

Kandidatarbete

Redaktör: Pål Kastman

Datum: 2015-05-13

Version 0.4

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
2015-05-11	0.3	Tredje utkastet av gemensam och individuella rapporter	Alla
2015-05-13	0.4	Fjärde 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	2
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	3
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	4
4.1	Kravinsamling	4
4.1.1	Förstudie	4
4.1.2	Kravspecifikation i Google docs	4
4.1.3	Kravrepresentation	5
4.1.4	Kravvalidering	5
4.2	Utveckling	5
4.2.1	SCRUM	6
4.2.2	Alpha state cards	6
4.2.3	Sammarbete i gruppen	6
5	Resultat	8
5.1	Översikt av systemet	8
5.1.1	Server och klient	8
5.1.2	Kartoteket	8
5.2	Tekniker	9
5.2.1	Översikt	9
5.2.2	Back-end	9
5.2.3	Front-end	10
5.2.4	Struktur	10
5.2.5	Säkerhetskopiering och reservsystem	10
5.3	Mekanismer	11
5.3.1	Översiktsvy	11
5.3.2	Handbok	12
5.3.3	Sökfunktion	12
5.3.4	Lista med handböcker	13
5.3.5	Tillstånd	13
5.3.6	Redigering	14

5.3.7	Operationsförberedelse	15
5.3.8	Plocklista	16
5.3.9	Kartoteket	16
5.4	Utvecklingen	17
5.4.1	SCRUM	17
5.4.2	Alpha state cards	17
5.4.3	Kundmöten	17
5.4.4	Samarbete i gruppen	18
6	Diskussion	19
6.1	Resultat	19
6.1.1	Back-end	19
6.1.2	Front-end	19
6.1.3	Struktur	20
6.1.4	Säkerhetskopiering och reservsystem	20
6.1.5	Översiktssida	20
6.2	Metod	20
6.2.1	Kravinsamling	20
6.2.2	Utvecklingen	21
6.3	Arbetet i ett vidare sammanhang	21
7	Slutsatser	21
8	Fortsatt arbete	21
8.1	Restlista	21
8.2	Referenser	22
Del II	Enskilda utredningar	23
A	Kartoteket - Daniel Rapp	23
A.1	Inledning	23
A.1.1	Syfte	23
A.1.2	Frågeställning	23
A.1.3	Avgränsningar	23
A.2	Bakgrund	24
A.3	Metod	24
A.4	Resultat	25
A.4.1	Att se artiklarna	25
A.4.2	Modifiering av artiklarna	26
A.4.3	Sökning av artiklarna	26
A.4.4	Integrering med handböckerna	27
A.5	Diskussion	27
A.6	Slutsatser	27
A.7	Referenser	28
B	Jonas Andersson	29
B.1	Inledning	29
B.1.1	Syfte	29
B.1.2	Frågeställning	29
B.1.3	Avgränsningar	29

B.2	Bakgrund	29
B.3	Teori	30
B.3.1	PHP	30
B.3.2	Python	30
B.3.3	Node.js	30
B.4	Metod	31
B.5	Resultat	31
B.5.1	Fördelar och nackdelar med Node.js	31
B.5.2	Fördelar och nackdelar med externa bibliotek	32
B.6	Diskussion	32
B.6.1	Resultat	32
B.6.2	Metod	32
B.7	Slutsatser	32
B.8	Referenser	33
C	Pål Kastman	34
C.1	Inledning	34
C.1.1	Syfte	34
C.1.2	Frågeställning	34
C.1.3	Avgränsningar	34
C.2	Bakgrund	34
C.3	Teori	35
C.3.1	Google docs	35
C.3.2	Latex	35
C.4	Metod	35
C.5	Resultat	36
C.5.1	Dokumentstandard	36
C.5.2	Revisionshantering	36
C.5.3	Kommentering	37
C.5.4	Referenser	37
C.6	Diskussion	37
C.6.1	Resultat	37
C.6.2	Metod	38
C.7	Slutsatser	38
C.8	Referenser	38
D	Daniel Falk	39
D.1	Inledning	39
D.1.1	Syfte	39
D.1.2	Frågeställning	39
D.1.3	Avgränsningar	39
D.2	Bakgrund	39
D.3	Teori	39
D.3.1	Prototyper	40
D.3.2	Intervjuer	40
D.3.3	Observationer	40
D.4	Metod	40
D.4.1	Intervjuer	40
D.4.2	Observationer	41
D.4.3	LoFi-prototyper	41

D.4.4	HiFi-prototyp	41
D.4.5	Användartest	42
D.5	Resultat	42
D.6	Diskussion	43
D.6.1	Resultat	43
D.6.2	Metod	43
D.7	Slutsatser	43
D.8	Referenser	43
E	Automatiserade tester av webbapplikationer. - Erik Malmberg	45
E.1	Inledning	45
E.1.1	Syfte	45
E.1.2	Frågeställning	45
E.1.3	Avgränsningar	45
E.2	Bakgrund	46
E.2.1	Travis CI	46
E.2.2	Javascript	46
E.2.3	JQuery	46
E.2.4	Node.js	46
E.2.5	Jasmine	46
E.3	Teori	47
E.3.1	Vattenfallsmodellen	47
E.3.2	Kontinuerlig integration och automatiserade tester	47
E.4	Metod	47
E.5	Resultat	50
E.6	Diskussion	52
E.6.1	Resultat	52
E.6.2	Metod	52
E.7	Slutsatser	52
E.7.1	Hur kan man använda webbaserade tjänster för att utföra kontinuerliga automatiserade tester av webbapplikationer?	53
E.7.2	Hur effektivt är det att använda en webbaserad tjänst för automatiserade tester?	53
E.7.3	Vilka typer av tester är svåra att utföra med en sådan tjänst?	53
E.7.4	Framtida arbete inom området	53
E.8	Referenser	54
F	Checkning av checklistor - Robin Andersson	55
F.1	Inledning	55
F.1.1	Syfte	55
F.1.2	Frågeställning	55
F.1.3	Avgränsningar	55
F.2	Teori	56
F.2.1	Websockets	56
F.3	Metod	57
F.4	Resultat	57
F.5	Diskussion	58
F.5.1	Resultat	58
F.5.2	Metod	58

F.6	Slutsatser	59
F.7	Referenser	59
G	Vidareutveckling av applikation för Region Östergötland - Albert Karlsson	60
G.1	Inledning	60
G.1.1	Syfte	60
G.1.2	Frågeställning	60
G.1.3	Avgränsningar	60
G.2	Bakgrund	60
G.2.1	MS SQL	61
G.2.2	Npm	61
G.2.3	Express	61
G.2.4	Edge.js	61
G.2.5	edge-sql	61
G.3	Teori	61
G.4	Metod	62
G.5	Resultat	62
G.6	Diskussion	64
G.6.1	Resultat	64
G.6.2	Metod	65
G.7	Slutsatser	65
G.8	Referenser	65

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 realiseras så arbetet blir lättare och mer effektivt?
- Vilka strategier kan användas för effektiv utveckling i en grupp där kunskapsnivån varierar?

1.4 Avgränsningar

Den här rapporten beskriver hur ett system för operationsförberedelser på universitetssjukhuset i Linköping kan realiseras. 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 TDDD77 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. mediatyp 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 information om lagerplats finns lättillgängligt gentemot hur lång tid det tar med nuvarande system. Resultatet av detta var att det tog 5 minuter för en person att plocka materialet då lagerplats fanns tillgängligt och 30 minuter för en sjuksköterska med lång erfarenhet med nuvarande system [1]. Detta visar att det finns stora tidsvinster i att bygga ett system som underlättar operationsförberedelser.

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 representeras. 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 våran 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

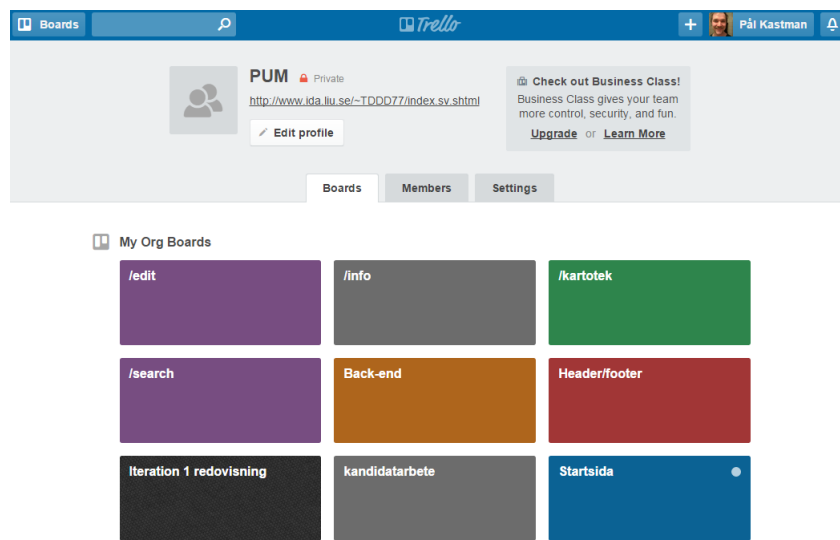
Vid varje iterationsslut hölls ett möte med kund där vi demonstrerade nya features och lät kunden testa systemet. Iterationerna gjorde att vi snabbt kunde rätta till eventuella missförstånd. Vi gick igenom vilka krav som var genomförda och vilka som skulle prioriteras till nästa iteration. Kravspecifikationen 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 Utveckling

Projektets utveckling har skett i fyra iterationer där den första iteration var en förstudie och i de avslutande tre iterationerna har utvecklingen av produkten skett. Nedan beskrivs vilka hjälpmedel som har använts för att underlätta arbetet.

4.2.1 SCRUM

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 (se figur 1).



Figur 1: Översiktsvy över alla Trello-boards

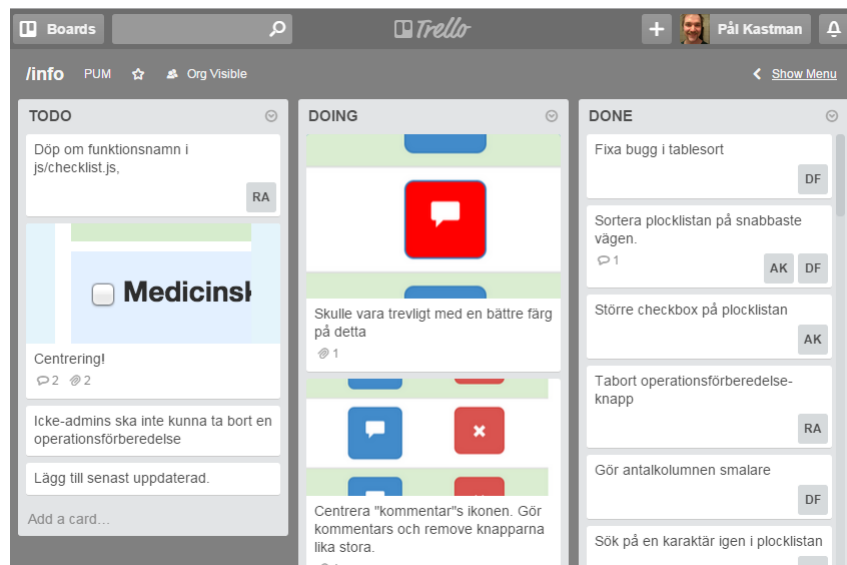
Boarden hade kategorierna TODO, DOING och DONE enligt figur 2. När en medlem valt en aktivitet så märktes aktiviteten med medlemmens namn och flyttades till den aktuella kategorin. Varje vecka har gruppen haft ett kort SCRUM-möte på 15 minuter. På mötet har varje gruppmedlem berättat vad som har gjorts den föregående veckan och vad som ska göras nästa vecka. SCRUM-mötet har oftast utförts i samband med varje veckas handledarmöte.

