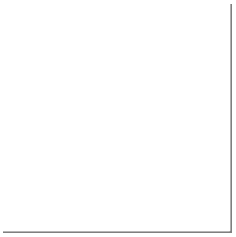


Datavetenskap

Olof Petterson

Ett webservicebaserat bokningssystem

Examensarbete, C-nivå

2002:21

Ett webservicebaserat bokningssystem

Olof Petterson

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Olof Petterson

Godkänd, 2002-06-05

Handledare: Donald F. Ross

Examinator: Donald F. Ross

Sammanfattning

Denna rapport beskriver förarbetet och utförandet vid utvecklingen av ett webservicebaserat bokningssystem. Systemet beskrivs på olika sätt från konceptuella bilder till hur koden fungerar i funktionerna. Systemet har beställts från ett företag där det är tänkt att fungera som ett internt bokningssystem för deras resurser såsom konferensrum och kringutrustning.

A Webservice Based Booking System

Abstract

This report describes the work before and during the development of a booking system based on a web service. The system is described from different angles such as conceptual design to the method functionality. This system has been ordered by a company, and will be used as an internal booking system for their resources such as conference rooms and other facilities.

Innehållsförteckning

Figurförteckning

1 Inledning

1.1 Översikt

Denna rapport fungerar som en beskrivning på vad jag har gjort under mitt examensarbete. Arbetet har bedrivits ute på ett företag där jag genom arbetet har skaffat mig nya kunskaper samt tillämpat de som jag redan hade. Arbetet har bestått av att skapa ett bokningssystem åt företaget. I detta kapitel kommer jag att beskriva hur rapporten är upplagd och vad som kommer att finnas med i den.

1.2 Rapportens upplägg

Det arbete som jag har utfört har jag delat upp i tre delar och det är därför lämpligt att jag beskriver dessa var för sig. Av denna anledning så ryms substansen av denna rapport i de tre följande kapitlen. I kapitel två beskrivs den första delen av förarbetet, nämligen den biten då jag undersökte vad jag skulle göra. Här specificeras även den uppgift som jag har arbetat med under större delen av tiden. Denna del är den som i ett projekt skulle kunna liknas vid analysfasen. Den andra delen av förarbetet beskrivs i kapitel tre. Här beskrivs designarbetet som i ett projekt skulle finnas under designfasen. Jag beskriver hur jag har tagit fram modeller för hur systemet skall byggas och hur det principiellt skall se ut. I kapitel fyra kommer en beskrivning på hur systemet har utformats. Här beskrivs de olika byggstenar som har använts samt hur de använts för att konstruera systemet.

2 Bakgrund

2.1 Företaget

Det företag där jag skall göra mitt examensarbete är ett ganska ungt IT-företag. Det har cirka 25 anställda som till största del är konsulter. Företaget arbetar mest med försäljning, support och modifiering av affärssystem Jeeves. Det är inom Jeeves som företagets anställda har sin kompetens men det förekommer även att de göra lösa applikationer för att tillgodose kundernas önskningar.

2.2 Översikt

Företaget använder sig av en rad olika lösningar, till exempel Jeeves, Lotus Notes och BackOffice, för att hantera sina administrativa behov såsom tidsrapportering och ekonomi. De har även tagit fram egna lösningar för att underlätta, till exempel har det gjorts en koppling mellan Lotus Notes och Jeeves så att tidsrapporterna automatiskt går in i Jeeves. Då det administrativa arbetat sker i flera olika program har det från de anställda framkommit önskemål om en enhetlig lösning där man kan sköta all administration. Av denna anledning så har företaget startat ett projekt för att skapa en verksamhetsportal och det är en del av denna portal som jag skall utveckla nämligen ett bokningssystem. Eftersom verksamheten länge har varit baserad på till exempel Jeeves så är det önskvärt att de gamla programmen skall ligga till grund för den önskade lösningen då det annars skulle bli väldigt kostsamt att överföra all information till det nya systemet.

2.3 Innehåll

För att reda ut vad portalen skulle innehålla gjorde jag en enkätundersökning bland företagets anställda denna återfinns i bilaga A. De fick frågor om vilken information de behöver i sitt dagliga arbete, vilken information de oftast söker efter samt vilken information om företaget de behöver. En sammanställning av resultaten finner ni som bilaga A. I samråd med min handledare på företaget diskuterade vi oss fram till vilken del som jag skulle implementera. Vi kom fram till att jag skulle konstruera ett bokningssystem eftersom ett enhetligt sätt att

boka företagets resurser länge har varit önskat. Bokningssystemet passade ganska bra med de önskemål som jag själv hade på vad jag skulle tillbringa min tid med.

2.4 Uppgift

Ett bokningssystem för företagets resurser skall utvecklas. En SQL-databas skall ligga till grund för systemet. Vidare skall systemet utvecklas med C# .net arkitektur och vara webservice baserat vad detta innebär kommer jag att beskriva i kapitel 3.3. Ett av följande två gränssnitt skall implementeras, en windowsapplikation eller ett HTML-gränssnitt. Ett HTML-gränssnitt som går att presentera med hjälp av Sharepoint Portal Server borde vara den bästa lösningen. För detta gränssnitt finns ännu ingen specifikation för detta skall utarbetas tillsammans med en referensgrupp. Det skall finnas tre typer av objekt samt tjänster som skall gå att boka.

- **Fasta objekt** placerade på en viss plats
  Exempel på ett fast objekt är konferens, grupp och laborationsrum. Dessa skall kunna bokas för en person under en bestämd tid. När man bokar ett fast objekt så skall man få förslag på vilka övriga tilläggsbokningar man kan göra.

- **Mobila objekt** som kan befinna sig på olika platser
  Projektorer, laborationsservrar eller pol-bilar. Dessa skall kunna bokas för en person under en bestämd tid. Eller som tillägg till fasta objekt t.ex. konferensrum och projektor.

- **Förbruknings objekt** som används som tillägg vid en boning
  Förbrukningsartiklar skall kunna beställas för ett antal personer t.ex. kaffe och bullar för fem personer. Om ett förbruknings objekt beställs så skall det skapas en e-post beställning till en fördefinierad person.

- **Tjänster**
  Tjänster skall gå att beställa till bestämda tidpunkter, till exempel telefonkonferenser.

