Kravspesifikasjon

Javaprosjekt - Programutvikling

Laget av Rudi Yu s231776 og Audun Brustad s236341

Høgskolen i Oslo og Akershus
Dataingeniør

Table of Contents

Introduksjon

Kravspesifikasjon utformet av:

Navn	**Studentnummer**
Rudi Yu Haugan	S231776
Audun Brustad	S236341

I kravspesifikasjonen går vi gjennom oppbygningen av programmet, hensikten til programmet og hvilke krav vi stiller til programmet. Det er laget Use-Caser for hoveddelene av programmet. Videre gjøres flere beslutninger i valg av datastrukturer og oppbygning av selve programmet. Vi går også gjennom arbeidsmetode, hvilke verktøy som er planlagt å bruke og hvordan vi skal måle fremgang/arbeid.

Use-cases

Registrere forsikring

Beskrivelse:
- Registrere en gitt type forsikring på en allerede eksisterende kunde i programmet.

Utløser:
- En ansatt trykker på ”registrer forsikring”-knapp

Aktører:
- Ansatt
- Programmet/systemet

Forutsetninger:
1. En ansatt med tilgang og tilstrekkelige rettigheter i programmet
2. Kunden forsikringen registreres på eksisterer i registeret
3. Korrekt utfylt skjema for forsikringstypen

Mål:
- Å korrekt registrere en gitt type forsikring på en kunde i registeret.

Fremgangsmåte:
1. En ansatt starter prosessen
2. Skjemaet fylles ut
3. Forsikringen lagres på person

Unntak:
1. Kunden eksisterer ikke
2. Skjemaet er feilaktig innført

- Ved alle unntak vil brukeren av programmet få skrevet ut en feilmelding og bedt om å gjøre steg 1 på nytt.

Søk

Beskrivelse:

- Gjøre et søk etter person (via personnummer), forsikring (via forsikringsnummer), skademelding (skademeldingsnummer) eller annet søkbart objekt og returnerer resultat(er).

Utløser:

- En ansatt trykker på ”søk”-knapp.

Aktører:

- Ansatt
- Programmet/systemet

Forutsetninger:

1. En ansatt
2. At det er skrevet noe i søkefelt
3. At elementet det søkes etter eksisterer

Mål:

- Å korrekt skrive ut resultat(er) det ble søkt etter.

Fremgangsmåte:

1. En ansatt starter prosessen
2. Programmet tolker hva det søkes etter
3. Programmet returnerer/skriver ut resultater av søket

Unntak:

1. Det er ikke skrevet noe i søkefeltet
2. Det er skrevet feil/programmet klarer ikke tolke hva det søkes etter
3. Det finnes ingen resultater

Registrere Kunde

Beskrivelse:

- Registrere en ny kunde i systemet

Utløser:

- En ansatt trykker på ”Registrer kunde”-knapp.

Aktører:

- Ansatt
- Programmet/systemet

Forutsetninger:

1. En ansatt
2. At brukeren ikke eksisterer fra før
3. All nødvending informasjon om brukeren

Mål:

- Registrere en ny kunde korrekt

Fremgangsmåte:

1. En ansatt starter prosessen
2. Skjema for registrering av ny kunde fra systemet
3. Skjema fylles inn
4. Systemet sjekker forutsetninger
5. Systemet registrerer ny kunde
6. Bruker får resultat

Unntak:

1. Skjema er ikke riktig fylt inn
2. Kunde finnes fra før

- Ved unntak får brukeren en feilmelding og fortsetter prosessen fra steg 3.

Registrere Skademelding

Beskrivelse:

- Registrere en ny skademelding i systemet

Utløser:

- En ansatt trykker på ”Registrer skademelding”-knapp.

Aktører:

- Ansatt
- Programmet/systemet

Forutsetninger:

1. En ansatt
2. Nødvendig informasjon om skaden

Mål:

- Registrere en ny skademelding korrekt

Fremgangsmåte:

1. En ansatt starter prosessen
2. Systemet gir valg om type skademeldinger
3. En ansatt velger hvilken type skademelding
4. Systemet gir riktig skjema i forhold til skadetype
5. Ansatt fyller inn informasjon
6. Systemet sjekker forutsetninger
7. Systemet registrerer skademelding
8. Ansatt får resultat

Unntak:

1. Skjema er ikke riktig fylt inn

- Ved unntak får brukeren en feilmelding og fortsetter ved steg 5.

