CHALMERS

Ärendehanteringssystem på web

Examensarbete inom Högskoleingenjörsprogrammet i datateknik

JOHAN NILSSON HANSEN

Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2014

Ärendehanteringssystem på web

Johan Nilsson Hansen

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

Institutionen för data- och informationsteknik
Göteborg, 2014

Sammanfattning

Datakraft i Småland AB är ett företag i Gnosjöregionen som säljer och utvecklar affärssystem till företag. Om en kund får problem och behöver hjälp finns det idag inget system för att hantera detta. För att göra det enklare för både företag och kund vill de nu utveckla ett onlinebaserat ärendehanteringssystem. I ärendehanteringssystemet ska kunden kunna lägga in nya och överskåda befintliga ärenden hos företaget. I denna rapport presenteras planering och utveckling utav detta projekt. Även metoder för säkerhet och optimering utav systemet utvärderas. Resultatet av projektet är ett system bestående av en webbapplikation, en webbtjänst och en databas som tillsammans erbjuder funktionalitet för ärendehantering på ett snabbt och säkert sätt.

Abstract

Datakraft i Småland AB is a company located in the Gnosjö region who sells and develop business system for companies. If a customer is in need of support there currently is no system available for managing this. So to make it easier for both the customer and the company, they now want to develop an online based system for support management where the customer can create new and read existing errands. In this thesis the planning and development process of this project is described in great detail. The thesis also presents evaluations of methods for securing and optimizing the system. The result of the project is a secure and rapid system consisting of a web application, a web service and a database who together offers all the functionality needed for a support management system.

The report is written in Swedish.

Keywords: ASP.NET, SOAP, Web, Security, Optimization

Förord

Detta arbete har utförts som ett examensarbete i data- och informationsvetenskap på Chalmers tekniska högskola och arbetet har ägt rum på företaget Datakraft i Småland AB. Först vill jag tacka Datakraft i Småland AB för att jag fått göra mitt examensarbete där. Sedan vill jag tacka min handledare på företaget, Kristoffer Skogbert, för support och handledning under arbetets gång. Slutligen vill jag även tacka min handledare och examinator på Chalmers, Christer Carlsson, för all hjälp med rapportskrivandet.

Innehållsförteckning

1 **Inledning** ... 1
- 1.1 Bakgrund ... 1
- 1.2 Problem ... 1
- 1.3 Syfte ... 1
- 1.4 Mål ... 1
- 1.5 Avgränsningar ... 1
- 1.6 Kravspecifikation ... 2
- 1.7 Frågeställningar ... 2

2 **Metod** ... 3
- 2.1 Implementation ... 3
- 2.2 Scrum ... 3

3 **Teknisk bakgrund** ... 4
- 3.1 Pyramid Business Studio ... 4
- 3.2 Pyramid Konsult Studio ... 4
- 3.3 Webbtjänst ... 4
- 3.4 SOAP ... 4
- 3.5 WSDL ... 5
- 3.6 Pyramid Webbtjänst ... 6
- 3.7 .NET Plattformen ... 7
- 3.8 ASP.NET ... 7
- 3.9 JQuery ... 8
- 3.10 SSL ... 8
- 3.11 URL-routing ... 9
- 3.12 Cookies ... 9

4 **Implementation** ... 10
- 4.1 Systemstruktur ... 10
- 4.2 Databas ... 11
- 4.3 Webbtjänst ... 12
  - 4.3.1 Protokoll ... 12
  - 4.3.2 Struktur ... 12
  - 4.3.3 Funktionalitet ... 13
  - 4.3.4 Spridning ... 14
  - 4.3.5 Testning ... 14
- 4.4 Webbapplikation ... 15
  - 4.4.1 Valet av arkitektur ... 15
  - 4.4.2 Vyer ... 16
  - 4.4.3 Mail ... 19
  - 4.4.4 URL-routing ... 19
  - 4.4.5 Kommunikation webbtjänst ... 20
  - 4.4.6 Säkerhet ... 21
  - 4.4.7 Optimering ... 21

5 **Resultat** 24
5.1 Databas 24
5.2 Webbtjänst 24
5.3 Webbapplikation 25

6 **Diskussion** 28
6.1 Scrum 28
6.2 Tidsplanering 28
6.3 Val av arkitektur 28
6.4 Mailserver 29
6.5 Miljö och samhällsaspekter 29

7 **Framtida arbete** 30
7.1 Sammanslagning hemsidor 30
7.2 Kommunikation webbtjänst 30
7.3 Mobilanpassa 30

Litteraturförteckning 31

Appendix A. Mätdata 34

Appendix B. Tidsplan 35

Beteckningar

CIL	Common Intermediate Language. Bytekod för exekvering i en virtuell maskin. Alla .NET-språk kompileras till CIL-kod.
CLR	Common Language Runtime. Virtuell maskin för exekvering av CIL-kod.
CRUD	Create Read Update Delete. Grundläggande operationer för databashantering.
CSS	Cascading Style Sheets. Språk för design och formatering av dokument.
HTML	HyperText Markup Language. Språk för att skapa webbsidor.
HTTP	HyperText Transfer Protocol. Protokoll för överföring av data på Internet.
HTTPS	HyperText Transfer Protocol Secure. Protokoll för säker överföring av data på Internet. Använder sig av SSL för kryptering.
Pyramid	Modulärt affärssystem utvecklat utav Unikum Datasystem AB.
SMTP	Simple Mail Transfer Protocol. Protokoll för överföring av e-mail.
SSL	Secure Socket Layer. Protokoll för initiering och kryptering av datatrafik.
SOAP	Simple Object Access Protocol. Protokoll för utbyte av strukturerad information i XML-format.
TCP	Transmission Control Protocol. Protokoll för pålitlig överföring på Internet.
URL	Uniform Resource Locator. En textsträng som används för att identifiera resurser på Internet.
Visual Kosmos	Programmeringsspråk för utveckling i systemet Pyramid.
WSDL	Web Service Description Language. Språk för att beskriva en webbtjänst som använder sig utav SOAP.
XML	eXtensive Markup Language. Taggspråk med regler för hur kodning av ett dokument ska ske.

1 Inledning

1.1 Bakgrund

Datakraft i Småland AB är ett IT-företag med fäste i Gnosjöregionen som i huvudsak säljer skräddarsydda affärssystem till företag. Om en kund får problem och behöver support av företaget finns det inget system för att hantera detta. Vid problem ringer eller mailar kunden företaget och beskriver sitt problem. Företaget sparar sedan ner en supportförfrågan i sin databas och skickar den vidare till supportavdelningen. Supportavdelningen får sedan bearbeta problemet i sin takt och när de är färdiga kontaktas kunden och ärendet anses som färdig.

1.2 Problem

Problemet med att inte ha något system för att hantera supportärenden är att kunden inte har möjlighet att följa upp på sitt ärende. Kunden står i blindo hela vägen från det att de skickat in ärendet tills dess att det är färdig. Detta problem är speciellt tydligt vid mindre kritiska ärenden, då dessa kan ligga länge innan de bearbetas. På grund av detta vill företaget nu utveckla ett webbaserat system för att sköta detta.

1.3 Syfte

Syftet med detta projekt är att ge kunder en möjlighet att följa upp sina ärenden.

1.4 Mål

Målet med detta projekt är att utveckla ett webbaserat system för ärendehantering.

1.5 Avgränsningar

Företaget har i sitt befintliga system redan en databas där de sparar alla förfrågningar ifrån kunder. Detta projekt kommer därför inte att behandla några jämförelser utav olika databaser.

Vidare är databasen med ärenden helt integrerad i Pyramid. För att utifrån kunna kommunicera med Pyramid finns det ett inbyggt stöd för att skapa en webbtjänst med protokollet SOAP. Då SOAP är det enda stödda protokollet kommer andra tekniker inte att övervägas i skapandet av webbtjänsten.

Företaget är Microsoft-certifierade och vill därför att webbapplikationen ska utvecklas i språket ASP.NET. Därför kommer inte olika språk för webbutveckling att utvärderas.

1.6 Kravspecifikation

Systemet skall integreras med företagets befintliga system Pyramid. För att lösa detta skall två olika system utvecklas; en webbapplikation och en webbtjänst. Dessa skall erbjuda användare möjlighet att logga in och stödja create-, read-, update- och delete-operationer (CRUD) på ärenden. Systemet skall innehålla två olika typer av ärenden; support och interna. Supportärenden skall alltid gå via kundtjänst. De interna ärendena är endast till för kunden och skall inte ses av supportavdelningen. De interna ärendena kan ses som en kom-ihåg lista för kunden.

Webbtjänsten skall integreras i Pyramid och fungera som mellanhand mellan webbapplikationen och databasen. De olika delar som webbtjänsten skall stödja är:

- Validera inloggning av användare.
- Returnera ett företags ärenden.
- Redigera ett befintligt ärende.
- Skapa nya ärenden.
- Ta bort ärende.

Webbapplikationen skall begära data ifrån webbtjänsten och presentera denna information till användaren. Här skall följande delar utvecklas:

- Gränssnitt för inloggning.
- Presentera ärenden i listform.
- Filtrera listan med ärenden.
- Möjlighet till redigering av ärenden som inte är påbörjade.
- Formulär för att skapa nya ärenden.
    - Nya supportärenden skall mailas till kundtjänst.
    - Nya interna ärenden skall läggas till direkt i databasen.
- Möjlighet att radera interna ärenden.

1.7 Frågeställningar

Utöver det som står i kravspecifikationen finns det även några frågeställningar som ligger till grund för delar av rapporten. Dessa är i följande ordning:

- Vilken ASP.NET-arkitektur lämpar sig för projektet?
- Hur går det att garantera säker användning utav systemet?
- Hur kan laddningstiden för en sida optimeras?

2 Metod

I denna del presenteras information kring hur utvecklingen av projektet planerats. Först beskrivs hur implementeringen lämpligen bör utföras och sedan med vilken utvecklingsmetodik detta ska se.

2.1 Implementation

Projektet är uppdelat i två delar som ska utvecklas, en webbapplikation och en webbtjänstdel. För att bestämma vilken del som är lämpligast att börja med måste man utreda hur flexibla de är att anpassa sig efter varandra. Webbtjänsten, som kommer att utvecklas i Pyramid, har ett specifikt sätt att kommunicera utåt och är därför låst och kan få problem att ändra sig efter hur webbapplikationen beter sig. Webbapplikationen däremot ger fria händer för programmeraren att anpassa programmet och har ett stort stöd för flertalet protokoll. Med detta i åtanke kommer webbtjänsten att utvecklas först för att sedan låta webbapplikationen anpassa sig efter den.

Webbtjänsten kommer att utvecklas med hjälp av verktyget Pyramid Konsult Studio, som finns inbyggt i Pyramid. Pyramid använder sig utav programmeringsspråket Visual Kosmos. För att lära sig Visual Kosmos kommer det att i början avsättas tid för generell programmering i det. Detta för att få en insikt i språkets möjligheter och begränsningar innan systemering sker. Utan detta finns det risk för att systemeringen inte går att genomföra och behöver ändras efterhand.

Efter en tids bekantskap med Pyramid ska en studie av SOAP-protokollet utföras. Det som behöver undersökas är hur SOAP fungerar i allmänhet och sedan hur det fungerar i Pyramid. Detta kommer att följas av en systemering av webbtjänsten för att slutligen implementera den.

För att säkerställa att webbtjänstens funktionalitet fungerar som förväntat måste verktyg för testning bestämmas.

När implementationen av webbtjänsten börjar bli klar kommer webbapplikationen att påbörjas. Webbapplikationen ska utvecklas med ramverket ASP.NET. Detta kommer att ske med hjälp av programmet Microsoft Visual Studio. Tid kommer att avsättas för upplärning av ASP.NET och för att ta fram allmän information om vad som bör beaktas när en webbapplikation utvecklas. Speciellt behöver säkerhetsaspekter på webben undersökas.

