1DV411 – Webbprojekt I  
Slutrapport

KartApp med datalagring  
***En webbapplikation med tillhörande datalagring***

*Jonathan Wilsson  
Jesper Tullberg,  
Alexandra Johansson  
2013-03-25*

1 Sammanfattning

Denna rapport beskriver det projektarbete som vi utförde för kursen Webbprojekt 1DV411, vid Linnéuniversitetet i Kalmar. Kursledaren delade in kursdeltagarna i grupper som sedan blev tilldelade projekt.Vår grupp fick möjligheten att jobba med Linnéuniversitet som uppdragsgivare vilket var både roligt och utmanande.

Rapporten beskriver vår arbetsgång med de hinder vi stött på, de tekniker vi använt oss av och det slutgiltiga resultatet.

Projektet gick ut på att bygga om Linnéuniversitets kart applikation.  
Det innebar att effektivisera datalagringen, skriva ett publikt API[[1]](#footnote-1), göra om det befintliga användar gränssnittet samt göra ett administrativt gränssnitt för att förenkla arbetet för de som administrerar kartans olika positioner. Den tidigare applikationen användes i huvudsak av studenter och övriga som vill hitta på campus.

2 Förord

Projektet har gett oss studenter möjlighet både att öka vår kunskap rent programmatiskt men framför allt har det gett oss en inblick i vad det innebär att jobba med ett större projekt i grupp. Vi vill tacka Linnéuniversitetet, vår handledare Emil Carlsson, Mats Loock, Sven-Åke Johansson, Tobias Ohlsson, och alla andra som har hjälp oss under arbetets gång.

Innehållsförteckning  
  
1 Sammanfattning..................................................................................................1

2 Förord .................................................................................................................1

3 Inledning/Bakgrund 3

4 Syfte/Mål 4

5 Projektorganisation 5

6 Genomförande ................................................................................................ 7

6. 1 Metodik 7

6.2 Teknik 8

6.2.1 Databas 8

6.2.2 Resultat databas 10

6.2.3 API 11

6.2.4 Resultat API 11

6.2.5 Administratörs gränssnitt 12

6.2.6 Resultat Administratörsgränssnsitt 13

6.2.7 Användargränssnitt 14

6.2.8 Resultat användargrässnitt 15

7 Resultat/Måluppfyllelse 16

8 Avikelse/Eferkalkyl 17

9 Förslag på vidare utveckling 18

10 Eventuell övertagande organisation................................................................ 18

3 Inledning/bakgrund

Vår grupp utgörs av Alexandra Johansson, Jesper Tullberg och Jonathan Wilsson.

Linnéuniversitetet har sedan en tid tillbaka en fungerande applikation som underlättar navigeringen runt Kalmar-, Växjö- och Pukeberg campusområde.

Den tidigare lösningen innefattade en Iphone- och en Androidapplikation samt en javascript baserad webbapplikation som låg inbäddad på Linnéuniversitets hemsida.  
Dessa applikationer använde sig av samma datalagring, två excelfiler[[2]](#footnote-2), som med hjälp av ett Ruby[[3]](#footnote-3) script omvandlades till en textfil som sedan applikationerna hämtade sin data från.

Denna datalagringsmodell skapade flera problem. Bland annat så fick excelfilerna fungera ett som administrativt gränsnitt vilket medförde att mycket felaktig data sparades, då det inte fanns någon form av validering. Utöver detta så upplevde administratörerna att det var svårt att veta hur datan skulle föras in, med tanke på att de inte fick hjälp att hämta ut t.ex. kartpositioner mm.

Den tidigare applikationen var utvecklad under knappa tidsförhållande med en stram budget vilket skapade ineffektiva lösningar som ledde till dessa problem.

4 Syfte och mål

Projektet kom att bestå av att lösa tidigare nämnda problem.

Kundens grundkrav för projektet var

- Nytt system för datalagring med MsSQL[[4]](#footnote-4) Databas i grunden

- Administrativt gränssnitt i ASP.NET MVC 4[[5]](#footnote-5), för underhåll av datan i databasen

- Publikt RESTful[[6]](#footnote-6) API i ASP.NET MVC 4, för lätt åtkomst av datan i databasen

- Nytt användargränssnitt i ASP.NET MVC 4 / Javascript[[7]](#footnote-7), som använder API:et för dataåtkomst

Grundkraven på databasen, utöver teknikval, var att den skall vara så optimerad och dynamisk som möjligt. T ex så framkom det av kund önskemål om att kunna lägga till nya typer av platser, t ex parkeringsplatser, vilket dåvarande modell inte stödde.