4.2.2 Alpha state cards

Ett system med alpha state cards har använts för att gruppen skulle kunna ha koll på 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.

4.2.3 Sammarbete i gruppen

Under utvecklingsfasen delade vi till en början upp oss i två grupper som fokuserade på front-end och back-end. När vi sedan kom längre fram i projektet så övergick de flesta av oss till att endast arbeta på front-end.



Figur 2: Boarden info

5 Resultat

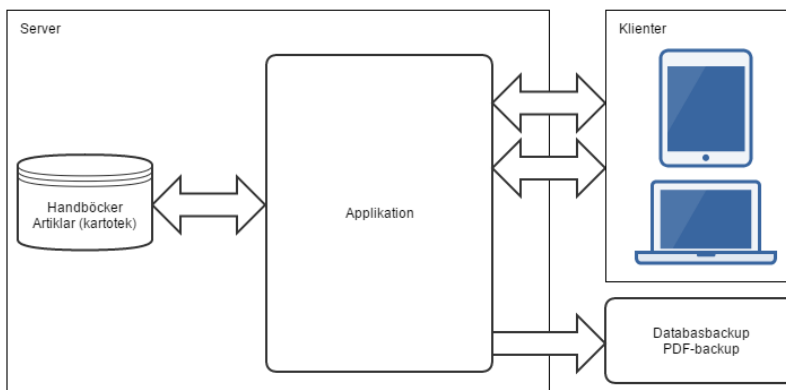
Denna del av dokumentet presenterar resultatet av vårt arbete. Här kommer beskrivs både hur systemet ser ut och används, samt hur det är uppbyggt rent tekniskt. Vidare beskrivs också hur utvecklingen inom gruppen fungerat.

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 3.



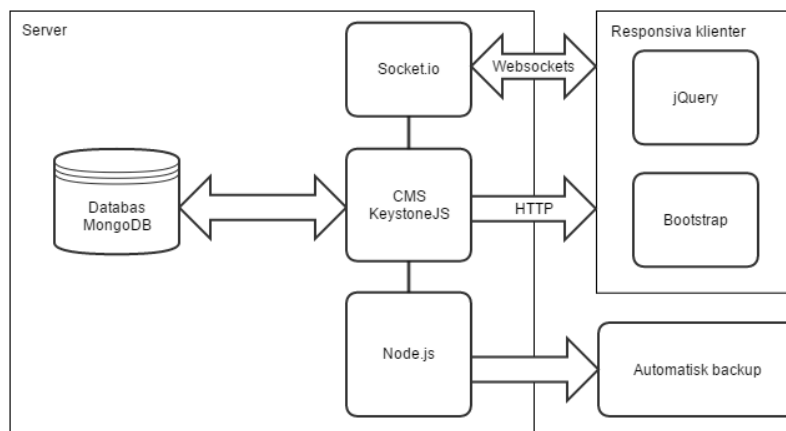
Figur 3: Översikt av systemet

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.



Figur 4: En översikt över tekniken

Läs mer om kartoteket i "Kartoteket" nedan.

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 4 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. Grunden till programmet är node.js vilket är en plattform för att utveckla självständiga program i javascript med inbyggd pakethanterare.

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


För realtidskommunikation används ett programmeringsinterface vid namn Socket.IO.¹ Socket.IO 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.

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

Publicerad Handbok

4-kärlsangiografi

Specialitet: Endovaskulär, intern

Medicinskteknisk utrustning OP	Röntgenapparat AXIOM Artis Zee
	
Sterildrapering	Angiodynset med båda ljumskar synliga
Engångsmaterial	1 st Angiodynset 1 st Kanyl DCM-18-7.0 1 st Introducer Fr 5 RS (ingen ledare behövs) 1 st Angiografikateter 5F Pig 1 st Optiray kontrastspruta

Figur 5: Början på en pdf-kopia.

5.2.3 Front-end

På klientsidan används bootstrap², jQuery³ och LESS.

5.2.4 Struktur

Varje enskild sida har minst en skriptfil, en css fil och en html fil. På de sidor där det blivit stora skript så har skriptfilen delats upp i flera filer.

5.2.5 Säkerhetskopiering och reservsystem

Det ställs stora krav på att handböckerna alltid ska finnas tillgängliga då de ska användas på ett sjukhus där ett fel kan få stora konsekvenser. Detta innebär att ett reservsystem måste finnas till hands ifall systemet slutar fungera. Detta har lösts genom ett system där pdf-kopior av handböcker skapas. Handböckerna hamnar i en mapp-struktur där de sorteras på specialitet och operation. Kopior sparas med ett versionnummer, vilket gör att kopior av gamla versioner av operationer fortfarande finns kvar och kan skrivas ut. Var denna mapp-strukturen ska hamna bestäms i en konfigurationsfil. Kopiorna skapas genom att en funktion, som kollar igenom alla handböcker för att se om har uppdaterats sedan senaste kopieringen, körs med ett givet tidsintervall som ställs in i konfigurationsfilen. Om en operation har uppdaterats så körs en funktion som använder en modul, som heter wkhtmltopdf, för att kalla på ett program som också heter wkhtmltopdf. Wkhtmltopdf använder en osynlig webbläsare för att skapa en pdf-kopia. Hur kopiorna ser ut kan ses i figur 5 och figur 6. Utseendet ändras genom att speciell css för utskrift finns och wkhtmltopdf körs med utskrift som mediatyp.

Säkerhetskopiering av databasen finns implementerat. Det finns ett tidsintervall som går att ställa in i konfigurationsfilen så körs en funktion som använder en

²TODO: Bättre referens.

³TODO: Bättre referens.

Antal	Namn	Artikelnummer	Klinik	Förråd	Sektion	Hylla	Fack
1	Aci-slang dagset	284504	Operation Ortopeden US	SS	A	1-2	
1	Artärkanyl 45Mm. Steril	340901	GEM op/ane mtrl NORD	NO	50	E	
1	Cvk-Set Venkateter 1.2X 90 Mm S Selacon-T Med Integrerad Flowschwich Grön	681000	GEM op/ane mtrl NORD	NO	15	G	1

Figur 6: Plocklista i en pdf-kopia.

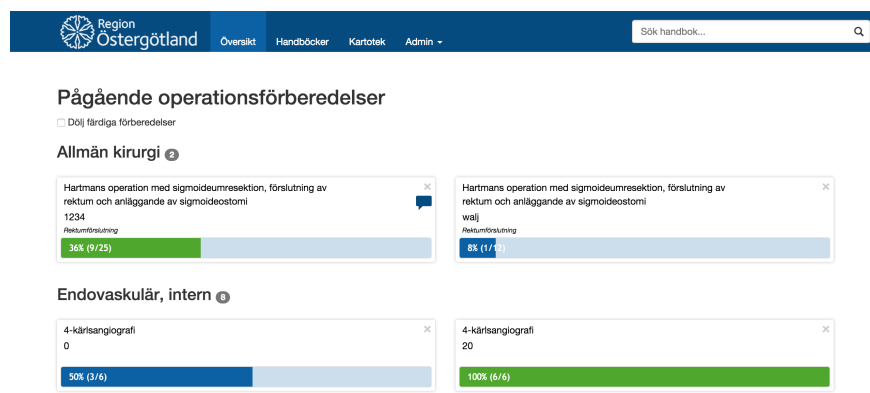
modul som heter mongo-utils för att kalla på mongodump som tar en kopia på databasen och lägger den i en mapp med dagens datum.

5.3 Mekanismer

Här beskrivs syftet och funktionalitet hos olika delar av produkten.

5.3.1 Översiktsvy

Som tidigare nämnts finns en översiktsvy över alla operationsförberedelser. Den-
na sida visar alla operationsförberedelser och hur långt de är fortskridna, alltså
hur många procent av förberedelserna och artiklarna som checkats av. Här
används Socket.IO för att hela tiden hålla information uppdaterad.



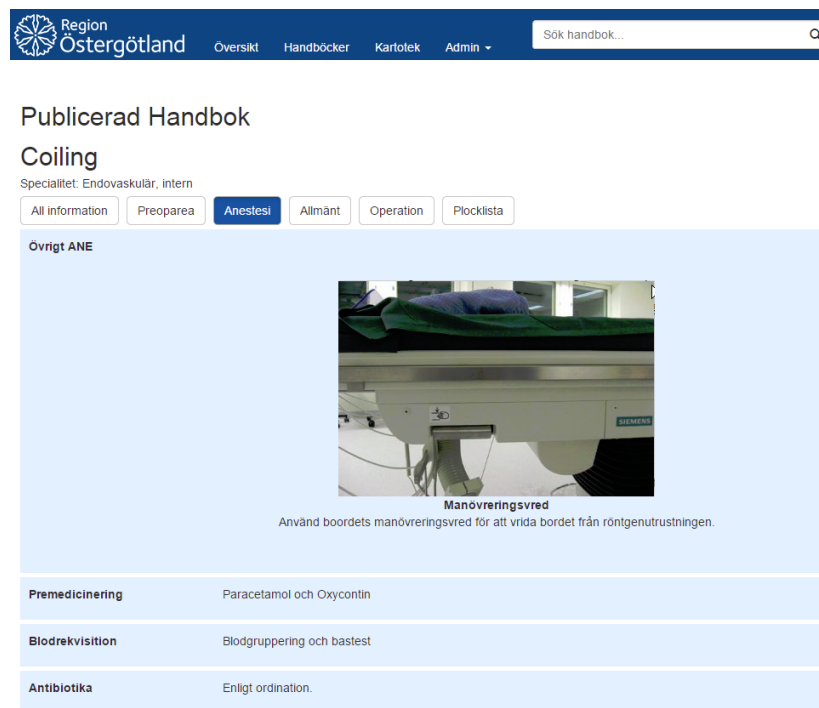
Figur 7: Bild på översiktsvyn

I figur 7 kan man se hur förberedelserna är kategoriserade beroende på vilken kirurgisk specialitet handboken tillhör. Ibland är det olika samordnare beronde på specialitet och det är då enkelt för en samordnare att hitta de operationerna som personen är ansvarig över. En ikon för kommentarer dyker upp om någon valt

att kommentera en artikel. En samordnade kan då klicka på kommentarsikonen för att få upp en lista över dessa kommentarer.