Statistikk

Beskrivelse:
- Vise ønsket statistikk etter valgte parametere

Utløser:
- En ansatt trykker på ”søk”-knapp.

Aktører:
- Ansatt
- Programmet/systemet

Forutsetninger:
1. En ansatt
2. Parametere til statistikken
    a. Tidsrom
    b. Felter
    c. Visning

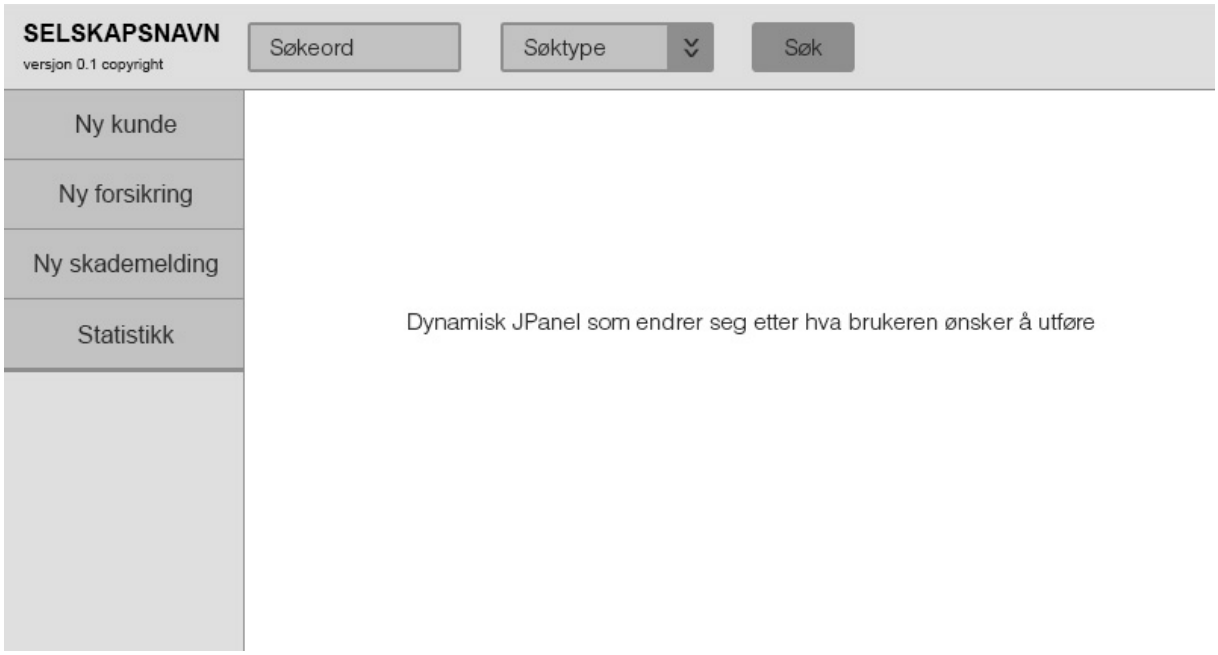
Mål:
- Hente ut historiske data over et tidsrom

Fremgangsmåte:
1. En ansatt starter prosessen
2. Den ansatte kommer med krav
3. Programmet tolker kravene til statistikken
4. Programmet returnerer/skriver ut resultatet av kravene

