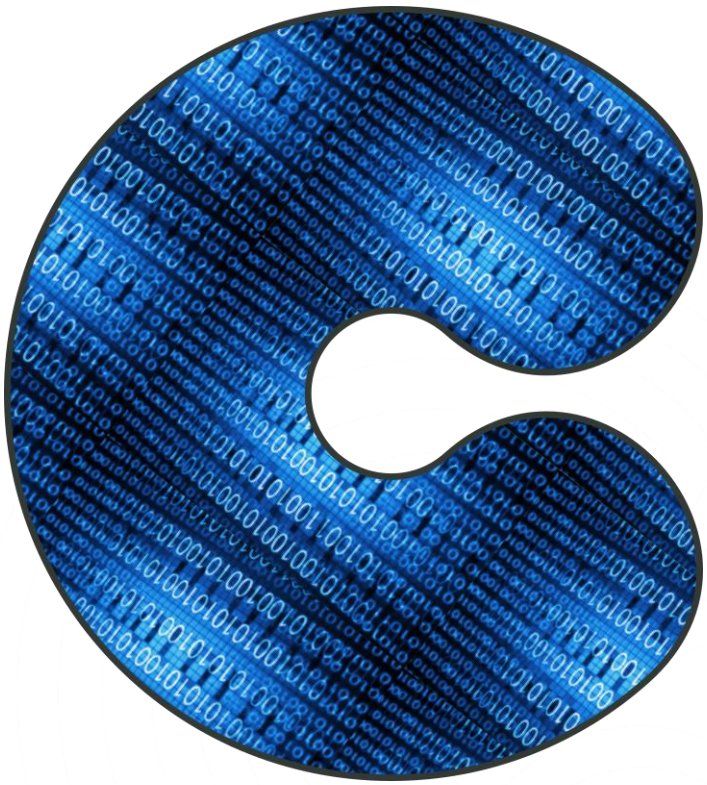


CREUNA DASHBOARD

Jesper Fosgaard – Januar 2015

Erhvervsakademi Sjælland Campus Roskilde – Datamatiker uddannelsen – Hovedopgave

INDHOLDSFORTEGNELSE

FORORD

Som afslutning på Datamatiker uddannelsen skal der udarbejdes en hovedopgave, der består af en rapport og en efterfølgende mundtligt eksamination. Denne rapport markerer slutningen på de 5 semestre uddannelsen har varet. Selve undervisningen er forgået på Erhvervsakademi Sjælland Campus Roskilde. De første 4 semestre bestod af undervisning i systemudvikling, systemdesign, og 2 vilkårlige valgfag, som i mit tilfælde var Android udvikling og Spil udvikling. Det 5 semester består af et obligatorisk praktikophold, og aflevering af hovedopgave.

Jeg havde mit praktik ophold i virksomheden Creuna Danmark, der er beliggende på Hammerensgade 4 på Østerbro. I forlængelse af mit praktik ophold var jeg så heldig at få lov til at udarbejde min hovedopgave i sammenarbejde med Creuna.

Jeg vil gerne sende en stor tak til Creuna og dets medarbejdere. Først og fremmest for at jeg overhoved fik praktikpladsen, og efterfølgende fik lov til lave min hovedopgave der. Dernæst fordi at de har behandlet mig ligeværdigt, og været utroligt hjælpsomme. Samtidig med at de har haft rigeligt at se til, så har de også haft tid til at lære fra sig, og delt de erfaringer de har gjort sig i hverdagen. Det har været en inspiration af være en del af Creuna.

1. PROJEKTETABLERING

1.1 Indledning

I dette afsnit vil jeg beskriver selve omfanget af opgaven. Det involverer følgende områder.

- En beskrivelse af produktet
- Krav til produktet
- En tidsplan for projektet
- En beskrivelse af problemet

1.2 Introduktion til projektet

Jeg har fået til opgave at udarbejde et Dashboard, til en allerede eksisterende web applikation kaldet Creuna ProjektPortal. Creuna ProjektPortal er en portal der samler flere eksisterende web applikationer til en applikation. Det drejer sig bl.a. om et time registrerings værktøj ved navn TimeGrip, og et gammelt service site, hvis opgaver ProjektPortalen har overtaget.

Portalen skal bruges af Creunas egne medarbejdere og Creunas kunder. Portalen er udviklet af, Senior System Udvikler Michael Eiland, praktikant Chakib Benhaddou, og praktikant Jesper Fosgaard. Den er udviklet i Asp.net MVC 5 med Entity Framework Code first. Back-enden er skrevet i c# og front-enden er skrevet i HTML, CSS, JavaScript, AngularJS, samt Razor.

Dashboardet er en videreudvikling af ProjektPortalen, og skal fungere som bindeled for al den data der eksistere i applikationen. Internt skal Dashboardet være med til at skabe overblik og indsigt i de projekter og sager som Creuna er involveret i. Eksternt skal det bruges af Creunas kunder, til bl.a. at se status på igangværende sager, samt at oprette sager. Derudover skal det være muligt at se kontrakt informationer, og de involverede interessenter.

1.3 Projektpartner

Min partner i dette projekt er Creuna Danmark A/S. Creuna er et af Danmarks førende digitale bureauer, og er en del af Skandinavien førende digitale bureau Creuna A/S. Der er mere en 330 medarbejdere fordelt i Danmark, Sverige, Norge og Finland. I Danmark er er der ca. 90 medarbejdere, som er fordelt på to kontorer. Det ene ligger i Århus på Vesterbro Torv 1-3. Det andet ligger på i København på Hammerensgade 4 på Østerbro. Det er i sammenarbejde med afdeling i København at jeg skal udarbejde produktet og rapporten.

1.4 Oplæg fra projektpartner

1.5 Kravspecifikation

Der er fra Creuna stillet nogle krav til udviklingen af DashBoardet. Her er tale om funktionelle krav og ikke-funktionelle krav. De funktionelle krav, er en liste med krav, over de funktioner, som Creuna ønsker at applikationen skal kunne. De ikke-funktionelle krav, er krav der går ud over funktionaliteten, men som stadig har stor betydning for projektet. I dette afsnit vil jeg beskrive de forskellige krav.

1.5.1 FUNKTIONELLE KRAV

De funktionelle krav er delt op i to kategorier, "Need to have" og "Nice to have". Need to have, er krav der skal være opfyldt, og Nice to have, er krav der ville være rare at have, men som ikke er nødvendige for at applikationen kan bruges. Ydermere er der underkategorier til hver kategori, som er fælleskrav, projektlederkrav, udviklerkrav og kundekrav. Grunden til at der er forskellig krav for de forskellige brugere, er fordi at de tre brugergrupper vil være interesseret i forskellige typer data. Der vil da også være krav som er fælles og de er beskrevet i det tilhørende afsnit

Need to have

- Fælleskrav
    - DashBoardet skal bestå informationsmoduler
    - Hver modul skal indeholde en række data, som enten skal vises som en liste, eller være en visualisering af data i form af en graf eller et diagram.
    - DashBoardet skal kun bestå af data der er relevante for brugeren. Det vil sige at en Creuna medarbejders Dashboard skal vise de data og moduler der er relevant for ham/hende og en kundes DashBoard skal vise de data og moduler der er relevant for ham/hende.
    - I hver modul skal det være muligt at få yderligere information omkring den data som er repræsenteret. Dette kan ske ved at enten at klikke på forekomsten eller holde musen over forekomsten.
    - Der skal være en separat menu, hvor det skal være muligt at tilgå resten af ProjekPortalens sider.
    - Der skal anmodes om et login når en bruger prøver at tilgå DashBoardet.
- Projektlederkrav
    - Projektlederen skal kunne se en oversigt over de projekter han/hun er involveret i.
    - Projektlederen skal under hver projekt kunne se hvilken fremgang der har været på de opgaver der er på projektet.

    - Projektlederen skal kunne se en liste med de opgaver han/hun har, sorteret efter hvilke opgaver der er mest kritiske.
    - Projektlederen skal kunne se en liste over de opgaver hvor han/hun ikke har registreret timer på.
- Udviklerkrav
    - Udvikleren skal kunne se en oversigt over de projekter han/hun er involveret i.
    - Udvikleren skal under hver projekt kunne se hvilken fremgang der har været på de opgaver der er på projektet.
    - Udvikleren skal kunne se en liste med de opgaver han/hun har, sorteret efter hvilke opgaver der er mest kritiske.
    - Udvikleren skal kunne se en liste over de opgaver hvor han/hun ikke har registreret timer på.

- Kundekrav
    - Kunden skal kunne se en oversigt over projektet. Oversigten skal indeholde en liste over de personer der er involveret i projekt.
    - Kunden skal kunne se status på de sager der ligger under projektet. Både sager der er lukkede, igangværende og kommende.

Nice to have

- Fælleskrav
    - Hvert modul kan som kan organiseres, som brugeren ønsker det.

1.5.2 IKKE-FUNKTIONELLE KRAV

De ikke funktionelle krav er krav der beskriver nogle begrænsninger for projektet og applikationen. Det drejer sig om versionsstyring, sikkerhed og release datoer.

Versionsstyring

- Der skal bruges et versionsstyringsværktøj, så at alle versioner af applikationen nemt kan integreres. Der udover skal det fungere som en sikkerhedsline i forhold til at løse runtime fejl.

Sikkerhed

- Det skal ikke være muligt at tilgå nogle af Dashboardets sider eller informationer uden den rettet autorisation.

Release datoer

- Der skal releases en fungerende kopi af applikationen, minimum en gang hver anden uge, gerne en gang om ugen hvis muligt.
- Den endelige release dato for det færdige projekt er fredag den 19. december

1.6 Problemformulering

1.6.1 PROBLEMSTILLING

Hvis man som Creuna medarbejder eller kunde af Creuna, vil have overblik over fremgang på sager eller information om disse. Så skal man på nuværende tidspunkt søge disse informationer i forskellige systemer.

Der er meget få af disse processer der er automatiseret. Det betyder at man i høj grad skal være aktiv for at holde sig opdateret omkring en specifik sag eller løsning. Derudover er det meget svært at danne sig et overblik over hvad status på alle sagerne. Der er derfor behov for at udvikle et DashBoard så man nemt og hurtigt kan se fremdriften på de sager man er involveret i.

1.6.2 UDDYBENDE SPØRGSMÅL

- Er det muligt at lave en fuld funktionel brugergrænseflade og back-end til Dashboardet, der opfylder de krav Creuna har sat, inden for tidsfristen?
- Hvordan kan man udvikle et DashBoardet for Creuna?
- Hvilken udviklingsmetode er Anvendelig?

1.7 DELKONKLUSION FOR PROJEKTETABLERING

Der er i projektetableringen blevet skabt et rigtig godt fundament for den videre udvikling af DashBoardet. Især de meget specifikke krav var med til at hjælpe på forståelsen af problemet. Det skal blive spændende at se hvilke udfordringer der komme i de videre forløb.

2. PROBLEMLØSNING

For at løse problemstillingen vil jeg gøre brug af forskellige metoder, værktøjer og teknologier. Jeg vil i dette kapitel beskrive de metoder der gøres brug af. Efterfølgende vil jeg også beskrive de værktøjer der bruges i projektet, samt de teknologier der anvendes.

2.1 Udviklingsmetoder

2.1.1 INDLEDNING

Der findes mange forskellige udviklingsmetoder og de har hver især deres styrker og svagheder. Jeg har i via mit studie erfaring med to forskellige udviklings metoder. Det drejer sig om Unified process og

Agile systemudvikling med Scrum som process styringsmetode og Extreme Programming (XP) som udviklingsmetode.

2.1.2 VALG AF UDVIKLINGSMETODE

Når det kommer til valg af udviklingsmetoder var spørgsmålet ikke om det skulle være UP eller en Agil metode. Det var mere et valg mellem hvilke af de agile metoder vi skulle vælge. Jeg har i forvejen arbejdet med Scrum og XP på ProjektPortalen, og med gode resultater. Da DashBoardet er en videre udvikling af dette, vil det umiddelbart være mest logisk at fortsætte med Scrum og XP. Men fordi det fungere godt på et projekt, er det jo ikke ens betydende med at det ville fungere på et andet. Derfor har jeg kigget på, om Scrum og XP nu også var det rette for dette projektet. Jeg startede med at skrive en liste med de krav jeg og Creuna havde til projektet og se om det kunne hjælpe mig i valget.

Krav til udviklingen

- Det skal være let og hurtigt at komme i gang med da der ikke er tid eller behov til store analyser af systemet.
- Det skal passe til en gruppe på 3-4 mand. Men må ikke være låst til et antal hvis behovet for mere mandskab skulle opstå.
- Det skal være nemt for alle interessenter at få overblik over fremgangen i system, også selvom de ikke har kendskab til programmering.
- Der skal leveres fungerende software hver uge.
- Det skal være let at lave ændringer i projektet.

Efter jeg havde stillet disse krav op, var det ret klart hvilken metode der skulle vælges. Den agile tilgang blev valgt, med Scrum som process styrings framework, og XP som udviklingsmetode. Det gjorde den fordi at de ting som Scrum indeholder, passer rigtig godt til de krav der stillet i forhold til projektstyring. Det passer bl.a. til kravet om at det skal være hurtigt at komme i gang med, og at der ikke skal være nogen grundig analyse af systemet og virksomheden. Det passer også godt i forhold til størrelsen på det team der skal udvikle DashBoardet, og på de kompetencer de besidder. Når det kommer til kravet om at det skal være nemt for alle interessenter at få overblik over fremgangen i projektet. Så er Scrum også ideel, i det at der i Scrum bliver prædiket om gennemsigtighed. Den gennemsigtighed skulle gerne være med til at udelukke eventuelle tvivlspørgsmål, da alle i teorien burde have adgang til samme viden om projektet.

Når det kommer til selve udviklingen af applikationen så vil der blive benytte forskellige praktikker fra XP. Når man bruger XP som udviklingsmetode, så opstår spørgsmålet om hvilke praktikker der skal benyttes ofte. For nogle virksomheder er processen måske beskrevet meget klart og man har nogle faste måder at gøre det på. Andre virksomheder har måske nogle faste praktikker som de altid bruger, men til eller fravælger så praktikker alt afhængigt af hvilken type projekt det er. I forhold til at kunne levere

software hver uge der virker, så vil gøres der brug at en praktik der hedder Continuous Integration. Derudover vil vi benytte os af Pair Programming.

2.1.3 SCRUM

Scrum er et agilt system udviklings framework, der bruges til process styring. Scrum er relativt let at lære, og hurtigt at komme i gang med. Det kræver selvfølgelig at der er nogen der har kompetencer til at styre hele processen. Samtidig kræver det at de personer der er involveret i udviklingen, er engageret i hele processen og er villige til at tilpasse sig og lære. Scrum ligger i den grad vægt på det menneskelige aspekt i Udvikling.

Det Agile Manifest beskriver bl.a. følgende:

*"**Individer og samarbejde** frem for processer og værktøjer"*

Dette statement fortæller i høj grad hvad Scrum går ud på. Det primære i agil udvikling er sammenarbejdet mellem de individer der er involveret i projektet, og ikke de metoder og værktøjer der bliver benyttet. Det er ikke kun mellem Product Owner og Development Teamet at dette gælder. Det gælder i lige så høj grad i sammenarbejdet med de eksterne partnere.

Scrum er et Iterativ og inkrementel Framework. Det Iterative består i noget der i Scrum kaldes Sprint. Et sprint er et tidsinterval der kan vare fra en uger en måned, i nogle tilfælde kan det være længere. Men det anbefales ikke at et Sprint vare længere end en måned da det kan gå ud over de principper som Scrum er bygget på. I disse sprints foregår der forskellige aktiviteter, aktiviteterne er beskrevet i afsnittet Scrum Aktiviteter som findes længere nede på siden. Det inkrementel aspekt ligger i at der som minimum i slutningen af sprintet leveres fungerende kode. Derved bliver der tilføjet flere funktioner til applikationen. I Scrum er der tre kategorier, som er Scrum roller, Scrum Artefakter, og Scrum Aktiviteter. De tre kategorier har så hver deres virkeområde og indeholder hver især nogle forskellige elementer. Disse kategorier er og deres elementer er beskrevet i de kommende afsnit.

2.1.3.1 Scrum Roller

Product Owner

Der vil på projektet være tilknytte en Product Owner. En Product Owner har ansvaret for at administrere Product Backloggen. Det vil sige at det er Product Owners ansvar at der bliver udarbejdet Backlog Items og at disse Backlog Items er udført med den korrekte information. Det er også Product Owners ansvar at administrere Product Backloggen og Sprint Backloggen. I det ligger det at der konstant skal ligge Backlog Items i Product Backloggen, så disse kan bliver overført til Sprint Backloggen når det er aktuelt. Ydermere er Product Owner også ansvarlig for at de Backlog Items der ligger i Sprint Backloggen, er items der er af højeste prioritet, og at Sprint Backloggen indeholder den mængde Backlog items som Development Teamet kan gennemføre i det givende Sprint

Det er Product Owner der har ansvaret for udarbejdelse af Backlog Items, og for administration af eventuelle ændringer. Det er som sådan ikke nødvendigt at det er Product Owner der udfører arbejdet, det kan lige så godt være en anden fra organisationen eller mere nærliggende en eller flere fra Development Teamet. Men Product Owner har det overordnede ansvar for at tingene bliver udført korrekt.

Development Team

Development Teamet er det team af udviklere der udarbejder de features, der er beskrevet i de forskellige Backlog Items Det er et team som oftest består af 3-9 Udviklere, det er eksklusiv Product Owner og Scrum Master. Dette spæn er på ingen måde et krav, men en anbefaling man ofte ser i diverse Scrum Guides. Er antallet af udvikler under tre, kan der opstår problemer i forhold til at opretholde et fast udviklings tempo. Der skal kun være en mand som ikke er til stede på projektet før at halvdelen af teamet er væk. Der er simpelthen for stor usikkerhed i kun at være to. Derudover kræver det at de begge to stort set besidder de samme kvalifikationer. Hvis ikke så kan der opstå et scenarie hvor at personen der er fraværende, er den eneste i det team der har de fornødne kvalifikationer til at løse en specifik opgave. Er teamet på over ni udviklere, så er man der hvor at det begynder at blive en for stor opgave for en Product Owner at administrere. Der kan man for at løse problemet, dele gruppen op to mindre Development Teams, og have en Product Owner tilknytte til hver af dem. Det kræver selvfølgelig at der rent organisatorisk er muligt at have to Product Owners allokeret på samme projekt

Scrum master

En Scrum Master er en person i organisationen der har flere opgave at varetage. Scrum Masterens primære opgave er at udbrede kenskabet til Scrum, til Development Teamet, Product Owner og organisationen. De tre person grupper har forskellig gavn af Scrum Masteren. Når det kommer til Development Teamet. Så er det primære, at sørge for at de kan arbejde uhindret, og at opklare eventuelle misforståelser eller mangler. Udover at bistå med vide omkring Scrum, så er det også Scrum Masters opgave at facilitere de forskellige Scrum Aktiviteter, som Daily Scrum og Sprint Planning Meeting. Når det kommer til Product Owneren, så er det primære, at videreformidle Product Owners visioner til Development Teamet. Derudover kan Scrum Masteren komme med forslag til teknikker og værktøjer der kan bruges til at styre af Product og Sprint Backloggen. Til sidste så er der også Scrum Masterens opgave at udbrede Scrum til organisationen, samt at arbejde sammen med andre Scrum Mastere i organisationen, hvis sådanne findes.

2.1.3.2 Scrum Artefakter

Der er to Artefakter i Scrum. Det er Product Backlog og Sprint Backlog. For at forklare hvad Product Backlog og Sprint Backlog er, så er der nogle andre begreber man skal kende til. Det er begreber som Scrum Board og Backlog Item. Et Scrum Board er en tavle som består af nogle Inkrementale skridt. Den

er ofte fysisk repræsenteret i det område hvor Development teamet sidder. Den kan også være digitaliseret, så man har sit Scrum Board i en Applikation. Ofte er brugen af begge dele faktisk tilfældet. Ved at bruge det analoge Scrum Board, kan man hurtigt få overblik over projektet og fremgangen. Det eneste det kræver er at tage et hurtigt kig på tavlen. Ved at bruge et digitalt Scrum Board så opnår man nogle andre fordele. Det er bl.a. at man kan se historik på sine sprints så man kan analysere på sin arbejdsproces. Man kan også helt automatisk få udregnet et Burndown Chart, som er en visualisering af det manglende arbejde og den tid der er tilbage af sprintet. Derudover er der den fordel, at man ikke fysisk behøver at være tilstede for at se Boardet og de Backlog Items der er placeret på det.

Product Backlog	Sprint Backlog	In progress	Review	Test	Done

Et simpelt eksempel på et Scrum Board

Som man kan se på illustrationen, så er Boardet opdelt i skridt. I starten er der en kolonne der hedder Product Backlog. Det er her Product Backlog er repræsenteret. De efterfølgende kolonner tilhører Sprint Backloggen. Disse kolonner kan variere fra projekt til projekt og fra virksomhed til virksomhed.