5.3.2 Handbok

En handbok innehåller information om en operationsförberedelse. All information i en handbok är uppdelad i olika rubriker. Dessa rubriker är i sin tur uppdelade i olika processer. I figur 8 ser man dels de olika processerna (Preoparea, anestesi, allmänt och operation) samt rubrikerna som hör till processen Anestesi (Övrigt, premedicinering, blodrekvisition och antibiotika).

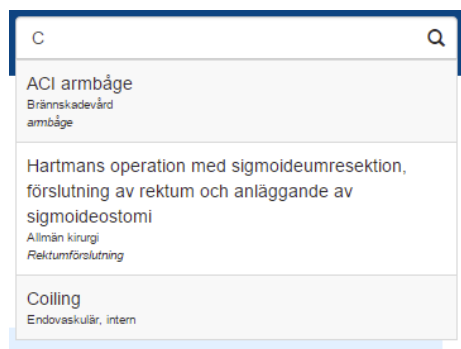


Figur 8: En handbok

Man kan även se att handböckerna har stöd för bilder.

5.3.3 Sökfunktion

Ett krav är att det ska vara lätt att hitta en handbok. Därför kan man söka både på operationens namn men även på alternativa sökord (taggar), dessa är bra att ha då de medicinska termerna ibland kan vara svåra att komma ihåg. I figur 9 kan man se sökresultaten där namnet på operationen står i större storlek, och sökorden kursivt i mindre storlek.



Figur 9: Sökfunktionen

5.3.4 Lista med handböcker

Applikationen innehåller en enkel lista med alla handböcker. En vanlig användare kan se handböcker som är publicerade medan en administratör också kan se handböcker i tillstånden utkast, redigering och granskning. Listan går att sortera på valfri kolumn och kan även grupperas beroende på vilken specialitet handboken tillhör. Se figur 10 för ett exempel.

The image shows a web application interface for 'Region Östergötland'. It has a navigation bar with 'Oversikt', 'Handböcker', 'Kartotek', and 'Admin'. A search bar is present with the text 'Sök handbok...'. Below the navigation bar, there is a section titled 'Antal publicerade handböcker: 13'. There are two dropdown menus: 'Specialitet:' with 'Allmän kirurgi' selected, and 'Tillstånd:' with 'Alla tillstånd' selected. Below these is a table with the following data:

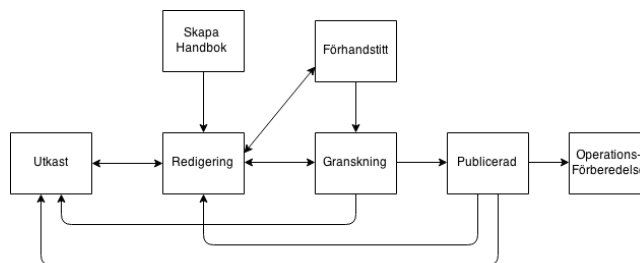
#	Namn	Sökbord	Specialitet	Tillstånd	Senast sparad
1	Appendectomy		Allmän kirurgi	Utkast	22/3 2015 13:52
2	Exstirpation TC	Tumör,,tumör,hjärntumör,TC	Allmän kirurgi	Redigering	14/3 2015 10:32
3	Handboktest		Allmän kirurgi	Publicerad	27/3 2015 10:51
4	Hartmans operation med sigmoideumresektion, förslutning av rektum och anläggande av sigmoideostomi	Rektumförslutning	Allmän kirurgi	Publicerad	26/3 2015 18:29
5	Laparoskopisk pankreasresektion	Laparoskopi,lap.skopi,pankreas,pankreasresektion,pankreatumör	Allmän kirurgi	Granskning	22/3 2015 11:20
6	Mag-tarmkanalsförtorkning		Allmän kirurgi	Utkast	15/3 2015 14:31

Figur 10: Lista med handböcker

5.3.5 Tillstånd

Flödet för att skapa en handbok involverar flera steg och en handbok kan finnas i olika tillstånd. Se figur 11 för en översy.

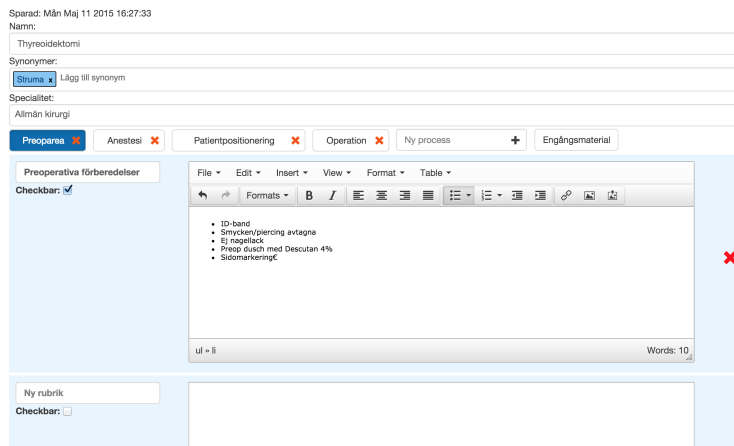
När man först vill skapa en handbok så hamnar man i ett redigeringsläge, där man kan lägga till information om handboken, som artiklar i plocklista, beskrivning av operationen. Härifrån kan man välja att förhandsgranska, för att se om den ser ok ut, eller skicka till granskning, där den går vidare till publicering efter någon annan läst igenom materialet och de tycker det ser rätt ut. Om man har en handbok i redigeringsläge som inte är klar så hamnar den även i "utkast"-läge, likt en email-system. Slutligen kan en publicerad handbok gå vidare till att bli en operationsförberedelse om det är dags att utföra en sådan operation som beskrivs i handboken.



Figur 11: Handböcker och deras olika tillstånd

5.3.6 Redigering

I redigeringsvyn skapar man innehållet i en handbok. Det går bland annat lägga till synonymer, processteg med tillhörande information och artiklar från kartoteket. I figur 12 visas ett exempel på hur denna redigering ser ut.



Figur 12: Redigering av plocklista för handbok

I exemplet har underrubriken "Preoperativa förberedelser" lagts till under processteget "Preopera". Denna redigering använder sig av en wysiwyg-editor. Innehållet kan på så sätt formateras efter olika behov. I exemplet har innehållet strukturerats som en punktlista.

I redigeringsvyn kan man utöver att redigera all information även sortera processer och rubriker genom att dra och släppa dem.

Under processteget engångsmaterial skapar man plocklistan. Ett exempel visas i figur 13. Artiklar kan läggas till genom att söka på dem i kartoteket och antalet kan redigeras.

Innan en redigerad eller ny handbok publiceras måste den granskas av en annan person. Detta görs genom att trycka på knappen skicka till granskning längst ned i redigeringsvyn.

Sparad: Mån Maj 11 2015 16:27:33

Namn: Thyreoidsektomi

Synonymer: [Lägg till synonym](#)

Specialitet: [Allmän kirurgi](#)

Preoparea ☒ Anestesi ☒ Patientpositionering ☒ Operation ☒ Ny process Engångsmaterial

Antal	Namn	Artikelnummer	Klinik	Förråd	Sektion	Hylla	Fack
- 2 +	Artärkanyt 45Mm. Steril	340901	GEM op/ane mtri NORD	NO	50	E	<input checked="" type="checkbox"/>
- 1 +	Kardborre/Slanghållare	258280	GEM op/ane mtri NORD	NS	39	I	<input checked="" type="checkbox"/>

knivblad

Namn	Artikelnummer	Klinik	Förråd	Sektion	Hylla	Fack
Knivblad 11	37110101	GEM op/ane mtri SYD	SS	39	F	<input checked="" type="checkbox"/>
Knivblad 15	37110103	GEM op/ane mtri SYD	SS	39	F	<input checked="" type="checkbox"/>
Knivblad 21	371103	GEM op/ane mtri SYD	SS	39	F	<input checked="" type="checkbox"/>
Knivblad 25	37110505	GEM op/ane mtri SYD	SS	39	F	<input checked="" type="checkbox"/>

Figur 13: Redigering av plocklista för handbok

5.3.7 Operationsförberedelse

En operationsförberedelse är en instans av en handbok vilket medför att att flera operationsförberedelser kan skapas från samma handbok. En operationsförberedelse ser nästan likadan ut som en handbok. Det som skiljer sig är att vissa av rubrikerna för förberedelser och artiklar i plocklistan går att checka av när de är förberedda eller plockade.

Region Östergötland [Översikt](#) [Handböcker](#) [Kartotek](#) [Admin](#)

Operationsförberedelse

☒ Hartmans operation med sigmoideumresektion, förslutning av rektum och anläggande av sigmoideostomi

Specialitet: Allmän kirurgi
Synonymer: Rektumförslutning
Linda-ID: 1234

[All information](#) [Anestesi](#) [Preoparea](#) [Engångsmaterial](#)

Beskrivning	Avlägsnande av sigmoideum, förslutning av rektum och sigmoideostomi
Indikation	Oftast divertikulit eller cancer med komplikationer
Medicinskeknisk utrustning OP	Diatermi, Sug
Patientposition OP	Rygg
Elimination OP	Cystofix

Figur 14: Förberedelsevyn

I figur 14 kan man se att rubrikerna går att checka av när de är klara. Jämför med figur 8 där rubrikerna inte går att checka av. När operationen är klar kan en administratör ta bort operationsförberedelsen.

5.3.8 Plocklista

En operationsförberedelse har en plocklista med artiklar som behövs till operationen. Den ligger under fliken engångsmaterial. Listan används för att hitta artiklar i lagret och för att checka av dem när de plockats. För att underlätta plockningen och göra den mer effektiv kan man välja att sortera artiklarna beroende på plockplats.

Operationsförberedelse

☒ 4-kärlsangiografi

Specialitet: Endovaskulär, intern
Linda-ID: 0

	Antal	Namn	Artikelnummer	Klinik	Förråd	Sektion	Hylla	Fack
<input checked="" type="checkbox"/>	1	Aci-slang dagset	284504	Operation Ortopeden US	SS	A	1-2	<input type="button" value="Kommentar"/>
<input type="checkbox"/>	3	Artärkanyl 45Mm. Steril	340901	GEM op/ane mtrlr NORD	NO	50	E	<input type="button" value="Kommentar"/>
<input checked="" type="checkbox"/>	1	Cvk-Set Venkateter 1.2X 90 Mm S Selacon-T Med Integrerad Flows witch Grön	681000	GEM op/ane mtrlr NORD	NO	15	G	<input type="button" value="Kommentar"/>
<input type="checkbox"/>	2	Introducer Peel away 14cm x 0.38mm	21901	GEM op/ane mtrlr SYD	SO	34	A	<input type="button" value="Kommentar"/>
<input type="checkbox"/>	1	Swan-lock red (till slutet system)	27298	Operation NK US	NS	56	F	<input type="button" value="Kommentar"/>

☐ Sortera artiklar på plockplats

Figur 15: Plocklista

I figur 15 kan man även se att det finns stöd för att lägga en kommentar på en artikel. Det är vanligt att en artikel är slut eller utbytt och man kan då lägga en kommentar på varför man inte kunde hämta den artikeln och även hur man har löst det istället. Ett exempel på kommentar visas i figur 16.

