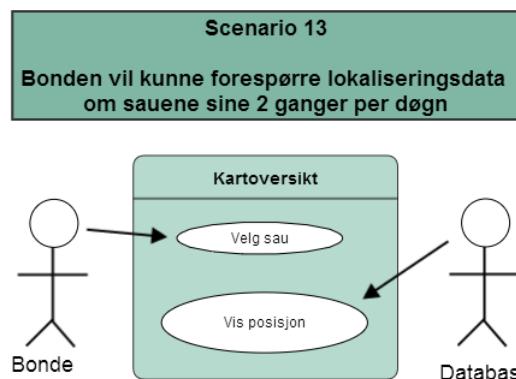
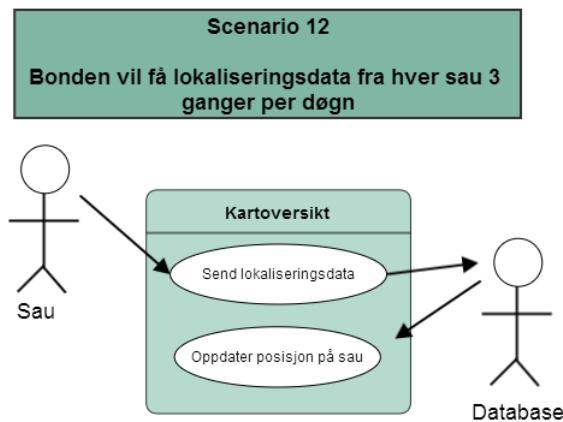
Scenario 7
Bonden vil registrere en reserve kontaktperson



Figur 11.17: Use Case Diagram



Figur 11.18: Use Case Diagram



Figur 11.19: Use Case Diagram

Tekstlige use cases

De tekstlige use casene under refererer til de funksjonelle kravene, der use case nummeret og kravnummeret er ekvivalente. User story 12 og 13 har ingen interaksjon med brukeren og vi mente det var unaturlig å skrive tesktlige use cases for dem.

Tabell 11.5: Use case 1

Use case 1	Som en bonde vil jeg kunne registrere meg som bonde i systemet.
Aktør:	Bonde
Prebetingelser:	1. Bonden må ha navigert seg til riktig vindu for å registrere en ny bruker.
Postbetingelser:	1. Ny bruker opprettet for den gitte bonden.
Normal hendelsesflyt:	1. Fyller ut nødvendig informasjon 2. Bekreft registrering
Variasjoner:	1. Bonden fyller ut den nødvendig informasjon. 2. Bekrefter registreringen. 3. Bonden har fylt ut ugyldig informasjon. 4. Feilmelding ber bonden fylle ut korrekt informasjon.
	1. Bonden fyller ut den nødvendig informasjon.
	2. Bonden avbryter registrering.

Tabell 11.6: Use case 2

Use case 2	Som en bonde vil jeg kunne registrere sauene i systemet.
Aktør:	Bonde
Prebetingelser:	1. Bonden må være logget inn i systemet. 2. Bonden må navigere seg til riktig vindu for å registrere sauene.
Postbetingelser:	1. Sauen er lagt til i databasen og vises på nettsiden.
Normal hendelsesflyt:	1. Fyller ut nødvendig informasjon om sauene. 2. Bekreft registrering
Variasjoner:	1. Bonden fyller ut ugyldig informasjon. 2. Feilmelding ber bonden fylle ut gyldig informasjon.

Tabell 11.7: Use case 3

Use case 3	Som en bonde vil jeg kunne logge inn i systemet med brukernavn og passord.
Aktør:	Bonde
Prebetingelser:	1. Bonden må ha mulighet til å skrive inn brukernavn og passord. 2. Bonden må være registrert i databasen.
Postbetingelser:	1. Bonden skal være innlogget i systemet.
Normal hendelsesflyt:	1. Bonden starter programmet. 2. Bonden skriver inn brukernavn og passord.
Variasjoner:	1. Bonden eksisterer ikke i databasen. 2. Feilmelding viser at bonden har skrevet inn feil brukernavn/passord.
Variasjoner:	1. Bonden skriver inn et ugyldig brukernavn/passord. 2. Feilmelding viser at bonden har skrevet inn feil brukernavn/passord.

Tabell 11.8: Use case 4

Use case 4	Som en bonde vil jeg kunne logge ut av systemet.
Aktør:	Bonde
Prebetingelser:	1. Bonden må være logget inn i systemet.
Postbetingelser:	1. Bonden trykket på "logg ut" knappen. 2. Bonden blir logget ut av systemet.

Tabell 11.9: Use case 5

Use case 5	Som en bonde vil jeg kunne redigere en registrert bonde.
Aktør:	Bonde
Prebetingelser:	1. Bonden må være logget inn i systemet. 2. Bonden må ha navigert seg til vinduet for redigering av profil.
Postbetingelser:	1. Endringene må være lagret i databasen.
Normal hendelsesflyt:	1. Bonden redigerer ønsket informasjon. 2. Bonden bekrefter redigeringen.
Variasjoner:	1. Bonden fyller ut ugyldig informasjon. 2. Feilmelding ber bonden fylle ut gyldig informasjon.

Tabell 11.10: Use case 6

Use case 6	Som en bonde vil jeg kunne redigere registrerte sauier.
Aktør:	Bonde
Prebetingelser:	1. Bonden må være logget inn i systemet. 2. Det eksisterer minst en registrert sau i databasen. 3. Bonden må ha navigert seg til vinduet for redigering av sauier.
Postbetingelser:	1. Den redigerte informasjonen om sauen blir lagret.
Normal hendelsesflyt:	1. Bonden redigerer ønskelig informasjon. 2. Bonden bekrefter endringene.
Variasjoner:	1. Bonden fyller ut ugyldig informasjon. 2. Feilmelding ber bonden fylle ut gyldig informasjon.

Tabell 11.11: Use case 7

Use case 7	Som en bonde vil jeg kunne registrere en reserve-kontaktperson"i systemet.
Aktør:	Bonde
Prebetingelser:	1. Bonden må ha navigert seg til vinduet for registrering av profil og kontaktperson.
Postbetingelser:	1. Bonden er opprettet i databasen med en reserve-kontaktperson.
Normal hendelsesflyt:	1.Bonden fyller ut den nødvendige informasjonen. 2. Bonden bekrefter registreringen.
Variasjoner:	1. Bonden fyller ut ugyldig informasjon. 2. Feilmelding ber bonden fylle ut gyldig informasjon.
	1. Bonden fyller ut nødvendig informasjon. 2. Bonden avbryter registrering.

Tabell 11.12: Use case 8

Use case 8	Som en bonde vil jeg kunne redigere en registrert reserve-kontaktperson".
Aktør:	Bonde
Prebetingelser:	1. Bonden må være innlogget i systemet.
	2. Bonden må ha navigert seg til vinduet for registrering av profil og kontaktperson.
Postbetingelser:	1. Endringen til reserve-kontaktpersonen blir lagret i databasen.
Normal hendelsesflyt:	1.Bonden redigerer på ønskelig informasjon. 2. Boden bekrefter endringene.
Variasjoner:	1. Bonden fyller ut ugyldig informasjon. 2. Bonden bekrefter endringene. 3. Feilmelding viser at boden har oppgitt ugyldig informasjon.

Tabell 11.13: Use case 9

Use case 9	Som en bonde vil jeg kunne hente ut informasjon om sauene fra systemet (vekt, alder, id, puls, etc.).
Aktør:	Bonde
Prebetingelser:	1. Bonden må være innlogget i systemet.
	2. Bonde har minst en sau registrert.
	3. Bonden har navigert seg til vinduet for å redigere og legge til sauер.
Postbetingelser:	1. Informasjon om sauene blir vist på skjerm bildet.
Normal hendelsesflyt:	1.Bonden velger en gitt sau fra listen over de registrerte sauene. 2. Boden får en oversikt over informasjonen til den gitte sauen.

Tabell 11.14: Use case 10

Use case 10	Som en bonde vil jeg kunne sjekke på kart hvor en sau sist var registrert.
Aktør:	Bonde
Prebetingelser:	1. Bonden må være innlogget i systemet. 2. Bonden har minst en sau registrert i systemet. 3. Bonden har navigert seg til vinduet for kartoversikten.
Postbetingelser:	1. Sauen blir vist fram på kartet.
Normal hendelsesflyt:	1. Bonden velger en gitt sau.
Variasjoner:	1. Kartet viser posisjonen til alle sauene i stedet for kun en sau.

