CHALMERS

OpenBooking

Ett verksamhetsanpassat bokningssystem i öppen källkod

Examensarbete inom Högskoleingenjörsprogrammet i Datateknik

HANNES ELVEMYR

Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2015

OpenBooking
Ett verksamhetsanpassat bokningssystem i öppen källkod

Hannes Elvemyr

Institutionen för data- och informationsteknik
Chalmers tekniska högskola
412 96 Göteborg
Tel: 031-772 1000
Fax: 031-772 3663

Figurer och diagram i rapporten är gjorda av Hannes Elvemyr om inget annat anges.

Institutionen för data- och informationsteknik
Göteborg, 2015

SAMMANFATTNING

Tillsammans med Mullsjö folkhögskola har en enkel men fungerande webbapplikation för bokning av arrangemang i skolans kurs- och konferensverksamhet utvecklats. Nuvarande lösning för att hantera dessa bokningar är otillräcklig och hela bokningsprocessen behöver formaliseras samt effektiviseras. Det nya bokningssystemet ska förbättra hanteringen av bokningar för personalen, förhindra dubbelbokningar och även göra bokningsinformationen mer tillgänglig. Arbetet har genomförts med en agil utvecklingsmetod, närmare bestämt testdriven utveckling (TDD). För implementationen har Ruby on Rails använts, ett ramverk för webbutveckling i programmeringsspråket Ruby. Detta ramverk valdes efter en mindre jämförelse mellan det tidigare nämnda och Django, skrivet i språket Python.

FÖRORD

Föreliggande rapport beskriver ett examensarbete i datateknik på 15 högskolepoäng utfört av Hannes Elvemyr på Chalmers tekniska högskola under år 2014 - 2015.

Jag vill rikta ett stort tack till Uno Holmer för god handledning. Jag vill också rikta ett tack till Mullsjö Folkhögskola för att de vågade satsa på projektet trots att det var oklart vilket resultat som kunde förväntas. Till sist, speciellt tack till Hans Noreliusson som har varit kontaktperson på folkhögskolan och även Anna Sandin som tillsammans med Hans varit med under utvecklingen och delat med sig av sina erfarenheter och kunskaper.

INNEHÅLLSFÖRTECKNING

FÖRKORTNINGAR OCH FÖRKLARINGAR

API	Application programming interface
Django	Ramverk skrivet i Python för webbutveckling
ORM	Object-relation Mapping
Python	Programmeringsspråk
Ruby	Programmeringsspråk
Rails	Ramverket Ruby on Rails för webbutveckling i Ruby
TDD	Den agila metoden Test-Driven Development

Begrepp rörande kurs- och konferensverksamheten på Mullsjö folkhögskola

Arrangemang	En kortkurs, konferens eller övernattning på vandrarhemmet.
Bokning	En bokning av ett arrangemang i kurs- och konferensverksamhet.
Resurs	Något som kan reserveras för en bokning i bokningssystemet, exempelvis ett boenderum eller ett konferensrum.

1 INTRODUKTION

Detta kapitel ger läsaren en introduktion till rapporten genom att förklara bakgrunden till projektet och samarbetet med Mullsjö folkhögskola. Därefter följer syfte, mål och avgränsningar. Sist i kapitlet ges en kort rapportöversikt för vidare läsning.

1.1 Bakgrund

Mullsjö folkhögskola är belägen utanför Jönköping och har förutom ordinarie undervisning för sina elever även en kurs- och konferensverksamhet. Denna verksamhet har växt under åren och omfattar nu följande arrangemang: kortkurser, konferenser och vandrarhem. Nuvarande lösning för att hantera bokningar av dessa arrangemang är inte längre anpassad för verksamhetens storlek. Kurs- och konferensansvariga på folkhögskolan upplever att processen är ineffektiv och att risken för felbokningar är överhängande, särskilt under intensiva perioder. Nuvarande lösning kan låsa sig då flera användare är aktiva samtidigt och det finns ingen automatisk kontroll för att förhindra dubbelbokningar. Dessutom är det önskvärt att bokningsinformationen för personalen.

1.2 Syfte

Ett syfte med projektet är att bidra till folkhögskolas arbete med att effektivisera hanteringen av arrangemangsbokningar samt finna lösningar på problem med nuvarande arbetssätt. Detta för att underlätta för kurs- och konferensansvariga.

Ytterligare ett syfte är det egna lärandet i mjukvaruutveckling samt att få en verklighetsbaserad erfarenhet av att arbeta med agila utvecklingsmetoder generellt och testdrivet specifikt.

1.3 Mål

Projektets mål är att utveckla en fungerande och stabil grund för ett bokningssystem för ovan nämnd kurs- och konferensverksamhet. Systemet ska vara anpassat för verksamhetens behov och från en kravspecifikation ska de mest fundamentala delarna som behövs för att göra arrangemangsbokningar implementeras. Det ska designas för att göra bokningsprocessen mer effektiv, underlätta för personalen, förhindra dubbelbokningar och stödja samtidig användning av flera användare. Dessutom ska bokningsinformationen göras mer lättillgänglig för personalen.

Ett mer generellt mål är att besvara frågan om det är möjligt att med relativt små resurser, såsom i ett examensarbete som detta, effektivisera en verksamhet i en medelstor organisation likt Mullsjö folkhögskola med hjälp av moderna metoder och verktyg.

1.4 Avgränsning

Även om projektet kommer att innefatta en mindre jämförelse av två olika ramverk för webbutveckling kommer inte tyngdpunkten att ligga här. Istället kommer fokus vara på att faktiskt försöka utveckla ett användbart system med hjälp av något av dessa ramverk.

Systemet förväntas inte vara färdigutvecklat i slutet av projektet. Det ska snarare ses som att grundläggande funktioner för ett sådant system ska vara implementerade. Dock ska dessa funktioner vara testade och väl fungerande och en utbyggnad av systemet ska vara möjlig och eventuellt redan påbörjad.

Ytterligare en avgränsning avser systemmiljön på folkhögskolan. Projektet kommer att fokusera kring bokningssystemets koppling mellan folkhögskolans schemaläggningsprogram och elevregister. Kopplingen till folkhögskolans kalendarium och ekonomisystem kommer inte att diskuteras i denna rapport.

1.5 Rapportöversikt

Kapitel 4 till 6 är främst riktade till den teknikintresserade läsaren. De som är intresserade av agil och testdriven utveckling kan koncentrera sig på kapitel 3, 6 och 8. Är man mest intresserad av att se resultatet ger kapitel 6 en rik beskrivning med mycket bilder. För de som vill läsa om hur systemet påverkar verksamheten är kapitel 2 samt kapitel 6 och 8 att föredra.

2 VERKSAMHETSBESKRIVNING

Det mjukvarusystem som projektet förväntas resultera i är tänkt att så småningom användas i kurs- och konferensverksamheten på Mullsjö folkhögskola. För att förstå hur systemet behöver fungera ges här en kort beskrivning av denna verksamhet. Därefter följer en problembeskrivning av befintlig lösning för att förstå de krav som finns på det nya systemet.

2.1 Kurs- och konferensverksamheten

På Mullsjö folkhögskola innefattar kurs- och konferensverksamheten tre olika delar: kortkurser, konferenser och vandrarhem. *Kortkurser* är kurser som skolan själv bedriver, exempelvis sommarkurser i estetisk verksamhet eller veckokurser för personer med dövblindhet. *Konferenser* är sammankomster där företag eller organisationer hyr skolans lokaler för eget bruk. Skolan erbjuder då även mat och övernattningsrum. Folkhögskolan hyr också ut övernattningslokaler i form av ett *vandrarhem*.

Verksamheten kräver administrativt arbete där information är ett mycket centralt begrepp och de olika arrangemangen har specifika behov.

> För *kortkurser* behövs alltid deltagarnas personuppgifter. För undervisningen behöver lokaler reserveras och i regel ingår logi för alla deltagare och lärare. Ekonomin är enskild för varje deltagare.
>
> Gällande *konferenser* handlar det om lokalreservationer, extra utrustning med möblemang och logistik, offerter, orderbekräftelser och ekonomihantering.
>
> I fallet *vandrarhem* gäller reservation av övernattningsrum för gästerna, uppgifter till gästerna rörande folkhögskolan, bokningsbekräftelser och även ekonomihantering.

Om mat ingår behövs uppgifter om vilka måltider som ska serveras, mattider och specialkost. Vid övernattning behövs uppgifter om den boende och övernattningsrum ska reserveras.

För att hantera informationsflödet används i nuläget kalkylblad och textdokument.

> *Kalkylblad:* Kalendarium, deltagarlistor med ekonomiuppgifter, bokningar som innehåller offertunderlag, korrespondens med mera.
>
> *Textdokument:* Offerter och bokningsbekräftelser.

Dessa kalkylblad och dokument behöver synkroniseras med följande övriga system på folkhögskolan:

> *Folkhögskolans kalendarium:* Större kalendarium för hela folkhögskolan som behöver synkroniseras med kurs- och konferensverksamhetens kalendarium.
>
> *Elevregister:* Folkhögskolan har ett elevregister som delvis används för att skapa underlag för statliga bidrag. Eftersom folkhögskolan söker sådana bidrag för

kortkursdeltagare behöver deras personuppgifter införas i detta register.

Schemaläggningsprogram: Används för att hantera scheman för lärare, elever och lokalreservationer, främst klassrum men även konferensrum, skolans aula samt gymnastiksal.

Ekonomisystem: Förutom att hantera övrig ekonomi på folkhögskolan hanterar detta system även betalning av genomförda arrangemang och bokningsavgifter med mera.

2.2 Problem med nuvarande lösning

Nuvarande lösning för att boka arrangemang är inte anpassad för verksamhetens omfattning. Det uppstår ofta problem då flera samtidiga administratörer vill arbeta och de tvingas då invänta varandra för att kunna arbeta vidare. Det finns inte heller någon funktion för att förhindra dubbelbokningar utan detta kontrolleras istället manuellt av personalen.

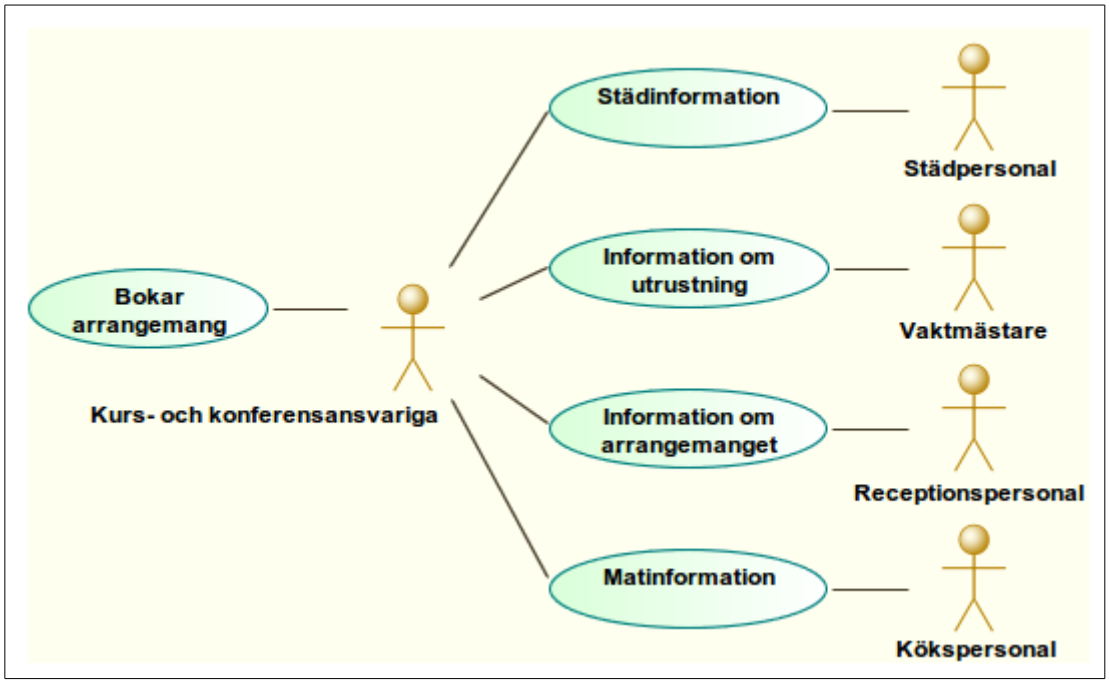
Förutom kurs- och konferensansvariga så berör bokningar även följande personalgrupper på folkhögskolan:

Kökspersonalen behöver information om vilken mat som ska tillagas, när den ska serveras, antal gäster och specialkost.

Receptionen är informationsnavet för gäster på skolan. Receptionens personal måste vara informerad om vad som pågår på skolan för att kunna ge korrekt information till gäster.

Städpersonalen behöver veta när och vilka lokaler som är uthyrda och när nästa uthyrning av lokalen sker så de kan planera sitt arbete.

Vaktmästarna tillhandahåller bland annat extrautrustning för arrangemanget, alltifrån extrasängar i övernattningsrum till stafflier för akvarellkurser.



Figur 2.1: Roller i kurs- och konferensverksamheten

Vid varje enskild bokning är en eller flera av ovanstående personalgrupper berörda vilket leder till ett stort informationsflöde. De har inte direkt tillgång till informationen som behövs utan istället är det kurs- och konferensansvarigas ansvar att informationen når alla berörda. Det är speciellt viktigt att eventuella ändringar i bokningar når berörd personal snabbt så de inte arbetar med gammal information. Figur 2.1 illustrerar de olika rollerna i verksamheten.

För exempel på vad för slags information som de olika personalgrupperna behöver, se Appendixfigur 1, Appendixfigur 2 och Appendixfigur 3.

3 METOD

Detaljer kring hur bokningssystemet ska fungera är inte känt i förväg. Därav kommer utvecklingen av systemet att ske i tätt samarbete med Mullsjö folkhögskola och en kravspecifikation kommer att upprättas under projektets gång. Projektet kommer att inledas med en jämförelse av två utvecklingsramverk. Eftersom ingen tidigare erfarenhet finns av något av de ramverk som valet står emellan är en snabb inlärningsprocess helt avgörande. Därför kommer sökandet efter en passande guide eller bok för att lära sig det ramverket som väljs att prioriteras högt. Helst ska en sådan guide eller bok bygga på konkreta kodexempel för att utvecklingen ska komma igång så snart som möjligt. För utvecklingsarbetet kommer agil utveckling att användas och även testdriven utveckling. I början av varje iteration ska ett möte med folkhögskolan hållas där en redovisning av föregående iteration ges följt av en planering av nästkommande iteration. Dessa möten ska vara regelbundna och ge folkhögskolan möjlighet att komma med återkoppling på arbetet.

3.1 Agil utveckling

År 2001 skrevs det agila manifestet ”Agile Software Development Manifesto” [1] av The Agile Alliance. Utvecklingsmetoden togs fram som ett alternativ till de, enligt The Agile Alliance, tungrodda metoder där dokumentation driver utvecklingen. Att arbeta agilt har sedan dess blivit mycket populärt och fler och fler ansluter sig till manifestet [2]. Det agila manifestet består av tolv principer och ett urval är citerade nedan [1]:

> *”Vår högsta prioritet är att tillfredställa kunden genom tidig och kontinuerlig leverans av värdefull programvara.”*
>
> *”Välkomna förändrade krav, även sent under utvecklingen.”*
>
> *”Leverera fungerande programvara ofta”.*
>
> *”Verksamhetskunniga och utvecklare måste arbeta tillsammans”.*
>
> *”Kommunikation ansikte mot ansikte är det bästa och effektivaste sättet att förmedla information”.*
>
> *”Fungerande programvara är främsta måttet på framsteg.”*

3.1.1 Test-Driven Development

Det sägs att den agila utvecklingsmetoden Test-Driven Development (TDD eller testdriven utveckling) är uppfunnen eller återupptäckt av Kent Beck, en av författarna till [1]. I sin bok ”Test Driven Development: By Example” [3] beskriver han två grundläggande koncept för att skriva mjukvara; varje kodrad i ett program ska vara täckt av automatiska tester för att verifiera dess korrekthet och kod ska aldrig dupliceras.