Operationsförberedelse

☒ 4-kärlsangiografi

Specialitet: Endovaskulär, intern
Linda-ID: 0

Kommentar för: Artärkanyl 45Mm. Steril

Finns på sal

Figur 16: Kommentar

Plocklistan kan under en pågående operation också redigeras. I figur 17 visas hur man i redigeringsläget kan lägga till nya artiklar. Man kan i detta läge också ändra antalet för en specifik artikel eller välja att ta bort artiklar.

5.3.9 Kartoteket

Kartoteket diskuteras i djup detalj nedan.

Namn	Artikelnummer	Klinik	Förråd	Sektion	Hylla	Fack	
Sax Häft 12Cm		GEM op/ane mtrrl NORD	NO	Kontor1	E	01:01	+
Sax Kontor 215Mm		GEM op/ane mtrrl NORD	NO	Kontor1	E	01:02	+
TRIAD "ligasure" diatermisax Monopolär diatermi		Operation Öron US	SS	50	G		+

Figur 17: Redigering av plocklista för operationsförberedelse

5.4 Utvecklingen

Här nedan beskrivs resultatet av hur utvecklingen inom gruppen har fungerat, och huruvida de olika hjälpmedlen har hjälpt eller stjälpt.

5.4.1 SCRUM

Utvecklingen i SCRUM-form har fungerat bra, dock har utvecklingen som beskriven i metod inte varit i rent SCRUM-format utan har modifierats för att bättre passa gruppen. Ett möte med handledare har hållits en gång i veckan där handledaren har haft möjlighet att ta upp saker som denne tyckt har behövts, sedan har gruppen hållit ett kort SCRUM-möte där man fått förklara vad man gjort förra veckan och vad man planerat att göra kommande vecka. Under mötets gång har man noterat om det är någonting som har behövt diskuteras vidare och detta har då noterats, varpå gruppen har gått igenom dessa frågor efteråt. Detta har fungerat väldigt bra då man på detta sättet inte har fått några större avbrott och alla hela tiden kunna hålla fokus. Efter SCRUM-mötena så har ordet varit fritt och alla har haft möjligheten att komma med synpunkter eller åsikter över någonting.

5.4.2 Alpha state cards

Korten har kontinuerligt uppdaterats av teamledare, dock har de kanske inte använts lika flitigt av gruppen som tanken var att de skulle?

5.4.3 Kundmöten

I inledningen hölls totalt fyra möten, det första var ett längre möte, där endast teamledaren, kundansvarige och arkitekten närvarade från projektgruppen. Detta möte hölls främst som en introduktion men var även ett tillfälle för kunden att visa upp sitt nuvarande system och få framföra idéer om hur system skulle kunna se ut och fungera istället. Sedan gjordes ett studiebesök hos kunden där alla gruppmedlemmar deltog. Man fick se hur en sjuksköterska förberedde inför en operation och även se deras nuvarande system i bruk. Under det tredje mötet

så utformade man kravspecifikationen som man sedan spikade under det fjärde mötet.

Efter varje iteration hölls ett möte på plats hos kunden för att diskutera hur arbetet fortskred, detta var även ett perfekt tillfälle att ta upp saker man var osäker kring, men var även bra för båda parter att föreslå ändringar i systemet. Mötet efter iteration två bestämdes att det skulle sättas upp en server där kunden skulle kunna testköra systemet, det bestämdes också att några från projektgruppen skulle närvara vid dessa tester. Detta gjordes två gånger där arkitekt och kundansvarige medverkade vid första tillfället, kvalitetsansvarige och dokumentansvarige medverkade vid det sista tillfället

5.4.4 Samarbete i gruppen

Samarbetet i gruppen har fungerat väldigt bra, kodstugorna som gruppen höll under förstudien var alla nöjda med, de som var mindre erfarna med webbprogrammering kände att de fick all den hjälp de behövde när de körde fast i sitt arbete och därför kändes det naturligt för hela gruppen att fortsätta med dessa även under utvecklingsfasen.

6 Diskussion

6.1 Resultat

6.1.1 Back-end

Vi valde att skriva back-end i javascript dels för att minska inlärningskurvan eftersom det är samma språk som används på front-end och dels för att realtidskommunikation mellan klienterna underlättas. Node.js valdes som plattform på back-end eftersom den har en väldigt enkel pakethanterare som underlättar när flera personer arbetar i ett projekt eftersom om en person lägger till en modul till projektet så behöver inte alla andra personer i projektet installera den modulen manuellt utan behöver endast köra ett enkelt kommando så installeras alla moduler som är tillagda till projektet.

KeystoneJS underlättade utvecklingen väldigt mycket till en början. Några punkter som beskriver vad KeystoneJS hjälper till med följer:

- Hjälper till att abstrahera systemet.
- 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 Handlebars⁴ och Less⁵.

6.1.2 Front-end

Anledningen till att vi använder oss av bootstrap på front-end är för att bootstrap har många färdiga CSS-klasser så man behöver inte skriva lika mycket CSS själv. De klasser som finns är lätta att använda för att skapa responsiva hemsidor. All kod är dessutom testad för att fungera på olika webbläsare vilket kan vara krångligt att lösa om man skriver all CSS från grunden.

JQuery används för att lättare hämta ut ett element på hemsidan och ändra data i det. Det finns även många bra jQuery-bibliotek att hämta som gör att man slipper "uppfinna hjulet" i många fall.

Fördelen med LESS är bland annat att man kan använda variabler och enkla funktioner i stilmallarna. Exempelvis kan variablerna användas för att spara de olika färgerna på hemsidan för att enkelt kunna byta ut dem.

⁴TODO: bättre referens. <http://handlebarsjs.com/> (2015-04-18)

⁵TODO: Bättre referens. <http://lesscss.org/> (2015-04-18)

6.1.3 Struktur

En nackdel med jQuery gentemot andra bibliotek som exempelvis angular⁶ är att det lätt blir spaghettikod⁷. För att strukturera så bra som möjligt är det viktigt att koden är bra uppdelad. Uppdelningen vi använder oss av fungerar ganska bra, men koden är ändå lite rörig.

6.1.4 Säkerhetskopiering och reservsystem

Det som kan göras för att få säkerhetskopieringen bättre är att implementera att en ny pdf-kopia skapas så fort en operationsförberedelse har uppdaterats.

Det krävs mycket arbete för att lägga in alla handböcker, vilket gör att det är viktigt att databasen säkerhetskopieras med jämna mellanrum. Men det tar en del datorkraft att göra en säkerhetskopiering. Därför är det bra att det går att ställa in tidsintervallet som bestämmer hur ofta säkerhetskopiering av databasen ska ske.

6.1.5 Översiktssida

Översiktssidan är tänkt att den ska kunna vara framme på en monitor och inte kräva någon interaktion för att se resultat för olika operationsförberedelser. Det är därför viktigt att den sidan använder realtidskommunikation, det är därför allt uppdateras vi Socket.IO till översiktssidan.

6.2 Metod

6.2.1 Kravinsamling

Metoden som användes för kravframställning 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 prioriterade 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 snabba 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. Vid användartesterna upplevdes inte detta som att det saknades vilket kan ses som att vi gjorde en bra avvägning. 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. Studiebesöket under förstudien var bra för att få en inblick i arbetet. Vi hade dock kunnat behöva mer tid för observation. Vid studiebesöket var hela gruppen med och det blev lite stressigt och mycket folk.

⁶TODO: Referens.

⁷TODO: Måste defineras?

Det kunde ha varit bättre att ha fler studiebesök uppdelade på färre personer för.

6.2.2 Utvecklingen

DET HÄR KÄNNES SOM ETT RESULTAT? Utvecklingen i SCRUM-form har fungerat bra, dock har utvecklingen som beskriven i metod inte varit i rent SCRUM-format utan har modifierats för att bättre passa gruppen. Ett möte med handledare har hållits en gång i veckan där handledaren har haft möjlighet att ta upp saker som denne tyckt har behövts, sedan har gruppen hållit ett kort SCRUM-möte där man fått förklara vad man gjort förra veckan och vad man planerat att göra kommande vecka. Under mötets gång har man noterat om det är någonting som har behövt diskuteras vidare och detta har då noterats, varpå gruppen har gått igenom dessa frågor efteråt. Detta har fungerat väldigt bra då man på detta sättet inte har fått några större avbrott och alla hela tiden kunna hålla fokus. Efter SCRUM-mötena så har ordet varit fritt och alla har haft möjligheten att komma med synpunkter eller åsikter. Samarbetet i gruppen har fungerat väldigt bra, kodstugorna som gruppen höll under förstudien var alla nöjda med, de som var mindre erfarna med webbprogrammering kände att de fick all den hjälp de behövde när de körde fast i sitt arbete och därför kändes det naturligt för hela gruppen att fortsätta med dessa även under utvecklingsfasen.

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.

8.2 Referenser

Referenser

[1] Får nog fixa en bättre referens här.

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

Förutom ett förbättrat kartotekssystem så är Region Östergötland också i behov av ett bättre system för att hantera deras lager på ett mer automatiserat sätt. Bland annat så skulle de behöva ett system som låter dem checka in vilka varor från lagret de hämtat ut, istället för att checka av manuellt, vilket kan vara felbenäget.

Vi valde dock att avgränsa oss från att bygga denna lösning, på grund av tidsbrist. Istället fokuserade vi på att förbättra kärnfunktionaliteten i applikationen.

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 figur 18). Dessa är för tillfället helt separata applikationer.

Artikel	Klinik	UC_ArtNr	Lev_ArtNr	Förråd	Sektion	Hylla	Fack
36 Ligacip extra M	Operation Öron US		LT200				
Abs Silikonförband 5X12,5Cm självhäft...	GEM op/ane mtrlf NORD	28414		NS	39	F	1
Abs silikonförband 5 x 12,5cm självhäft...	GEM op/ane mtrlf SYD	28414		SS	32	D	2
Abs Material - Strössel (T. Kemikaller)	GEM op/ane mtrlf NORD		41522	NO	B	B4	2
Abs Skydd Fixerförys	GEM op/ane mtrlf NORD	351916	Aercomfort	NO	B	B3	11
Abs Skydd För Hentar 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 CLIC absor...	GEM op/ane mtrlf SYD	80971		SO	45	D	

Figur 18: Gamla kartoteket

Så om en artikel har utgått 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 utgår 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 allt 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.

Den gamla lösningen byggde på MSSQL, medans vårt system använder MongoDB. Vi valde detta val både för att MongoDB är lättare att integrera med KeystoneJS, som vi använde för att strukturera vår Node.js kod, samt av prestandaanledningar[1].

A.3 Metod

Precis som resten av systemet så är kartoteket skrivet på webben, och därmed i Javascript, HTML och CSS. Vi har även använt Git för versionshantering.

Jag måste också nämna att trots att det är jag som skriver om kartoteket, så är det inte endast jag som har implementerat det. Koden och gränssnittet har givetvis varit ett stort grupparbete mellan alla medlemmar.

