

Kandidatarbete

Redaktör: Pål Kastman

Datum: 2015-04-20

Version 0.2

Dokumenthistorik

Datum	Version	Utförda ändringar	Utförda av
2015-03-13	0.1	Första utkast av gemensam och individuella rapporter	Pål Kastman
2015-04-20	0.2	Andra utkastet av gemensam och individuella rapporter	Alla

Projektidentitet

Detta dokument gäller för grupp 7 i kursen TDDD77 på Linköpings universitet

Namn	Ansvarsområde	E-post
Daniel Rapp	Teamledare	danth407@student.liu.se
Daniel Falk	Analysansvarig	danfa519@student.liu.se
Jonas Andersson	Arkitekt	jonan111@student.liu.se
Albert Karlsson	Kvalitetssamordnare	albka735@student.liu.se
Erik Malmberg	Testledare	erima694@student.liu.se
Pål Kastman	Dokumentansvarig	palka285@student.liu.se

Kund

Region Östergötland

Kundkontakt

Daniel Hall, daniel.hall@regionostergotland.se

Erik Sundvall, erik.sundvall@regionostergotland.se

Ingrid Hallander, ingrid.hallander@regionostergotland.se

Handledare

Lena Buffoni, lena.buffoni@liu.se

Handledare

Kristian Sandahl, kristian.sandahl@liu.se

Innehåll

Del I	Gemensamma erfarenheter och diskussion	1
1	Inledning	1
1.1	Motivering	1
1.2	Syfte	1
1.3	Frågeställning	1
1.4	Avgränsningar	1
2	Bakgrund	2
2.1	Programspråk och bibliotek	2
2.1.1	Node.js	2
2.1.2	Keystone.js	2
2.1.3	Socket.IO	2
2.1.4	MongoDB	3
2.1.5	Handlebars	3
2.1.6	REST	3
2.1.7	Wkhtmltopdf	3
3	Teori	3
4	Metod	3
4.1	Kravinsamling	3
4.1.1	Förstudie	4
4.1.2	Kravspecifikation i Google docs	4
4.1.3	Kravrepresentation	4
4.1.4	Kravvalidering	5
4.2	Utvecklingsmetod	5
4.3	Forskningsmetod	5
5	Resultat	6
5.1	Översikt av systemet	6
5.1.1	Server och klient	6
5.1.2	Kartoteket	6
5.2	Tekniker	7
5.2.1	Översikt	7
5.2.2	Back-end	7
5.2.3	Front-end	8
6	Diskussion	9
6.1	Resultat	9
6.2	Metod	9
6.3	Arbetet i ett vidare sammanhang	9
7	Slutsatser	9
8	Fortsatt arbete	9
8.1	Restlista	9

Del II	Enskilda utredningar	10
A	Kartoteket - Daniel Rapp	10
A.1	Inledning	10
A.1.1	Syfte	10
A.1.2	Frågeställning	10
A.1.3	Avgränsningar	10
A.2	Bakgrund	11
A.3	Metod	11
A.4	Resultat	12
A.4.1	Att se artiklarna	12
A.4.2	Modifiering av artiklarna	13
A.4.3	Sökning av artiklarna	13
A.5	Diskussion	13
A.5.1	Resultat	13
A.5.2	Metod	13
A.6	Slutsatser	13
A.7	Referenser	13
B	Jonas Andersson	14
B.1	Inledning	14
B.1.1	Syfte	14
B.1.2	Frågeställning	14
B.1.3	Avgränsningar	14
B.2	Bakgrund	14
B.3	Teori	14
B.4	Metod	15
B.5	Resultat	15
B.6	Diskussion	15
B.6.1	Resultat	15
B.6.2	Metod	15
B.7	Slutsatser	15
B.8	Referenser	15
C	Pål Kastman	16
C.1	Inledning	16
C.1.1	Syfte	16
C.1.2	Frågeställning	16
C.1.3	Avgränsningar	16
C.2	Bakgrund	16
C.3	Teori	17
C.4	Metod	17
C.5	Resultat	17
C.6	Diskussion	18
C.6.1	Resultat	18
C.6.2	Metod	18
C.7	Slutsatser	18
C.8	Referenser	18
D	Daniel Falk	19

D.1	Inledning	19
D.1.1	Syfte	19
D.1.2	Frågeställning	19
D.1.3	Avgränsningar	19
D.2	Bakgrund	19
D.3	Teori	19
D.3.1	Prototyper	19
D.3.2	Intervjuer	20
D.3.3	Observationer	20
D.4	Metod	20
D.4.1	Intervjuer	20
D.4.2	Observationer	20
D.4.3	LoFi-prototyper	20
D.4.4	HiFi-prototyp	21
D.4.5	Användartest	21
D.5	Resultat	21
D.6	Diskussion	22
D.6.1	Resultat	22
D.6.2	Metod	22
D.7	Slutsatser	23
D.8	Referenser	23
E	Automatiserade tester med Travis CI - Erik Malmberg	24
E.1	Inledning	24
E.1.1	Syfte	24
E.1.2	Frågeställning	24
E.1.3	Avgränsningar	24
E.2	Bakgrund	24
E.2.1	Travis CI	25
E.2.2	Javascript	25
E.2.3	JQuery	25
E.2.4	Node.js	25
E.2.5	Jasmine	25
E.3	Teori	25
E.3.1	Vattenfallsmodellen	25
E.3.2	Kontinuerlig integration och automatiserade tester	26
E.4	Metod	26
E.5	Resultat	28
E.6	Diskussion	30
E.6.1	Resultat	30
E.6.2	Metod	30
E.7	Slutsatser	30
E.7.1	Hur kan man använda webb-baserade tjänster för att utföra kontinuerliga automatiserade tester av webb-applikationer?	30
E.7.2	Hur effektivt är det att använda en webb-baserad tjänst för automatiserade tester?	30
E.7.3	Vilka typer av tester är svåra att utföra med en sådan tjänst?	30
E.7.4	Framtida arbete inom området	30
E.8	Referenser	30

F	Checkning av checklistor - Robin Andersson	31
F.1	Inledning	31
F.1.1	Syfte	31
F.1.2	Frågeställning	31
F.1.3	Avgränsningar	31
F.2	Teori	32
F.2.1	Websockets	32
F.3	Metod	33
F.4	Resultat	33
F.5	Diskussion	34
F.5.1	Resultat	34
F.5.2	Metod	34
F.6	Slutsatser	35
F.7	Referenser	35
G	Vidareutveckling av applikation för Region Östergötland - Albert Karlsson	36
G.1	Inledning	36
G.1.1	Syfte	36
G.1.2	Frågeställning	36
G.1.3	Avgränsningar	36
G.2	Bakgrund	36
G.3	Teori	36
G.3.1	MSSQL	37
G.3.2	npm	37
G.3.3	Express	37
G.3.4	Edge.js	37
G.4	Metod	37
G.5	Resultat	37
G.6	Diskussion	38
G.6.1	Resultat	38
G.6.2	Metod	38
G.7	Slutsatser	38
G.8	Referenser	38

Del I

Gemensamma erfarenheter och diskussion

1 Inledning

Detta avsnitt behandlar varför detta projekt utförs.

1.1 Motivering

Region Östergötland har idag ett system med handböcker som en sjuksköterska går igenom inför varje operation. I dessa handböcker finns bland annat förberedelseuppgifter och plocklistor. Handböckerna är idag inte interaktiva på något sätt, istället skrivs plocklistan och förberedelseuppgifterna ut och bockas av för hand. Plattformen med handböcker kan heller inte återanvändas på olika avdelningar på grund utav licensproblem. Utöver detta system så finns ett annat separat system, som heter kartoteket, för uppgifter om vilka artiklar som finns och var i lagret de ligger. Detta gör att personalen som ska förbereda inför operationer behöver gå in i två olika system om de inte vet var alla artiklar ligger.

1.2 Syfte

Uppgiften som gruppen har fått är att skapa ett nytt system med handböcker som har interaktiva förberedelse-och plocklistor. Listorna ska uppdateras kontinuerligt när de bockas av så flera personer kan jobba på dem samtidigt. Plocklistan ska också innehålla uppgifter om var artiklarna ligger. Tanken är att personalen ska använda en iPad för listorna så de kan gå runt och plocka i lagret och bocka av samtidigt.

I mån av tid ska också extra funktionalitet implementeras. Till exempel sortera plocklistan med avseende på närmsta väg mellan artiklarna, lagersaldo och media i handböckerna.

Hela systemet ska ligga under en open-source licens så det kan användas fritt av alla.

1.3 Frågeställning

Rapporten ska besvara följande frågeställningar.

- Hur kan ett system för operationsförberedelser realiserars så arbetet blir lättare och mer effektivt?
- Kan man använda plattformen keystone för att bygga detta system?

1.4 Avgränsningar

Den här rapporten beskriver hur ett system för operationsförberedelser på universitetssjukhuset i Linköping kan realiserars. Andra sjukhus kan ha andra rutiner vilket får konsekvensen att systemet inte fungerar där.

Kunden ville också att ett lagersaldo i systemet skulle integreras och göra det möjligt att skanna av artiklar då dessa plockas. Vidare ville kunden även att systemet skulle ge möjlighet att välja mellan olika kliniker i regionen. Tiden kändes inte tillräckligt för att åstadkomma dessa implementeringar därför klargjordes det i ett tidigt skede att få bra grundläggande funktionalitet hade högre prioritet än dessa funktioner. En kompromiss gjordes där dessa krav fick prioritet 2 vilket innebar att kraven var önskvärda och skulle implementeras i mån av tid, eller prioritet 3 där kraven sågs som en framtida utbyggnad.

Hur man ska jobba i projektgrupper med olika kunskapsnivåer tas fram genom undersökningar i projektgrupp 7 i kursen TDDC77 på Linköpings Universitet. Resultaten av denna undersökning kan kanske därför inte appliceras i situationer som inte kan likställas med situationen i projektgruppen.