Grundkraven för det administrativa gränssnittet var att det skulle vara intuitivt att jobba i, t ex att lägga till ny data skall vara enkelt och att datan ska valideras för att undvika fel.  
Övriga teknikkrav var inloggningsfunktionalitet med Windows Authentication [[8]](#footnote-8).

Grundkraven  för API:et var att det ska vara öppet och RESTful. Det ska vara enkelt att använda API:et och det ska erbjuda tydlig APIdokumentation[[9]](#footnote-9).

Grundkraven för användar gränssnittet var att det ska uppfylla funktionalitet från tidigare applikation samt använda sig utav det nya API:et

Tillsammans i gruppen satte vi också upp önskvärda krav, bland annat en filtreringsfunktion och vägbeskrivning. Detta efter att vi studerat andra liknande system. Efter att riskerna för dessa krav utvärderades beslutade vi att genomföra kraven.

5 Projektorganisation

**Experter**

Patrik Granlöv, systemutvecklare på it-avdelningen samt SCRUM-master[[10]](#footnote-10).  
Karl Hermansson, systemutvecklare på it-avdelningen, ansvarig testmiljö.

**Projektdel Övervakare                     Ansvarig**

API                           Alexandra                        Jesper

Användargränssnitt        Jesper                               Jonathan

Admingränssnitt Jesper Alexandra/Jesper

Databas                      Jonathan                           Alexandra

Testning                     Alexandra                         (resp. del)

Dokumentation           Jonathan                              Grupp

Mjukvaruarkitektur   Jesper                              Jesper/Alexandra

Projektredovisning     Grupp                               Jesper/Alexandra

Rapport Grupp Jesper/Alexandra

Kundkontakt         Projektledare                    Projektledare

**Iterationer**

Varje vecka har en ny projektledare utsetts, detta för att alla i gruppen ska ha testat på att utföra de uppgifter som krävs av en projektledare. Det har också satts fasta kundmöten, handledarmöten och gruppmöten.

**Samarbete**

Vi upplever att informationsflödet mellan grupp och kund har fungerat mycket bra. Vi har haft en väldigt kunnig och engagerad kund som har bidragit med mycket kunskap och varit väl involverade. I ett tidigt skede av projektet så sattes fasta tider för kundmöte, vilket har underlättat kommunikationen. Vi har fått väldigt bra konstruktiv feedback som vi från vecka till vecka har kunnat bygga vidare på.

Det uppkom en del problem som vi inte kunde förutse och därav inte förebygga, Vi ser detta som en naturlig del utav processen. Det var bl a ändrade krav från kund och ändringar som inte blev godkända av kund.

Redan i starten av projektet kom vi som grupp överens om att jobba mycket tillsammans när det gäller dokumentationen, för att alla hela tiden skulle vara uppdaterade med varandras arbetsområden och eventuella problem.

Alla i gruppen är distanselever och det har därför bitsvis varit väldigt svårt att hitta tider som passar alla. Detta blev svårare och svårare ju längre vi kom på projektet och gruppmöten samt kundmötena blev lidande av detta och det resulterade i en klyfta i arbetet. Vi känner att arbetsbelastningen blev större pga utav detta, men vi anser att det hade varit omöjligt att hantera detta på något annat sätt efter omständigheterna.

6 Genomförande: metodik, teknik

**6.1 Metodik**

Efter att i grupp diskuterat de olika kraven från kund så delades ansvarsområden ut. Vi diskuterade vem som kände sig bekvämast i vilken roll och delade upp arbetet därefter. Turligt nog så hade vi väldigt bra gruppdynamik vilket gjorde att de olika arbetsområderna kändes självklara. Vi gjorde delar av mjukvaruarkitekturen, för att vid implementation ha en grund att stå på, ett samspelt flöde och en klar översikt av projektets  alla olika delar, och hur mycket tid som varje del krävde. Vi gjorde en väldigt grov tidsuppfattning på projeketets olika delar, vilket senare visade sig inte stämma.