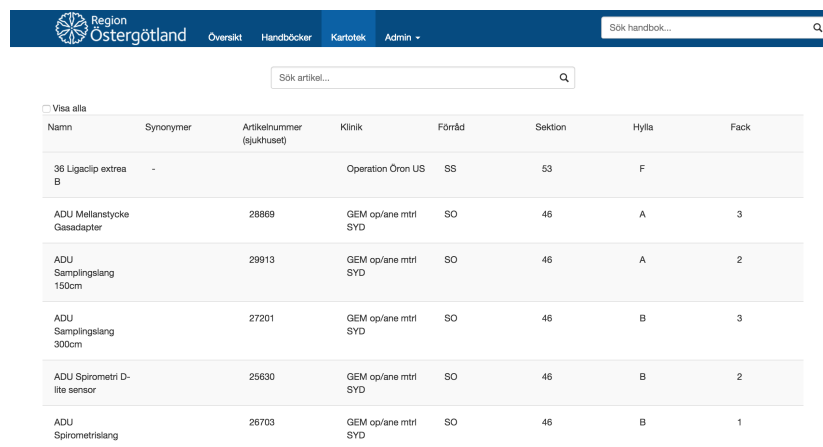
A.4 Resultat

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 resultatet kommer presenteras 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 19.



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 19: Kartoteket

Först 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 figur 19 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 till och modifiera information om artiklar, vilket är vad de två följande delarna handlar om.

A.4.2 Modifiering av artiklarna

För att modifiera artiklar så måste man som sagt vara administratör.

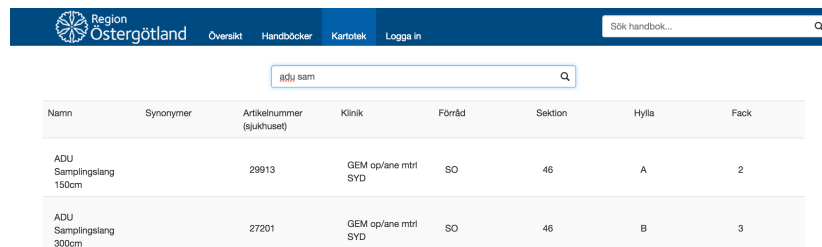
Med vårt gränssnitt så är det enkelt att modifiera information om en artikel. Om man vill ändra på, t.ex., artikelnamnet så är det bara att klicka på det! En input-ruta kommer då upp som låter dig ändra namnet till någonting mer passande. Detta är inspirerat av Trello, som vi använder för att organisera saker att göra. I Trello så finns en liknande funktionalitet för att ändra namn på olika "Boards".

För att ta bort artiklar så finns det ett smidig "X" till vänster om artikelraden.

Vi har valt att göra en del av den här informationen obligatorisk, eftersom sjukhuset hade problem i sitt tidigare system att vissa sjuksköterskor skippade att skriva in en del viktig information om artikeln.

A.4.3 Sökning av artiklarna

Sista kärnfunktionaliteten i kartoteket är sökning. Eftersom artiklarna är sorterade i bokstavsordning, och kartoteket har en "infinite scroll"-funktionalitet, så går det i teorin att hitta alla artiklar utan att söka. I våra diskussioner kunden på sjukhuset så verkar det också som det finns vissa personer som primärt använder sig av denna metod för att hitta artiklar. Däremot är detta inte en speciellt effektiv lösning, så därför har vi valt att implementera en sökfunktion.



Namn	Synonymer	Artikelnummer (sjukhuset)	Klinik	Förråd	Sektion	Hylla	Fack
ADU Samplingelang 150cm		29913	GEM op/ane mtrl SYD	SO	46	A	2
ADU Samplingelang 300cm		27201	GEM op/ane mtrl SYD	SO	46	B	3

Figur 20: Sökning i kartoteket.

Sökfunktionen är enkel. Längst upp på sidan finns det en liten sökruta. Användaren har möjlighet att söka på artikelnamn, artikelnummer eller synonymer till namnet. Se ett exempel på sökning av artikelnamn i figur 20. När man söker på någonting så byts hela tabellen ut mot sökresultatet. Hela upplevelsen är inspirerat av "Google Instant".

A.4.4 Integrering med handböckerna

Kartoteket står inte i isolation. En nyckelanledning till att det är så användbart är för att det är integrerat med handböckerna. Det leder oss till frågeställningen:

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

Svaret på detta anser jag vara: Ja! Och i detta avsnitt hade jag tänkt övertyga om denna slutsats.

Integreringen sker primärt på två ställen.

1. När man skapar en ny handbok.
2. När man förbereder en operation.

När man skapar en handbok så är kartoteket integrerat på det sättet att när man lägger till artiklar som måste hämtas från lagret så söker man i kartoteket för att hitta dessa. Detta är en förbättring av förra systemet då det förut inte fanns det ingen koppling mellan dessa system.

När man förbereder en operation så är kartoteket integrerat på så sätt att om en artikel slutar säljas, eller den av någon anledning tas bort från kartoteket, informerar vi användaren om att den har utgått.

Som vi har nämnt tidigare så fanns ingen av dessa av dessa funktionaliteter förut. Eftersom vi dessutom har lyckats reproducera all funktionalitet som fanns i det förra systemet så anser jag att frågan är klart och tydligt positivt bekräftad.

A.5 Diskussion

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

Svaret är otvivelaktigt: Ja! Det finns givetvis små buggar här och där. Ingen produkt blir någonsin 100% bugg-fri. Men i våra informella intervjuer under de två studiebesök vi gjorde hos kunden under iteration 3, där kunden har testat vårt system en hel dag, så verkade de enormt nöjda.

Om vi hade haft 300 timmar extra-tid att lägga på det här projektet så hade vi aboslut spenderat det på att integrera systemet med lagret samt med agresso, deras system som hanterar de ekonomiska delarna av artiklarna (för det mesta pris).

A.6 Slutsatser

Allt som allt så blev vi väldigt nöjda med kartoteket. Det finns såklart, som alltid, saker vi skulle vilja förbättra om vi hade mer tid, som integrering i lagret med t.ex. saldo. Men vi anser det som finns nu, kärnfunktionaliteten, ger en märkbar förbättring gentemot det nuvarande systemet.

A.7 Referenser

- [1] Parker, Zachary and Poe, Scott and Vrbsky, Susan V., "Comparing NoSQL MongoDB to an SQL DB" vid *Proceedings of the 51st ACM Southeast Conference*, New York, ACM, 2013, 5:1–5:6. Tillgänglig: <http://dl.acm.org/citation.cfm?id=2500047> [Hämtad 2015-05-11].

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 även valt att skriva det mesta själv och inte använt mig av så mycket ramverk och bibliotek. I detta projekt valde jag, som arkitekt, att byta ut PHP mot Node.js och att istället för ren HTML och CSS använda oss av Handlebars och Less. Dessutom har jag lagt mycket vikt på att använda många andra bibliotek och ramverk för att slippa uppfinna hjulet på nytt.

B.1.1 Syfte

Syftet med denna del av rapporten är att analysera om vad det finns för fördelar och nackdelar med att använda Node.js till ett sådant här projekt. Det ska även undersökas vad det finns för fördelar och nackdelar med att använda många externa ramverk och bibliotek.

B.1.2 Frågeställning

Frågeställningar

- Vad finns det för fördelar/nackdelar med att använda Node.js till detta projekt?
- Vad finns det för fördelar/nackdelar med att använda så många externa bibliotek som vi gjorde i detta projekt?

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 alla jämförelser av Node.js vara mot dessa och inga andra. Exempelvis vägs fördelar och nackdelar mot PHP och Python.

B.2 Bakgrund

I detta projekt har jag haft rollen som arkitekt och alltså tagit fram grundstenarna till programmet. Jag valde en arkitektur byggd på Node.js med många bibliotek och ramverk. Node.js valdes framförallt av två anledningar.

Först och främst så kom alla in i projektet med olika erfarenheter och kunskaper inom webbprogrammering och vissa hade aldrig tidigare skapat en hemsida. Trots detta ville vi snabbt komma igång och jag ville därför ta fram en arkitektur som var så lite nytt som möjligt. Eftersom man på ett eller annat sätt är tvungen att skriva Javascript när man programmerar till webben så tyckte jag det var bra att hålla sig till så få nya språk som möjligt.

Den andra anledning till att jag valde Node.js beror på kunden. Visserligen hade inte kunden några specifika krav på arkitekturen men däremot är vissa av kraven i kravspecifikationen olika lätta att implementera beroende på vilken plattform man väljer. Det fanns ett krav som gick ut på att flera personer skulle kunna kryssa av en lista samtidigt och alla som var inne på sidan skulle se uppdateringen i realtid. Detta kan vara svårt att lösa på många plattformar men blir väldigt enkelt med websockets som Node.js har bra stöd för. Detta till skillnad mot exempelvis PHP där websocket är krångligare att använda.

I och med att jag inte hade så mycket erfarenhet av Node.js och att använda många bibliotek sedan tidigare så var det svårt att veta om det skulle passa bra till detta projekt. Dessutom hade jag ingen tidigare erfarenhet av webbprogrammering i grupp. Under projektets gång har jag samlat på mig mer erfarenheter som underlättar för att göra en bedömning av detta arkitekturval.

B.3 Teori

B.3.1 PHP

PHP är ett programmeringsspråk som man kan skriva direkt i sin HTML-kod. Det skapades från början för att göra det enklare att lägga in funktionalitet på en hemsida. Du behövde inte längre skapa ett helt program i C, eller liknande, som i sin tur genererade HTML-kod för att exempelvis skapa en enkel gästbok. Nu kunde du enkelt lägga in PHP-kod direkt i HTML-koden som kördes i samband med att klienten frågar efter sidan. Det saknar även en standardiserad pakethanterare vilket gör att det är svårare att använda sig av många bibliotek.

B.3.2 Python

På senare tid har flera språk som inte är skapade för webben från början fått stöd för att göra hemsidor. Ett exempel är Python vilket alltså inte är byggt för webben till en början men ramverk som exempelvis Flask[2] gör det möjligt att starta en enkel webbserver som genererar HTML-kod. Till skillnad från PHP så separerar alltså Python logiken från själva HTML-koden och man skriver inte Python direkt i HTML-filerna. Programmeringsstilen är ganska lik Node.js och man använder sig ofta av liknande designmönster, men eftersom språket inte är skapat för webben från början kan det få problem med exempelvis prestanda när antalet användare ökar.

B.3.3 Node.js

Internet ändras hela tiden. Om man går tillbaka några år så bestod mest internet av statiska hemsidor med enkla gästböcker eller liknande. Idag består internet av sociala nätverk och sidor med dynamiskt innehåll som kan ha tusentals användare varje dag. Node.js är skapat med detta i åtanke. Det är inte skapat för att göra tunga beräkningar men däremot för att snabbt kunna distribuera information till flera användare samtidigt. En stor skillnad på internet

idag gentemot när PHP skapades är att när man har laddat klart en hemsida så kan fortfarande saker uppdateras. Exempelvis kan man på många sidor direkt se när en ny kommentar skickas in eller som i detta projekt så ska flera personer kunna se när man kryssar i en checkruta. Detta är fullt möjligt i både PHP och Python också men Node.js är skapat just för att lösa sådana problem utan att prestanda ska bli lidande.