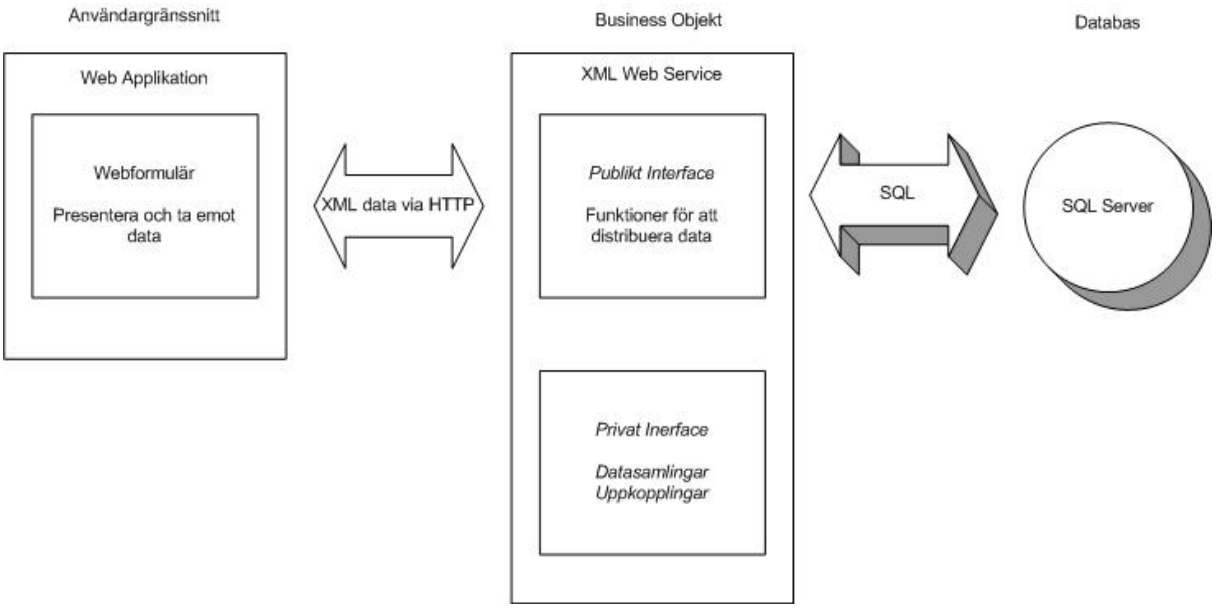
2.5 Sammanfattning

I kapitel två har jag beskrivit företaget där jag har gjort mitt arbete. Jag har även beskrivit hur jag har kommit fram till vad uppgiften skall innehålla och hur jag kom fram till dess innehåll. Uppgiften definierades i kapitel 2.4 och det är denna definition som jag har arbetat efter. Uppgiften bestod av att skapa ett bokningssystem för företagets resurser. Resurserna var uppdelade i tre olika kategorier, fasta, mobila och förbrukningsobjekt, samt tjänster. Uppgiften skulle lösas i programmeringsspråket C# och systemet skulle även fungera som en webservice.

3 Design

3.1 Översikt

För att få en stabil grund att stå på har jag valt att göra en gedigen design. Designen syftar främst till att ge mig själv en bild av systemet under utvecklingen men även för att presentera arbetets gång för handledaren. Till att börja med ritade jag en översikt över systemet så att jag lättare skulle kunna se dess olika.



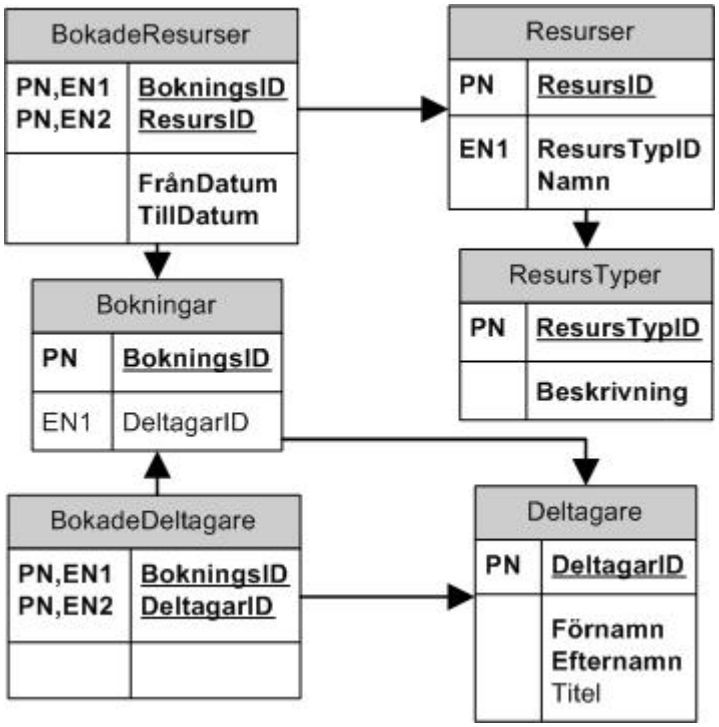
Figur 3-1: Översikt över systemet

I Figur 3-1 kan man se hur systemet skall se ut i stora drag. Jag kommer nu att beskriva de tre huvuddelarna av systemet från botten och uppåt. Jag börjar med datadelen som kommer att utgöras av en databas. Senare kommer jag även att beskriva businessobjektet och användargränssnittet som slagits samman till mjukvarudesign och beskrivs i kapitel 3.3.

3.2 Datamodell

Ur problemformuleringen, kapitel **Fel! Hittar inte referenskälla.**, har jag försökt skapa de tabeller som skall finnas i databasen. Jag har strävat efter att få databasen så dynamisk som möjligt så att den inte skall begränsa kommande delar av arbetet. Designen har skett med hjälp av ett inbyggt verktyg i SQL Server då detta visade sig vara det enklaste sättet. För ett

sedan visualisera modellen och ge en möjlighet att presentera den här så importerade jag databasen till Microsoft Visio där ett diagram konstruerades. Datamodellen ser ni i Figur 3-2 där tabellerna visas. Primärnyckeln i en tabell markeras genom att vara understruken samt ha beteckningen PN framför sig. I tabellerna finns även främmandenycklar och de markeras med beteckningen ENx.