Projektet utfördes i veckovis iterationer där ett gruppmöte, kundmöte och handlednings möte var inplanerat samt att vi hade gemensam grupptid då vi jobbade tillsammans. Mot slutet blev arbetet mer individuellt då specifika krav och delar skulle implementeras. Vi satte upp mål och önskvärda resultat för varje iteration och planerade vår tid därefter. Vid slutet av varje iteration, i samband med den veckans kundmöte skedde en leverans. Vid varje leverans fick vi feedback från kund på tester som de utfört och eventuella buggar eller önskvärda ändringar som de hittade. Utvecklartester[[11]](#footnote-11) utfördes kontinuerligt under implementationen, unittester[[12]](#footnote-12) ströks för att vi upplevde att det inte fanns ett behov av det samt att eftersom systemen var i ständig förändring så minskar det ett unittests effektivitet då det måste uppdateras i takt med systemet. I slutfasen av implementationen utfördes användartester[[13]](#footnote-13) utefter den testspecifikation som tagits fram. Detta visade på ytterligare buggar och brister som åtgärdades innan slutleverans. Risker som kom upp under varje iteration diskuterades, prioriterades och omprioriterades efter behov. Under hela projektet har vi arbetat med versionhantering via Github[[14]](#footnote-14).

**6.2 Teknik**

6.2.1 Databas

Databasen är en MsSQL databas.

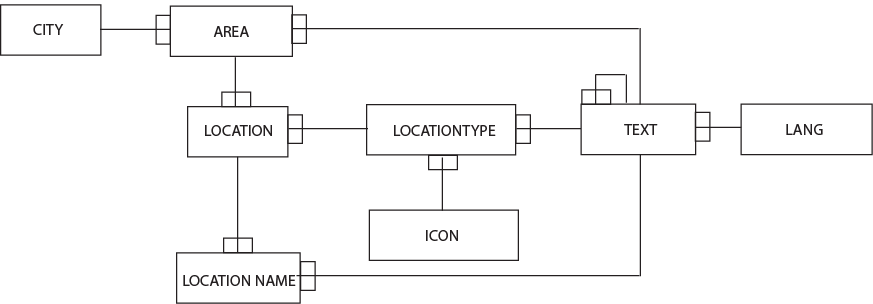
I tidigt skede intervjuades kund för att ta fram en så effektiv och optimerad databasmodell som möjligt. Konverteringen från tidigare excelfiler till den nya databasen innebar många problem. Som kunden också upplevt fanns det mycket felaktig data i filerna, när det kom till att hitta de faktiska objekt som skulle utgöra tabeller i den nya modellen så var det svårt att hitta relationer och samband i datan pågrund av detta.

För att göra arbetet enklare så lades de två filerna ihop till ett dokument och importerades till en databas. Därefter kunde datan undersökas lättare. Trots detta så framkom det ytterligare fel som skett vid importeringen då excelfilernas alla rader inte var utformade på samma sätt. T ex så hade vissa fält angivit lattitud och longitud med hjälp utav komma tecken istället för punkter vilket medförde att dessa fält i databasen blev null. Dessa fel upptäcktes vid en jämförelse av excelfilerna och den importerade datan nu i databasen. Många småfel och tolkningar av datan ledde till att det tog lång tid att definiera och ta fram koncept till databasmodellen.

Efter att ha analyserat datan i databasen så gjordes en normalisering[[15]](#footnote-15) av den. Det som fastställdes då var vilka fält som för skulle bli egna objekt, dvs tabeller, och deras relationer. Det undersöktes också vilka beroende som fanns mellan de olika fälten och kunden fick förklara mycket angående logiken i datan, varför den såg ut som den gjorde och hur det i verkligenheten fungerade.

Till en början så byggdes en databasmodell med huvudobjekten Byggnad och Rum, men detta byttes sedan ut mot det mer dynamiska objekten Area och Location. En Area är ett område som kan innehålla flera olika platser. Detta medförde att t ex ett fält i tabellen Location, inte behövde vara just ett rum utan kunde också vara t ex en parkeringsplats. Det var just den typ av dynamik som kunden hade efterfrågat. Kunden upptäckte också under ett kundmöte att den var tillräckligt dynamisk för att också kunna kartlägga personalen med hjälp utav denna databasmodellen. De talades även en hel del från deras sida att de i framtiden skulle kunna intergrera databsen med bla deras personaldatabas.