Backlog Item

Et Backlog Item kan i vores optik være to forskellige ting. Enten en Feature eller en Bug. En feature er en ny funktion der skal tilføjes til applikationen. En Bug er en fejl i applikationen der skal rettes. Et Backlog Item indeholder mange forskelige informationer. I dens simpleste for er det repræsentere ved en User Story. En User story er skrevet ud fra et bruger perspektiv, og kan se ud på følgende måde

”**Som en** (Fodboldtræner), **vil jeg** (se en liste med tilmeldte spiller til dagens træning), **så at jeg kan** (tilpasse mine træningsøvelser til antallet af deltagere) ”

Ud over historien, så består den også af en Business Value. Det er en værdi som Product Owner fastsætter, og er en værdi der markere hvor værdifuld featuren er. Jo højre værdi jo mere værdifuld er den. Ud over historien så skal Der også beskrives nogle Accept Kriterier. Det er nogle kriterier der skal være opfyld før at man kan erklære featuren for færdig. Et Backlog Item beskriver ikke hvordan, featuren skal udvikles. Men er blot en beskrivelse af hvad formålet er med den.

Product Backlog

En Product Backlog er en liste af ordnede Backlog Items. Det Backlog Items som har den højeste Business Value er den der ligger øverst. I Product Backloggen ligger alle de Backlog Items der er beskrevet indtil videre. I starten vil der som ofte ikke være særligt mange Backlog Items. Men

efterhånden som features bliver udviklet, så kommer der flere til. Product Backloggen er dynamisk. Det vil sige at den konstant ændre sig i forhold til antallet af Backlog Items og deres rækkefølge.

Sprint Backlog

Sprint Backloggen er der hvor de Backlog Items der skal udføres befinder sig. Disse Backlog Items gennemgår en rejse fra Sprint Backloggen og til Done. Det skulle gerne være sådan et alle Backlog Items når til Done stadiet inden sprintet er udløbet. Hvis ikke det er tilfældet, så må man justere hvor mange Backlog Items man har med i sine sprints.

2.1.3.3 Scrum Aktiviteter

Sprint Planning Meeting

Et Sprint Planning Meeting er et møde hvor det kommende Sprint planlægges. Hele Scrum Teamet deltager i dette møde. Formålet med mødet er bl.a. at få en fælles forståelse for de features der skal laves, samt at udvælge features til sprintet. Det er Product Owners opgave at præsentere de forskellige features der ligger i Product Backloggen. Det er så Development Teamets opgave at vælge hvor mange af de Backlog Items der bliver præsentere de vil have med i sprintet.

Daily Scrum

Daily Scrum er et møde af 15 min varighed som afholdes hver dag. Hvornår på dagen er ikke så vigtig, det vigtigste er at det bliver holdt. Det går i korte træk ud på at Development Teamet og Scrum Master mødes. Det er et meget uformelt møde der ofte udføres stående, eventuelt ved Scrum Boardet. Derfor kaldes det også af nogen for Stand Up meeting. Når Teamet så er samlet så fortæller hver mand hver især hvad man har lavet siden sidste gang man mødtes og hvad man skal lave til man mødes næste gang. På mødet kan man også tage problemer op man er støt på. Det må dog ikke gå hen og blive en langvarig diskussion, så Scrum Masteren skal sørge for at alle for taletid og at ingen taler for længe.

Sprint Review

Et Sprint Review er et møde sidst i sprintet. Det er af ca. 1 times varighed hvis sprintet varer en uge. Er sprintet længere øges længden af mødet eksponentielt. På mødet som er uformelt, snakkes der om hvad der er gået godt i det sprint der er ved at være overstået, og hvad der kan gøres bedre i næste. Samtidig kigger Product Owner på om de alle Backlog Items der er færdige, lever op til det man kalder definition of done. Definition of done er en beskrivelse på hvornår et Backlog Item kan ses som færdig. Det vil ofte være når et Backlog Items Accept Kriterier er opfyldt. Er der features som ikke er færdig, så snakkes der også om hvorfor det ikke lykkedes at udvikle det ønskede antal features. Der kan også blive diskuteret om de teknologier og værktøjer som Development Teamet bruger. Det sker ofte at et Development Team tilføjer nye teknologier og værktøjer til projektet. Så længe at det har en værdi for Teamet og at produktiviteten ikke falder, så bydes al forandring og nytænkning velkommen.

2.1.4 Extreme Programming

Der vil sammen med Scrum blive bruget nogle praktikker fra Extreme Programming. Extreme Programming er en software udvikling metode der har til formål at skabe bedre og mere effektiv udvikling, og bedre produkter. Der findes mange forskelige praktikker i XP, men teamet har valgt at fokuserer på to, som vi mener vil skabe værdi for os og virksomheden.

2.1.4.1 Pair Programming

Pair Programming er en disciplin hvor to udviklere sidder udvikler sammen. Den ene person sider og koder, den anden sidder og observere. Der er selvfølgelig både fordele og ulemper ved Pair Programming. En af fordelen er at man ved at være to, kan minimere fejl. Man har to sæt øjne på samme kode. En anden fordel er at man lære en masse ved at sidde sammen med en anden og kode. Ikke to udviklere koder ens, og der er som ofte forskelige løsninger på alle problemer. Derudover giver det en følelse af fællesskab omkring koden. Det er det man i XP kalder for Collective Code Ownership. En af ulemperne er der er to mand på samme opgave, derved vil man i teorien udføre halvt så meget arbejde. Der hersker dog en del tvivl omkring hvorvidt denne teori også er helt korrekt. For selvom det muligvis er rigtigt at der vil blive lavet mindre, så bliver der også samtidig lavet mindre fejl. I sidste ende skulle de færre fejl resultere i mindre arbejde med fejlrettelser, og dermed ville det tabte arbejde ved at være to blive udlignet.

2.1.4.2 Continuous integration

Continuous integration går ud på at man hver dag flere gange om dagen integrere det arbejde man har lavet med det andre i teamet har lavet. På den måden skulle man gerne undgå at man ender i et integrations helvede, fordi at der er flere dage ellers ugers arbejde der skal integreres. Man tilføjer også konstant værdi til projektet og gør det nemmere at release i slutningen af sprintet.

2.2 Udviklingsværktøjer

Der vil på projektet blive benytte en del forskellige værktøjer. Det drejer bl.a. om forskellige typer software og forskellige Frameworks. I dette afsnit vil jeg gennemgå de forskellige programmer og frameworks der gøres brug af.

2.2.1 SOFTWARE

For at udvikle DashBoardet benyttes der en række forskellige programmer. Det er programmer der på den ene eller anden måde gør det muligt at udvikle DashBoardet og samtidig overholde de krav de er stillet af Creuna.

Vi gør brug af følgende programmer.

- **Visual Studio 2013** – Dette er motoren i hele projektet. Visual Studio er hvor al kode bliver produceret. Det er også her at version styringen forgår.

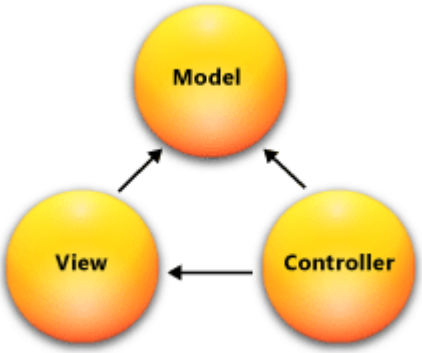
- **Visual Studio Online** – er et online process styrings værktøj, hvor man kan planlægge sprint, oprette Backlog items. Det fungere også som et digitalt Scrum Board, med alle de muligheder der hører med.
- **SQL Server Management Studio** - er et værktøj til at styre sin SQL server og det data der ligger i den.
- **GIT** –er et versionsstyringsværktøj, der i sammenarbejde med Visual Studio online, vedligeholder de forskellige branches der bliver udviklet på.
- **Azure** – Azure er et cloud baseret server miljø som benyttes på projektet. Hele løsningen ligger i skyen, undtagen udviklingsmiljøet.

2.2.2 UDVIKLINGS FRAMEWORK

Vi vil på projektet gøre brug af en række forskellige Udviklings Frameworks. Da DashBoardet er videreudvikling af ProjektPortalen, vil det give god menig at fortsætte med at bruge de frameworks der allerede bliver brugt. Jeg vil i dette afsnit gennemgå de forskellige frameworks og vores brug af dem.

2.2.2.1 Asp.net MVC

Asp.net MVC er Microsofts version af MVC Frameworket.
MVC (Model-View-Controller) er delt op i tre lag. Modellaget som er her vi har al vores forretningslogik. Viewlaget er det lag der visualisere de forskellige elementer og informationer. Controllerlaget er det lag hvor logikken ligger. Model laget er der hvor data klasser er defineret.



ASP.net MVC Frameworket blev valgt af flere årsager. For det første er det meget hurtigt at komme i gang med. I Visual Studio er det muligt at få rigtig meget foræret når man opretter et nyt MVC site. Det drejer sig bl.a. om standard CRUD funktioner, Brugerstyring og automatisk implementering af test.

2.2.2.2 Entity Framework 5 Code First.

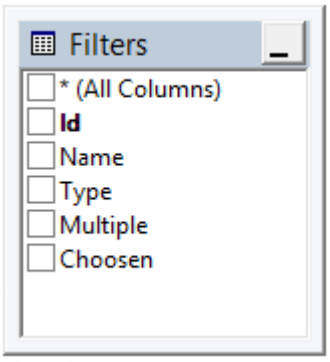
Entity Framework er en Object-relational mapper (ORM). ORM er teknik der bruges til at konvertere data. Det kan eksempel vis være en række i en tabel i ens database, der bliver konverteret til et objekt i ens kode. Entity Framework gør det let at arbejde med data og man behøver som udgangspunkt ikke at rode rundt i en database. Entity Frameworket fungere med mange typer databaser, som bl.a. MSSQL og MySQL. Det betyder at man ikke bundet af at have en speciel type database, for at kunne bruge Entity Frameworket. På projektet benyttes en teknik der hedder Entity Framework Code First. Code First er en

teknik hvor man skriver sine data klasser første og så får man Entity Frameworket til at oprette tabeller i databasen der passer til de data klasser man har defineret.

Nedefor ses et eksempel på en klasse der først blev defineret i Model laget og derefter ved hjælp af Code First migreret til databasen.

```
public class Filters
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Type { get; set; }
    public bool Multiple { get; set; }
    public bool Choosen { get; set; }

}
```



I Entity framework Code first er det også super nemt at opdatere sine tabeller. Her skriver man blot en ny parameter ind i den tilhørende data klasse og derefter eksekvere man en opdatering ved hjælp af en kommando i Visual Studio Package manager konsollen.

2.2.2.3 Razor

Razor er den standard View Motor som ASP.Net gør brug af. Razor fungere som en template og er ikke bundet af ASP.net MVC Frameworket. Man kan derfor bruge Razor i alle applikationer der gøre brug af en View Template. Razor gør det utroligt let at vise data, og det kræver ikke meget kode at gøre ret komplekse ting. Det som Razor motoren gør er at den tager Razor syntaksen og kompilere den til C# og HTML. For at starte en Razor sætning bruger man et @ tegn. Det gør at Razor motoren ved hvad den skal kompilere. Razor motoren er smart og den kan kende forskel på hvad der er Razor og hvad der er standard HTML, også selv om de forskellige syntakser er på samme linje eller i samme kode blok.

Neden for ses et eksempel på en løkke skrevet i Razor. Det første der sker er at man instantiere noget data som templaten skal bruge. I dette tilfælde er det en liste af objekter af typen TeamProject.

```
@model IEnumerable<PFMWeb.Models.PFM.TeamProject>

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Name)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.TeamProjectId }) |
            @Html.ActionLink("Details", "Details", new { id=item.TeamProjectId }) |
            @Html.ActionLink("Delete", "Delete", new { id=item.TeamProjectId })
        </td>
    </tr>
}
```

Dernæst kommer så løkken. Det er en simpel foreach løkker der itererer over listen af TeamProject objekter. Foreach løkken indeholder både Razor og HTML elementer, som er en tabelrække (<tr>), to tabelfelter (<td>), en tekst (@Html.DisplayFor()), og tre links (@Html.ActionLink()).

Outputtet er så som vi ser nedenfor. En tabel indeholdende de elementer som vi definerede i vores View.

FHCBooking	Edit | Details | Delete
Falck Danmark	Edit | Details | Delete
Falck Applus	Edit | Details | Delete

Ud over Razor bruges også AngularJS til at visualisere data i de forskellige views. AngularJS vil blive beskrevet i et næste afsnit. Grunden til at bruge to forskelligt måder at vise data på er at Razor er super godt til hurtigt og nemt at vise data fra en datakilde. Skal man bruge flere kilder, så er AngularJS langt mere brugbar. Der ud over, har det en række andre fordele.

2.2.2.4 AngularJS

AngularJS er et JavaScript framework der giver en mulighed for at lave dynamisk web apps og skabe dynamisk indhold på eksisterende web applikationer. AngularJS er et framework der implementer MVC på client-side, og giver en muligheden for at lave det man kalder single page applikations. En singlepage applikation er en applikation, hvor man kun laver et enkelt request til serveren. Når serveren har responderet, tager AngularJS over og står for al interaktion mellem de forskellig lag.

Forskellen på AngularJS og Razor som også bliver brugt i applikationen er at hvor Razor bygger sin egen HTML ud fra Razor templaten. Så udvider AngularJS de eksisterende HTML tags, ved at tildele properties til de eksisterende tags. Dermed kan man skabe dynamisk indhold og binde data til de klassiske statiske HTML tags.

Nedefor ses et eksempel på et <td> tag, der får sat en dynamisk colspan, ved at data-binde en værdi til HTML property

```
<td colspan="{{customerCaseOutput.columns.length}}">
```

På Dashboardet bruges AngularJS til dynamiske elementer der skal opdateres uden at selve siden bliver opdateret. Razor vil blive brugt til det dynamiske data der ikke kræver opdatering før siden bliver genindlæst.

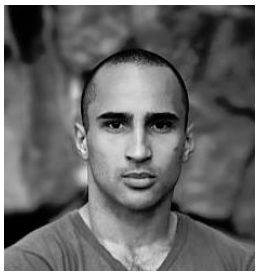
2.2.2.5 HTML, CSS og JQuery

Der vil på Dashboardet blive brugt en blanding af HTML, CSS og JQuery til at skabe, organisere og style indhold på Dashboardet.

2.3 Ressourcer

2.3.1 MENNESKELIG RESSOURCER

Der er på projektet tilknytte en rækker personer der udfylder forskellige roller. Her er en kort præsentation af disse personer og deres ansvarsområder.

Navn: Christian Jacobsen **Titel:** Head of technology **Ansvarsområde:** Christian vil fungere som Product Owner på projektet	
Navn: Michael Eiland **Titel:** Senior System Developer **Ansvarsområde:** Michael vil fungere som Scrum Master på projektet. Der udover vil han og bistå i udviklingen af DashBoardet, så frem det skulle blive aktuelt	
Navn: Chakib Benhaddou **Titel:** System Udviklings praktikant **Ansvarsområde:** Chakib har til ansvar at udvikle Dashboardet i sammenarbejdet med resten af teamet	
Navn: Jesper Fosgaard **Titel:** System Udviklings praktikant **Ansvarsområde:** Jesper har til ansvar at udvikle DashBoardet sammen med resten af udviklingsteamet	

2.3.2 TIDSRESSOURCER

Hver sprint varer 1 uge, mandag til fredag. Der har på forhånd nogle klare punkter i hver sprint, men hvad der skal arbejdes med i de forskellige sprint bliver først klarlagt til Sprint Planning Meeting. Arbejdstimerne er ikke faste, og det kan gøre at ikke alle er tilstede samtidig. Det er dog vel set, at man

er til stedet i tidsrummet mellem kl. 08.00 og 18.00. Hvordan man så vælger er planlægge ens tid er op til en selv. Der er dog de obligatoriske møder som man skal deltage i, med mindre man selvfølge har en valid grund til at melde fra. Et typisk sprint vil se sådan ud.

Mandag: Vi mødes kl. 09.00. Her afholdes et Sprint Planning Meeting. Der er afsat to timer til dette møde. Her aftales hvilke features fra Product Backloggen der skal med i sprintet.

Tirsdag: Her møder man ind nå man har lyst, dog senest 09.45 hvor Daily Scrum går i gang. Her snakker vi om hvordan dagen i går gik og lidt omkring hvad resten af dagen skal bruges på. Mødet varer maks. 15 min.

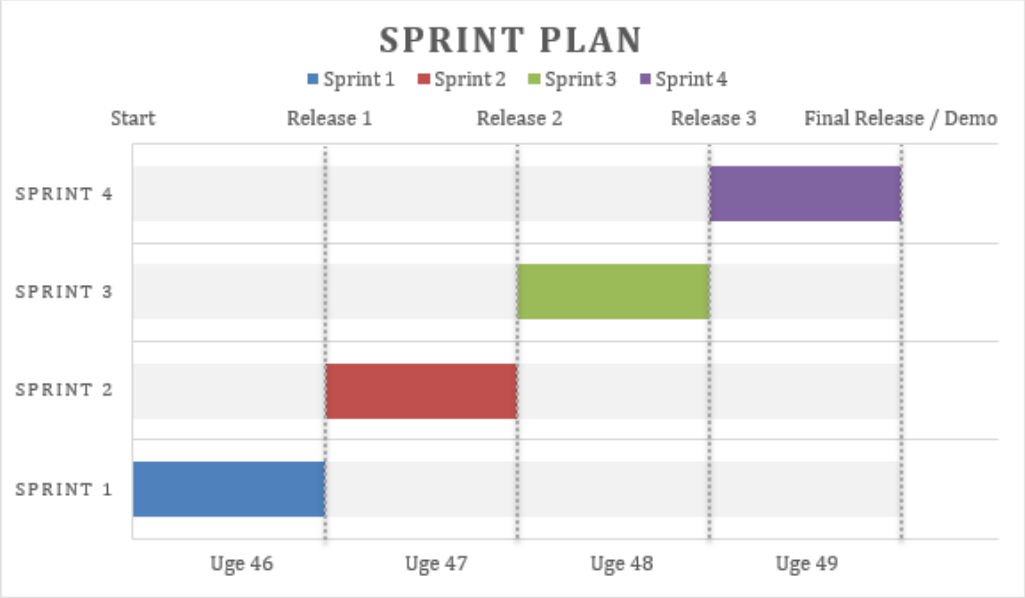
Onsdag: onsdag foregår på samme måde som tirsdag. Daily Scrum kl. 09.45 og derefter arbejdes der videre.

Torsdag: Også torsdag foregår på samme måde som de to foregående dage.

Fredag: Dagen starter kl.09.00 hvor vi mødes til morgenmad og snakker om løst og fast. Kl. 09.45 er der Daily Scrum. Herefter arbejdes der med de opgaver man har indtil klokken er 14.30. Kl. 14.30 er der release af løsningen. Der holdes efterfølgende Sprint Review, hvor vi gennemgår sprintet.

2.4 Projektplan

Der er afsat fire uger til udviklingen af DashBoardet. De fire uger er op delt i fire sprints, af en uges varighed. I slutningen af hver sprint er en release. I slutningen af de fire uger er der en Final Release af DashBoardet. Der vil umiddelbart efter Final Release være en live demonstration af Dashboardet og dets funktioner. På illustrationen ses en sprint plan der viser hvornår de forskellige sprint ligger , og hvornår der er release og demonstration.



2.5 DELKONKLUSION FOR PROBLEMLØSNING

Der er i dette afsnit blevet forklaret om de metoder og værktøjer der vil blive benyttet på projektet. Meningen med problemløsningen har været at få afgrænset de ressourcer der er til rådighed, samtidig med at der gives en introduktion til de elementer der vil blive berørt i udarbejdelsen af løsningen. Der er blevet skabt et godt udgangspunkt til selve udvikling af DashBoardet.

3. Sprint 1

3.1 SPRINT 1 PLANNING MEETING

På første møde blev vi af Product Owner Christian Jacobsen, præsenteret for tolv Backlog Items. Disse tolv Backlog Items indeholder features der indtil videre er ønsket. Der vil løbende blive tilføjet flere Backlog Items til Product Backloggen. De første Backlog Items tager udgangspunkt i den interne brug af applikationen. Det gør det fordi at det i første omgang vil være et internt værktøj. Det vil først bliver tilgængeligt for kunderne når systemet er modent nok.

De tolv Backlog Items består af et Id, en Titel, en User Story, Accept Kriterier og en Business Value. Vores Product Backlog ser på nuværende tidspunkt ud på følgende måde.

Backlog Item id	Titel	Business Value
1	Oversigt over de daglige registeret timer pr. case.	40

Beskrivelse	Accept kriterie
As a Creuna Admin **I want to** Se en oversigt over de daglige registeret timer pr. case **So that** Man kan få et bedre overblik over de timer brugt siden dagen før	• Viser registreringer den aktuelle uge med seneste først. • Alle aktuelle cases • Grafisk illustrering

Backlog Item id	Titel	Business Value
2	Menu med Liste med Projekter	100

Beskrivelse	Accept kriterie
As a Creuna Admin **I want to** Se en menu med de projekter jeg er tilknyttet **So that** Man nemt kan tilgå data tilknyttet projekter	• Vise menu med projekter • Ved at klikke på et menu punkt skal man få vist projektets data

Backlog Item id	Titel	Business Value
3	Oversigt med de cases der skal udarbejdes, kritiske først.	70

Beskrivelse	Accept kriterie
As a Creuna Admin **I want to** Se en oversigt med de cases der skal udarbejdes, gerne med de kritiske (prioriteret) først. **So that** Man kan få et bedre overblik over projektets udvikling her og nu	• Hver status skal afgøre om den optræder på den ene(kundens) eller den anden(internt) --- Se hvem der har bolden liste • Tabelvisning af en art med case nummer og titel

Backlog Item id	Titel	Business Value
4	Liste med status ændringer på en case, samt hvem har bolden	100

Beskrivelse	Accept kriterie
As a Creuna Admin **I want to** Se en status over de seneste ændringer der er fortaget pr. case. **So that** Man kan få et bedre overblik over projektets udvikling, og se hvem der har bolden og skal Reagere på ændringen.	• Vise de 25 sidste status ændringer • Det skal være muligt at se en liste med alle status ændringer, samt udvælge de ændringer der afventer Creuna eller kunden. • Ved at klikke på en status ændring i listen, kommer man til den specifikke case.