2.2 Scrum

För att kunna genomföra projektet på ett strukturerat sätt ska en variant av Scrum användas som utvecklingsmetodik. De väsentligaste delarna som ska användas ifrån Scrum är loggarna och det iterativa arbetssättet.

I ett tidigt stadie ska produkt-loggen skapas genom att identifiera de olika delarna som kommer att behöva implementeras. Utifrån produkt-loggen skapas det en sprint-logg med högt prioriterade ärenden. Varje sprint pågår under en vecka och när en ny sprint påbörjas ska den föregående veckan utvärderas. Med hjälp utav loggarna ska det säkerställas att projektet fortskrider som förväntat.

3 Teknisk bakgrund

3.1 Pyramid Business Studio

Pyramid Business Studio är ett modulärt affärssystem utvecklat av Unikum datasystem AB [1]. Det innehåller ett rikt standardpaket med moduler anpassade till företag. Till exempel finns det stöd för att hantera order och kunder. Alla moduler i Pyramid använder sig av så kallade dialoger för att arbeta. En dialog är en grafisk ruta som kan anpassas med både bakomkörande kod och grafiska komponenter.

3.2 Pyramid Konsult Studio

Det finns i Pyramid en inbyggd utvecklingsmiljö kallad Pyramid Konsult Studio. Här går det att skapa nya och redigera befintliga moduler och detta görs i Unikums egenutvecklade språk Visual Kosmos.

3.3 Webbtjänst

En webbtjänst är en lösning för att kommunicera över Internet. Webbtjänster har funktionalitet i den bakomliggande koden som de exponerar på nätverk för att andra applikationer ska kunna ta del av dem. Kommunikationen sker med SOAP-protokollet (se kapitel 3.4) och ett anrop till en webbtjänst kan liknas vid ett metodanrop över Internet.

3.4 SOAP

SOAP står för Simple Object Access Protocol och är ett protokoll för utbyte av information med en webbtjänst. SOAP-meddelande skrivs i XML och det finns en fördefinierad struktur för hur de ska se ut för att kunna tolkas. Hela meddelandet måste omslutas utav taggen Envelop för att indikera att det är ett SOAP-meddelande. Inuti denna tagg skrivs sedan själva meddelandet. Meddelandet kan bestå utav tre olika komponenter; Header, Body och Fault (se figur 3.1).

Header är en frivillig tagg och används för att beskriva meddelandet till tolkaren. Den innehåller taggen Transaction som innehåller all metadata och ett ID på transaktionen. Ifall tolkaren inte är den slutgiltiga mottagaren kan man i Transaction specificera om och vart meddelandet ska skickas vidare. Detta görs med elementet actor som innehåller nästa mottagare [2]. Transaction kan även säga åt tolkaren att använda en speciell teckenkodning för att kunna läsa innehållet i meddelandet. Detta sker med elementet encodingStyle. Slutligen finns det ett element kallat mustUnderstand som indikerar om tolkaren måste använda taggen Transaction för att kunna läsa resterande i meddelandet, vilket utgörs av den obligatoriska taggen Body.