Figur 3-2: Datamodell

Tabellen Bokningar i Figur 3-2 är den tabell som håller samman de andra. Denna tabell innehåller identifieraren för bokningen, BokningsID, samt en identifierare, DeltagarID, för den person som äger bokningen. Till denna tabell så finns en tabell BokadeDeltagare kopplad som innehåller identifierare för övriga deltagare till bokningen. Dessa DeltagarID är nycklar tagna från en tabell, Deltagare, som innehåller bland annat namn på de personer som skall gå att boka som deltagare i en bokning eller som ägare av en bokning. Tabellen BokadeResurser har en sammansatt nyckel, BokningsID-ResursID, som gör det möjligt att boka flera resurser under samma bokning. Denna tabell innehåller även tiden då den aktuella resursen har bokats. Resurserna samlas i tabellen Resurser där de har en identifierare samt ett namn. Här finns även en främmande nyckel som refererar till tabellen ResursTyper där de olika typerna av resurser samlas. Tabeller såsom ResursTyper och Deltagare har jag lagt till i syfte att göra databasen mer flexibel eftersom man i dessa tabeller kan lägga till framtida deltagare eller resurstyper så är man inte låst vid de som finns idag som man hade varit om man till exempel

hade valt att ha statiska fält med ett antal valmöjligheter. Den logik som skall finnas med i systemet har jag valt att flytta ut från databasen businessobjektet eller användargränssnittet främst för att databasen skall vara flexibel och eventuellt kunna användas till andra saker än det bokningssystem som jag skall utveckla.

3.3 Mjukvarudesign

Systemet skulle fungera som en webservice och dessutom vara baserat på Microsofts .net arkitektur [15]. För att ge en förståelse för hur en webservice fungerar så kommer jag i detta kapitel att beskriva hur businessobjektet kommer att fungera som en webservice. Jag kommer även att förklara lite om C#, som är det språk som systemet skall implementeras i, samt .net.

3.3.1 Webservice

Enligt [1] så är en webservice baserad på en standard som kallas Global XML Web Service Architecture. Denna standard bygger på SOAP, WSDL och UDDI.

- SOAP är ett lättvikts XML baserat protokoll som definierar hantering av meddelanden. SOAP definierar även kodningsregler för serialisering av data samt en metod för att göra så kallade Remote Procedure Calls, RPC.
- WSDL är ett XML format för att definiera en webservice. Det finns mer att läsa om WSDL på W3C [13].
- UDDI är ett samarbete som gör det möjligt för verksamheter att dela ut och komma åt sina egna och andras tjänster på internet. Vill man fördjupa sig i UDDI så finns det mer information på [14].

I mitt fall så är det jag vill använda mig av den funktionalitet hos en webservice som gör att man kan göra metoder tillgängliga på internet. Webservice delen, som utgörs av businessobjektet, kommer att fungera som ett gränssnitt mot internet med databasen bakom sig. Detta gör att man kan placera databaslogiken i businessobjektet och tillhanda hålla enkla funktioner för att läsa och skriva till databasen. De funktioner som tillhandahålls av businessobjektet kan sedan användas för att konstruera gränssnitten mot användarna. Det finns också att läsa på [1] att en webservice är både platformsoberoende och oberoende av vilket programmeringspråk man använder sig av.

3.3.2 C# och .net

C# har enligt [3] tagits fram av Microsoft i syfta att tillgodose det behov som har växt fram de senaste åren. Det har funnits en lucka där det har fattats ett programmeringsspråk som har

både styrkan från C/C++ och enkelheten från till exempel Visual Basic. För att fylla detta tomrum har C# utvecklats. C# är ett objektorienterat språk som bygger på både C/C++ och Java. Tillsammans med Microsofts .net plattform så får man ett integrerat stöd för att enkelt skapa effektiva internet applikationer. Det beskrivs i [2] hur man använder något som kallas IDE för att utveckla program i C#. Ett IDE eller Integrated Development Environment är en grafisk utvecklingmiljö som möjliggör snabbare och enklare utveckling av program. Microsoft har dessutom lagt in moderna finesser i C# som till exempel hjälper till att eliminera enkla programmeringsfel. För att ge exempel på dessa kan jag nämna att C# har en skräphanterare som tar hand om dynamiskt allokerad data, variabler är typsäkra samt att de initialiseras av utvecklingmiljön.

3.3.3 Användargränssnitt

Att användargränssnittet skulle vara webbaserat var den enda riktlinje som jag hade när det gällde att utforma användargränssnittet. Jag hade i designfasen inte någon större uppfattning om hur de komponenter som fanns i utvecklingmiljön fungerade eller hur de såg ut. Jag valde därför att inte göra någon design eller prototyp vad gällde användargränssnittet. Istället tänkte jag utforma det allt eftersom funktionaliteten blev klar i businessobjektet. Den funktionelitet som skulle finnas med i användargränssnittet bestod av att man skulle kunna lägga till, ta bort och ändra bokningar. Man skulle även på något sätt kunna logga in för att visa vem man var.

3.4 Sammanfattning

I kapitel tre har jag fokuserat på förarbetet. Jag har beskrivit hur jag bit för bit har lagt en grund för arbetet. Jag presenterade i kapitel 3.1 en bild, Figur 3-1, över hur systemet skulle se ut. Från denna bild delade jag sedan upp systemet i tre olika delar nämligen databas, businessobjekt samt användargränssnitt. I kapitel 3.3.1 - 3.3.3 beskrev jag hur jag har designat de olika delarna. Databasen designades i SQL Server genom att ta fram en datamodell. Businessobjeketet diskuterades och betydelsen av dess komponenter beskrevs. Till sist beskrev jag hur utformandet av användargränssnittet kommer att gå till.