Backlog Item id	Titel	Business Value
5	Liste med Backlog items	100

Beskrivelse	Accept kriterie
As a Creuna Admin **I want to** Se en liste over de seneste nye backlog items. **So that** Så man nemt kan se hvilke nye sager der er Blevet oprettet.	• Vise de 25 sidste backlog items • Der skal vises hvor mange backlog items der ligger i listen.

Backlog Item id	Titel	Business Value
6	Oversigt med prioriteret cases der skal udarbejdes, kritiske først.	80

Beskrivelse	Accept kriterie
As a Creuna Admin **I want to** Se et diagram over de sager jeg er involveret i og eres fremskidt. **So that** Man hurtigt kan få et overblik over de sager der sidst har skiftet status. Samt se deres fremdrift.	• Vise de 5 sidste cases der er lavet status ændringer på • Følgende data skal vises. CaseId, titel, alle statusser.

Backlog Item id	Titel	Business Value
7	Oversigt over prioriteret tasks på en case.	80

Beskrivelse		Accept kriterie
As a	Creuna Medarbejder	• Hver status skal afgøre om den optræder på den ene(kundens) eller den anden(internt) --- Se hvem der har bolden liste • Kun den interne liste • Highlight dem som er assigned
I want to	Se en oversigt over prioriteret tasks på en case.	
So that	Man som medarbejder ved hvilke opgaver der skal udføres og hvilken prioritet hver enkelt task har.	

Backlog Item id	Titel	Business Value
8	Oversigt over de berørte aktiviteter uden time registrering.	30

Beskrivelse		Accept kriterie
As a	Creuna Medarbejder	• Se aktiviteter der er berørt (Modified By/Created) for den aktuelle uge uden time registrering • Visningen skal være en liste.
I want to	Se en oversigt over de berørte aktiviteter uden time registrering.	
So that	Det er muligt at sikre at alle timer bliver registret, så man får faktureret al arbejdet	

Backlog Item id	Titel	Business Value
9	Oversigt over de ugentlige registeret timer på mine projekter	70

Beskrivelse		Accept kriterie
As a	Creuna Kunde	• Vis registreringer fra uge til uge • Alle aktuelle cases • Grafisk illustrering
I want to	Se en oversigt over de ugentlige registeret timer på de projekter jeg har som kunde.	
So that	Man kan få et bedre overblik over de afviklet time registreringer	

Backlog Item id	Titel	Business Value
10	Oversigt over ændringer der er fortaget på en case.	100

Beskrivelse		Accept kriterie
As a	Creuna Medarbejder	• Viser ændringer den aktuelle uge • Alle ændringer, også negative ændringer. • Grafisk illustration, hvor plot er den nuværende, og highlight viser den ændring • Skal se begge lister
I want to	Se ændringer der er fortaget på en case	
So that	Man kan få et bedre overblik over projektets udvikling, ydermere gør det enklere at følge med, samt reagere hvis man skal reagere på noget	

Backlog Item id	Titel	Business Value
11	Oversigt over hvem har bolden	80

Beskrivelse		Accept kriterie
As a	Creuna Medarbejder	• Hver status skal afgøre om den optræder på den ene(kundens) eller den anden(internt) --- Se hvem der har bolden liste • Tabelvisning af en art med case nummer og titel
I want to	Se hvem har bolden.	
So that	Man kan få et bedre overblik prioriteret cases.	

Backlog Item id	Titel	Business Value
12	Oversigt over registreret timer for den indeværende måned	50

Beskrivelse		Accept kriterie
As a	Creuna Medarbejder	• Se total timer • Link til de respektive time
I want to	Se registreret timer for den indeværende måned	
So that	Man som kunde kan se hvor mange timer der er brugt og hvor mange man har igen.	

3.1.1 Planning Poker

Efter at havde blevet præsenteret for de tolv Backlog Items, var det tid til at estimere de forskellige Items og udvælge dem der skulle med i første Sprint. Det blev gjort ved hjælp af Planning Poker. Planning Poker er en disciplin hvor man ved hjælp af et spil kort, spiller en form for poker. Hver medlem af udviklingsteamet har et spil kort. Hver kort repræsentere en værdi der enten kan svare til Storypoints eller timer. Der findes mange forskellige skalaer som man kan benytte på sine kort. Men den mest benyttede er efter Fibonacci skalaen.

EKSEMPEL PÅ KORT DER BENYTTER FIBONNACI SKALAEN

I første sprint blev der estimere ud fra timer og fremgangs måden er ret simpel. Teamet bliver præsenteret for et Backlog Item og hver medlem skal så vurdere hvor mange timer den enkelte Feature vil tage at udvikle. Man udvælger så et kort og lægger det på bordet, med bagsiden op. Når alle har lagt et kort på bordet, vender man kortene samtidig. Der skulle gerne være nogenlunde enighed om antallet af timer der skal bruges, men der kan i tilfælde være Features som bliver estimeret meget forskellige. Her er det så op til de individer, der har estimater der ligger langt fra hinanden, at argumentere for hvorfor de mener at det valgte antal timer er korrekt. Denne diskussion skulle gerne være med til at komme nærmere på et præcist antal timer. Er det ikke muligt vil det i de fleste tilfælde være Scrum Masteren der bestemmer det endelige antal timer. Når man er færdig med at estimere er det tid til at udvælge de Features der skal med i sprintet. Man udvælger features på baggrund af hvor kritiske de er og hvor lang tid de tager. Har man flere Features der med samme kritiske værdi, så vælge man den som der tager mindst tid at lave. Der var i første sprint i alt 50 timer til rådighed i teamet. Men kvag af der i sprintet ville blive benytte Pair Programming så var der reelt kun 25 timer til udvikling. Der ud over skulle der indregnes et par timer til at oprette MVC Controller og View, samt AngularJS elementer. Der ville også blive udført en refaktorering af måden Angular er implementeret på. Derfor blev følgende Backlog item valgt

Backlog Item id	Titel	Business Value	Estimat
2	Menu med Liste med Projekter	100	10
Beskrivelse		Accept kriterie	
As a Creuna Admin **I want to** Se en menu med de projekter jeg er tilknyttet **So that** Man nemt kan tilgå data tilknyttet projekter		• Vise menu med projekter • Ved at klikke på et menu punkt skal man få vist projektets data	

3.2 OPSTART AF DASHBOARDET

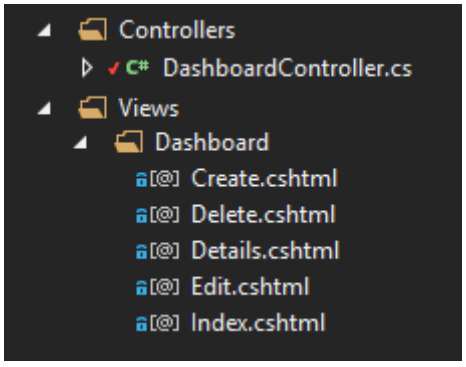
Der var fra start ikke lavet ikke lavet noget design til DashBoardet, dog skulle det følge de samme konventioner som på Creuna ProjektPortal. Derfor ville Dashboardet, blive designet samtidig med at de første features blev implementeret. Designet ville med høj sandsynlighed også blive ændret i takt med udviklingen.

For at gøre arbejdet mere overskueligt blev opstarten af DashBoardet delt op i mindre tasks. Følgende tasks blev defineret.

- Oprette DashBoard Controller
- Oprette DashBoard View
- Oprette AngularJS Controller.

3.2.1 Oprette DashBoard Controller og View

Normalt ville man på et projekt som dette oprette en model først. Men da DashBoardet er en samling af flere forskellige data klasser, så oprettes der ikke en dedikeret model til DashBoardet. Den model der bruges er den der allerede er oprette i forbindelse med ProjektPortalen. Der oprettes i stedet en Controller med et View, som bruger Entity Frameworket. Det er en forholdsvis simpel process at oprette en Controller med et tilhørende View i Visual Studio 2013. Der er intet der ligger til hindring for at man kan oprette en tom controller uden nogen relationer. Skal man benytte Entity Frameworket til fulde så er det klart mest hensigtsmæssigt at oprette en samlet Controller/View pakke. Det gøres ved at man vælger en Model Klasse samt en data kontekst klasse som controlleren og viewet har relation til. Dernæst navngiver man controlleren. View folderen vil så arve navnet fra controlleren og automatisk blive der tilføjet en controller i controller mappen, og der bliver tilføjet en View mappe med 5 filer.



Billedet til højre viser mapper og filer der bliver tilføjet når man oprettet en controller med et tilhørende view. I Dashboard mappen er der blevet tilføjet 5 filer, der hver repræsenterer et View. Til hver View hører der minimum et ActionResult i Controllere. Det er en Get metode, der returnere Viewet. Til flere af de Views hører også Post metoder. De forskellige metoder og deres formål vil bliver forklaret nærmere nedenfor. Der vil på i første omgang på ikke blive gjort brug af Create, Delete, Details eller Edit på Dashboardet. Der vil derfor ikke blive gået i detaljen med disse sider, ud over en kort beskrivelse af de forskellige Views og de metoder de gør brug af i kontrolleren.

- **Create** – Create er som standard en side der er dedikeret til oprettelse af et item af den tilføjede models type. Der er i controlleren to forskellige Create metoder. En Get metode og en Post. Get metoden bruges til at returnere Create viewet til klienten. Som det ses nedefor er det et ActionResult der ene og alene returnere Viewet.

```
public ActionResult Create(){
      return View();
}
```

Post Metoden ses nedenfor. Hvis man kigger over selve metodekaldet, kan man se at der er tilføjet to attributter der ikke er på Get metoden. Det er [HttpPost] som sættes for at fortælle at denne metode kun skal kaldes i tilfældet af at der komme et HttpRequest fra klienten på lige præcis denne Action. Derudover er der også en [ValidateAntiForgeryToken] attribut. Den bruges til at sikre at der ikke bliver Postet indhold fra en ukendt kilde til ActionResultet. Måden den sikre det på, er ved at sende en værdi med HttpPostet fra klienten, til det matchende ActionResult i controlleren. Hvis denne værdi ikke stemmer overens med den der findes, vil forespørgslen blive afvist. I metodehoved er der en [Bind] attribut der bruges til at det der bliver postet nu rent faktisk også høre til datamodellen. Den eneste parameter metoden godtager er et objekt af typen TeamProject. Selve metode består af en if statement, der tjekker om teamProject objektet svare til det objekt man har defineret i sin model klasse. Er den det, tilføjes objektet til databasen i form af det data objektet består af. Det gøres ved at kalde TeamProjects.Add. Add metoden tilføjer en teamProject entitet til data konteksten. Data kontekst vil blive forklaret nærmere i et senere afsnit. Dernæst gemmer man evt. ændringer i datakonteksten ved at kalde SaveChanges(). Lykkedes dette bliver man dirigeret til DashBoardets index side. Hvis ikke sendes man tilbage til Create Viewet sammen med objektet og med en fejlmeddelelse.

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create([Bind(Include = "TeamProjectId,Name")]
TeamProject teamProject)
{
if (ModelState.IsValid)
            {
                  db.TeamProjects.Add(teamProject)
                  db.SaveChanges();
                  return RedirectToAction("Index");
            }
            return View(teamProject);
}
```

- **Delete** – Delete består også af en Get og Set metode**.** Begge metoder tager imod en Id parameter, og den er obligatorisk**.** Hvis ikke der sendes et id med, returneres der er fejl.

```
public ActionResult Delete(int? id)
{
      if (id == null)
      {return new HttpStatusCodeResult(HttpStatusCode.BadRequest);}
```

```
        TeamProject teamProject = db.TeamProjects.Find(id);
        if (teamProject == null){return HttpNotFound();}
        return View(teamProject);


        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public ActionResult DeleteConfirmed(int id)
        {
                    TeamProject teamProject = db.TeamProjects.Find(id);
                    db.TeamProjects.Remove(teamProject);db.SaveChanges();
                    return RedirectToAction("Index");
        }
```

- **Details.cshtml** – På details siden vises som standard, det objekts data man ønsker at få vist. Details består kun af en Get metode, da siden kun er til for at vise data.

- **Edit.cshtml** - Lige som med Delete.cshtml findes der er Get og Set metode. Dette view bruges til at editere objektets data. På samme måde som Delete, returnere Get metoden blot et View. Post Metoden har også mange ligheder med Post metoden i Delete. Forskellen ligger i at de nye ændringer skal gemmes. Det sker på følgende måde.
  Som man kan se på metoden nedefor er både attributter og metodehoved de samme. Der hvor ændringerne bliver udført er i Db.Entry(teamProject).State. Her kaldes Entry metoden med en teamProject entitet som parameter. Entry State sættes til at være Modified og med den tildeling, ændres entiteten og ændringerne gemmes efterfølgende.

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit([Bind(Include = "TeamProjectId,Name")] TeamProject teamPr
oject)
{
        if (ModelState.IsValid){
        db.Entry(teamProject).State = EntityState.Modified;
        db.SaveChanges();
return RedirectToAction("Index");
      }
      return View(teamProject);
}
```

- **Index**– er DashBoardets startside. Det er den første side man bliver præsenteret for når man lander på DashBoardet. Det er et View med en liste af objekter af typen TeamProjects. I controlleren ser metoden ud på følgende måde. En Get metode der returnere et View med en parameter.

```
public ActionResult Index()
{
      return View(db.TeamProjects.ToList());
```

```
}
```

I viewet ser det ud på følgende måde.

```
@model IEnumerable<PFMWeb.Models.PFM.TeamProject>
@{Layout = null;}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <table class="table">
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Name)
            </th>
            <th></th>
        </tr>
    @foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Name)
            </td>
            <td>
                @Html.ActionLink("Edit", "Edit", new { id=item.TeamProjectId }) |
                @Html.ActionLink("Details", "Details", new { id=item.TeamProjectId
 }) |
                @Html.ActionLink("Delete", "Delete", new { id=item.TeamProjectId }
)
            </td>
        </tr>
    }
    </table>
</body>
</html>
```

Index og Create kræver fra start af ikke en parameter. Det gør derimod de andre. I ASP.net MVC kan man definere hvilke parametre der er kræves i sine kald. Det gør man ved at brug af en routing konfiguration. I denne konfiguration kan man også definere hvordan ens url skal og må se ud.

Nedenfor ses hvordan løsningens routing konfiguration ser ud. I RouteConfig klassen, er der er RegisterRoutes metode der tager imod en RouteCollection, som er en samling af router. I metoden kaldes en routes.MapRoute metoden der sætter en rute. Det er her vi kan definere hvordan vores rute skal se ud. Som standard er der en Default Route der indeholder et controller navn, et action navn og et id. Ud over de tre parametre tilføjer man en default route, som bestemmer hvilken controller og hvilken action der kaldes når den rene url tilgås. Id

parameteren har værdien UrlParameter.Optional. Det vil sige at denne parmeter er valgfri. Ikke alle sider benytter denne parameter. Index og Create siderne indeholder som standard ikke imod nogen id parameter. Det gør derimod de resterende sider i DashBoard mappen.

```
public class RouteConfig{
            public static void RegisterRoutes(RouteCollection routes) {
            routes.MapRoute("Default", "{controller}/{action}/{id}",
            new {controller = "Dashboard", action = "DashboardAdmin",
            id = UrlParameter.Optional});
            }
}
```

3.2.1.1 Model Klasse

I dette tilfælde er der som sagt ikke en dedikeret DashBoard Model klasse, så i stedet tilknyttes TeamProject klassen. TeamProject klassen benyttes i første feature. Derfor er det nærliggende at det er den der bliver tilknyttet Controlleren. Nedenfor ses TeamProject Klassen. Den består af et Id, et Navn, og en række typestærke virtuelle lister.

```
public class TeamProject
    {
        public int TeamProjectId { get; set; }
        [Required]
        [Display(Name = "Team name")]
        public string Name { get; set; }
        public virtual List<TeamProjectTFS> TeamProjects { get; set; }
        public virtual List<TeamProjectTG> TeamProjectTGs { get; set; }
        public virtual List<BacklogItem> BacklogItems { get; set; }
        public virtual List<Contract> Contracts { get; set; }
        public virtual List<ContractHourItem> ContractHourItems { get; set; }
        public virtual List<Project> Projects { get; set; }
    }
```

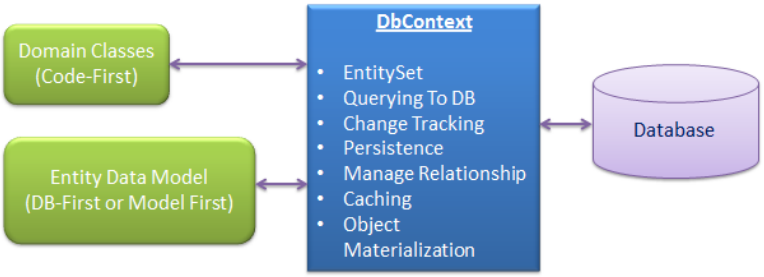
3.2.1.2 Data Kontekst

Ud over en Model Klasse så skal der også tilknyttes en Data Kontekst klasse. I Entity Framework benytter man en kontekst klasse der er bindeled mellem databasen og domæne klassen. Klassen instantiere en klasse kaldet DbContext.

```
public class PFMcontext : DbContext{

}
```

DbContext klassen tager sig bl.a. af at vedligeholde relationerne mellem tabellerne i databasen og de tilsvarende data klasser. Derudover tager den sig også af at laver forespørgsler til databasen i form at CRUD (Create, Read, Update, Delete) kommandoer.



MODEL OVER DBCONTEXT KLASSEN FORINDELSER

For at DbContext klassen ved hvilke tabeller der evt. skal oprettes og/eller skabes relationer til, instantiere man klasserne ved hjælp af en DbSet metode. De klasser man vil have repræsentere i sin database skal være til stede i DbContext Klassen.

```
public class PFMcontext : DbContext{
        public DbSet<CustomerCase> CustomerCases { get; set; }
}
```

3.2.2 Oprette AngularJS Controller

For at kunne bruge AngularJS i sine Views skal man først oprette et modul og en Controller. Der er flere måder man kan bruge moduler og controllere på. Man kan have et modul for hver af sine controllere, eller man kan have et modul, for nogle eller alle sine controllere. Dashboardet gør brug af det sidste. Her er et modul, som fungere som en container, og inde i containeren ligger så alle de controllere der bliver brugt. Nedefor ses en implementering af et modul. Her tildeles et Angular modul, til en variabel.

```
var portal = angular.module('portal', []);
```

Modulet ligger i en separat fil kaldet app.js. Dette gøres for at skille de to lag ad. Selve controlleren ligger i en mappe kaldet controllers. I denne mappe ligger alle de controllere der bruges på projektet. Den Controller der bliver brugt på DashBoardet ligger i en fil kaldet dashboard.js. Nedefor ses implementationen. Modulet (portal) tildeles en controller. Controlleren navngives "dashboardController" og tildeles en række services som controlleren skal eller kan gøre brug af.

```
portal.controller('dashboardController', ['$scope', '$http', '$location', '$q',
'localStorageService', function($scope, $http, $location, $q,
 localStorageService, mainDashBoardData) {}
```

De services der gøres brug er følgende.

- **$scope –** et objekt, det indeholder de data der bliver sendt fra vores request.
- **$http –** en service som opretter en forbindelse til httpserveren.
- **$location** - en service som gør det muligt at tilgå og ændre url'en i browseren.
- **$q –** en service der bruges til at køre funktioner i controlleren asynkront.
- **localStorageService** – en service der bruges til at hente og sende data til browserens localStorage.

Ud over servicerne, oprettes der en funktion. Det er i denne funktion af alle andre funktioner ligger. Der er også her vi initialisere variabler og tildeler data til objekter.

3.3 KODNING AF FEATURE "MENU MED LISTE AF PROJEKTER"

Den første Feature der skal implementeres er en menu med en liste af projekter. Listen skal indeholde de projekter som man har adgang til. Meningen med menuen er, at man ved at klikke på et menupunkt, får vist det data der høre til det projekt man vælger. Menuen skal være en slider menu, som man kan gemme og vise ved at trykke på en +/- knap på brugergrænsefladen. Menuen skal glide ud fra venstre mod højre. For igen at gøre arbejdet lettere at gå til blev featuren delt op i tasks.