B.4 Metod

I takt med att vi kom igång med projektet så fick vi i gruppen hela tiden nya erfarenheter med Node.js. Vi har upptäckt både fördelar och nackdelar med arkitekturen som jag inte hade tänkt på tidigare.

Utifrån vad jag har hört och sett från gruppen tillsammans med mina egna åsikter har jag tagit fram vad jag ser för fördelar och nackdelar med Node.js i detta projekt. Jag har även utifrån tidigare erfarenheter tagit med några potentiella fördelar och nackdelar som kan uppstå med denna arkitektur i längden.

Till sist har jag utifrån mina tidigare erfarenheter vägt fördelarna och nackdelarna gentemot hur det kan fungera i PHP och Python.

B.5 Resultat

Under denna rubrik presenteras resultatet av undersökningen.

B.5.1 Fördelar och nackdelar med Node.js

Följande ska skrivas om i flytande text! Fördelar:

- Många bibliotek med npm
- Enkelt att börja med
- Samma språk som på klienten
- Simpelt med realtidsuppdateringar (socket.io)

Nackdelar:

- Ibland svårt att strukturera koden
- Behövs inte speciellt tunga beräkningar för att prestandan ska bli lidande och man får därför lägga mycket logik på klienten. (exempelvis redigera-sidan)
- Inte skapat för relationsdatabaser vilket passar bättre till vissa av sakerna (kartoteket exempelvis)
- Non-blocking är svårt att greppa till en början

B.5.2 Fördelar och nackdelar med externa bibliotek

Följande ska skrivas om i flytande text! Fördelar:

- Ofta snygga och effektiva lösningar som hade tagit lång tid att skriva själv
- Går snabbt att nå resultat
-

Nackdelar:

- Konstiga buggar
- Förlitar sig på andra programmerare
- Svårt att sätta sig in i hur allt fungerar och känns ibland som saker sker av magi.
- Svårt att hålla reda på alla Licenser

B.6 Diskussion

B.6.1 Resultat

Ska skrivas:

- Varför väger fördelarna över nackdelarna
- Vad kan man göra åt nackdelarna?
- Var Node.js ett bra val?
- Var det ett bra val att använda sig av mycket bibliotek eller skulle man varit mer konservativ?
- Skulle man kunna gjort något annorlunda?

B.6.2 Metod

Ska skrivas:

- Kunde jag gjort något annorlunda?
- Hade jag gjort något annorlunda om det funnits mer tid?
- Mer research om andra liknande projekt?
- Skulle jag tagit mer hänsyn till kunden?

B.7 Slutsatser

Ska skrivas:

- Var Node.js ett bra val?
- Kunde PHP eller Python ha varit bättre alternativ?

B.8 Referenser

- [1] <http://php.net/manual/en/history.php.php>
Hämtad 2015-05-11.
- [2] <http://flask.pocoo.org/>
Hämtad 2015-05-11.
- [3] <http://radar.oreilly.com/2011/07/what-is-node.html>
Hämtad 2015-05-11.

C Pål Kastman

C.1 Inledning

I detta projekt har jag haft rollen som dokumentansvarig, en del av det arbete jag har utfört i denna roll har gått ut på att se till att dokumenten ser ordentliga ut, och att hitta bra lösningar för dokumentation.

I denna individuella del av rapporten så undersöker jag hur det är att arbeta med Latex i denna typ av projektform jämfört med andra ordbehandlingsprogram.

C.1.1 Syfte

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

C.1.2 Frågeställning

- Uppmuntrar Latex till en viss typ av samarbete jämfört med andra ordbehandlingsprogram?
- 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 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.

Ett alternativ som blir allt vanligare är Google Docs, vilket är ett webbaserat ordbehandlingsprogram som lanserades av Google 2006 [2] efter att man hade köpt upp företaget Upstartle [3]. En stor fördel med detta program är att det är gratis så länge man bara har ett konto hos Google.

Ett annat alternativ är Latex, vilket egentligen bara är 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

I denna del behandlas en del teori om Google docs och Latex.

C.3.1 Google docs

Google docs använder sig av molnet för att spara filer och man behöver därför inte oroa sig över säkerhetskopiering. Textformateringen görs främst genom att använda det grafiska gränssnittet, men kan även göras genom tangentbordskommandon. Det finns ett antal inbyggda funktioner för att t.ex. importera bilder eller skapa tabeller, det finns även en inbyggd funktion för att kommentera texten.

En stor fördel med Google docs är att dokumenten live-uppdateras hela tiden och att man på så sätt kan arbeta flera personer samtidigt i ett dokument.

C.3.2 Latex

Latex är i motsats till Google docs inget ordbehandlingsprogram, utan istället ett märkspråk liksom HTML. När man i 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 dokumenten 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

I detta projekt så har vi valt att inledningsvis skriva dokumentationen i Google Docs för att i ett senare skede, då det var dags att påbörja denna kandidatrapport gå över till att använda Latex. Vi valde att göra på detta sätt för att så

snabbt som möjligt komma igång då det är lätt för flera personer att arbeta samtidigt i samma dokument.

I denna rapport så skrev vi inledningsvis den gemensamma delen i samma fil, och de individuella delarna i separata filer som vi sedan importerade till den gemensamma. I ett senare skede lade vi även ut vissa delar av den gemensamma rapporten i separata filer. Alla Latexfiler har vi haft i samma repository som källkoden.

Varje dokument har en dokumenthistorik där tanken är att när man gör en ändring så ger man dokumentet en ny version och skriver vilka ändringar man gjort.

Vi använde IEEE-standarden för referenser.

C.5 Resultat

I denna del presenteras resultatet, det är givetvis inte bara resultatet av mitt eget arbete utan givetvis projektgruppens arbete som en helhet.

C.5.1 Dokumentstandard

När vi i projektets inledning använde Google docs så var det mycket svårare att hålla sig till någon sorts dokumentstandard, där man bl.a. använder samma typsnitt på rubriker och text. Även om alla projektmedlemmar försökte att hålla sig till den standard som vi satte upp så blev det ändå att någon del av dokumenten skiljde sig från resten när vi redigerade samtidigt i dem.

Detta fungerade bättre i Latex, men inte helt, då det även här uppstod problem med att olika personer gjorde på olika sätt.

C.5.2 Revisionshantering

Under projektets gång har vi försökt att hålla isär på olika revisioner av dokument så att man i ett senare skede kan gå tillbaka och se vilka ändringar som har gjorts mellan iterationerna. Detta har vi dock haft en del problem med i Google docs då dokumenten hela tiden sparas när man skriver och det inte på något smidigt sätt går att se historiken. Detta löste vi genom att göra en ny kopia på en fil när man ska skapa en ny revision.

I kandidatrapporten var inte detta ett lika stort problem då vi, som tidigare nämnt, använde oss utav revisionshanteringssystemet git. Detta gjorde att vi utan problem kunde gå tillbaka och se vad som hade ändrats i dokumentet. Dock har detta inte heller varit helt problemfritt då vi ibland fått mergekonflikter när vi laddat upp dokumenttext. Detta beror på att om flera personer ändrar på samma rader i ett dokument så kan git inte avgöra vilken text som ska sparas och då måste användarna lösa detta tillsammans, vilket inte alltid varit så enkelt om man sitter och skriver på olika ställen.

Ett annat problem har varit att om en gruppmedlem har lagt till en bild så måste denne komma ihåg att lägga till bilden i git så att även denna laddas

upp, det som har skett annars har varit att när någon annan har pullat koden så saknas bilden och det går inte att kompilera dokumentet.

C.5.3 Kommentering

Kommentering av varandras texter t.ex. vid korrekturläsning har skett smidigt i Google docs med den inbyggda kommenteringsfunktion. Den projektmedlem som varit ansvarig för texten har sedan fått välja mellan att lösa problemet eller att ge ett svar på kommentaren.

Kommentering av texter i Latex har vi gjort på github istället för i dokumenten.

C.5.4 Referenser

Referenser har varit lite av ett problem, eftersom vi hade bestämt att använda IEEE-standarden men det i Google docs inte finns något inbyggt system för detta. Detta gjorde att vi fick lägga till referenserna manuellt.

I Latex däremot så fanns det ett antal sätt att göra detta på, vilket istället ledde till att vi

C.6 Diskussion

Här diskuteras den metod vi har använt och det resultat vi har uppnått.

C.6.1 Resultat

Även om Latex är ett väldigt trevligt sätt att skriva dokument på så upplever jag att det ändå tar ett bra tag bara för att lära sig grunderna, man får dock helt klart en bra hjälp med att hålla samma dokumentstandard. Vi kunde dock ha fått ett ännu bättre resultat om vi hade gått igenom Latex mer noggrant inom gruppen och bestämt oss för vad man får och inte får göra, då det finns många sätt att lösa ett problem på.

Man kan ibland känna sig väldigt begränsad i Google docs, t.ex. så kan man inte på något sätt välja på vilken sida sidnumreringen ska börja. Dock tycker jag ändå att detta kan vara till en fördel, för om det endast finns en lösning till ett problem så slipper man leta efter det bästa sättet, vilket ibland blir fallet i Latex.

Kommentering av dokument är någonting som har varit svårt att lösa, kommenteringsfunktionen i Google docs fungerar jättebra så länge man vet vad man ska kommentera på, alltså så länge man kommenterar det man själv skriver så upplever jag inte några problem, men om man ska korrekturläsa någon annans dokument så är det väldigt svårt att veta exakt vad som blivit tillagt sedan sist i dokumentet, särskilt om revisionshanteringen inte sköts på rätt sätt.

C.6.2 Metod

Kandidatrapporten hade kunnat strukturerats upp mer till en början, om vi hade gjort det så tror jag att vi hade fått mindre komplikationer med mergekonflikter, dock så följer det rätt så naturligt att vi då, hade behövt ha en bättre strukturering på vem som skriver vad, men då flera personer har jobbat på samma delar inom projektet så kommer man i slutändan även att behöva skriva på samma delar och då kvarstår ändå detta problem.

Att inleda med att skriva dokument i Google docs tycker jag var en bra idé då man samtidigt kunde börja med Latex i ett lägre tempo och på så sätt ha mer tid till att fråga andra när man stöter på problem, ett annat alternativ hade kunnat vara att använda något av webbalternativen för Latex, nackdelen med detta är att alla hade blivit tvungna att skaffa sig ett konto där.

C.7 Slutsatser

Även om Latex hjälper till att hålla en standard så bör man ändå bestämma internt i projektgruppen vilka olika metoder man ska använda för att t.ex. infoga referenser då det finns flera sätt att lösa samma saker på.

Det kan vara en nackdel för flera gruppmedlemmar att arbete samtidigt i samma delar, det spelar inte så stor roll om man delar upp dokumenten då delarna ändå kommer att ligga i endast en fil. Men det går dock ändå att lösa genom att kommunicera med de andra som behöver skriva på samma delar.