I Body specificeras vilken metod det är frågan om. Beroende på om det är en begäran av data eller ett svar på en begäran kommer det att finnas information om in- och utparametrar till metodanropet. Varje parameter får en egen tagg innehållande datan.

```xml
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <h:Transaction xmlns:h="http://www.unikum.se/"
        actor="www.url.se"
        encodingStyle="http://www.w3.org/2003/05/soap-encoding"
        mustUnderstand="1">123</h:Transaction>
  </soap:Header>
  <soap:Body>
    <m:returnActivities xmlns:m="http://www.unikum.se/pws">
      <m:companyNumber>3784</m:companyNumber>
    </m:returnActivities>
    <soap:Fault>
      <faultcode>soap:Client</faultcode>
      <faultstring>Couldn't find company with specified number</faultstring>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

Figur 3.1 Ett SOAP-meddelande som anropar metoden returnActivities.

Den sista delen är Fault och är en frivillig tagg som läggs i Body. Fault används för att indikera fel som skett när meddelandet bearbetats. I Fault finns två taggar, faultcode och faultstring [3]. Taggen faultcode innehåller en fördefinierad kod som indikerar var och vems fel det är. Till exempel finns felkoderna soap:Client och soap:Server. Taggen faultstring innehåller en läsbar sträng som beskriver felet och kan väljas fritt.

För att skicka ett SOAP-meddelande finns det flera alternativ. SOAP är skapat så att meddelandena blir helt oberoende av transportprotokoll [4], vilket medför att meddelandena kan skickas med till exempel HTTP, SMTP eller TCP. Det vanligaste sättet är att skicka med hjälp av HTTP-protokollet [5]. Detta görs genom att SOAP-meddelandet omsluts av ett HTTP-meddelande som sedan skickas som vanlig Internettrafik.

I SOAP finns ingen standard som säger hur metoderna ska se ut för att kunna anropas. Detta medför att en användare inte vet vilka metoder en webbtjänst har eller hur de används [2]. För att lösa detta problem finns det ett språk kallat WSDL som används för att beskriva exakt hur tjänsten kan användas.

3.5 WSDL

WSDL står för Web Service Description Language och används för att beskriva en webbtjänst. Det är skrivet i XML och används för att ge användare av tjänsten en beskrivning på all funktionalitet som webbtjänsten erbjuder [6]. Informationen i ett WSDL-dokument kan delas upp i två delar, abstrakta och konkreta definitioner.

De abstrakta definitionerna i dokumentet är skriva på ett sådant sätt att de är oberoende av både transportprotokollet och hårdvaran som används och kan därför återanvändas. Följande abstrakta definitioner finns: [7]

- Types: Här finns information om alla datatyper tjänsten använder sig utav. Det kan vara både enkla och sammansatta typer.
- Messages: Varje metod i webbtjänsten har totalt två meddelanden, en för att begära data och en för att svara på begäran.
- Operations: Här kopplas ett metodnamn till alla dess Messages.

- Port types: En samling Operations som kopplas till en webbtjänst.

De konkreta definitionerna beskriver allt maskinspecifikt och kan inte återanvändas om till exempel tjänsten flyttas eller ändras. Informationen som finns här är följande [7]:

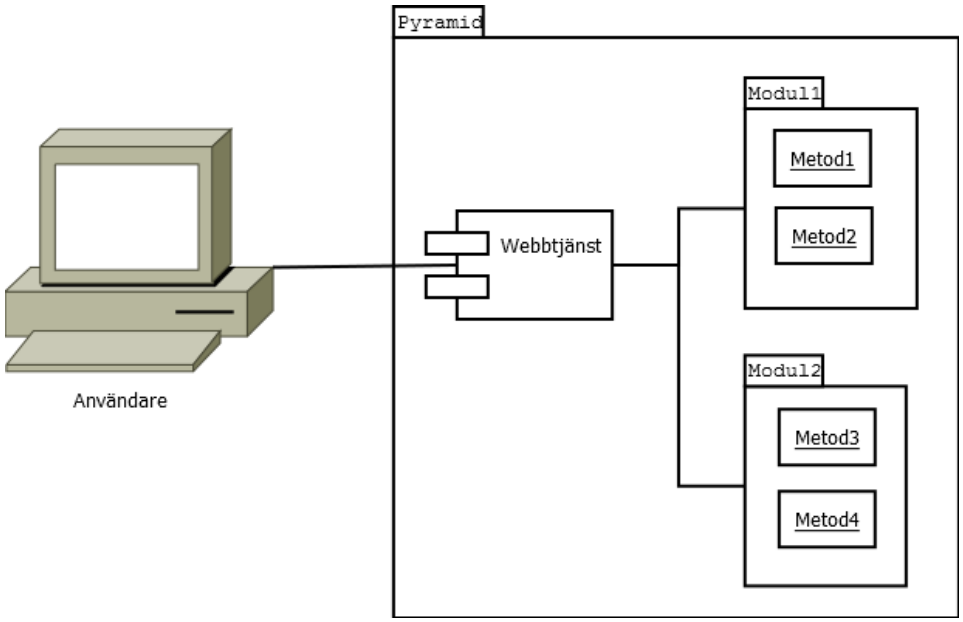
- Bindings: Kopplar Operations till ett specifikt transportprotokoll
- Ports: En Port kopplar en Binding till en nätverksadress.
- Services: Här samlas alla Ports för den specifika tjänsten.

Alla definitionerna tillsammans utgör WSDL-dokumentet och beskriver gränssnittet för att använda webbtjänsten. Dokumentet delas ut till alla klienter som ska använda tjänsten. Vissa utvecklingsmiljöer kan utifrån WSDL-dokument generera klasser och metoder till det aktuella programmeringsspråket. Dessa kan sedan användas precis som vanligt i koden, med skillnaden att metodanropen görs över Internet.

3.6 Pyramid Webbtjänst

I Pyramid finns det en integrerad webbtjänst som körs i bakgrunden. Den har till ansvar att ta hand om inkommande och utgående SOAP-meddelanden. Till webbtjänsten kopplas moduler som innehåller metoderna (se figur 3.2). När ett SOAP-anrop kommer till webbtjänsten skickar den anropet vidare till den modul som metoden finns i. Metoden får sedan exekvera och bygga upp ett SOAP-svar. När svaret är klart tar Pyramid och skickar tillbaka det till användaren som skickade anropet.

Att en modul ska kopplas till webbtjänsten indikeras via att dess namn börjar på UDWS. Vid uppstart letar Pyramid efter dessa och startar dem. Vid initiering och anrop av modulerna finns det tre olika argument som Pyramid använder för att indikera vilken typ av händelse som sker; Register, WSDL och EXEC. Under uppstart av en modul anropar Pyramid modulen med argumentet Register. Register indikerar att modulen ska registrera alla metoder tjänsten vill ha tillgängliga i webbtjänsten. Registreringen används av Pyramid för att veta vilken modul som har hand om vilket metod. Efter registreringen är klar används argumentet WSDL som ber modulen att beskriva de registrerade metoderna mer utförligt. Alla in- och utparametrar för metoderna och dess datatyper ska beskrivas här. Parametrarna och datatyperna används av Pyramid för att bygga upp ett WSDL-dokument över tjänsten. Det sista argumentet EXEC används när webbtjänsten anropats utifrån och en metod ska exekveras. Pyramid letar upp vilket modul metoden ligger i och anropar den med argumentet EXEC. Pyramid bifogar även ett extra argument som innehåller namnet på metoden som anropats och det är upp till modulen att se till att rätt metod exekveras.



Figur 3.2 Överblick Pyramid webbtjänst.

3.7 .NET Plattformen

.NET-plattformen är utvecklad utav Microsoft och stödjer både skapandet och körandet utav .NET-applikationer. All kod som utvecklas i ett .NET-språk kompileras till en bytekod kallad Common Intermediate Language (CIL). Bytekoden exekveras i den virtuella maskinen Common Language Runtime (CLR) [8]. Detta gör att .NET blir språkoberoende och det finns möjlighet att använda valfritt CIL-kompatibelt språk när ett program utvecklas till .NET-miljö, till exempel C#, C++ och Visual Basic [9] .

3.8 ASP.NET

ASP.NET är en teknologi för webbutveckling skapat utav Microsoft och är en del av .NET-plattformen. ASP.NET används för att skapa dynamiska webbapplikationer och har flera olika arkitekturer för att göra detta, bland annat Web Forms och ASP.NET MVC. Om det finns ett behov utöver vad dessa erbjuder finns det ett rikt bibliotek med påbyggnadsmoduler. Till exempel finns moduler för användning utav SOAP eller JQuery (se kapitel 3.9) i webbapplikationen.

Serverdelen programmeras i valfritt CIL-kompatibelt språk. Till skillnad från skriptspråk, där koden måste tolkas varje gång den ska användas, kompileras koden första gången en sida anropas. Detta medför att koden kan återanvändas utan vidare tolkning. Återanvändningen snabbar upp tiden det tar för applikationen att svara på ett anrop [10]. Nackdelen med ett språk som kompileras är att i första anropet blir det en extra fördröjning jämfört mot ett skriptspråk.

3.9 JQuery

JQuery är ett bibliotek skapat för att underlätta och abstrahera användningen av programmeringsspråket JavaScript. Det används för att köra kod på klientsidan i en webbapplikation. Detta för att ge en mer dynamisk webbapplikation där inte alla händelser måste gå via servern. Vanligvis används det för att manipulera koden som visas i webbklienterna eller för att hantera händelser [11].

3.10 SSL

Secure Sockets Layer (SSL) är ett protokoll som utvecklades av Netscape Corporation för att förse deras webbservrar och klienter med säker dataöverföring. SSL är idag en standard för säker överföring med HTTP [12]. SSL består av två delar; en för att initiera en säker session och bestämma krypteringsalgoritm och en för att sedan överföra data mellan två maskiner.

Initieringen sker med en serie meddelanden som skickas mellan klient och server. Både symmetriska och osymmetriska nycklar används. Handskakningen sker enligt följande [13]:

1. Klienten skickar ett ClientHello-meddelande innehållande information om klientens krypteringsalternativ och förslag på hur kommunikationen ska krypteras.
2. Servern bestämmer krypteringsmetod och meddelar klienten detta via ett ServerHello-meddelande.