- Hente data fra databasen og sende det til Angular controlleren som et JSON objekt.
- Hente data i MVC controlleren og sende det til Viewet.
- Visualisere data i Viewet

3.3.1 Hente data fra databasen og sende det til Angular controlleren som et JSON objekt.

Det første skridt på vejen til at lave menuen, er at hente det data menuen skal fyldes med. Til at lave database forespørgsler, bruges LINQ to Entities. Derudover benyttes der også lambda expressions i de forskellige forespørgsels metodekald. I implementationen nedenfor, ses det at der først skabes en ny database kontekst kaldet db. Det er den kontekst der skal bruges af LINQ til at lave forespørgsler med. Nedeunder konteksten, hentes der en liste af TeamProjects. Det gøres ved at man først vælger hvilken data klasse forespørgslen skal foregå på. Dernæst kaldes en Select metode, i den metode benyttes Lambda expressions til at bestemme hvilke typer data der skal hentes fra TeamProjects. I dette tilfælde hentes hele objekt (x => x). Ville man kun hente navnet på TeamProjects ud kunne det ske med følgende kommando (x => x.Name). Til sidst kaldes ToList metoden, der tilføjer de valgte objekter til en liste. Liste tildeles til variablen customers.

```
private PFMcontext db = new PFMcontext();
var customers = db.TeamProjects.Select(x => x).ToList();
```

Næste skridt er at oprette et Json objekt som kan returneres. Først deklareres et data objekt. Inde i det data objektet oprettes et nyt customers member. Derefter tildeles Customers en række key/value par. Igen bruges der LINQ til at lave forespørgslen og lambda expressions til at skabe parrene.

```
var data = new
{
        customers = customers.Select( c => new
                    {
                         name = c.Name,
                         id = c.TeamProjectId,
                    }),
};
```

Til sidst returneres Json objektet, som indeholder data objektet, samt en request behavior. Denne behavior tillader at der sendes data tilbage fra klienten.

```
return Json(data, JsonRequestBehavior.AllowGet);
```

3.3.2 Hente data i Angular controlleren og sende det til Viewet.

Det data der skal sendes til Viewet kommer fra Angular controlleren. Det er altså Angular controlleren der kalder MVC controlleren og ikke Viewet som det normalt ville være hvis man benytter Razor. I praksis fungere det ved at lave et httpRequest og tildele det til en variabel. Derved kan man benytte requestet i flere forskellige funktioner. Der er kun dette ene post request i controlleren. Det vil sige at post Requestet fungere som både et post og get request. Post requestet fungere som i et normalt post request. Men i stedet for at benytte en separat get request. Så bruges Post requestets succes respons til at transportere data med tilbage fra MVC controlleren. Nedenfor ses http requestet. Først tildeles $scope.output et id, som svarer til id'et på det projekt man vil se. Dernæst begynder selve requested. http requestet deklareres med en request metode og en url, samt data og en timeout parameter. Som det kan ses kalder det DashboardResult metoden i Dashboard controlleren 'Dashboard/DashboardResult'. Ved succes sendes der data tilbage og data'en tildeles så til $scope.customers. Det er dette scope, Viewet bruger til at hente og vise data.

```
$scope.output = {id: localStorageService.get('Customer')};
           var httpRequest = $http(
               {
                   method: 'POST',
                   url: '/Dashboard/DashboardResult',
                   data: $scope.output,
                   timeout: canceler.promise
               })
               .success(function(data, status) {
                   $scope.customers = data.customers;
                 })
               .error(function(data, status, headers, config) {
                   $scope.error = {
                       data: data,
                       status: status,
                       headers: headers,
```

```
                    config: config
                };
            });
        };
```

Ud over et succes respons, sendes der i tilfælde af en fejl et error respons. Som udgangspunkt håndteres disse fejl internt. Men de gemmes dog i et scope, med forskellige typer data, som senere kan bruges til fejlretning.

3.3.3 Visualisere data i Viewet

For at bruge Angular i sine views, så skal der i viewet defineres hvilket modul og hvilken controller viewet skal benytte. Det gøres ved at tildele et Angular moduler og nogle direktiver til sine html tags. Modulet starter altid med ng-. Først tildeler man et Angular direktiv kaldet ng-app. Direktivets værdi svarer til det modul (”portal”), som der er defineret i app.js. Alt der ligger inde for det tag som direktivet er tildelt, vil benytte dette modul.

```
var portal = angular.module('portal', ['LocalStorageModule']);
<div class="container-fluid" ng-app="portal">
```

Dernæst implementeres et ng-controller direktiv som svare til den controller der skal benyttes. I dette tilfælde er det dashboardControlleren.

```
portal.controller('dashboardController', []){};
<div id="wrapper" ng-controller="dashboardController">
```

Til at få vist data i Viewet, benyttes der en usorteret html liste <ul>. Inden i denne liste er der et list item <li> der er tildelt et ng-repeat direktiv. Dette direktiv implementer en løkke som itererer over et dataset. I dette tilfælde er det en foreach løkke der itererer over customers objektet, som er en liste af customer objekter.

```
<li ng-repeat=" customer in customers" ng-click="Refresh(customer.id)">
```

På samme list item, tildeles et andet direktiv, ng-click. Dette direktiv svare til JavaScripts on-click metode, eller APS.Nets onClick. ng-click kalder en Refresh() funktion i Angular controlleren. Den medtager en parameter som er det id som projektet har. Refresh funktionen gør to ting. Først tildeler den et objekt til localStorageServicen, derefter Kalder den loadFilters(), som er den funktion http requested ligger i.

```
$scope.Refresh = function(id) {
            localStorageService.add('Customer', id);
            $scope.loadFilters();
        };
```

Inde i hver list item udskrives en streng, som er navnet på projektet. Nedenfor ses Angulars måde at udskrive strenge på i viewet.

```
{{customer.name}}
```

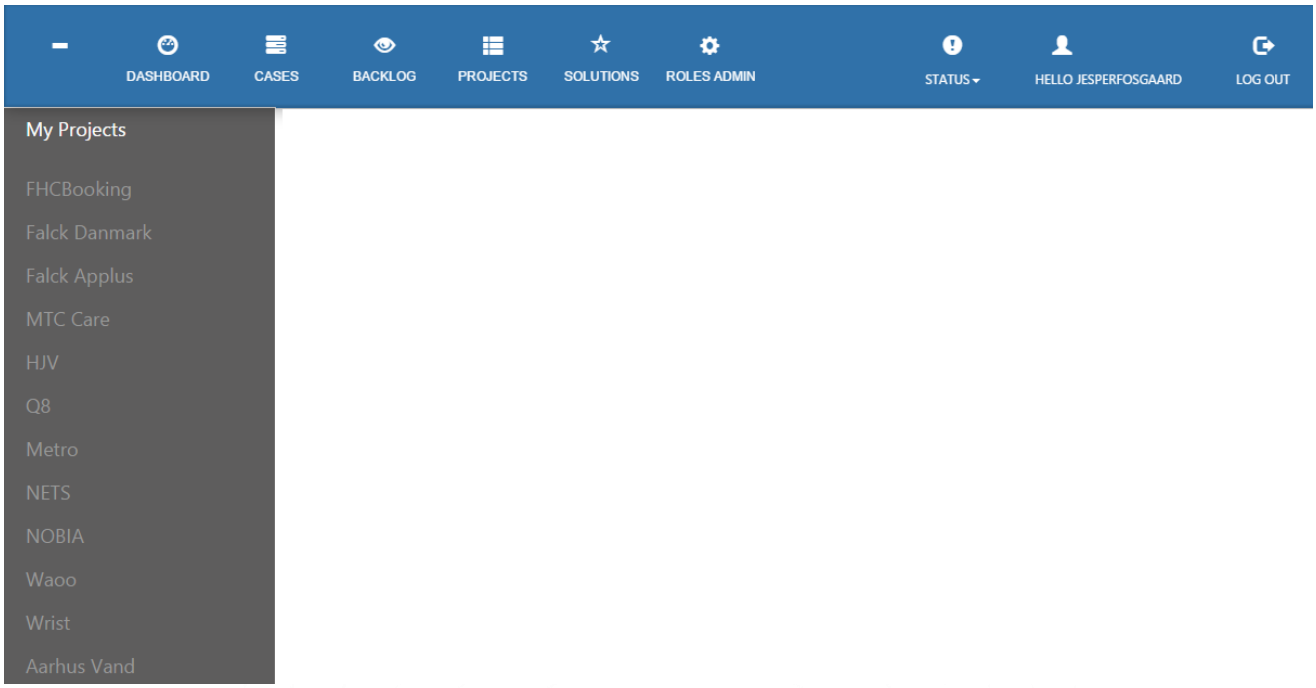
Nedefor ses hele implementationen i Viewet.

```
<div class="container-fluid" ng-app="portal">
        <div id="wrapper" ng-controller="dashboardController">
            <div id="sidebar-wrapper">
                <ul class="sidebar-nav">
                    <li class="sidebar-brand" style="color:white">
                        My Projects
                    </li>
                    <li ng-repeat=" customer in customers" ng-
        click="Refresh(customer.id)">
                        {{customer.name}}
                    </li>
                </ul>
            </div>
        </div>
</div>
```

Til sidst skal det gøres muligt at gemme menuen væk. Det gøres ved at bruge Angular sammen med JQuery. Denne funktion ligger i en separat fil kaldet adhoc.js. Der bliver bundet en click function på et div tag med id'et "menu-toggle". I denne function bindes en toggleClass funktion til et tag der har id'et "wrapper". ToggleClass gør det muligt at tilføje eller fjerne en stylingklasse på et tag. I dette tilfælde tilføjer og fjerner den en styling klasset kaldet "toggled" til taget.

```
$("#menu-toggle").click(function (e) {
    e.preventDefault();
    $("#wrapper").toggleClass("toggled");
});
```

Menuen er nu færdig og resultatet er blevet følgende. I venstre side ses menuen. Det er som tidligere beskrevet en liste med projektnavne. Trykker man på et at disse navne, vises det data der er tilknytte projektet. Der er på nuværende tidspunkt ikke vist noget data, da implementeringen af disse features først foregår i de kommende sprints. Øverst i venstre hjørne ses et – tegn. Ved at trykke på dette tegn, glide menuen til venstre, og man kan derved få et større overblik over de data der bliver vist.



3.4 BRUGEN AF PAIR PROGRAMMING

I først sprint blev der gjort brug af XP praktikken pair programming. Måden hvorpå det foregik, var at den anden Praktikant Chakib og jeg, sad sammen og skiftedes til at kode. Vi valgte at gøre det i første sprint, da vi begge mente, at ved at sidde sammen i staten, så ville det give os det samme udgangspunkt fremadrette. Rent praktisk sad vi i starten og diskuterede forskellige emner. Det var blandt andet brugen af teknologier og udseende. Efter at havde komme frem til en løsning vi begge kunne stå inde for. Så gik selve programmeringen i gang. Vi brugte et værktøj på kanbanflow.org til at tage tid på de features vi programmerede. Værktøjet var en Pomodoro timer, som er et ur der tæller ned fra en given tidsværdi. I vores tilfælde var det 20 min. Når de 20 minutter var overstået, skiftede vi plads. Værktøjet er ret smart i og med at der er mange muligheder. Blandt andet kan man indlægge pauser imellem nedtællingerne, samt en masse andre funktioner.

3.5 DELKONKLUSION FOR SPRINT 1

Der blev i første sprint brugt en del tid på opsætning af Angular. Inden DashBoardet, var der på ProjektPortalen tildelt et modul til hver controller. Ud over det så lå alle moduler og controllere i samme fil. Det gjorde det enormt uoverskueligt, og derfor var det nødvendigt med en refaktorering. Der blev i sprintet ikke gjort så meget brug at Scrum Boardet. Grunden til det var at der kun var et Backlog item i vores sprint Backlog, og samtidig med det blev der gjort brug af pair programming. Det vil dog uden tvivl blive brugt mere i de næste sprint. I forhold til par programmering, så var det, det helt rigtigt at gøre i første sprint. Vi har ved at sidde sammen fået en fællesforståelse for, i hvilken retning DashBoardet skal. Det er muligt at vi kunne havde nået en del mere hvis vi hver især havde arbejdet på forskellige features. Men vi har opnået en kontakt, som vi ikke havde opnået, ved at arbejde selvstændigt. Alt i alt er første

sprint forløbet som vi regnede med. En lang opstart med refaktorering, og klargøring. Og så en relativ ukompliceret implementeringen af første feature.

4. Sprint 2

4.1 SPRINT 2 PLANNING MEETING

Det var nu tid til at planlægge andet sprint. Teamet samledes til det obligatoriske Sprint planning meeting, og begyndte at udvælge backlog items fra product backloggen til sprint backloggen. Igen i dette sprint var der 25 timer til rådighed for mig.

Følgende backlog item blev udvalgt.

Backlog Item id	Titel	Business Value	Estimat
4	Liste med status ændringer på en case, samt hvem har bolden	100	20

Beskrivelse	Accept kriterie
As a Creuna Admin **I want to** Se en status over de seneste ændringer der er fortaget pr. case. **So that** Man kan få et bedre overblik over projektets Udvikling, og se hvem der har bolden og skal Reagere på ændringen.	• Vise de 25 sidste status ændringer • Det skal være muligt at se en liste med alle status ændringer, samt udvælge de ændringer der afventer Creuna eller kunden. • Ved at klikke på en status ændring i listen, kommer man til den specifikke case.

Jeg blev valgte at udføre Backlog item 2 ” Liste med status ændringer på en case, samt hvem har bolden”

4.2 OPSTART AF FEATURE ”LISTE MED STATUS ÆNDRINGER PÅ EN CASE, SAMT HVEM HAR BOLDEN”

Der var fra Creuna et stor ønske om, at man både internt og eksternt, kunne se hvem der havde bolden, når det havde været en ændring på en case. Når der tales om hvem der har bolden, så menes der at man skal kunne se, hvem der har ansvaret for at reagere på en ændring. Der er ofte tidligere været forsinkelser i forskellige processer, fordi at det har været uklart hvem der skulle reagere på en ændring. Det koster tid og penge, og det er ikke rentabelt, hverken for Creuna eller kunden. Det giver også et indtryk af at der ikke er hundred procent styr på processen, og det er ikke sådan Creuna er interesseret i at fremstå. Der blev derfor i samme omgang, stille mig den opgave at jeg skulle definer et flow for ændringer af statusser, samt hvem bolden lå hos. Dette flow ville blive visualiseret, i form at et flowchart, der nemt skulle give et overblik over tingene. Selve listen med status ændringer skulle vises som en tabel med rækker og kolonner. Derud over skulle det være muligt at filtrere på rækkerne, så man enten kunne se alle rækker, dem der afventer kunden eller dem som afventer Creuna. Som det sidste skulle der også være en tæller der viste hvor mange status ændringer der lå i den valgte liste. Denne tæller skulle selvfølgelig opdateres sammen med filtreringen.

Igen blev featuren delt op i nogle tasks.

- Lav flow over status ændringer
- Status data klasse
- Status Controller
- Status View

4.3 UDARBEJDELSE AF FLOW FOR STATUS ÆNDRINGER

For at kunne udarbejde et flow for statusændringer på en case, var det vigtig at der blev definere en livscyklus for en Case. Der skulle altså være en klar definition på hvornår en Case startede, hvornår den ændrede status og hvem der ændrede status, samt hvornår en case var lukket. Der var også to forskellige faser af processen hvor casen kunne befinde sig. Den ene fase var analysefase, hvor en case blev oprette og defineret. Den anden fase var udførelsesfasen, hvor at en case blev udført og testet.

Det første der skulle defineres var opstarten af en case. Her er det to scenarier der kan udspille sig. Det første scenarier er at en kunde oprette en case som, kan udføres uden nogen form for analyse. Dette gælder typisk for et service projekt, hvor mindre ændringer, eller fejlretninger kan blive implementer uden den store planlægning. Her går en case fra at være oprette, til direkte at ændre status til ”gå i gang” Det andet scenarie er at en kunde oprette en case, hvor at der både skal udarbejdes en analyse og et løsningsforslag. Det er to meget forskellige scenarier, især det sidste er meget tidskrævende. Det er det fordi at løsningsforslaget er en del af en rekursiv process. Et løsningforslag sendes ofte frem og tilbage mellem Creuna og kunden, indtil den rette løsning er fundet. For at undgå forsinkelser, er det derfor vigtig at vide hvem der sidder med bolden, så de ansvarlige interessenter kan reagere. Det giver også en gennemsigtighed i processen som man i høj grad ønsker.

Første del af flowet var på plads og forløbere således. En kunde oprette en case og tildeler den en af følgende statusser.

- **”Ny sag”** som er en case hvor der skal udarbejdes analyse og løsningsforslag.
- **”Ny sag gå gerne i gang”** som er en case kan udføres, uden at der skal udarbejdes en analyse eller et løsningsforslag.

I tilfælde af at en case har statussen ”ny sag”, så vil næste skridt være. Udarbejdelse af analyse og løsningsforslag. I denne del af processen er der to dele. De har hver især fået en status beskrivelse

- ”**Analyse og løsningforslag udarbejdes”** som er her Casen analysere og forslag til løsningen findes.
- ”**Løsningsforlags afventer information”** denne status sættes når der fra Creuna er sendt et ønske om mere information. Det kan være der mangler information, så analysen ikke kan udføres tilstrækkeligt. Det kan også være yderligere forklaring i forhold til den specifikke case.

Når denne del menes at være færdig, sendes løsningsforslaget til godkendelse hos kunden. Her får den følgende status.

- "**Løsningsforslag afventer godkendelse**" når en case har denne status, så er det op til kunden at afgøre om forslaget godkendes eller ej.

Godkendes løsnings forslaget, tildeles følgende status.

- "**Løsningsforslag godkendt, klar til udførelse"** Denne status markere at analysefasen er færdig og udførelsesfasen kan påbegynde.

Godkendes løsningsforslaget ikke, så får casen denne status.

- "**Løsningsforslag afvist"** i tilfælde af at en løsningsforslag afvises, skal et nyt løsningforslag udarbejdes. Det betyder at den forrige process gentages indtil at et løsningsforslag godkendes.

Der er også to andre statusser som en case kan få i denne fase. Det er disses følgende status.

- "**I bero"** det kan ske at der opstår et scenarie hvor en case bliver sat i bero. Det kunne være i tilfælde af at der kom en case som havde større værdi, eller som var særlig kritisk.
- " **Sag afvist "** det kan ske at der opstår et scenarie hvor en case bliver sat i bero. Det kunne være i tilfælde af at der kom en case som havde større værdi, eller som var særlig kritisk.

Det anden del af flowet er udførelsen af casen. Når denne fase indledes får casen følgende status.

- "**Under udførelse"** denne status markerer, at en case er ved at blive udført.

Når udførelsen af en case er færdig, tildeles den følgende status.

- "**Rette i udvikling"** denne status fortæller blot at case er færdig udviklet.

Når en case opnår denne status, er den klar til at blive teste. Der er flere tests der skal gennemføres og i forskellige miljøer. Det første er et test miljø, som er det første miljø der teste i fra kundes side. Dernæst følger test i et præproduktionsmiljø. Begge miljøer indeholder de samme typer af statusser, og med de samme konsekvens. Statusserne er følgende.

- "**Klar til testmiljø"** denne status betyde af casen er klar til et test forløb og at test forløbet påbegyndes.

Det er op til kunde at godkende tests. Godkendes testen ikke får den status

- **"Test i testmiljø ikke accepteret"** denne status betyder at case må en tur tilbage i udvikling.

Godkendes testen derimod sker der følgende.

- ”**Test godkendt”** statussen markere at test er godkendt og næste test kan begynde.

Den skifter derefter automatisk status til.

- ”**Godkendt i test miljø”** statussen betyder at testen er godkendt og er klar til næste test.

Casen er nu klar til at blive godkendt i præproduktionsmiljøet. Her er proceduren fuldstændig den samme som i testmiljøet. De følgende statusser kan opnås.

- ”**Klar til test i præproduktionsmiljø”** Statussen markere at casen er klar til test i præproduktionsmiljøet.
- ”**Test i Præproduktionsmiljø ikke accepteret”** Statussen markere at teste ikke godkendes og casen skal tilbage i udvikling.
- **”Præproduktionstest godkendt”** Statussen markere at testen er godkendt, og den får automatisk status ”Godkendt i Præproduktionsmiljø”.
- **”Godkendt i Præproduktionsmiljø”** Statussen markere at testen er godkendt i Præproduktionsmiljø og casen er klar til at blive lagt i produktion.

Godkendes testen i Præproduktionsmiljø, så indledes sidste fase af processen. Først får casen følgende status.

- **”Lagt i produktion”** Statussen markere at casen er lagt i produktionsmiljøet og afventer sidste godkendelse for at blive lukket.

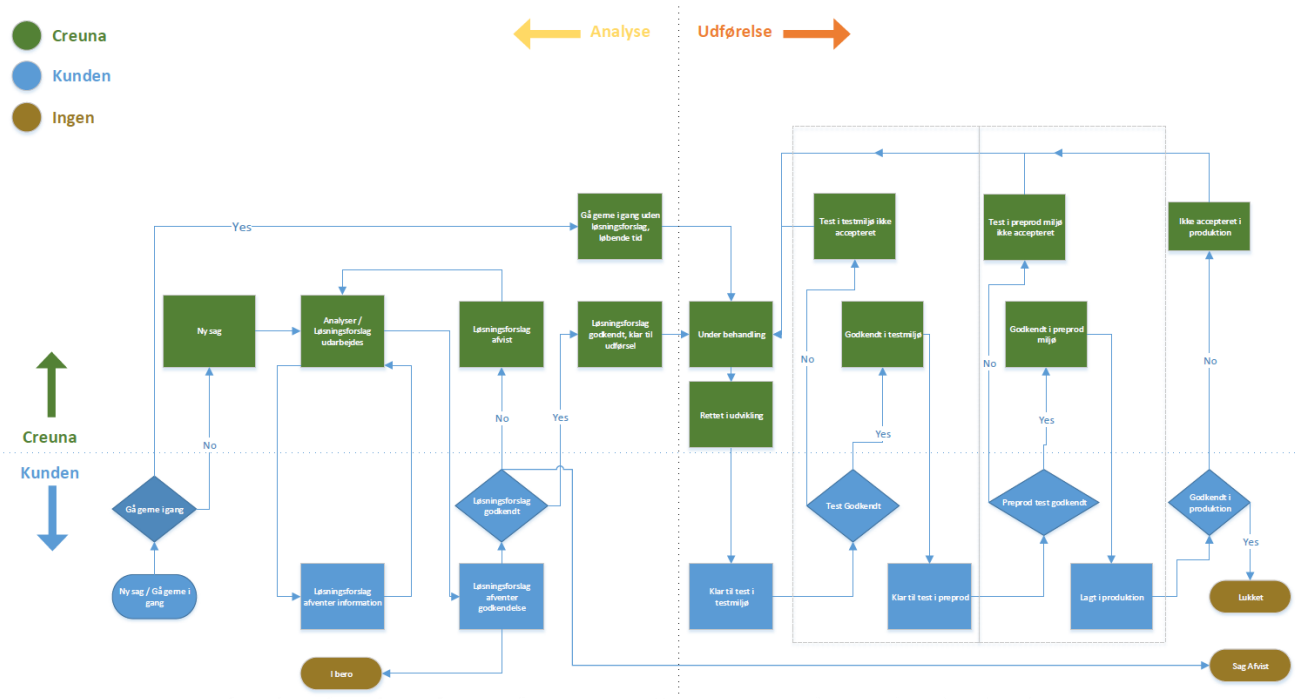
Godkendes casen ikke i produktionsmiljøet opnår den følgende status.