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 kravinsamlingsmetoder och verktyg som använts för att ta fram krav. Erfarenheterna används för att undersöka om metoderna och verktygen kan användas för att tillfredsställa 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 och gör inga jämförelser med andra kravinsamlingsmetoder. 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 ideér och designers 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 få information om det nuvarande arbetet inom området och problem relaterade till det. Det är också bra för att få fram de stora målen med ett projekt. Många ser det som den huvudsakliga insamlingstekniken. De ger mycket information men för att lösa kritiska problem behövs ofta andra tekniker för att komplettera intervjuer. [2]

D.3.3 Observationer

Observationer är ett bra verktyg för att få information om nuvarande system eller arbetssätt. Det är ett bra sätt att komplettera intervjuer då användare ofta har svårt att förklara vad de verkligen gör. [2]

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 21. Det första tillfället fokuserade på gränssnittsdesign



Figur 21: LoFi-prototyper

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 redigeringsvyn 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.

D.4.4 HiFi-prototyp

Själva 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.

D.4.5 Användartest

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 sätt, 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.

D.5 Resultat

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 sätt kompletterade dessa metoder varandra och det kändes nödvändigt för en lyckad kravinsamling.

LoFi-Prototyper kunde användas i förstudien för att bekräfta vår 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. Intervjuer och observationer kunde användas som komplement till varandra på ett bra sätt. Det kan diskuteras om dessa metoder är tillräckliga för en kravinsamling. För ett projekt i en storlek som vårt kan de vara tillräckliga. Prototyper användes i vårt projekt som ett verktyg för att samla in krav. Andra verktyg kunde dock ha använts för att ytterligare komplettera dessa metoder. Vi la inte så stort fokus på systemets olika roller i projektet utan nöjde oss med två stycken. För att analysera de olika rollerna kunde verktyg som user-stories ha använts. Prototyper kan se olika ut och användas på många olika sätt. I vårt projekt visades att de var bra för att samla in krav och snabbt reda ut otydligheter.

D.6.2 Metod

Metoden som användes för att svara på frågeställningarna var att utföra själva projektet och utgå ifrån dessa erfarenheter. Projektet gav således svar på att intervjuer, observationer och prototyper gick att använda och var bra för just detta projekt. Metoden har inte gjort några jämförelser till andra kravinsamlingsmetoder vilket skulle kunna vara intressant.

D.7 Slutsatser

Intervjuer och observationer är bra grundmetoder för att analysera kundens behov. I detta projekt kändes de som naturliga och nödvändiga tillvägagångssätt. De kan med fördel kompletteras med olika verktyg och tekniker såsom prototyper. 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 av webbapplikationer. - 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 webbaserat 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 webbaserade tjänster för att utföra kontinuerliga automatiserade tester av webbapplikationer.

E.1.2 Frågeställning

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

- Hur kan man använda webbaserade tjänster för att utföra kontinuerliga automatiserade tester av webbapplikationer?
- Hur effektivt är det att använda en webbaserad 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 läggs upp på 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 webbapplikation 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 webbaserad tjänst för att köra automatiserade enhetstester och integrationstester på projekt som finns på GitHub. Travis CI är gratis att använda och byggt på öppen källkod som är tillgänglig under en MIT-licens. 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 Javascriptbibliotek 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 webbserverar. Node.js är baserat på öppen källkod som är tillgänglig under en MIT-licens. 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`. Jasmine är baserat på öppen källkod som är tillgänglig under en MIT-licens.

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, vilket 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.io.

E.4 Metod

Här beskrivs den metod som användes för att besvara frågeställningarna.

Arbetet inleddes med att en teoriundersökning genomfördes angående vilka metoder, ramverk och tjänster som skulle kunna användas för att genomföra testerna i projektet.

De ramverk för testning som undersöktes var Jasmine, Qunit och Mocha. Det ramverk som valdes för arbetet med projektet var Jasmine eftersom det är enkelt att konfigurera och har ett tydligt och intuitivt syntax. Att Jasmine var enkelt och tydligt var viktigt eftersom det var många andra uppgifter förutom testning som skulle utföras i projektet och tiden för arbetet med projektet var begränsad till 300 timmar. Det var alltså ingen i projektgruppen som hade tid att lära sig en komplicerad syntax i ett ramverk även om det varit mer kraftfullt.

De tjänster för automatiserade tester som undersöktes var Travis CI, Codeship och Drone.io. Även här så gjordes valet beroende på hur enkla tjänsterna var att konfigurera och använda. Den tjänst som verkade enklast och således valdes var Travis CI.

Arbetet med Travis CI inleddes med att tjänsten kopplades till projektets repository på GitHub. Kopplingen utfördes genom att administratören för repositoryn

loggade in på travis-ci.org med sitt GitHubkonto 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.tmpl'` 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 utvalda 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 webbaserad 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 *Tabell E.5.1* 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 webbaserad 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.

Resultatet av testerna som utfördes för att besvara den andra frågeställningen visas i *Tabell E.5.1*.

Tabell E.5.1

Testnummer	Datum (ÅÅ-MM-DD)	Tid (hh-mm-ss)	Tester 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

Medelvärdet avrundat till närmsta heltal är: 10556 tester per sekund.

Standardavvikelsen avrundat till närmsta heltal är: 1710.

Den tid det tar från det att koden läggs upp på GitHub till dess att testkoden börjar köras är i genomsnitt ca en minut. Sedan tar det även ca 30 sekunder för att alla node-moduler att installeras innan själva testfallen börjar köras.

Undersökningen om vad om är svårt att testa ledde till följande resultat. Det visade sig att vanliga Javascriptfunktioner är lätta att testa så länge de ligger utanför jQueryfunktionen `$(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 Javascriptfunktioner utanför jQuery-funktionen `$(document).ready()` eftersom koden då blir lättare att testa.

JQueryfunktioner är i allmänhet svårare att testa än vanliga Javascriptfunktioner. Anledningen är att jQueryfunktioner startas av och manipulerar HTML-objekt. Ett sätt att lösa detta är att använda en egen `_Specrunner.html`. Då kan HTML-objekten i den filen användas för att testa jQueryfunktionerna.

E.6 Diskussion

Under den här rubriken diskuteras rapportens resultat och metod.

E.6.1 Resultat

Resultatet angående hur effektivt det var att använda Travis CI var inte särskilt förvånande. Att den lilla testfunktionen kunde köras ca 10^4 gånger per sekund verkar rimligt. Det intressanta med resultatet var att det kan dröja ca en minut från att koden läggs upp på GitHub till dess att testfallen börjar köras och att effektiviteten kan variera med ca $\pm 30\%$ vid olika tidpunkter.

Det är även värt att påpeka att hur koden skrivs och struktureras i väldigt hög grad påverkar hur lätt den blir att testa. I alla fall med den metod som använts under arbetet med den här rapporten.

E.6.2 Metod

Den metod för testning som användes under arbetet med den här rapporten var medvetet väldigt enkel och inte särskilt kraftfull. Anledningen var att det var mycket annat som skulle göras i projektet och tiden var begränsad till 300 timmar. I efterhand visade det sig dock att det hade varit möjligt att lägga lite mer tid på att använda en lite mer avancerad metod för att kunna testa en större del av koden i projektet. Nu användes den överblivna tiden istället till att göra fler manuella tester av koden.

Det är även värt att nämna ett en nackdel med metoden som använts för tester under projektet är att om ett eventuellt fel upptäcks så är koden redan upplagd på GitHub. Det hade varit bättre om koden kan testas innan den lades upp på GitHub.

E.7 Slutsatser

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

E.7.1 Hur kan man använda webbaserade tjänster för att utföra kontinuerliga automatiserade tester av webbapplikationer?

Ett sätt att använda en webbaserad tjänst för att utföra kontinuerliga automatiserade tester av en webbapplikation är att använda Travis CI tillsammans med Jasmine på det sätt som beskrivits under avsnittet *Metod*. Observera att den webbapplikation som testades var skriven med Javascript. Node.js användes på serversidan och jQuery användes på klientsidan.

E.7.2 Hur effektivt är det att använda en webbaserad tjänst för automatiserade tester?

Det enkla testfallet som användes kunde köras ca 10^4 gånger per sekund och det tar ca 60 sekunder från att koden läggs upp på GitHub till det att den börjar köras på servern.

Om man använder ett antal testfall (med liknande komplexitet som det som användes i den här rapporten) i storleksordningen $60 \cdot 10^4$ eller mindre så kommer den största väntetiden vara tiden från att koden läggs upp på GitHub till det att den börjar köras på servern. Själva testfallen kommer att köras på mindre än en minut.

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

Kortfattat kan man säga att vanliga Javascriptfunktioner är väldigt enkla att testa med de metoder, ramverk och tjänster som använts under arbetet med den här rapporten. Men JQueryfunktioner och Javascriptfunktioner i JQueryfunktioner är svårare att testa. För en mer detaljerad beskrivning se rubriken *Resultat*.

E.7.4 Framtida arbete inom området

Det finns mycket arbete som skulle kunna utföras inom området som behandlats i den här rapporten.

Det vore intressant att undersöka hur stor andel av de totala felen som upptäcks med den typ av testning som har beskrivits i den här rapporten. Det kräver dock mer tid och resurser än vad som var tillgängligt under arbetet den här rapporten.

Det vore även intressant att undersöka vilka nackdelar och fördelar det finns med webbaserade testmiljöer jämfört med lokala testmiljöer.

En annan undersökning som skulle kunna utföras är att jämföra flera olika ramverk för testning av Javascript. Till exempel skulle man kunna jämföra Jasmine och Mocha. Man skulle även kunna jämföra flera olika webbaserade tjänster för

automatiserade tester. Till exempel skulle man kunna jämföra Travis CI med Codeship och Drone.io.

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.

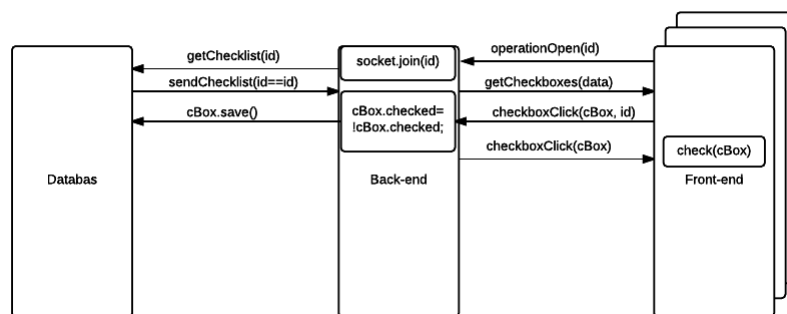
F.3 Metod

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 med mycket mellanrum och upplevs därav inte som 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.

Att jag började med att implementera plocklistan och inte förän den var klar implementera förberedelselistan tror jag också var en bra idé. Jag kunde då till en början fokusera på en typ av checklista och se till att få den att fungera bra. Sedan när förberedelselistan skulle implementeras så kunde jag återanvända väldigt mycket av den kod som jag skrivit till plocklistan.