2 Bakgrund

Studenterna som studerar kursen TDDD77 fick i januari 2015 ett uppdrag att utföra ett kandidatarbete. Först fick gruppen rangordna flera projektdirektiv för att sedan få ett av dessa uppdrag tilldelat sig. Grupp 7 fick då projektet operationsförberedelser som skickades in av Region Östergötland.

2.1 Programspråk och bibliotek

I detta projekt kommer följande programspråk och bibliotek att användas.

2.1.1 Node.js

Node.js är en plattform för att skapa applikationer till framförallt webbservrar. Det finns en inbyggd pakethanterare vid namn npm som gör det enkelt att inkludera både små och stora bibliotek i sina projekt. Det är därför väldigt enkelt att använda sig av ett bibliotek istället för att skriva all funktionalitet själv.

2.1.2 Keystone.js

Keystone.js är ett så kallat "content management system" som bland annat innehåller ett kraftfullt administrationsverktyg och en databashanterare.

2.1.3 Socket.IO

Socket.IO är en modul till Node.JS. Socket.IO använder sig av websockets för att kommunicera mellan front-end och back-end. Med hjälp av socket.IO så kan man skicka data från en klient till alla andra anslutna klienter och visa datan som skickades utan att någon sida behöver laddas om. Socket.IO har olika komponenter för front-end och back-end. Händelser som skickas från den ena sidan hanteras av motsvarande händelsehanterare på den andra sidan. Varje händelse identifieras med hjälp av en sträng. Det finns några färdiga händelser i socket.io exempelvis händelsen "connection" på serversidan fås då en klient har anslutit till socket.io.

2.1.4 MongoDB

MongoDB är en NoSQL dokumentbaserad databas som stödjer många olika plattformar.

2.1.5 Handlebars

Handlebars är ett så kallat "template language" som är byggt från Mustache. Handlebars ger bland annat möjlighet till att lägga in mindre logik i html kod samt att från html filer komma åt variabler som finns definierade i javascript filer.

2.1.6 REST

2.1.7 Wkhtmltopdf

Wkhtmltopdf är ett program som tar en hemsida eller en html-fil och skapar en pdf av den. Programmet har många olika parametrar som kan ändras t.ex. medietyp så pdf-filen kan se ut som en utskrift av hemsidan. För att kunna använda detta programmet med Node.js finns det också en Node.js-modul med samma namn som skickar kommandon till programmet.

3 Teori

För att kunna svara på frågeställningar så måste några begrepp i dem definieras. Effektivt definieras i detta fallet som att det tar kortare tid för operationsförberedelser och kräver mindre av personalen, till exempel att personalen inte ska behöva hålla lika mycket information i huvudet. Lättare definieras som att det går snabbare för nyanställda och personer som tillfälligt är på en avdelning de inte brukar jobba på, att lära sig rutinerna och kunna utföra arbetet.

Effektiviseringen av sjukvården och hur den ska ske är frågor som diskuterats flitigt de senaste åren. Region Östergötland har gjort tester för att kontrollera hur en operationsförberedelse kan effektiviseras. Ett av dessa tester gick ut på att undersöka hur lång tid det tar att plocka lagerartiklar då lagerplats finns lättillgängligt gentemot hur lång tid det tar med nuvarande system. Resultatet var att.. [referens Elisabeth].

4 Metod

I detta avsnitt beskrivs det hur arbetet med projektet har gått till.

4.1 Kravinsamling

Denna del beskriver hur kravframställningen gått till i detta projekt. Först beskrivs arbetet med att analysera kundens behov under förstudien. Vidare beskrivs hur kravspecifikationen arbetades fram och hur kraven valdes att presenteras. Avslutningsvis beskrivs hur kraven validerades. I en enskild utredning(referera) så beskrivs mer ingående vilka metoder och verktyg som använts för att ta fram krav. Fokus ligger där på prototyper och användartester.

4.1.1 Förstudie

Den största delen utav analysarbetet skedde under förstudien. Här identifierades de olika intressenterna och en kravspecifikation utarbetades. Vår första kontakt med projektet var en projektbeskrivning där kunden formulerade sina mål och visioner av projektet. Några viktiga krav gavs också såsom att prototyp-design skulle genomföras i samarbete med kunden. Ett första möte utav fyra under förstudien gav sedan mer information och vi började vår kravinsamling. Vi kunde konstatera att vi hade två olika intressenter att arbeta med. Dels sjuksköterskorna som är användare av systemet och dels CMIT, Centrum för medicinsk teknik och IT, som ansvarar för sjukhusets IT-miljöer. För att förstå användarnas behov hölls ett studiebesök där vi fick en visning av nuvarande system och hur det används. Detta var nödvändigt för att verkligen förstå vad det var som behövde göras och vad som kunde förbättras. Från CMIT:s sida hölls mer tekniska möten där teknikval diskuterades. Här var det viktigt att ta reda på vilka begränsningar som fanns och vilka val som passade våra och deras erfarenheter.

4.1.2 Kravspecifikation i Google docs

Kravspecifikationen skrevs i Google docs. Detta valdes eftersom kraven utarbetades tillsammans med kund. Ett gemensamt redigerbart dokument gav en möjlighet för oss att arbeta på olika platser under kravframställningen vilket var effektivt. En kommentarsfunktion gav oss möjligheten att kommentera krav och föreslå förbättringar. Under interna möten och kundmöten var den gemensamma redigeringen också till nytta då kravformuleringar snabbt kunde genomföras.

4.1.3 Kravrepresentation

Kraven gavs en prioritetsordning för att kunna urskilja de mest väsentliga kraven. Detta kändes nödvändigt då vi var begränsade av en tidsbudget. En prioritering av kraven gav utrymme för vidareutveckling i mån av tid. Krav med prioritet 1 var att betrakta som grundkrav som skulle genomföras för att projektet skulle ses som godkänt. Krav med prioritet 2 var att betrakta som önskvärda och som skulle genomföras om då grundkraven var genomförda. Krav med prioritet 3 var krav som fångats upp men som skulle ses som framtida utbyggnad.

Kraven numrerades för att lätt kunna refereras till under projektets gång. De delades också in i olika sektioner efter deras del i systemet. Sektionerna var plocklistor, handböcker, kartotek, lagersystem. Två extra sektioner användes också för generella krav och för leveranser. Denna uppdelning kändes naturlig för detta projekt.

Kravspecifikationen skrevs med stöd från standarden IEEE 830. Enligt standarden ska ett krav vara korrekt, otvetydigt, färdigt, konsekvent, prioriterat, verifierbart, modifierbart och spårbart. Detta eftersträvades men det kan diskuteras om alla krav passerar dessa filter. I slutändan var det ändå vår gemensamma förståelse för kravet tillsammans med kunden som accepterades.

Enligt standarden uttrycktes kraven på ska-form. Ett exempel på ett krav från projektet är följande: *"Plocklistor ska innehålla information om artikelns namn, förråd, sektion, hylla och fack"*.

4.1.4 Kravvalidering

Vid varje iterationsslut hölls ett möte med kund där vi demonstrerade nya features och lät kunden testa systemet. Iterationerna gjorde att vi snabbt kunde rätta till eventuella missförstånd. Vi gick igenom vilka krav som var genomförda och vilka som skulle prioriteras till nästa iteration. Kravspecifikationen var på så sett dynamisk och uppdaterades under projektets gång. En färgkodning användes för att markera vilka krav som var godkända, vilka som hade påbörjats och vilka som återstod.

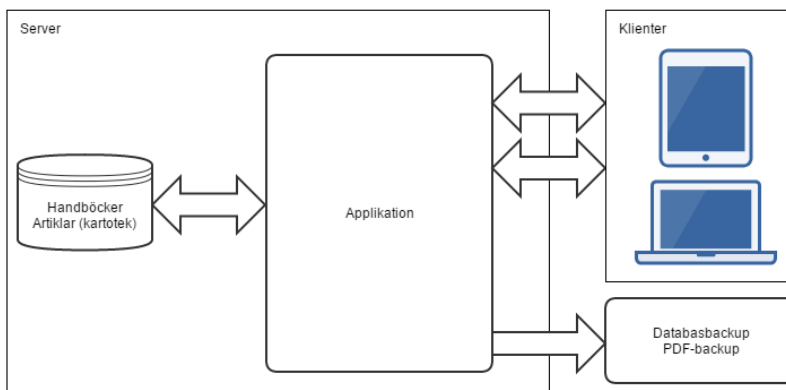
4.2 Utvecklingsmetod

Projektet har utvecklats iterativt med en utvecklingsmetodik som påminner mycket om SCRUM. Aktiviteterna i projektet har visats på en gemensam SCRUM-board med hjälp av webbsidan Trello. Boarden hade kategorierna TODO, DOING och DONE. När en medlem valt en aktivitet så märktes aktiviteten med medlemmens namn och flyttades till den aktuella kategorin. Varje vecka har gruppen haft ett kort SCRUM-möte på 15 minuter. På mötet har varje gruppmedlem berättat vad som har gjorts den föregående veckan och vad som ska göras nästa vecka. SCRUM-mötet har oftast utförts i samband med varje veckas handledarmöte.

Ett system med alpha state cards har använts för att gruppen ska kunna veta hur långt projektet har fortskridit. Korten har även använts till att identifiera aspekter av projektet som kan förbättras och som gruppen behöver arbeta mer med. De aspekter av projektet som kunnat förbättras har använts som mål för kommande iterationer. De olika korten har uppdaterats kontinuerligt under projektets gång.

Nästan all utveckling har skett med hela gruppen samlad i samma lokal. Det har gjort att alla frågor om andra gruppmedlemmars kod har kunnat besvaras snabbt. Det har gjort att arbetet kunnat hålla hög fart och att inga onödiga förseningar har uppstått på grund av osäkerheter i hur en del av koden fungerar. Problem har också kunnat lösas fort eftersom alla gruppmedlemmars olika kompetens har funnits tillgänglig.