3. Servern skickar sin publika nyckel (osymmetrisk) till klienten i ett ServerKeyExchange-meddelande.
4. Servern skickar ett ServerHelloDone-meddelande till klienten och väntar på svar.
5. Klienten genererar en nyckel och krypterar denna med serverns publika nyckel och skickar sedan tillbaka den med ClientKeyExchange-meddelande. Denna nyckel blir den gemensamma nyckeln (symmetrisk).
6. Klienten skickar ett ChangeCipherSpec-meddelande för att indikera att all data till klienten härefter ska krypteras.
7. Klienten skickar slutligen ett Finish-meddelande som ber servern verifiera att krypteringen initierats på ett säkert sätt genom att dekryptera ett hashvärde korrekt.
8. Servern verifierar krypteringen och svarar med ett ChangeCipherSpec-meddelande för att be klienten att kryptera all nästkommande trafik till servern.
9. Servern skickar slutligen ett Finish-meddelande som ber klienten verifiera att krypteringen initierats säkert.
10. Handskakningen är nu klar.

Efter att handskakningen godkänts av båda parterna är initieringsfasen klar. All data som skickas därefter krypteras och dekrypteras med den gemensamma nyckeln och de båda parterna kan nu kommunicera säkert.

3.11 URL-routing

URL-routing är ett sätt för en webbapplikation att acceptera länkar som inte direkt hänvisar till någon fysisk fil. Istället för att länkarna pekar på filer som ska visas, finns det i webbapplikationen en översättningstabell med routes. En route är en mappning mellan en egendefinierad länk och en fil. Detta tillåter webbapplikationen att, istället för att ha långa och krångliga URLer, använda mer beskrivande länkar [14]. En vanlig länk kan se ut som följande:

http://localhost:80/Pages/productinfo.aspx?name=BookName

En sådan länk är både lång och svår att komma ihåg för användaren. Med URL-routing går det ändra länkstrukturen till valfritt format och samma sida kan till exempel se ut såhär för användaren:

http://localhost:80/Products/BookName

Denna URL är mer lättläslig än den förra och har en mer logiskt uppbyggnad. Om en användare till exempel vill gå in på sidan och kolla på en produkt är det enkelt att förstå att länken för detta är http://localhost:80/Products/OtherBookName.

3.12 Cookies

Cookies utvecklades utav Netscape Communications för att ge Internet en form av minne. En Cookie är en liten textfil som en webbserver ber om att få spara i en användares dator. Filen kan till exempel innehålla ett unikt id genererat till användaren. Cookien skickas med i alla nästkommande anrop som görs från klienten. Med hjälp av textfilen kan webbservern identifiera vilka anrop som hör till vilken användare och på så sätt visa olika information för olika användare. Detta används till exempel för att visa en användares kundvagn på en shoppingsida. Cookies används också vanligen för att identifiera en användare som inloggad och kallas då för autentiseringcookie.

Det finns ett direktiv som säger hur hantering av Cookie får ske på en hemsida. För att få använda Cookies måste två saker vara uppfyllda [15]:

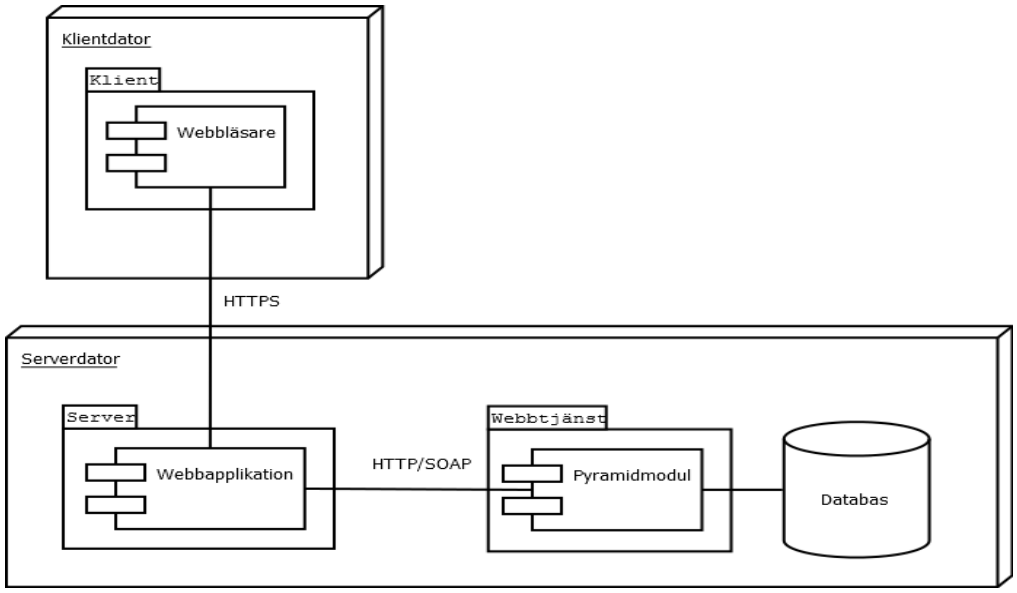
- Användaren ska ha tillgång till klar och förståelig information angående Cookiens användning.
- Användaren måste ge sitt medgivande för användning utav cookies.

4 Implementation

I detta avsnitt beskrivs alla de innefattande delarna i slutprodukten. Först ges en grundläggande överblick av systemet och därefter presenteras de olika delarna med ett bottom-up-perspektiv med start i databasen och slut i webbapplikationen.

4.1 Systemstruktur

I systemet finns det totalt fyra olika aktörer; klient, server, webbtjänst och databas (se figur 4.1). På klienten körs en webbläsare som visar information till användaren. Klienten hämtar data ifrån en webbapplikation som den kommunicerar med via HTTPS. Webbapplikationen ansvarar för att hämta information och bygga upp vyer åt användaren. Webbapplikationen kommunicerar med en webbtjänst via HTTP- och SOAP-protokollet. Webbapplikationen och webbtjänsten är stationerade på samma serverdator men behöver inte vara det. Webbtjänsten är integrerad i systemet Pyramid och har direktkontakt med den lokala databasen.



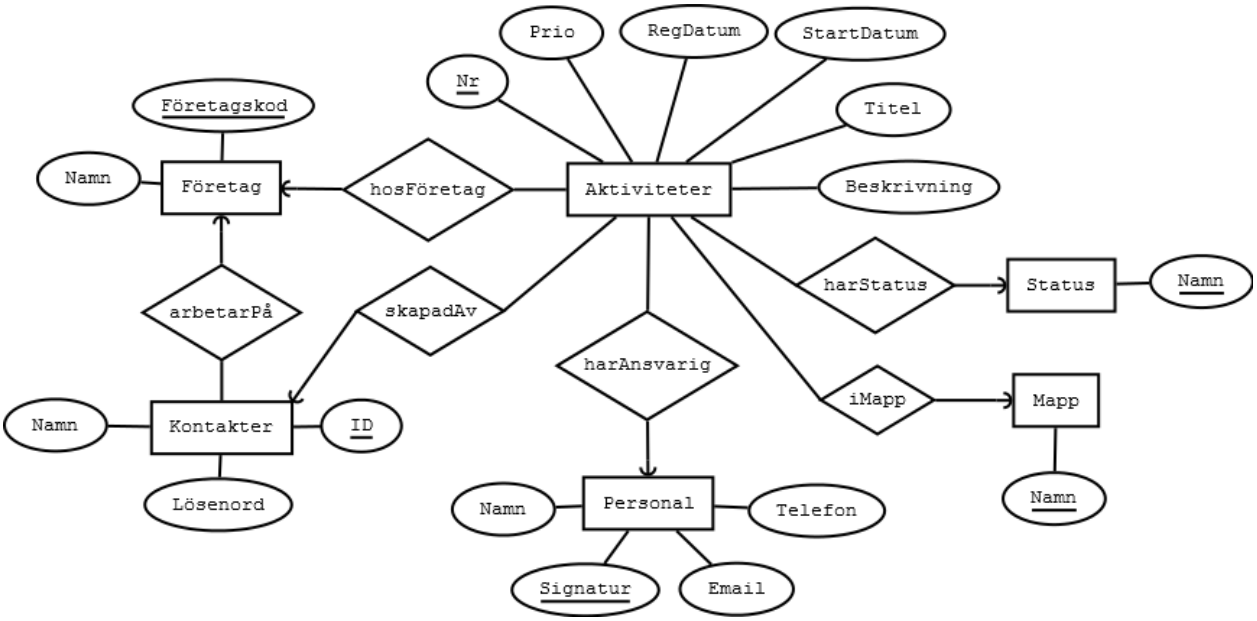
Figur 4.1 Överblick av system.

Arbetsflödet i systemet sker enligt följande:

1. En användare startar en klient. Klienten begär information ifrån servern.
2. Servern tar emot klientens begäran. Servern kontrollerar om den behöver hämta mer data för att kunna svara. Om inte går den vidare med punkt 7.
3. Servern begär ny data ifrån webbtjänsten.
4. Webbtjänsten tar emot begäran och ber i sin tur databasen att hämta information.
5. Databasen letar upp det som begärts och returnerar datan till webbtjänst.
6. Webbtjänsten lägger till den nya informationen i ett svarspaket och svarar på servern begäran.
7. Servern bygger upp sidan som ska visas och skickar tillbaka den till klienten.
8. Klienten ritar upp sidan för användaren.

4.2 Databas

Pyramid har en integrerad databas och det finns i standard Pyramid flertalet fördefinierade tabeller som kan användas för ärendehanteringen. De som valts att användas har ritats upp i ett ER-diagram (se Figur 4.2).



Figur 4.2 ER-diagram över databas.

I tabellen Företag finns all information om ett företag samlad. Tabellen Företag används för att koppla kunder och ärenden till ett företag. I Personal finns all kontaktinformation om anställda på Datakraft. Denna tabell används för att ge ärenden en ansvarig på Datakraft. Tabellen Kontakter inloggningsuppgifter för en användare. Alla användare är kopplade till ett specifikt företag. Tabellen Aktiviteter används för att spara information om ett ärende. Tabellen Status används för att hålla information om vilken status ett ärende har. Den sista tabellen heter Mapp och den innehåller information om vilket mapp ett ärende tillhör. Mapp används för att kategorisera ett ärende.

4.3 Webbtjänst

All information som ska presenteras på hemsidan måste vara tillgänglig utanför databasen. Detta medför ett problem då databasen endast kan anropas inifrån systemet Pyramid. På grund av detta behöver det utvecklas en serverdel inuti Pyramid som agerar som mellanhand mellan databas och användare. För att lösa detta utvecklades en webbtjänst i Pyramid.

4.3.1 Protokoll

I Pyramid finns det endast stöd för att skapa en webbtjänst som använder sig utav SOAP-protokollet. SOAP är helt transportprotokollsoberoende och kan därför användas utav protokollen HTTP, SMTP, TCP med flera. SOAP har endast implementerat stöd för användning med HTTP. Detta har gjort HTTP till standard för överföring av SOAP-meddelanden [16]. Då HTTP är ett standardprotokoll för webbkommunikation finns fördelen att de flesta brandväggar är enkla att konfigurera för att släppa igenom trafik. Detta gör det smidigt att installera en SOAP-tjänst på en server.

Att det endast finns implementerat stöd för HTTP medför att det med andra transportprotokoll måste utvecklas ett eget stöd för dessa. Detta är en tidskrävande process som, så vida det inte finns skäl att inte använda HTTP, är onödig.

I detta projekt kommer det inte vara något problem med att använda HTTP. Webbapplikationen (se kapitel 4.4) använder sig delvis av HTTP-protokollet för kommunikation och därför kommer standardporten för HTTP att vara öppen oberoende av valet av protokoll i webbtjänsten. Så för att inte behöva utveckla ett eget stöd för kommunikation och för att minimera antalet öppna portar används protokollet HTTP.

4.3.2 Struktur

För att ta emot de tre argument som Pyramid kan anropa webbtjänstmoduler med; Register, WSDL och EXEC, har koden utformats med en enkel struktur (Se figur 4.3).

Vid uppstart av webbtjänsten används argumentet Register och modulen registrerar då namnet på alla metoder som ska göras tillgängliga i tjänsten. När sedan WSDL-dokumentet ska skapas kommer argumentet WSDL att användas. Modulen beskriver då alla komplexa typer som används i webbtjänsten. Detta följs av en beskrivning av metodernas in- och utparametrar. Efter detta är metoderna klara för att användas. När en metod anropas skickas metodnamnet som ett extra argument. Detta för att indikera vilken metod som ska exekveras.