- **”Ikke acceptere i produktion”** statussen betyder at casen igen må tilbage i udvikling.

Godkendes casen i produktionsmiljøet opnår den følgende status.

- **”Godkendt i produktion”** Markere at casen er godkendt i alle faser og sagen kan nu lukkes.
- **”Lukket”** Sagen er lukket

Casen er nu lukket og det markere enden på dens livscyklus. Alle statusser er defineret og flowchartet kan tage form. Det færdige resultat ses nedefor.

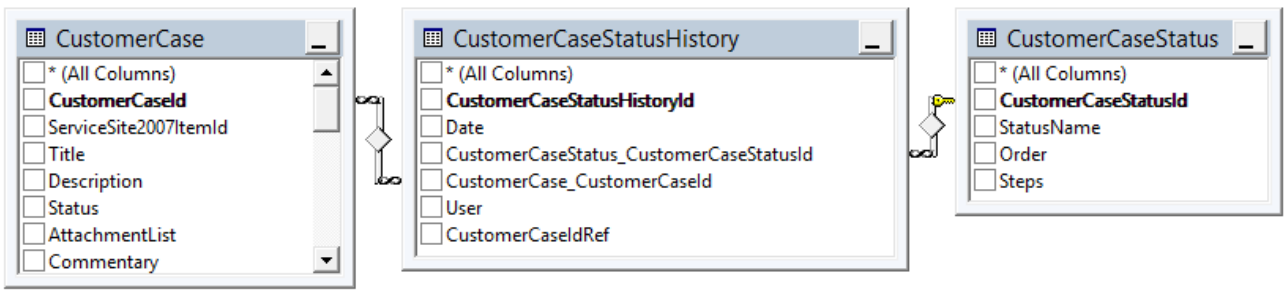


4.4 KODNING AF FEATURE "LISTE MED STATUS ÆNDRINGER PÅ EN CASE, SAMT HVEM HAR BOLDEN"

4.4.1 Status data klasser

Der er tre klasser involveret i statusserne. Det er CustomerCases klasse, som er indeholder alt information om selve casen. Så er det CustomerCaseStatus, der indeholder information om de forskellige statusser en case kan gennemgå. Til sidst er CustomerCaseStatusHistory klassen som er en klasse der indeholder information om de status ændringer der er sket på en case. CustomerCases klassen var på forhånd oprette da Projekt portalen gøre brug samme klasse. Der for var det kun CustomerCaseStatus og CustomerCaseStatusHistory der skulle oprettes. Efter oprettelse, blev de to klasser med hjælp fra Entity Frameworket, migreret til databasen.

Nedenfor ses data modellen.



De tre tabeller er forbundet gennem brugen af primære og fremmede nøgler. Der utrolig nemt at skabe disse relationer i Entity Frameworket. Det eneste det kræver, er at man tilføje den klasse, man ønsker relationen til.

Creuna DashBoard

Nedenfor ses det hvordan CustomerCaseStatus klassen er tilføjet til CustomerCaseStatusHistory klassen. Blot ved at tilføje klassen, skabes relationen mellem de to klasser.

```
public class CustomerCaseStatusHistory
   {
       public virtual CustomerCaseStatus CustomerCaseStatus { get; set; }
   }
```

4.4.2 Status Controller

For at hente statussen ud fra de forskellige cases, benyttes samme data objekt som blev brugt til at hente menu data med. Objektet udvides med et nyt member som kaldes rows. Dette member er egentlig et nyt objekt, der indeholder en liste med CustomerCase objekter. Disse objekter indeholder på forhånd ikke nogen status ændringer. Men der er i CustomerCase klassen tilføjet en virtuel liste af typen CustomerCaseStatusHistory.

```
public virtual List<CustomerCaseStatusHistory> CustomerCaseStatusList { get; set; }
```

Ved at tilføje denne liste til klassen, giver det en den mulighed at der kan tildeles CustomerCaseStatusHistory objekter til listen, som på den måde kan tilgås via CustomerCase objektet. Grunden til at implementationen er lavet på denne måde er blandt andet for at undgå for meget kode, og redundant kode, samtidig undgår man også at lave flere kald til controlleren. Man kunne sagtens lave en separat liste af CustomerCaseStatusHistory objekt, og tilføje den til data objektet. Men da der på en senere feature skal bruges både CustomerCases og deres status historik. Så er dette umiddelbart den mest optimale løsning. Det kan dog på sigt vise sig ikke at være den mest optimale måde at gøre det på, og er det tilfældet, vil en refaktorering af koden finde sted.

Før det hele kan tilføjes til data objektet, så skal der hentes en liste af CustomerCase objekter. Det gøres som i lignende situationer, ved at tildele listen til en variabel. I dette tilfælde hentes kun de cases der er tilknytte det projekt man har valgt i sin menu. Dernæst benyttes så en foreach løkke til at itererer over listen. I løkken oprettes endnu en variabel (child) og igen tildeles en list af objekter. Her er det så CustomerCaseStatusHistory objekter der fyldes i listen. I database forespørgslen er to betingelser. Betingelserne er at enten CustomerCaseId eller CustomerCaseIdRef skal være det samme som id'et på den case der er aktiv i iterationen. Er en af de to betingelse opfyld, så tilføjes objektet til listen, som efterfølgende tilføjes til CustomerCaseStatusListen.

```
var cases = db.CustomerCases.Where(c => c.Team.TeamProjectId == id).ToList();
        foreach (CustomerCase c in cases) {
        var child = db.CustomerCaseStatusHistory.Where(
        cc => cc.CustomerCaseIdRef == c.CustomerCaseId ||
        cc.CustomerCase.CustomerCaseId == c.CustomerCaseId).ToList();
        c.CustomerCaseStatusList.AddRange(child);
        }
```

Efter at CustomerCaseStatusHistory objekterne er blevet tilføjet til CustomerCaseStatusListen, skal Cases objektet tilføjes til data objektet. Nedenfor kan man se hvordan cases tildeles til rows. I koden ses det kun at der oprettes en statusList med objekter fra CustomerCaseStatusListen, og ikke hele CustomerCase objektet. Det skyldes at det i dette tilfælde ikke er relevant for beskrivelsen af implementationen.

```
rows = cases.Select( x => new {
        statusList = x.CustomerCaseStatusList.Where(y => y.Date < dateTo
        && y.Date > dateFrom).Select(y => new {
        caseId = x.CustomerCaseId,
        title = x.Title,
        order = y.CustomerCaseStatus.Order,
        name = y.CustomerCaseStatus.StatusName,
        date = y.Date.ToString("g"),
        step = y.CustomerCaseStatus.Steps,
        user =
        alpha.Users.Where(z => z.Id == y.User).Select(v => v.UserName).FirstOrDefault(),
        }),
}).AsEnumerable(),
```

Nu er rows blevet fyldt med data og det kan nu bruges i Angular controlleren.

4.4.2 Status Angular Controller

Inden i http requesteds succes respons, tilføjes rows data til et scope kaldet $scope.rows. Ud over det kaldes der en funktion kaldet $scope.statusListFunc().

```
$scope.rows = data.rows;
$scope.statusListFunc(0);
```

StatusListFunc er en funktion der tager imod en step parameter. Denne parameter er et tal og det bruges til at bestemme hvilken type status der skal vises. Tallet svarer til det tal der er i step kolonnen, i CustomerCaseStatusHistory tabellen .0 svarere til alle statusser. 1 svarer til alle statusser der afventer Creuna. 2 svarer til alle de statusser der afventer kunden. I funktionen ses det at der hentes data fra $scope.rows ind og tildeles til en variabel. Der oprettes yderligere to variabler, element og list. Dernæst er der en for løkke, der itererer over statusData som indeholder CustomerCases. Løkken fortsætter så længe at antallet af iterationer ikke overstiger det antal elementer der er i statusData. Inde i den løkke ligger så en anden for løkke. I denne løkke itereres der over den statusliste, som CustomerCasen indeholder. I løkken findes der også en if else betingelse. Denne betingelse operere på den før nævnte step parameter. Er den 0, hentes alle statusser ud. Er den en 1 eller 2 hentes henholdsvis de statusser der afventer Creuna eller kunden. Alt afhængigt af hvilken af betingelserne der opfyldes, så tilføjes de valgte elementer til list variablen. Til sidst tilføjes list til $scope.statusListAll. Det er dette scope der bruges i Viewet, til at vise statuslisten.

```
$scope.statusListFunc = function (step) {
           var statusData = $scope.rows;
           var element = [];
           var list = [];
           for (var i = 0; i < statusData.length; i++) {
              for (var k = 0; k < statusData[i].statusList.length; k++) {
                  if (step == 0) {
                        element = statusData[i].statusList[k];
                        list.push(element);
                   }
                   else {
                       var t = statusData[i].statusList[k].step;
                       if (t == step) {
                           element = statusData[i].statusList[k];
                           list.push(element);
                       }
                   }
               }
           }
           $scope.statusListsAll = list;
       };
```

4.4.3 Status Viewet

Viewet der bruges er Index, som det er på hele DashBoardet. Til at visualisere listen gøres der brug af et panel som indeholder, en tæller over antallet af forekomster i listen, en titel, tre filter knapper og en tabel som indeholder statusserne. Nedenfor ses starten af implementeringen hvor tælleren, titlen, og de tre knapper ligger. Tælleren får sat en værdi, ved at kalde statusListsAll.length. StatusListsAll er det scope listen blev tilføjet til i Angular controlleren. Titlen er en simpel streng i et h3 tag. De tre knapper er lidt mere interessante. De har hver især en ng-click, som kalder en statusListFunc funktion, med en parameter. Function ligger i Angular controlleren, og er den samme som der blev beskrevet i afsnitte om status controlleren.

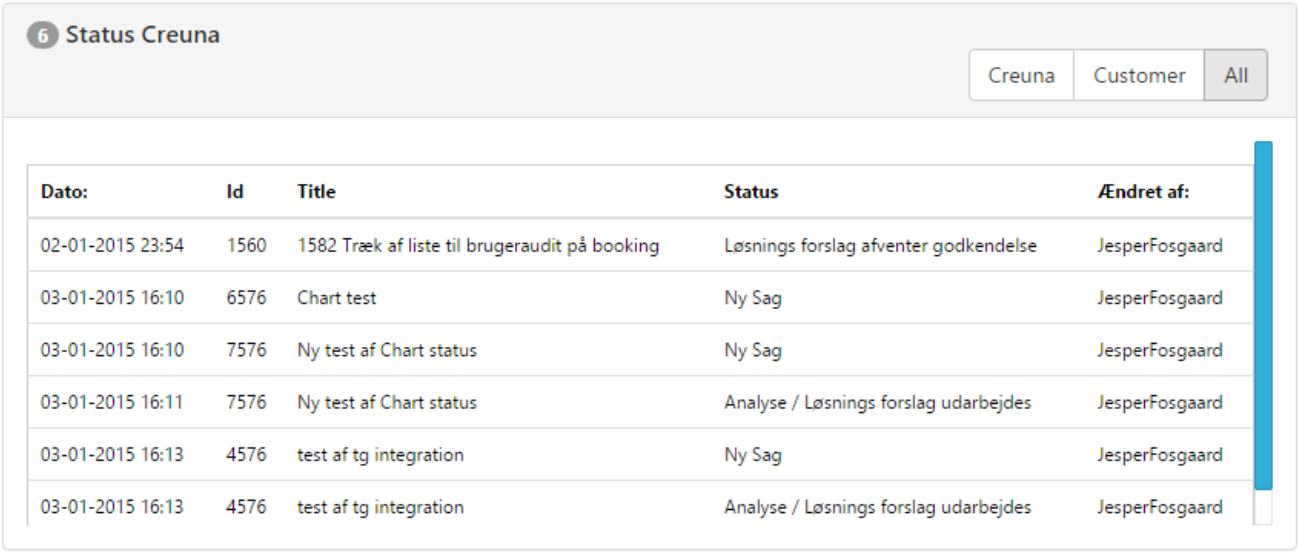
```
<div class="panel panel-default">
        <div class="panel-heading">
        <h3 class="panel-title"><span class="badge">{{statusListsAll.length}}</span>
        Status Creuna</h3>
        <div class="pull-right">
        <div class="btn-group" role="group">
        <button type="button" class="btn btn-default" ngclick="statusListFunc(1)">Creuna
        </button>
        <button type="button" class="btn btn-default"
        ng-click="statusListFunc(2)">Customer
        </button>
        <button type="button" class="btn btn-default" ng-click="statusListFunc(0)">All
        </button>
        </div>
        </div>
        </div>
</div>
```

Nå det kommer til selve listen, så benyttes der som sagt en tabel til at vise listens rækker og kolonner. Nedenfor ses det mest væsentlige af implementationen. Som ved visning af menuen, så benyttes der

også her ng-repeat, som her benytter statusListsAll scopet. For hver forekomst, udskrives en række med Dato, CaseId, titel, navn, og brugeren der sidst har ændret status. Der sættes på rækken også en ng-click som kalder en function, med rækkens caseId som parameter. Funktionen ligger i Angular controlleren, og den kalder en url ('/CustomerCases/Edit/' + id) som viser den case, statussen tilhører.

```
<tbody ng-repeat="statusListAll in statusListsAll">
        <tr ng-click="showCustomerCase(statusListAll.caseId)" style="cursor: pointer">
                <td>{{statusListAll.date}}</td>
                <td>{{statusListAll.caseId}}</td>
                <td>{{statusListAll.title}}</td>
                <td>{{statusListAll.name}}</td>
                <td>{{statusListAll.user}}</td>
        </tr>
</tbody>
```

Nedenfor ses det færdig resultat. Udseendet på panelet er der ikke lagt meget arbejde i. Der vil højst sandsynligt blive ændret på udseendet, når de kommende features bliver implementeret.



4.5 DELKONKLUSION FOR SPRINT 2

Sprint to var efter min opfattelse, et rigtig fedt sprint. For det første var det en super fed opgave at få, at skulle definere et helt nyt flow for statusserne. Derudover var det også fedt at komme i gang med at få kodet nogle features der rent faktisk gav noget værdi til DashBoardet. Det må man i høj grad sige at panelet med status ændringerne gav. De næste sprint bliver mindst lige så interessante. Masser af fede features skal laves, og masser af nye udfordringer venter.

5 Sprint 3

5.1 SPRINT 3 PLANNING MEETING

Det var nu tid til at afholde tredje sprint planning meeting. På dette møde blev der præsenteret to nye Backlog items. Der var fra projektleder gruppen, et ønske om at få tilføjet muligheden for downloade CustomerCase listen og Backlog Listen som xsl filer. Det ønskede de fordi, der var en del af projektlederne der i høj grad benyttede Excel til at lave deres egne data filtreringer. Derfor var det meget aktuel at disse to features blev implementeret. Jeg havde ansvaret for udførelsen af 3 backlog items i dette sprint. De 3 Backlog items ses neden for.

Backlog Item id	Titel	Business Value	Estimat
5	Liste med Backlog items	100	10
Beskrivelse		**Accept kriterie**	
As a Creuna Admin **I want to** Se en liste over de seneste nye backlog items. **So that** Så man nemt kan se hvilke nye sager der er Blevet oprettet.		• Vise de 25 sidste backlog items	

Backlog Item id	Titel	Business Value	Estimat
13	Downloade liste med CustomerCases	90	5
Beskrivelse		**Accept kriterie**	
As a Creuna Admin **I want to** Downloade en liste med CustomerCases som Kan åbnes i et regneark. **So that** Man Kan lave sine egne udtræk og filtreringer.		• Downloade en listen som en xsl fil	

Backlog Item id	Titel	Business Value	Estimat
14	Downloade liste med backlog items	90	5
Beskrivelse		**Accept kriterie**	
As a Creuna Admin **I want to** Downloade en liste med backlog items som Kan åbnes i et regneark. **So that** Man Kan lave sine egne udtræk og filtreringer.		• Downloade en listen som en xsl fil	

5.2. KODNING AF FEATURE

Der skulle som sagt udarbejdes tre backlog items af mig i dette sprint. Den første feature der blev lavet var listen med backlog items. Liste blev lavet på samme måde som listen med status ændringerne. Listen blev lagt ind i et panel, som også indeholdte en tæller, en titel og tabellen med Backlog items. Da de to implementationer er meget identiske, så vil denne feature ikke blive forklaret nærmere. Derimod er udarbejdelsen af Download funktionen mere interessant at se på.

5.2.1 Controlleren

I DashBoard controlleren ligger den metode der gør det muligt at downloade listen. Nedenfor ses metoden. Metoden er et ActionResult, der tager imod en parameter. Parameteren er en streng, og den

indeholder navnet på den type liste man ønsker at hente. Ved at benytte en parameter til at bestemme typen, så kan man nøjes med at lave en metode til at downloade filer med. Det kan ses i de to if betingelser der er i metoden. Her tjekkes værdien af listType parameteren. Er værdien "backlog" så hentes en liste med Backlog Item og tildeles til en GridView datasource. Er værdien "case" så hentes i stedet en liste med CustomerCases, og datasourcen tildeles så denne liste. Derefter binder man så data til GridViewet, så GridViewet kan gemmes til en fil. Derefter sættes en række Response properties. Blandt andet sættes en AddHeader property, hvori det bestemmes hvilket navn filen skal have, samt, fil endelsen. Til sidst returneres en RedirectToAction, der kalder index metoden

```
public ActionResult ExportData(string listType)
        {
            var date = DateTime.Now;
            var filename = listType + "List" + date;
             var sw = new StringWriter();
            var htw = new HtmlTextWriter(sw);
            var gv = new GridView();
            gv.DataSource = "";
            if (listType == "backlog")
            {
                gv.DataSource = db.BacklogItems.ToList();
            }
            if (listType == "case")
            {
                gv.DataSource = db.CustomerCases.ToList();
            }
            gv.DataBind();
            Response.ClearContent();
            Response.Buffer = true;
            Response.AddHeader("content-disposition",
            "attachment; filename=" + filename + ".xls");
            Response.ContentType = "application/ms-excel";
            Response.Charset = "";
            gv.RenderControl(htw);
            Response.Output.Write(sw.ToString());
            Response.Flush();
            Response.End();

            return RedirectToAction("Index");
        }
```

5.2.1 Viewet

I Viewet er implementationen meget simpel. Det kræve blot en knap for hver at de typer lister man ønsker at hente. Der benyttes i dette tilfælde ikke nogen Angular controller. Her kaldes ActionResultet direkte fra Viewet. Det gøres ved at benytte et a tag, hvori man ved hjælp af Razor, kalder en Url.Action. Url.Action tager imod et Action navn (ExportData), et controller navn (DashBoard), og i dette tilfælde en RouteValue (new {listType = "case"})"). RouteValue er den eller de parametre man ønsker at sende til sit ActionResult. Alternativt kunne man bruge en knap der kaldte en function i Angular controlleren, der

indeholder et HttpPost request og så sende parameteren med det. Det virker dog som en anelse overkill i dette tilfælde

```
<a data-ng-href="@Url.Action("ExportData", "Dashboard",
        new {listType = "case"})" target="_self" class="add-case btn btn-blue-color">
        + Save to excel
</a>
```

5.3 BRUGEN AF SCRUM BOARDET

I dette sprint blev der gjort stort brug af Scrum Boardet. Der havde i de tidligere sprints ikke været det store behov for at bruge Scrum Boardet. Det skyldes at der ikke blev udarbejdet særligt mange backlog items i de sprints. Vi sørgede i høj grad at flytte rundt på backlog items, så Scrum Boardet konstant afspejlede den fase de forskellige Backlog items var i.

5.4 DELKONKLUSSION FOR SPRINT 3

Tredje sprint fløj efter min opfattelse afsted. Projektet havde nået til et sted hvor at kerne funktionaliteten var på plads. Det drejede sig i sprintet mest om at tilføjer mere at det samme, blot med andre typer af data. Der blev selvfølgelig også implementere to nye features. Der var dog ikke nogen større problemer ved at tilføje dem, og derfor må sprintet anses for at være overstået uden besvær.