4.3 Forskningsmetod



Figur 1: Översikt av systemet

5 Resultat

Denna del av dokumentet presenterar resultatet av vårt arbete. Här kommer vi diskutera både hur systemet ser ut och används, samt hur det är uppbyggt rent tekniskt.

5.1 Översikt av systemet

Det finns två huvuddelar i systemet. Handböckerna och kartoteket.

Handböckerna beskriver hur man förbereder olika typer av operationer. Varje handbok har en lista med artiklar som behövs till operationen, en så kallad plocklista. Artiklarna är kopplade till kartoteket, som bland annat innehåller information om var i förråden artiklarna finns placerade.

När en patient registreras skapas en instans av en handbok. I denna instans kan man checka av en lista med artiklar som ska användas under operationen och även andra förberedelser. En samordnare kan se en översikt på hur långt man har kommit med de förberedelserna för varje instans. Se en översikt i figur 1.

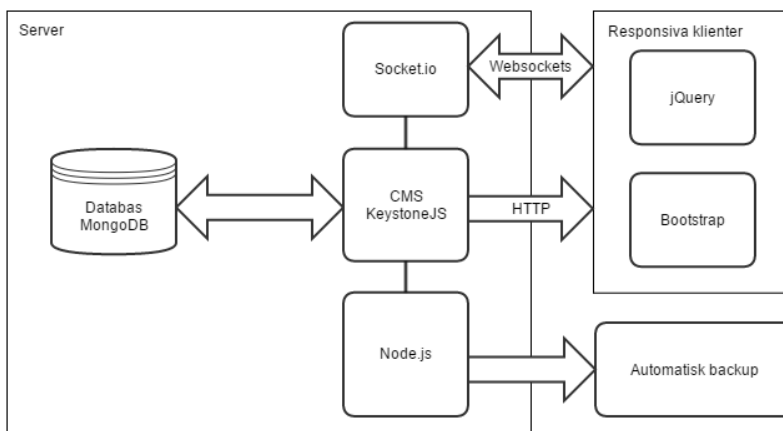
5.1.1 Server och klient

Applikationen består av en server som distribuerar en hemsida till flera klienter. Servern kan köras i en Windowsmiljö. Hemsidan är responsiv och fungerar på surfplattor och datorer i olika format.

5.1.2 Kartoteket

I kartoteket finns information om alla artiklar som Region Östergötland har i förråden. Här finns bland annat information om var artiklarna är placerade samt information relaterade till inköp av artiklar.

Läs mer om kartoteket i "Kartoteketnadan.



Figur 2: En översikt över tekniken

5.2 Tekniker

Vi börjar med att beskriva tekniken bakom systemet.

5.2.1 Översikt

Programmet är uppdelat i två delar, en serverdel och en klientdel. Serverdelen består av databaskopplingar som kopplas ihop och distribueras ut genom hemsidor till klienterna.

I figur 2 kan man se en översikt över de mest betydelsefulla tekniker och bibliotek som används för att bygga upp programmet.

5.2.2 Back-end

Koden till servern har skrivits helt i javascript. Dels för att underlätta inlärningskurvan genom att använda samma språk som på klienten och dels för att realtidskommunikation mellan klienterna underlättas. Grunden till programmet är node.js vilket är en plattform för att utveckla självständiga program i javascript med inbyggd pakethanterare. Pakethanteraren gör det enkelt att installera externa bibliotek.

Det största och mest betydelsefulla ramverket för detta projekt är KeystoneJS, ett CMS-ramverk till node.js. Följande är några punkter KeystoneJS underlättar:

- Hjälper till att abstrahera systemet. Se avsnittet om MVC nedan.
- Skapar automatiskt en administreringssida för varje databasmodell. Huvuddelen av administreringen har dock bytts ut då den automatgenererade kan vara något begränsad.
- Har ett inbyggt användarsystem som är lätt att modifiera och byta ut.
- Sköter all kommunikation över http-protokollet till klienterna, dvs. gör hemsidan åtkomlig.

- Har inbyggt stöd för templatespråk som exempelvis Handlebars¹ och Less².

För realtidskommunikation används ett programmeringsinterface vid namn socket.io³ användas. Det är väldigt smidigt eftersom det väljer automatiskt hur datan ska skickas beroende på vilken webbläsare som används och vad den stödjer. Socket.io är event-baserat vilket betyder att man skapar events på antingen klient eller serversida som man sedan kan trigga från motsatt sida. Vanliga javascript-object kan skickas tillsammans med eventen.

5.2.3 Front-end

¹TODO: bättre referens. <http://handlebarsjs.com/> (2015-04-18)

²TODO: Bättre referens. <http://lesscss.org/> (2015-04-18)

³TODO: bättre referens. <http://socket.io/> (2015-04-18)

6 Diskussion

6.1 Resultat

6.2 Metod

6.3 Arbetet i ett vidare sammanhang

7 Slutsatser

8 Fortsatt arbete

Här följer en sammanfattning om fortsatt arbete med systemet. Restlistan behandlar krav som inte implementerats eller som borde vidareutvecklas. Även förslag på vidareutveckling som inte tas upp av kravspecifikationen tas upp. Dessa förslag är sådant som kommit fram under utvecklingens gång.

8.1 Restlista

Följande krav är markerade som gula i kravspecifikationen vilket betyder att de bör vidareutvecklas för att ses som klara: Följande krav är markerade som röda i kravspecifikationen vilket betyder att de inte är påbörjade: Alla krav som behandlar lagersystemet finns kvar:

Krav nr 46 Ska finnas ett saldo för varje artikel i förrådet.

Krav nr 47 Ska gå att skanna av artiklar (streckkod) som plockas i en specifik plocklista.

Krav nr 48 Ska gå att lagra in en artikel till förrådet som inte blivit använd genom att skanna in den.

Krav nr 49 Saldot ska kunna användas som underlag till beställning.

Krav nr 50 Beställningsunderlaget ska ha ett sådant format att det går att importera i Agresso.

Krav nr 51 En inventeringsfunktion ska finnas.

Övriga förslag på vidareutveckling:

Artikelsök Vid sökning på artikel vid redigering av handbok kan det vara bra om man sorterar resultaten så att de artiklar som tillhör samma enhet som handboken visas först.

Avancerad sökfunktion En mer avancerad sökfunktion med * och sökning på flera ord som inte behöver vara direkt efter varandra.

Förvalda kommentarer Under användartester observerades att samma kommentarer skrivs ofta så det kan vara bra med en lista på vanliga kommentarer som man kan välja ifrån.

Roller och rättigheter Olika roller bör identifieras och en funktion för att tilldela användare olika rättigheter bör implementeras.

Del II

Enskilda utredningar

A Kartoteket - Daniel Rapp

A.1 Inledning

Idag är information om Region Östergötlands operationsartiklar, så som priser och placeringen i lagret på tandborstar, tandkräm, handskar och annan medicinsk utrustning, hanterat av ett internt system. Detta system kallas ett ”*kartotek*”, och är helt enkelt en sorts artikeldatabas. I vårt system så ska detta uppdateras och förbättras på olika sätt.

A.1.1 Syfte

Syftet med denna del är att beskriva vad kartoteket är samt hur vår förbättrade lösning är uppbyggd.

A.1.2 Frågeställning

Frågeställningar:

- Går det att integrera systemet för handböcker med kartoteket utan att förlora funktionalitet?

A.1.3 Avgränsningar

Artikel	Klinik	UC_ArtNr	Lev_ArtNr	Föränd	Sektion	Hylla	Fack
36 Ligacip vedres M	Operation Öron US		LT200				
Abs Silikonförband 5X12,5Cm Självhåll...	GEM op/ane mtrlf NORD	28414		NS	39	F	1
Abs silikongörband 5 x 12,5cm självhåll...	GEM op/ane mtrlf SYD	28414		SS	32	D	2
Abs Material , Strössel (T. Kemikaler)	GEM op/ane mtrlf NORD		41522	NO	B	B4	2
Abs Skydd Fixerribyxa	GEM op/ane mtrlf NORD	351916	Arconfort	NO	B	B3	11
Abs Skydd För Hemar Robot	GEM op/ane mtrlf NORD	33121		NO	B	B2	4
Abs-förband 10x10 cm	GEM op/ane mtrlf SYD	330102		SS	33	F	2
Abs-Förband 10X10Cm (S)	GEM op/ane mtrlf NORD	330102		NS	38	A	1
Abs-förband 10x20cm	GEM op/ane mtrlf SYD	33010201		SS	33	F	1
Abs-förband 20x40	GEM op/ane mtrlf SYD	33010402		SS	33	G	1
Abs-Förband 20X40Cm (S)	GEM op/ane mtrlf NORD	33010402		NS	38	A	2
Abs-Förband Häft 10X35Cm (S)	GEM op/ane mtrlf NORD	330404		NS	38	B	4
Absorber Co2	GEM op/ane mtrlf NORD	80971	Mx5004	NO	42	F+G	
Absorptionskalk, Infinity ID CLUC absor...	GEM op/ane mtrlf SYD	80971		SO	45	D	

Figur 3: Gamla kartoteket

A.2 Bakgrund

I dagsläget använder Region Östergötland sig av två separata system för att förbereda operationer. En handbok (se ovan) och ett kartotek (se 3). Dessa är för tillfället helt separata applikationer.