```
if (arg1 == "Register") {
    registerWSDL();
} else if (arg1 == "WSDL") {
    createWSDLTypes();
    createWSDLMethods();
} else if (arg1 == "EXEC") {
    if (arg2 == "Metodnamn1") {
        Metodnamn1();
    } else if (arg2 == "Metodnamn2") {
        Metodnamn2();
    }
    ....
}
```

Figur 4.3 Struktur över webbtjänst i pseudokod.

4.3.3 Funktionalitet

Tjänsten måste erbjuda metoder för olika typer av skrivning och läsning som behövs i databasen. För att tillgodose behoven i detta projekt behövdes det utvecklas flera olika metoder.

Då en användare ska kunna logga in måste det finnas en metod för att hämta ut inloggningsuppgifter. Metoden tar emot användarnamn och lösenord från servern och jämför dessa mot de befintliga i databasen. Ifall de inte överensstämmer skickas ett felmeddelande till användaren, annars skickas ett meddelande som indikerar att inloggningen godkänts. I det sistnämnda meddelandet skickas även användarens namn och företagskod. Dessa behövs för att kunna nyttja andra funktioner i webbtjänsten.

Webbtjänsten har även funktionalitet för att kunna returnera en användares alla ärenden. Denna metod tar emot en företagskod som används för att hämta ut alla ärenden ur databasen. Ärendena läggs i en lista som skickas till användaren. Om företagskoden inte finns i databasen kommer ett SOAP-fel att skickas för att indikera ett fel i anropet. Detta för att anroparen omöjligen ska ha tilldelats en icke existerande företagskod under inloggningen och därför måste anropet antingen ha manipulerats eller innehålla fel.

För att skapa nya interna ärenden infördes funktionalitet för detta. Funktionen tar emot ett nytt ärende som sparas i databasen.

De interna ärendena ska även kunna raderas utav användare. Därför utvecklades en metod som sköter detta. Metoden tar emot ett aktivitetsnummer och användarens namn. Användarens namn jämförs mot namnet för skaparen av ärendet. Överensstämmer dessa inte skickas ett SOAP-fel till användaren. Detta för att garantera att ingen ska kunna ta bort en annan användares ärenden.

Slutligen måste även webbtjänsten erbjuda funktionalitet för att redigera ärenden. Det implementerades en metod som tar emot ett ärende och uppdaterar det befintliga i databasen. Den gamla datan kommer att finnas kvar i databasen för att kunna följa upp ett ärende. Om uppdateringen gick bra besvaras anropet med ett godkännande, annars skickas ett SOAP-fel som tyder på att ett fel skett i servern.

4.3.4 Spridning

För att låta en klient veta hur tjänsten används behövs ett WSDL-dokument skapas. Detta genereras via en tjänst i Pyramid som hämtar upp all information den tagit emot under registrering av webbtjänsten. Därefter skapar Pyramid automatiskt alla bindningar till tjänsten och genererar WSDL-dokumentet. Dokumentet delas sedan ut till webbapplikationen som på så vis får kännedom om webbtjänsten.

4.3.5 Testning

För att testa en webbtjänst manuellt krävs att programmeraren skapar SOAP-meddelanden i XML och skickar till webbtjänsten för svar. Sedan inspekteras svaret för att verifiera webbtjänstens funktionalitet. Testningen måste återupprepas varje gång en ny metod tillkommer eller någon gammal ändras. Detta kan bli ett monotont arbete och risken finns att testningen inte blir fullständigt gjord varje gång. Så för att förenkla testningen av webbtjänsten behövs det någon form av automatisk testning.

I detta projekt har programmet SoapUI [17] använts för att skapa automatiska tester utav webbtjänsten. SoapUI genererar, med hjälp av WSDL-dokumentet, en uppsättning anrop som kan användas för testning. Varje anrop får ett eller flera påståenden (assertions) som måste gälla i svaret för att det ska anses lyckat.

Alla metoder har test för lyckade och misslyckade anrop. Där det går att tillämpa testas även extrempunkter, till exempel negativa värden.

4.4 Webbapplikation

För att användare ska kunna komma åt informationen implementerades en webbapplikation. Webbapplikationens ansvar är att begära information ifrån webbtjänsten och sedan presentera den på lämpligt sätt för användaren.

4.4.1 Valet av arkitektur

ASP.NET har två olika ramverk för att utveckla webbsidor. Den första arkitekturen kallas för Web Forms och är den äldsta. Den har ett rikt bibliotek med komponenter (serverkontroller) som ersätter många av standardkomponenterna i HTML. Serverkontrollerna genererar HTML-kod, vilket snabbar upp utvecklingsprocessen [10]. Den genererade koden anpassas efter vilken webbläsare användaren har och därför ser användargränssnittet likadant ut i alla webbläsare [18]. Web Forms är händelseorienterat, vilket innebär att programmeraren slipper tänka på olika typer av HTTP-anrop i koden [18]. Istället kopplas händelser till en metod i den bakomliggande koden och denna anropas automatiskt. För att även kunna spara komponenternas tillstånd finns det ett integrerat ViewState där information kan lagras [19]. Den stora nackdelen med Web Forms är att användargränssnittet är starkt kopplat till logiken. Detta medför att det blir svårt att köra automatiska tester utav logiken [20]. Den starka kopplingen gör även att det är svårt att återanvända kod.

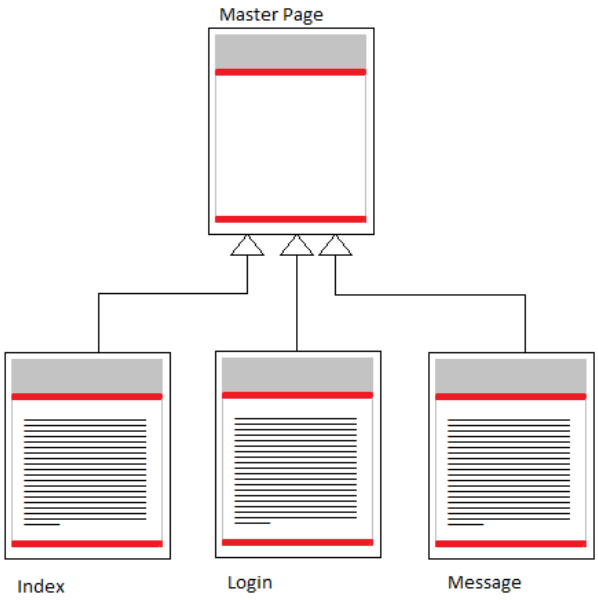
Den andra arkitekturen heter ASP.NET MVC och uppmanar användandet utav designmönstret MVC. ASP.NET MVC är ett nyare alternativ till Web Forms och är utvecklad med testning i åtanke [18]. Logiken i MVC är skild ifrån det grafiska användargränssnittet och är därför mycket enklare att testa [21]. I ASP.NET MVC har programmeraren full kontroll över all kod som skickas till klienterna [22]. Nackdelen med detta är att programmeraren själv måste sätta sig in i all kod istället för att låta webbapplikationen generera den. ASP.NET MVC har en högre inlärningskurva då arkitekturen är mer komplex [20].

Då projektet hade en mycket begränsad tid för programmering behövs en arkitektur med snabb inlärning och utveckling. Web Forms har den stora fördelen att mycket kod kan autogenereras. Med Web Forms behöver inte koden anpassas till flertalet olika webbläsare. Detta ger en snabb utveckling och tid kan istället användas till design av projektet. Att Web Forms är en äldre och mer etablerad arkitektur medför även att det finns mycket dokumentation och hjälp att få. Arkitekturens dåliga stöd för testning är ett mindre problem, då stora delar av koden kan testas med tredje parts program.

Det valdes att använda Web Forms då det ger större fokus på designen av projektet.

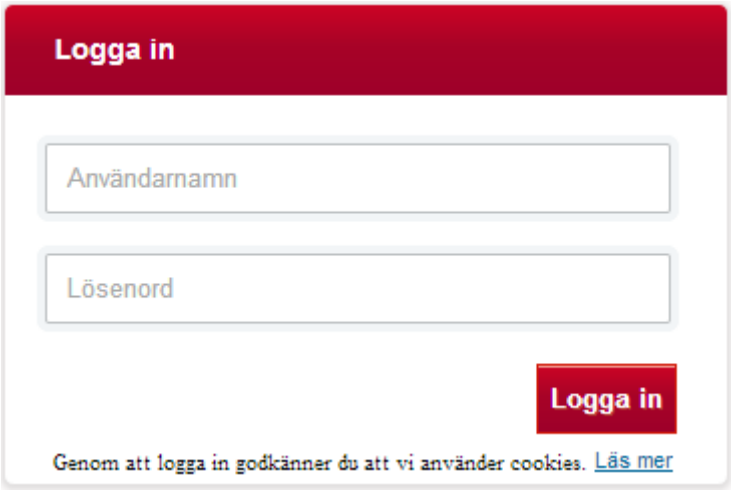
4.4.2 Vyer

Webapplikationen består utav tre olika sidor. För att få samma struktur på sidor implementerades en mastersida som samtliga sidor ärver sin grundstruktur ifrån. Mastersidan bestämmer layout för huvud, meny och fot och lämnar i kroppen utrymme för sidorna att lägga in sin layout i (se figur 4.4).



Figur 4.4 Vyerna ärver utseende från en master page.

Den första sidan som visas är login. Här ombeds användaren att skriva in inloggningsuppgifter. Uppgifterna skickas sedan till webbapplikationen som ber webbtjänsten att validera dem. Om valideringen misslyckas visas ett felmeddelande, annars omdirigeras användaren till indexsidan. Samtidigt som användaren vidarebefordras initieras en autentiseringcookie för sessionen. Denna används för att markera användaren som inloggad. Cookien går att använda i tio minuter, men tiden förnyas när användaren skickar nya anrop till hemsidan. För att få använda Cookies på hemsidan godkänner användaren detta vid inloggning (se figur 4.5).



Figur 4.5 Godkännande av Cookies vid inloggning.

När inloggningen är validerad kommer man till indexsidan. Här visas användaren en tabell med alla ärenden som användarens företag har. Ärendena hämtas via webbtjänsten. Tabellen visas med hjälp av serverkontrollen GridView som automatiskt genererar en grundläggande tabellstruktur. Första gången sidan visas fylls tabellen med ärenden ifrån den bakomliggande koden. För att få tabellen användarvänlig har flertalet funktionaliteter implementerats.

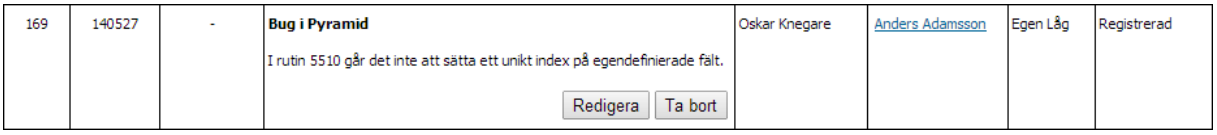
I huvudet på tabellen finns det funktionalitet för att sortera raderna. Vid skapande av tabellen genererar serverkontrollern JavaScripts-kod för att hantera klickhändelser på tabellhuvudet. Varje kolumnhuvud har ett sorteringsuttryck. Vid klick på ett kolumnhuvud skickas sorteringsuttrycket till webbapplikationen. Webbapplikationen använder sorteringsuttrycket för att initiera den nya sorteringen. Därefter byggs tabellen om på nytt och visas för användaren.

Det finns möjlighet till att filtrera och söka i tabellen. Alternativen för detta är placerade ovanför tabellhuvudet (se figur 4.6). Filtreringsalternativen uppdateras dynamiskt när ny data tillkommer i tabellen. Val av filtrering och nya sökord skickas till webbapplikationen som filtrerar tabellen enligt dessa.

Figur 4.6 Filtrering och sökning i tabell.

Användaren har möjlighet att justera kolumnernas bredd. Detta sker med hjälp utav ett tredjepartsplugin kallat ColResizable [23]. ColResizable använder sig av JQuery för att manipulera tabellen. Pluginet sparar tabellens aktuella tillstånd i webbläsarens session och kan därför komma ihåg användarens val av kolumnbredd.

Alla rader i tabellen har en standard höjd för att ge tabellen ett stilrent utseende. Alla ärenden har dock inte samma mängd information, vilket gör att information måste skalas bort på vissa ärenden. Detta har lösts genom att först ha en förhandsvisnings av ärendena. När en användare sedan klickar på ett ärende utvidgas raden och resterande information görs synlig. Detta har implementerats på klientsidan via JQuery som initierar en klicklyssnare på alla rader för att öppna och stänga respektive rad. När en rad är öppen visas alternativ för ärendet (se figur 4.7). Om ärendet är ett internt ärende visas en knappt för radering. Vid radering ombeds webbtjänsten att ta bort ärendet genom att bifoga ärendets nummer. Om ett ärende ännu inte är påbörjat finns en knapp för redigering. Vid tryck på denna blir texten för beskrivning redigerbar.



Figur 4.7 Alternativ för ändring av ärende.

Alla ärenden har en ansvarig på företaget som står listad i tabellen. Ifall en användare vill komma i kontakt med den ansvariga finns möjlighet att se den ansvarigas visitkort. Genom att klicka på den ansvarigas namn visas en ruta med information om mail och telefonnummer (se figur 4.8). Denna information hämtas i samband med hämtning av ärenden. Visning och hantering av informationen kan då ske i klienten, vilket sparar prestanda på servern. Informationen sparas i dolda fält i HTML-koden och visas med hjälp utav JQuery.

Figur 4.8 Visitkort för ansvarig på företaget.