Tabell 11.15: Use case 11

Use case 11	Som en bonde vil jeg kunne få alarm til meg selv og reserve-kontaktperson"via e-post, sms eller telefon om en sau blir angrepet.
Aktør:	Bonde
Prebetingelser:	1. Sauen må være under angrep.
Postbetingelser:	1. Informasjon om angrepet på en gitt sau blir sendt ut via e-post, sms eller telefon.
Normal hendelsesflyt:	1. En ugitt sau blir angrepet. 2. Informasjon om at en gitt sau blir angrepet blir sendt ut til bonden og reserve-kontaktpersonen.
Variasjoner:	1. En ugitt sau blir angrepet. 2. Informasjon om at en gitt sau blir angrepet blir ikke sendt til bonden eller reserve-kontaktpersonen siden den registrerte informasjonen viser til feil epost eller telefonnummer.

Tabell 11.16: Use case 14

Use case 14	Som en bonde vil jeg kunne hente ut en logg med informasjon om tidligere angrep og lokasjoner.
Aktør:	Bonde
Prebetingelser:	1. Bonden må være innlogget i systemet. 2. Bonden må ha minst en sau registrert i systemet. 3. Bonden må ha navigert seg til vinduet for loggen.
Postbetingelser:	1. Skjermbildet viser tidligere angrep og lokasjoner.
Normal hendelsesflyt:	1. Skjermbildet viser en logg over alle sauene i en tabellform.
Variasjoner:	1. Skjermbildet viser en logg over alle sauene i en tabellform. 2. Bonden velger en bestemt sau. 3. Skjermbildet viser en logg over den bestemte sauens tabellform.

11.3 Sprinter

11.3.1 Sprint 1

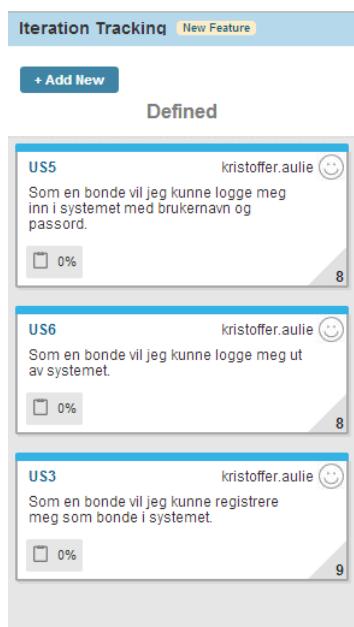
Tid: 13. september - 26. september

Project Backlog

Siden dette var vår første sprint, startet vi med å gå gjennom alle systemkravene vi hadde fått utdelt og blitt enige om med studass. Ut i fra dette laget vi foreløpige User Stories som omfattet alle kravene til systemet, og som dermed utgjorde vår Project Backlog. Denne prioriterte vi etter viktighet.

Sprint Backlog

Etter vi definerte Project Backlog, diskuterte vi oss fram til hva som skulle gjennomføres i den første sprinten, samt delte inn hver user story i mindre “tasks” som vi estimerte i antall timer. Dermed fikk vi beregnet hvor mange timer hver enkelt user story ville ta, og passet på å velge ut user stories til denne sprinten slik at antall timer ikke overskred 5 timer per person i uka (slik beskrevet under Tidsestimering). Resultatet kan sees på bildet under.



Figur 11.20: Sprint backlog 1

Sprint review møte

Siste dagen i sprinten hadde vi et sprint review møte med studass for å vise hva vi hadde gjort, og få et anslag på hvordan vi ligger an. Alle user storiene i sprint backlog for denne sprinten ble gjennomført før fristen, og vi klarte å holde oss innenfor det antallet timer vi hadde til rådighet. Studass følte at vi hadde klart å gjøre det han hadde forventet, og mente at vi var på riktig spor.

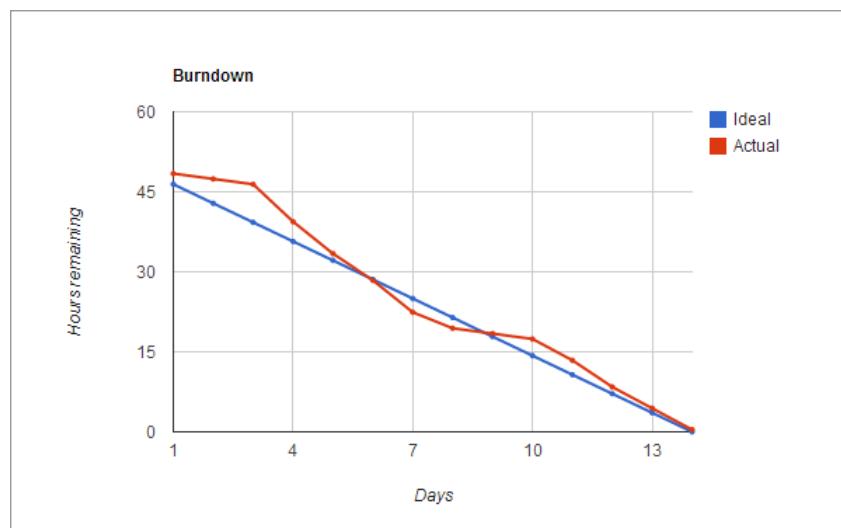
Sprint retrospective

Etter å ha evaluert den første sprinten ser vi at det viste seg å være en god avgjørelse å ta hensyn til uforutsette problemer i tidsestimeringen. Dette er fordi vi støtte på flere oppstartsproblemer, blant annet med versjonkontrollen git. Dette tok litt tid å fikse for enkelte av gruppemedlemmene,

og ville nok ha ført til at vi hadde overskredet tidsestimatet dersom vi ikke hadde lagt til ekstra tid for tilfeldigheter.

Siden vi klarte å fullføre user storyene vi hadde plassert i vår Sprint Backlog, føler vi at sprinten gikk nokså bra. Kommunikasjonen i gruppa har fungert greit, folk har sagt ifra på forhånd om de er syke eller må være borte. Hjemmearbeid har også blitt gjort til avtalt tid. Til neste sprint kan vi likevel prøve å booke møterom litt tidligere, og bruke mer faste møtesteder slik at folk ikke bruker lang tid på å finne fram, noe som har vært tilfelle tidligere og ført til forsinkelser.

Som vi ser i burndown chartet under, brukte vi litt tid på å komme i gang, men klarte etter hvert å få unna arbeidsoppgavene.



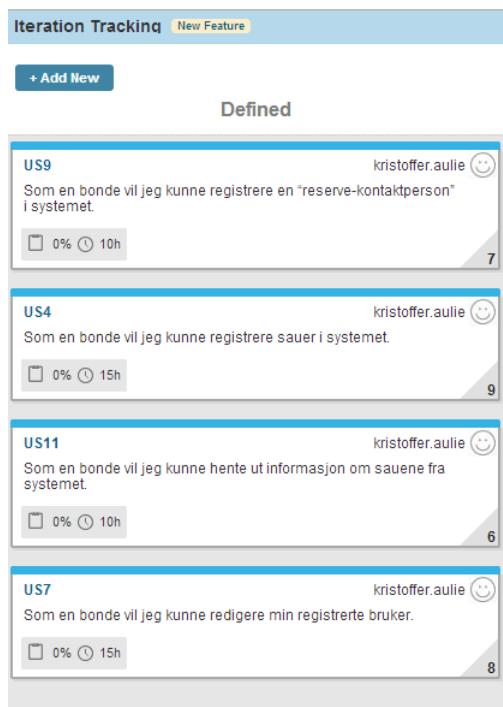
Figur 11.21: Burndown chart sprint 1

11.3.2 Sprint 2

Tid: Tid: 27. september - 10. oktober

Sprint Backlog

Siden vi klarte å gjennomføre forrige sprint på innenfor planlagt tidsrom, gjorde vi ingen endringer i antall timeverk det samlede estimatet av user storyene bestod av. Ut ifra dette, samt hvilke user stories som hadde høy prioritet og var naturlig å implementere på dette stadiet, kom vi fram til at Sprint 2 ble seende ut som på bildet under.