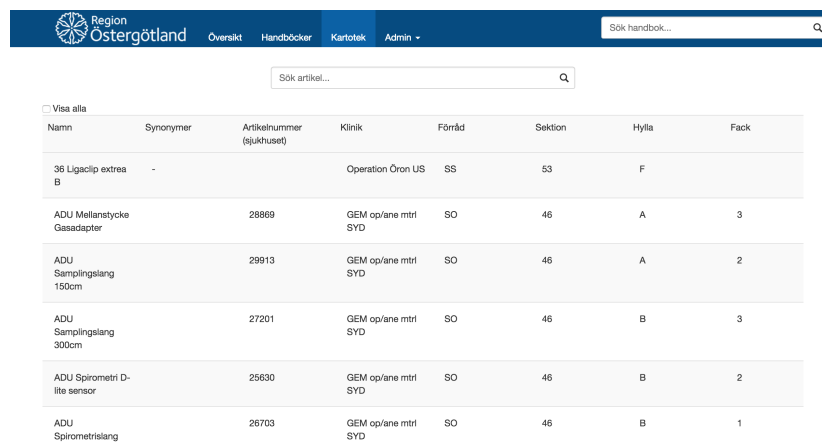
Så om en artikel slutas säljas eller Region Östergötland väljer att inte köpa in en viss artikel längre så tar de bort artikeln från kartoteket. Problemet som då uppstår är att detta inte reflekteras i handböckerna. Så om t.ex. en ”*Oral-B Pro 600 CrossAction*” tandborste används i en ”*Laparoskopisk sigmoideumresektion*”, och tandborsten slutas säljas så tas den bort från kartoteket, men eftersom handboken för operationen inte är kopplad till kartoteket så uppdateras det inte att denna artikel inte längre finns i lagret.

Vår förbättrade lösning integrerar systemet som hanterar handböcker tillsammans med ett nytt kartotek, där allesammans är byggt på webben. När en artikel ändras eller tas bort i kartoteket så ändras den även i alla handböcker för operationer som kräver denna artikel.

A.3 Metod

Precis som resten av systemet så är kartoteket skrivet på webben, och därmed i javascript, html och css.

TODO: Skriv mer.



The screenshot shows the 'Kartoteket' interface for Region Östergötland. It features a navigation bar with 'Översikt', 'Handböcker', 'Kartoteket', and 'Admin'. A search bar is present in the top right. Below the navigation bar, there is a search input field labeled 'Sök artikel...'. The main content area displays a table of medical equipment with the following columns: Namn, Synonymer, Artikelnummer (sjukhuset), Klinik, Förråd, Sektion, Hylla, and Fack. The table lists several items, including '36 Ligacip extra B', 'ADU Mellanstycke Gasadapter', 'ADU Samplingslang 150cm', 'ADU Samplingslang 300cm', 'ADU Spirometri D-lite sensor', and 'ADU Spirometrislang'.

Namn	Synonymer	Artikelnummer (sjukhuset)	Klinik	Förråd	Sektion	Hylla	Fack
36 Ligacip extra B	-		Operation Öron US	SS	53	F	
ADU Mellanstycke Gasadapter		28869	GEM op/ane mtrl SYD	SO	46	A	3
ADU Samplingslang 150cm		29913	GEM op/ane mtrl SYD	SO	46	A	2
ADU Samplingslang 300cm		27201	GEM op/ane mtrl SYD	SO	46	B	3
ADU Spirometri D-lite sensor		25630	GEM op/ane mtrl SYD	SO	46	B	2
ADU Spirometrislang		26703	GEM op/ane mtrl SYD	SO	46	B	1

Figur 4: Kartoteket

A.4 Resultat

Resultatet av vårt arbete är ett förbättrat kartotekssystem som integrerar data från handboken till ett uniformt system.

Kärnfunktionaliteten i kartoteket är möjligheten att se, modifiera och hitta artiklar. Så denna del är uppdelad i tre delar.

A.4.1 Att se artiklarna

När man först kommer in på sidan för att hantera kartoteket så blir man välkommen av en stor tabell som innehåller alla artiklar i kartoteket (runt 3000 för tillfället). Se figur 4 När man först kommer in så laddas endast runt 50 artiklar, men om man skrollar ner så laddas fler.

Det finns två olika sätt att se informationen. Två olika vyer.

Den ena är standardvyn, som man ser om man inte är inloggad eller inte är administratör. Denna vyn kan ses i 4 och innehåller endast den mest nödvändiga informationen om artiklarna, som namn, klinik, förråd, etc.

Den andra vyn är administratörsvyn. Man kommer endast in på denna vy om man är administratör. Om man aktiverar denna så utvidgas tabellen för att visa mer information om artiklarna, bland annat pris på artiklarna. Vi har valt att inte inkludera en bild på denna information då detta är sekretessbelagt.

I den här vyn finns också möjligheten att ta bort, lägga och modifiera information om artiklar, vilket är vad nästa två delarna handlar om.

A.4.2 Modifiering av artiklarna**A.4.3 Sökning av artiklarna****A.5 Diskussion**

Har vi lyckats med kartoteket? Och framför allt, är kunden nöjd? Fick kunden de dom önskade?

A.5.1 Resultat**A.5.2 Metod****A.6 Slutsatser****A.7 Referenser**

B Jonas Andersson

B.1 Inledning

Tidigare när jag har utvecklat webbprogram så har jag använt mig av PHP tillsammans med HTML, css och javascript. Jag har dessutom valt att skriva mycket själv och inte förlita mig på ramverk och bibliotek. I detta projekt valde jag, som arkitekt, istället att byta ut PHP mot node.js. Dessutom har jag lagt mycket vikt på använda bibliotek så mycket som möjligt för att slippa uppfinna hjulet på nytt.

B.1.1 Syfte

Syftet med denna del av rapporten är att analysera vad det finns för fördelar och nackdelar med att använda node.js gentemot andra vanliga språk för webben. Det ska även undersökas hur inlärningskurvan beror på språk och externa ramverk och bibliotek.

B.1.2 Frågeställning

Frågeställningar

- Vad finns det för fördelar/nackdelar med node.js gentemot andra språk för webben?
- Vad finns det för fördelar/nackdelar med att använda mycket externa bibliotek?

B.1.3 Avgränsningar

Det finns många programmeringsspråk som man kan använda i samma syfte som node.js. Eftersom jag enbart har tidigare erfarenheter av PHP och Python så kommer node.js jämföras mot dessa och inga andra.

B.2 Bakgrund

B.3 Teori

Node.js är en plattform för att skapa applikationer till framförallt webbservrar. Det finns en inbyggd pakethanterare vid namn npm som gör det enkelt att inkludera både små och stora bibliotek i sina projekt. Det är därför väldigt enkelt att använda sig av ett bibliotek istället för att skriva all funktionalitet själv.

Javascript är det programmeringsspråk som används i Node.js. På klienten, d.v.s. i webbläsaren, är man tvingad att använda sig av javascript eller något programmeringsspråk som kan kompileras till javascript. Genom att använda node.js får man därav samma språk på både server och klient.

B.4 Metod

B.5 Resultat

B.6 Diskussion

B.6.1 Resultat

B.6.2 Metod

B.7 Slutsatser

B.8 Referenser

C Pål Kastman

C.1 Inledning

I detta projekt så har vi valt att inledningsvis skriva dokumentationen i Googles ordbehandlingsprogram Google Docs för att i ett senare skede, då det var dags att påbörja denna rapport gå över till att använda typsättningssystemet LaTeX.

Vi valde att göra på detta sätt för att så snabbt som möjligt komma igång, då man bland annat i Google Docs kan se live vad andra skriver.

C.1.1 Syfte

Syftet med denna individuella del är att undersöka funktionaliteten i LaTeX och väga fördelar mot nackdelar.

C.1.2 Frågeställning

- Vad finns det för begränsningar i LaTeX
- Kommer det att vara en fördel eller en nackdel för flera gruppmedlemmar att jobba samtidigt i samma delar av rapporten.

C.1.3 Avgränsningar

Denna rapport kommer att avgränsas för att endast jämföra LaTeX, Microsoft Office och Google Docs, detta för att inte behöva jämföra alla olika ordbehandlingsprogram.

C.2 Bakgrund

Dokumentering är någonting som är viktigt att göra när man arbetar i projektform, dels för egen del utifall man behöver gå tillbaka och se vad som gjorts, men även för andras skull ifall man kanske får en ny medarbetare som ska integreras i projektet. En fråga som man alltid behöver besvara är i vilket ordbehandlingsprogram dokumentationen skall skrivas. Det populäraste alternativet kan tänkas vara Microsoft Word, vilket har funnits sedan 1983 [1].

Ett annat alternativ som blir allt vanligare är Google Docs, vilket är ett web-baserat ordbehandlingsprogram som lanserades av Google 2006 [2] efter att man hade köpt upp företaget Upstartle [3].

Ett tredje och inte lika vanligt alternativ är LaTeX, vilket inte är ett ordbehandlingsprogram utan istället ett märkspråk så som t.ex. HTML, där man istället för med ett grafiskt gränssnitt formaterar sin text, sätter sin text inom speciella taggar så att när koden senare kompileras får rätt utseende.

LaTeX är egentligen bara en uppsättning makron skrivna för språket Tex, vilket skapades av den amerikanske matematikern Donald Knuth 1978 [4]. I början av 80-talet så vidareutvecklade Leslie Lamport Tex med hjälp av dess makrospråk till det som idag är LaTeX [5].

Vad det gäller min egen bakgrund i LaTeX så hade jag när detta projektet påbörjades endast använt det i ett tidigare projekt, under det projektet blev jag dock inte så insatt i LaTeX utan fyllde endast i material i de redan färdiga mallarna vi hade.

Under kursens gång har jag dock skrivit en laborationsrapport i Latex och därigenom skaffat mig lite mer erfarenhet.

C.3 Teori

När Microsoft utvecklade Word på 80-talet så ville de naturligtvis att dåtidens datorer skulle klara av att ladda dokument utan att det skulle vara alltför prestandakrävande, därför konstruerade man dess filformat (.doc) binärt och gjorde konstruktionen väldigt komplex. Detta medförde att bara personer som var riktigt insatta i detta filformat klarade att ändra direkt i dessa filer. Google docs däremot använder sig av molnet för att spara filer och man behöver därför aldrig oroa sig över säkerhetskopiering. Textformatering i båda dessa program görs främst genom att använda det grafiska gränssnittet, men kan även göras genom tangentbordskommandon.