Ett annat stort problem var att excelfilen innehöll översättningar. Detta var något som gruppen till en början beslutade att stryka från databasen. Vi resonerade att det istället skulle hanteras utav backend lagret i applikationerna med t ex en översättningfil. Men från kund fick vi feedback om att detta skulle finnas med i databasen, vilket försvårade databasmodellen rejält. Eftersom bara visa objekt fält skulle översättas så ledde detta till en mycket komplex databasmodell*(se bild 1, komplex modell).* Detta resulterade i långa svarstider och mycket svår hantering av datan i bland annat backend lagret för det administrativa gränssnittet, samt API. Detta var ett utav de största problemen som vi hade under projeketets gång. Det gick åt mycket tid att få fram en databasmodell som var optimerad, utformad på ett korrekt sätt och samtidigt uppfyllde kundens krav, men det gick också mycket tid på att göra ändringar som automatiskt blev ärvda ner i de andra redan påbörjade lagrena.

*Bild 1 komplex modell*  
  


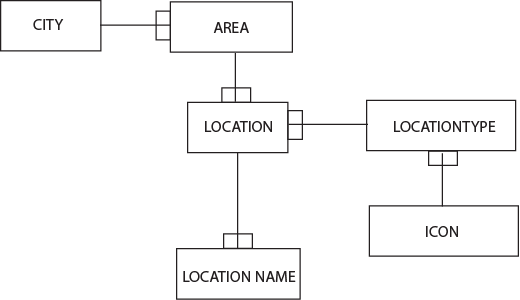
Efter handledning och långa diskussioner med bl a Mats Loock och Sven-Åke Johnasson och framtagandet av olika koncept för databasmodellen, så ändrades databasen till sitt slutgiltiga format*(se bild 2, slutgiltig modell)*. Istället för att översättningen dras ut till ett eget objekt så läggs två namnfält till på de objekt som ska kunna översättas. T ex Area har fälten area\_swe och area\_eng. Även om modellen inte är korrekt utformad enligt normalformerna[[16]](#footnote-16) då det finns dubeletter i fälten och tabellen kan anses som växande[[17]](#footnote-17), så är den optimerad i effektivitet, snabbhet och efter kundens önskemål. Detta sätt att utforma databasen på hade inte fungerat vid en stor datamängd eller t ex om det fanns en önskan från kund att snabbt och enkelt kunna lägga till översättning av fler språk än engelska, vilket har diskuterats och uteslutits med kunden. Om gruppen själva hade fått styra översättningarna och dess hantering hade detta skett i applikationerna istället. Det hade gett en mer korrekt databasmodell och besparat oss mycket tid i form av omarbetning.

Till en början hade vi en vision om att jobba mot så mycket SPROCS[[18]](#footnote-18) som möjligt, men tog sedan beslut om att behålla den logiken i applikationerna istället. Dels för att kunden lättare skulle få en överblick över koden och vad som händer i alla lagrena, men också för att vi ansåg att Entity Framework[[19]](#footnote-19), som vi jobbade med i ASP:NET MVC 4 gav oss mycket av den funktionaliteten.

SPROCS användes dock för att föra in gammal data från excel filerna. Excel filerna sammanfogades till ett dokument och konverterades med hjälp av zamzar.com[[20]](#footnote-20) Till ett xml dokument. Av xml dokumentet så användes ASP.NET MVC 4 och SPROCS för att föra in den gamla datan. Även här möttes vi av problem som den felaktiga datan i excel filerna orsakade. Mycket felsökning krävdes för att importera frisk data till den nya databasen och modifikationer av den gamla datan var nödvändig.

6.1.2 Resultat databas

Det var svårt att hitta en lagom nivå mellan en korrekt databas och en databas som uppfyllde kundens krav. Men till slut lyckades vi hitta en lösning som uppfyllde kundens önskemål, men behöll en rimlig nivå av komplexitet och effektivitet. Vi lade ner alldeles för mycket tid på lagrade procedurer i början av projektet, men just i det läget ansågs det vara viktigt. Sammantaget är det svårt att utesluta sådana hinder utan att veta projektets alla delar i detalj.