Vid skapandet av fot i tabellen visade det sig att GridView har ett begränsat stöd för sidonavigering och tillbyggnad. På grund av detta fick foten skapas manuellt för att ge en mer funktionell design. Det implementerades stöd för att ge användare möjlighet att ange hur många ärenden som ska visas per sida, en sidonavigerare och en text som visar vilka ärenden som visas för nuvarande (se figur 4.9). GridView-kontrollern genererar JavaScripts-kod till tabellfoten för att skicka nya val till webbapplikationen.

Figur 4.9 Struktur för tabellfot.

Den sista sidan är message och är till för att användaren ska ha möjlighet att skapa nya ärenden. På message-sidan fyller användaren i ett formulär som beskriver ärendet. Validering utav datan sker i klienten och webbapplikationen. I klienten valideras datan innan den skickas till webbapplikationen och detta sker med JQuery. Om inte valideringen godkänns skickas inte något anrop till servern. Detta för att spara kraft i webbapplikationen. Att bara validera data på klientsidan medför en säkerhetsbrist. Detta då JavaScript kan stängas av eller manipuleras. Därför måste även validering ske i webbapplikationen. Webbapplikationen validerar med hjälp av ASP.NET's integrerade stöd för detta. I valideringarna kontrolleras att alla fält är ifyllda och att mailfältet är rätt formaterat. När validering är godkänd skickas ärendet olika beroende på om ärendet valts som internt eller inte. Om ett ärende är internt sparas det direkt i databasen. Är det istället ett supportärende mailas formulärdatan till kundtjänst.

4.4.3 Mail

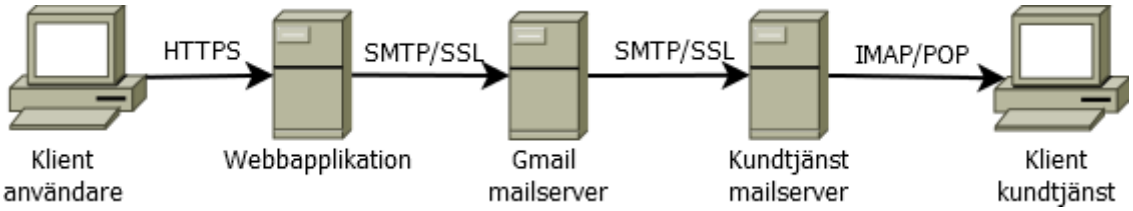
För att skicka ut mail från webbapplikationen behövs en SMTP-server. Detta kan fås genom att antingen installera en egen på en serverdator eller genom att använda en befintlig mailtjänst.

Fördelen med att installera en egen SMTP-server är att all mail går via egna servrar. Detta gör att ingen tredje part kan ta del av informationen. Nackdelen med en egen SMTP-server är att det är ett tidskrävande arbete att installera och underhålla den. Det finns även ett problem med att en del Internetleverantörer blockerar all utgående mailtrafik som inte går via deras mailservrar [24]. Detta för att förhindra att spam skickas över deras nät [25].

Genom att använda en befintlig mailtjänst behövs inte någon egen installation av mailserver. Problemet med blockering av trafik undgås då mailen skickas ifrån en annan server där utgående mailtrafik godkänts av Internetleverantören. Nackdelen med att använda en befintlig mailtjänst är att mail skickas via en tredjepart där inte allt kan kontrolleras. Ett annat problem är att mailets avsändarfält får mailtjänstens domännamn, vilket kan ge ett oseriöst intryck.

Då mailen som skickas i webbapplikationen endast ska gå till kundtjänst kommer kunder aldrig att se dessa. På grund av detta är avsändarfältets mail oväsentlig. För att inte behöva spendera tid på ett område där det redan finns flertalet existerande lösningar valdes det att använda en befintlig mailtjänst.

Det valdes att använda Googles mailtjänst Gmail. Detta görs genom att webbapplikationen skickar över meddelandet till Gmails mailserver som i sin tur skickar iväg det till kundtjänst (se figur 4.10). För att garantera säker överföring till och från deras mailserver krypteras all SMTP-trafik med SSL-protokollet.



Figur 4.10 Tillvägagångsätt för sändning av mail.

4.4.4 URL-routing

För att undvika långa och krångliga URL-länkar i adressfältet har URL-routing implementerats i webbapplikationen. Alla sidor kopplas till en routinglänk enligt ett routing schema (se figur 4.11). Istället för att använda den faktiska adressen går det att länka till sidans routinglänk. När servern tar emot ett anrop till en routinglänk omdirigeras anropet till den faktiska adressen. Till exempel används länken ”http://localhost/login” istället för ”http://localhost /Pages/login.aspx” för att gå till loginsidan.

Mappning	Fysisk adress
""	"~/Pages/index.aspx"
"message"	"~/Pages/message.aspx"
"login"	"~/Pages/login.aspx"

Figur 4.11 URL-routing schema för webbapplikation.

Detta ger ett mer stilrent utseende åt applikationen. URL-routing har även fördelen att alla sidornas länkar pekar på routinglänkar och inte de faktiska adresserna. Detta medför att ifall en sida flyttas eller byter namn räcker det med att endast ändra routinglänken till den nya adressen. Utan URL-routing måste alla länkar kontrolleras och ändras, vilket kan vara svårt då det är enkelt att missa att ändra en länk.

4.4.5 Kommunikation webbtjänst

För att kommunicera med webbtjänsten används Visual Studios Web References. En Web Reference kopplas till WSDL-dokumentet som webbtjänsten genererade. Med hjälp av dokumentet genererade Visual Studio klasser med alla sammansatta datatyper och metoder som krävs för att använda tjänsten.

För att webbtjänsten inte ska behöva hantera onödigt många anslutningar valdes det att varje klient maximalt får ha en samtida koppling till webbtjänsten. Detta ökar effektiviteten hos webb-applikationen och webbtjänsten. Nackdelen med endast en anslutning är att anrop inte kan ske parallellt. Detta problem är speciellt tydligt om det är långt avstånd mellan maskinerna. Varje anrop måste vänta en round-trip-time innan nästa anrop kan köras. I detta projekt är webbapplikationen och webbtjänsten stationerade på samma serverdator vilket gör att round-trip-time är liten och därför inget problem.

Alla metodanrop till webbtjänsten har lagts i en omslagsklass. Klassen skapar en anslutning till webbtjänsten vid första anropet och använder därefter denna i efterföljande anrop. Eftersom en användare maximalt får ha en samtida anslutning har klassen implementerats med en variant av designmönstret Singleton. Designmönstret Singleton gör att användaren aldrig kan ha mer än en instans av klassen och därmed bara en anslutning. För att se till att varje användare ska kunna ha en egen anslutning har det gjorts en liten avvikelse mot standarddesignen i Singleton. Istället för att spara instansen av klassen i en statisk variabel så sparas den i sessionen för användaren. Därmed gäller bara Singleton-instansen för en viss användare och alla användare kan ha en egen anslutning.

Varje anslutning kan maximalt vara öppen under tiden som webbapplikationen skapar en sida åt användaren. När sidan är klar och har skickats till användaren stängs anslutningen. I nästa anrop öppnas den igen vid behov. Detta görs för att öka effektiviteten. Kostnaden för att öppna en ny anslutning vid varje sidgenerering är en liten fördröjning. Dock är det en liten kostnad jämfört med att hela tiden hålla en anslutning öppen. Att hålla en anslutning öppen ger en overhead i systemet då det tar både minne och processortid [26]. Anslutningen kan även vara helt onödig, då det inte går att förutsäga om användaren kommer att behöva använda anslutningen igen. Slutligen finns även problemet att det finns en begränsning i hur många anslutningar som kan vara uppkopplade samtidigt.

4.4.6 Säkerhet

För att kunna använda hemsidan på ett säkert sätt används SSL för kryptering av datatrafik. Den mest kritiska sidan i webbapplikationen är loginsidan där en användares inloggningsinformation ska överföras till servern för validering. Utan SSL skulle denna information skickas i klartext över Internet. Detta gör det enkelt för packetsniffare att få tag i dessa uppgifter. Den initiala tanken var att endast kräva SSL användning på inloggningssidan, men detta lämnar ett säkerhetshål i webapplikationen. Efter att en användare har loggat in korrekt får användaren en autentiseringscookie för att bekräfta sin identitet. Autentiseringcookien är giltig i tio minuter och skickas med i varje dataöverföring. Om någon sniffar upp ett paket ifrån klienten som innehåller denna Cookie kan de använda den själva och identifiera sig som någon annan. För att lösa detta problem valdes det att implementera hela webapplikationen med HTTPS. Detta implementerades genom att ifall en användare försöker anropa hemsidan med HTTP-protokollet, omdirigeras anropet till motsvarande sida med HTTPS. På detta vis blir Cookien alltid krypterad och kan inte sniffas utan att knäcka krypteringsalgoritmen. Med HTTPS på hela sidan blir även all data krypterad, vilket kan vara en fördel då känslig företagsinformation kan förekomma. Den stora nackdelen med HTTPS är att det blir en overhead på all datatrafik, vilket kräver upp till två gånger mer tid av servern [12].

Då informationen som skickas i webbapplikationen kan vara känslig är säkerheten viktigare än snabbheten. Mängden datatrafik som skickas till klienten är liten, vilket gör att krypteringen går snabbt.

4.4.7 Optimering

Snabbheten i en hemsida påverkar i stor grad en användares intryck av den. En slö sida ger en dålig användarupplevelse, minskar användandet utav tjänsten och kan ge ett dåligt intryck av företaget [27]. Detta gäller även för små tidsfördröjningar. I en undersökning av Googles hemsida noterades att när laddningstiden för deras hemsida ökade med en halv sekund, sänktes deras trafik med 20 % [28]. I ett annat fall när de minskade Google Maps startsidas storlek från 100kb till 80kb ökade deras trafik med 10 % [29]. Därför är det viktigt att minimera väntetiden. Bland det viktigaste för att snabba upp en hemsida är att minimera sidans storlek och antalet HTTP-anrop som utförs [30] [31].

4.4.7.1 Komprimering

Att minimera sidans storlek görs genom att komprimera data som skickas. Först och främst är det CSS och JavaScript som kan komprimeras.

Komprimering av CSS minskar CSS-filens storlek och snabbar upp tiden för tolkning och körning av den [32]. Komprimeringen sker genom att all onödig kod tas bort eller effektiviseras. Ofta finns det i koden extra blank-tecken, radbrytning eller annan intendering av kod som lagts dit av programmeraren för att göra koden mer lättläslig. Dessa är onödiga vid tolkning i webbläsaren och kan tas bort. Ifall flera CSS-selektorer har liknande innehåll kan dessa slås ihop till en. Slutligen kan även vissa färgkoder förkortas till enklare hexadecimala värden. Alla dessa komprimeringar sker lämpligast med hjälp av ett automatiserat verktyg. I detta projekt valdes det att använda det onlinebaserade verktyget CSS Compressor & Minifier.

Komprimeringen innebar att CSS-koden minskade i storlek från 17,3kb till 11,1kb, vilket är en minskning på 36 %. För att undersöka hur mycket minskningen påverkar tjänsten testades indexsidan med och utan komprimering. Detta skedde genom att uppmäta data i tio efterföljande anrop. I testen

uppmättes sidans storlek och den genomsnittliga laddningstiden (se figur 4.12). Sidans totala storlek minskade med 0,8 %. På den genomsnittliga laddningstiden syns en förbättring av sidans laddningstid med 10 %.

Kompression	CSS	JavaScript	Övrigt	Total storlek	Antal HTTP-anrop	Genomsnittlig laddningstid
Utan komprimering	17,3kb	678,7kb	84,9kb	780,9kb	12st	467,8ms
CSS-komprimering	11,1kb	678,7kb	84,9kb	774,7kb	12st	421,2ms

Figur 4.12 Jämförelse prestanda med och utan CSS-komprimering.