Laurie Williams et al. [4] beskriver arbetsproceduren för TDD på följande sätt. Utvecklaren arbetar i små korta cykler som börjar med att skriva ett test för automatisk enhetstestning (*eng. unit testing*). Detta test körs sedan för att verifiera att det fallerar, då ingen programkod är skriven än. Först därefter görs själva implementeringen av

programkoden. Testet körs på nytt och denna gång verifieras att den nya programkoden gör att testet lyckas. Efter detta steg börjar cykeln om och ett nytt testfall skrivs. Periodvis körs alla testfall för att göra en så kallad regressionstestning.

Det har visat sig att testdriven utveckling kan vara en mycket effektiv utvecklingsmetod. En av dem som förespråkar TDD är Robert C. Martin, en erfaren mjukvarukonsult och tillsammans med Beck en av författarna till [1]. I sin artikel ”Professionalism and Test-Driven Development” [5] skriver han om vanliga problem som mjukvaruutvecklare stöter på och argumenterar för hur TDD kan vara till hjälp. Ett exempel på ett sådant problem är avlusning, något som ofta är tidskrävande och utmanande. Med TDD kan felaktigheter upptäckas redan vid testning vilket kan leda till att tiden för avlusning kan minskas drastiskt. Ett annat exempel är att många utvecklare enligt Martin inte vill skriva om kod som fungerar. Men om TDD används bygger utvecklaren upp en bas av tester. Om svårförståelig kod då omstruktureras kan utvecklaren efter omskrivningen enkelt köra igenom testbasen för verifiera att koden fortfarande är korrekt. Detta leder enligt Martin till mindre komplicerad kod och färre kodrader.

Vidare skriver Janzen och Saiedian i artikeln “Does Test-Driven Development Really Improve Software Design Quality?” [6] att utvecklare som använder TDD tenderar att skriva färre och mindre komplexa kodrader och att testen som skrivs täcker större delar av programmet. Laurie Williams et al. skriver om en studie på IBM som visade att den grupp som använde TDD hade 40 % färre defekter på den utvecklade mjukvaran jämfört med den grupp som inte använde samma utvecklingsmetod [4].

4 TEKNISK BAKGRUND

Detta kapitel ger en bakgrund till olika tekniska detaljer som rapporten berör. Först beskrivs två ramverk för webbutveckling då ett av dessa är tänkt att användas för implementeringen av bokningssystemet. Därefter följer information om kalendern Fullcalendar, användargränssnittet Bootstrap och till sist kort om versionshanteringsverktyget Git.

4.1 Ramverk

Det finns många olika ramverk för webbutveckling. Nedan beskrivs två stycken; först presenteras respektive programmeringsspråk och därefter följer en beskrivning av själva ramverket.

4.1.1 Ruby

Ruby är ett relativt nytt programmeringsspråk som släpptes 1995 [7]. Skaparen av språket, Yukihiro Matsumoto, hämtade inspiration från ett flertal programmeringsspråk såsom Ada, Lisp och Perl, för att skapa ett nytt funktionellt och imperativt språk. Det är också ett objektorienterat och dynamiskt skriptspråk där allt ses som objekt. Nu är Ruby ett populärt programmeringsspråk och enligt en undersökning som IEEE Spectrum har gjort så ligger språket på en åttondeplats av världens mest populära programmeringsspråk år 2014 [8]. Enligt Rubys officiella hemsida beror en stor del av språkets popularitet på dess webbramverk Ruby on Rails [7].

För mer information om Ruby se [7].

4.1.2 Ruby on Rails

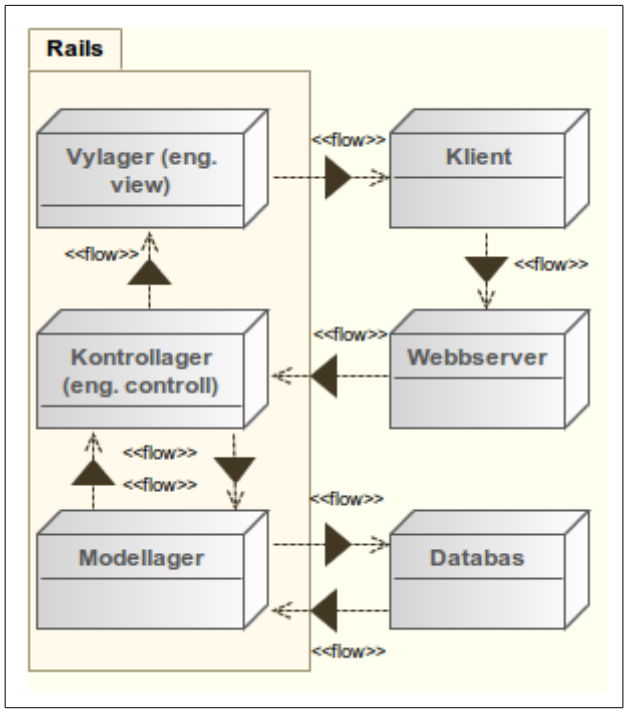
Ruby on Rails (Rails) är ett ramverk för webbutveckling skrivet i programmeringsspråket Ruby. Ramverket är släppt som öppen källkod under MIT-licensen. Rails bygger på följande två grundprinciper [9]:

> *”Convention Over Configuration”* - Ramverket är byggt på vissa lösningar som anses bättre än andra när det handlar om webbutveckling. Det är vedertagna och allmänt accepterade lösningar som utvecklaren uppmanas använda direkt istället för att behöva göra detaljinställningar för varje ny webbapplikation.
>
> *”Don't Repeat Yourself” (DRY)* - För att få hålla programkoden flexibel och lätt att underhålla så erbjuds ett antal verktyg för att uppmuntra utvecklaren att inte upprepa kod.

Ett konkret exempel på den första principen är att Rails är byggt på Model-View-Controller (MVC) mönstret. Ett annat exempel är att ramverket erbjuder en färdig ORM (Object-relation Mapping) för modellklasser. Istället för att i varje nytt projekt behöva definiera hur databaskopplingen ska fungera så erbjuder Rails en färdig lösning för detta.

Ett exempel på den andra principen är att Rails bistår med en mängd funktioner och verktyg för att förhindra duplicering av programkod. Exempelvis finns så kallade ”Partials”

som kan återanvändas i flera vyklasser. Rails kan också generera kodskelett åt utvecklaren. Ett sista exempel är att modellklassen automatiskt hämtar rätt data från databasen med hjälp av klassnamnet.



Figur 4.1: Förenklad arkitekturskiss av Rails.

Figur 4.1 visar en förenklad arkitekturskiss av Rails. Från internetläsaren (klienten) kommer anrop in till webbservern. Från webbservern går anropet vidare till kontrollagret. I detta lager finns klassen *ActionController* som alla kontrollklasser i applikationen ärver från. Om data behövs från modellen anropas aktuell klass via CRUD-operationer. CRUD står för Create, Read, Update och Delete och används i webbapplikationer då de motsvarar HTTP-anropen POST, GET, PUT/PATCH, DELETE. *ActiveRecord* är en klass i modellagret som alla modellklasser i applikationen ärver och som sköter all kommunikation med databasen. *ActiveRecord* svarar kontrollklassen som sedan via vylagret skickar tillbaka information till klienten.

ActiveRecord erbjuder olika verktyg för att hantera databaskopplingen. Om en modellklass ska sparas ner till databasen behöver en databastabell skapas. Detta görs via *ActiveRecordMigrations*, som även används för att uppdatera tabellerna under utvecklingens gång. En utförlig förklaring finns på [10]. För att kunna representera associationer mellan klasser används *ActiveRecordAssociations*, se [11] för mer information.

Till Rails finns en mängd olika plugins, så kallade ”Ruby gems”, som ger ramverket utökad funktionalitet. Se [12] för mer information om olika gems. Några exempel är:

sdoc: genererar dokumentation automatiskt från kommentarer i källkoden.

bcrypt-ruby: hashfunktion för kryptering av exempelvis lösenord.

RSpec: ramverk för testning.

Capybara: används för integrationstestning. Simulerar en användare som interagerar med webbapplikationen.

Det finns många guider på Internet för att lära sig Rails. Ett exempel är boken ”Ruby on Rails Tutorial” skriven av Michael Hartl som finns tillgänglig gratis på Internet [13]. Boken går ut på att från grunden bygga en fungerande och relativt avancerad blogg med stöd för olika användare och administratörer, autentiseringssystem med mera. All kod i denna bok är släppt under licensen MIT-licensen och Beerware License.

Värt att notera är Rails syntax för klasser om de är indelade i moduler. Först står namnet på modulen följt av två kolon och därefter står klassnamnet: *ModulNamn*::*KlassNamn*.

Mer information om Rails finns på [14] och [9] samt en kortare sammanfattning på [15].

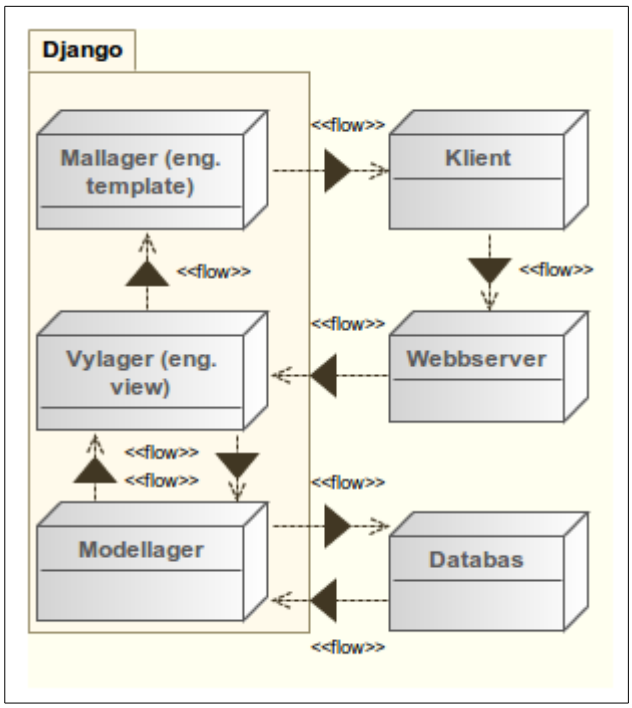
4.1.3 Python

Python är ett objektorienterat skriptspråk som är dynamiskt typat och skrivet i öppen källkod [16]. Språket utvecklades av Guido van Rossum på 1990-talet som en efterträdare till ABC [17]. Syntaxen är enkel och anpassad för snabb programmering. Python är numera mycket populärt och IEEE Spectrum har rankat Python på en femteplats av de mest populära programmeringsspråken i världen år 2014 [8].

För mer information om Python se officiell hemsida [18].

4.1.4 Django

Django är ett av Pythons ramverk för webbutveckling. Det är skrivet i öppen källkod och byggt för effektiv och snabb webbutveckling [19]. Django bygger likt Rails på konceptet DRY, att utvecklaren ska slippa upprepa kod, och erbjuder lösningar så som färdig ORM och hjälp att skapa formulär. Det medföljer även ett administratörsgränssnitt från start.

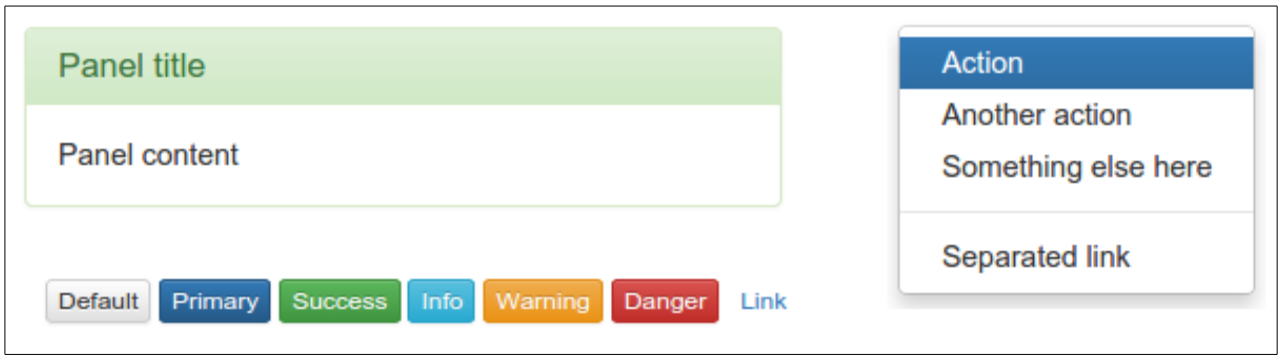


Figur 4.2: Förenklad arkitekturskiss av Django.

Istället för att använda den kända beteckningen MVC för designmönstret så kallar Django detta för MTV, vilket står för Model-Template-View [20]. MTV:s Template motsvarar MVC:s View och MTV:s View motsvarar MVC:s Controller. Figur 4.2 visar en förenklad arkitekturskiss över Django där varje lager i MTV-strukturen syns. Mallagret (*eng. template*) erbjuder ett användarvänligt gränssnitt. Om användaren interagerar med mallagret skickas en begäran (*eng. request*) från klienten ner till vylagret (*eng. view*) via webbservern. Som beskrivet ovan i avsnittet om Rails används CRUD-operationer om data behövs från modellagret som i sin tur hämtar informationen från databasen. Modellagret svarar vylagret som sedan skickar data till användaren via mallagret.

4.2 Bootstrap

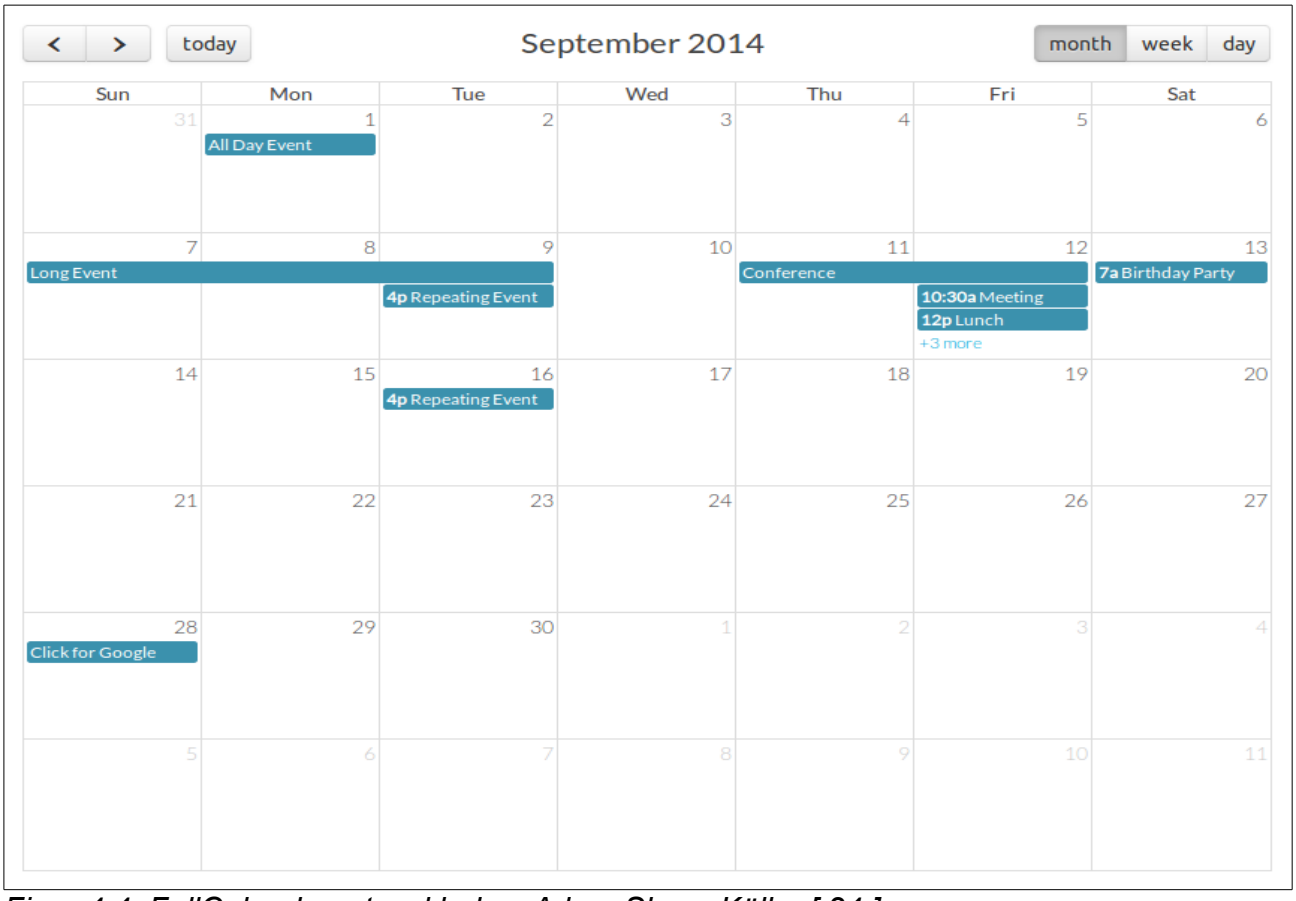
Bootstrap är ett ramverk för webbutveckling för att snabbt och enkelt skapa användargränssnitt [21]. Mer specifikt innehåller ramverket CSS-filer (Cascading Style Sheets) och JavaScript som är färdiga att använda. Ramverket är släppt under MIT-licensen. Figur 4.3 visar exempel på vad Bootstrap erbjuder.