Latex är i motsats till Word och Google docs inget ordbehandlingsprogram, utan istället ett märkspråk liksom HTML. När man i Word och Google docs ändrar textformateringen genom ett grafiskt gränssnitt så markerar man istället sin text med taggar, som senare när man kompilerar koden ger det önskade utseendet.

Detta gör att man som användare behöver bry sig mindre om utseendet av dokumentet och kan istället fokusera på innehållet. Eftersom Latex inte är något program i sig utan mer ett programspråk så behöver man något program att redigera koden i, detta kan göras i vilken textredigerare som helst, men det finns även program som kan kompilera koden åt användaren så att man kan se direkt vilka ändringar som görs. Det har på senare år dykt upp flera webbsidor där man kan skriva Latexdokument live, tillsammans med andra användare (ungefär som Google docs). Exempel på sådana sidor kan vara Overleaf [6], eller Authorea [7]. Latex har väldigt många inbyggda kommandon och det är väldigt lätt att t.ex. skriva formler vilket har gjort Latex till standard inom den vetenskapliga sektorn [8].

C.4 Metod

Vi har till en början valt att under projektets gång arbeta i endast en fil för den gemensamma delen av kandidatprojektet. Vad det gäller de individuella delarna så har vi valt att lägga dessa i separata filer och sedan importera dessa till den gemensamma. Vi sparar alla dokumentfiler i samma repository på github som vi har källkoden.

Genom att göra på detta sätt så kan vi garantera att det inte kommer att uppstå några konflikter i de individuella delarna. Det som vi inte vet, är huruvida det kommer att uppstå problem då alla gruppmedlemmar skall skriva på den gemensamma delen.

C.5 Resultat

När vi i projektets inledning använde Google docs så märkte vi att det var väldigt svårt att hålla sig till någon sorts dokumentstandard där man t.ex. använder samma typsnitt på rubriker och text. Även om alla projektmedlemmar har försökt att hålla sig till den standard vi satt upp så blev det ändå ibland att någon del av dokumenten blev annorlunda när vi redigerade samtidigt i

dessa. Detta fungerade mycket bättre i Latex när vi skrev kandidatrapporten, latex hjälper verkligen användaren till att hela tiden hålla samma standard i dokumenten.

Vi har under projektet försökt att hålla isär på alla olika revisioner av dokument så att man senare kan gå tillbaka och se vilka ändringar som har gjorts från en iteration till en annan, detta har varit svårt att upprätthålla med Google docs då dessa dokument hela tiden sparas kontinuerligt och det inte finns något sätt att gå tillbaka i ett dokument för att se vilka ändringar som gjorts. Detta har gjort att när vi ändrat i dokument, behövt spara undan en kopia av den föregående revisionen vilket inte alltid har gjorts. I kandidatrapporten däremot har vi använt oss utav revisionshanteringsprogrammet git och på så sätt lätt kunnat revisionshantera även våra dokument och enkelt kunnat se vilka ändringar som gjorts. Detta har även hjälpt oss så att vi smidigt har kunnat korrekturläsa varandras texter.

C.6 Diskussion

C.6.1 Resultat

C.6.2 Metod

C.7 Slutsatser

C.8 Referenser

- [1] http://ia801406.us.archive.org/21/items/A_History_of_the_Personal_Computer/eBook12.pdf
Sida 11-12. Hämtad 2015-04-20.
- [2] http://googlepress.blogspot.se/2006/06/google-announces-limited-test-on-google_06.html
Hämtad 2015-04-28.
- [3] <http://googleblog.blogspot.se/2006/03/writely-so.html>
Hämtad 2015-04-28.
- [4] <https://gcc.gnu.org/ml/java/1999-q2/msg00419.html>
Hämtad 2015-04-20.
- [5] <http://research.microsoft.com/en-us/um/people/lamport/pubs/pubs.html#latex>.
Hämtad 2015-04-20.
- [6] <https://www.overleaf.com/>
Hämtad 2015-04-28.
- [7] <https://www.authorea.com/>
Hämtad 2015-04-28.
- [8] <ftp://ftp.dante.de/tex-archive/info/intro-scientific/scidoc.pdf>
Sida 2. Hämtad 2015-04-28.

D Daniel Falk

D.1 Inledning

Jag har i detta projekt haft rollen som analysansvarig vilket innebär en analys av kundens behov och framställning av krav utifrån dessa. Denna enskilda del beskriver vilka metoder som använts för att ta fram krav som tillfredsställer kundens verkliga behov.

D.1.1 Syfte

Syftet med denna del av rapporten är att undersöka olika metoder och verktyg för kravframställning. Fokus ligger på intervjuer, observationer och prototyper. Rapporten går speciellt in på hur prototyper använts för att ta fram, testa och validera krav i detta projekt.

D.1.2 Frågeställning

- Är intervjuer och observationer bra metoder för att analysera kundens behov?
- Hur kan man arbeta med prototyper för att framställa krav?
- Hur kan man använda prototyper för att testa och validera krav?

D.1.3 Avgränsningar

Rapporten har sin utgångspunkt i hur kravframställningen gått till i detta projekt. Metoden bör inte ses som ett allmänt tillvägagångssätt.

D.2 Bakgrund

Att förstå kundens verkliga behov utgör grunden för ett lyckat projekt. Ett projekt faller ofta på grund av ofullständiga krav[1]. För att samla in krav kan flera olika metoder och tekniker användas. Metoderna kan variera och vara olika bra på att fånga upp olika typer av krav.

D.3 Teori

Teorin beskriver kravinsamlingsmetoder och verktyg som använts i detta projekt.

D.3.1 Prototyper

Prototyp är ett ord som har sina rötter i grekiskan och betyder *första form*[3]. Deras syfte är att i ett tidigt skede beskriva hur det färdiga systemet ska fungera. Prototyper kan användas för att testa olika idéer och designar och kan variera i detaljrikedom. En tydlig skillnad är den mellan enkla LoFi-prototyper skissade på papper och datorbaserade HiFi-prototyper som mer liknar det riktiga systemet. LoFi-prototyper är ett bra verktyg för att snabbt kunna diskutera ett designval då det kräver väldigt lite arbete. En styrka hos LoFi-prototyper är att användare har lätt att komma med kritiska kommentarer utan att känna att

de förolämpar designern[3]. Prototyper kan vara temporära eller evolutionära[3]. En temporär prototyp är en prototyp som slängs efter att man har använt den och utvärderat den. En evolutionär prototyp slängs inte utan byggs vidare på. Man kan se det som en tidig version av det slutgiltiga systemet.

D.3.2 Intervjuer

Intervjuer är en bra teknik för att...

D.3.3 Observationer

Observationer kan användas...

D.4 Metod

Denna del beskriver hur intervjuer, observationer och prototyper har använts som metoder för kravframställning i detta projekt. Metoden beskriver också hur systemet testats för att validera och testa kraven. Frågeställningarna besvaras utifrån denna erfarenhet.

D.4.1 Intervjuer

Möten med kund har genomförts vilka kan ses som intervjuer. Dessa möten har skett dels med IT-ansvariga, verksamhetsutvecklare och sjuksköterskor. För att samla in krav under dessa möten har vi dels fört anteckningar på papper eller dator och dels spelat in på mobil. När vi varit flera personer på möten har vi gått igenom och diskuterat våra anteckningar i efterhand. Vid oklarheter har vi antecknat dessa för att förtydliga med kund. Inspelning användes främst vid första mötet och var användbart då vi fick mycket information att sätta oss in i.

D.4.2 Observationer

För att få en inblick i hur operationsförberedelserna fungerar i dagsläget genomfördes ett studiebesök på sjukhusets operationsavdelning. Vi fick här se problemet ur sjuksköterskornas synvinkel. Vi fick se hur man arbetade i artikel-lagret i dagsläget vilket var nödvändigt för att kunna genomföra en förbättring. De svårigheter som beskrivits blev tydligare och det blev lättare att föreställa sig en lösning på problemet.

D.4.3 LoFi-prototyper

Under förstudien valde vi att vid två tillfällen göra LoFi-prototyper som vi tog med till kund. Vid dessa tester kunde vi se om vi var på rätt bana när det gällde design och struktur. LoFi-prototyperna utjordes av enkla pappersskisser. Ett exempel visas i figur 5. Det första tillfället fokuserade på gränssnittsdesign och navigation i systemet. Prototyperna visades för en sjuksköterska som fick föreställa sig systemet. De olika korten gick igenom för att se om vi hade hittat en bra design och om navigationen var logisk.

Vid det andra tillfället hade vi en intern brainstorming för att ta fram en design för redigeringsvy av en handbok. Två olika förslag togs fram och presenterades för kunden. Vi observerade och antecknade försökspersonernas olika



tankar samt spelade in en ljudupptagning på mobiltelefon för att kunna gå tillbaka och analysera.

Självva systemet som vi utvecklat kan ses som en HiFi-prototyp. Den är evolutionär på så sätt att den byggts vidare på under varje iteration. Huruvida kunden kommer fortsätta utvecklingen av denna prototyp på egen hand eller inte är inte bestämt. Ett alternativ är att se vårt system som en prototyp till ett nytt system. Prototypen presenterades och utvärderades i samband med kundmöten vid varje iterationsslut. Ett mer omfattande användartest utfördes i iteration 3.

I iteration 3 av projektet genomfördes användartester av systemet vid två tillfällen. Vid dessa tester kördes det nya systemet parallellt med det gamla. Testen fokuserade på skapande av handböcker och genomförande av operationsförberedelser. Testen genomfördes dels på egen hand av verksamheten och dels med delar av projektgruppen på plats. Vid användartesterna fick vi in feedback från användarna om vad som var bra och vad som kunde göras bättre. Ett exempel på designval som kunde utredas vid testningen var om en operationsförberedelse skulle kunna sättas som klar även om inte alla artiklar var plockade. Olika alternativ hade diskuterats innan men vid testning blev det tydligt att det var en önskvärd funktionalitet. Andra exempel på saker som upptäcktes vid användartesten var hur engångsmaterialet skulle sorteras på bästa sett, hur navigationen mellan olika sidor kunde fungera effektivare och att vissa saker inte användes. Systembuggar kunde också rapporteras och åtgärdas under testningen. Kunden höll också en intern utvärdering om hur användarna upplevde systemet.