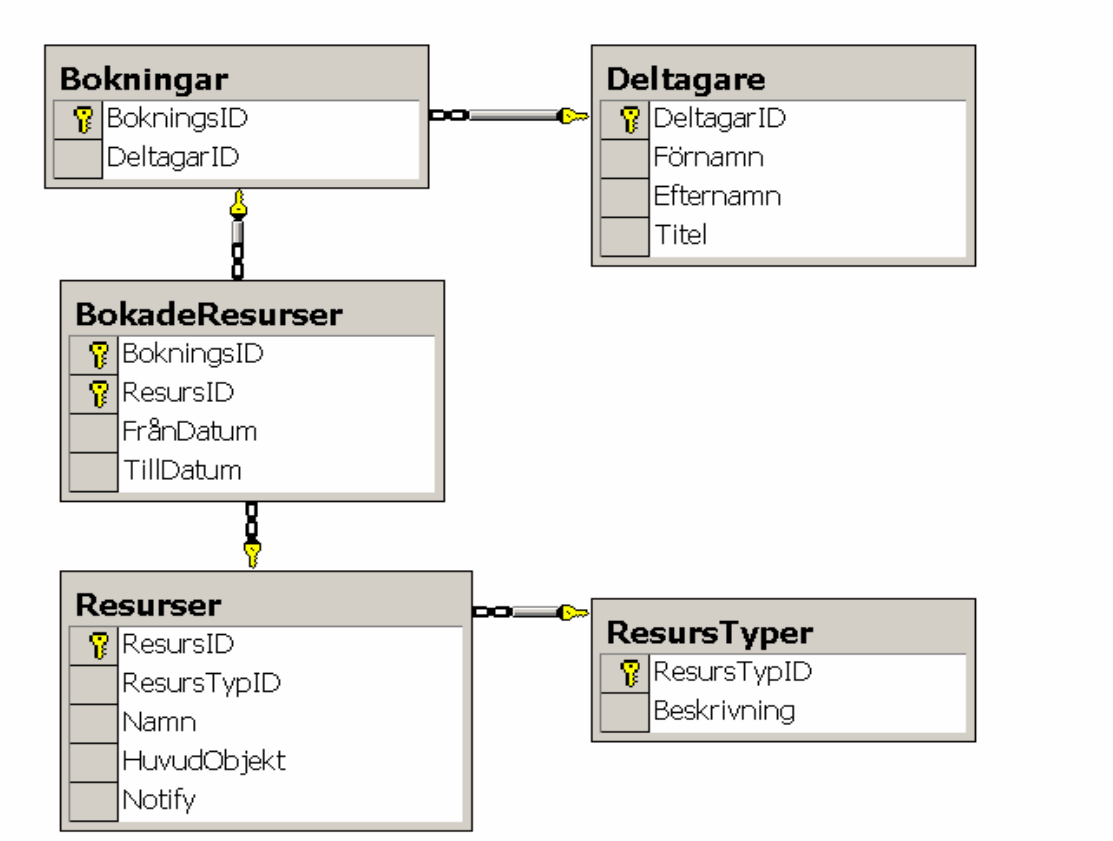
4 Implementation

4.1 Översikt

Jag kommer här att beskriva hur jag har gått tillväga när jag har utvecklat systemet. Jag kommer även att gå in lite djupare på de delar som jag anser behöver förklaras lite bättre hur de fungerar. Beskrivningen kommer att börja med databas delen och sedan förflytta sig steg för steg och till sist komma fram till användargränssnittet. På detta sett så får man gradvis de kunskaper som behövs för att kunna förstå sammanhanget när jag beskriver tillämpningarna av det jag tidigare gjort när jag skall beskriva hur interaktionen med användaren fungerar.

4.2 Databasen

För att hantera databasen har jag valt att använda SQL Server 2000, delvis för att det är det system som de använder på företaget för att hantera deras databaser men även för att det innehåller de funktioner som jag söker. I SQL Server var det enkelt att definiera sin databas eftersom SQL Server innehöll grafiska hjälpmedel för att definiera en databas. I SQL Server kunde man enkelt skapa tabeller och specificera datatyper, relationer och constraints. Den slutgiltiga databasen skildes sig åt från den som jag hade tänkt mig från början. Tabellen BokadeDeltagare togs bort på grund av att jag missuppfattat direktiven om systemet och därför antagit att man även skulle kunna boka deltagare. Jag har även lagt till två fält i tabellen resurser för att ge möjlighet till att avgöra om en resurs kunde bokas ensam eller ej samt om ett e-post meddelande skull sändas när en viss resurs hade bokats. Jag skapade även några vyer för att enklare komma åt data från flera olika tabeller. För att ge exempel på en sådan vy kan jag nämna att jag skapade en vy som innehöll informationen från tabellen resurser samt informationen från tabellen BokadeResurser. Denna vy använde jag sedan för att kunna visa både information om när resursen var bokad samt namnet på resursen. Den slutgiltiga datamodellen visas i Figur 4-1. Denna figur är till skillnad från Figur 3-2 inte ritad i Microsoft Visio utan det är ett screenshot från SQL Server.



Figur 4-1: Slutgiltig datamodel

4.2.1 Tabellbeskrivning

Jag kommer här att snabbt gå igenom tabellerna och förklara det som inte framgår i diagrammet, detta för att klargöra betydelsen av del av fälten i tabellerna.

4.2.1.1 Bokningar

Denna tabell fungerar som spindeln i nätet eftersom den identifierar de enstaka bokningarna genom primärnyckel, BokningsID. Här finns även en referens till den som äger bokningen, DeltagarID.

4.2.1.2 Deltagare

Denna tabell innehåller kort och gott de personer som kan komma att boka de olika resurserna. Alla fält i denna tabell är av typen varchar 50 förutom den primära nyckeln, DeltagarID, som är av typen int. Fälten i denna tabell är ganska självförklarande.

4.2.1.3 BokadeResurser

Denna tabell är en kopplingstabell mellan Bokningar och resurser. Detta för att ge möjlighet till att en bokning skall kunna innehålla flera resurser och att en resurs skall kunna vara bokad

i flera olika bokningar. Posterna identifieras av den sammansatta primärnyckel, (BokningsID, ResursID), och den enda nya information som finns i denna tabell är mellan vilka tider resurserna skall bokas. Dessa är av typen datetime.

4.2.1.4 Resurser

Här hittar man de olika resurser som skall ligga till grund för bokningarna. De identifieras genom den primära nyckeln, ResursID, och har sedan ett antal egenskaper. ResursTypID refererar till tabellen ResursTyper. Varje resurs har sedan ett Namn av typen varchar 50 och två flaggor HuvudObjekt och Notify. HuvudObjekt talar om huruvida resursen ensam skall kunna bokas. Huvudobjekten är främst menade att vara konferensrummen och pol-bilarna. Notifyflaggan visa om ett email skall skickas till den person som senare definieras för att tala om att en bokning av denna resurs har skett. Denna funktion har dock inte implementerats ännu eftersom företaget ej fastslagit vem som meddelandet skall sändas till.