Figur 4.3: Exempel på komponenter från Bootstrap. Kollage med bilder från [22].

4.3 FullCalendar

FullCalendar är en populär kalender skriven i JavaScript av Adam Shaw [23]. Kalendern kan visa händelser men ger dock ingen lösning för hur dessa ska hanteras, detta är upp till utvecklaren själv som väljer att använda kalendern. FullCalendar är släppt under MIT-licensen. Figur 4.4 visar ett exempel på en vy i kalendern.



Figur 4.4: FullCalendar, utvecklad av Adam Shaw. Källa: [24].

Företaget Vinsol har gjort ett plugin till Rails, kallat FullcalendarEngine (Rails kallar ett plugin för Engine) som integrerar denna kalender i en Rails-applikation [25]. Detta plugin hanterar också händelserna i kalendern. Det är skrivet i öppen källkod och släppt under MIT-licensen.

4.4 Git

Git är ett populärt versionshanteringssystem för källkod. Läsaren förväntas ha grundläggande kunskap om hur Git fungerar. För mer information se Gits officiella hemsida [26].

För att arbeta effektivt med Git är det viktigt att ha en modell för hur arbetet ska organiseras. Ett exempel på en sådan modell är en som Driessens har gjort [27]. Den bygger på att huvudgrenen (*eng. master branch*) hålls så ren som möjligt och endast innehåller kod som är klar för produktion, alltså fungerande kod som är testad, avlusad och färdig för att användas av slutanvändaren. Den gren som man i huvudsak arbetar mot är istället en så kallad utvecklingsgren (*eng. develop branch*) och varje ny funktion som ska läggas till i programmet görs i en egen gren utifrån utvecklingsgrenen. När funktionen är färdig sammanfogas den (*eng. merge*) med utvecklingsgrenen. När sedan utvecklingsgrenen innehåller tillräckligt mycket nytt som är färdigt för produktion så sammanfogas utvecklingsgrenen med huvudgrenen. På detta vis kan man säkerställa att koden i huvudgrenen alltid är körbar och fungerande.

5 SYSTEMDESIGN

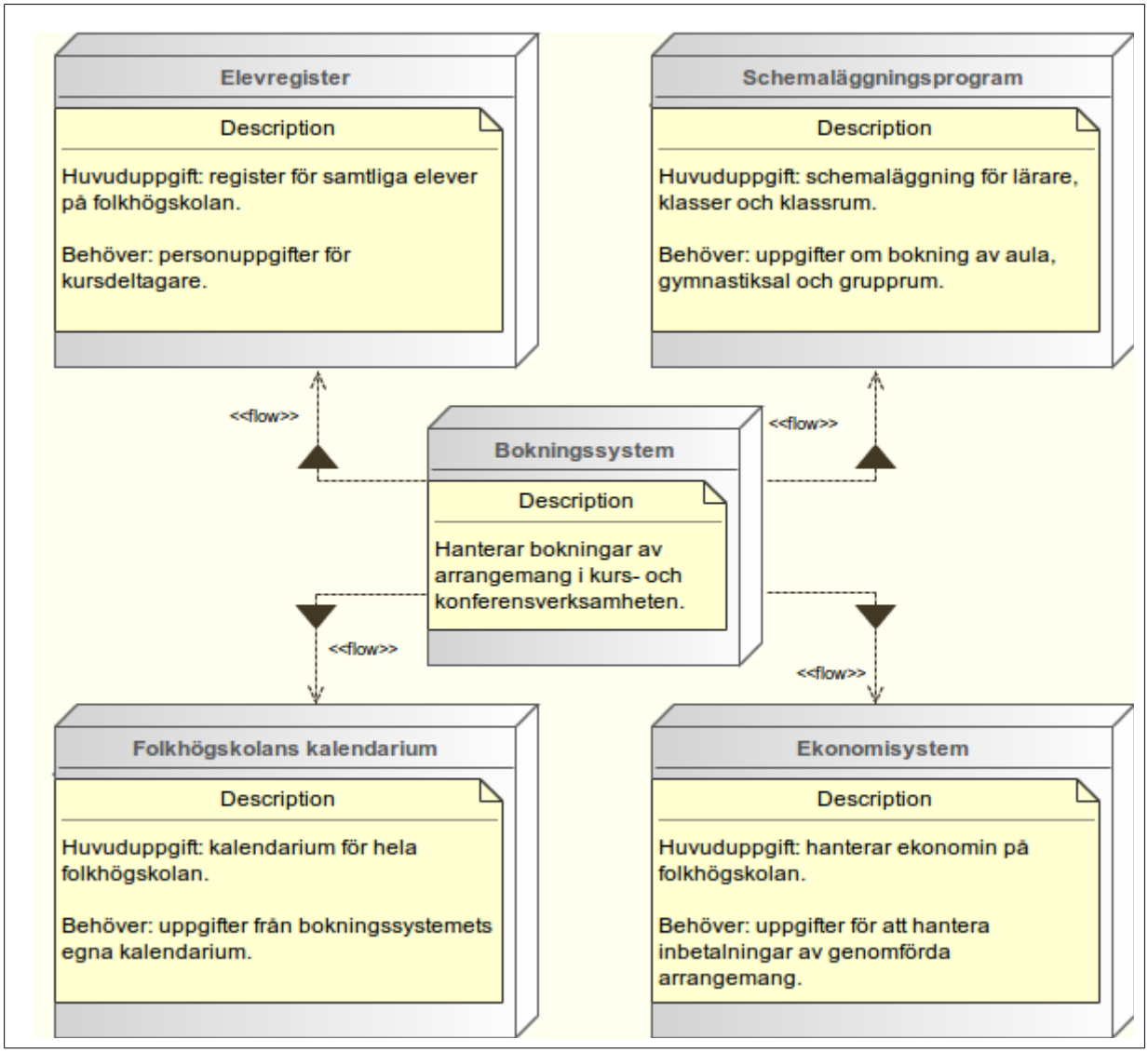
Här introduceras det bokningssystem som ska utvecklas i projektet genom att förklara dess design. Först ges en domänbeskrivning för att belysa vad systemet ska hantera och hur systemmiljön ser ut där systemet är tänkt att installeras så småningom. Efter det beskrivs användarfall och grundläggande begrepp. Sist följer en redogörelse för hur ramverk, databaskoppling och öppen källkod påverkar designen.

5.1 Domän

Det nya bokningssystemet ska ersätta de kalkylblad och textdokument som i dagsläget används för att hantera bokningar (se avsnitt 2.1). Mer specifikt ska systemet hantera bokningar av kortkurser, konferenser och vandrarhem. För mer detaljer om vad en bokning av ett arrangemang på Mullsjö folkhögskola är, se Appendixfigur 4 i Appendix I – Diagram och figurer som visar hur en vandrarhemsbokning sker i dagsläget (flödet för kortkurser och konferenser är liknande). Se även Appendix III – Kravspecifikation för mer information.

5.2 Systemmiljö

Bokningssystemet ska installeras i en redan existerande systemmiljö på folkhögskolan. I denna miljö ingår ett antal olika program som bokningssystemet måste förhålla sig till, se Figur 5.1. Fokus kommer att ligga på kopplingen mellan bokningssystemet och elevregistret samt schemaläggningsprogrammet. Kommunikation med ekonomisystemet och skolans övriga kalendarium faller utanför projektets ram.



Figur 5.1: Systemmiljö på Mullsjö folkhögskola

5.2.1 Elevregister

Bokningssystemet behöver hantera personuppgifter från kortkursdeltagare, dels för att skapa och hantera förläggningslistor men också för att koppla deltagare till kurser och skapa väntelistor då kurser är fulla. Denna information behöver sedan överföras från bokningssystemet till elevregistret när arrangemanget är genomfört och folkhögskolan ska söka bidrag för kursen. Då elevregistret inte har något stöd för kommunikation med tredjepartsapplikationer finns ingen möjlighet att automatiskt överföra informationen. Istället behöver bokningssystemet designas så att deltagarinformation kan exporteras. Informationen får sedan importeras manuellt i elevregistret.

5.2.2 Schemaläggningsprogram

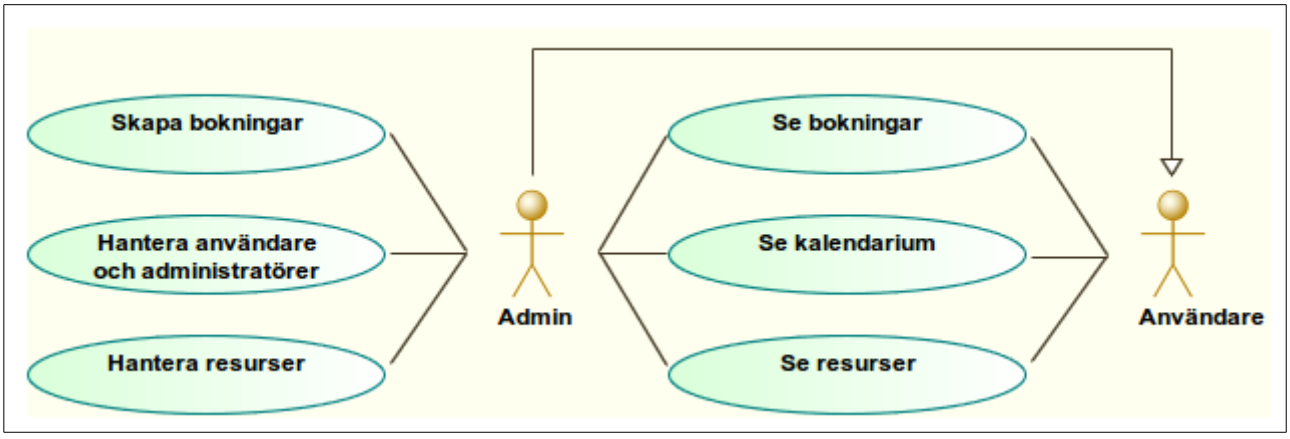
Schemaläggningsprogrammet som används körs lokalt på en av folkhögskolans egna servrar. Det finns inget API för kommunikation med tredjepartsapplikationer. Enligt tillverkaren är en plattform under utveckling dit schemaläggningsprogrammet ska kunna exportera data. Tredjepartsprogram ska sedan kunna kopplas till denna plattform för att läsa den data som schemaläggningsprogrammet exporterat. Det finns ännu inga beslut om

att kunna skriva data till plattformen och det är oklart när någon färdig fungerande produkt finns tillgänglig. Tillverkaren har inte heller något API tillgängligt för hur kommunikation med plattformen kommer att ske och därmed kan bokningssystemet ej designas för kommunikation med schemaläggningsprogrammet.

Om det hade funnits möjlighet till kommunikation skulle bokningssystemet kunna delegera reservationer av lokaler till schemaläggningsprogrammet, istället måste nu båda hantera lokalreservationer. Om man tillåter att båda programmen hantera samma lokaler går dubbelreservationer inte att förhindra. Det går inte heller att överlåta alla lokalreservationer åt schemaläggningsprogrammet då bokningssystemet måste länka samman vilka lokalreservationer som hör ihop med vilka arrangemang. Den lösning som har valts är att dela upp så att bokningssystemet hanterar övernattningsrum och konferenslokaler medan schemaläggningsprogrammet hanterar alla klassrum. Tyvärr behöver, enligt folkhögskolan, både schemaläggningsprogrammet och bokningssystemet ha uppgifter om aulan, gymnastiksalen och grupprum. Eventuella dubbelreservationer av dessa får därför hanteras manuellt.

5.3 Användare och användningsfall

Folkhögskolan önskar två olika typer av användare för bokningssystemet, dels administratörer med fullständiga rättigheter och dels ”vanliga” användare som främst ska läsa information från systemet. Administratörer för systemet är exempelvis administratörer för kurs- och konferensverksamheten och vanliga användare kan vara kökspersonal eller vaktmästare. Figur 5.2 beskriver några grundläggande användningsfall som behövs i bokningssystemet.

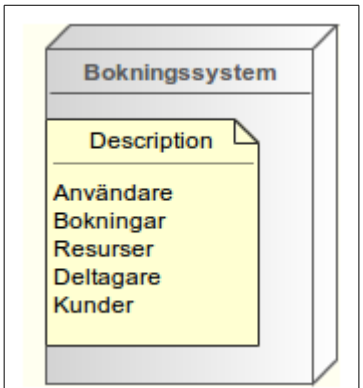


Figur 5.2: Grundläggande användningsfall

Administratörer ska kunna skapa och hantera bokningar. Användare ska kunna se kalendarium för bokningar och även kunna nå enskilda bokningar för att se mer detaljerad information. För en bokning behöver ett antal resurser reserveras, såsom övernattningsrum och konferensrum. Administratörer hanterar dessa resurser och en användare kan läsa information om dem.

5.4 Grundläggande begrepp

För att bokningssystemet ska kunna hantera ovan beskrivna användningsfall används de grundläggande begrepp som visas i Figur 5.3: användare, bokningar, resurser, deltagare och kunder.



Figur 5.3: Grundläggande begrepp i bokningssystemet

Användare: Att systemet behöver hantera användare har redan beskrivits i avsnitt 5.3. Vad som bör tilläggas är att för hanteringen av dessa användare måste bokningssystemet ha någon form av autentiseringsmekanism, dels för att se till att obehöriga inte får tillgång till bokningssystemet och dels för att särskilja användare från administratörer.

Bokning: En bokning är naturligtvis det centrala begreppet i systemet. Den behöver ha information om själva arrangemanget, såsom ett namn, beskrivning och tid. En bokning behöver också kunna reservera resurser.

Resurs: För att genomföra ett arrangemang behövs resurser, såsom lokaler. En administratör ska kunna hantera resurserna i systemet, exempelvis lägga till nya resurser och ta bort sådana som inte används längre.

Deltagare: För kortkurser måste deltagarna registreras för att systemet ska kunna avgöra när kurser är fulla samt för att skapa förläggningslistor och reservlistor. En administratör ska kunna ange personuppgifter till en deltagare, såsom namn och personnummer.

Kunder: En bokning behöver kunna associeras med en kund. Då det är ganska vanligt med återkommande kunder i folkhögskolans kurs- och konferensverksamhet är det en stor fördel för administratörerna för kurs- och konferensverksamheten om kunduppgifter kan sparas till nästa bokning. Det är också nödvändigt att ha kontaktuppgifter till kunderna.

5.5 Ramverk

Eftersom resurserna för detta projekt är mycket begränsade är ett effektivt ramverk för mjukvaruutvecklingen helt avgörande. Några enkla sökningar på Internet visar att det åtminstone finns ett hundratal populära alternativ för ramverk för webbutveckling att tillgå [28] [29]. Det hade kunnat vara ett projekt i sig att jämföra och utvärdera olika ramverk med varandra. Även om detta projekt inte fokuserar på en sådan jämförelse eller utvärdering så var ändå valet tvunget att göras. Följande krav fanns från projektstart:

- Då bokningssystemet ska släppas som öppen källkod är det en fördel om även ramverket är det eftersom det ofta medför att även tillägg till ramverket är öppna.
- Ramverket ska vara anpassat för snabb webbutveckling av två anledningar. Dels är resurserna för projektet knappa och dels är ett av syftena med projektet att utvärdera om väl anpassade verktyg faktiskt ger en så snabb utveckling som de utlovar.