Figur 11.22: Sprint backlog 2

Sprint review møte

Vi hadde igjen et møte med studass der vi viste han hva vi hadde gjort siden sist. Vi demonstrerte funksjonaliteten vi hadde implementert, og diskuterte hva som bør gjøres videre i programmet. Vi klarte å implementere alle user storiene i denne sprint backloggen i tide, i tillegg til en ekstra user story som vi la til i sprinten senere siden vi ble ferdige med sprint backloggen raskere enn planlagt. Vi forklarte dette for studass, og han var enig i at det var riktig å legge til ekstra arbeidsoppgaver i backloggen når vi hadde tid til overs, så lenge det dokumenteres og begrunnes. Utviklingen av programmet er med andre ord tilfredsstillende for "kunden", og vi fikk beskjed om å fortsette slik videre.

Sprint retrospective

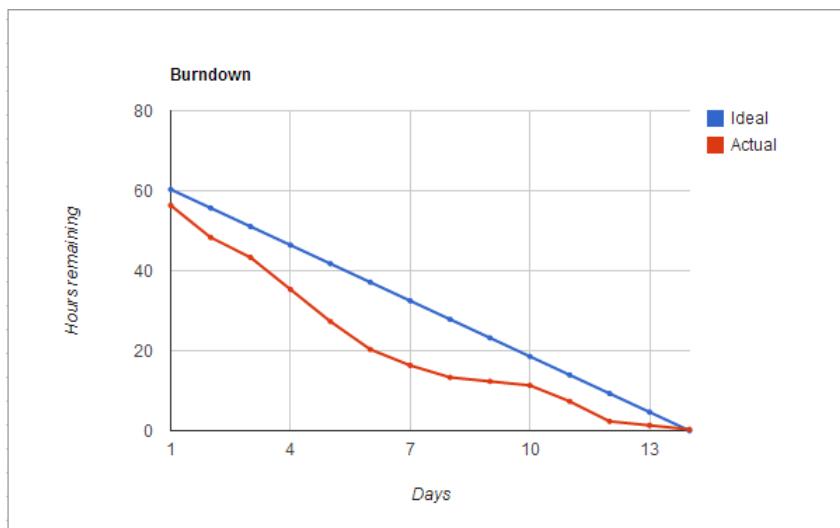
Etter møtet med studass satt vi sammen i gruppa og diskuterte hvordan denne sprinten hadde vært. I forrige sprint ble vi enige om at vi burde bruke faste møtesteder for å unngå sløsing av tid, noe vi har klart å gjøre bedre i denne sprinten. Vi har igjen vært flinke til å si i fra dersom vi ikke har mulighet til å delta på et møte, og alt arbeid har blitt gjort som planlagt.

Alle user stories i sprint backloggen ble gjennomført på mindre tid enn estimert, noe som førte til

at vi la til en ekstra user story i denne sprinten. Dette gjorde vi fordi Scrum er en smidig metode der det er naturlig med endringer i planene underveis, i tillegg til at user storyen vi la til var nokså knyttet til andre user stories i sprint backloggen, og var derfor et naturlig neste-steg.

Til neste sprint kan vi gjerne estimere litt mindre tid på hver user story, samt bli flinkere til å sette user stories som er tett knyttet sammen i samme sprint.

I burndown chartet under kan vi se at vi hele veien lå foran skjema.



Figur 11.23: Burndown chart sprint 2

11.3.3 Sprint 3

Tid: 11. oktober - 24. oktober

Sprint Backlog

I denne sprinten skal vi arbeide med relativt kompliserte oppgaver som vil kreve mye tid til research og prøving-og-feiling. Selv om vi i utgangspunktet la til for liten arbeidsmengde i forrige sprint, vil denne sprinten være litt annerledes siden arbeidsoppgavene ikke er like rett-fram som de i forrige sprint. Derfor vil vi beregne ekstra tid for undersøkelser siden vi ikke har vært så mye borti kartverk i Java Swing før. Sprint backloggen til Sprint 3 kan sees i bilde under.



Figur 11.24: Sprint backlog 3

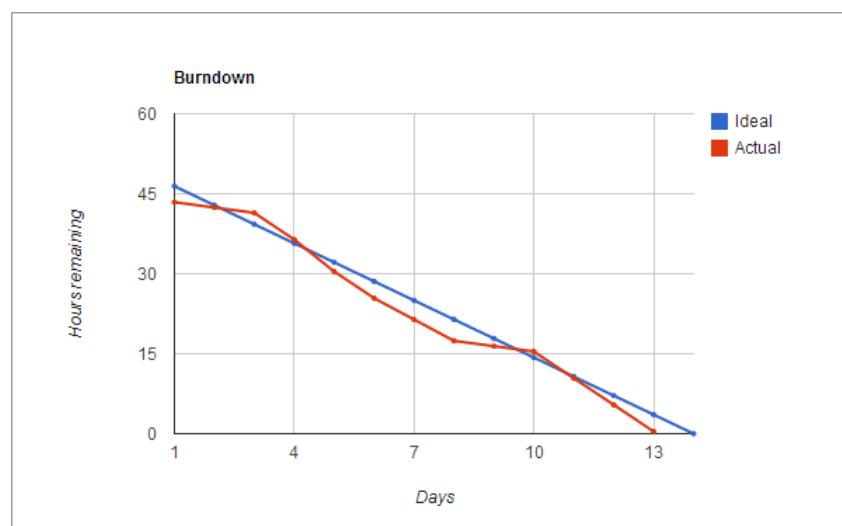
Sprint review møte

I dette møtet med studass fikk vi vist det vi hadde gjort siden sist. Igjen hadde vi klart å få utført alle oppgavene i sprinten før innen tidsfristen, og vi forklarte samtidig hva vi tenkte å gjøre videre med det vi hadde implementert. Studass kom med innspill underveis, noe som var behjelplig med hvordan vi tenker å fullføre applikasjonens funksjonalitet og utseende. Samtidig fikk vi vist litt av hva vi hadde skrevet i rapporten, i tillegg til at vi pratet endel om hvordan vi burde løse rapportskrivingen for å få et best mulig resultat. Dette var meget hjelpsomt, og vi har fått et bedre innblikk i hvordan rapporten bør være.

Sprint retrospective

Igjen fikk vi utført alle user stories som var med i sprinten. Vi brukte nokså mye tid på research denne gangen i forbindelse med hvilket kartverk vi skulle benytte i applikasjonen. Det tok litt tid å finne ut hvilken karttype vi skulle bruke, noe som innebar prøving og feiling. Det viste seg derfor å være et godt valg å beregne mye tid for kartimplementasjonen denne sprinten, siden selve utforskningen egentlig tok mer tid enn implementasjonen. Vi ser at deadlineen nærmer seg, så i neste sprint må vi antageligvis jobbe litt mer enn vi har gjort hittil. Vi ble enige i å legge resten av user storyene fra produkt backloggen inn i neste sprint, slik at vi kan ha litt ekstra tid før innleveringen for å fikse på småting.

Av burndown chartet under, kan vi se at vi klarte å arbeide nokså jevnt hele sprinten, selv om det denne gangen var få og store oppgaver.



Figur 11.25: Burndown chart sprint 3

11.3.4 Sprint 4

Tid: Tid: 25. oktober - 7. november

Sprint Backlog

I denne sprinten vil vi legge inn resten av user storyene i produkt backloggen slik at vi kan bli ferdige med hovedimplementasjonen etter denne sprinten. Vi må antageligvis jobbe litt mer denne sprinten for å klare å bli ferdige med alt, men vi skal nok likevel klare å holde det innenfor timeantallet vi har til rådighet, så lenge vi jobber effektivt. Siden vi har blitt litt mer drevne på programmeringen, vil nok arbeidsoppgavene kunne utføres raskere enn tidligere.