6. Sprint4

6.1 SPRINT 4 PLANNING MEETING

Det var nu tid til at planlægge fjerde og sidste sprint. Teamet samledes som ved de tidligere sprint planning meetings og begyndte at udvælge backlog items fra product backloggen til sprint backloggen. Der ville i dette sprint være en del planlagt fravær fra teamet. Michael havde ferie, Og Chakib skulle have fri et par dage til private foretagender. Derfor var jeg den eneste til at udvikle nye features i det sidste sprint. Der var i dette sprint også kun 4 dage til rådighed. Grunde til det var at der om fredagen skulle være en præsentation af DashBoardet og Projekt Portalen. Der blev derfor udvalgt et enkelt Backlog Item. Den feature der skulle udarbejdes, var et diagram der kunne vise et bestemt antal cases og deres fremskridt. Det valget backlog item ses nedenfor.

Backlog Item id	Titel	Business Value	Estimat
6	diagram med cases og deres statusser.	80	25

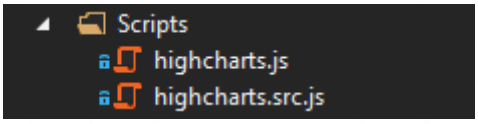
Beskrivelse	Accept kriterie
As a Creuna Admin **I want to** Se et diagram over de sager jeg er involveret i og Deres fremskidt. **So that** Man hurtigt kan få et overblik over de sager der Sidst har skiftet status. Samt se deres fremdrift.	• Vise de 5 sidste cases der er lavet status ændringer på • Følgende data skal vises. CaseId, titel, alle statusser.

6.2 OPSTART AF FEATURE " LAVE ET DIAGRAM MED CASES OG DERES STATUSSER"

Der skulle som sagt udvikles et diagram der kunne vise fremdrift på en række sager. Den diagram type der blev valgt var et plot/linje diagram. Det var hensigten at der i Diagrammets Y akse skulle være navne på de valgte cases. X aksen skulle bestå at de forskellige statusser en CustomerCase kan have.

For at lave diagrammet blev der valgt et JavaScript bibliotek ved navn HighChart. Der blev kigget på en hel del forskellige biblioteket, men jeg endte med at vælge HighChart. Det gjorde jeg b.la. fordi at bibliotek har en virkelig god dokumentation. Da det i mit tilfælde var første gang jeg skulle lave et diagram af denne type, så var dokumentationen i den grad vigtig. Derudover skulle api'et være nemt at implementer. Det viste det sig i den grad at være.

Der skulle kun tilføjes to filer til projektet, for at gøre brug at HighChart biblioteket. De to filer tilføjes til script mappen i projektet. Derudover tilføjes det også til BundleConfig filen i app_start mappen.



6.3 KODNING AF FEATURE " LAVE ET DIAGRAM MED CASES OG DERES STATUSSER"

6.3.1 Diagram controller

Der skal i diagrammet bruges to typer data. Den ene er en liste med CustomerCase objekter. Denne liste findes og sendes allerede i controlleren, og skal derfor ikke tilføjes. Den anden type data, den der mangler er en liste med status navne. Denne liste skal bruges til at udfylde værdier i x aksen på diagrammet. Igen skulle der oprettes et data objekt member, her kaldet projectStatus. Derefter hentes og tildeles alle status navne til projectStatus. Nedenfor ses implementationen.

```
projectStatus =  db.CustomerCaseStatus.Where(x => x.Order >= 15 ).Select( s => new
                        {
                         name = s.StatusName,
                        }),
```

6.3.2 Diagram Angular controller

Det er i Angular controlleren at diagrammet bliver skabt. Det er her den klart sværeste del af implementeringen ligger. Som ved tidlige features så bruges Success responset igen til at kalde en function i Angular Controlleren. Denne gang kalder den en $scope.chart funktion, der tager imod to parametre. Som beskrevet i afsnittet om diagram controlleren, så benyttes to typer data. Den en er en liste med CustomerCases. Den er repræsenteret i data.rows. Den anden er listen med projectStatus. Den er repræsentere i data. ProjectStatus.

```
$scope.chart(data.rows, data.projectStatus);
```

Nedenfor ses funktionen. Man kan splitte funktionen op i to dele. Den første del er hvor man henter data. I denne del hentes data ind via funktionen og bliver så gennem en række løkker, tilføjet til nogle Arrays. Disse arrays indeholder det data der skal fyldes i diagrammet.

```
$(function() {

                $scope.chart = function(item, cat) {
                    setTimeout(function () {
                    var dl = [];
                    for (var x = 0; x < item.length; x++) {
                        var t = item[x].title;
                        dl.push(t);
                    };
                    var dataTest = [];
                    var status = [];
                    for (var j = 0; j < cat.length; j++) {
                        var st = cat[j].name;
                        status.push(st);
                    }
                    for (var i = 0; i < item.length; i++) {
                        var d = [];
                        var n = [];
                        var na = "date";
                        n.push(na);
                        for (var k = 0; k < item[i].statusList.length; k++) {
                            var step = i;
                            d.push(step);
                        }
                        var nd =
                        {
                            name: n,
                            data: d
                        };
                        dataTest.push(nd);
                    };
```

Den anden del er implementationen af diagrammet. Her starter man med at tilføjet et HighCharts element til et div tag, via dets id (#container). Dernæst sættes en række attributter. Der findes en lang række attributter, man kan tilføje. De tre vigtigste er dog xAxis, yAxis og Series. Uden disse tre ville der ikke være noget data at vise. Selvom implementationen ser simpel ud, så er det langt fra en let opgave at få lavet diagrammet som man ønsker det. Især ikke hvis det skal laves noget funktionalitet som går ud over det dokumentationen beskriver.

```
                $('#container').highcharts({
                    title: {
                        text: 'Case status progress',
                        x: -20
                    },
                    xAxis: {
                        categories: status,
                    },
                    yAxis: {
                        title: {
```

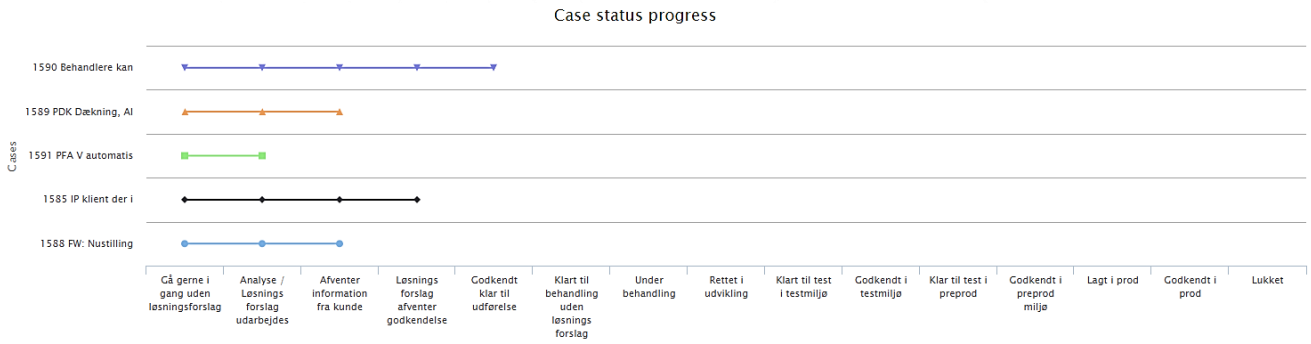
```
                text: 'Cases'
            },
            categories: dl,
        },
        tooltip: {
            valueSuffix: ""
        },
        legend: {
            enabled: false
        },
        series: dataTest,
    });}, 200);
};
```

6.3.3 Diagram Viewet

Der er ikke meget at forklare om implementationen af diagrammet i viewet. Der er nemlig stort set ikke noget arbejde at lave. Alt arbejdet lå som tidligere beskrevet i Angular controlleren. Det eneste det kræver er en enkelt line kode. Et div tag med et id, og nogle styles til at bestemme højde og vidde på diagrammet.

```
<div id="container" style="min-width: 310px; height: 400px; margin: 0 auto"></div>
```

Det færdig resultat ses på billedet nedenfor.



6.4 PRÆSENTATION AF DASHBOARDET

På præsentationen af DashBoardet og Projekt Portalen, var følgende personer tilstede, Jes Rasmussen og Birgitte Lundgren som repræsenterede projektleder gruppen. Bo Petersen og Casper Jensen som repræsentere udviklerene, og var dem der skulle overtage og videreudvikle applikationen. Ali Razvi som repræsenterede teamleder gruppen. Der udover var Michael Eiland, samt Chakib Benhaddou, og mig. To sidstnævnte skulle stå for præsentationen

På mødet startede vi med at fortælle lidt om de problemer som portalen og DashBoardet skulle afhjælpe. Dernæst forklarede vi kort om arkitekturen i applikationen, og de værktøjer vi havde brugt. Vi gav derefter en grundig forklaring af de forskellige funktionaliteter der var blevet implementeret. Først blev

funktionaliteterne vist på brugergrænsefladen. Derefter kom der en mere teknisk gennem gang af funktionaliteterne i udviklingsmiljøet. Dette var primært tilsigtet de to udviklere og team lederen.

Derefter var præsentationen slut og således også sprintet.

6.5 DELKONKLUSION FOR SPRINT4

Det fjerde sprint var et hektisk et af slagsen. Der var ikke meget tid til at implementer den sidste feature. Starten var forholdsvis smertefri. Her blev HighCharts biblioteket tilføjet til projektet og det var lige til. Data til diagrammet var også lige til at hente. Derimod løb jeg ind i store problemer med hensyn til udarbejdelse af diagrammet i Angular controlleren. Der skulle mange forsøg og mange rettelser til før at det lykkedes at blive færdig med det. Det færdige resultat er dog ikke det der i sidste enden vil blive brugt. Der vil blive tilføjet yderligere data til diagrammet. Her kan blandt andet nævnes dato for hver status ændring. Men det vigtigste af alt var at acceptance kriteriet var blevet opfyldt, og derved kunne backlog itemet kaldes for done. Det samme kunne sprintet.

7. Konklusion.

Jeg vil i dette afsnit evaluere på hele udviklingsforløbet. Jeg vil blandt andet komme med en konklusion på problemstillingen. Jeg vil også evalurer på brugen af udviklingsmetoden, samt de værktøjer der er blevet brugt. Derudover vil jeg komme med en evaluering af produktet, rapporten og til sidst komme med en afsluttende bemærkning.

7.1 PROCESSEN

7.1.2 Arbejdsgang

Der har på projektet været en enorm god arbejdsgang. Især har Chakib og mig arbejdet meget tæt sammen, og har fået et rigtig godt kammeratskab. Vi har komplimenteret hinanden rigtig godt. Vi har hver især nogle spids kompetencer og dem formåede vi at få gjort god brug af på tværs af opgaverne. Der har også været et rigtig godt sammenarbejde med Michael Eiland. Michael var ikke særligt aktiv i udarbejdelsen af DashBoardet, men han har stået klar med råd og vejledning, når der opstod problemer. Alt i alt har det været et rigtig godt arbejdsmiljø, vi har arbejdet i.

7.2 UDVIKLINGSMETODE.

Der er på projektet blevet brugt Scrum og Extreme Programming, som process styring og systemudviklings metoder. Dette var set i bakspejlet, det helt rigtige valgt. Der var fra start af ikke den store tvivl om at vi skulle arbejde Agilt. Ej heller om vi skulle bruge Scrum. Scrum var blevet brugt i tidligere projekter, senest på ProjektPortalen, som jeg også var en stor del af. Derfor var det, det

naturlige valg. Extreme Programming var ikke noget der blev gjort særlig meget brug af før, men der var nogle dele af XP som teamet rigtig gerne ville afprøve. Derfor var det også spændende at se om det gav noget til projektet. Med hensyn til Scrum så var det især de obligatoriske møder vi gjorde brug af. Især Sprint Review møderne var extremt givende. Ikke kun i forhold til dette projekt men også på en masse andre områder. Der blev på mødet ofte diskuteret om udvikling, hvor viden, meninger og holdninger blev delt. Jeg syntes i hvert fald at det var dybt fascinerende, at høre på rigtig dygtige udvikler diskutere om alverdens former for udvikling. Scrum Boardet benyttede vi i nogle perioder meget og i andre ikke så meget. Det var helt sikkert et område teamet kunne havde gjort mere ud af. Men ofte var det en eller to features der skulle udvikles. Samtidig var vi kun to udviklere på projektet. Så ofte gav det bare mere menig at snakke sammen. Skulle jeg gøre det om ville jeg nok benytte Scrum Boardet i lidt højere grad. Med hensyn til Extreme Programming så syntes jeg personligt at det var virkelig givende at udføre par programmering. Der var i hvert fald stor enighed i teamet om at det havde været godt for alle. Det er bestemt ikke sidste gang at jeg vil prøve kræfter med par programmering. Konklusion på valget af udviklingsmetode, må være at skulle jeg med den viden jeg har i dag, udvikle DashBoardet forfra. Så ville jeg benytte de helt samme udviklingsmetoder.

7.3 VÆRKTØJER

Der er på projektet blevet arbejdet med en række forskellige værktøjet og frameworks. De har givet en udfordringer på hver deres måde. Jeg tror ikke det kan udgås at der opstår problemer i et udviklingsprojekt. Især ikke når man bruger flere forskellige frameworks og udviklingssprog. Der har været flere værktøjer der har været spændende at arbejde med. Især AngularJS er enormt spændende, da det giver en muligheden for at lave rigtig fede ting i brugergrænseflade, uden at det bliver for tidskrævende. Jeg tror at AngularJS de næste år vil få et voldsomt boom rundt omkring i de danske udviklingsvirksomheder. Det er nemt at integrere, der findes super god dokumentation, der er masser tutorials på nettet, og så er det ikke mindst udviklet og vedligeholdt af Google. Det har i hvert fald alle forudsætninger for at slå igennem som det fortrukne Web applikations framework. Jeg vil også fremhæve MVC5 med Entity Framework Code First. Det er også super fedt arbejde med. Det giver en muligheden for at lave hurtige og datatunge applikationer på ingen tid. I hvert faldt i forhold til f.eks. en applikation udviklet i Asp.Net webforms. Man kan diskutere om tiden er løbet fra Razor. Jeg vil i hvert fald i langt højere grad foretrække at bruge Angular til at vise elementer i brugergrænsefladen. Visual Studio Online var også spændende at stifte bekendtskab med. Det er et kæmpe univers, og jeg vil ikke engang prøve at komme ind på de mulighede de ligger i det. Men en ting kan siges om det, og det er at udvikling i skyen er kommet for at blive.

7.4 PRODUKTET

Jeg vil i forhold til projektet komme den konklusion at vi stort set nåede i mål med det der blev forventet at Creuna og af os selv. Dashboardet er nu i en fase hvor det kan benyttes, og det løser helt basalt nogle af de problemer som Creuna havde i forhold til at se fremdrift og ændringer på sagerne. Det var fra start

af aldrig meningen at Dashboardet skulle være et komplet Dashboard. Meningen var at grundstenen skulle ligges og kerne funktionerne skulle være på plads. I hvert fad i en sådan grad af det kunne benyttes. Fremadrette skal der videreudvikles på DashBoardet, og der er massere muligheder for tilføje flere funktioner til det. Der blev på præsentationen snakket en del om muligheden for at integrere Trello, som er slags kanban board på nettet. Det bliver i hvert fald spændende at se hvad der kommer til at ske med DashBoardet og hele ProjektPortalen. Jeg er godt tilfreds med resultatet af DashBoardet. Der er altid ting man gerne ville havde lavet anderledes, men alt i alt så er jeg tilfreds. Når det kommer til spørgsmålet om vi var i stand til at udvikle et fyldt funktionelt DashBoard inde for tidsfristen. Så syntes jeg at svaret både er ja og nej. Ja i forhold til de backlog Items der nåede at blive udført. Men nej i forhold til de krav der blev stillet fra starten. Jeg vil ikke komme med nogle forklaring på hvorfor det ikke lykkedes, jeg vil blot sige at der fra Creunas side var stor tilfredshed med det vi havde opnået. De var enormt glade for at de nu kunne se en ende på de meget tidskrævende manuelle opgaver der lå i at holde sig opdateret.

7.5 RAPPORTEN

Jeg vil egentlig ikke komme med så mange ord omkring rapporten. Jeg vil blot fortælle at det har været en hård kamp at komme igennem det alene. Der er meget at holde styr på i skriveprocessen. Det kan til tider blive rimelig ensomt, at sidde og skrive alene. Dog vil jeg sige at jeg under omstændighederne er godt tilfreds med rapporten. Der er helt sikkert områder der kunne være beskrevet mere og andre mindre. Det gælder om at finde en balance i indholdet. Jeg syntes at det er lykkes i forhold til de præmisser der var.

7.6 AFSLUTTENDE ORD

Jeg vil her til sidst blot fortælle at jeg syntes at det har været en fantastisk rejse at udvikle Dashboardet. Det skyldes i den grad Creuna og de medarbejdere der arbejder der. De har været så ubeskriveligt søde, hjælpsomme og flinke. Det skyldes også mit sammenarbejde med Chakib, Michael og Christian som har været helt eminent. Chakib er blevet mere end et godt bekendtskab, og vi skal uden tvivl holde kontakten ved lige fremover. Michael har været en støtte i forhold til det udviklingsmæssige. Han har hverken blandet sig for meget eller for lidt. Sidst men ikke mindst så har Christian været som en mentor for mig. Ikke kun i forhold til udvikling, men bare i det hele taget. Han er om nogen manden at takke for at mit ophold hos Creuna har været så succesfuldt. Jeg på en og samme tid utrolig stolt men også ked af at skulle forlade Creuna. Men et nyt kapitel skal til at begynde og det ser jeg utrolig meget frem til.

8. Litteratur liste

8.1 WEBSITES

https://docs.angularjs.org/api

https://docs.angularjs.org/guide

http://api.highcharts.com/highcharts

http://www.highcharts.com/docs/getting-started/your-first-chart

http://api.jquery.com/

http://www.asp.net/mvc/overview/getting-started/getting-started-with-ef-using-mvc/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application