Unntak:
1. Ugyldig tidsrom
2. Ingen gyldige felter
3. Ingen resultater

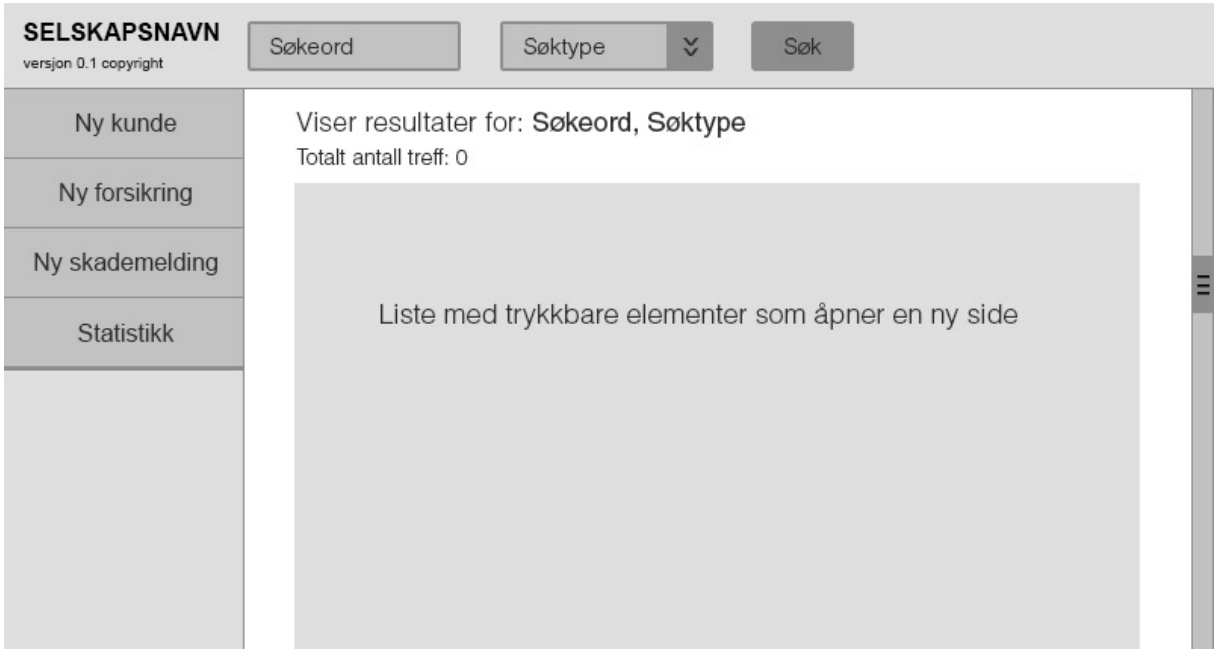
- Ved unntak får brukeren en feilmelding og vil fortsette ved steg 2.

GUI



Over er en skisse av programmets hovedutforming. Vi har satset på et oversiktlig brukergrensesnitt som gjør det enkelt for brukeren å finne målet. Programmet er forholdsvis enkelt utformet ved at kun de mest nødvendige elementene er presentert til brukeren ved oppstart av programmet, som for eksempel å opprette ny kunde og foreta et enkelt søk. Disse elementene er synlige og tilgjengelige for brukeren til enhver tid, uansett hva brukeren holder på med. Det er bare et dynamisk panel (representert av en hvit firkant på bildet) som oppdateres når brukeren starter og utfører forskjellige prosesser. Vinduet skal være dynamisk ved endring av vidde og bredde, det vil si at komponentene tilpasser seg vinduets størrelse dersom brukeren endrer størrelsen.

På hver framvisning av de forskjellige opprettbare elementene (kunde/forsikring/skademelding) er det en «Endre»-knapp. Tanken bak denne er at brukeren/administratoren skal kunne endre gitt informasjon (for eksempel endre fakturaadresse dersom kunden flytter er et eksempel). Dette er gjort ved at informasjonen fra før er plassert i en «uneditable» JTextField som blir «editable» når brukeren trykker «Endre».