- Det måste vara ett ramverk som är relativt populärt eftersom det ökar chansen för att nödvändiga tillägg finns och för att möjligheten att hitta hjälp från andra vid behov är större.
- Det ska finnas färdigt stöd för databaskoppling (ORM).
- För det egna lärandeändamålet är det önskvärt med ett helt nytt programmeringsspråk, vilket utesluter ramverk skrivna i Java.

Två ramverk som uppfyller ovanstående krav är Rails och Django. Dessa två valdes ut för att sedan jämföras med varandra. Båda har ett stort utbud av plugins [12], [30]. Några funktioner som troligen var nödvändiga var:

- Kalenderfunktion för att visa bokningar.
- Datumväljare för bokningar.
- Administrationsgränssnitt för att hantera hela systemet.

Rails har tillägg för samtliga ovanstående funktioner. Till Django kunde inget passande tillägg för en kalenderfunktion hittas. Dessutom har Django en annan nackdel. 2008 kom version 3 av Python som Django är skrivet i. Denna version är inte bakåtkompatibel med den äldre version 2 [31]. Version 3 ses som framtidsversionen och skulle vara det naturliga valet för detta projekt. Tyvärr är många av de paket som finns för Django utvecklade för Python version 2 och har i skrivande stund inte uppdaterats till version 3 [30]. Valet föll således på Rails.

Förutom ett ramverk för webbutveckling behövs även ett för att skapa användargränssnitt. För detta har Bootstrap valts, se avsnitt 4.2. Detta val påverkar inte i lika stor grad systemdesignen och därav har ingen jämförelse gjorts mellan olika ramverk.

5.6 Databas

Det var redan från början ett krav att ramverket skulle erbjuda ORM, se avsnitt 5.5. Detta medför att en stor del av databasdesignen är given utifrån ramverket. I Rails, det ramverk som används i detta projekt, skapas databastabellerna med Active Record Migrations och eventuella relationer mellan objekt hanteras med Active Record Associations. Se 4.1.2 för mer information om hur databaskopplingen hanteras av Rails.

5.7 Öppen källkod

För att bokningssystemet ska kunna släppas som öppen källkod underlättar det om även de tredjepartsmoduler som används är skrivna i öppen källkod, något som bidrar till systemdesignen. I motsats till att använda stängd källkod öppnar det för möjlighet att anpassa modulerna vid behov och i motsats till att utveckla modulerna från början ges mer tid över till den övriga utvecklingen. Då tredjepartsmoduler används är det viktigt att systemet är designat så att dessa moduler är utbytbara. Om tillverkaren av tredjepartsmodulen upphör med utvecklingen kan hela systemet i värsta fall bli oanvändbart i framtiden. Dock är inte sårbarheten lika stor om det handlar om öppen källkod då utvecklingen kan tas över av andra.

Ett exempel på en tredjepartsmodul som behövs i bokningssystemet är en som kan presentera bokningar i ett kalendarium. Det är en viktig del i bokningssystemet för att kunna ge användarna en överblick över beläggningen.

6 UTFÖRANDE

Implementationen av bokningssystemet har skett i iterationer enligt agil utveckling. Detta kapitel beskriver två av dessa iterationer, en tidig och en senare, för att visa på processen i projektet. För mer utförlig information om möten med folkhögskolan under projektet se Appendix II – Mötesanteckningar. Kapitlets sista avsnitt beskriver hur inlärningen av ramverket skedde samt kort om hur implementeringsarbetet och testningen har utförts.

6.1 Tidig iteration

Under ett tidigt möte med folkhögskolan diskuterades vilka olika objekt bokningssystemet behöver hantera, vilket resulterade i de grundläggande begrepp som avsnitt 5.4 beskriver. Från de grundläggande begreppen gjordes sedan ett UML-diagram som kan ses i Figur 6.1.

Den centrala klassen i UML-diagrammet är *Booking* som representerar en bokning av ett arrangemang. Som tidigare nämnts finns det tre olika typer av bokningar. På något sätt behöver systemet kunna skilja dessa åt och här används arv för detta. Klasserna *ConferenceBooking*, *CourseBooking* och *HostelBooking* ärver av basklassen *Booking*.

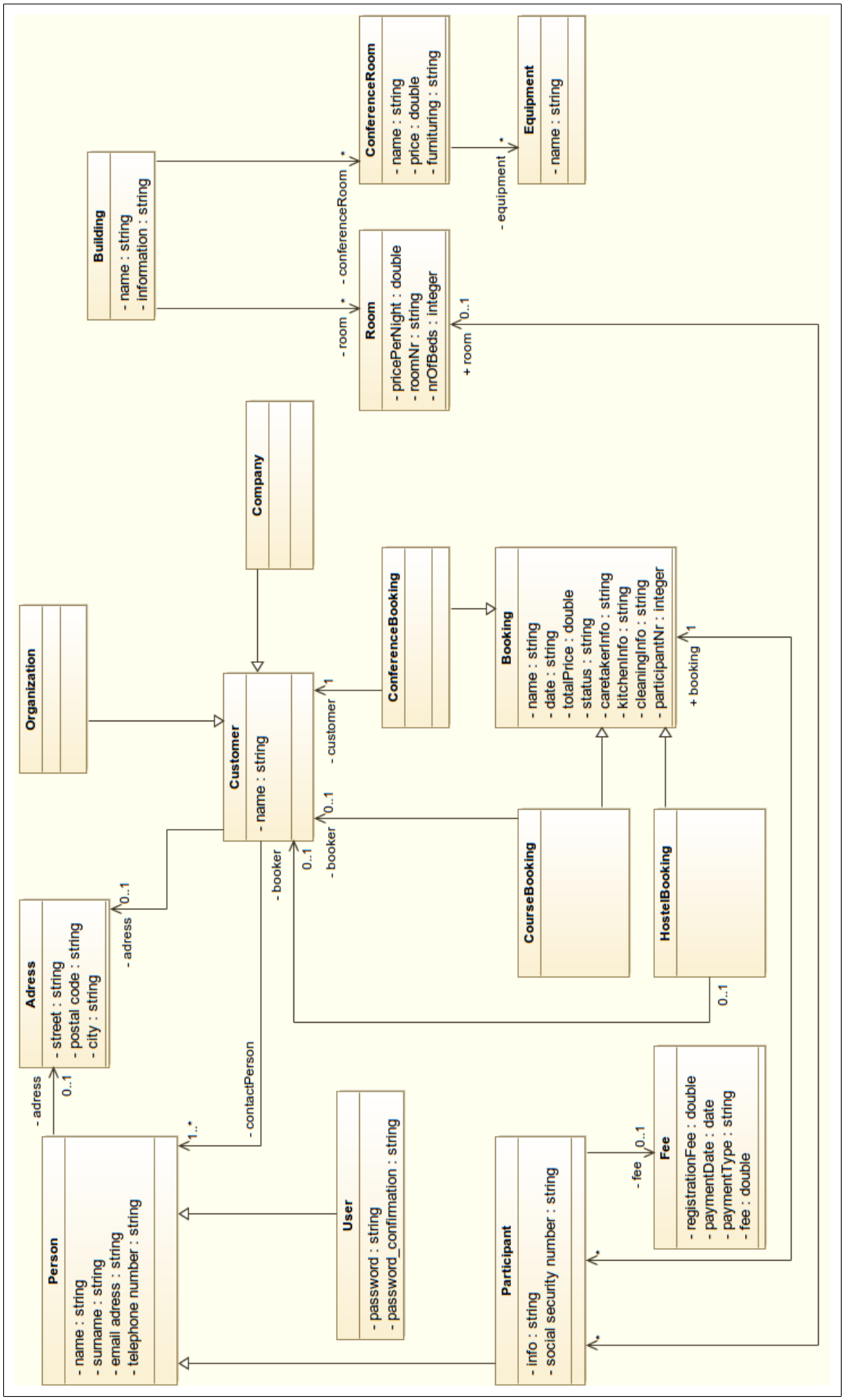
En bokning behöver associeras med någon typ av kund. I Figur 6.1 representeras en kund med klassen *Customer*. En kund kan delas upp i två underkategorier, organisation eller företag, som representeras av klasserna *Organization* och *Company*. För att kunna hantera en bokning behövs även kontaktuppgifter till en person. Klasserna *Person* och *Adress* används för detta.

Personer behövs i två andra sammanhang i systemet; dels som användare av själva systemet men också som deltagare i arrangemang. Klassen *User* representerar användare medan klassen *Participant* representerar deltagare i ett arrangemang.

En bokning behöver kunna reservera rum. I Figur 6.1 finns rum representerade av klasserna *Room* och *ConferenceRoom*. För att kunna hantera alla rum på folkhögskolan behövs även en koppling till vilken byggnad som rummet ligger i, vilket klassen *Building* används för.

Figur 6.1 är inte ett färdigt diagram utan ser ut precis så som det gjorde under utvecklingen vid detta stadium. Det finns en del fel i diagrammet, bland annat saknas relationen mellan en bokning och ett rum.

Efter denna iteration kunde man skapa användare i systemet, logga in och skapa en enkel bokning.



Figur 6.1: Första UML-diagrammet för bokningssystemet

6.2 Senare iteration

Vid mötet efter iterationen beskriven i föregående avsnitt diskuterades kalenderfunktionen för systemet och hur en sådan kunde se ut och fungera. FullCalendar, se avsnitt 4.3, demonstrerades och det verkade som om denna kalender uppfyllde folkhögskolans krav. Efter detta möte inleddes arbetet med att implementera en FullCalendar för Rails i form av FullcalendarEngine, se avsnitt 4.3. För att kunna implementera kalenderfunktionen behövde UML-diagrammet för bokningssystemet moduleras om, se resultatet i Figur 6.2. Det som visas i kalendern är instanser av klassen *FullcalendarEngine::Event* (Event). Varje bokningsobjekt behöver ett eget eventobjekt för att kunna representeras i kalendern. Klassen *Event* håller reda på start och stopptid för eventet samt en titel och en beskrivning. I föregående iteration fanns dessa attribut i klassen *Booking*. För att slippa duplicering flyttades denna information till klassen *Event*. Associationen är enkelriktad och klassen *Event* vet ingenting om klassen *Booking* vilket medför att den förstnämnda inte behöver modifieras och att FullcalendarEngine kan användas rakt av.

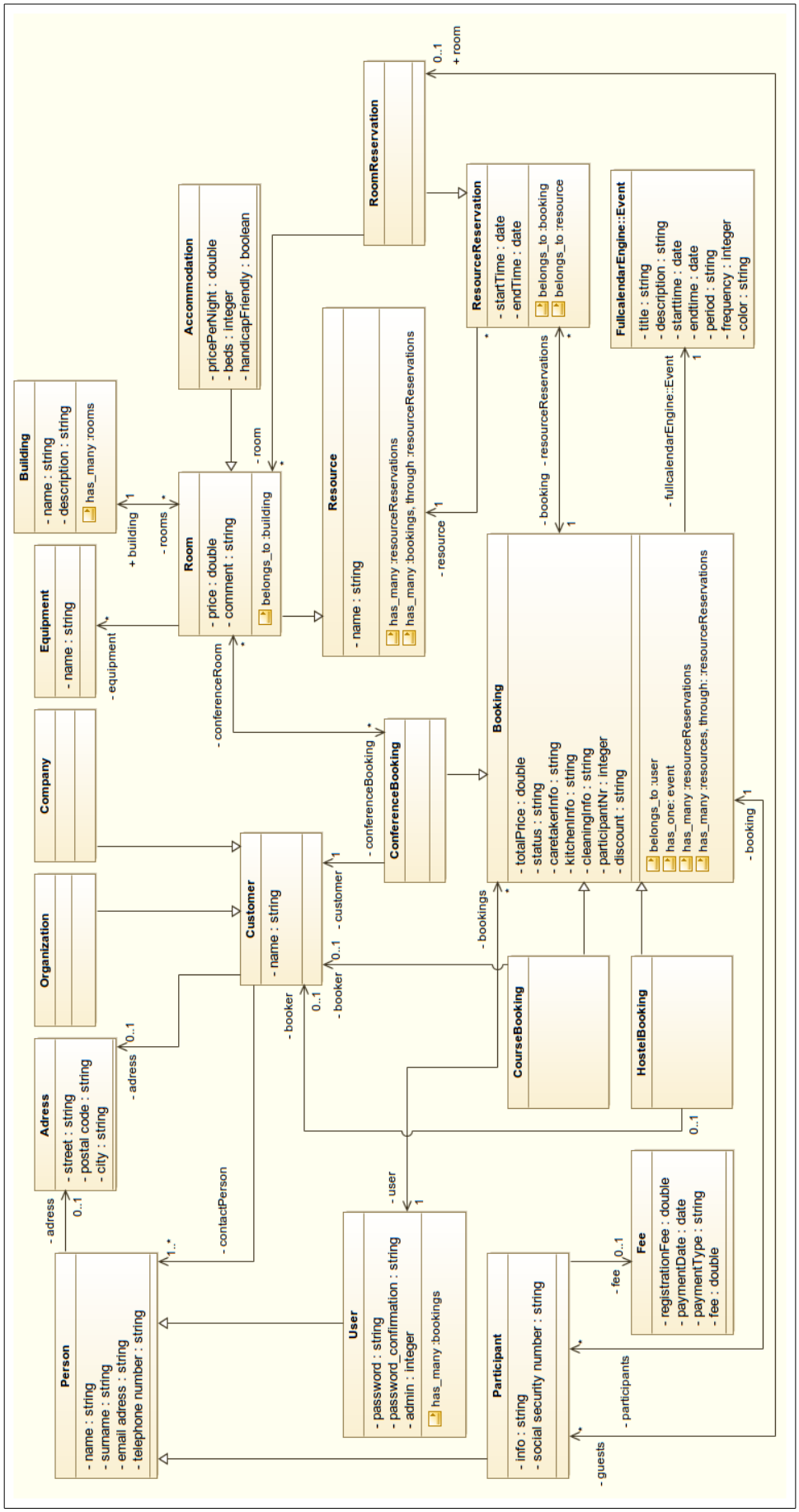
Vidare kan systemet nu hantera administratörer. Det finns ingen egen klass för administratörer; istället används ett attribut av boolesk datatyp hos en vanlig användare. Är attributet sant har användaren administratörsrättigheter. Användare kan bara skapas av administratörer och användarna själva har inte möjlighet att skaffa utökade rättigheter, det privilegiet kan bara ges från administratörer.

En bokning associeras nu med en administratör, vilket innebär att man kan se vem som har skapat en bokning. Resurserna i systemet har nu generaliserats från att bara innefatta rum till att nu innefatta vilken resurs som administratören av systemet behagar skapa. Resurser representeras av klassen *Resource* och klassen *Room* ärver av *Resource*. Istället för att som innan ha en klass för rum och en klass för konferensrum finns nu en klass för rum generellt och en klass för övernattningsrum. Detta för att det är just övernattningsrum som faktiskt skiljer sig från de övriga rummen på folkhögskolan och dessa rum som behöver några extra attribut för att kunna beskrivas tillräckligt noggrant.

En reservation av en resurs för en bokning sker nu via klassen *ResourceReservation*, som håller information om vilken tid resursen ska bokas. För att reservera ett rum används klassen *RoomReservation* som ärver av *ResourceReservation* och som även kopplar samman reservationen med en person, exempelvis då rummet som ska reserveras är ett övernattningsrum.

Efter denna iteration visas skapade bokningar i den tillagda kalendern. En användare kan antingen vara en vanlig användare eller en administratör. En administratör har extra privilegier såsom att ta bort resurser och skapa bokningar. På en administratörs profilsida syns senast gjorda bokningar som administratören har skapat.

Under en senare fas av utvecklingen visade det sig att FullcalendarEngine inte stödde alla funktioner i FullCalendar. Eftersom FullcalendarEngine är skriven i öppen källkod kunde en ny Git-förgrening (*eng. fork*) av programmet göras där en funktion för färgval för händelser implementerades. Dessutom följde några stilregler för vylagret med FullcalendarEngine som inte var önskvärda, varför dessa togs bort i den nya förgreningen.