All JavaScripts-kod i projektet kan också komprimeras. Detta minskar filstorleken, vilket medför en minskning av överföringstid. Komprimeringen tar bort alla extra tecken och kommentarer. Även alla variabel- och parameternamn ändras till namn med färre bokstäver. För att komprimera koden har verktyget Google Closure Compiler använts, då det är ett av de verktyg som har bäst komprimering [33].

Precis som med CSS-komprimeringen har JavaScripts-koden testats på indexsidan (se figur 4.13). Storleken på JavaScripts-koden minskade med 54 %. Detta påverkade i stor grad den totala storleken på sidan som minskade med 47 %. Den minskade mängden överförd data förbättrade den genomsnittliga laddningstiden med 12 %.

Kompression	CSS	JavaScript	Övrigt	Total storlek	Antal HTTP-anrop	Genomsnittlig laddningstid
Utan komprimering	17,3kb	678,7kb	84,9kb	780,9kb	12st	467,8ms
JS-komprimering	17,3kb	312,4kb	84,9kb	414,6kb	12st	415,1ms

Figur 4.13 Jämförelse prestanda med och utan JS-komprimering.

4.4.7.2 Buntning

Att minimera antalet filer som överförs kan minska den genomsnittliga laddningstiden Speciellt märkbar är minskningen om det är fler än sex filer som ska överföras. Detta då de flesta stora webbläsare begränsar antalet parallella HTTP-anslutningar till sex [34]. Ska fler filer överföras får de vänta tills en anslutning blir ledig. Att minska antalet filer kan ske genom att flera filer av samma typ buntas ihop till en stor fil [35].

För att bunta ihop filer finns det i ASP.NET ett ramverk kallat Microsoft Web Optimization Framework. Med detta ramverk buntas JavaScripts- och CSS-filer ihop vid uppstart av webbapplikationen. För att testa hur effektiv buntningen är testades indexsidan i webbapplikationen. På indexsidan finns det två olika CSS-filer och fyra olika JavaScripts-filer som hämtas med varje sidhämtning. I testet uppmättes laddningstiden med och utan buntning (se figur 4.14). Med buntning minskade antalet HTTP-anrop från tolv till åtta. Detta påverkade den genomsnittliga laddningstiden med 46 ms, vilket motsvarar en förbättring med 10 %.

Kompression	CSS	JavaScript	Övrigt	Total storlek	Antal HTTP-anrop	Genomsnittlig laddningstid
Utan komprimering	17,3kb	678,7kb	84,9kb	780,9kb	12st	467,8ms
Buntning	17,3kb	678,7kb	84,9kb	780,9kb	8st	421,8ms

Figur 4.14 Jämförelse prestanda med och utan buntning av filer.

4.4.7.3 Implementation

Då både komprimeringen och buntningen gav stora förbättringar till webbapplikationen valdes det att använda båda. Detta för att försäkra sig om att användare inte ska få ett slött intryck av tjänsten. Tack vare den minskade filstorleken sparas även bandbredd i systemet. För att se den totala förbättringen dessa optimeringar gav upphov till testades indexsidan (se figur 4.15). Den totala storleken på sidan minskade med 48 %, antalet HTTP-anrop som behövs med 33 % och laddningstiden med 22 %.

Kompression	CSS	JavaScript	Övrigt	Total storlek	Antal HTTP-anrop	Genomsnittlig laddningstid
Utan komprimering	17,3kb	678,7kb	84,9kb	780,9kb	12st	467,8ms
CSS + JS + Buntning	11,1kb	312,4kb	84,9kb	408,4kb	8st	365,6ms

Figur 4.15 Jämförelse prestanda med och utan optimering.

5 Resultat

Resultatet av detta projekt är att ett webbaserat system för ärendehantering har utvecklats. Systemet innehåller fyra olika aktörer; databas, webbtjänst, webbapplikation och klient. Alla aktörer har specifika uppgifter och samspelar med varandra.

5.1 Databas

Databasen är integrerad i Pyramid och används för att lagra all data i systemet. Informationen som finns är följande:

- Information kring kunders ärenden.
- Inloggningsuppgifter för kunder.
- All supportpersonal hos uppdragsgivaren.
- Alla kunder hos uppdragsgivaren.

5.2 Webbtjänst

Då databasen är integrerad i Pyramid behövdes det i Pyramid utvecklas en webbtjänst. Detta för att göra informationen tillgänglig utanför Pyramid. Webbtjänsten använder sig utav HTTP och SOAP-protokollet för kommunikation. För att webbapplikationen ska veta hur webbtjänsten används ett WSDL-dokument. Webbtjänsten har stöd för att validera inloggning utav användare och för att hantera ärenden. För att systemet ska klara CRUD-operationer på ärendena erbjuder tjänsten följande funktionalitet:

- Returnera alla ärenden för ett specifikt företag
- Redigera befintliga ärenden
- Skapa ett nytt ärende åt kund
- Ta bort ärenden

Webbtjänsten har testats med verktyget SoapUI. Varje metod i webbtjänsten har test för lyckade och misslyckade anrop. I tillämpliga fall finns det även test med extrempunkter. Själva testningen sker automatiskt och eventuella fel rapporteras. Tack vare dessa automatiska tester garanteras det att hela systemet testas efter varje ändring och funktionaliteten säkerställs.

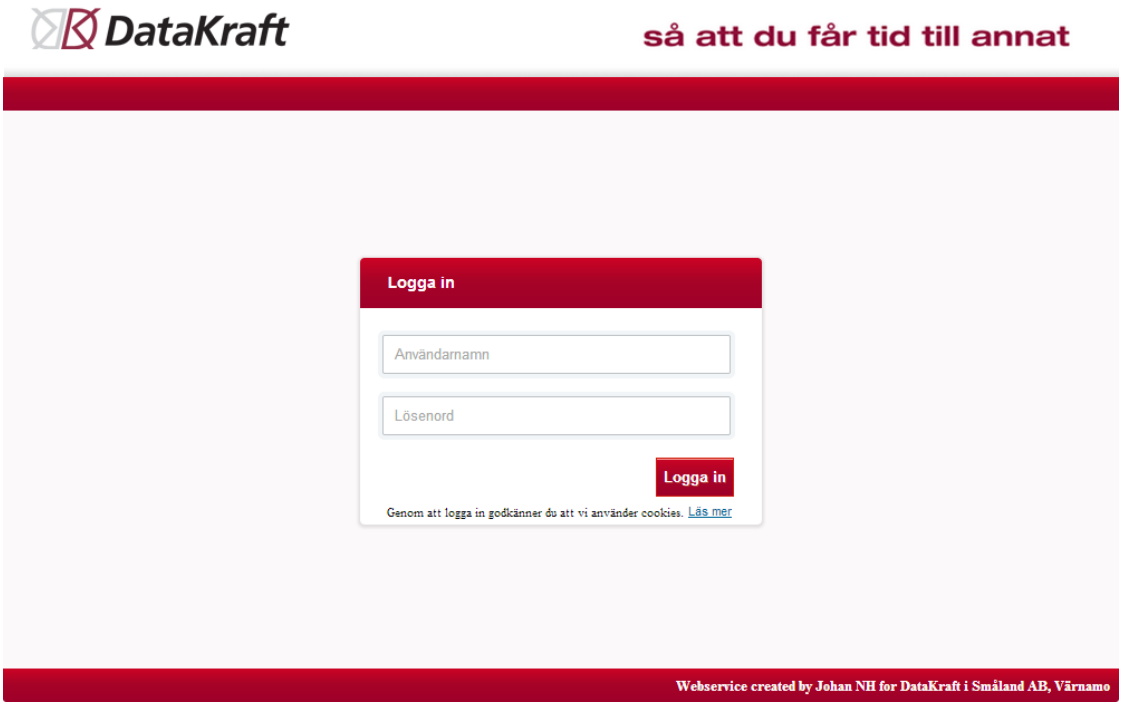
5.3 Webbapplikation

Webbapplikationen ansvarar för att hämta information ifrån webbtjänsten och presentera den för användaren. För att snabbt komma igång med ASP.NET och kunna lägga fokus på design valdes det att använda arkitekturen Web Forms. Detta då Web Forms innehåller ett rikt bibliotek med serverkontrollers och är händelseorienterat, vilket gör att mycket HTML-kod genereras.

Kommunikation mellan klient och webbapplikationen sker med HTTPS-protokollet. Detta för att garantera användaren säker användning utav systemet. Detta åstadkoms genom att om användaren försöker gå in på en sida med HTTP-protokollet omdirigeras anropet till motsvarande sida med HTTPS.

I webbapplikationen finns det tre olika sidor; login, index och message.

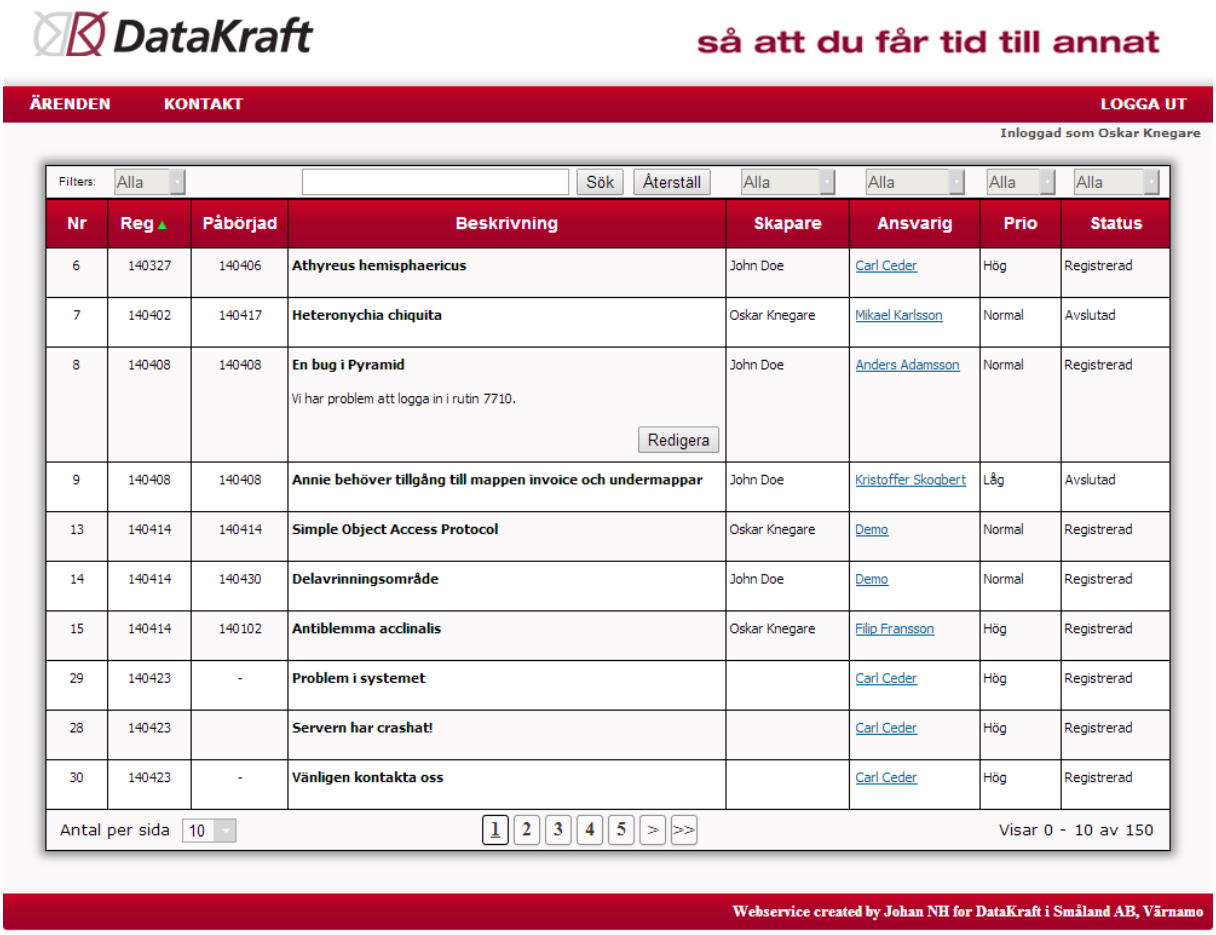
Om en användare går in på hemsidan utan att vara inloggad visas loginsidan. På loginsidan får användaren skriva in sina uppgifter (se figur 5.1). För att validera uppgifterna ombeds webbtjänsten jämföra inloggningsuppgifterna mot de i databasen. Webbapplikationen kontrollerar svaret ifrån webbtjänsten och om valideringen godkänts omdirigeras användaren vidare till indexsidan. Har valideringen misslyckats visas ett felmeddelande för användaren som ombeds försöka igen.



Figur 5.1 Utseende inloggningssida.

Indexsidan visas användaren en tabell innehållande alla ärenden skapade utav företaget. Till en början visar tabellen en förhandsvisning av ärendena. Detta för att alla rader i tabellen ska få samma initiala storlek, vilket ger ett strukturerat utseende. Ett ärende kan inspekteras mer ingående genom att trycka på en rad (se figur 5.2).

När en rad är öppen finns det knappar som visas beroende på ärendets status och typ. Om ett ärende inte är påbörjat finns det en knapp för att redigera dess beskrivning. När ett ärende redigeras skickas all ny data till webbtjänsten som uppdaterar det gamla ärendet. Ärendets gamla beskrivning sparas för att ge möjlighet att se om ett ärende ändrats. Ifall ett ärende är skapat som ett internt ärende finns möjlighet att ta bort det.

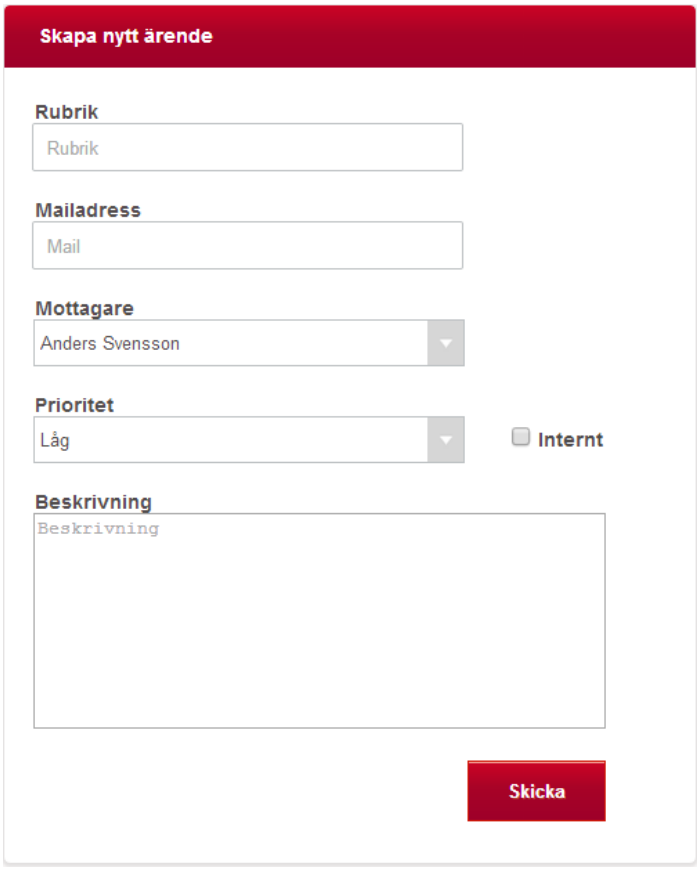


Figur 5.2 Utseende tabell på indexsidan med ärende nummer åtta öppet.

Tabellen är interaktiv och har stöd för att sortera, filtrera och justera kolumnerna. Sortering sker genom klick på en kolumnheader och kan sorteras stigande eller fallande ordning. Filtreringensalternativen uppdateras dynamiskt så att varje gång ny data tillkommer läggs de nya värdena in. Filtreringen sker i webbapplikationen som returnerar alla ärenden som matchar. Justering av kolumnbredd implementerades med JQuery.

Att skapa ett nytt ärende sker på sidan message. På message-sidan får användaren fylla i ett formulär som beskriver problemet (se figur 5.3). För att tillgodose kravet att det ska gå att skapa både interna ärenden och supportärenden finns checkboxen internt. Om denna markeras går ärendet som internt och sparas direkt ner i databasen. Om ärendet är ett supportärende mailas formuläret till uppdragsgivarens kundtjänst. Mailet skickas med Googles mailtjänst Gmail.

Webbapplikationens sidor har optimerad laddningstid. Detta tack vare komprimering och buntning av datan som överförs. All CSS- och JavaScripts-kod har komprimerats och alla filer har buntats ihop. De olika optimeringarna testades på indexsidan genom att uppmäta den genomsnittliga laddningstiden (se Appendix A). Totalt förminskades den totala mängden data som överfördes med 48 % och antalet HTTP-anrop med 33 %. Den genomsnittliga laddningstiden blev 22 % snabbare.



Figur 5.3 Formulär för skapande av nytt ärende.

6 Diskussion

6.1 Scrum

Användandet utav Scrum har bidragit positivt till projektet. Tack vare en produkt-logg var det enkelt att få en överblick av arbetsuppgifterna som ska genomföras. Detta gjorde det enklare att planera i förväg. Det blir möjligt att se hur projektet utvecklas och om milstolpar uppnås.

6.2 Tidsplanering

I början av projektet skapades en tidsplan med riktlinjer för när delar ska vara implementerade (se Appendix B). Det som fungerat bäst enligt tidsplanen var att hålla rapportskrivningen kontinuerlig under arbetets gång. Att skriva om saker när de implementeras ökar förståelsen för dem. Att tidigt börja med rapporten gör det även möjligt att få tidig feedback på den.

Det gjordes en överskattning på hur lång tid det skulle ta att lära sig systemet Pyramid. Bara hälften av den utsatta tiden för detta behövdes. Detta tack vare Pyramids rika bibliotek med standardmoduler som gjorde att det gick väldigt fort att komma igång med systemet.

Den snabba inlärningen av Pyramid gav mer tid åt design och konstruktion av webbapplikationen. Tiden som avsattes för design av webbapplikationen räckte till att få igång en grundläggande funktionalitet, men det visade sig att det fanns mer att göra än som först uppskattats. Det som tagit mest tid var finjustering och anpassning av komponenter. Detta då det finns mycket finjustering att göra. Till exempel är det enkelt att skapa en tabell, men då behövs även en sidonavigerare i tabellen vilket är mer avancerat att implementera.

6.3 Val av arkitektur

Valet av arkitektur har haft stor betydelse för hur webbapplikationen utformats. Web Forms valdes på grund av möjligheten att autogenerera komponenter istället för att skriva all HTML-kod för hand. De autogenererade komponenterna visade sig dock i många fall snarare stjälpa än hjälpa. Detta för att inte allt kan autogenereras. Allt utöver standardvalen i en komponent måste skapas för hand, vilket är ett tidskrävande arbete. Detta medförde att den valda arkitekturen påverkade tiden att implementera webbapplikationen mycket mindre än som först uppskattats. Troligen hade det inte tagit mycket längre tid med ASP.NET MVC. Med ASP.NET MVC hade det blivit en låg koppling mellan de grafiska och logiska delarna. Detta hade medfört en bättre projektstruktur som hade gått att testa utan tredjepartsprogram. Därför hade det i efterhand varit klokare att använda ASP.NET MVC.

6.4 Mailserver