4.2.1.5 ResursTyper

I denna tabell identifieras posterna av primärnyckeln, ResursTypID, av typen int. Här finns även ett fält, Beskrivning, av typen varchar 50. Detta fält nyttar till att beskriva resurstypen. De olika typer som fanns vid implementationstillfället var mobila, fasta och förbrukningsresurser.

4.3 Business Objektet

Det är i business objektet som den egentliga intelligensen ligger. Det är det här objektet som kommunicerar med databasen och sedan offentliggör datan. Det är business objektet som skall fungera som en webservice. Det vill säga att business objektet skall tillhandahålla de metoder som webapplikationen senare skall använda. Jag börjar med att beskriva de huvudsakliga byggstenarna som har använts för att konstruera detta objekt. Avslutningsvis kommer jag sedan att knyta ihop dessa för att beskriva hur jag har satt ihop dem för att skapa helheten.

4.3.1 DataSet

Enligt MSDN [4] så är ett **DataSet** är en samling data som ligger i minnet. Datan kommer ursprungligen från en databas. Ett **DataSet** kan innehålla en eller flera så kallade **DataTable**s. Dessa representerar tabeller från databasen och kan relateras till varandra med **DataRelation** objekt. Man kan även lägga till constraints i ett **DataSet** för att behålla integriteten i

tabellerna. Ett **DataSet** läser och skriver data och scheman som XML dokument. XML är en standard för att definiera data och ett **DataSet** objekt kan därför med fördel användas för att transportera data över internet genom HTTP. Datan kan sedan användas av vilken applikation på vilken plattform som helst bara den stödjer XML. De olika **DataTable** samlas i ett **DataTableCollection** där de sedan kan refereras. För att ändra den data som finns i de olika **DataTable** objekten så arbetar man på radnivå. Man kan med enkelhet ta bort, lägga till eller uppdatera rader. Raderna i tabellerna kommer man åt som **DataRow** objekt där man sedan kommer åt de enskilda cellerna i en post.

Jag har valt att i största möjliga grad använda mig av **DataSet** för att transportera data. Detta gäller dock inte i alla fall då det visade sig omständligt att jobba mot **DataSet** i de fall man bara skulle skicka små mängder av data och jag har i dessa fall valt att skicka primitiva datatyper. Som jag beskrev ovan så skulle det ha varit möjligt att mappa hela databasen mot ett **DataSet** objekt. Jag har dock valt att använda ett **DataSet** per tabell. Detta för att göra en enklare distinktion mellan de olika objekten och för att göra programmet mer lättförståeligt.

4.3.2 SqlConnection

För att göra kopplingen mot databasen så använder man sig av en **SqlConnection**. Enligt MSDN [5] representerar ett **SqlConnection** objekt en unik uppkoppling mot en SQL Server datakälla. Använder man sig av ett klient/server databassystem så är uppkopplingen detsamma som en nätverksuppkoppling med servern. En **SqlConnection** används tillsammans med **SqlDataAdapter** och **SqlCommand** för att höja prestandan vid användandet av Microsoft SQL Server som databassystem i sina applikationer. Den största fördelen med denna typ av koppling mot databasen är dess enkelhet. Det är mycket enkelt att skapa uppkopplingen. Det finns dock en sak man bör tänka på när man använder ett **SqlConnection** objekt. När programflödet går ur det scope där uppkopplingen skapades stängs inte ens **SqlConnection** utan man måste explicit stänga den med ett anrop till någon av dess metoder **Close** eller **Dispose**.

4.3.3 SqlDataAdapter

En **SqlDataAdapter** fungerar enligt MSDN [6] som en bro mellan ett **DataSet** och SQL Server över vilken man hämtar data. **SqlDataAdapter** skapar denna bro genom metoderna **Fill** och **Update** som genom korrekta SQL satser kan användas på följande vis. **Fill** används för att fylla ett **DataSet** objekt med all den information som finns i databasen medan **Update**

används för att uppdatera databasen så att den överensstämmer med den data som finns i ett **DataSet** objekt. En **SqlDataAdapter** innehåller också egenskaperna **SelectCommand**, **InsertCommand**, **DeleteCommand**, **UpdateCommand** och **TableMappings** för att göra hämtning och uppdatering av data enklare. En **SqlDataAdapter** innehåller alltså all den information och funktionalitet som man behöver för att hämta data från eller uppdatera data i databasen. Som jag i kapitel 4.3.5 skall beskriva så är det mycket enkelt att skräddarsy en **SqlDataAdapter** så att den passar för just det användningsområde som man har tänkt sig.

4.3.4 SqlCommand

Ett **SqlCommand** kan ses som en lättviktsadapter. För att konstruera ett objekt av denna klass behöver man ange en sträng som representerar ett SQL kommando och en referens till ett **SqlConnection** objekt. Man kan även senare specificera ett SQL kommando genom egenskapen **CommandText**. När man vill exekvera kommandot mot databasen så kan man gå tillväga på flera olika sätt. Skall man exekvera en fråga som inte ger något svar så gör man ett anrop till **ExecuteNonQuery**. Vill man däremot exekvera en fråga som ger ett svar så använder man sig av metoden **ExecuteReader** som returnerar en **SqlDataReader** som man sedan använder för att läsa svaret från. Man kan även använda **ExecuteScalar** som returnerar en skalär från databasen, denna metod är användningsbar då man söker efter till exempel ett maximumvärde i en tabell. Information om övriga sätt att exekvera ett kommando finns hos MSDN [7].

```
[WebMethod]
public int exempel()
{
        sqlConnection1.Open();
        SqlCommand cmd = new SqlCommand(. . .);
        int result = (int)cmd.ExecuteScalar();
        sqlConnection1.Close();
        return result;
}
```

I exemplet ovan visas ett fall då man till exempel vill ta fram ett max värde i en kolumn. Man deklarerar de ett **SqlCommand** och definierar en lämplig SQL sats. För att sedan få ut max värdet görs ett anrop till **ExecuteScalar** som finns i **SqlCommand**. Jag kommer i kapitel 4.3.5 vidare beskriva hur dessa komponenter fungerar.