Figur 6.2: Senare UML-diagram för bokningssystemet

6.3 Att lära sig ramverket samt om implementering och testning

Då ingen tidigare erfarenhet fanns av varken Ruby eller Ruby on Rails var en snabb inlärning nödvändig. Det finns en mängd olika guider och böcker på Internet för att lära sig ramverket. En bok som verkade passa bra var Michael Hartls ”Ruby on Rails Tutorial”, se avsnitt 4.1.2, då boken förklarar hur man bygger en relativt avancerad webbapplikation från grunden och då all källkod som är tryckt i boken är fri att använda under MIT-licensen. Samtidigt som inlärningen av språket och ramverket skedde kunde därför grunden för bokningssystemet byggas.

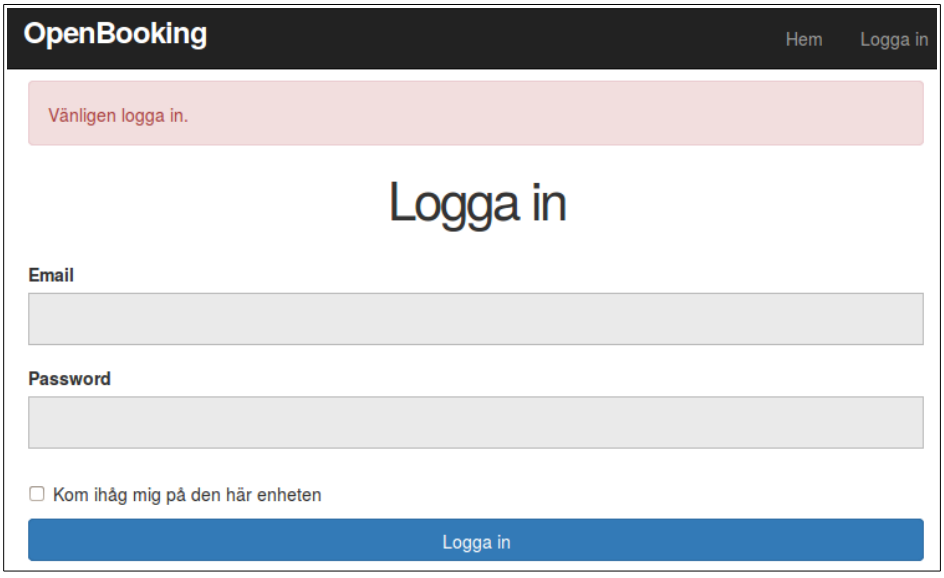
Michael Hartls lär ut Rails genom att använda sig av TDD, se avsnitt 3.1.1, och mer specifikt av testramverket RSpec för Rails. Hela implementeringsarbetet, både under tiden boken följdes och efteråt, har utförts med TDD med hjälp av RSpec. Hartls förespråkar också versionshanteringssystemet Git [26]. Detta verktyg har använts under hela utvecklingen och har genomförts enligt en modell av Driessens [27].

7 RESULTAT

Detta kapitel visar resultatet av projektet i form av skärmdumpar från bokningssystemet och förklarande text. Det implementerade systemet kallas för OpenBooking. Systemet är skrivet i öppen källkod och licensen som används är MIT-licensen, detta på grund av att Rails och använda tredjepartsmoduler är publicerade under samma licens. För den intresserade läsaren finns källkoden att hitta på GitHub [32].

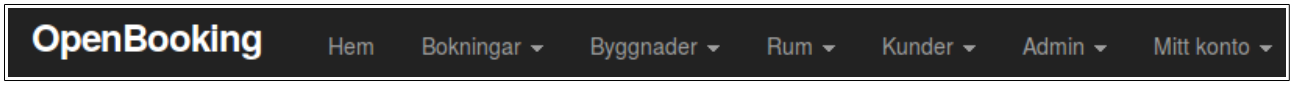
Figur 7.1: Förstasidan för bokningssystemet

Besökare till OpenBooking möts först av en välkomstsida, Figur 7.1. Systemet kräver inloggning och om besökaren försöker att interagera med systemet utan att logga in kommer ett varningsmeddelande upp och det sker en omdirigering till inloggningssidan, Figur 7.2.



Figur 7.2: Inloggningssida med varningstext

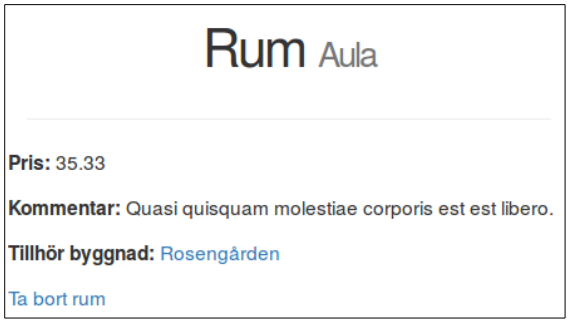
För att logga in anges e-postadress och lösenord. Ett autentiseringssystem kontrollerar att mejladressen finns registrerad och att lösenordet är korrekt. Om mejladressen inte är registrerad eller om lösenordet inte är korrekt visas ett förklarande felmeddelande.



Figur 7.3: Meny för inloggade användare

När en användare är inloggad i systemet finns fler menyalternativ att välja på, se Figur 7.3. Hem-alternativet tar användaren till startsidan för bokningssystemet, därefter följer länkar till objekt som systemet hanterar: bokningar, byggnader, rum och kunder. Dessa alternativ har undermenyer där man kan välja på att ”Skapa ny” eller ”Se alla”. Under ”Admin”-alternativet kan administratörer hantera systemets användare. Längst till höger finns ett alternativ som behandlar användarens konto.

Figur 7.4: Ett urval av exempelrum som lagts till i bokningssystemet

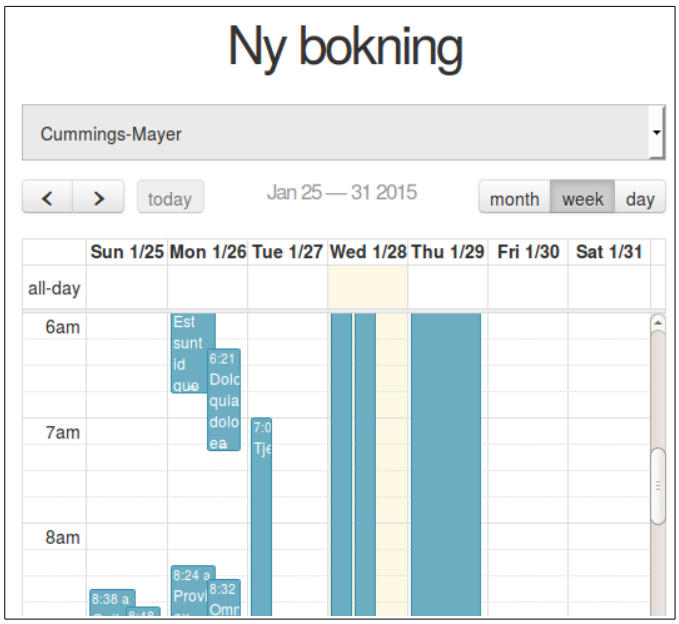


Figur 7.5: Exempel på vy för ett specifikt rum

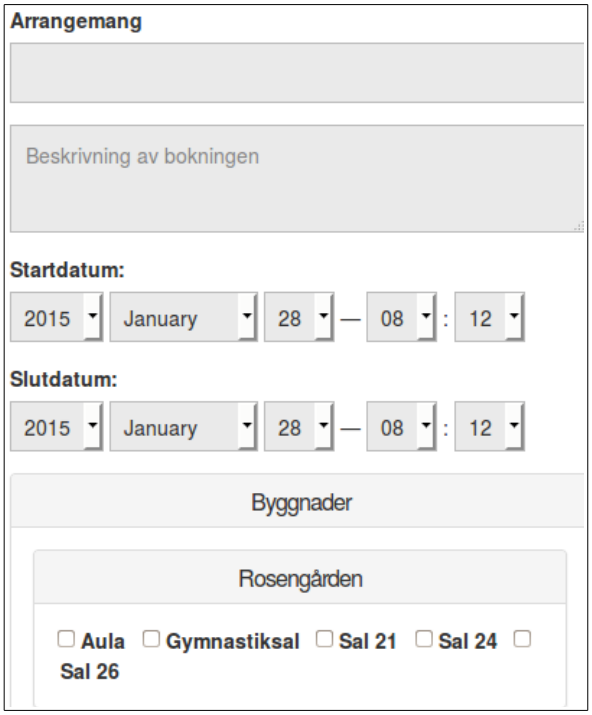
Figur 7.6: Vy för att skapa nytt rum

Klickar man exempelvis på ”Rum” och sedan ”Se alla” får man en vy där alla rum i systemet listas, se Figur 7.4 för ett urval av dessa. Här syns namnet på rummet och är användaren administratör finns även en snabblänk för att ta bort byggnaden. Klickar man på rummets namn kommer man vidare till en vy där rummets egenskaper visas, se Figur 7.5. Användaren kan också skapa nya rum och då ser vyn ut som Figur 7.6. På samma sätt fungerar det för bokningar, byggnader och kunder.

Bokningsobjektet är det mest omfattande och för att skapa en ny bokning är lite mer information som behövs. Figur 7.7 visar den första delen där administratören först får välja vilken kund som bokningen tillhör. Därefter visas bokningskalendern för att påminna administratören om vilka bokningar som redan finns i systemet.



Figur 7.7: Vy för ny bokning, del 1



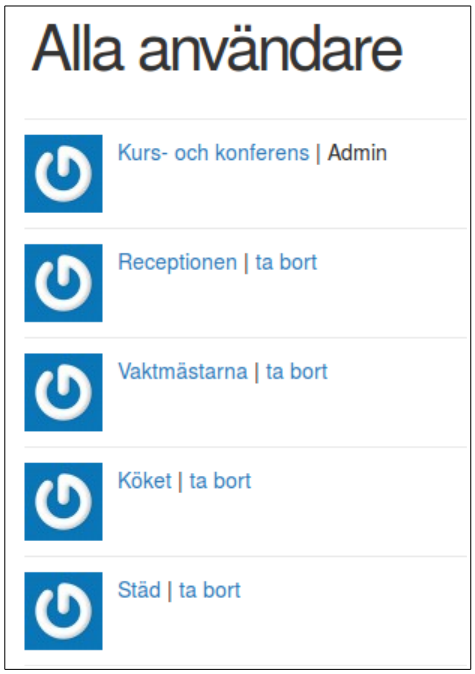
Figur 7.8: Vy för ny bokning, del 2

Figur 7.8 visar fortsättningen av sidan för att skapa nya bokningar där administratören får fylla i titel på arrangemanget och en beskrivning följt av start- och slutdatum för bokningen. Efter det listas alla byggnader i systemet och inuti rutan för byggnaden visas de rum som tillhör byggnaden. I nuvarande utformning av systemet visas även sådana rum som inte kan bokas, det vill säga systemet anpassar sig inte till den ifyllda tiden och tar inte bort de rum som redan är bokade vid denna tid. Däremot säger systemet till om man försöker boka ett rum som redan är bokat. Användaren får då en varningstext som säger att rummet är upptaget och bokningen går inte igenom förrän ett annat rum är valt. På liknande sätt valideras nödvändig information för alla objekt i systemet. Ett nytt rum kan inte kan skapas utan namn och det måste tillhöra en byggnad, en bokning kan inte ha ett slutdatum före startdatumet, en ny kund kan inte skapas utan att ange ett namn och så vidare.

Figur 7.9: Bokningskalender

Samtliga bokningar i systemet visas i bokningskalendern, se Figur 7.9. Denna kalender kan förutom via vyn för nya bokningar även nås genom att klicka på ”Bokningar” i menyn och sedan välja ”Bokningskalender”. Det finns olika vyer för månad, vecka samt dag. Användaren kan bläddra fram och tillbaka i kalendern för att se beläggningen.

En administratör kan lista alla användare i systemet. Administratören kan ta bort andra användare, men inte sig själv eftersom systemet då skulle kunna bli utan administratör.



Figur 7.10: Vy som visar användare i systemet

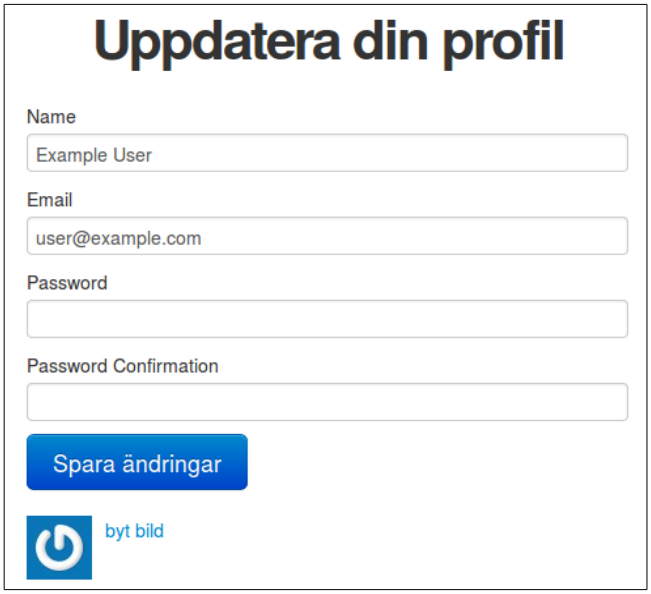
Dock saknas fortfarande stöd för att administratör ska kunna skapa nya användare eller ge dem administratörsrättigheter. Figur 7.10 är ett exempel på en sådan vy.

Via det sista menyalternativet, ”Mitt konto”, kan användaren nå sin egen profilsida. Här

visas användarens namn, om administratörsrättigheter finns och användarens profilbild, se Figur 7.11. Om användaren har rättighet till att skapa bokningar syns även de senaste bokningarna till höger i vyn.

Figur 7.11: Personlig vy för varje användare

Varje användare har också en inställningssida där man kan ändra namn, mejladress och lösenord samt ställa in visningsbild, se Figur 7.12.

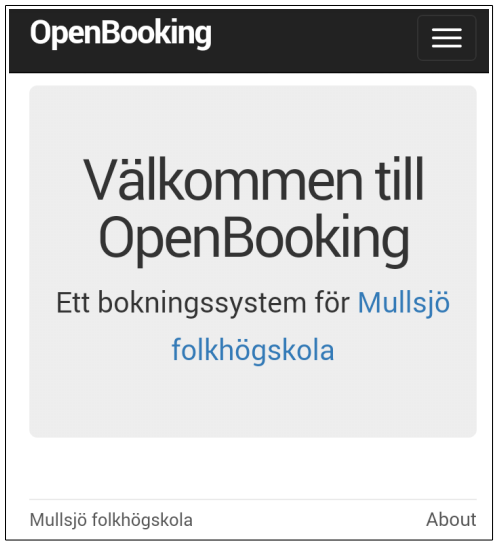


Figur 7.12: Vy för personlig profil

7.1 Responsivitet

Bokningssystemets grafiska gränssnitt är responsivt, det vill säga att det anpassar sig efter enheten som läser sidan. Är skärmbredden liten så anpassar sig elementen därefter, varför bokningssystemet är användbart även på enheter med mindre skärmupplösning, såsom en

smartphone eller en surfplatta. Figur 7.13 visar en skärmdump tagen direkt från en smartphone. Istället för att menyn visas i överkant visas en knapp med tre vertikala sträck på längst upp till höger. Då användaren klickar på denna knapp fälls en menyn ner.



Figur 7.13: Skärmdump på förstasidan av bokningssystemet från en smartphone

7.2 Testning

Resultatet av den testdrivna utvecklingen går att mäta. Appendixfigur 5 visar ett utdrag ur en rapport som beskriver hur stor procent av alla programrader som är testade och i skrivande stund är 99.73% av totalt 1498 kodrader täckta.

7.3 Kravspecifikation

I resultatet ingår också den kravspecifikation som tagits fram under projektet. Denna kan ses i Appendix III – Kravspecifikation.