*Bild 2 slitgiltig modell*  
  


6.1.3 API

API:et är RESTful och skrivet i ASP.NET MV4 WEB API[[21]](#footnote-21). Från början var det tänkt att det skulle hantera alla typer av anrop, alltså hämta, skapa, ta bort och uppdatera data eftersom att det administrativa gränssnittet skulle använda sig av API:et för datahanteringen. Detta innebar att en autentisieringslösning var tvunget till att finnas för att skydda datan. Vi såg det som en stor risk då vi inte har jobbat med den typen av autentisiering som kunden önskade tidigare.

Vi påpekade för kund att de hade önskat ett öppet API[[22]](#footnote-22) och att autentisiering hindrade den visionen, och kom slutligen överrens om att stryka det och i API:et endast stödja anrop i form av att hämta data och att det administrativa gränssnittet istället jobbar direkt mot databasen.  
  
API:et gav till en början väldigt långsamma svar vid anrop som en följd av ooptimerad datahantering i Entity Framework. Detta berodde dels på den komplexa databasmodellen men även till viss del på bristande kunskap om ramverket. I och med att databasen ändrades kunde mycket ut av komplexiteten skalas bort och svarstiderna ses nu som mycket goda med en max svarstid på 3 millisekunder.   
  
Det framkom en hel del problem under utvecklingen. Det som tog mest tid var routing[[23]](#footnote-23) inställningar, att svara med rätt svarskoder samt namngivningen på svarselementen. Detta berodde dels på att tekniken var ny för oss men även för att vi fick strikta krav från kund på hur de ville att förfrågningar och svar skulle utformas.

6.1.4 Resultat API  
Eftersom att API:ets storlek krympte avsevärt då anrop för att ta bort, skapa och uppdatera togs bort så tog det relativ liten del av projekttiden.

Både vi och kunden är nöjda med slutresultatet och det uppfyller alla krav som kunden ställde på det.

6.1.5 Administratörs gränssnitt

Vi arbetar genomgående med ASP.NET MVC 4 och följer MVC strukturen. Utöver att vi arbetat efter MVC så är appliaktionen uppbyggd som en flerlagersapplikation[[24]](#footnote-24) med dataåtkomst lager, service lager, affärslogik lager mm. Vi arbetar med ett generisk repositorium[[25]](#footnote-25) för att minska kodduplicering.

Till en början skulle gränssnittet jobba mot API:et för datahantering, vilket genom diskussion med kund ändrades. Vi ansåg att det inte fanns anledning att använda API:et för detta, utan skulle endast öka komplexiteten och utomstående beroenden. Gränssnittet jobbar nu direkt mot MsSQL databasen.

Det administrativa gränssnittet började implementeras när databasen hade sin komplexa modell och det gjorde som ovan nämnt att det var mycket svårjobbat. Detta var en av de stora tidstjuvarna i projektet och många timmar lades ner på att försök optimera och felsöka en svårjobbad kod. För att tex jobba med en plats så var det tvunget att kombinera flera olika tabeller. Ett fält i tabellen Location består av följande kolumner: lattitud, longitud, våning och för att få ut resterande information som platstyp, stad, namn, och vilket område som platsen ligger i, så har Locationtabellen relationer till tabellerna Area, City, LocationName, LocationType. Den komplexa modellen hade en egen tabell som då höll alla texter(namn) och översättningar till dessa tabeller. Detta innebar att applikationen var tvungen att göra väldigt många databasanrop för att få ut en plats och all dess information, vilket gav långa svarstider. Efter att modellen ändrades så kunde också koden skalas av ordentligt och det skapade en enklare och mer läsbar kod.