http://www.mountaingoatsoftware.com/agile/scrum

http://www.stackoverflow.com

8. Appendix

Indholdsfortegnelse

9.1 APPENDIX 1 – KILDEKODE TIL DASHBOARDET.

9.1.1 Modellen

```
//using System.Activities.Statements;
using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Dynamic;
using System.IO;
using System.Web;
using Newtonsoft.Json;
using Newtonsoft.Json.Converters;
using PFMWeb.Models.PFM.TGProxy;
using TFSProxy;
using TGProxy;

namespace PFMWeb.Models.PFM
{
    public class TeamProject
```

```
{

    public int TeamProjectId { get; set; }

    [Required]
    [Display(Name = "Team name")]
    public string Name { get; set; }

    //How to Find TG
    //How to Find TFS
    public virtual List<TeamProjectTFS> TeamProjects { get; set; }
    public virtual List<TeamProjectTG> TeamProjectTGs { get; set; }
    public virtual List<BacklogItem> BacklogItems { get; set; }
    public virtual List<Contract> Contracts { get; set; }
    public virtual List<ContractHourItem> ContractHourItems { get; set; }
    public virtual List<Project> Projects { get; set; }
}

public class Contract
{
    public int ContractId { get; set; }
    public string Name { get; set; }
    public virtual List<ContractTemplate> ContractTemplates { get; set; }
    public virtual List<ContractHourItem> ContractHourItems { get; set; }
    public DateTime LastUpdated { get; set; }
    //public virtual List<TeamProject> TeamProjects { get; set; }
    public int TeamProjectTeamprojectId { get; set; }
}

public class TeamProjectTG
{
    public int TeamProjectTGId { get; set; }
    public int Prj_ID { get; set; }
    public string ProjectName { get; set; }
    public bool MatchByEmne { get; set; }
    public TGAccountType AccountType { get; set; }
    public virtual ICollection<ContractHourItem> ContractHourItems { get; set; }


    //How to Find TG
    //How to Find TFS
}

public class ContractTemplate
{
    public int ContractTemplateId { get; set; }
    public string Name { get; set; }
    public ContractTemplateType ContractTemplateType { get; set; }
    public DateTime PeriodStart { get; set; }
    public DateTime PeriodEnded { get; set; }
    public virtual List<TeamProjectTG> TeamProjectTGs { get; set; }
    public virtual List<ContractHourItem> ContractHourItems { get; set; }
    public DateTime LastUpdated { get; set; }
    public int ExpectedHours { get; set; }
    public ExpecteType ExpecteType { get; set; }
    public virtual Contract Contract { get; set; }

}

public enum ContractTemplateType
```

```
{
    Custom = 0,
    RepeatMonth = 1
}

public class ContractHourItem
{
    public int ContractHourItemId { get; set; }
    public string Name { get; set; }
    public DateTime PeriodStart { get; set; }
    public DateTime PeriodEnded { get; set; }
    public int ExpectedHours { get; set; }
    public ExpecteType ExpecteType { get; set; }
    public ContractTemplate ContractTemplate { get; set; }
    public virtual List<TeamProjectTG> TeamProjectTGs { get; set; }
    public virtual List<EntryTG> EntryTGs { get; set; }
    public DateTime LastUpdated { get; set; }
    public decimal TotalHours { get; set; }
    public OnTrackState OnTrackState { get; set; }
    public bool JobDone { get; set; }
    public virtual Contract Contract { get; set; }

}

public enum OnTrackState
{
    Unknown = 0,
    NoTheWay = 1,
    NotAchieved = 2,
    Reached = 3
}

public class EntryTG
{
    public int Id { get; set; }
    public decimal Quantity { get; set; }
    public Person Person { get; set; }
    public DateTime Date { get; set; }
    public string Description { get; set; }
    public string Unit { get; set; }
    public decimal UnitPrice { get; set; }
    public int itm_ID { get; set; }
    public int Prj_ID { get; set; }
    public virtual TGEntry TGEntry { get; set; }
    public virtual int? TGEntryId { get; set; }
    public DateTime LastUpdated { get; set; }
    public virtual ICollection<ContractHourItem> ContractHourItems { get; set; }
}

public class ActivityEntryTG
{
    public int ActivityEntryTGId { get; set; }
    public int Prj_ID { get; set; }

    public string Project { get; set; }
    public string Resource { get; set; }
    public decimal? Quantity { get; set; }

    public string Organisation { get; set; }
    public int Kode { get; set; }
```

```
        public int Sekvens { get; set; }
        public string ProjectAndNumber { get; set; }

        public string Enhed { get; set; }

        public string BudgetTimer { get; set; }

        public string Beskrivelse { get; set; }

        public string Aktivitet { get; set; }
        public string Disponibel { get; set; }


        public DateTime LastUpdated { get; set; }
        public bool IsDeleted { get; set; }

    }

    public enum ExpecteType
    {
        Must100Procent = 0,
        MaxIfNeeded = 1,
        IfPossible = 2,
    }

    public class TeamProjectTFS
    {
        public int TeamProjectTFSId { get; set; }
        public int TeamProjectsTFS { get; set; }
        public string AreaPath { get; set; }
        //How to Find TG
        //How to Find TFS
    }

    public class Project
    {
        public int ProjectId { get; set; }
        public string Projectname { get; set; }
        public bool MatchByEmne { get; set; }
        public TGAccountType AccountType { get; set; }
        public virtual ICollection<ContractHourItem> ContractHourItems { get; set; }
    }

    public class Person
    {
        public int PersonId { get; set; }
        public string Name { get; set; }
        public string Email1 { get; set; }
        public string Email2 { get; set; }
        public string Email3 { get; set; }
        public int? TGid { get; set; }
        public string TFSName { get; set; }
        public virtual List<PersonIdentity> PersonIdentitys { get; set; }
    }

    public class PersonIdentity
    {
        public int PersonIdentityId { get; set; }
        public virtual Person PersonInfo { get; set; }
```

```
        public string UserName { get; set; }
    }

    public class PersonConfirm
    {
        public int PersonConfirmId { get; set; }
        public virtual Person Person { get; set; }
        public Guid ConfirmGuid { get; set; }
        public DateTime Created { get; set; }
        public string Email { get; set; }
        public bool IsConfirmed { get; set; }
        public DateTime? IsConfirmedUpdatedAt { get; set; }
        public string UserName { get; set; }
    }

    public class LogMail
    {
        public int LogMailId { get; set; }
        public string ToAdrress { get; set; }
        public string CCAdrress { get; set; }
        public string BCCAdrress { get; set; }
        public string FromAdrress { get; set; }
        public string Subject { get; set; }
        public string Body { get; set; }
        public string Headers { get; set; }
        public DateTime SendtTime { get; set; }
    }

    public class CustomerCase
    {
        public int CustomerCaseId { get; set; }
        public int ServiceSite2007ItemId { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public string Status { get; set; }
        public string AttachmentList { get; set; }
        public string Commentary { get; set; }
        [DataType(DataType.Date)]
        public DateTime CreatedAt { get; set; }
        public string ExpectedRelease { get; set; }
        public int? Priority { get; set; }
        public string RettetInReleaseNr { get; set; }
        public string ErrorInReleaseNr { get; set; }
        public string Consequence { get; set; }
        [DataType(DataType.Date)]
        public DateTime LastModifyed { get; set; }
        public string Estimate { get; set; }
        public bool Active { get; set; }
        public string FileUrl { get; set; }
        public ActivityEntryTG ActivityTG { get; set; }
        public string SpId { get; set; }
        public string TeamProject { get; set; }
        public TeamProject Team { get; set; }
        public FileUploadDb FileUploadDb { get; set; }
        public virtual List<CaseFiles> CaseFileses { get; set; }
        public virtual List<CustomerCaseStatusHistory> CustomerCaseStatusList { get; set;
}
        //public virtual CustomerCaseStatus CustomerCaseStatusName { get; set; }
        //public virtual List<string> StatusList { get; set; }
        public virtual CustomerCaseStatusHistory CustomerCaseStatusHistory { get; set; }
```

```
        public IEnumerator<CustomerCase> GetEnumerator()
        {
            throw new NotImplementedException();
        }


    }

    public class CustomerCaseStatusHistory
    {
        public int CustomerCaseStatusHistoryId { get; set; }
        public DateTime Date { get; set; }
        public virtual CustomerCaseStatus CustomerCaseStatus { get; set; }
        public  CustomerCase CustomerCase { get; set; }
        public int CustomerCaseIdRef{ get; set; }
        public string User { get; set; }
    }

    public class CustomerCaseStatus
    {
        public int CustomerCaseStatusId { get; set; }
        public string StatusName { get; set; }
        public int Order { get; set; }
        public int Steps { get; set; }
    }

    public class CustomerCaseColumns
    {
        public int Id { get; set; }
        public string ColumnId { get; set; }
        public string Name { get; set; }
        public bool ColumnChoosen { get; set; }
        public string Type { get; set; }
        public bool Multiple { get; set; }
        public virtual List<Object> Values { get; set; }
        public string DropDownName { get; set; }
        public virtual List<string> FilterDatasList { get; set; }

    }

    public class CustomerCaseColumnMapping
    {
        public int Id { get; set; }
        public CustomerCaseViews CustomerCaseViews { get; set; }
        public CustomerCaseColumns CustomerCaseColumns { get; set; }
        public virtual List<CustomerCaseFilterData> CustomerCaseFilterData { get; set; }
    }

    public class CustomerCaseViews
    {
        public int Id { get; set; }
        public string FilterName { get; set; }
        public virtual List<CustomerCaseColumnMapping> CustomerCaseColumnMapping { get; s
et; }
        public string User { get; set; }

    }

    public class CustomerCaseFilterData
    {
```

```
        public int Id { get; set; }
        public CustomerCaseColumns CustomerCaseColumns { get; set; }
        public String FilterInput { get; set; }
        public CustomerCaseColumnMapping CustomerCaseColumnMapping { get; set; }
    }


    public class BacklogItem
    {
        [Display(Name = "ID")]
        public int BacklogItemId { get; set; }

        public TeamProject Team { get; set; }

        [DisplayFormat(DataFormatString = "{0,20}")]

        [Required(ErrorMessage = "Title is required")]
        [Display(Description = "Title")]
        public string Title { get; set; }

        public decimal HoursUsedTG { get; set; }
        public decimal? HoursBudgetTG { get; set; }
        public decimal HoursRemaningWorkTFS { get; set; }
        public decimal HoursOnBudget { get; set; }

        [Range(0, 10000, ErrorMessage = "Value must be between 0 and 10000")]
        [Display(Name = "Efford")]
        public decimal? EffordTFS { get; set; }

        public int AreaIdTFS { get; set; }

        [Required]
        [Display(Name = "Area")]
        public string AreaPathTFS { get; set; }

        // Removed is only required in GUI not in Backend shoud this check be implemented
like this?
        // As an alternativ use a ViewModel
        //[Required]
        [Display(Name = "Assigned To")]
        [DisplayFormat(NullDisplayText = "Not assigned")]
        public string AssignedToTFS { get; set; }

        [Range(-1, int.MaxValue, ErrorMessage = "Value must grater than or equal to -1")]
        [Display(Name = "Backlog Priority")]
        public int BacklogPriorityTFS { get; set; }

        [Display(Name = "Business Value")]
        public decimal? BusinessValueTFS { get; set; }

        public string ChangedByTFS { get; set; }
        public DateTime ChangedDateTFS { get; set; }
        public DateTime? ClosedDateTFS { get; set; }

        // Removed is only required in GUI not in Backend shoud this check be implemented
like this?
        // As an alternativ use a ViewModel
        //[Required]
        [Display(Name = "Description")]
        public string DescriptionTFS { get; set; }
```

```
        public int IterationIdTFS { get; set; }

        [Display(Name = "Iteration Path")]
        public string IterationPathTFS { get; set; }

        public string NodeNameTFS { get; set; }

        [Required]
        [Display(Name = "Reason", Description = "Reason")]
        public string ReasonTFS { get; set; }

        [Required]
        [Display(Name = "State")]
        public string StateTFS { get; set; }

        public string TagsTFS { get; set; }
        public string TitleTFS { get; set; }
        public bool IsDeletedTFS { get; set; }
        public bool dummy { get; set; }

        public virtual List<Person> Persons { get; set; }
        public virtual List<TgActivityEntry> TGActivityEntry { get; set; }
        public virtual List<TGEntry> FromTGEntrys { get; set; }
        public virtual List<TFSTaskItem> FromTFSTasksItem { get; set; }
        public virtual List<TFSBacklogItem> FromTFSBacklogItem { get; set; }
        public virtual List<TFSBugItem> FromTFSBugItem { get; set; }
        public decimal? HoursToEffordFactor { get; set; }
        public string BugReproStepsTFS { get; set; }
        public string BugSystemInfoTFS { get; set; }
        public string BugFoundInTFS { get; set; }
        public string BugSeverityTFS { get; set; }
        public EnmBacklogType BacklogType { get; set; }
        public int ServiceSite2007ItemId { get; set; }
    }

    public enum EnmBacklogType
    {
        Other = 0,
        Backlog = 1,
        Bug = 2,
        Feature = 3
    }

    public class SolutionInfo
    {
        public int SolutionInfoId { get; set; }

        [Display(Name = "Solution name")]
        public string SolutionName { get; set; }

        [Display(Name = "Primary domain")]
        public string PrimaryDomain { get; set; }

        [Display(Name = "Other domains")]
        public string OtherDomains { get; set; }

        [Display(Name = "Features domains")]
        public string FeaturesDomains { get; set; }
```

```
        [DataType(DataType.MultilineText)]
        [Display(Name = "Solution data comments")]
        public string CommentsToMaster { get; set; }

        [Required]
        [Display(Name = "Service level agreement ")]
        public ServiceLevelAgreement? ServiceLevelAgreement { get; set; }

        [DataType(DataType.MultilineText)]
        [Display(Name = "Service level comments")]
        public string CommentsToService { get; set; }

        [Required]
        [Display(Name = "Solution type  ")]
        public CategoriesSolutionType? CategoriesSolutionType { get; set; }


        // public CategoriesSubType? CategoriesSubType { get; set; }
        [Required]
        [Display(Name = "Solution subtypes")]
        public virtual List<CategoriesSubType> CategoriesSubTypesList { get; set; }

        public virtual List<string> SubTypeList { get; set; }
        [Required]
        [Display(Name = "Platform type")]
        public CategoriesPlatformType? CategoriesPlatformType { get; set; }

        [DataType(DataType.MultilineText)]
        [Display(Name = "Categorie comments")]
        public string CommentsToCategories { get; set; }

        public virtual List<Person> Persons { get; set; }
        [Required]
        [Display(Name = "Project manager")]
        public virtual List<string> ProjectManagerList { get; set; }

        [Required]
        [Display(Name = "System architect")]
        public virtual List<string> SystemArchitectList { get; set; }

        [Required]
        [Display(Name = "Primary developer")]
        public virtual List<string> PrimaryDeveloperList { get; set; }

        [Required]
        [Display(Name = "Developer")]
        public virtual List<string> DevelopersList { get; set; }

        [DataType(DataType.MultilineText)]
        [Display(Name = "Team comments")]
        public string CommentsToTeam { get; set; }

        [Display(Name = "Project backlog is in Vso")]
        public bool ProjectBacklogIsInVso { get; set; }

        [Display(Name = "Project tool link")]
        public string ProjectToolLink { get; set; }

        [Display(Name = "Code repository in Vso")]
        public bool CodeRepositoryInVso { get; set; }
```

```
        [Display(Name = "Code repository link")]
        public string CodeRepositoryLink { get; set; }

        [Required]
        [Display(Name = "Version control")]
        public VersionControlFormat? VersionControlFormat { get; set; }

        [Display(Name = "Application insight Dev")]
        public bool ApplicationInsightsDev { get; set; }

        [Display(Name = "Application insight Test")]
        public bool ApplicationInsightsTest { get; set; }

        [Display(Name = "Application insight PreProd")]
        public bool ApplicationInsightsPreProd { get; set; }

        [Display(Name = "Application insight Prod")]
        public bool ApplicationInsightsProd { get; set; }

        [Display(Name = "Muscula Dev")]
        public bool MusculaDev { get; set; }

        [Display(Name = "Muscula Test")]
        public bool MusculaTest { get; set; }

        [Display(Name = "Muscula PreProd")]
        public bool MusculaPreProd { get; set; }

        [Display(Name = "Muscula Prod")]
        public bool MusculaProd { get; set; }

        [Display(Name = "totalValidator Dev")]
        public bool TotalValidatorDev { get; set; }

        [Display(Name = "totalValidator Test")]
        public bool TotalValidatorTest { get; set; }

        [Display(Name = "totalValidator PreProd")]
        public bool TotalValidatorPreProd { get; set; }

        [Display(Name = "totalValidator Prod")]
        public bool TotalValidatorProd { get; set; }

        [DataType(DataType.MultilineText)]
        [Display(Name = "Utility comments")]
        public string CommentsToUtilities { get; set; }

        [DataType(DataType.MultilineText)]
        [Display(Name = "Comments")]
        public string Comments { get; set; }

        public virtual List<SolutionPersons> SolutionPersonses { get; set; }
        public virtual List<CategorieSubTypes> CategorieSubTypeses { get; set; }



    }
```

```
public class SolutionPersons
{
    public int Id { get; set; }
    public int PersonId { get; set; }
    public SolutionRoles SolutionRoles { get; set; }
    public string SolutionName { get; set; }
    public Person Person { get; set; }
    public SolutionInfo SolutionInfo { get; set; }


}

public class CategorieSubTypes
{
    public int Id { get; set; }
    public int SubTypeId { get; set; }
    public string SubType { get; set; }
}


public enum SolutionRoles
{
    ProjectManager,
    SystemArchitect,
    PrimaryDeveloper,
    Developer
}

public enum ServiceLevelAgreement
{
    Level1,
    Level2,
    Level3
}

public enum CategoriesSolutionType
{
    CorporateWebsite,
    Campaign,
    Intranet,
    ECommerce,
    Mobile,
    App,
    Tool,
    Other
}

public enum CategoriesSubType
{
    CorporateWebsite,
    B2B,
    B2C,
    Campaign,
    Intranet,
    ECommerce,
    Mobile,
    App,
    Tool,
    Other
```

```
}

public enum CategoriesPlatformType
{
    SharePoint,
    SiteCore,
    EpiServer,
    OtherCms,
    AspNetWebForm,
    AspNetMvc,
    Other
}

public enum VersionControlFormat
{
    Git,
    Tfvc,
    Other
}

public class FilterDb
{
    public int Id { get; set; }
    public int UserId { get; set; }
    public SavedFilters SavedFilters { get; set; }
    public virtual List<FiltersBacklogItem> FiltersBacklogItems { get; set; }
    public String Name { get; set; }

}

public class FiltersBacklogItem
{
    public int Id { get; set; }
    public string ColumnId { get; set; }
    public string Name { get; set; }
    public bool ColumnChoosen { get; set; }
    public string Type { get; set; }
    public bool Multiple { get; set; }
    public string CssWidth { get; set; }
    public virtual List<Object> Values { get; set; }
    public string DropDownName { get; set; }
    public virtual List<string> FilterDatasList { get; set; }

}

public class SavedFilters
{
    public int Id { get; set; }
    public string FilterName { get; set; }
    public int UserId { get; set; }
    public virtual List<FilterMappingTest> FilterMappingTest { get; set; }
    public string User { get; set; }
    public string Name { get; set; }
}

public class FilterMapping
{
    public int Id { get; set; }
    //  public SavedFilters SavedFilters { get; set; }
    // public FiltersBacklogItem FiltersBacklogItem { get; set; }
```

```
}

public class FilterMappingTest
{
    public int Id { get; set; }
    public SavedFilters SavedFilters { get; set; }
    public FiltersBacklogItem FiltersBacklogItem { get; set; }
    public virtual List<FilterData> FilterDataList { get; set; }
}


public class Filters
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Type { get; set; }
    public bool Multiple { get; set; }
    public bool Choosen { get; set; }

}

public class FilterData
{
    public int Id { get; set; }
    public FiltersBacklogItem FiltersBacklogItems { get; set; }
    public String FilterInput { get; set; }
    public FilterMappingTest FilterMappingTest { get; set; }
}

public class FilterInput
{
    public int Id { get; set; }

    public string FilterInputData { get; set; }
}

public class DropDowns
{
    public string Name { get; set; }
    public virtual List<Object> Value { get; set; }
}

public class DropDownList
{
    public string Name { get; set; }
    public bool Choosen { get; set; }
}


public class FileUploadDb
{

    public int FileUploadDbId { get; set; }
    public int GroupId { get; set; }
    public string FileName { get; set; }
    public int CustomerCaseId { get; set; }

    public byte[] File { get; set; }
}
```

```
    public class FilesViewModel
    {
        [DisplayName("Select File To Upload")]
        public IEnumerable<HttpPostedFileBase> File { get; set; }
    }

    public class CaseFiles
    {
        public int Id { get; set; }
        public string FileName { get; set; }
        public byte[] File { get; set; }
        public CustomerCase CustomerCase { get; set; }
    }

        public virtual Currency Currency { get; set; }
    }
    */


    public class Logging
    {
        public int Id { get; set; }
        public DateTime Timestamp { get; set; }
        public String Code { get; set; }
        public String Message { get; set; }
        public String User { get; set; }
        public String Url { get; set; }
    }
}
```

9.1.2 Controlleren

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Net;
using System.Web;
using System.Web.Mvc;
using System.Web.UI;
using System.Web.UI.WebControls;
using Microsoft.AspNet.Identity.EntityFramework;
using PFMWeb.Logins;
using PFMWeb.Models.DB;
using PFMWeb.Models.PFM;

namespace PFMWeb.Controllers
{
    [Authorize(Roles = RolesKnown.Admin)]
    public class DashboardController : Controller
    {
        private PFMcontext db = new PFMcontext();
```

```
        private readonly LoginManager _loginManager = new LoginManager();
        private IdentityDbContext alpha = new IdentityDbContext();

        public JsonResult MainDashboardResult()
        {
            var customers = db.TeamProjects.Select(x => x).ToList();

            var data = new
            {
                customers = customers.Select(c => new
                {
                    name = c.Name,
                    id = c.TeamProjectId,
                    projects = c.TeamProjectTGs.Select(p => new
                    {
                        projectName = p.ProjectName,
                        prjId = p.Prj_ID,

                    })
                })
            };

            return Json(data, JsonRequestBehavior.AllowGet);
        }
        public JsonResult DashboardResult(int? id)
        {

            DateTime date = DateTime.Now;
            var dateFrom = date.AddDays(-50);
            var dateTo = date;


            var customers = db.TeamProjects.Select(x => x).ToList();
            if (id != null)
                {

                var backlog = db.TeamProjects.Where(b => b.TeamProjectId == id).Select(b
=> b.BacklogItems).Select(ba => ba);
                var cases = db.CustomerCases.Where(c => c.Team.TeamProjectId == id).Order
By(c => c.LastModifyed).Take(30).ToList();
                var statusListHistory = new  List<CustomerCaseStatusHistory>();
                foreach (CustomerCase c in cases)
                {
                    var child = db.CustomerCaseStatusHistory.Where(cc => cc.CustomerCaseI
dRef == c.CustomerCaseId || cc.CustomerCase.CustomerCaseId == c.CustomerCaseId).ToList();
                    c.CustomerCaseStatusList.AddRange(child);
                    statusListHistory.AddRange(child);
                }
                var statusList = db.CustomerCaseStatus.Select(x => x).ToList();
                var statusNameList = db.CustomerCaseStatus.Where(x => x.Order >= 15 ).ToL
ist();
                if (cases.Count == 0)
                {

                    ViewBag.Nothing = "No Cases on this project";

                }

                var data =
                       new
```

```
                        {
                            rows =
                                cases.Select(
                                    x =>
                                        new
                                        {
                                            active = x.Active,
                                            attachmentList = x.AttachmentList,
                                            commentary = x.Commentary,
                                            consequence = x.Consequence,
                                            createdAt = x.CreatedAt.ToString("g"),
                                            customerCaseId = x.CustomerCaseId,
                                            description = x.Description,
                                            errorInReleaseNr = x.ErrorInReleaseNr,
                                            estimate = x.Estimate,
                                            expectedRelease = x.ExpectedRelease,
                                            fileUrl = x.FileUrl,
                                            lastModifyed = x.LastModifyed.ToString("g"),
                                            priority = x.Priority,
                                            rettetInReleaseNr = x.RettetInReleaseNr,
                                            serviceSite2007ItemId = x.ServiceSite2007ItemI
d,
                                            status = x.Status,
                                            title = x.Title,
                                            selected = false,
                                            statusList =
                                                x.CustomerCaseStatusList.Where(y => y.Date
 < dateTo && y.Date > dateFrom).Select(
                                                    y =>
                                                        new
                                                        {
                                                            caseId = x.CustomerCaseId,
                                                            title = x.Title,
                                                            order = y.CustomerCaseStatus.O
rder,
                                                            name = y.CustomerCaseStatus.St
atusName,
                                                            date = y.Date.ToString("g"),
                                                            step = y.CustomerCaseStatus.St
eps,
                                                            user =
                                                                alpha.Users.Where(z => z.I
d == y.User)
                                                                    .Select(v => v.UserNam
e)
                                                                    .FirstOrDefault(),
                                                        }),
                                            order =
                                                x.CustomerCaseStatusList.Select(y => y.Cus
tomerCaseStatus.Order)
                                                    .LastOrDefault(),