Figur 11.26: Sprint backlog 4

Sprint review møte

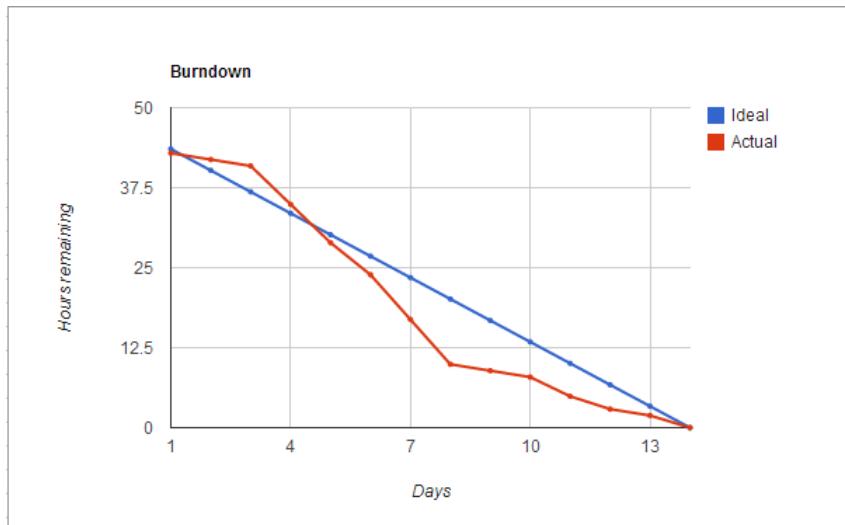
Etter denne sprinten ble vi helt ferdig med resten av programmet. Vi viste hele systemet til studass, spesielt de delene han gav innspill i ved forrige sprint review møte, samt den nye funksjonaliteten vi implementerte i denne sprinten. Han var fornøyd med det vi hadde gjort, og konstaterte at programmet nå var ferdig. Vi fikk også innspill på flere av diagrammene vi hadde laget, i tillegg til at vi fikk spurt om testene vi nå skal utføre bør inkluderes i sprintene. Vi bestemte oss for at det bør inkluderes i neste sprint, slik at sprintene til slutt inkluderer alle oppgavene direkte relatert til systemet.

Sprint retrospective

Vi måtte jobbe nokså hardt denne uka, og til tross for at vi jobbet meget effektivt, gikk vi noen få timer over estimatet vi hadde satt opp på forhånd. Dette er nokså naturlig i avslutningsfasen i et prosjekt, siden alt skal skikkelig på plass og fungere sømløst sammen. Dette fører til at små bugs stadig oppdages og må fikses, noe som medfører ekstra arbeidstimer. Til slutt klarte vi å nå målet med sprinten, nemlig å bli ferdig med hele systemet. Dette fører til at vi nå har nokså god

tid til å gjennomføre skikkelig testing av programmet, samt bruke resten av tiden på å fullføre rapporten vår. Denne sprinten var helt klart den mest krevende hittil, og selv om dette var vår mest effektive sprint hittil, gikk vi litt over tidsestimatet. Under testingen vil vi derfor passe på at vi ikke undervurderer størrelsen av testoppgavene, slik at vi ikke ender opp med å få dårlig tid likevel.

I burndown chartet under kan vi se at vi kom tregt i gang, og oppdaget at vi måtte jobbe flere timer enn planlagt. Da vi gjorde dette klarte vi å havne foran skjema.



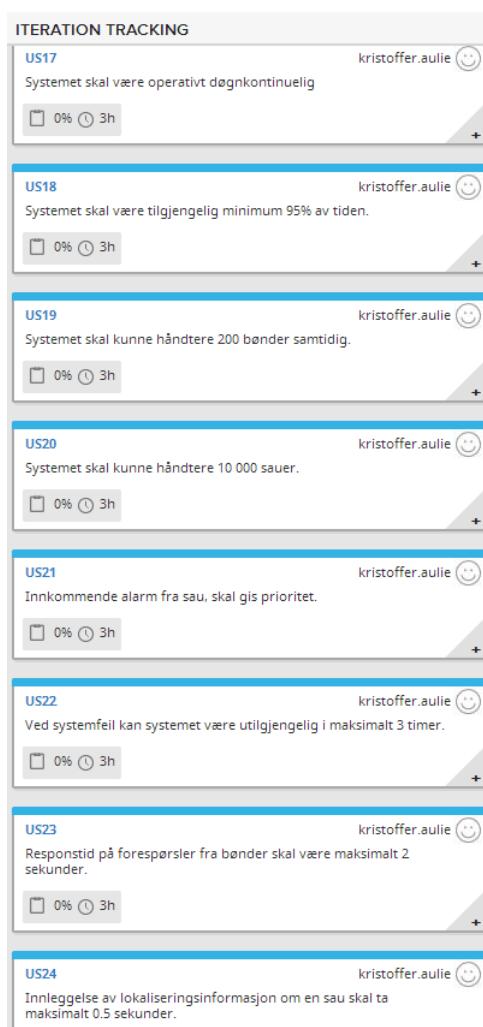
Figur 11.27: Burndown chart sprint 4

11.3.5 Sprint 5

Tid: Tid: 8. november - 14. november

Sprint Backlog

Denne sprinten vil være vår siste sprint, og vil på flere måter være litt annerledes enn de tidligere sprintene. Siden all funksjonaliteten allerede er implementert, vil vi bruke denne sprinten til å gjennomføre tester, samt sikre at de ikke-funksjonelle kravene er oppfylt. Sprinten vil forøvrig var i kun én uke, til forskjell fra tidligere da vi har brukt to-ukers sprinter. Grunnen til dette er at testingen ikke krever like mye tid som de tidligere implementeringsoppgavene, spesielt siden vi også har passet på å teste endel underveis i sprintene. Det nærmer seg også innlevering, så det er viktig at vi får ferdig denne sprinten i passe god tid før hele prosjektet skal leveres. Etter erfaringene fra forrige sprint, passer vi på å ikke undervurdere størrelsen av testoppgavene denne sprinten.



Figur 11.28: Sprint backlog 5

Sprint review møte

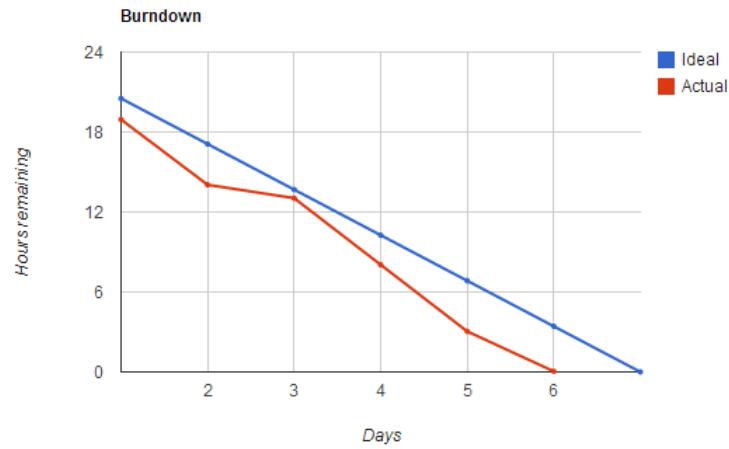
Siden testene ikke førte til store forandringer i programmet, hadde vi ikke noe fysisk møte med studass i slutten av denne sprinten. I stedet hadde vi avtalt å sende inn rapporten, slik at han kan gi oss noen siste tilbakemeldinger før innlevering. I forrige sprint review møte ble det konstatert

at programmet var ferdig, så vi anser nå prosjektet som fullført.

Sprint retrospective

I forrige sprint retrospective gjorde vi oss selv oppmerksom på at vi måtte passe på å ikke undervurdere testoppgavene. Vi jobbet nokså mye, og klarte derfor å bli ferdig innen tidsfristen. Det viste seg å være en god avgjørelse å legge inn en kort siste sprint for å gjennomføre testing, siden dette førte til at vi kunne ha full fokus på de store testene, i tillegg til rapportskrivingen som medfører i testingen.

Siden vi var godt forberedt på arbeidsmengden testene ville kreve, kan vi se av burndown chartet under at vi hele veien lå foran skjema.