8 DISKUSSION

Detta kapitel för en diskussion kring projektet och dess resultat. Den använda metoden och systemdesignen kritiseras, följt av några synpunkter på hur det är att utveckla mjukvara på egen hand såsom i detta projekt. Sist ges förslag på fortsatt utveckling för systemet samt hur det kan generaliseras och användas i andra sammanhang.

8.1 Resultatet

Målet med projektet var att implementera de mest fundamentala delarna av ett bokningssystem för att hantera arrangemangsbokningar för Mullsjö folkhögskola. OpenBooking är inget färdigutvecklat bokningssystem, men en god grund som är kapabelt att sköta både inloggning, hantering av användare och administratörer, kunder, rum och byggnader. I systemet kan enklare bokningar skapa och man kan se beläggningen i bokningskalendern. Det är webbaserat, förhindrar dubbelbokningar och fungerar för flera samtidiga användare. Systemet bidrar också till smidigare kommunikation mellan berörda personalgrupper för kurs- och konferensbokningar då alla kan komma åt bokningsinformationen i realtid. Dessa krav finns samlade i Appendix III – Kravspecifikation och som det är nu så uppfyller OpenBooking denna specifikationen. Dock har kravspecifikationen liksom systemet utvecklats i en iterativ process och är i behov av fortsatt utveckling innan den beskriver ett färdigt system. För att se vad systemet fortfarande saknar se avsnitt 8.5.

Att kravspecifikationen inte är fullständig beror också på att det har varit en stor utmaning att formalisera bokningsprocessen på folkhögskolan. Personalen hade god kunskap om vad som inte fungerade med nuvarande lösning och de hade många förslag på hur ett nytt system kunde fungera, men själva processen har visat sig mer komplicerad än vad vi först trodde. Appendixfigur 5 i Appendix I – Diagram och figurer illustrerar komplexiteten genom att visa hur en vandrarhemsbokning sker. Liknande flödesscheman skulle behövas för att beskriva resterande bokningar. Ett problem som uppstod vid formaliseringen var att vissa begrepp som användes var överlagrade. Exempelvis användes ordet ”bokning” för en bokning av arrangemang men också för bokning av rum. Det fanns också redan ett annat bokningsprogram, i rapporten kallat ”schemaläggningsprogrammet”, som redan användes för viss ”bokning av rum”. Vidare så visade det sig att de tre olika typerna av arrangemangsbokningar som personalen från början beskrev inte riktigt räcker till för att beskriva verkligheten. Exempelvis är en bokning av folkhögskolans gymnastiksal varken en konferensbokning, en kursbokning eller en vandrarhemsbokning. Formaliseringen av bokningsprocessen är alltså inte färdig.

Det mer generella målet med projektet var att utreda om det går att utveckla ett relativt avancerat mjukvarusystem under en kort tidsperiod med hjälp av moderna metoder och verktyg. Resultatet av projektet blev ett enkelt men fungerande bokningssystem. Ramverket som använts har bidragit till att utvecklingen gick snabbt, men man kan ändå konstatera att mjukvaruutveckling fortfarande är komplicerat. Ett berömt citat från F. J. Brooks lyder ”The complexity of software is an essential property, not an accidental one.” [33]. Även om ramverk kan bidra mycket till processen så är det fortfarande ofta en komplex verklighet som ska beskrivas.

Ett syfte med projektet var det egna lärandet. Jag tycker mig fått en god inblick i hela

processen kring mjukvaruutveckling och hur det är att arbeta agilt och testdrivet. Att få utveckla något som är tänkt att användas i verkligheten och att ha en kund som har egna åsikter och tankar har gett unika erfarenheter som andra kurser under studieperioden på Chalmers inte kunnat ge.

För att kommentera omfattningen på examensarbetet kan det enligt Chalmers förväntas vara det arbete som en erfaren ingenjör utför på fyra heltidsveckor [34]. Jag anser att det för- och planeringsarbete, utarbetandet av kravspecifikationen, inlärning av programmeringsspråk och ramverk samt påbörjad implementering och testning är ett fullt rimligt resultat efter fyra heltidsveckor.

Resultatet av den testdrivna utvecklingen kan ses i Appendixfigur 5, vilken visar att nästan samtliga av alla rader programkod i bokningssystemet är testade. Testet säger visserligen ingenting om *hur* en kodrad är testad utan bara *att* raden har blivit anropad när testerna kördes. Trots att inte detta ger någon svar på om det är bra tester eller inte så visar det på att testdriven utveckling verkligen främjar testningen. Vidare diskussion om testningen först i avsnitt 8.2 och 8.4.

8.2 Metodkritik

Det har varit positivt att arbeta både agilt och testdrivet. Metodkapitlet citerar några utdrag från det agila manifestet [1]. Nedan följer dessa utdrag och kommentarer kring hur det gick att arbeta efter dem.

> *”Vår högsta prioritet är att tillfredställa kunden genom tidig och kontinuerlig leverans av värdefull programvara.”* Redan i tredje mötet med folkhögskolan kunde en mycket simpel modell av bokningssystemet visas. Istället för att bara diskutera kring abstrakta idéer, tankar, illustrationer och diagram kunde dialogen utgå ifrån något verkligt och handfast och personalen kunde komma med återkoppling och tycka till om den faktiska mjukvaran.
>
> *”Välkomna förändrade krav, även sent under utvecklingen.”*. Genom de kontinuerliga och personliga mötena vi haft samt den agila metoden att arbeta i iterationer har det varit möjligt att hantera förändrade krav under projektets gång. Samtidigt har det varit viktigt att i samtal med folkhögskolan även diskutera vad som är möjligt att genomföra då kravlistan annars kan bli orealistisk.
>
> *”Leverera fungerande programvara ofta”*. Den testdrivna utvecklingsmetoden har bidragit till denna punkt. Istället för att testa programmet när det är klart har testningen skett under utvecklingens gång vilket bidragit till att det som demonstrerats för folkhögskolan faktiskt har fungerat.
>
> *”Verksamhetskunniga och utvecklare måste arbeta tillsammans”*. Enligt personalen på folkhögskolan har de känt sig mycket delaktiga i projektet och tyckt att kommunikationen fungerat bra. Kontakten och mötena har varit regelbundna och det har funnits möjlighet att ställa frågor om hur verksamheten fungerar. I och med att jag varit ensam i processen har jag som haft kundkontakten också varit den som utvecklat mjukvaran. Detta har varit positivt då jag inte behövt förlita mig på dokumentation som andra har skrivit utan istället kunnat använda mig av förstahandsinformation direkt från de verksamhetskunniga.

”Kommunikation ansikte mot ansikte är det bästa och effektivaste sättet att förmedla information”. Även om vi har haft några kortare möten över Internet så har vi försökt prioritera faktiska möten på plats. Den kontinuerliga och personliga kontakten med folkhögskolan har varit mycket positiv och verkligen främjat utvecklingen av systemet.

”Fungerande programvara är främsta måttet på framsteg.” Detta blir mycket tydligt då man har kontakt med dem som faktiskt är intresserade av att använda mjukvaran i slutändan. Det är trots allt för dem som utvecklingen sker. Att programvaran faktiskt har fungerat vid demonstration beror till stor del på TDD.

Naturligtvis finns det även nackdelar och utmaningar med den agila arbetsmetoden. En viktig fråga att ställa sig är om kunden kan få för mycket att säga till om. Det är lätt att börja diskutera funktioner som inte är möjliga att implementera när kunden får tänka helt fritt, då denna ofta saknar inblick i hur implementationsprocessen sker. Det är viktigt att beskriva och vägleda kunden i samtalet för att inte skapa en overklig bild av vad mjukvaran kommer att kunna då utvecklingen är färdig. En annan utmaning är den iterativa arbetsprocessen som innebär att man kontinuerligt behöver modellera om och omstrukturera kod. Dock är TDD en mycket bra tillgång vid omstrukturering då testbasen kan användas som en indikator på om den omskrivna koden fortfarande gör vad den ska.

Något att tillägga om användandet av TDD är att många buggar och fel i programmet kunnat upptäckas redan under testning. Vid provkörning av bokningssystemet har det sällan varit saker som inte fungerat, vilket gör att systemet känns stabilt. Det är också en motiverande arbetsmetod då det ger en känsla av att man kommer framåt när testerna går igenom. För att kritisera metoden kan nämnas att det inte är optimalt att samma person både testar och implementerar då det kan vara svårt att hitta fel i kod som man själv har skrivit. Därav bör TDD kompletteras med tester skrivna av andra personer. TDD kan också leda till en falsk trygghet då det inte finns något som säger att testerna faktiskt är tillräckliga.

Vid projektets början fanns ingen erfarenhet av vare sig Ruby eller Rails. Att inlärningen kunnat ske så snabbt beror mycket på Michael Hartls kompetenta bok [13], speciellt då utvecklingen av bokningssystemet kunde pågå parallellt med inlärningsprocessen.

Med tanke på att resurserna för detta projekt har varit mycket små är jag imponerad av hur långt implementeringen av systemet är gången. Att jag hunnit med så mycket implementering är mycket tack vare ramverket Rails. Rails två grundprinciper, DRY och ”convention over configuration”, har gett stor effekt och lett till tidsbesparing. Naturligtvis tar det tid att lära sig själva ramverket, men besparingen i stort är ändå klart övervägande. Jag har ännu inte märkt att ramverket har begränsat utvecklingen på något sätt.

8.3 Systemdesign

Man kan tycka att det är onödigt att bokningssystemet har hand om lokalreservationer då det redan finns ett schemaläggningsprogram på folkhögskolan som kan reservera lokaler. Ett förslag till folkhögskolan var att använda dessa program parallellt; skapa bokningar i bokningssystemet och reservera lokaler i schemaläggningsprogrammet. Detta förslag var dock inte intressant för administratörerna för kurs- och konferensverksamheten dels för att de inte vill arbeta i flera system samtidigt och dels för att schemaläggningsprogrammet inte har alla de funktioner som önskas vid lokalbokning för kurs- och

konferensverksamheten, såsom kostnad och koppling till arrangemanget.

Något annat som kan diskuteras angående systemdesignen är användandet av tredjepartslösningar. Redan från början stod det klart att sådana lösningar var helt nödvändig för att kunna genomföra projektet. Som det beskrivs i avsnitt 5.7 bidrar detta till systemdesignen, vilket kan tyckas som en sårbarhet. Vad händer om tillverkaren av tredjepartslösningen plötsligt upphör med utvecklingen? I webbsammanhang blir tekniker ofta snabbt omoderna vilket skulle kunna leda till att system som använder tredjepartslösningen blir oanvändbara. Samtidigt kan det tyckas vara en styrka; istället för att lägga tid på att uppfinna hjulet igen så används redan färdig källkod. Något som bör poängteras är den positiva effekten med att använda tredjepartslösningar som är skrivna i öppen källkod. Är det så att tillverkaren lägger ner utvecklingen kan man helt enkelt välja att själv föra utvecklingen av produkten framåt. Samtidigt finns det ofta flera andra på Internet som också är villiga att fortsätta med produkten och finns det inte det så är anledningen ofta att det finns något bättre alternativ att använda istället

En annan effekt med öppen källkod är att det finns möjlighet att vid behov göra eventuella ändringar av koden. Som avsnitt 6.2 beskriver saknade tillägget FullcalendarEngine vissa funktioner, vilket då kunde åtgärdas. Nackdelar med att göra på detta vis är att den tillagda koden naturligtvis måste underhållas om FullcalendarEngine uppdateras. Det kan naturligtvis också vara så att tillverkaren av FullcalendarEngine valde att inte implementera funktionen från början av en anledning.

Som avsnitt 6.1 beskriver är klassen *Booking* en mycket central del i bokningssystemet och använder sig av tillägget FullcalendarEngine för att kunna visa bokningar i bokningskalendern. Man kan ifrågasätta att en så central klass använder en tredjepartslösning, som dessutom innebär att klassen *Booking* inte längre själv håller grundläggande information om bokningen såsom tid och titel. För att lösa informationsproblemet skulle man kunna tänka sig att klassen *Booking* istället ärver från *Event*. Då hade klassen haft titel och tid för bokningen men det skulle innebära att kopplingen till FullcalendarEngine blir ännu hårdare. Kopplingen till FullcalendarEngine går inte att komma ifrån om man vill återanvända kod. I detta fall råkar det vara en central klass som använder sig av tredjepartslösningen.

8.4 Mjukvaruutveckling på egen hand

Mjukvaruutveckling är allmänt känt som en komplicerad process. Att utföra utvecklingen på egen hand gör inte saken enklare. Jag har under projektets gång uppmärksammat vikten av att ha någon att diskutera med, fråga, tänka högt och resonera med. Bara att behöva förklara för någon annan hur man tänker gör att man får tänka igenom sin idé och hitta felaktigheter och brister i den. Beslut i systemdesignen är avgörande för hur bra implementeringsarbetet går och det är verkligen en nackdel att vara ensam i designarbetet. I detta projekt har jag tagit stor hjälp av min handledare för att diskutera systemarkitektur och designval, något som utan detta stöd hade varit mycket svårt och utmanande.

8.5 Fortsatt utveckling

Det är olyckligt att de andra system som finns på folkhögskolan inte har något kommunikationsgränssnitt för tredjepartsapplikationer. Detta gör att automatisk kommunikation med dessa system inte är möjlig. Avsnitt 5.2.2 beskriver dock att det

eventuellt kommer att utvecklas en plattform till schemaläggningsprogrammet som folkhögskolan använder sig av. Men även om denna plattform skulle utvecklas och tillåta både skrivning och läsning av data skulle en automatisk kommunikation inte kunna gå att implementera direkt. Schemaläggningsprogrammet körs nämligen lokalt på folkhögskolan och data skickas regelbundet till en central server hos tillverkaren. Det är informationen från denna centrala server som en eventuell plattform skulle kunna använda sig av. Schemaläggningsprogrammet skickar dock inte uppgifter i realtid till servern utan endast någon gång per dygn och dessutom är det inte riktigt all information som skickas. Om bokningssystemet skulle delegera uppgiften att reservera lokaler skulle kommunikationen mellan schemaläggningsprogrammet och den centrala servern behöva ske i realtid då det annars finns risk för dubbelbokningar och den informationen som skickas skulle behöva kompletteras.

Det är naturligtvis minst lika olyckligt att bokningssystemet självt inte har kommunikationsgränssnitt mot tredjepartsprogram. Ett bra exempel på vad man bör implementera i framtiden är ett API för att hämta och läsa data från bokningssystemet. Det ska dock tilläggas att bara databasen som bokningssystemet använder sig av är öppen för fler användare så går det att nå informationen därifrån.

Då OpenBooking inte är färdigt än finns många utvecklingsmöjligheter. Bland annat borde det finnas en knapp på sidan där nya bokningar skapas som kan filtrera ut de rum som är bokade vid angivet datum. Vidare behöver systemet på något sätt särskilja mellan de olika arrangemangstyperna. De olika bokningstyperna borde också ha olika färg i bokningskalendern för att visuellt visa skillnaden. Bokningar behöver också någon sorts status för om de bara är preliminära. Systemet kan inte heller räkna ut vad rummen som är bokade kostar sammanlagt, vilket det borde kunna. En annan stor brist som finns är att användarhanteringen inte riktigt är färdig. Administratörer måste lätt kunna skapa nya användare och även göra andra användare till administratörer vid behov. Ytterligare en brist är att resurser i systemet än så länge bara kan läggas till och tas bort, det finns alltså inget stöd för att uppdatera eller ändra information än.

8.6 Generalisering

Bokningssystemet är visserligen byggt för att passa situationen på Mullsjö folkhögskola men samtidigt är systemet generellt designat och bör vara användbart för andra folkhögskolor och organisationer som hanterar bokningar likt folkhögskolan i Mullsjö. Informationen som hör till en bokning är allmängiltig; titel, beskrivning, start- och stopptid, och de resurser som sedan kopplas till en bokning skapar systemadministratören själv. Det behöver alltså inte vara just rum som bokas som i folkhögskolans fall, resurserna kan lika gärna vara personal, utrustning, fordon och så vidare.