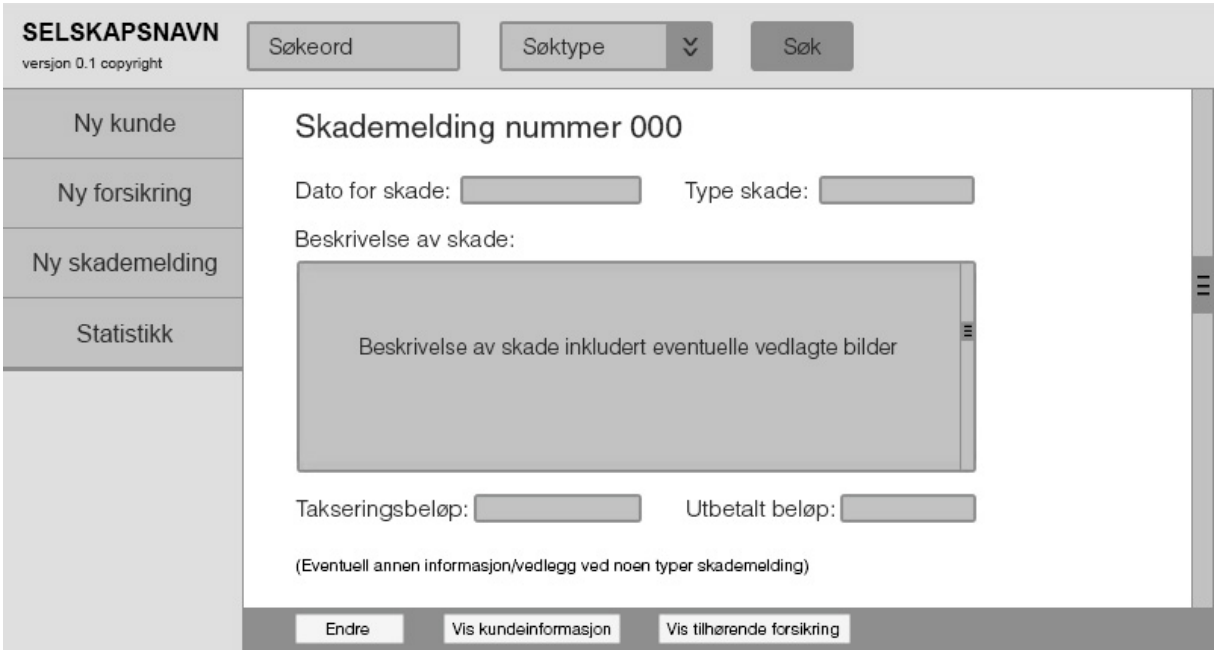


Over er panelet oppdatert etter at brukeren har foretatt et enkelt søk (ved bruk av menyen øverst i vinduet). I den grå boksen vil en liste med klikkbare resultater presenteres til brukeren. Hvilken type (kunde/forsikring/skademelding) kommer an på hva brukeren har valgt på nedfalls-menyen «søktype».

Her er panelet oppdatert når brukeren har søkt og klikket seg inn på en person. Den viktigste informasjonen blir presentert til brukeren midt på vinduet, med en knapp som tar brukeren til den gitte kundens skademeldinger. Til høyre i panelet kommer en liste med alle kundens forsikringer, kort oppsummert med forsikringsnummer,

forsikringstype og hvilken dato forsikringen ble opprettet. Hvert element i listen skal være klikkbart og ta brukeren til en egen side med informasjon om forsikringen.

Her er panelet oppdatert når man ser på en forsikring. Det grå feltet nederst i vinduet er for informasjon som varierer etter hvilken type forsikring som vises.

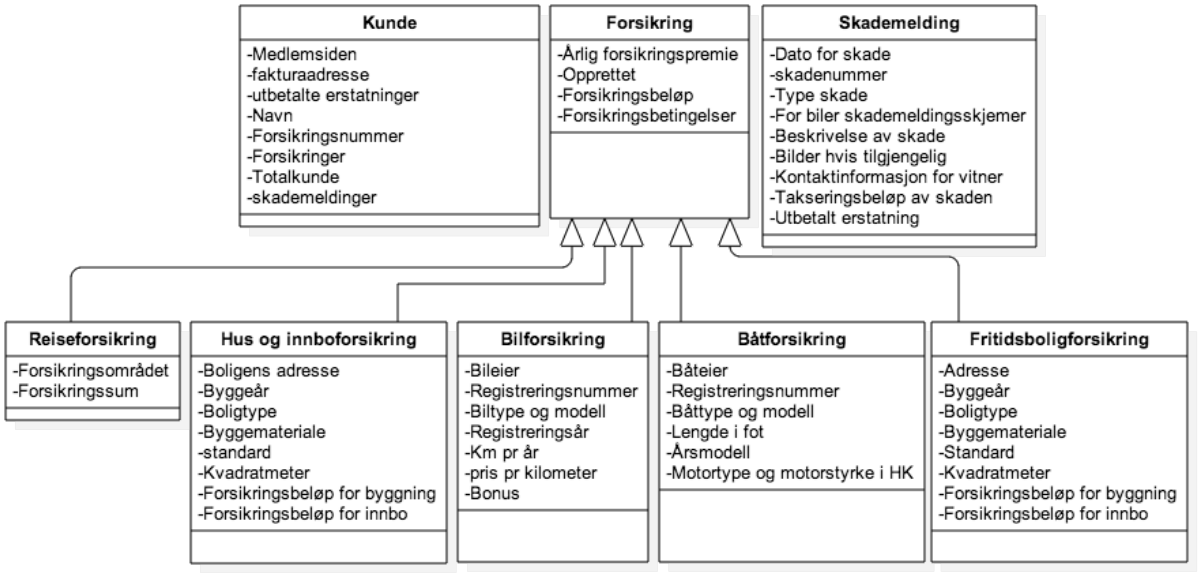


Over er hvordan panelet ser ut når brukeren har klikket seg inn på en skademelding. Feltet i midten er en JEditorPane med HTML-formatering med en beskrivelse og bilder av skaden.