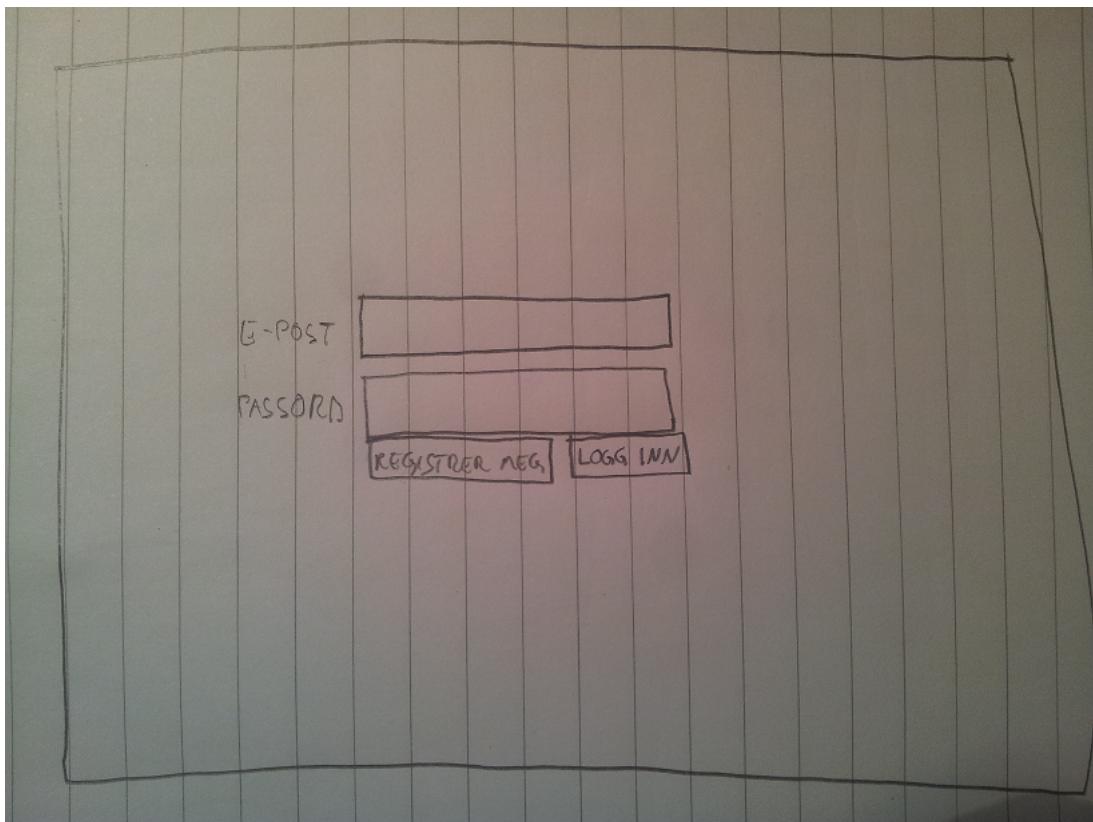


Figur 11.29: Burndown chart sprint 5

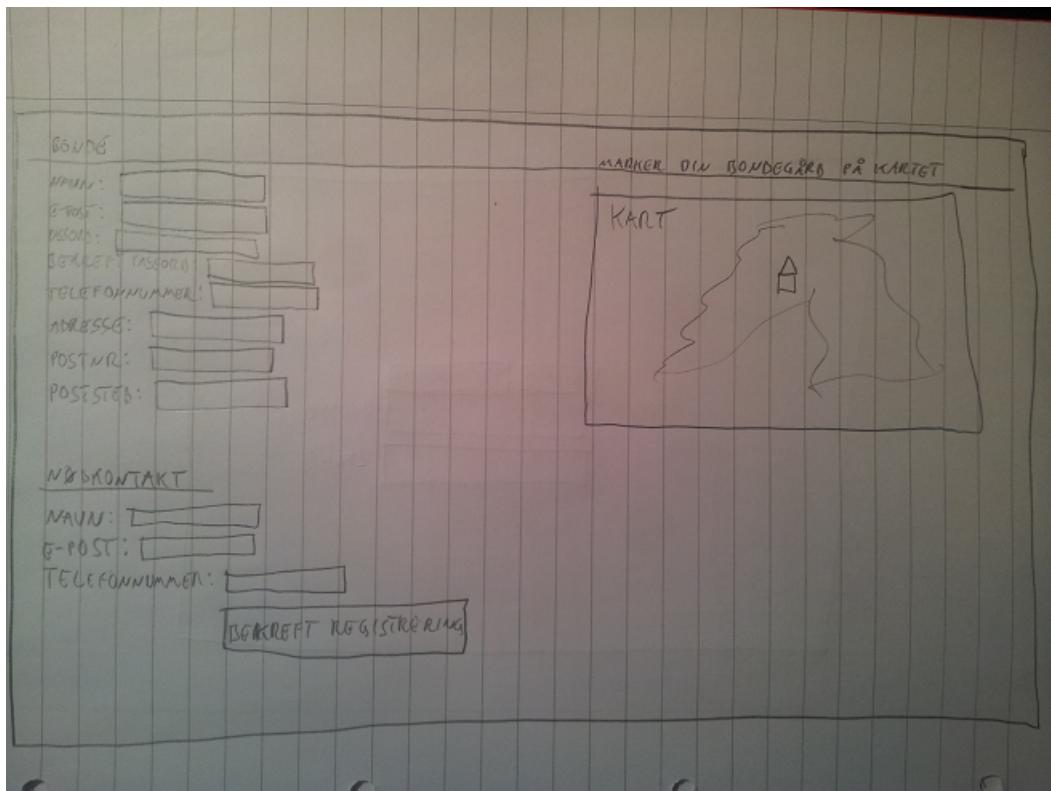
11.4 Testing

11.4.1 Papirprototypetest

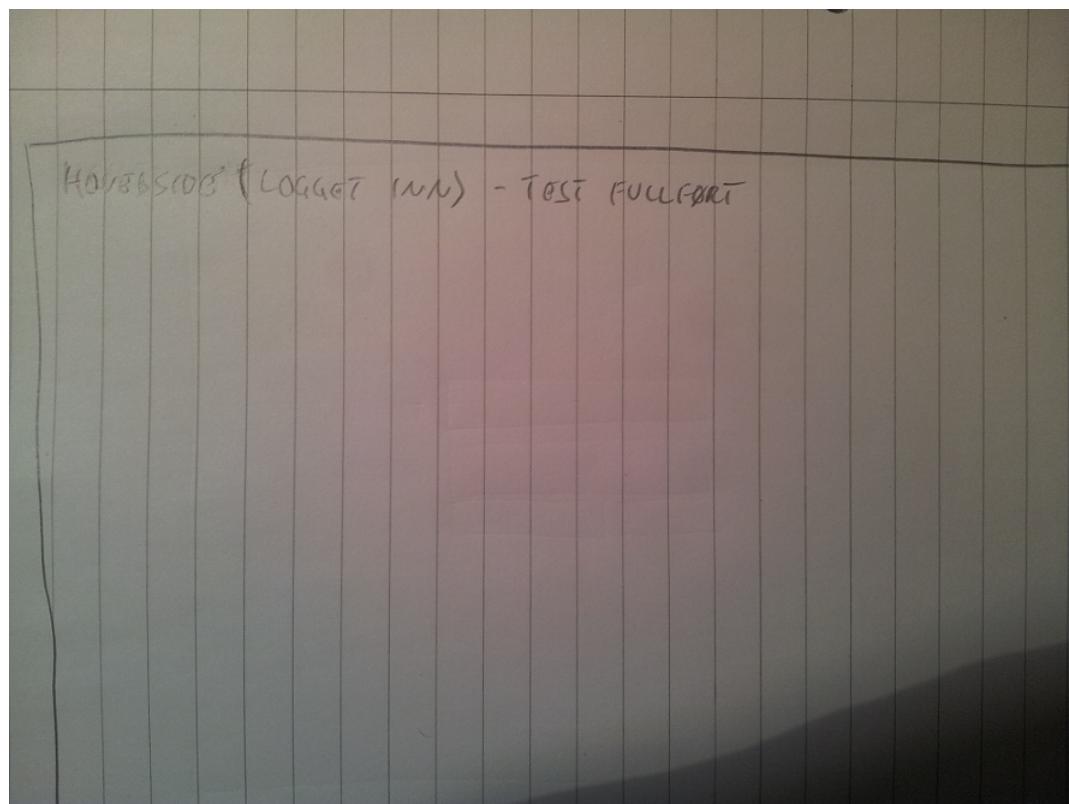
Under er et eksempel på en papirprototypetest for å registrere en bonde og nødkontakt.