Att använda intervjuer och observationer som metoder för kravinsamling kändes naturligt för detta projekt. Det var ett bra tillvägagångssätt för att få en helhetsbild av vad kunden ville ha. Intervjuerna gav oss kundens mål och visioner

av projektet. Observationerna gav oss en inblick i hur användarna använde systemet och saker som var otydliga i intervjuerna kunde fångas in i observationen. På så sett kompletterade dessa metoder varandra och det kändes nödvändigt för en lyckad kravinsamling.

LoFi-Prototyper kunde användas i förstudien för att bekräfta våran bild av vad som skulle byggas. De var ett bra verktyg för att reda ut otydligheter. De var effektiva på så sätt att de gick snabbt att tillverka och gav mycket feedback tillbaks. Det kändes lättare att diskutera systemet när vi hade något visuellt framför oss. De gjorde det lättare att formulera krav och bidrog på så sätt till kravinsamlingen.

HiFi-prototypen kunde användas för att testa systemet utifrån kraven. Vid varje iterationsmöte kunde systemet utvärderas och kraven kunde både formuleras om och prioriteras om. Vi kunde också använda prototypen för att validera vilka krav som var genomförda.

Vid användartesterna i iteration 3 kunde systemet användas för att testa om kraven uppfyllts och formulerats efter kundens behov. Här framkom vissa förändring på krav såsom att mer information skulle visas för en artikel. Prototypen kunde på så sätt användas för att testa och validera krav. För en eventuell vidareutveckling kan observationerna användas som underlag för att skriva nya krav. TODO: Fyll på med resultat från användarutvärdering

D.6 Diskussion

Här diskuteras resultatet av den enskilda utredningen och hur metoden fungerat för att undersöka frågeställningarna.

D.6.1 Resultat

Resultatet bekräftar synen på vad som sägs i teorin...

D.6.2 Metod

TODO:Skriv om/flytta detta Metoden som användes fokuserade på att i förstudien samla in så mycket krav som möjligt genom möten, intervjuer och observationer. Att arbeta fram kraven tillsammans med kund kändes nödvändigt för att få en fullständig bild. En sak som vi inte prioriterades men som kanske hade kunnat hjälpt arbetet skulle varit att lägga större fokus på olika roller. Slutsystemet har två roller vilket är admins och icke-admins. En avvägning gjordes där dessa två enkla roller valdes. Fördelen var att vi kunde lägga fokus på att snabbare kunna testa andra delar av systemet med högre prioritet. Nackdelen var att dessa funktionaliteter inte hann testas och under förstudien ledde till en viss förvirring. Att använda use-cases eller user-stories diskuterades också under förstudien. Detta bortprioriterades då vi ansåg att de var överflödiga. Denna avvägning ledde till att det var lite svårare att kommunicera kraven.

Metoden som användes för att svara på frågeställningarna var att utföra själva projektet. Projektet gav således svar på att intervjuer och observationer var bra för detta projekt. Huruvida metoden passar andra projekt kan diskuteras.

D.7 Slutsatser

Intervjuer och observationer är bra metoder för att analysera kundens behov. I detta projekt kändes de som naturliga och nödvändiga tillvägagångssätt. Prototyper är ett väldigt kraftfullt verktyg för kravinsamling och kan användas på flera olika sätt. LoFi-prototyper är bra i början av ett projekt för att snabbt kunna kommunicera ideér och hitta en bra design. De hjälper på så sätt till i kravinsamlingsprocessen. HiFi-prototyper är bra både för att testa om kraven formulerats på ett bra sätt och för att validera systemets funktionalitet.

D.8 Referenser

- [1] Hull. E, Ken. J och Jeremy. D, Requirements Engineering, Third edition. London: Springer, 2011.
- [2] Lauesen, S. (2002) Software Requirements: Styles and Techniques. Harlow: AddisonWessly.
- [3] Arvola, M. (2014) Interaktionsdesign och UX: Om att skapa en god användarupplevelse.

E Automatiserade tester med Travis CI - Erik Malmberg

E.1 Inledning

Den här enskilda utredningen är en del av kandidatrapporten i kursen TDDD77 vid Linköpings universitet. Utredningen behandlar en del av utvecklingen av ett webb-baserat system för att underlätta förberedelser inför operationer på sjukhusen i Östergötland. Systemet utvecklades på uppdrag av Region Östergötland.

E.1.1 Syfte

Syftet med den här enskilda delen av kandidatarbetet är att ge insikt i hur kontinuerlig integration och automatiserade tester kan användas för att effektivisera testandet i ett projekt som använder en agil utvecklingsmetod. Speciellt ska det undersökas hur väl det går att använda webb-baserade tjänster för att utföra kontinuerliga automatiserade tester av webb-applikationer.

E.1.2 Frågeställning

De frågeställningar som ska besvaras i den här enskilda delen av rapporten är:

- Hur kan man använda webb-baserade tjänster för att utföra kontinuerliga automatiserade tester av webb-applikationer?
- Hur effektivt är det att använda en webb-baserad tjänst för automatiserade tester?
- Vilka typer av tester är svåra att utföra med en sådan tjänst?

I den andra frågeställningen så definieras effektivitet som antalet test som kan utföras per sekund. I svaret på frågeställningen ska även testfallen specificeras noggrant så att svaret inte blir tvetydigt. Det kan även vara intressant att ta reda på hur lång tid det tar från det att koden pushas till GitHub till det att testfallen börjar köras på servern.

E.1.3 Avgränsningar

Inga undersökningar kommer att utföras om hur andra lösningar än Travis CI kan användas för kontinuerlig integration och automatiserade tester. De testfall som kommer användas kommer uteslutande att vara skrivna med ramverket Jasmine. Den webb-applikation som kommer att testas kommer att vara skriven med programmeringsspråket javascript och använda javascriptbiblioteken Node.js och JQuery.

E.2 Bakgrund

Här beskrivs de tjänster, språk och bibliotek som använts under arbetet med den här enskilda rapporten.

E.2.1 Travis CI

Travis CI är en webb-baserad tjänst för att köra automatiserade enhetstester och integrationstester på projekt som finns på GitHub. Travis CI är byggt på öppen källkod och är gratis att använda. Tjänsten har stöd för många olika programmeringsspråk, men det som är relevant för innehållet i den här rapporten är javascript med node.js. För att konfigurera Travis CI används filen `.travis.yml` som placeras i det aktuella projektets repository på GitHub.

E.2.2 Javascript

Javascript är ett programmeringsspråk som i första hand används på klientsidan på webbsidor. Javascript exekveras av webbläsaren och arbetar mot ett gränssnitt som heter Document Object Model (DOM).

E.2.3 JQuery

JQuery är ett javascript-bibliotek som kan användas för att förenkla programmeringen av javascript på klientsidan av en webbsida. JQuery innehåller lättanvänd funktionalitet för händelsehantering och modifiering av HTML-objekt. JQuery är gratis och baserat på öppen källkod som är tillgänglig under en MIT-licens.

E.2.4 Node.js

Node.js är en runtime environment för internetapplikationer. Det kan till exempel användas för att skapa webbservrar. Node.js är baserat på öppen källkod och det är enkelt att lägga till nya moduler för att anpassa det system man vill använda. För att lägga till nya moduler används node package manager (npm).

E.2.5 Jasmine

Jasmine är ett ramverk för testning av Javascript. Den node-modul som används är `grunt-contrib-jasmine` som använder task runnern Grunt för att köra testfall som skrivits med Jasmine. Grunt konfigureras med filen `Gruntfile.js`.

E.3 Teori

Här beskrivs den teori som ligger till grund för rapporten och som visar varför frågeställningarna är relevanta.

E.3.1 Vattenfallsmodellen

I vattenfallsmodellen genomförs all integration och alla tester efter att implementeringen är slutförd. Om ett problem då identifieras under integrationen så är det krångligt att gå tillbaka och åtgärda problemet. Det kan leda till förseningar av projektet. Om felet som upptäcks är så allvarligt att en betydande omdesign måste ske så kommer utvecklingen i stort sett att börja om från början och man kan räkna med en hundra procentig ökning av budgeten, både vad gäller pengar och tid [1].

E.3.2 Kontinuerlig integration och automatiserade tester

Kontinuerlig integration kan leda till att problemen identifieras tidigare i utvecklingsprocessen. Problemen blir då lättare att åtgärda. Automatiserade tester kan effektivisera testprocessen och det finns många tillgängliga lösningar för att köra automatiserade tester [2]. Några av de vanligaste är Travis CI, Codeship och Drone.

E.4 Metod

Arbetet inleddes genom att Travis CI kopplades till projektets repository på GitHub. Kopplingen utfördes genom att administratören för repositoryn loggade in på travis-ci.org med sitt GitHub-konto och aktiverade en webhook för repositoryn.

Inställningarna för Travis CI konfigurerades med filen `.travis.yml` i projektets repository. Språket valdes till javascript med node.js med inställningen: *language: node_js*. Versionen av node.js valdes till version 0.10 med inställningen: *node_js: "0.10"*.

De nödvändiga node-modulerna installerades med hjälp av node package manager (npm). Grunt installerades med kommandot: *npm install -g grunt-cli*. Grunt-contrib-jasmine installerades med kommandot: *npm install grunt-contrib-jasmine*.