Klassediagram

Grunnklasser

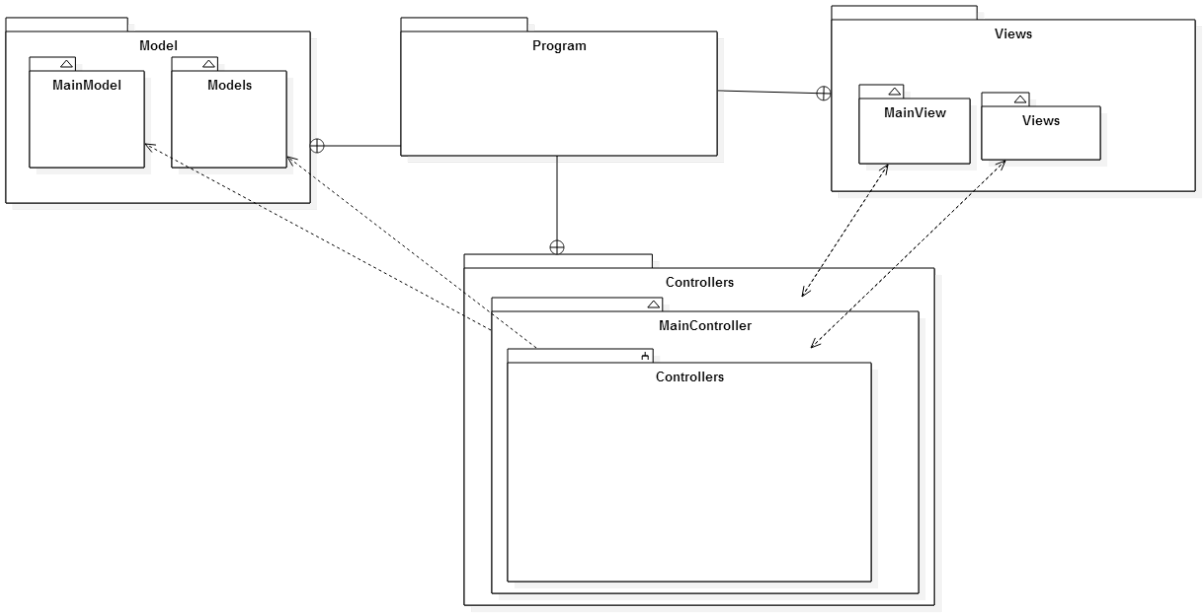
Systemet vårt vil bestå av følgende ”grunnklasser”. Disse inneholder kun informasjon.



Revision 1

MVC

Vi planlegger å bruke MVC strukturen i programmeringen for å skille mellom visningslogikk og datalogikk (Front/back-end). Hovedsakelig blir programmet vårt organisert slik:



Hovedprogrammet laster inn MainController, MainView og MainModel. Alle disse har forskjellige oppgaver.

MainModel
Inneholder alle registre som lagres på fil. Disse blir lastet inn og delt med MainController ved oppstart og lagret til fil ved programslutt.

MainView
Inneholder hovedframe til programmet.

MainController
Har som ansvar å holde styr på alle mindre kontrollere, views og modeller.

Model
Bruker registrene fra fil til å modellere bedriftens logikk. Håndterer hovedsakelig all data inn og ut av systemet.

Controller
Styrer handlingene til brukeren. Henter informasjon fra model og laster de inn i view.

View
Styrer alle GUI elementer og har ansvar for fremvisning av data.

Datastrukturer(Collections)

Vi kommer til å bruke collections til prosjektet. Disse brukes hovedsakelig til data som skal lagres til fil. Etter å ha kikket og vurdert de forskjellige typene collections har vi funnet ut av vi kommer til å bruke HashSet hovedsakelig i programmet. Slik programmet er delt opp vil det være bedre å sortere etter behov enn å ha dataene

lagret i for eksempel et TreeSet. Dette gjelder hovedregistrene hvor alle objektene lagers til fil.

I tilfellene hvor data ikke skal lagres til fil vil hvert enkelt bruk av collections bli vurdert for situasjonen.