4.3.5 Funktionsbeskrivning

Jag skall nu förklara hur man använder de byggstenar som jag just har beskrivit. För detta ändamål har jag valt att visa ett användningsfall genom att illustrera detta med en funktion.

```
[WebMethod]
public DataSet4 GetBokningarByBokningsID( int bokningsID )
{
        DataSet4 bokningar = new DataSet4();
        SqlConnection1.open();
        bokningarAdapter.SelectCommand.CommandText =
                "SELECT * FROM Bokningar WHERE BokningsID = "
                + bokningsID;
        bokningarAdapter.Fill(bokningar);
        bokadeResurserAdapter.SelectCommand.CommandText =
                "SELECT * FROM Bokningar";
        SqlConnection1.close();
        return bokningar;
}
```

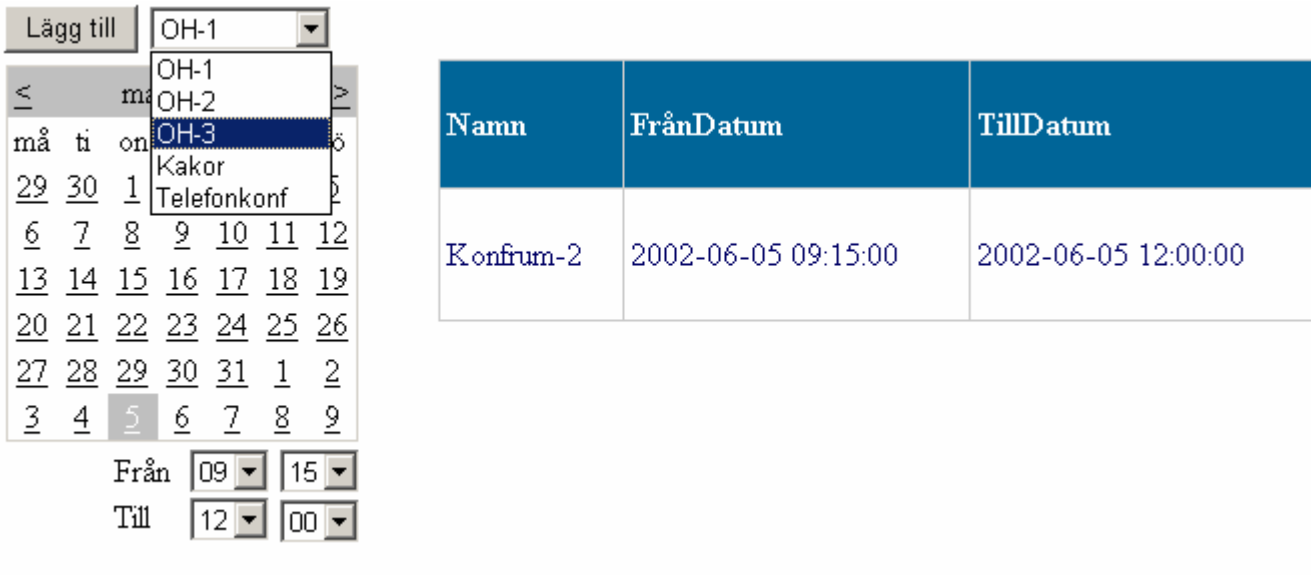
Denna funktion illustrerar en så kallad **WebMethod** som hämtar data ur tabellen **Bokningar** från databasen. Först skapas ett **DataSet** objekt som senare skall fyllas med information. Uppkopplingen mot databasen öppnas genom ett anrop till metoden **Open** i **SqlConnection1**. För att kunna hämta bara de rader som innehåller det bokningsid som skickas in till funktionen genom parametern **bokningsID** så omdefinieras **CommandText** för **SelectCommand** så att den passar ändamålet. Då adapterns **SelectCommand** innehåller ett passande kommando så görs ett anrop till metoden **Fill**. Detta göra att adapterns **SelectCommand** exekveras mot databasen och resultat läggs i det **DataSet** som skickas med som argument. Eftersom jag har definierat om adapterns **SelectCommand** så är det bra att återställa det som ändrats. Detta görs genom att adapterns **SelectCommand** återställs till sitt standardvärde, nämligen att ställa om selectsatsen så att den väljer alla rader, och man stänger uppkopplingen mot databasen genom att anropa metoden **Close** som finns i **SqlConnection**.

Detta var bara ett exempel på hur man kan använda dessa byggstenar för att komma åt databasen. Det finns naturligtvis många fler men de flesta av funktionerna i business objektet har ett liknande utseende och jag har därför valt att inte vidare gå in på dessa. Jag har byggt

denna typ av funktioner för att distribuera datan från samtliga tabeller i databasen. Det finns en getfunktion för varje tabell som tar fram all data. I de fall då det behövs ett urval så har jag gjort getfunktioner som tar argument som sedan används i selectsatsen. I de tabeller där man skall kunna lägga till data så finns det addfunktioner för detta. Till sist finns det och deletefunktioner för de tabeller där man skall kunna ta bort data.

4.4 Användargränssnittet

Här kommer jag precis som i föregående kapitel att börja med att beskriva de olika byggstenar som jag har använt för att bygga upp användargränssnittet. Efter att jag lagt fram information om byggstenarna så kommer jag att beskriva hur jag har använt dem för att bygga ihop ett gränssnitt mot användaren. För att ge en grafisk bild av de olika kontroller som jag skall beskriva presenterar jag ett exempel i Figur 4-2.



Figur 4-2: Exempel på kontroller

Längst upp till vänster i Figur 4-2 finns det en **Button** som det står lägg till på. Till vänster om knappen finns en **DropDownList** med ett antal resurser som går att välja. Delvis dold av rullgardinsmenyn finns en **Calendar** som har ett datum markerat. Under kalendern finns fyra rullgardinsmenyer för att välja tid samt två stycken etiketter av typen **Label**. Till höger i figuren visas ett **DataGrid** innehållande en bokad resurs.