Task runnern Grunt konfigurerades med filen `Gruntfile.js` i projektets repository. En task för Jasmine laddades in med inställningen: *grunt.loadNpmTasks('grunt-contrib-jasmine')*. Tasken konfigurerades med följande kod i `Gruntfile.js`.

```
module.exports = function(grunt) {

  grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'),
    jasmine: {
      test: {
        src: './public/js/*.js',
        options: {
          vendor: [
            'public/js/lib/jquery/jquery-2.1.1.js',
            'node_modules/jasmine-jquery/lib/jasmine-jquery.js'
          ],
          keepRunner: true,
          specs: 'test/*-spec.js',
          template: 'test/template/spec-template.tmpl'
        }
      }
    }
  });

  grunt.loadNpmTasks('grunt-contrib-jasmine');
}
```

Med *src: './public/js/*.js'* valdes de filer som skulle testas. Med *vendor*

valdes andra filer som var nödvändiga för att köra testerna. Raden *keepRunner: true* gör att filen `_SpecRunner.html` sparas efter att testerna körts. Filen kan sedan öppnas i en webbläsare och innehåller detaljerad information om utfallet av testerna. Med *specs: 'test/*-spec.js'* valdes de testfall som skulle köras. Alla filer som slutar med `-spec.js` i mappen `test` anses alltså vara testfall som ska köras. Raden *template: 'test/template/spec-template.tpl'* gör att testerna körs med en speciell SpecRunner som även kan innehålla HTML som testfallen kan modifiera. Eftersom Travis CI använder npm för att starta testerna så definierades testskriptet för npm med raden *"test": "grunt jasmine -verbose"* i filen `package.json` i projektets repository.

Testfallen skrevs med Jasmine. Jasmine har en enkel och intuitiv syntax. Ett exempel på ett testfall skrivet med Jasmine följer nedan.

```
describe('The function splitOnce', function() {

  it('can split a string with a char correctly', function() {
    var str = 'a.b.c.d';
    var res = splitOnce(str, '.');
    expect(res[0]).toBe('a');
    expect(res[1]).toBe('b.c.d');
  });
})
```

På den första raden beskrivs vilken del av koden det är som ska testas. På nästa rad beskrivs vad det är som ska testas i den utvalde delen av koden. Med funktionen `expect` kontrolleras att koden har utfört testet på det sätt som förväntats. Flera `expect`-funktioner kan användas i samma testfall.

Ett speciellt testfall skrevs för att besvara den andra frågeställningen om hur effektivt det är att använda en webb-baserad tjänst för automatiserade tester. Det speciella testfallet visas nedan.

```
describe('Travis CI', function() {

  it('can do a lot of tests', function() {
    var date = new Date();
    var startTime = date.getTime();
    var time = startTime;
    var i = 0;

    while (time < startTime + 1000) {
      var str = 'a.b.c.d';
      var res = splitOnce(str, '.');
      expect(res[0]).toBe('a');
      expect(res[1]).toBe('b.c.d');

      i++;
      date = new Date();
      time = date.getTime();
    };

    console.log(i);
  });
})
```

```
});
```

Testfallet testar funktionen `splitOnce()` så många gånger som möjligt under en sekund. Antalet gånger som funktionen hann köras skrivs sedan ut på skärmen med en `console.log()`. Testfallet har körts flera gånger på olika datum och olika tidpunkter. Resultatet av testerna presenteras i en tabell under rubriken Resultat. För att antalet ska få en konkret betydelse visas även funktionen `splitOnce()` nedan.

```
var splitOnce = function(str, split) {  
  var index = str.indexOf(split);  
  if (index === -1) {  
    return [str, ''];  
  }  
  
  return [str.slice(0, index), str.slice(index + 1)];  
};
```

Funktionen tar två paramterar. En sträng (`str`) som ska delas upp och en sträng (`split`) som anger vilket tecken eller vilken teckenkombination som ska dela upp strängen. Funktionen delar endast upp strängen i två delar även om (`split`) förekommer på flera positioner i (`str`). Funktionen returnerar en array med två element. De två elementen är de två delarna av den ursprungliga strängen. Om den andra parametern (`split`) inte existerar i den första parametern (`str`) så returneras hela strängen i det första elementet och en tom sträng i det andra elementet.

För att besvara den tredje frågeställningen om vilka typer av tester som är svåra att utföra med en webb-baserad tjänst så gicks koden igenom och försök till att skriva testfall genomfördes med de olika delarna av koden. Resultatet av dessa tester redovisas under avsnittet Resultat.

E.5 Resultat

Här presenteras resultatet av rapporten.

Ett sätt att använda en webb-baserad tjänst för att utföra kontinuerliga automatiserade tester av en webb-applikation är att använda Travis CI tillsammans med Jasmine på det sätt som beskrivits under avsnittet Metod. Observera att den webb-applikation som testades var skriven med javascript. Node.js användes på serversidan och JQuery användes på klientsidan.

Resultatet av testerna som utfördes för att besvara den andra frågeställningen visas i tabellen nedan.

Tabell ??

Testnummer	Datum (ÅÅ-MM-DD)	Tid (hh-mm-ss)	Test per sekund
1	15-05-01	13:53:34	11380
2	15-05-01	14:27:27	12462
3	15-05-01	14:47:14	9386
4	15-05-03	10:21:57	14093
5	15-05-03	15:43:20	9875
6	15-05-04	09:42:39	10156
7	15-05-04	11:14:43	11933
8	15-05-04	17:42:34	10056
9	15-05-05	08:32:15	12216
10	15-05-05	08:55:02	9767
11	15-05-05	09:15:16	10153
12	15-05-05	10:12:57	6992
13	15-05-06	14:35:46	8703
14	15-05-06	17:01:37	10984
15	15-05-06	18:37:00	10186

Medelvärde är: TODO

Undersökningen om vad som är svårt att testa ledde till följande resultat. Det visade sig att vanliga javascript-funktioner är lätta att testa så länge de ligger utanför JQuery-funktionen `$(document).ready()`. Det kan illustreras med några rader kod.

```
function easyToTest() {  
    return 0;  
}  
  
$(document).ready(function() {  
  
    $('#id1').click(  
        var test = easyToTest();  
    );  
  
    $('#id2').click(  
        //Hard to test  
        var test = 0;  
    );  
});
```

Det är alltså rekommenderat att skriva alla javascript-funktioner utanför JQuery-funktionen `$(document).ready()` eftersom koden då blir lättare att testa.

JQuery-funktioner är i allmänhet svårare att testa än vanliga javascript-funktioner.

E.6 Diskussion

E.6.1 Resultat

E.6.2 Metod

E.7 Slutsatser

Under den här rubriken presenteras svaren på frågeställningarna och framtida arbete som skulle kunna utföras inom det område som behandlats av rapporten.

E.7.1 Hur kan man använda webb-baserade tjänster för att utföra kontinuerliga automatiserade tester av webb-applikationer?

E.7.2 Hur effektivt är det att använda en webb-baserad tjänst för automatiserade tester?

E.7.3 Vilka typer av tester är svåra att utföra med en sådan tjänst?

E.7.4 Framtida arbete inom området

E.8 Referenser

- [1] W.W. Royce, "Managing the development of large software systems," *Proceedings of IEEE WESCON*, pp. 2, aug, 1970. [Online]. Tillgänglig (nytryckt med annan sidnumrering): <http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf>. [Hämtad april 28, 2015].
- [2] O. Karlsson, "Automatiserad testning av webbapplikationer," Linköpings univ., Linköping, Sverige, 2014, pp. 43. [Online]. Tillgänglig: <http://www.diva-portal.org/smash/get/diva2:727654/FULLTEXT01.pdf>. [Hämtad april 19, 2015].

F Checkning av checklistor - Robin Andersson

F.1 Inledning

Vårt system ska innehålla två olika typer av checklistor på olika webbsidor. Om flera användare är inne på samma sida samtidigt och en person checkar en checkruta så ska den checkrutan bli checkad för alla användare som är inne på den webbsidan.

Den ena typen av checklista är en plocklista där det finns information om vilka engångsmaterial som behövs för den operationsförberedelse som användaren är inne på samt vart de materialen kan hämtas någonstans. När en sjuksköterska har plockat en artikel så checkar denne av den artikeln i plocklistan.

Den andra typen av checklista är en lista av operationssalsförberedelser där det kan stå en längre beskrivning om vad som ska göras och när en sjuksköterska har utfört hela den förberedelsen så checkar denne av den förberedelsen i förberedelselistan.

Checklistorna kommer att implementeras med hjälp av handlebars, javascript, jquery samt Socket.IO. För information om dessa språk/paket hänvisar jag till bakgrunden i den gemensamma rapporten.

F.1.1 Syfte

Syftet med denna del av projektet är att flera sjuksköterskor samtidigt ska kunna förbereda operationer genom att plocka olika artiklar samt förbereda operationssalen och checka av det som är utfört utan att det ska bli några konflikter med att flera sjuksköterskor plockar samma artikel eller liknande.

F.1.2 Frågeställning

- Går det att anpassa checklistan för en surfplatta medan den samtidigt innehåller information om var artiklar befinner sig samt hur många av varje artikel som behövs?
- Kommer Socket.IO vara tillräckligt snabbt för att flera personer ska kunna checka av artiklar samtidigt utan förvirring?

F.1.3 Avgränsningar

Eftersom denna del av projektet endast innehåller checkande av checklistor så saknas etiska aspekter.

F.2 Teori

F.2.1 Websockets

WebSockets består av ett nätverksprotokoll och ett API som gör det möjligt att skapa en WebSocket uppkoppling mellan en klient och en server. Med hjälp av WebSocket API:t så kan man hantera en full-duplex kanal som kan användas till att skicka och ta emot meddelanden. En WebSocket anslutning skapas genom att gå över från HTTP protokollet till WebSockets nätverksprotokoll när en initial handskakning mellan server och klient sker. När en WebSocket anslutning finns uppkopplad så kan WebSocket meddelanden skickas fram och tillbaka mellan metoderna som finns definierade i WebSockets gränssnitt. När WebSockets används i ett system så används asynkrona eventlyssnare för att lyssna på kommunikationen. WebSockets API är rent eventbaserat vilket betyder att klienter inte behöver kontrollera om servern har uppdaterats utan att de istället uppdaterar när de får ett event. [1]

Kanalen som WebSocket använder sig av kommunicerar över nätet med hjälp av en enda socket. WebSockets använder sig av väldigt mycket mindre trafik och har mycket kortare latens än Ajax. WebSockets använder sig av de vanliga HTTP portarna (det vill säga port 80 och port 443) [2]

Huvuddelen av denna del av projektet handlar om kommunikation med Socket.IO som använder sig utav WebSockets för att kommunicera mellan server och klienter.