Controllers[[26]](#footnote-26) med tillhörande CRUD[[27]](#footnote-27) vyer skapades och därefter jobbade vi med att hantera själva utseendet. Från kunden fick vi tillång till ett annat administrativt gränssnitt som de använde sig av och ville vi kunde härma gällade utseendet.Vi använde en del javascript, dels för att få ett dynamiskt utseende på när man tar bort och lägger till flera namn till en och samma plats, men också för att kunna lägga till koordinater till ett område eller plats. För att hämta koordinater så använder vi oss av Google Maps API[[28]](#footnote-28). Gränssnittet har modifierats flera gånger rent utseendemässigt efter feedback från kund för att det ska fungera så smidigt som möjligt för administratörer att skapa, redigera och ta bort data från databasen.

6.1.6 Resultat Adminsitratörs gränssnitt

Vi hade underskattat tidsåtgången för det administrativa gränssnittet och dess komplexitet. Detta medförde att utvecklingstiden överskred det som vi hade planerat och avsatt. Optimering av kod, framför allt vyer och css, blev bortprioriterat pga tidsbrist. Gränsnittet uppfyller dock alla kundens krav, och har rättats enligt kundens feedback.  Kunden var väldigt specifik i sin feedback vilket var både bra och dåligt. Någonting som kunden var väldigt specifik om var utseendet vilket medförde att vi inte själva behövde lägga speciellt mycket tid på det, dock så var de väldigt specifika i vissa avseende gällande funktionalitet och beteende som inte vi ansåg som bra lösningar vilket orsaka stora problem i sena iterationer vilket inte var bra. I många av fallen handlade det om att kunden inte var öppen för förändring, dock tycker vi att allt löste sig. Vi upplever dock gränssnittet som mycket stabilt och fullständigt vid slutleverans och kund blev mycket nöjd.

6.1.7 Användargränssnitt

Användargränssnittet är i grunden en ASP.NET MVC 4 applikation men arbetar nästan uteslutande med JavaScript mot API:et och sköter allting på klientsidan. Den centrala delen i användargränssnittet är en karta från Google som vi har implementerat med hjälp av Google Maps API. Google Maps API erbjuder även funktionalitet för vägbeskrivningar som vi har valt att använda oss av. Allt för att besökaren på ett så enkelt sätt som möjligt ska hitta på Linnéuniversitetet.

Kunden ville till en början att vi skulle göra en applikation som liknande den som fanns och var inbäddad på hemsidan sedan tidigare. Den applikationen hade inte en väldigt hög funktionalitet och vi höll inte med om att det var den optimala lösningen för en karttjänst. Tillsammans med kunden kom vi fram till nya krav för funktionaliteten. När funktionaliteten ökade så kände vi också att den skulle behöva mer plats än vad den skulle få som inbäddad i hemsidan, för att användningen fortfarande skulle vara lätt.

Vi valde att undersöka detta närmare genom att i visionen undersöka andra liknande system. För att kunna avgöra vilket som passade bäst så undersökte vi både en fristående webbsida med helskärmsläge från Stanford, och en från Karlstad Campus som hade en karta som var inbäddad i själva hemsidan. Vi upplevde att den som Karlstad Campus tillhandahöll fungerade väldigt bra och mycket utav det som gjorde att det fungerade så bra var att karta inte var dynamisk utan betod av en bild. När vi sedan tittade på Stanfords karta som mer liknade den karta och funktionalitet som var tänkt för det gränssnitt som vi nu skulle utveckla, så upplevde vi att det skulle vara svårt att få samma användarupplevelse i en inbäddad variant med lika mycket funktionalitet. Vi tog efter diskussion tillsammans med kund ett beslut om att bygga gränssnittet med layout efterliknande Stanfords kartapplikation. Därför är applikationen idag fristående. Tillsammans med övriga riktlinjer för utseendet som vi fick av kund, gällande mer generella delar så som teckensnitt och färger, så jobbade vi fram ett gränssnitt.

Stora delar av strukturen och rent kodmässigt har det fungerat bra i utvecklingen. Mycket av detta beror på att vi i stor utsträckning har använt oss av välkända ramverk och bibliotek som exempelvis jQuery samt tidigare nämnda Google Maps API. Dessa har gjort så att vi inte behöver tänka så mycket på hur olika webbläsare beter sig utan fokus har kunnat ligga på att bygga ett gränssnitt med mycket välanpassad användarfunktionalitet. Då det kommer till den responsiva designen har det dock varit en del problem. Dels blev ena spalten där rutan för sökningar och filter väldigt smal om man använde sig av en smal skärm och problem då det inte fungerade alls på Android. Detta kan ha berott på att vi inte använde oss av något ramverk för detta utan skrev all CSS[[29]](#footnote-29) från grunden. Dessa problem löstes genom att låta innehållet staplas på höjden vid en tidigare brytpunkt än vad som var tänkt från början.