4.4.1 DataGrid

För att enkelt visa en tabell på en websida så har jag valt att använda ett **DataGrid**. Denna kontroll har väldigt stora möjligheter till förändring och är lätt att anpassa så att det pass väl in i det man tänkt sig. Information om denna kontroll finns att hitta på MSDN [8]. Man kan binda ett **DataGrid** till en datakälla, till exempel ett **DataSet**. Detta gör man genom att specificera ett **DataSet** som datakälla i egenskapen **DataSource**. Den enskilda tabellen specificeras i egenskapen **DataMember**. Data från datakällan binds sedan till kontrollen genom ett anrop till metoden **DataBind** som finns i **DataGrid** Det finns flera olika kolumntyper att välja mellan när man konstruerar ett **DataGrid**, till exempel **BoundColumn**, **ButtonColumn** och **EditCommandColumn**. Per default så skapas alla kolumner automatiskt och de är då av typen **BoundColumn**. I denna typ av kolumn presenteras datan som text. Man kan även lägga till kolumnerna manuellt och då kan man även välja att lägga till andra typer av kolumner. Jag har vid ett par tillfällen valt att använda mig av en så kallad **EditCommandColumn**. Denna kolumn visar editerings knappar för varje rad. Som default visas en editknapp som om man trycker på den sätter egenskapen **EditItemIndex** till den aktuella raden. Detta gör att de fält som finns i kolumner av typen **BoundColumn** presenteras i en **TextBox** så att man kan editera dess innehåll. Där editknappen fanns visas nu istället en updateknapp och en cancelknapp. Vill man använda dessa knappar så får man specificera händelsehanterare för dem. Eftersom man själv specificerar knapparnas beteende så är det enkelt att genom att ändra texten på dem använda dem på ett sätt som passar en bättre. En annan smidig funktion hos denna kontroll är att man kan visa informationen uppdelad på olika sidor. Man kan till exempel bestämma att det bara skall visas fem rader per sida. Som default visas länkknappar i botten av kontrollen för att ge möjlighet till att byta sida. Placeringen samt utseendet av dessa länkknappar kan man ändra så att det skall passa ens ändamål. Dock fungerar inte heller dessa knappar av sig själva utan man får specificera en händelsehanterare för att ge dem en betydelse.

4.4.2 DropDownList

När man behöver lista ett antal alternativ, till exempel de olika resurserna, som användaren skall välja mellan så har denna kontroll visat sig vara mycket användningsbar. En **DropDownList** en kontroll som i en rullgardin presenterar en lista där man kan välja ett element. Enligt MSDN [9] kan man enklelt ändra denna kontrolls utseende för att det skall passa ens ändamål. En nackdel är dock att man inte kan lägga till element i listan under runtime. Man måste bestämma sig i förväg vilka element som skall finnas med i listan och

specificera dessa. Man kan dock komma runt detta genom att man kan koppla denna kontroll till en datakälla vilket ger möjlighet att dynamiskt bestämma listans innehåll. Man kan då till exempel skapa en **ArrayList** där man lägger till element. Man anger sedan denna som datakälla genom att sätta egenskapen **DataSource** referera till listan. För att binda datan i listan så anropar man **DataBind** som finns i **DropDownList**. Man kan även binda en DropDownList mot ett **DataSet** men man kan då endast visa en kolumn ur en tabell.

4.4.3 Button

En **Button** fungerar ungefär som man kan tänka sig. Man använder denna kontroll för att skapa en tryckknapp på sin websida. Man kan använda **Button** på två olika sätt, dels som en submitknapp och dels som en commandknapp. Jag har valt att endast använda submit varianten då denna visat sig tillräcklig. För en submitknapp kan man specifiera en händelsehanterare för den händelsen att knappen blir intryckt. I denna kan lägger man den kod som man vill ska exekveras när användaren trycker på knappen. Man kan enkelt genom egenskapen **Text** definiera en text som finnas på knappen. Mer information om hur en commandknapp fungerar finns på MSDN [10].

4.4.4 Calendar

För att på ett enkelt sätt kunna välja vilket datum en bokning skall gäll så har jag valt att använda mig av en **Calendar**. En **Calendar** är en webcontrol som visar en månads alla dagar på ett sätt som påminner om en almanacka. Det finns stöd för en rad olika tideräkningar för att ge möjligheten att använda denna i olika kulturer världen över. I denna kontroll kan man specificera om man skall kunna markera en dag, vecka eller en hel månad. Jag har valt att använda den med standardinställningar och går därför inte djupare in på alla dess funktioner men om du är intresserad av att läsa mer om denna kontroll så finns det mer information på MSDN [11].

4.4.5 Label

Enligt MSDN [12] skall man använda kontrollen **Label** för att visa text på en bestämd plats på sin websida. Denna kontroll är inte statiskt bunden till texten utan kan dynamiskt bindas till en ny text genom egenskapen **Text**. Detta gör att en **Label** är mycket användbar när man vill visa till exempel meddelande på ett snyggt sätt eftersom texten inte är innesluten i någon slags ruta eller liknande.

4.4.6 Funktionsbeskrivning

De olika websidor som används för att presntera och ta emot data från användaren har gjorts som webformulär. Ett webformulär, eng web form, i .Net ser ut som en vanlig html-sida. Bakom denna ligger det dock en webapplikation som ger möjlighet till betydligt större funktionalitet än hos en vanlig html-sida. När man konstruerar ett webformulär så generarar IDE't själv den asp, det finns mer om asp hos MSDN [16], kod som behövs och i de flesta fall kan man programmera ett webformulär precis som vilken klass som helst. Jag skall nu beskriva hur jag har använt byggstenarna jag beskrivit ovan för att konstruera de olika webformulär som har behövts.

4.4.6.1 Inloggning

För att lösa inloggningen har jag valt att göra detta på ett något förenklat sätt. Istället för att användaren får skriva in sitt namn och eventuellt lösenord så har jag valt att ha en fast lista med namnen på användarna. Detta främst för att öka enkelheten båda programmeringsmässigt och användarmässigt. På grund av att systemet är tänkt att användas i företagets intranät så finns det ingen anledning att tänka på säkerhetsaspekter.

Gränssnittit mot användaren är uppbyggt av en **DropDownList** samt två knappar av typen **Button**. Rullgardingslistan är bunden till en vy i databasen som jag skapade för att ge möjlighet att visa både för och efternamn. Denna lista är till för att användaren skall välja sitt namn och sedan trycka på en av de två kanpparna. Genom knapparna har användaren möjligheten att välja mellan att skapa en ny bokning eller ändra en tidigare skapad bokning. I Figur 4-3 visas ett exempel på hur inloggingsrutan kan se ut.