Arbeidsfordeling

Vi har valgt å i hovedsak dele oss, slik at Rudi tar for seg back-end og Audun tar for seg view/GUI. Grunnen til at vi har gjort akkurat dette er at Rudi fra tidligere av har noe erfaring med MVC, og da slipper Audun å bruke «dyrebar» tid på å lære seg dette, og kan istedet sette seg inn i GUI-oppbygning. Selv om vi har delt oss på denne måten kommer vi fortsatt til å ha mye mer hverandres del å gjøre, og kommer til å samarbeide om alt helt til prosjektet er ferdig. Vi har planlagt å bruke «pair programming», som er en utviklerteknikk som går ut på at to utviklere jobber sammen på samme stasjon. En fungerer som «driver» og skriver koden, mens «observatøren» gjennomgår, dobbeltsjekker og kommenterer hver linje skrevet av driveren.

Utviklermiljø

Gruppen har blitt enige om å bruke følgende utviklerverktøy:

- En enkel editor av eget valg.
    - Brukes for kjapp tilgang til en enkel editor.
- Bruke Netbeans 8.0.2
    - Netbeans inneholder flere funksjoner vi gjerne vil bruke i prosjektet.
        - ActionList som gjør leting etter TO-DO kommentarer enkelt
        - innebygget GIT støtte,
        - Javadocs støtte
        - Mere
- Trello.com
    - Bra for organisering av hvem som gjør hva
- Git/Github
    - Program for versjonshåndtering som forenkler sammarbeid og håndtering av kode
- Skype
    - Brukes til for video/lyd-samtaler og sammarbeid
- Java SE DK 8
    - Java Standard Edition Development Kit 8 er Javas standard klassebibliotek

Systemkrav

Programmet kommer til å kreve en datamaskin med en Java SE versjon 1.8 installert og minstekravene for å kjøre Java SE 1.8

Javaversjon

Det er et krav at programmet kun benytter seg av Java SE 1.8. Programmet vil da kreve at Java SE 1.8 er installert.

Prosjektplan

Tidsperiode: 13.04.15 – 20.05.15
Antall uker: 5

Sprints

Vi deler prosjektets 5 uker inn i 5 ”Sprints”. En sprint sin livssyklus vil være slik:

Starten

Varighet: 1-2 dager
Beskrivelse: Her ser man over funksjonaliteten man har og funksjonaliteten man vil lage, diskuterer hva som fungerer og hva som ikke fungerer, prioriterer over hva som **må** gjøres og hva som **burde** gjøres. Man legger altså en miniplan over ”Sprinten”.

Hovedfase

Varighet: 3-4 dager
Beskrivelse: Her utføres koding/oppgaver etter prioritering fra starten. Hvis problemer dukker opp kan man se på det sammen eller utsette det til neste sprint avhengig av om det er en prioritet.

Slutten

Varighet: 1-2 dager
Beskrivelse: Her ser man hurtig over hva man gjort og sørger for at dokumentasjon/kommentering er i orden. Når alt er i orden er sprinten ferdig og kan analyseres.

Sprintoppdeling

I de 3 første sprintene kommer vi til å fokusere mest på koding mens i de 2 siste vil fokuseres mer på dokumentasjon. Fordelen med denne oppdelingen er at vi får en veldig fleksibel arbeidsmetode som gjør at vi kan arbeide så mye vi har tid til med prosjektet. De konstante revurderingen av fokus og prioriteringer gjør at vi vil ha en bedre oversikt hvilke prioritering som må tas.

Minstekrav

I første sprint kommer fokus til å være på å lage en prototype av programmet. Altså en enklest mulig versjon av programmet. Etter dette legger vi så på funksjonalitet litt etter litt. Fordi vi har lite erfaring med et prosjekt i en slik størrelse vil vi prøve å sikre god prioritering. Ved å lage en enklest mulig versjon, legge til litt funksjonalitet og revurdere hver uke vil vi kunne gjøre om på prioriteringene våre kjapt og sikre fokus på de viktigste kravene.

- Håndtere kunder(Opprette/redigere)
- Håndtere forsikringer(Opprette/redigere)
- Håndtere skademeldinger(Opprette/redigere)
- Utføre beregninger
    - Utbetalte erstatninger
    - Totalkunde
    - Bilbonus
- Lagring/lesing til/fra fil
- Søking

Milepæler

1. Fungerende prototype
2. Alle use-cases fungerende
3. All kode kommentert og fungerende uten bugs