F.6 Slutsatser

All nödvändig information i plocklistorna fick plats med bra marginal, därav anser jag att svaret på min första frågeställning är: Ja!

Jag har lyckats implementera checkning av både plocklistor och förberedelselistor med ungefär samma kod. Checkningen uppdateras för alla klienter som är in-
ne i samma operationsförberedelse. Det är nästan en sekunds fördröjning när
en person checkar den första checkrutan i en checklista tills dess att de andra
anslutna klienterna får den checkrutan checkad. Efter det att den första rutan
är checkad så uppdateras checkningar utan märkbar fördröjning till de andra
anslutna klienterna. Därav anser jag att svaret på min andra frågeställning är:
Ja!

Eftersom kunden har prövat checkningen av cheklistor med flera personer sam-
tidigt och de tyckte att den fungerade bra som helhet så anser jag att syftet
med denna del av projektet är upplevt.

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

- Hur kan man ersätta KeystoneJS då varken administratörgränssnittet eller databaskopplingen används?
- Vad behöver ändras för att byta ut databasen från MongoDB till MS SQL?

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. Rapporten förutsätter också vissa kunskaper inom Node.js så som att komma igång med en enkel applikation. Den behandlar inte heller alla delar av ersättning av KeystoneJS utan fokuserar främst på hur routern-delen och databaskopplingen ska ersättas.

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 MS SQL då Region Östergötland inte tillåter MongoDB på sina servrar. Då utvecklingen av applikationen har fortgått har KeystoneJS fått en mindre och mindre roll i applikationen. En av KeystoneJS 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 jobbar 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 KeystoneJS, vilket var den enkla integrationen med MongoDB genom Mongoose. Detta ledde till tankar om att KeystoneJS kanske skulle kunna bytas ut mot egenskriven kod eller mindre och mer lättförståeliga moduler utan jättemycket arbete.

En beskrivning och förklaring för många av modulerna som kommer tas upp finns att läsa i avsnitt 2.1 i den gemensamma delen.

G.2.1 MS SQL

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

G.2.2 Npm

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

G.2.3 Express

Express är ett ramverk för Node.js som gör det lättare att bygga webbapplikationer.

G.2.4 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.2.5 edge-sql

Edge-sql är en modul som gör det möjligt att exekvera T-SQL kommandon direkt i Node.js med hjälp av Edge.js. Den använder asynkront ADO.NET, som är ett bibliotek från Microsoft för att få tillgång till data och dataservice, för att få tillgång till MS SQL.

G.3 Teori

För en oerfaren utvecklare kan ett projekt från början se mycket svårt ut att genomföra. Då kan färdiga ramverk göra att utvecklingen går snabbare och lättare. Men efter att utvecklaren har jobbat ett tag och fått nya kunskaper kan denne inse att projektet kanske inte alls är så svårt och ett ramverk bara har gjort applikationen mer komplicerad och svårt att förstå. Då kan det vara intressant att se hur applikationen skulle kunna skrivas om utan dessa ramverk.

MongoDB är ett relativt nytt databas-system som klassificeras som en NoSQL-databas. MongoDB används av en hel del företag [3], men de flesta använder fortfarande någon typ av SQL-databas [4]. Att ha flera olika typer av databaser är uteslutet för många företag då det leder till större underhållskostnader och kräver kunskap om flera system. Därför är det intressant att se hur MongoDB kan bytas ut mot en SQL-databas, i detta fallet MS SQL.

G.4 Metod

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

För att utvärdera moduler för MS SQL och för att vad som krävs för att få applikationen att funka med MS SQL 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 MS SQL och testas. Om inte modulen uppfyller kraven på enkelhet och stabilitet så kommer ett annat alternativ tas fram och testas.

G.5 Resultat

En av modulerna som används av KeystoneJS heter Express och används av KeystoneJS för att hantera router-delen. Denna modulen kan också användas för att hantera hantera alla förfrågningar [1] och rendera sidor utan KeystoneJS. För att använda Handlebars i Express så behövs en modul som heter express-handlebars. Den kan sättas som renderingsmotor för Express genom koden nedan. *DefaultLayout* sätts till en mall som bestämmer hur utformningen av sidorna ska se ut. I just applikationen som det skrivs om i detta fallet heter den *default.hbs*.

```
var express = require('express');
var hbsExpre = require('express-handlebars');
var app = express();
app.engine('hbs', hbsExpress({defaultLayout: 'default'}));
app.set('view engine', 'hbs');
```

Om någon middleware behöver användas kan den läggas in i Express enligt exemplet nedan. Alla förfrågningar som går genom '/' kommer då köras igenom funktionen middleware. Om man bara vill att en funktion ska köras för en sida så lägger man helt enkelt in sökvägen för den sidan istället för '/'.

```
var router = express.Router();
router.use('/', middleware);
```

En middleware som behövs för denna applikationen är less-middleware. Den används för att kompilera om less-filer till CSS-filer. Då en förfrågning efter en CSS-fil inkommer så kollar den först upp om filen existerar, om så är fallet skickas den ut. Annars så letar modulen upp en less-fil med samma namn och kompilerar den till en CSS-fil för att sedan skicka ut den.

Alla rutter kan läggas i en fil som sedan exporteras för att användas av Express enligt nedan.

```
router.get('/', function(req, res) {
```

```
res.render('overview', data);
});
```

Detta gör att sidan overview renderas med datan data genom Handlebars. Istället för att ha en anonym funktion så kan man kalla på en namngiven funktion som ligger i annan fil, precis som i nuvarande applikation för att låta den funktionen hämta ut all information för att sedan kalla på *res.render*. Handlebars-filerna som finns i nuvarande applikation kommer i så fall inte behöva ändras.

```
router.get('/', overview);
router.get('/info/:slug', info);
...
var overview = function() {
  data = getData();
  res.render('overview', data);
};
```

Det finns också några api:er som måste skrivas om. I nuvarande applikation så används en modul som heter keystone-rest och används för att skapa ett REST-api för alla databasmodeller. När inte KeystoneJS används längre så kan denna modulen inte heller användas. Alltså måste ett eget REST-api skrivas, vilket ett enkelt exempel för kan ses nedan.

```
router.route('/api/:slug').post(function(req, res) {
  // Gör post för slug
}).get(function(req, res) {
  // Gör get för slug
}).put(function(req, res) {
  // Gör put för slug
}).delete(function(req, res) {
  // Gör delete för slug
});
```

Det sista som behöver göras är att sätta så applikationen använder *router*.

```
app.use('/', router);
```

Detta leder fram till nästa frågeställning om hur datan till renderingen ska tas fram då databasen är utbytt till MS SQL istället för MongoDB.

Det finns flera olika moduler för att koppla ihop en Node.js-applikation med MS SQL. En av dessa är en officiell modul från Microsoft [2]. Denna har dock inte släppts i slutgiltig version utan finns bara som en förhandstitt, vilken släpptes den första augusti 2013 och har enligt Github inte blivit uppdaterad sedan dess.

En annan modul är Edge.js som kan användas för att köra .NET kod direkt i en Node.js-applikation och på så sätt hämta data från en MS SQL databas. Det finns en modul till Node.js som heter edge-sql som just är till för att hämta data från en MS SQL databas. Denna modulen gör att man direkt kan skriva SQL-kod i Javascript-koden. En exempel på hur en funktion, för att hämta en operation, kan se ut och vad den skulle ge för utdata kan ses nedan. För att

koden ska fungera måste och en omgivningsvariabel vara satt till en anslutningssträng för att kunna ansluta till databasen eller så kan man direkt lägga in anslutningssträngen då man definierar funktionen.

```
var edge = require('edge');
// Om anslutningssträng är satt som omgivningsvariabel.
var getOperation = edge.func('sql', function() {/*
    SELECT * FROM operations
    WHERE title LIKE '%@title%'
*/});

// Utan anslutningssträng som omgivningvariabel.
var getOperation = edge.func('sql', {
source: function() {/*
    SELECT * FROM operations
    WHERE title LIKE '%@title%'
*/}, connectionString: 'Data Source=.....;'});

getOperation({ title: 'Kärl' }, function (error, result) {
    if (error) throw error;
    console.log(result);
});

// Resultat
[{title: '4-kärlsangiografi',
linda_id: '0',
tage: '',
state: 'Publicerad',
specialty: 1,
template: 1,
isDone: 0,
lastPrinted: 2015,
version: 1.0,
lastUpdated: 2015}]
```

G.6 Diskussion

Här diskuteras alternativ till resultat och metod och varför just dessa valdes.

G.6.1 Resultat

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 en sådan inte har släppts officiellt än och den version som finns tillgänglig inte är färdigutvecklad eller aktiv [5] och innehåller kända buggar [6] så kan inte dess funktionalitet garanteras.

Att använda Edge.js och edge-sql istället för någon annan modul gör att det inte finns några begränsningar i hur hämtningen från databasen ska gå till. Denna lösningen gör så att den bara skickar vidare förfrågan till ADO.NET vilket gör att just den delen av lösningen är officiell och garanterad att fungera. Här ligger ju istället osäkerheten i att det skulle vara något fel på Edge.js och edge-sql. Detta känns som en mindre risk än att en modul som är skriven direkt för koppling med MS SQL skulle ha buggar. Genom denna lösningen blir det också mycket enkelt för utvecklare att komma igång om de redan har kunskap inom SQL, vilket man kan förutsätta om de har fått i uppgift att göra en sådan implementation. Med Edge.js finns det också andra språk som databasfunktioner kan skrivas i som till exempel C#.

G.6.2 Metod

Metoden som användes innebar att mycket information lästes i artiklar. En del tester gjordes, men inte riktigt i den omfattning som vore önskvärt. Med fler tester så hade resultatet kunnat verifierats bättre. Anledningen till att det inte gjordes mer tester var tidsbrist.

G.7 Slutsatser

Eftersom hela router-systemet i KeystoneJS bygger på Express kan den enkelt bytas ut mot bara Express. Edge.js tillsammans med edge-sql gör det väldigt lätt att få till en databaskoppling mot en MS SQL databas.

G.8 Referenser

- [1] Express 4.x api [Online] Tillgänglig: <http://expressjs.com/4x/api.html> [Hämtad 2015-05-11]
- [2] Microsoft Driver for Node.js for SQL Server Preview [Online] Tillgänglig: <https://www.microsoft.com/en-us/download/details.aspx?id=29995> [Hämtad 2015-05-11]
- [3] Who uses MongoDB? [Online] Tillgänglig: <http://www.mongodb.com/who-uses-mongodb> [Hämtad 2015-05-11]
- [4] DB-engines Ranking [Online] Tillgänglig: <http://db-engines.com/en/ranking> [Hämtad 2015-05-11]
- [5] Github för node-sqlserver [Online] Tillgänglig: <https://github.com/Azure/node-sqlserver> [Hämtad 2015-05-11]
- [6] Github issues för node-sqlserver [Online] Tillgänglig: <https://github.com/Azure/node-sqlserver/issues> [Hämtad 2015-05-11]