8.7 Hållbarhet

Som ingenjör är det viktigt att ha med det nutida hållbarhetstänket i arbetet. I detta projekt finns ett tydligt hållbarhetsproblem i form av stressig arbetsmiljö. Projektet visar också hur ett sådant problem kan åtgärdas med hjälp av mjukvara. I nuvarande lösning för att hantera arrangemangsbokningar finns ingen automatisk kontroll för att förhindra dubbelbokningar eller för att verifiera uppgifterna i en bokning, vilket leder till onödig stress för personalen. Ett annat problem är att bokningsinformationen inte finns tillgänglig för alla berörda personalgrupper. Istället måste hela informationsflödet gå via kurs- och

konferensansvariga vilket skapar en onödig arbetsbörda som inte tillför verksamheten någon nytta. Det nya bokningssystemet löser ovan nämnda problem och ger personalen på folkhögskolan en mer hållbar arbetsbelastning. På liknande sätt skulle mjukvara kunna användas på andra arbetsplatser för att förbättra arbetsmiljön för människor i stort.

9 SLUTSATS

Detta examensarbete har resulterat i ett enkelt men fungerande bokningssystem kallat OpenBooking. Systemet har många grundläggande funktioner för att hantera bokningar samtidigt som det fortfarande fattas en del för att kunna användas i ett verkligt sammanhang. Systemet är anpassat för kurs- och konferensverksamheten på Mullsjö folkhögskola.

Utvecklingsarbetet har skett enligt agila metoder vilket har varit mycket positivt. Det ledde till att projektet kunde komma igång snabbt och redan på tredje mötet med folkhögskolan fanns en början till en produkt att utgå ifrån och diskutera kring. Något som är viktigt att ha i åtanke när man arbetar agilt är att vara tydlig mot beställaren om vad som är möjligt att genomföra inom projektets ram. Öppenheten för förändring som den agila metoden medför kan annars leda till att kravspecifikationen blir alltför omfattande och orealistisk att implementera. Den kontinuerliga kontakten med kunden, ansikte mot ansikte, är otroligt viktig och personalen på folkhögskolan säger att de känt sig mycket delaktiga i processen tack vare detta.

Användningen av TDD har bidragit till att många fel i programkoden har kunnat upptäckas tidigt i utvecklingen, vilket lett till att bokningssystemet som har utvecklats är stabilt och fungerande. Nästintill samtliga kodrader i bokningssystemet har täckts av tester, vilket är ett mycket högt resultat. Vidare gör den testdrivna utveckling att man kan lita på att det som skapats faktiskt fungerar som det är tänkt vilket underlättar mycket vid demonstration av produkten.

Det moderna ramverket Ruby on Rails som är designat för snabb utveckling har i detta projekt visat sig vara ett effektivt hjälpmedel. Trots detta har projektet visat att mjukvaruutveckling ändå är komplicerat då det ofta är en komplex verklighet som ska efterliknas. Speciellt svårt är det att utveckla på egen hand jämfört med att arbeta i grupp. Att göra systemdesignen själv, se bristerna i sina egna idéer och hitta lösningar för dem kan vara mycket svårt och åtminstone i detta projekt har jag varit beroende av hjälp från andra för att lösa sådana utmaningar.

Eftersom projektet i hög grad varit verklighetsbaserat har det gett erfarenheter som tidigare kurser under studieperioden varit oförmögna till. Speciellt intressanta och lärorika har de många mötena med folkhögskolans personal varit.

10 REFERENSER

[1] The Agile Alliance, ”Manifest för Agil systemutveckling”, *agilemanifesto.org*. [Online] Tillgänglig: http://agilemanifesto.org/iso/sv [Hämtad: 15 dec, 2014].

[2] The Agile Alliance, ”Independent Signatories of The Manifesto for Agile Software Development”, *agilemanifesto.org*. [Online] Tillgänglig: http://agilemanifesto.org/sign/ [Hämtad: 15 dec., 2014].

[3] K. Beck, Test Driven Development: By Example, Portland: Addison-Wesley Longman, 2003.

[4] L. Williams, E. M. Maximilien, M. Vouk. “Test-Driven Development as a Defect-Reduction Practice”, *14th Int. Symposium on Software Reliability Engineering, 2003. ISSRE 2003.*, ss 34-45, 2003.

[5] Robert C. Martin, “Professionalism and Test-Driven Development”, IEEE Software, vol. 24, nr. 3, ss 32-26, 2007.

[6] David S. Janzen, Hossein Saiedian. “Does Test-Driven Development Really Improve Software Design Quality?”, IEEE Software, vol. 25, nr. 2, ss 77-84, 2008.

[7] Officiell hemsida för Ruby. [Online]. Tillgänglig: https://www.ruby-lang.org/en [Hämtad: 12 dec., 2014].

[8] S. Cass, ”Top 10 Programming Languages”, IEEE Spectrum. [Online] Tillgänglig: http://spectrum.ieee.org/computing/software/top-10-programming-languages [Hämtad: 12 dec., 2014].

[9] ”Getting Started with Rails”, *RailsGuides*. [Online] Tillgänglig: http://guides.rubyonrails.org/getting_started.html [Hämtad 17 dec., 2014].

[10] ”Active Record Migrations”, *RailsGuides*. [Online] Tillgänglig: http://edgeguides.rubyonrails.org/active_record_migrations.html [Hämtad: 6 jan., 2015].

[11] ”Active Record Associations”, *RailsGuides*. [Online] Tillgänglig: http://guides.rubyonrails.org/association_basics.html [Hämtad: 6 jan., 2015].

[12] *The Ruby Toolbox*. [Online] Tillgänglig: https://www.ruby-toolbox.com [Hämtad: 28 nov., 2014].

[13] M. Hartl, *Ruby on Rails Tutorial*. [Online] Tillgänglig: https://www.railstutorial.org [Hämtad: 28 nov., 2014].

[14] Officiell hemsida för Ruby on Rails. [Online] Tillgänglig: http://rubyonrails.org [Hämtad: 12 dec., 2014].

[15] ”Intermediate Rails: Understanding Models, Views and Controllers”,

b*etterexplained.com*.
[Online] Tillgänglig: http://betterexplained.com/articles/intermediate-rails-understanding-models- views-and-controllers [Hämtad: 17 dec., 2014].

[16] Python Software Foundation, ”The Python Tutorial”, *docs.python.org*.
[Online] Tillgänglig: https://docs.python.org/2/tutorial/ [Hämtad: 16 dec., 2014].

[17] Python Software Foundation, ”History and License”, *docs.python.org*.
[Online] Tillgänglig: https://docs.python.org/2/license.html [Hämtad: 16 dec., 2014].

[18] Officiell hemsida för Python
[Online] Tillgänglig: https://www.python.org [Hämtad: 12 dec., 2014].

[19] Python Software Foundation, ”Index of Packages”, *pypi.pyhon.org*.
[Online] Tillgänglig: https://pypi.python.org/pypi/Django [Hämtad: 16 dec., 2014].

[20] Django Software Foundation, d*jangoproject.com*.
[Online] Tillgänglig: https://www.djangoproject.com [Hämtad: 12 dec., 2014].

[21] Twitter, Inc., ”Bootstrap Readme”, *GitHub*.
[Online] Tillgänglig: https://github.com/twbs/bootstrap [Hämtad: 28 nov., 2014].

[22] Twitter, Inc., ”Bootstrap”, *Getbootstrap*.
[Online] Tillgänglig: http://getbootstrap.com [Hämtad: 12 dec., 2014].

[23] A. Shaw, ”FullCalendar Readme”, *GitHub*.
[Online] Tillgänglig: https://github.com/arshaw/fullcalendar [Hämtad: 28 nov., 2014].

[24] A. Shaw, ”FullCalendar”, *FullCalendar*.
[Online] Tillgänglig: http://fullcalendar.io [Hämtad: 12 dec., 2014].

[25] Vinsol, ”Fullcalendar Engine Readme”, *GitHub*.
[Online] Tillgänglig: https://github.com/vinsol/fullcalendar-rails-engine [Hämtad: 28 nov., 2014].

[26] Officiella hemsida för Git.
[Online] Tillgänglig: http://www.git-scm.com/ Hämtad: 6 jan., 2015].

[27] V. Driessen, ”A successful Git branching model”, *nvie.com*.
[Online] Tillgänglig: http://nvie.com/posts/a-successful-git-branching-model/
[Hämtad: 6 jan., 2015].

[28] Bestwebframeworks, bestwebframeworks.com.
[Online] Tillgänglig: http://www.bestwebframeworks.com/ [Hämtad: 6 jan.,2015].

[29] Wikipedia, ”Comparison of web application frameworks”, wikipedia.org.
[Online] Tillgänglig:
https://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks
[Hämtad: 6 jan., 2015].

[30] D. Greenfield, A. Roy, Two Scoops Press, ”Django Packages”, *Django Packages*.

[Online] Tillgänglig: https://www.djangopackages.com [Hämtad: 28 nov., 2014].

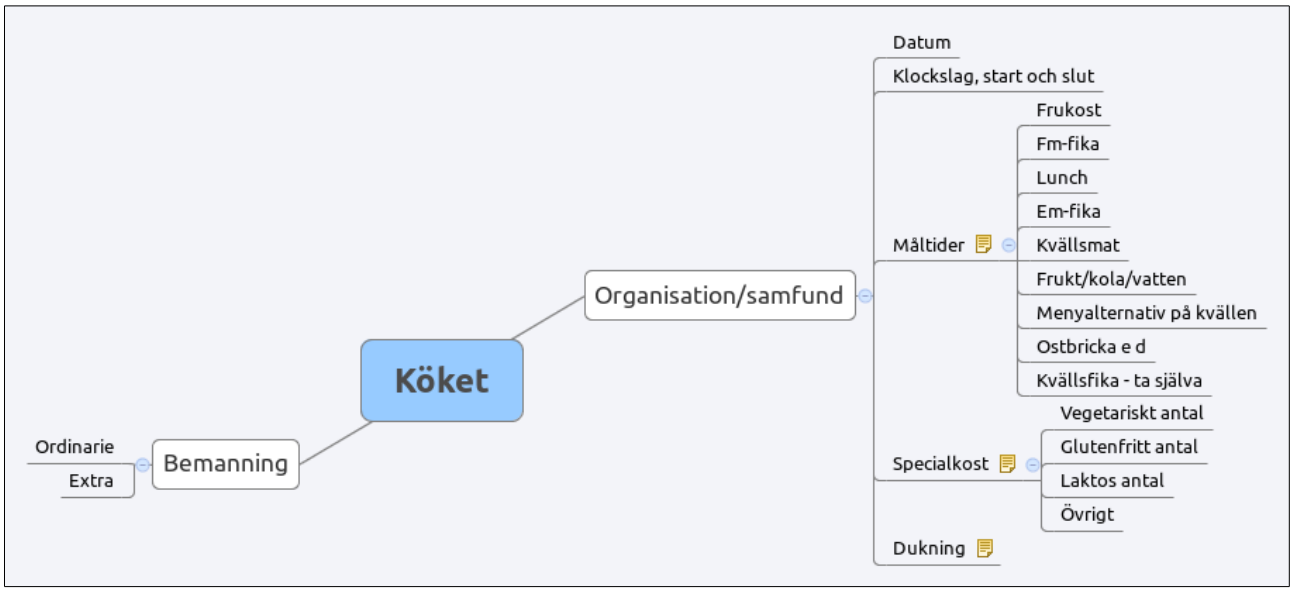
[31] ”Python2orPython3”, The *Python Wiki*.
[Online] Tillgänglig: https://wiki.python.org/moin/Python2orPython3 [Hämtad: 28 nov., 2014].

[32] Hannes Elvemyr, ”OpenBooking”, *GitHub*.
[Online] Tillgänglig: https://github.com/ehannes/OpenBooking [Hämtad: 2 feb, 2015].

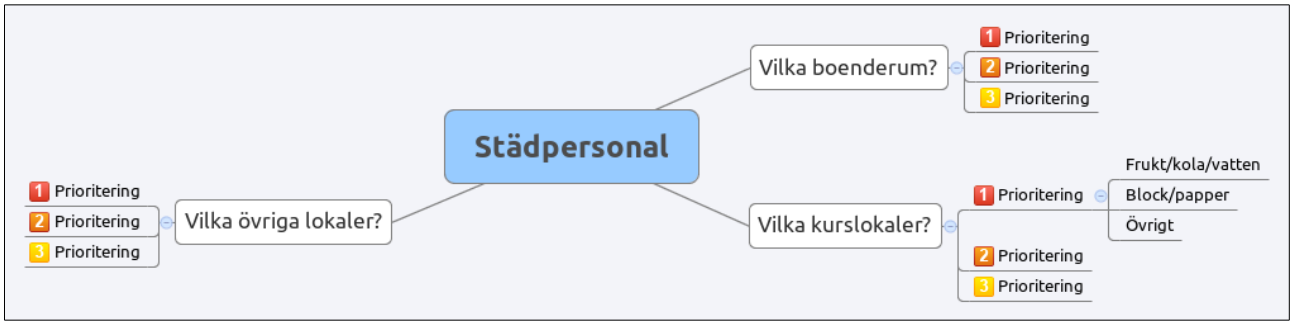
[33] F.P., Jr. Brooks, “No Silver Bullet Essence and Accidents of Software Engineering”, Computer, vol 20, no 4, ss 11, 1987.

[34] L. Svensson, ”Kort om examensarbeten på högskoleingenjörsprogram samt om förslag till sådana arbeten”, Version 1.0, September 10, 2014.