Att skicka mail via en befintlig mailserver fungerar väldigt bra funktionsmässigt men det har en säkerhetsaspekt som måste noteras. Mailen är krypterade med SSL både till och från mailservern för att inte någon utomstående ska kunna läsa informationen som skickas. I mailservern finns det inte några garantier på att informationen hålls konfidentiell. Informationen kan granskas av en tredjepart, till exempel företaget som äger mailservern. Informationen kan användas av företaget eller säljas vidare. För att inte känslig information ska kunna läcka ut är detta därför något som borde undersökas vidare.

6.5 Miljö och samhällsaspekter

Att erbjuda en tjänst på Internet kräver i många fall att tjänsten alltid är tillgänglig. Detta för att användare ska kunna nyttja systemet när som helst under dygnet. Samtidigt som detta gör en samhällsnytta genom att tjänsten alltid är tillgänglig, finns det en negativ aspekt i det. Tjänsten är stationerad på en serverdator som alltid måste vara i körning. Den ständiga driften medför en kontinuerlig elförbrukning. Såvida inte elen är 100 % ren medför detta en negativ påverkan på hållbar utveckling. Detta främst på grund av ökade koldioxidutsläpp som följd av elförbrukningen. Även användning av en tjänst sker på en enhet som använder energi, vilket medför en ytterligare påverkan.

För att spara energi är det därför viktigt att använda befintliga tjänster. Detta ger färre maskiner i kontinuerlig körning, vilket innebär en lägre energiförbrukning och klimatpåverkan. Därför ger användandet av befintliga tjänster en positiv effekt på hållbar utveckling. I detta projekt har tjänsten Gmail använts som mailserver istället för att installera en egen.

Trots att en tjänst drar energi kan den fortfarande ha en positiv effekt på hållbar utveckling och samhälle. Produkten i detta projekt ger kunder möjligheten att få support på ett mer effektivt och strukturerat sätt. Detta kan medföra att kunder snabbare får hjälp av företaget. Företagets support gäller främst att säkerställa att kundens system och servrar upprättahålls, vilket gör det kritiskt att få hjälp i tid. Detta då ett system som ligger nere kostar pengar. Ifall en kund erbjuder en tjänst inriktad på hållbar utveckling kan ett system som ligger nere påverka hållbar utveckling negativt. Detta projekt hjälper då företaget att snabbt få support och medför därför en positiv effekt på hållbar utveckling.

7 Framtida arbete

7.1 Sammanslagning hemsidor

Webbapplikationen har utvecklats fristående från uppdragsgivarens nuvarande hemsida. För att slutföra produkten behövs därför en sammanslagning utav de båda webbapplikationerna. Detta kan ske på två olika sätt; antingen stationeras båda på samma dator och båda webbapplikationerna slås samman eller så är de stationerade på olika serverdatorer och länkar till varandra. Vilket som lämpar sig bäst behöver utredas i kontakt med serveransvarig för att sedan göra en sammanslagning.

7.2 Kommunikation webbtjänst

För nuvarande skickas all data mellan webbapplikationen och webbtjänsten okrypterat. Då de båda är stationerade på samma serverdator skickas aldrig datan utanför nätverket. Tack vare detta spelar det ingen roll att trafiken är okrypterad. Det finns dock inget som tvingar de att vara på samma maskin, vilket gör att det i framtiden kan finnas ett behov av att kryptera den trafiken. Detta kan ske genom att antingen använda HTTPS istället för HTTP-protokollet eller genom att använda SOAPs egna säkerhetsstöd Web Service Security. Det behövs en jämförelse av de båda säkerhetsmekanismerna för att se vilken som passar bäst.

7.3 Mobilanpassa

Som webbapplikationen är utvecklad nu har den ett relativt statiskt utseende och har därför dåligt stöd för olika skärmstorlekar. Då användandet av både mobil och pekdator ökar finns det ett behov att stödja olika skärmstorlekar. Att implementera stöd för flera skärmstorlekar kan vanligtvis ske på tre olika sätt [36]. Det första sättet är att konstruera en hemsida som anpassar sig efter skärmstorleken. Det andra är att skapa en sida för varje typ av enhet och sedan låta servern visa en sida beroende på vad anroparen har för enhet. Det tredje alternativet är att ha två olika hemsidor, en för skrivbordsdatorer och en för mobila enheter. Det måste undersökas vilken metod som passar detta projekt bäst. Under implementation måste testning ske med flera olika enheter.

Litteraturförteckning

[1] Unikum, ”Unikum datasystem AB,” 2014. [Online]. Available: http://www.unikum.se/. [Använd 31 Mars 2014].

[2] F. Cubera et al, ”Unraveling the Web Service Web,” *Internet Computing, IEEE,* vol. 6, nr 2, pp. 86-93, 2002.

[3] J. Snel, D. Tidwell och P. Kulchencko, Programming Web Services with SOAP, Sebastopol: O'Reilly, 2001.

[4] M. Gudgin, ”SOAP Version 1.2 Part 1: Messaging Framework,” W3, 2007. [Online]. Available: http://www.w3.org/TR/soap12-part1/. [Använd 31 Mars 2014].

[5] H. Haas och A. Brown, ”Web Services Glossary,” W3, 2004. [Online]. Available: http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/. [Använd 31 Mars 2014].

[6] R. Chinnici et al, ”Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts,” W3, 2007. [Online]. Available: http://www.w3.org/TR/wsdl20-adjuncts/. [Använd 31 Mars 2014].

[7] E. Christensen et al, ”Web Services Description Language,” W3, 2001. [Online]. Available: http://www.w3.org/TR/wsdl. [Använd 15 April 2014].

[8] T. Nash, Accelerated C# 2010, New York: Apress, 2010.

[9] A. Yadaw, ”.NET Framework CLR: Common Language Runtime,” Infosec institute, 2013. [Online]. Available: http://resources.infosecinstitute.com/net-framework-clr-common-language-runtime/. [Använd 15 April 2014].

[10] A. Freeman, M. MacDonald och M. Szpuszta, Pro ASP.NET 4.5 in C#, New York: Apress, 2013.

[11] JQuery, ”JQuery,” The JQuery foundation, 2014. [Online]. Available: http://jquery.com. [Använd 1 April 2014].

[12] D. Apostolopoulus, V. Peris och D. Saha, ”Transport Layer Security: How much does it really cost?,” i *INFOCOM '99. Eighteen annual join conference of the IEEE computer and communications societies.* , New York, 1999.

[13] S. Thomas, SSL and TLS Essentials, Hoboken: John Wiley & sons, 2000.

[14] R. Erik, ”URL Routing,” Microsoft, 8 Januari 2014. [Online]. Available: http://www.asp.net/web-forms/tutorials/aspnet-45/getting-started-with-aspnet-45-web-forms/url-routing. [Använd 29 April 2014].

[15] ICO, ”Cookies,” Information Commisioner's Office, 1 Maj 2012. [Online]. Available: http://ico.org.uk/for_organisations/privacy_and_electronic_communications/the_guide/cookies.

[16] A. Skonnard, ”Understanding SOAP,” Microsoft, 2003. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms995800.aspx. [Använd 1 April 2014].

[17] ”SoapUI,” Smartbear, 2014. [Online]. Available: http://www.soapui.org/. [Använd 29 April 2014].

[18] M. Sukesh, ”WebForms vs MVC,” CodeProjects, 7 Februari 2013. [Online]. Available: http://www.codeproject.com/Articles/528117/WebForms-vs-MVC. [Använd 16 April 2014].

[19] S. Mitchell, ”Understanding ASP.NET View State,” 4GuysFromRolla, 1 Maj 2004. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms972976.aspx. [Använd 15 Maj 2014].

[20] D. Esposito, ”Comparing Web Forms and ASP.NET MVC,” MSDN, 2009. [Online]. Available: http://msdn.microsoft.com/en-us/magazine/dd942833.aspx. [Använd 1 April 2014].

[21] J. Rolando, Beginning ASP.NET MVC 4, New York: Apress, 2013.

[22] C. Tavares, ”Building Web Apps without Web Forms.,” MSDN, 2008. [Online]. Available: http://msdn.microsoft.com/en-us/magazine/cc337884.aspx. [Använd 1 April 2014].

[23] ColResizable, ”ColResizable,” ColResizable, 2014. [Online]. Available: http://quocity.com/colresizable/. [Använd 1 April 2014].

[24] J. Elberfeld, ”Take control of outgoing email with a personal SMTP server,” *Windows Professional*, vol. 6, nr 10, pp. 1-5, 2001.

[25] L. Seltzer, ”Shutting Down the Highway to Internet Hell,” eWeek, 08 April 2005. [Online]. Available: http://www.eweek.com/c/a/Security/Shutting-Down-the-Highway-to-Internet-Hell/. [Använd 05 Maj 2014].

[26] I. Grigorik, High-Performance Browser Networking, Sebastopol: O'reilly, 2013.

[27] L. Swanson, ”Web Performance Is User Experience,” O'Reilly, 14 Januari 2014. [Online]. Available: http://programming.oreilly.com/2014/01/web-performance-is-user-experience.html. [Använd 13 Maj 2014].

[28] S. Shankland, ”We're all guinea pigs in Google's search experiment,” Cnet, 29 Maj 2008. [Online]. Available: http://www.cnet.com/news/were-all-guinea-pigs-in-googles-search-experiment/. [Använd 13 Maj 2014].

[29] D. Farber, ”Google's Marissa Mayer: Speed wins,” ZDNet, 9 November 2006. [Online]. Available: http://www.zdnet.com/blog/btl/googles-marissa-mayer-speed-wins/3925. [Använd 13 Maj 2014].

[30] S. Souders, ”High Performance Web Sites: Rule 1 – Make Fewer HTTP Requests,” Yahoo, 3 April 2007. [Online]. Available: https://developer.yahoo.com/blogs/ydnoneblog/high-performance-sites-rule-1-fewer-http-requests-7163.html. [Använd 13 Maj 2014].

[31] M. West, ”Speeding Up Page Load Times,” Treehouse, 6 Februari 2013. [Online]. Available: http://blog.teamtreehouse.com/speeding-up-page-load-times. [Använd 13 Maj 2014].

[32] Google, ”Minimize payload size,” Google, 23 Augusti 2012. [Online]. Available: https://developers.google.com/speed/docs/best-practices/payload. [Använd 13 Maj 2014].

[33] J. Minatel, ”4 JavaScript Minifiers Compared,” Wrox, 3 December 2012. [Online]. Available: http://blogs.wrox.com/article/4-javascript-minifiers-compared/. [Använd 13 Maj 2014].

[34] R. Anderson, ”ASP.NET,” Microsoft, 23 Augusti 2012. [Online]. Available: http://www.asp.net/mvc/tutorials/mvc-4/bundling-and-minification. [Använd 13 Maj 2014].

[35] A. Goswami, ”ASP.NET Web Optimization Framework,” Code Project, 25 Mars 2014. [Online]. Available: http://www.codeproject.com/Articles/748849/ASP-NET-Web-Optimization-Framework. [Använd 14 Maj 2014].

[36] Google, ”Any place, any time, any device.,” 1 Oktober 2013. [Online]. Available: http://static.googleusercontent.com/media/www.google.com/sv//intl/ALL_ALL/think/multiscreen/pdf/multi-screen-consumer-whitepaper_research-studies.pdf. [Använd 07 Maj 2014].

Appendix A. Mätdata

Följande mätdata har uppmätts på indexsidan i webbapplikationen. För att inte laddningstiden ska bli felaktig har alla test utförts utan någon sparad cache i webbläsaren.

Kompression	Ingen	CSS	JavaScript	Buntning	Alla
Test 1	478ms	447ms	414ms	457ms	349ms
Test 2	565ms	432ms	417ms	387ms	373ms
Test 3	514ms	414ms	402ms	415ms	361ms
Test 4	536ms	405ms	433ms	438ms	351ms
Test 5	455ms	414ms	468ms	411ms	382ms
Test 6	472ms	428ms	440ms	409ms	359ms
Test 7	426ms	417ms	362ms	423ms	367ms
Test 8	397ms	419ms	401ms	431ms	391ms
Test 9	415ms	421ms	404ms	434ms	358ms
Test 10	420ms	415ms	410ms	413ms	365ms
Snitt	467,8ms	421,2ms	415,1ms	421,8ms	365,6ms

Kompression	Ingen	CSS	JavaScript	Buntning	Alla
CSS	17,3kb	11,1kb	17,3kb	17,3kb	11,1kb
JavaScript	678,7kb	678,7kb	312,4kb	678,7kb	312,4kb
Övrigt	84,9kb	84,9kb	84,9kb	84,9kb	84,9kb
Total	780,9kb	774,7kb	414,6kb	780,9kb	408,4kb

Kompression	Ingen	CSS	JavaScript	Buntning	Alla
HTTP-anrop	12st	12st	12st	8st	8st

Appendix B. Tidsplan