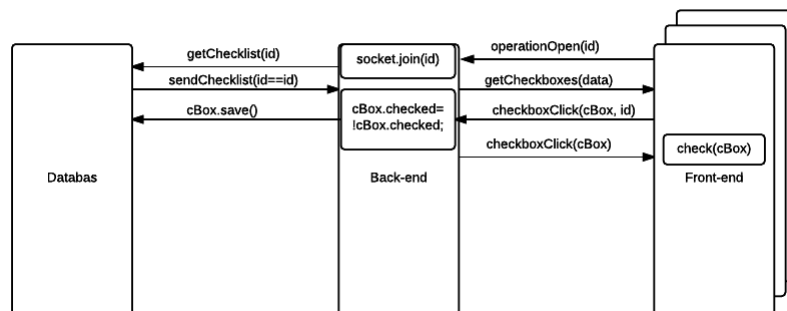
F.3 Metod

Jag började med att fundera på hur kommunikationen skulle fungera på för sätt. Jag skissade ner olika förslag på ett papper och kom på det sättet fram till hur jag skulle implementera kommunikationen. Sedan implementerade jag den och fick den att fungera för plocklistan. Därefter så refaktorerade jag koden för att få den kortare och mer lättläst. När plocklistan sedan var färdig implementerad så började jag fundera på hur jag skulle kunna använda mig av så mycket kod som möjligt från plocklistan till att implementera förberedelselistan. Efter det att jag implementerat förberedelselistan så refaktorerade jag igen för att få koden mer lättläst.

F.4 Resultat

Jag kom fram till att när en användare går in på en operationsförberedelse så kommer denne in i ett rum. Varje gång en person sedan checkar en checkbox så skickas ett Socket.IO meddelande till servern som innehåller information om vilken checkruta som ska checkas samt vilket rum checkboxen ska checkas i. Servern skickar sedan ett meddelande till det givna rummet vilken checkruta som ska checkas och alla klienter som är anslutna till det rummet checkar den givna checkrutan.

Figuren nedan visar detta flöde i ett sekvens liknande box-and-line-diagram.



När två klienter går in på en operation och en klient checkar en checkruta för första gången tar det strax under en sekund innan checkrutan checkas för den andra klienten. Därefter när någon klient checkar en checkruta så kan jag inte se någon fördröjning alls från det att en klient checkar en checkruta och en annan klient får den checkrutan checkad.

Kunden har provat att ha flera personer inne på samma plocklista samtidigt och checka av olika artiklar. Kunden tyckte att det fungerade bra och påpekade inte någon fördröjning.

All information som krävs för plocklistorna fick plats utan att det blev för plottrigt.

F.5 Diskussion

F.5.1 Resultat

Eftersom jag endast skickar data om vilken checkruta som ska checkas till de klienter som är inne på den operation som checkrutan blev checkad på så uppdateras checkningar snabbare än att göra den enkla lösningen att bara skicka datat till alla anslutna klienter. Att det tar nästan en sekund för en checkning att uppdateras på andra klienter för första gången är långsammare än förväntat. Men eftersom det endast gäller just första artikeln och att kunden har testat checkning med flera personer samtidigt utan att märka några problem så verkar detta inte vara något praktiskt problem. Att en checkning sedan kan uppdateras nästan helt utan fördröjning var bättre än vad jag hade förväntat mig.

F.5.2 Metod

Den metod jag använde mig av fungerade bra, men jag tror att jag skulle kunnat komma fram till samma resultat snabbare genom att göra kortare funktioner och vettigare namn redan från början istället för att göra något som funkar så snabbt som möjligt och sedan refaktorisera. För nu blev det väldigt förvirrande kod från början och jag var tvungen att sitta och tänka på vad kod jag skrivit faktiskt gjorde. Men att skissa olika förslag på ett papper först tror jag var en väldigt bra idé, det gjorde att jag fick några möjliga lösningar och sedan kunde jag överväga fördelar och nackdelar med de olika lösningarna för att sedan välja den som verkade bäst.

F.6 Slutsatser

Eftersom det fungerar bra med checkning av checklistor med hjälp av socket.io och all nödvändig information får plats utan att det upplevs som plottrigt så uppfylls syftet med denna del av projektet och min frågeställning har blivit besvarad.

F.7 Referenser

- [1] Wang Vanessa, Salim Frank, Moskovits Peter; The Definitive Guide to HTML5 WebSocket; New York City APress, 2013.
- [2] <http://www.scirp.org/journal/PaperInformation.aspx?PaperID=25428>

G Vidareutveckling av applikation för Region Östergötland - Albert Karlsson

G.1 Inledning

Denna del i rapporten behandlar vad som ska utredas och varför.

G.1.1 Syfte

Syftet med denna enskilda utredningen är att underlätta för fortsatt utveckling av webbapplikationen som projektgruppen har skapat. Alla delar i applikationen får inte användas i Region Östergötlands intranät, vilket leder till att en del måste bytas ut.

G.1.2 Frågeställning

- Vad används Keystone till i applikationen?
- Vad behövs för att ersätta Keystone?
- Vad behöver ändras för att byta ut databasen från MongoDB till MSSQL?
- Vilka moduler eller bibliotek i applikationen kräver en licens för kommersiell användning?

G.1.3 Avgränsningar

Denna rapport gäller endast för vidareutveckling för användning av Region Östergötland. Andra användare kan ha andra krav på applikationen som leder till att denna rapport är ofullständig eller felaktig.

G.2 Bakgrund

Region Östergötland ska ta över arbetet med utvecklingen av webbapplikationen efter att projektgruppen slutfört sitt arbete. För att applikationen ska kunna tas i bruk på riktigt så måste databasen bytas ut till MSSQL då Region Östergötland inte tillåter MongoDB på sina servrar. Då utvecklingen av applikationen har fortgått har Keystone fått en mindre och mindre roll i applikationen. En av keystones största fördelar är dess enkla och smidiga innehållshanterare. Detta har varit till stor nytta för att komma igång med projektet och få snabba resultat för de gruppmedlemmar som inte har hållt på med webbprogrammering innan. Men när applikationen är färdigutvecklad så används inte detta systemet alls längre och då databasen ska bytas till en annan typ försvinner också en annan stor del av Keystone, vilket var den enkla integrationen med MongoDB genom Mongoose. Detta ledde till tankar om att Keystone kanske skulle kunna bytas ut mot egenskriven kod eller mindre och mer lättförståliga moduler utan jättemycket arbete.

G.3 Teori

En beskrivning och förklaring för många av modulerna som kommer tas upp finns att läsa i avsnitt 3.

G.3.1 MSSQL

MSSQL är en databashanterare från Microsoft som använder det domänspecifika språket SQL för att extrahera data.

G.3.2 npm

Pakethanteraren npm används av Node.js för att hantera open-source paket. Det finns även en tillhörande hemsida för npm där teknisk dokumentation med mera för olika paket kan läsas.

G.3.3 Express

G.3.4 Edge.js

Edge.js är en modul, som är skapad av Tomasz Janczuk, som bland annat gör det möjligt att köra .NET kod direkt i samma process som Node.js.

G.4 Metod

För att få en bättre förståelse för Keystones roll i applikationen så läses först och främst Keystones tekniska dokumentation. En ny installation av Keystone görs för att kunna jämföra med projektkoden och få fram vilka komponenter som kommer från Keystone. Keystone är också beroende av många moduler. Dessa moduler utgör en stor del av den delen av Keystone som används i applikationen, t.ex. för routing. Modulerna kommer utvärderas för att se vilka som fortfarande skulle bidra till en version av applikationen utan Keystone.

För att utvärdera MSSQL-moduler och se vad som kommer krävas för att få applikationen att funka med MSSQL så kommer främst artiklar om ämnet läsas och då ett bra alternativ har hittats så kommer en liten del av applikationen tas och anpassas till MSSQL och testas. Om inte modulen uppfyller kraven på enkelhet och stabilitet så kommer ett annat alternativ tas fram och testas.

Information om olika moduler kommer tas från respektive utgivares hemsida eller, i de fall de inte finns, npmjs.com eller github. express MIT

G.5 Resultat

Keystone används främst till att skapa ett enkelt användargränssnitt för modifiering av innehållet i databasen.

<https://www.microsoft.com/en-us/download/details.aspx?id=29995>

Det finns flera olika moduler för att koppla ihop en Node.js-applikation med MSSQL. En av dessa är en officiell modul från Microsoft. Denna har dock inte släppts i slutgiltig version utan finns bara som en förhandstitt, vilken släpptes den första augusti 2013 och har enligt github repot inte blivit jobbad på sedan dess. Att använda en officiell modul från Microsoft hade gett många fördelar för Region Östergötland så som officiell support och garanti att den skulle fungera. Men då den inte släppts som slutgiltig version och inte verkar utvecklas längre så uppfyller den inte de krav som ställs.

Edge.js kan användas för att köra .NET kod direkt i applikationen och på så sätt hämta data från en MSSQL-server. Att använda Edge.js istället för något modul gör också att det inte finns några begränsningar i hur hämtningen från databasen ska gå till. Det finns en modul till Node.js som hete edge-sql som just

är till för att hämta data från en MSSQL databas. Denna modulen gör att man direkt kan skriva SQL-frågor i javascript-koden. Genom att skriva funktioner för uppdatering, sökning, borttagning och tilläggning för varje modell så skulle tiden för anpassningen till MSSQL minska, eftersom det inte blir lika mycket kod att ändra på.

G.6 Diskussion

G.6.1 Resultat

G.6.2 Metod

G.7 Slutsatser

G.8 Referenser