                                        }).AsEnumerable(),
                            statusAccess =
                                statusList.Select(
                                    y => new { id = y.CustomerCaseStatusId, name = y.Statu
sName, order = y.Order, }),
                            backlogs =
                                backlog.Select(
```

```
                                b =>
                                    new
                                    {
                                        items =
                                            b.Select(
                                                x =>
                                                    new
                                                    {
                                                        id  = x.BacklogItemId,
                                                        state = x.StateTFS,
                                                        title = x.Title,
                                                        priority = x.BacklogPriorityTF
S,
                                                        person = x.AssignedToTFS
                                                    })
                                    }),
                            customers =
                                customers.Select(
                                    c =>
                                    new
                                    {
                                        name = c.Name,
                                        id = c.TeamProjectId,
                                        projects =
                                            c.TeamProjectTGs.Select(
                                                p => new { projectName = p.ProjectName
, prjId = p.Prj_ID, })
                                    }),
                            thisCustomer =
                                  db.TeamProjects.Where(x => x.TeamProjectId == id).Select
(x => new
                                  {
                                      name = x.Name

                                  }),
                            projectStatus =  db.CustomerCaseStatus.Where(x => x.Order >= 1
5 ).Select( s => new
                            {
                             name = s.StatusName,
                            }),

                        };
                return Json(data, JsonRequestBehavior.AllowGet);

            }
            else {
            var data =
                      new
                      {
                        customers =
                              customers.Select(
                                  c =>
                                      new
                                      {
                                          name = c.Name,
                                          id = c.TeamProjectId,
                                          projects =
                                              c.TeamProjectTGs.Select(
                                                  p => new { projectName = p.ProjectName,
 prjId = p.Prj_ID, })
```

```
                                    }),
                        thisCustomer =
                              db.TeamProjects.Where(x => x.TeamProjectId == id).Select(
x => new
                              {
                                  name = x.Name

                              })

                    };
            return Json(data, JsonRequestBehavior.AllowGet);
            }



        }

        [Authorize(Roles = RolesKnown.Creuna)]
        public ActionResult DashboardPM()
        {
            ViewBag.Title = "Dashboard PM";

            return View();
        }
        [Authorize(Roles = RolesKnown.Admin)]
        public ActionResult DashboardAdmin(int? id)
        {

            ViewBag.id = id;
            ViewBag.Title = "Dashboard Admin";


            return View();
        }

        [Authorize(Roles = RolesKnown.Customer)]
        public ActionResult DashboardCustomer()
        {
            ViewBag.Title = "Dashboard Customer";

            return View();
        }

        public ActionResult MainDashboard()
        {
            ViewBag.Title = "Home Page";

            return View();
        }

        // GET: Dashboard
        public ActionResult Index()
        {
            return View();
        }

        // GET: Dashboard/Details/5
        public ActionResult Details(int? id)
        {
            if (id == null)
```

```
            {
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            TeamProject teamProject = db.TeamProjects.Find(id);
            if (teamProject == null)
            {
                return HttpNotFound();
            }
            return View(teamProject);
        }

        // GET: Dashboard/Create
        public ActionResult Create()
        {
            return View();
        }

        // POST: Dashboard/Create
        // To protect from overposting attacks, please enable the specific properties you
 want to bind to, for
        // more details see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Create([Bind(Include = "TeamProjectId,Name")] TeamProject tea
mProject)
        {
            if (ModelState.IsValid)
            {
                db.TeamProjects.Add(teamProject);
                db.SaveChanges();
                return RedirectToAction("Index");
            }

            return View(teamProject);
        }

        // GET: Dashboard/Edit/5
        public ActionResult Edit(int? id)
        {
            if (id == null)
            {
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            TeamProject teamProject = db.TeamProjects.Find(id);
            if (teamProject == null)
            {
                return HttpNotFound();
            }
            return View(teamProject);
        }

        // POST: Dashboard/Edit/5
        // To protect from overposting attacks, please enable the specific properties you
 want to bind to, for
        // more details see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Edit([Bind(Include = "TeamProjectId,Name")] TeamProject teamP
roject)
        {
```

```
            if (ModelState.IsValid)
            {
                db.Entry(teamProject).State = EntityState.Modified;
                db.SaveChanges();
                return RedirectToAction("Index");
            }
            return View(teamProject);
        }

        // GET: Dashboard/Delete/5
        public ActionResult Delete(int? id)
        {
            if (id == null)
            {
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            TeamProject teamProject = db.TeamProjects.Find(id);
            if (teamProject == null)
            {
                return HttpNotFound();
            }
            return View(teamProject);
        }

        // POST: Dashboard/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public ActionResult DeleteConfirmed(int id)
        {
            TeamProject teamProject = db.TeamProjects.Find(id);
            db.TeamProjects.Remove(teamProject);
            db.SaveChanges();
            return RedirectToAction("Index");
        }

        protected override void Dispose(bool disposing)
        {
            if (disposing)
            {
                db.Dispose();
            }
            base.Dispose(disposing);

        }

        public ActionResult ExportData(string listType)
        {
            var date = DateTime.Now;
            var filename = listType + " List " + date;
            var gv = new GridView();
            gv.DataSource = "";
            if (listType == "backlog")
            {
                gv.DataSource = db.BacklogItems.ToList();
            }
            if (listType == "case")
            {
                gv.DataSource = db.CustomerCases.ToList();
            }
            gv.DataBind();
```

```
            Response.ClearContent();
            Response.Buffer = true;
            Response.AddHeader("content-
disposition", "attachment; filename=" + filename + ".xls");
            Response.ContentType = "application/ms-excel";
            Response.Charset = "";
            var sw = new StringWriter();
            var htw = new HtmlTextWriter(sw);
            gv.RenderControl(htw);
            Response.Output.Write(sw.ToString());
            Response.Flush();
            Response.End();
            return RedirectToAction("Index");
        }
    }
}
```

9.1.3 AngularJS App.js

```
"use strict";


var portal = angular.module('portal', ['LocalStorageModule']);

portal.config(function ($locationProvider) {
    $locationProvider.html5Mode(true);
    $locationProvider.hashPrefix('!');
})
.config([
    'localStorageServiceProvider', function (localStorageServiceProvider) {
        localStorageServiceProvider.setPrefix('creuna');

    }
]);
```

9.1.4 AngularJS Controller Dashboard.js

```
"use strict";
portal.controller('dashboardController', [
    '$scope', '$http', '$location', '$q', 'localStorageService', 'mainDashBoardData', fun
ction($scope, $http, $location, $q, localStorageService, mainDashBoardData) {
        $scope.rows = [];
        $scope.dateFrom = [];
        $scope.dateTo = [];
        $scope.backlogs = [];
        $scope.customers = [];
        $scope.statusAccess = [];
        $scope.statusListsCreuna = [];
        $scope.statusListsCustomer = [];
        $scope.statusListsAll = [];
        $scope.statusLists = [];
        $scope.overallStatus = [];
        $scope.customerName = [];

        $scope.maindashboarddata = mainDashBoardData;



        $scope.max = 200;
        var canceler;
```

```
        // Refresh priority Todoo button
        $scope.Refresh = function(id) {
            localStorageService.add('Customer', id);
            $scope.maindashboarddata.setFac(id);
            var data = {
                id: $scope.maindashboarddata.getFac()
            };
            $scope.output = data;
            $scope.loadFilters();


        };
        $("#menu-toggle").click(function(e) {
            e.preventDefault();
            $("#wrapper").toggleClass("active");
        });

        // Select Row item and do Stuff - Still in progress
        $scope.select = function(item) {
            $scope.selected = item;
            //        console.log($scope.selected);
        };
        $scope.showCustomerCase = function(id) {
            var url = '/CustomerCases/Edit/' + id;
            window.location = url;

        };
        $scope.showBacklog = function(id) {
            var url = '/Backlog/Edit/' + id;
            window.location = url;

        };
        $scope.downloadList = function (id) {
            alert(id);
            $http({
                url: '/Dashboard/ExportData/',
                method: "POST",
                data: id,
            })
.success(function (data, status, headers, config) {
             alert(id);
}).error(function (data, status, headers, config) {
    alert(id);
});

        };

        //Loads the DashboardResult(MVC Controller Method) into the angular controller
        $scope.loadFilters = function() {

            if (canceler) {
                canceler.resolve();
            }

            canceler = $q.defer();

            var urls = {
                "local": '/Scripts/json.aspx',
                "project": '/Dashboard/DashboardResult'
```

```
        };


        $scope.output = {
            id: localStorageService.get('Customer')
        };
        var httpRequest = $http(
            {
                method: 'POST',
                url: urls.project,
                data: $scope.output,
                timeout: canceler.promise
            })
            .success(function(data, status) {
                $scope.error = null;
                $scope.formElements = data.formElements;

                $scope.rows = data.rows;

                $scope.totals = data.totals;
                $scope.customerName = data.thisCustomer;
                $scope.customers = data.customers;
                $scope.backlogs = data.backlogs;
                $scope.projectStats = data.projectStatus;
                $scope.chart(data.rows, data.projectStatus);


                $scope.getAllStatusNow();
                $scope.statusListFunc(0);

                if (data.statusAccess != undefined) {
                    $scope.statusChart(data.statusAccess);
                }
                if (data.rows != undefined) {

                    $scope.statusListsCreunaFuction($scope.rows);
                    $scope.statusListsCustomerFuction($scope.rows);
                    $scope.getAllStatusNow();
                    $scope.statusListFunc(0 );
                }


            })
            .error(function(data, status, headers, config) {
                $scope.error = {
                    data: data,
                    status: status,
                    headers: headers,
                    config: config
                };
            });


    };
    $scope.statusListFunc = function (item) {
        var statusData = $scope.rows;
        var element = [];
        var list = [];
        for (var i = 0; i < statusData.length; i++) {
           for (var k = 0; k < statusData[i].statusList.length; k++) {
```

```
                if (item == 0) {
                      element = statusData[i].statusList[k];
                      list.push(element);

                 }
                 else {
                     var t = statusData[i].statusList[k].step;
                     if (t == item) {
                         element = statusData[i].statusList[k];
                         list.push(element);
                     }
                 }
             }
         }
         $scope.statusListsAll = list;
};




$scope.getAllStatusNow = function() {

    $scope.statusLists = $scope.statusListsAll;

    angular.element("#getAllBtn").focus();


};
$scope.getFilters = function() {
    // Detect query scope
    if (!$.isEmptyObject($location.search())) {
        $scope.output = $location.search();
    }
    // Detect local storage scope
    else if (!$.isEmptyObject(localStorageService.get('output'))) {
        $scope.output = localStorageService.get('output');
    }

    $scope.loadFilters();
};

// Watch filters on action
$scope.$on('$locationChangeSuccess', function(object) {
    $scope.getFilters();

});

$(function() {


        $scope.chart = function(item, cat) {
            setTimeout(function () {
            var dl = [];
            for (var x = 0; x < item.length; x++) {
                var t = item[x].title.substring(0,20);

                dl.push(t);
            };

            var dataTest = [];
```

```
                    var status = [];
                    for (var j = 0; j < cat.length; j++) {
                        var st = cat[j].name;
                        status.push(st);
                    }
                    for (var i = 0; i < item.length; i++) {


                        var d = [];
                        var n = [];
                        var na = "date";
                        n.push(na);

                        for (var k = 0; k < item[i].statusList.length; k++) {

                            var step = i;
                            d.push(step);

                        }
                        var nd =
                        {
                            name: n,
                            data: d
                        };
                        dataTest.push(nd);

                    };
                    dataTest.push({ name: "1", data: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0], color: "transparent" });
                    $('#container').highcharts({
                        title: {
                            text: 'Case status progress',
                            x: -20 //center
                        },
                        xAxis: {
                            categories: status,

                        },
                        yAxis: {
                            title: {
                                text: 'Cases'
                            },
                            categories: dl,
                        },
                        tooltip: {
                            valueSuffix: ""
                        },
                        legend: {
                            enabled: false
                        },
                        series: dataTest,
                    });
                    }, 200);
                };
        });
    }
]);
```

9.1.5 Viewet DashboardAdmin.cshtml

```
<div id="wrapper" ng-controller="dashboardController">
    <!-- Sidebar -->
    @*    *@
    <div id="sidebar-wrapper">
        <ul class="sidebar-nav">

            <li class="sidebar-brand" style="color:white">
                My Projects
            </li>
            <li ng-repeat=" customer in customers" ng-click="Refresh(customer.id)">
                {{customer.name}} test
            </li>

        </ul>
    </div>
    <div style="background:#5e5d5d;  padding-top:10px; padding-
bottom:1px; color:#999999">
        <h3 class="text-center" ng-repeat="c in customerName ">{{c.name}}</h3>
    </div>
    <!-- Page Content -->
    <div id="page-content-wrapper">


        <div class="container-fluid" ng-hide="rows.length > 0">
            <div><h1>No Data...</h1></div>
        </div>
        <div class="container-fluid" ng-show="rows.length > 0">
            <div class="row">
                <div class="col-lg-12">

                    <div class="container-fluid">

                        <div class="row">

                            <div class="col-xs-12">
                                <div class="row ">
                                    <button class="btn btn-primary pull-right" ng-
click="SetDate()">Refresh Period</button>

                                    <div id="to" class="input-
append date datepicker pull-right" data-date-format="dd-mm-yyyy">

                                        <input class="input-daterange form-control" ng-
model="dateTo" size="16" type="text"><span class="add-on"><i class="glyphicon glyphicon-
calendar"></i></span>
                                    </div>
                                    <div id="from" class="input-
append date datepicker pull-right" data-date-format="dd-mm-yyyy">
                                        <input class="input-daterange form-control" ng-
model="dateFrom" ng-click="dateSet()" size="16" type="text"><span class="add-
on"><i class="glyphicon glyphicon-calendar"></i></span>
                                    </div>
                                    <div ng-if="{{dateFrom}}">
                                        From {{dateFrom}}| To | {{dateTo}}

                                    </div>
                                </div>
```

```
                                <div id="container" style="min-
width: 310px; height: 400px; margin: 0 auto"></div>
                                <div class="row">

                                </div>
                            </div>
                        </div>
                    </div>
                    <div class="row">
                        <div class="col-md-1">
                            <br />
                            <div class="panel panel-default clearfix">
                                <div class="panel-heading text-center" style="padding-
left: 0; padding-right: 0">
                                    <h5>Remaining</h5>
                                </div>
                                <div class="panel-body label-default">

                                    <div class="text-center">
                                        <span class="glyphicon glyphicon-
tasks"><h3>{{rows.length}}</h3> </span>

                                    </div>
                                </div>
                            </div>
                        </div>
                        <div class="col-md-2">
                            <br />
                            <div class="panel panel-default clearfix">
                                <div class="panel-heading text-center">
                                    <h5>Overall</h5>
                                </div>
                                <div class="panel-body label-default">

                                    <div class="text-center">
                                        <span class="glyphicon glyphicon-
tasks "><h3>{{overallStatus | number:0}}%</h3> </span>

                                    </div>
                                </div>
                            </div>
                        </div>
                        <div class="col-md-1">
                            <br />
                            <div class="panel panel-default clearfix">
                                <div class="panel-heading text-center">
                                    <h5>Total</h5>
                                </div>
                                <div class="panel-body label-default">

                                    <div class="text-center">
                                        <span class="glyphicon glyphicon-
tasks"><h3>{{rows.length}}</h3> </span>
                                    </div>
                                </div>
                            </div>
                        </div>
                        <div class="col-md-4">
                            <br />
                            <div class="panel panel-default clearfix">
```

```
                            <div class="panel-heading text-center">
                                <h5>Hours</h5>
                            </div>
                            <div class="panel-body label-default">
                                <div class="text-center">
                                    <div class="progress">
                                        <div class="progress-bar progress-bar-
success progress-bar-striped active" role="progressbar" aria-valuenow="45" aria-
valuemin="0" aria-valuemax="100" style="width: 50%">
                                            <span>250 Hours Used</span>
                                        </div>

                                    </div>
                                    <div class="progress">
                                        <div class="progress-bar progress-
bar " role="progressbar" aria-valuenow="45" aria-valuemin="0" aria-
valuemax="100" style="width: 100%">
                                            <span>500 Hours Estimated</span>
                                        </div>

                                    </div>
                                </div>

                            </div>
                        </div>
                    </div>



                    <div class="col-md-2">
                        <br />
                        <div class="panel panel-default clearfix">
                            <div class="panel-heading text-center">
                                <h5>Hours This Week</h5>
                            </div>
                            <div class="panel-body label-default">
                                <div class="text-center">
                                    <span class="glyphicon glyphicon-step-
backward"><h3>28</h3> </span>
                                </div>

                            </div>
                        </div>
                    </div>
                    <div class="col-md-2">
                        <br />
                        <div class="panel panel-default clearfix">
                            <div class="panel-heading text-center">
                                <h5>Hours This Month</h5>
                            </div>
                            <div class="panel-body label-default">
                                <div class="text-center">
                                    <span class="glyphicon glyphicon-fast-
backward"><h3>140</h3> </span>
                                </div>

                            </div>
                        </div>
                    </div>
```

```
                    </div>
                    <br />
                    <div class="row">
                        <div class="col-xs-6">
                            <div class="panel panel-default">
                                <div class="panel-heading">

                                    <h3 class="panel-
title"><span class="badge">{{statusListsAll.length}}</span> Status Creuna</h3>
                                    <div class="pull-right">
                                        <div class="btn-group" role="group">
                                            <button type="button" class="btn btn-
default" ng-click="statusListFunc(1)">Creuna</button>
                                            <button type="button" class="btn btn-
default" ng-click="statusListFunc(2)">Customer</button>
                                            <button id="getAllBtn" type="button" class="b
tn btn-default" ng-class="" ng-click="statusListFunc(0)">All</button>
                                            <a data-ng-
href="@Url.Action("ExportData", "Dashboard", new { listType = "case" })" target="_self" c
lass="add-case btn btn-blue-color">+ Save to excel</a>
                                        </div>

                                        <br />
                                        <br />
                                    </div>
                                    <br />
                                    <br />
                                    <div class="text-center">

                                    </div>
                                </div>
                                <div class="panel-body">
                                    <div style="overflow: auto; height: 250px;">
                                        <table class="table  table-responsive table-
hover">
                                            <thead>
                                                <tr>
                                                    <th>
                                                        Dato:
                                                    </th>
                                                    <th>
                                                        Id
                                                    </th>
                                                    <th>
                                                        Title
                                                    </th>
                                                    <th>
                                                        Status
                                                    </th>
                                                    <th>
                                                        Ændret af:
                                                    </th>

                                                </tr>
                                            </thead>

                                            <tbody ng-
repeat="statusListAll in statusListsAll">
```

```
                                                <tr ng-
click="showCustomerCase(statusListAll.caseId)" style="cursor: pointer">
                                                    <td>
                                                        {{statusListAll.date}}
                                                    </td>
                                                    <td>{{statusListAll.caseId}}</td>
                                                    <td ng-
width="50px" style="width: 50px; overflow: hidden;">
                                                        {{statusListAll.title }}
                                                    </td>
                                                    <td>
                                                        {{statusListAll.name}}
                                                    </td>
                                                    <td>
                                                        {{statusListAll.user}}
                                                    </td>

                                                </tr>
                                            </tbody>
                                            <tfoot>

                                            </tfoot>
                                        </table>
                                    </div>
                                </div>
                            </div>
                        </div>


                        <div class="col-xs-6" ng-repeat="backlog in backlogs">
                            <div class="panel panel-default">
                                <div class="panel-heading">
                                    <h3 class="panel-
title"><span class="badge">{{backlog.items.length}}</span> Backlog</h3>
                                    <div class="pull-right">
                                        <div class="btn-group" role="group">
                                            <a data-ng-
href="@Url.Action("ExportData", "Dashboard", new { listType = "backlog" })" target="_self
" class="add-case btn btn-blue-color">+ Save to excel</a>

                                        </div>

                                        <br />
                                        <br />
                                    </div>
                                    <br />
                                    <br />
                                </div>
                                <div class="panel-body">
                                    <div style="overflow: auto; height: 250px;">
                                        <table class="table  table-responsive table-
hover">

                                            <thead>
                                                <tr>
                                                    <th>
                                                        Title
                                                    </th>
                                                    <th>
                                                        State
                                                    </th>
```

```
                                                <th>
                                                    Priority
                                                </th>
                                                <th>
                                                    Changed By
                                                </th>

                                            </tr>
                                        </thead>

                                        <tbody ng-repeat="item in backlog.items">

                                            <tr ng-
click="showBacklog(item.id)" style="cursor: pointer">
                                                <td>{{item.title}}</td>
                                                <td>{{item.state}}</td>
                                                <td>
                                                    {{item.priority}}
                                                </td>
                                                <td>{{item.person}}</td>
                                            </tr>

                                        </tbody>
                                        <tfoot>

                                        </tfoot>
                                    </table>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>

            </div>
        </div>
    </div>
</div>
<!-- /#page-content-wrapper -->
```