6.1.8 Resultat Användargränssnitt

Ett av de största problemen med användargränssnittet har varit kommunikationen mot API:t, Det har skett mycket förändringar i det som i sin tur har krävt förändringar i användargränssnittet. Dessa förändringar berodde på ovannämnda ändringar i kraven för databasen.

Tyvärr så dröjde det väldigt lång tid innan vi kunde visa användargränssnittet för kunden vilket medförde att det kom väldigt mycket feedback på detta mot slutet av projektet. Det blev då väldigt mycket att göra eftersom vi skulle implementera alla krav samt göra ändringar efter kundens feedback. Det hade självklart varit bättre om vi hade kunnat visa det i ett tidigare stadie men det var inte möjligt på grund av tidigare nämnda problem samt bristande kommunikation i gruppen.

Trots detta är kunden väldigt nöjd med användargränssnittet och alla deras krav är uppfyllda.

7 Resultatbeskrivning/måluppfyllelse

Överlag är vi väldigt nöjda med vad vi har lyckats utveckla under projekttiden. Vi ser det som ett väldigt stabilt och dynamiskt system som ska kunna användas en längre tid. Vi anser att vi genomgående har haft en god kommunikation och en bra dokumention att utgå ifrån. Det som har underlättat mest under projektets gång har varit en nära kommunikation inom gruppen men också med kunden, detta har lett till en mycket bra feedback och därmed projektdelar som är mycket genomtänkta och testade. Vi har diskuterat om att arbetsfördelningen har varit lite skev och det har medfört att projektet till vissa delar har känts underskattade rent tidsmässigt samt att projektet tidvis har känts väldigt stort och att vi hade kunnat avgränsa och avsäga oss vissa uppgifter.

Vi upplever att vi har haft lite problem med att hitta ett effektivt sätt att hålla dokumentationen uppdaterad. Efter en reflektion över problemet inser vi att en helt webbbaserad hantering av dokument, tex google docs, hade varit bättre än att versionshantera diverse office dokument via Github.

Vi tycker överlag att kunden har varit tydlig i sina krav på uppgiften och att dom har engagerat sig i projektet. Vi anser att vissa delar hade kunnat lösas på ett bättre sätt, men då  kunden inte varit så pass öppen för förändringar som vi hade önskat, så har problemen fått lösas på ett mer komplexare sätt än vad vi hade velat.  
Kunden är nöjd med projeket och ser detta som en fungerande och bra lösning på deras problem. Dom har haft ett stort förtroende för oss och vår förmåga att lösa uppgiften.

Överlag i gruppen känner vi att våra programmatiska kunskaper har ökat, vi som har suttit med det administrativa gränssnittet anser att våra kunskaper i ASP.NET MVC har ökat och att vi fått en tydligare förståelse i hur ramverket fungerar. Den del av gruppen som suttit med databasen anser också att kunskaperna kring MsSQL och SPROCS är större nu än vid projekt start.   
  
Handledningar och övrig kurs dokumentation har fungerat väldigt bra och som ett stöd i planeringsarbetet.

8 Avvikelser/efterkalkyl

Under projektets gång har vi gjort många ändringar i kraven, dels krav som har ansetts som ouppnåeliga pga utav tidsramen för projektet, dels pga att det inte funnits behov för det som kraven inneburit, och dels för att applikationen har ändrats och därmed också kraven.

För det administrativa gränssnittet har kravet om att kunna uppdatera, skapa och ta bort en stad tagits bort. Detta efter att vi i diskussion med kund kom fram till att det endast skulle medföra en ökad risk att administratören utför felaktiga kommandon. Då Linnéuniversietet inte expanderar till nya städer väldigt ofta så ansåg vi att den delen istället ska gå att utföra med vissa programmerings skickligheter direkt i databasen.