Figur 11.30: Skjerm bilde 1



Figur 11.31: Skjerm bilde 2



Figur 11.32: Skjermbilde 3

11.4.2 Systemtest

For alle systemtestene har vi skrevet hvilke kravnummer fra kravspesifikasjonen testene dekker. Kravspesifikasjonen kan du finne her: [Kravspesifikasjon 11.2](#)

Testing av funksjonelle krav

Tabell 11.17: Test case 1 tabell

Test case 1	Registrere bruker
Kravnummer:	1, 7
Prebetingelse:	Bruker har programmet og java
Steg:	
1.	Start programmet
2.	Trykk "Registrer meg"
3.	Fyll inn brukerinformasjon
Forventet resultat:	Brukeren har opprettet en bruker, og logges inn
Faktisk resultat:	Brukeren ble opprettet og logget inn.
Konklusjon:	Vellykket

Tabell 11.18: Test case 2 tabell

Test case 2	Logge inn i programmet
Kravnummer:	3
Prebetingelse:	Bruk er registrert bruker
Steg:	
1.	Skriv inn e-post og passord i feltene til innlogginsskjermen
2.	Trykk "Logg inn"
Forventet resultat:	Brukeren er logget inn, og ser kartet på hovedfanen
Faktisk resultat:	Brukeren ble logget inn, og ser kartet på hovedfanen.
Konklusjon:	Vellykket

Tabell 11.19: Test case 3 tabell

Test case 3	Redigere bruker og nødkontakt
Kravnummer:	5, 8
Prebetingelse:	Bruk er logget inn
Steg:	
1.	Trykk på fanen: "Rediger bonde og nødkontakt"
2.	Fyll ut feltene du vil forandre, og skriv inn nåværende passord
3.	Trykk på: "Bekreft endringer"
Forventet resultat:	Det står: "Redigering vellykket", og informasjonen har blitt endret
Faktisk resultat:	Det står: "Redigering vellykket", og informasjonen har blitt endret
Konklusjon:	Vellykket

Tabell 11.20: Test case 4 tabell

Test case 4	Registrere sau
Kravnummer:	2
Prebetingelse:	Bruker er logget inn
Steg:	
1.	Trykk på fanen Legg til og rediger sau"
2.	Fyll ut informasjon om sauen i boksen til venstre
3.	Trykk på Bekreft registrering"
Forventet resultat:	Sauen har blitt registrert, og ligger i listen: "Mine sau"
Faktisk resultat:	Sauen har blitt registrert, og ligger i listen: "Mine sau"
Konklusjon:	Vellykket

Tabell 11.21: Test case 5 tabell

Test case 5	Redigere sau
Kravnummer:	6
Prebetingelse:	Bruker har en eller flere sau registrert
Steg:	
1.	Trykk på fanen Legg til og rediger sau"
2.	Trykk på sauen du vil redigere i listen: "Mine sau"
3.	Fyll ut feltene du vil forandre
4.	Trykk på: Bekreft redigering"
Forventet resultat:	Det står: Redigering av sau var vellykket!", og informasjonen har blitt forandret
Faktisk resultat:	Det står: Redigering av sau var vellykket!", og informasjonen har blitt forandret
Konklusjon:	Vellykket

Tabell 11.22: Test case 6 tabell

Test case 6	Se registrerte sau på kart
Kravnummer:	10, 12, 13
Prebetingelse:	Bruker har en eller flere sau registrert
Steg:	
1.	Trykk på fanen Kartoversikt"
2.	Trykk på: "Oppdater saurapporter"
Forventet resultat:	Registrerte sau vil vises på kartet rundt registrert gård
Faktisk resultat:	Registrerte sau vil vises på kartet rundt registrert gård
Konklusjon:	Vellykket

Tabell 11.23: Test case 7 tabell

Test case 7	Hente ut informasjon om sau
Kravnummer:	9
Prebetingelse:	Bruker har en eller flere sau registrert
Steg:	
1.	Trykk på fanen Legg til og rediger sau"
2.	Velg sauen du vil ha informasjon om fra listen: "Mine sau"
Forventet resultat:	Sauens informasjon vil bli vist i feltene til høyere i fanen
Faktisk resultat:	Sauens informasjon ble vist i feltene til høyere i fanen
Konklusjon:	Vellykket

Tabell 11.24: Test case 8 tabell

Test case 8	Se logg for en enkelt sau
Kravnummer:	14
Prebetingelse:	Bruker har en eller flere sauер registrert
Steg:	
1.	Trykk på fanen: 'Logg'
2.	Velg ønsket sau fra listen: 'Filtrer på sau'
3.	Velg ønsket tidperiode i kalenderfeltene under
Forventet resultat:	Alle hendelser for sauen vil vises i listen under
Faktisk resultat:	Alle hendelser for sauen vises i listen under
Konklusjon:	Vellykket

Tabell 11.25: Test case 9 tabell

Test case 9	Bonde og nødkontakt får varsling på e-post når sau er under angrep
Kravnummer:	11
Prebetingelse:	Bruker har en eller flere sauер registrert, og simulatoren kjører
Steg:	
1.	Generer angrep på sau med simulatoren.
Forventet resultat:	Etter simulatoren har oppdatert seg, vil bonden og nødkontakt få e-post hvis en sau har blitt angrepet
Faktisk resultat:	Bonde og nødkontakt fikk e-post med koordinater og kart som viser angrepslokasjonen.
Konklusjon:	Vellykket

Tabell 11.26: Test case 10 tabell

Test case 10	Logge ut av systemet
Kravnummer:	4
Prebetingelse:	Brukeren er logget inn i systemet
Steg:	
1.	Trykk på knappet: 'Logg ut', nederst i høyere hjørne av vinduet
Forventet resultat:	Bruker blir logget ut, og programmet går til innloggingsskjermen
Faktisk resultat:	Bruker ble logget ut, og programmet gikk til innloggingsskjermen
Konklusjon:	Vellykket

Testing av ikke-funksjonelle krav

For at disse testene skal være nøyaktige bør vi egentlig gjennomføre de i en mye større skala enn det vi gjør nå. Allikevel gir testene under oss en indikasjon på om systemet oppfyller de ikke-funksjonelle kravene.

I testene som går på responstid så har vi 200 bønder og 10 000 sauер i systemet, slik at vi kan sikre kapasiteten til systemet for alle kravene.

Tabell 11.27: Test case 11 tabell

Test case 11	Systemet skal være operativt døgnkontinuerlig
Kravnummer:	15
Steg:	
1.	Prøv å bruke systemet 10 ganger normalfordelt i løpet av et døgn.
Forventet resultat:	Systemet fungerer som det skal alle gangene.
Faktisk resultat:	Systemet fungerte som det skulle alle gangene.
Konklusjon:	Vellykket

Tabell 11.28: Test case 12 tabell

Test case 12	Systemet skal være tilgjengelig minimum 95% av tiden.
Kravnummer:	16
Steg:	
1.	La systemet stå på i et døgn.
2.	Beregn tiden det er tilgjengelig.
Forventet resultat:	Systemet er tilgjengelig hele tiden.
Faktisk resultat:	Systemet var tilgjengelig hele tiden.
Konklusjon:	Vellykket

Tabell 11.29: Test case 13 tabell

Test case 13	Systemet skal kunne håndtere 200 bønder samtidig.
Kravnummer:	17
Steg:	
1.	Legg inn 200 bønder.
2.	Gjennomfør kall fra alle de 200 bøndene samtidig.
Forventet resultat:	Systemet oppfører seg normalt.
Faktisk resultat:	Systemet oppførte seg normalt.
Konklusjon:	Vellykket

Tabell 11.30: Test case 14 tabell

Test case 14	Systemet skal kunne håndtere 10 000 sauер.
Kravnummer:	18
Steg:	
1.	Legg inn 10 000 sauер.
2.	Gjennomfør simuleringer for alle sauene.
Forventet resultat:	Systemet oppfører seg normalt.
Faktisk resultat:	Systemet oppførte seg normalt.
Konklusjon:	Vellykket

Tabell 11.31: Test case 15 tabell

Test case 15	Innkommande alarm fra sau, skal gis prioritet.
Kravnummer:	19
Steg:	
1.	Simuler alarm fra sau.
2.	Sjekk alarmrapporten på hovedsiden.
Forventet resultat:	Alarmrapporten vises øverst i rapportlista, og det er alarmikon ved siden av saueikonet på kartet.
Faktisk resultat:	Alarmrapporten vises øverst i rapportlista, og det er alarmikon ved siden av saueikonet på kartet.
Konklusjon:	Vellykket

Krav nummer 20 'Ved systemfeil kan systemet være utilgjengelig i maksimalt 3 timer', blir vanskelig for oss å teste. Siden vi har benyttet oss av NTNU sine servere, gå vi ut i fra at driftsavdelingen der vil få systemet raskt opp igjen ved feil. Dette har vi og tatt høyde for i risikomatrisen.