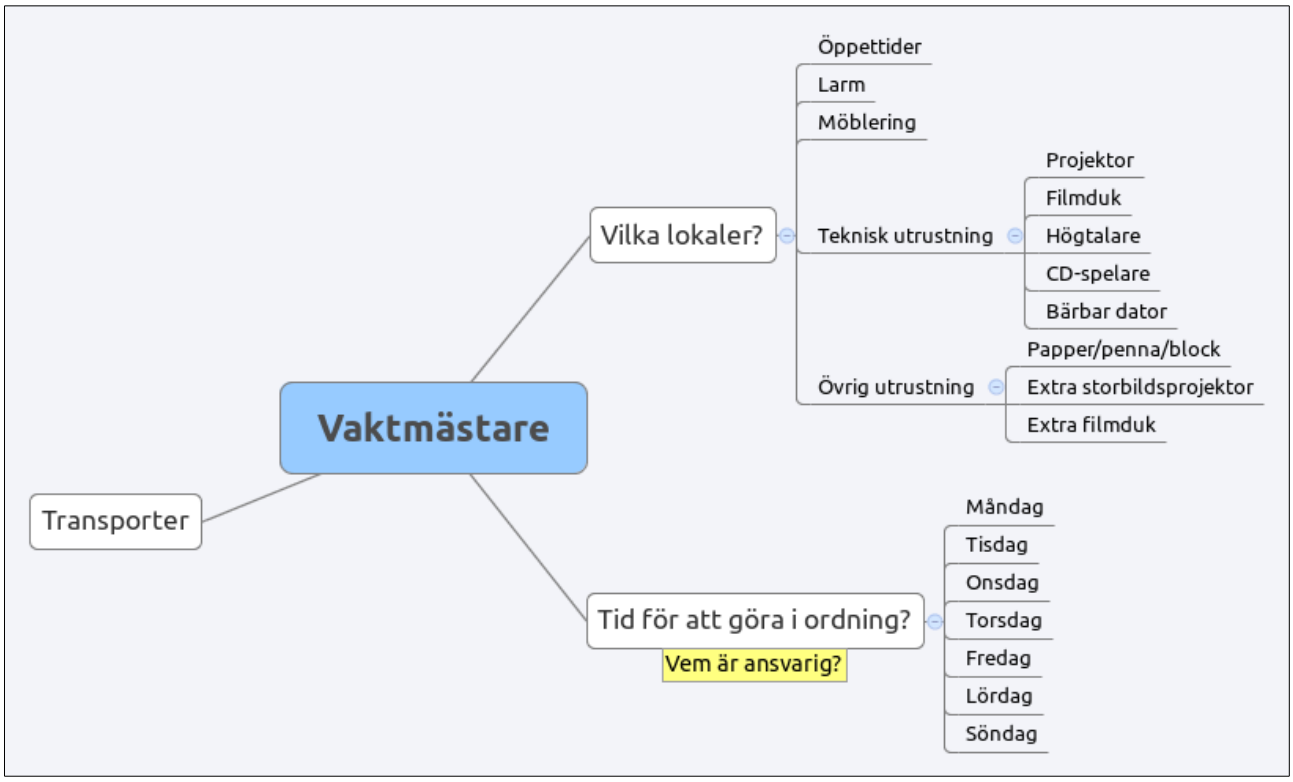
APPENDIX I – DIAGRAM OCH FIGURER



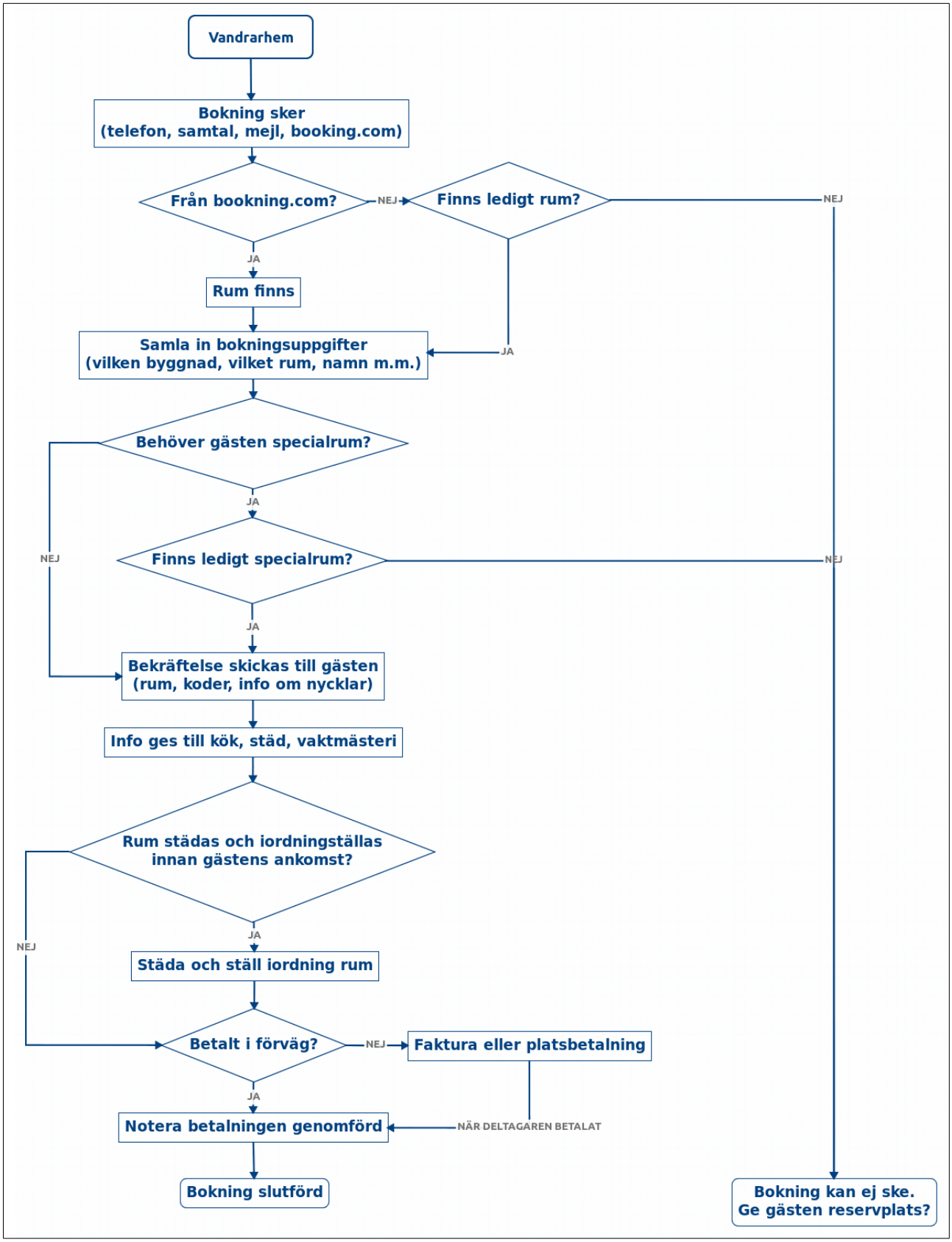
Appendixfigur 1: Exempel på information om arrangemang som kökspersonalen kan behöva. Bild från Hans Noreliusson.



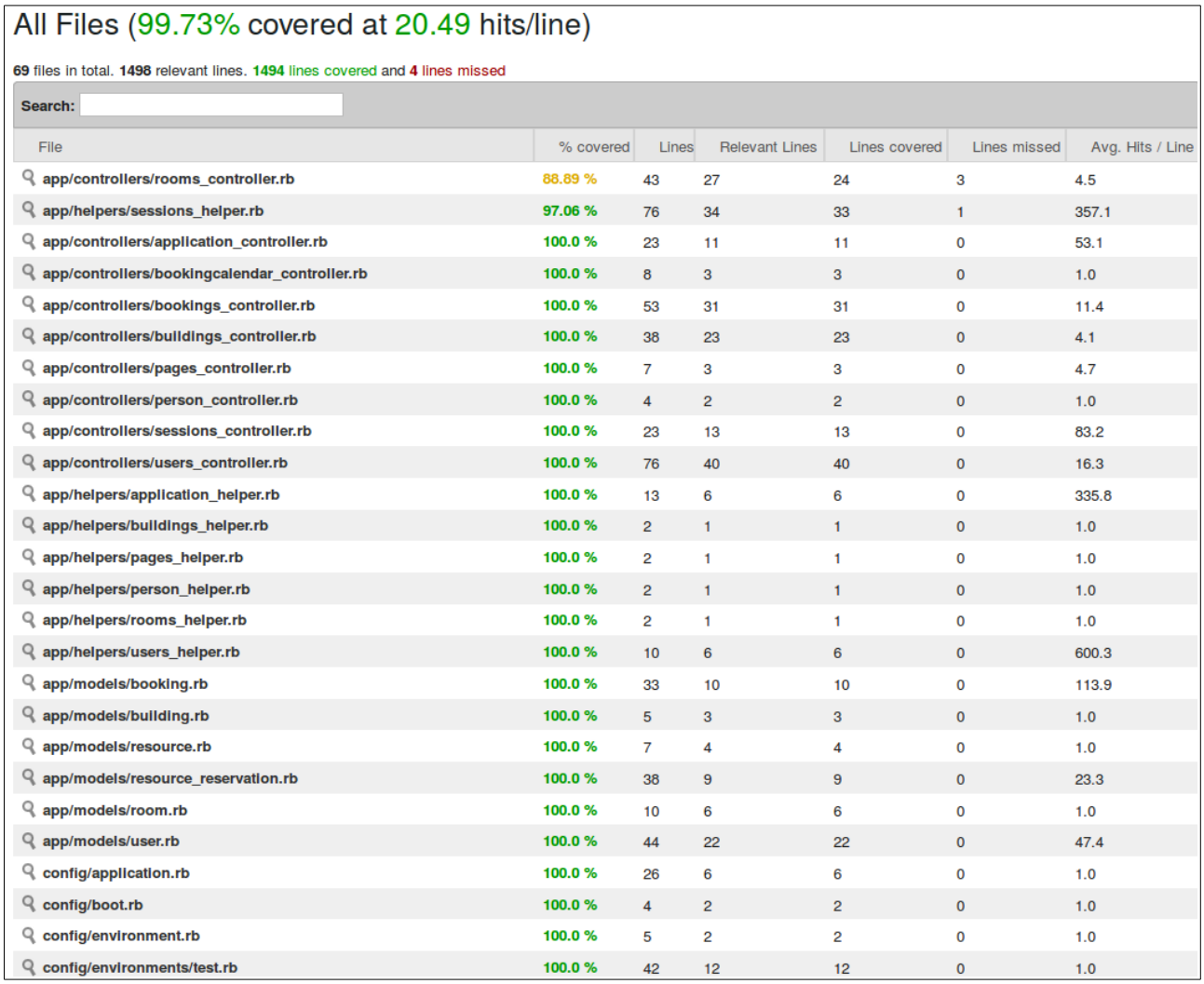
Appendixfigur 2: Exempel på information om arrangemang som städpersonal kan behöva. Bild från Hans Noreliusson.



Appendixfigur 3: Exempel på information om arrangemang som vaktmästarna kan behöva. Bild från Hans Noreliusson.



Appendixfigur 4: Flödesschema över vandrarhemsbokningar på Mullsjö folkhögskola



All Files (99.73% covered at 20.49 hits/line)

69 files in total. 1498 relevant lines. 1494 lines covered and 4 lines missed

Search:

File	% covered	Lines	Relevant Lines	Lines covered	Lines missed	Avg. Hits / Line
app/controllers/rooms_controller.rb	88.89 %	43	27	24	3	4.5
app/helpers/sessions_helper.rb	97.06 %	76	34	33	1	357.1
app/controllers/application_controller.rb	100.0 %	23	11	11	0	53.1
app/controllers/bookingcalendar_controller.rb	100.0 %	8	3	3	0	1.0
app/controllers/bookings_controller.rb	100.0 %	53	31	31	0	11.4
app/controllers/buildings_controller.rb	100.0 %	38	23	23	0	4.1
app/controllers/pages_controller.rb	100.0 %	7	3	3	0	4.7
app/controllers/person_controller.rb	100.0 %	4	2	2	0	1.0
app/controllers/sessions_controller.rb	100.0 %	23	13	13	0	83.2
app/controllers/users_controller.rb	100.0 %	76	40	40	0	16.3
app/helpers/application_helper.rb	100.0 %	13	6	6	0	335.8
app/helpers/buildings_helper.rb	100.0 %	2	1	1	0	1.0
app/helpers/pages_helper.rb	100.0 %	2	1	1	0	1.0
app/helpers/person_helper.rb	100.0 %	2	1	1	0	1.0
app/helpers/rooms_helper.rb	100.0 %	2	1	1	0	1.0
app/helpers/users_helper.rb	100.0 %	10	6	6	0	600.3
app/models/booking.rb	100.0 %	33	10	10	0	113.9
app/models/building.rb	100.0 %	5	3	3	0	1.0
app/models/resource.rb	100.0 %	7	4	4	0	1.0
app/models/resource_reservation.rb	100.0 %	38	9	9	0	23.3
app/models/room.rb	100.0 %	10	6	6	0	1.0
app/models/user.rb	100.0 %	44	22	22	0	47.4
config/application.rb	100.0 %	26	6	6	0	1.0
config/boot.rb	100.0 %	4	2	2	0	1.0
config/environment.rb	100.0 %	5	2	2	0	1.0
config/environments/test.rb	100.0 %	42	12	12	0	1.0

Appendixfigur 5: Rapport som visar hur stor del av programkoden som är testad. Bilden visar endast ett utdrag ur rapporten.

APPENDIX II – MÖTESANTECKNINGAR

Medverkande på möten med Mullsjö folkhögskola:
Anna Sandin - *Kurs- och konferensansvarig*
Hans Noreliusson - *Kurs- och konferensansvarig, biträdande rektor*
Sven-Olof Dahlin - *Ekonomichef*

Hannes Elvemyr var med på samtliga möten och mötesplats var alltid Mullsjö folkhögskola om inte annat nämns. Då utvecklingen började står även vilken agil iteration som redovisades på mötet.

Första mötet, 140702

Närvarande: Hans, Sven-Olof

Samtal fördes kring projektet, dess omfattning och utförande. Mullsjö folkhögskola beslutade att delta, främst i form av tid från personal. Hans blir utsedd till kontaktperson på folkhögskolan.

Andra mötet, 140917

Närvarande: Hans

Hans informerar om situationen, nuvarande lösning och problem med denna. Vidare diskuteras vad en bokning är för något och hur en sådan sker. Beslut tas om att först fokusera på bokningstypen ”konferens” och ett flödesschema ritas upp för att beskriva informationsflödet. Detta resulterade senare i ett liknande flödesschema för vandrarhemsbokningar, se Appendixfigur 4 i Appendix I – Diagram och figurer. Till sist diskuteras det nya bokningssystemet, funktioner och utformning vilket resulterade i en skiss över vilka objekt systemet ska behöver arbeta med. Denna skiss användes sedan som grund för att göra ett första UML-diagram för programmet, se Error: Reference source not found i Appendix I – Diagram och figurer.

Tredje mötet, 141009

Närvarande: Anna, Hans
Iteration: 1

En kort demonstration ges av en första mycket enkel version av bokningssystemet. Det hade vid mötet en statisk förstasida utan funktionalitet. Diskussion fördes kring vyer i programmet och det framarbetades en skiss av hur en vy för bokning av konferenser kan se ut.

Fjärde mötet, 141106

Närvarande: Anna, Hans
Iteration: 2

Mötet börjar med en demonstration av nya funktioner i programmet, bland annat autentiseringssystem, användare och administratörer. Sedan visas vad som är på gång till nästa demonstration, nämligen att kunna skapa enkla bokningar.

Diskussion hölls kring de olika rollerna, vanliga användare och administratörer, i systemet. Det beslutas att kurs- och konferensansvariga och förmodligen receptionen ska ha administratörskonton. Därefter ska olika personalgrupper, exempelvis vaktmästare, städ och kök, ha vanliga användarkonton, ett per personalgrupp. Grundtanken är att vanliga användare ska kunna läsa i systemet medan administratörer ska kunna göra nya bokningar och ändra i dem som finns. Vidare diskuteras och utvecklas UML-diagrammet för programmet.

Beslut tas att om möjligt använda samma formulär för att skapa alla olika sorters bokningar. När man sedan får upp formuläret är tanken att man först där kan välja vilken bokningstyp man vill skapa.

Femte mötet, 141121

Närvarande: Anna, Hans
Mötesplats: Mötet hölls över Internet
Iteration: 3

Ett kort möte som hölls över Internet. Den nya bokningsfunktionen demonstreras; man kan nu lägga till väldigt enkla bokningar i systemet. En bokning har bara en titel och en förklaring. Efter det visas pågående utveckling av en kalenderfunktion för att visa gjorda bokningar i systemet. Diskussion förs kring hur kalendern bör fungera.

Sjätte mötet, 141205

Närvarande: Anna, Hans
Mötesplats: Mötet hölls över Internet
Iteration: 4

Ännu ett kort möte där nya funktioner förklarades och nästa möte bokades in. Funktioner som presenterades var att bokningar automatiskt läggs in i ett kalendarium samt att man kunde lägga till byggnadsobjekt till bokningsprogrammet. Till dessa byggnadsobjekt ska sedan rum associeras och på så sätt ska man kunna hitta ett rum som man vill reservera.

Sjunde mötet, 141218

Mötet ställdes in då rapportskrivandet prioriterades framför utvecklingen i denna fas av projektet.

APPENDIX III – KRAVSPECIFIKATION

- För att skapa en ny arrangemangsbokning i systemet behövs följande information:
    - Start- och sluttid samt titel och beskrivning.
    - Koppling till vem kunden för arrangemanget är.
    - Vilka rum som ska bokas.
- Byggnader i systemet håller följande information:
    - Byggnadsnamn.
- Rum i systemet håller följande information:
    - Vilken byggnad rummet tillhör.
    - Namn på rummet samt pris och kommentar.
- Användare i systemet håller följande information:
    - Namn, e-mejl och lösenord.
- Kunder i systemet håller följande information:
    - Namn.
    - En kontaktpersons förnamn, efternamn, telefonnummer, mobilnummer och adress.
    - Vilka bokningar i systemet som en kund står för.
- Systemet ska hantera två olika nivåer av användare:
    - Administratörer – kan skapa nya användare, göra bokningar, lägga till nya resurser.
    - Vanliga användare – kan i regel läsa information men inte göra ändringar.
- Vidare ska bokningssystemet klara följande koncept:

- Automatiskt förhindra dubbelbokningar genom att meddela användaren att bokningen ej är möjlig.
- Tillåta flera samtidiga användare.
- Vara webbaserat och nås via en modern webbläsare.
- Autentisera användare för att skydda bokningssystemet från obehöriga. För autentisering ska en unik registrerad mejladress samt tillhörande lösenord användas.
- Samtliga berörda personalgrupper i kurs- och konferensverksamheten behöver kunna nå bokningsinformation från systemet.