Till en början skulle också det administrativa gränssnittet använda sig av API:et för att hantera data, vilket efter diskussion med kund ändrades till att gränssnitttet istället skulle jobba direkt mot databasen. Gruppen upplevde att en lösning där gränssnittet jobbade mot API:et bara medförde en ökad komplexitet och att API:et användes på fel sätt för att det då skulle fungera som en slags säkerhet mot databasen. Detta innebär också att kravet för inloggning på API:et togs bort eftersom det inte längre skulle kunna gå att göra anrop av den typ som skulle kunna vara skadlig för databasen.

Kunden ville också inledningsvis att vi skulle bygga om deras nativeapplikationer till IPhone och Android så att dom använde den nya datalagringen. Vilket vi i ett tidigt stadie insåg var helt utanför tidramen för projektet, så detta kravet ströks.

9 Förslag på vidareutveckling  
Kunden har redan uttryckt ett önskemål om att vidareutveckla applikationen, både att den kommer att växa med ny data samt att de vill koppla ihop den med andra tjänster som t ex deras personalsökningsapplikation. Vi har strävat efter att skriva koden på ett så dynamiskt sätt som möjligt för att underlätta detta jobb för kunden och att en ordentlig överlämning med kodgenomgång genomförs.

10 Eventuell övertagande organisation

Applikationen kommer att lämnas över till Linnéuniversitetet.

Enligt överenskommelse har universitetet fri tillgång till att vidareutveckla och distribuera systemet fritt.

1. Application programming interface. [↑](#footnote-ref-1)
2. Microsoft office Excel fil format. [↑](#footnote-ref-2)
3. Backend (språk som arbetar på servern) programmeringsspråk [↑](#footnote-ref-3)
4. Microsoft SQL server databas. [↑](#footnote-ref-4)
5. Ett MVC (model,view controller) ramverk för C#/HTML version 4. [↑](#footnote-ref-5)
6. Representational State Transfer, ett typ av protokoll för API [↑](#footnote-ref-6)
7. Frontend (språk som arbetar på klienten) programmeringsspråk [↑](#footnote-ref-7)
8. Automatiskt autentifiering mellan Microsoft Internet Information Services och andra Active Directory medvetna applikationer [↑](#footnote-ref-8)
9. Dokumentation som förklarar funktionalitet hos API:et för användaren. [↑](#footnote-ref-9)
10. Projektledare SCRUM utveckling (typ av systemutvecklingsprocess). [↑](#footnote-ref-10)
11. Tester som utfördes iterativt av utvecklaren under implementation. [↑](#footnote-ref-11)
12. Tester som systemet automatiskt utför. [↑](#footnote-ref-12)
13. Tester som utförs ur användarens perspektiv. [↑](#footnote-ref-13)
14. Ett webbbaserat versionshanteringssystem som bygger på git. [↑](#footnote-ref-14)
15. Systematiskt sätt att kontrollera strukturen i en databas. [↑](#footnote-ref-15)
16. De olika stegen som kontrollerar strukturen i en databas. [↑](#footnote-ref-16)
17. Ett växande fält i en databas är ett fält som kan tänkas öka i volym. [↑](#footnote-ref-17)
18. Stored procedures, används för att automatisera och samla logik i databas anrop. [↑](#footnote-ref-18)
19. Ett inbyggt system i .NET för att hantera databasanvändning. [↑](#footnote-ref-19)
20. <http://www.zamzar.com/uploadComplete.php?convertFile=xls&to=xml> [↑](#footnote-ref-20)
21. En påbyggnad av ramverket ASP.NET MVC 4, för Api utveckling. [↑](#footnote-ref-21)
22. Ett API som inte kräver säkerhetsnyckel [↑](#footnote-ref-22)
23. Mappning för urler(webbadresser) [↑](#footnote-ref-23)
24. Ett designmönster. [↑](#footnote-ref-24)
25. En flexibel dataåtkomst klass . [↑](#footnote-ref-25)
26. En del av MVC (model, view, controller) strukturen. [↑](#footnote-ref-26)
27. Create, Read, Update, Delete. [↑](#footnote-ref-27)
28. [https://developers.google.com/maps/](%20https:/developers.google.com/maps/) [↑](#footnote-ref-28)
29. Frontend (språk som arbetar på klienten) språk som beskriver en webbsidas utseende. [↑](#footnote-ref-29)