Tabell 11.32: Test case 16 tabell

Test case 16	Responstid på forespørsler fra bønder skal være maksimalt 2 sekunder.
Kravnummer:	21
Steg:	
1.	Legg inn 10 000 sauер for en bonde.
2.	Simuler rapporter for alle sauene.
3.	Logg inn som bonden.
4.	Trykk på "Oppdater saueraporter" og beregn tiden fra knappen trykkes, til rapportene er oppdatert.
Forventet resultat:	Alle rapportene hentes og vises på under 2 sekunder.
Faktisk resultat:	Alle rapportene ble hentet og vist på 293 millisekunder.
Konklusjon:	Vellykket

Tabell 11.33: Test case 17 tabell

Test case 17	Innleggelse av lokaliseringinformasjon om en sau skal ta maksimalt 0,5 sekunder.
Kravnummer:	22
Steg:	
1.	Simuler saueraport med lokaliseringinformasjon.
2.	Beregn tiden det tar å legge inn rapporten.
Forventet resultat:	Det tar under 0,5 sekunder å legge inn rapporten.
Faktisk resultat:	Rapporten ble lagt inn på 18 millisekunder.
Konklusjon:	Vellykket

11.4.3 Brukbarhetstest

Forklaring til SUS-skjema

Hver testperson setter strek i en rute for hvert utsagn. Det går på en skala fra 1 til 5 der 1 betyr uenig, mens 5 betyr enig. Vi har fire testpersoner som vi har kalt A, B, C og D, og markeringene til de respektive personene er vist med bokstaver i skjemaet under.

Resultatet av et SUS-skjema er mellom 0 og 100. Dette beregnes ved ved følgende regler:

- Utsagn 1, 3, 5, 7 og 9 = (Skalaposisjon - 1)
- Utsagn 2, 4, 6, 8 og 10 = (5 - Skalaposisjon)
- Endelig SUS-poengsum = $(2,5 * \text{Sum av poeng})$

Resultater:

A: 92,5

B: 90

C: 92,5

D: 92,5

Endelig gjennomsnittlig resultat = 91,9

Dette er et resultat vi er fornøyde med. Underveis i testen fikk vi også enkelte innspill som vi tok til følge, blant annet at enkelte knapper bør få andre navn, samt at de bør plasseres bedre.

Utsagn	1	2	3	4	5
1. Jeg tror at jeg vil bruke dette systemet ofte.					ABCD
2. Jeg synes systemet var unødvendig komplisert.	ABC	D			
3. Jeg synes systemet var enkelt å bruke.			B	ACD	
4. Jeg tror jeg vil trenge hjelp fra en teknisk anlagt person for å kunne bruke dette systemet.	ABC	D			
5. Jeg synes at hele systemet hang godt sammen.			B	ACD	
6. Jeg synes at det var for mye inkonsistens i systemet.	BCD	A			
7. Jeg vil tro at de aller fleste brukere vil kunne lære seg bruken av systemet meget raskt.			A	BCD	
8. Jeg synes systemet var tungvint å bruke.	ABD	C			
9. Jeg følte meg sikker da jeg brukte systemet.			BDC	A	
10. Jeg måtte lære meg mange ting før jeg kunne ta i bruk systemet.	D	ABC			

Figur 11.33: SUS-skjema

11.5 Brukermanual

Her følger en teknisk og en ikke-teknisk brukermanual. Den tekniske brukermanualen forklarer hvordan man kan hente inn prosjektet for å se og endre på kildekoden. Den ikke-tekniske brukermanualen forklarer hvordan sluttbrukeren av systemet kan bruke de forskjellige funksjonene som finnes.

11.5.1 Teknisk brukermanual

Tilgang til databasen

Logg inn via <https://mysqladmin.stud.ntnu.no/>

Brukernavn: kristoau_sheep

Passord: gruppe1

Databasenavn: kristoau_sheepwatch

Sette opp NetBeans med Java Development Kit

Last ned og installer NetBeans med JDK her (da slipper du å installere jdk separat): <http://www.oracle.com/technetwork/7-netbeans-download-432126.html>

Legg inn prosjektet i NetBeans manuelt

1. I NetBeans, velg File->Open Project
2. Bla deg fram til Gruppe1_kildekode -> prosjekt1
3. Velg både SheepWatchSimulator og SheepWatchUI
4. Trykk Open Project

Alle bibliotekersom trengsligger i prosjektenes lib-mapper.

Klone prosjektet i NetBeans fra GitHub

1. NetBeans har integrert støtte for versjonskontrollen GIT.
2. Følg denne guiden med inputen under: <https://netbeans.org/kb/73/ide/git.html#github>
Repository URL: git@github.com:iverasp/prosjekt1.git
3. Når du kloner, velg kun "Master" branch.

Alle biblioteker som trengs ligger i prosjektenes lib-mapper.

Oppdatere endringer til og fra GitHub

1. Hent de siste endringene ved å velge prosjektet og trykke Team->Git->Remote->Pull
2. Legg til de siste endringene du har gjort ved å trykke Team->Git->Commit og så Team->Git->Remote->Push

11.5.2 Ikke-teknisk brukermanual

Brukermanualen vil ta for seg en detaljert framgangsmetode for hvordan man skal bruke programmet "SheepWatch" til sitt fulle potensial. Manualen vil benytte seg av bilder og tekst for å beskrive programmet. Interaksjonen med brukeren blir markert med nummererte bildet og en tesktlig beskrivelse av interaksjonen følger. Manualen vil også ta for seg hvordan man kjører programmet og hvilke programvare som er nødvendig.

Innholdsfortegnelse

1. Kjøring av systemet
2. Login-vinduet
3. Registrering
4. Hovedvindu
5. Legg til og rediger sau
6. Redigere bonde og nødkontakt
7. Logg

Kjøring av systemet

For å kjøre dette systemet må man ha Java installert på maskinen. Java kan lastes ned her:
<https://java.com/en/download/index.jsp>

Kjøring av simulator:

1. Finn frem til "SheepWatchSimulator.jar". Den ligger i mappen Program/SheepWatchSimulator.
2. Kjør jar-fila. Denne skal kjøre hele døgnet. Det er viktig at LIB-mappa ligger i samme mappe som JAR-fila.

Merk at simulatoren genererer rapporter for sauene hver halvtime, men man kan også generere en simulering ved å trykke på knappen 'Simuler rapporter'.

Kjøring av klientprogram:

1. Finn frem til "SheepWatchUI.jar". Den ligger i mappen Program/SheepWatchClient.
2. Kjør jar-fila. Dette er klienten, og skal kunne startes og avsluttes etter eget ønske. Det er viktig at LIB-mappa ligger i samme mappe som JAR-fila.

Vi har opprettet en bruker som kan benyttes i forbindelse med testing av programmet. Dersom dere vil, kan dere endre e-postadressen til brukeren og nødkontakten i **Rediger bonde og nødkontakt**-funksjonen, slik at dere får testet alarmfunksjonen via e-post:

E-post: kristoffer.aulie@gmail.com

Passord: hei

Login-vinduet

Log-in vinduet er det første vinduet som dukker opp når du starter systemet. Under en normal hendelsesflyt til brukeren skrive inn e-postadressen (1) og passordet (2), for å så deretter trykke på “Logg inn” (4). Hvis brukeren ikke allerede har registrert en bruker eller ønsker å registrere en ny bruker kan han/hun trykke på “Registrer meg” (3).



Figur 11.34: Login-vindu

Registrering

Hvis boden må registrer en ny bruker navigere han seg på vinduet for [registrering](#) (Figur 2). En ny bruker må fylle ut den nødvendige personlige informasjonen i tekstfeltene markert med tallet 1. Informasjonen til reserve-kontaktpersonen blir deretter fylt ut i tekstfeltene markert med tallet 2. Brukeren må så registrer lokalisjonen til bondegården sin ved hjelp av kartfunksjonen (3). Til slutt kan brukeren bekrefte (4) eller avbryte(5) registreringen.

Vennligst fyll ut registreringsskjemaet. Alle feltet må fylles ut.

Bonde:

Marker din bondegård i kartet:

1

2

3

4

5

Nødkontakt:

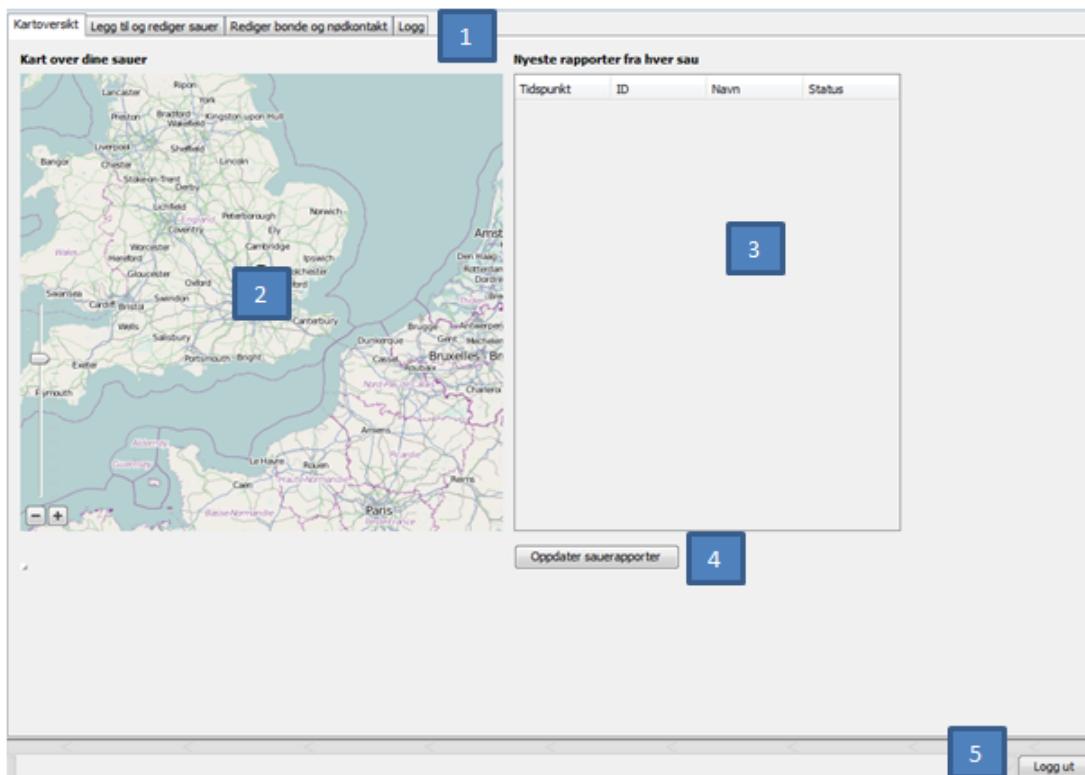
Bekreft registrering

Avbryt registrering

Figur 11.35: Registreringsvindu

Hovedvindu

Øverst på **hovedvinduet** er det en menybar (1) som lar brukeren navigere seg gjennom systemet. Hovedvinduet tar også for seg et kart (2) over de registrerte sauene og en rapport (3) til høyre for kartet. Rapporten vil ta for seg en liten, men informative rapport for sauene. Hvis brukeren trykker på “oppdater saueraport” (4) vil rapporten (3) bli oppdatert for brukeren. Brukeren har også mulighet til å logge seg ut av systemet (5) ved å trykke på “logg ut”.



Figur 11.36: Hovedvindu

Legg til og rediger sauer

Bonden har nå navigert seg til **vinduet** for å legge til og redigere sauene. Her har bonden mulighet til å legge inn en sau (1) ved å fylle inn de nødvendige tekstfeltene for å så bekrefte (2) registreringen. Den registrerte sauene vil da bli vist i listen (2) over sauene som bonden kan da markere ved å trykke på dem. Når bonden har markert en sau har han også mulighet til å slette den (4). Informasjonen til den markerte sauene vil også bli vist i tekstfeltene og radioknappene (5). Her har også brukeren mulighet til å redigere på den allerede registrerte sauene ved å redigere de ønskelige feltene (5) for å så bekrefte endringene (6).

Kartoversikt Legg til og rediger sau Rediger bonde og nækontakt Logg ut

Vennligst skriv inn registreringsinformasjonen for sauene.

Navn:

Vekt:

Kjenn: Vær Søye **1**

Fødselsdato: Nov 4, 2013

Helse:

2 Bekreft registrering

3

4 Slett sau

Informasjon om valgt sau..

Id:

Navn:

Vekt:

Kjenn: Vær Søye **5**

Fødselsdato: Nov 13, 2013

Dødsdato: Nov 13, 2013

Helse:

6 Bekreft redigering

Logg ut

Figur 11.37: Legg til og rediger sau-vindu

Rediger bonde og nødkontakt

Bonden har nå navigert seg til **vinduet** for å redigere personlig informasjon om bonden selv og nødkontakten. Brukeren har nå mulighet til å redigere personlig informasjon (1) og informasjon om nødkontakten (2) ved å redigere tekstfeltene. Videre kan han redigere posisjonen på bondegården (3) for å så bekrefte endringene ved å trykke “bekreft endringene” (4).

Kartoversikt Legg til og rediger sauer Rediger bonde og nødkontakt Logg ut

Vennligst fyll ut de feltene du vil forandre på, og bekreft med passord.

Personlig informasjon:	Nødkontakt:
Navn: <input type="text"/>	Navn: <input type="text"/>
E-post: <input type="text"/>	E-post: <input type="text"/>
Tелефonnummer: <input type="text"/>	Tелефonnummer: <input type="text"/>
Adresse: <input type="text"/>	
Postnr: <input type="text"/> 1	Postnr: <input type="text"/>
Poststed: <input type="text"/>	Poststed: <input type="text"/>
Nytt passord: <input type="text"/>	
Gjenta passord: <input type="text"/>	
Gammelt passord: <input type="text"/>	

Endre plassering av bondegård:

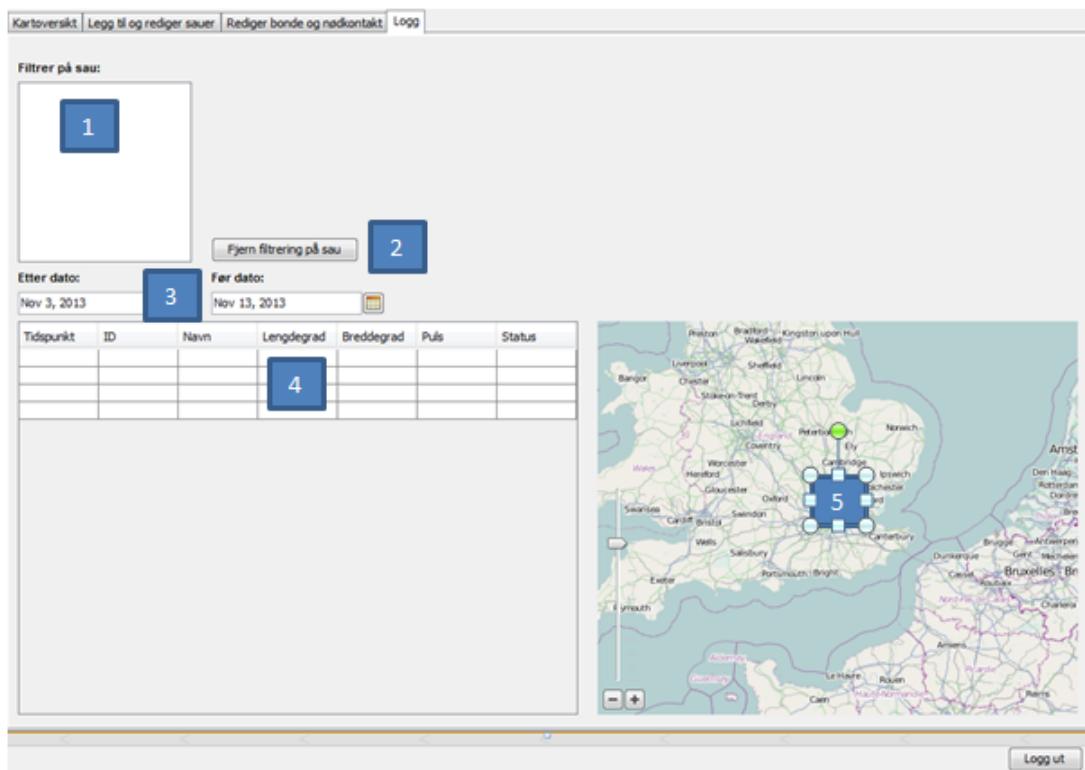
4 Bekreft endringer

Logg ut

Figur 11.38: Rediger bonde og nødkontakt-vindu

Logg

Her har brukeren navigert seg til [vinduet](#) for logg. En liste over de registrerte sauene (1) vil være synlig for brukeren. I utgangspunktet vil ingen av sauene være markert, men brukeren har mulighet til å velge ut en og en sau for å filtrere loggen. Hvis brukeren ønsker å fjerne filtreringen kan han trykke på “Fjern filtrering på sau” (2). Brukeren har også muligheten til å filtrere loggen ut i fra datoer (3). Selve loggen vil bli vist i en tabellform (4) avhengig av hvordan filtreringen gjort tidligere. Kartet nede til høyre (5) viser lokalisjonen til sauene avhengig av filtreringen gjort (1 og 3). Kartet kan enten vise noen bestemte sauer eller alle sauene.



Figur 11.39: Logg-vindu