Figur 4-3: Inloggning

Jag har valt att lägga inloggingen på en egen sida och sedan genom knapparna länka till de andra sidorna. Av estetiska skäl var min ambition från början att ha all funktionalitet på samma sida men då det visade sig var väldigt svårt programmeringsmässigt visade det sig vara ett betydligt bättre alternativ att dela upp funktionalieteten på olika sidor. En av anledningarna till att jag ville ha med allt på en sida var att jag skulle slippa att skicka parametrar mellan sidorna. Efter en halvdags lusläsande av referensinformation visade det sig dock väldigt enkelt att lösa detta problem. Servern tillhandahåller nämligen en samling,

Session, som är definierad under hela användarens session mot servern. För att spara information i denna samling så deklarerade man en ny nyckel som man sedan band till ett värde. Från denna sida så gav detta möjlighet att spara undan användarens namn samt ett bokningsid i det fall användaren valde att skapa en ny bokning.

4.4.6.2 Skapa ny bokning

Denna sida innehåller något mer funktionalitet och dessutom ett större urval av kontroller. För att välja vilken resurs man vill boka så har jag valt att presentera dessa genom en **DropDownList**. Denna är bunden till tabellen resurser i databasen. Under bokningens första skeden visas de resurser som kan bokas ensamma, det vill säga de som har huvudobjektsbiten satt i databasen. När en resurs av denna typen har bokats så binds kontrollen på nytt till datakällan som då innehåller alla de resurser som finns som tillval, det vill säga de resurser som inte har huvudobjektsbiten satt i databasen. När anvädaren sedan har gjort sitt val så är det tänkt att han skall välja vilket datum och tid han vill boka resursen. Detta gör han genom att markera ett datum i kalendern som är av typen **Calendar** samt välja tiden i de fyra rullgardingslistor av typen **DropDownList** som finns för detta ändamål. För att sedan lägga till resursen i sin bokning trycker användaren på en knapp som gör att den information han har matat in läggs till i databasen. För att presentera de bokade resurserna har jag valt att använda ett **DataGrid** som är bundet till en vy i databsen som jag skapat för att ge möjlighet till att visa resursnamnet tillsammans med bokningstiderna.

Namn	FrånDatum	TillDatum
OH-1	2002-05-09 04:30:00	2002-05-09 06:15:00
Konfrum-1	2002-05-09 04:30:00	2002-05-09 06:15:00
Telefonkonf	2002-05-09 05:30:00	2002-05-09 06:15:00

Figur 4-4: Skapa ny bokning

I Figur 4-4 visas ett exempel på hur denna sida kan se ut efter att användaren har bokat ett konferensrum tillsammans med en OH-apparat och en telefonkonferens. Bilden visar även mer i detalj hur valet av datum och tid kan göras. Denna sida använder sig av de parametrar som sparats i **Session**-samlingen. Dessa består av användarens namn samt det sktuella bokningsnumret.

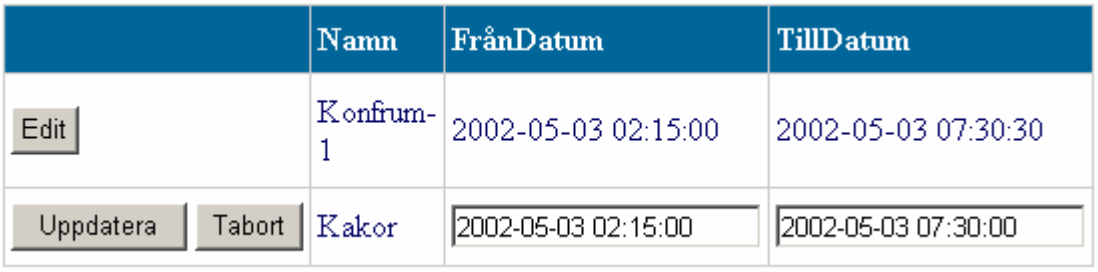
4.4.6.3 Ändra bokning

För användargränssnittet fanns det specificerat att man skulle kunna ta bort en bokning. Denna funktionalitet har fördelats på två olika sidor. Den första sidan som visas i Figur 4-5 presenterar användarens alla bokningar.

	BokningsID	FrånDatum	TillDatum
Ändra	35	2002-05-03 02:15:00	2002-05-03 07:30:30
Tabort Visa	36	2002-05-10 06:30:00	2002-05-10 09:30:00
1			

Figur 4-5: Ändra bokning ett

Informationen visas i ett **DataGrid** som skiljer sig något åt från det som fanns med på sidan för nya bokningar. Skillnaden består i att här finns det med en knappkolumn där man kan välja att ändra på en bokning. Gör användaren detta val genom att trycka på knappen märkt ändra så ges två nya valmöjligheter, dels att ta bort bokningen dels att visa densamma. De nya valmöjligheterna presenteras genom att den knapp som var märkt ändra försvinner och två nya märkta tabort och visa nu blir synliga detta illustreras i Figur 4-5 där användaren på andra raden har tryckt på knappen märkt ändra. Väljer man att tabort bokningen så tas alla poster bort från databasen som är relaterade till den aktuella bokningen. Väljer man däremot att visa bokningen så skickas man till en ny sida där den aktuella bokningen visas i sin helhet. Informationen om vilken bokning det handlar om lagras även här i **Session**-samlingen. I det **DataGrid** som bokningen presenteras finns det även här med en knappkolumn. Denna ger möjligheten till användaren att ändra tiden på en enskild bokad resurs eller helt tabort en resurs från bokningen. I Figur 4-6 visas ett exempel på hur detta kan se ut. Precis som i Figur 4-4 har användaren i Figur 4-6 tryckt på knappen märkt edit. Denna händelse har lett till att knapparna uppdatera och tabort visats och att fälten för tiderna har blivit editerbara.



Figur 4-6: Ändra bokning två

4.5 Sammanfattning

Kapitel 4 innehåller substansen i denna rapport. Här beskrivs hur utvecklingen av bokningssystemet har gått till. I kapitel 4.2 beskriver jag hur databasen fungerar. Här beskrivs tabellerna var för sig samt att det under utveklingsarbetets gång skett några förändringar vad gällde databasens utforming och vad dessa förändringar berodde på. Vidare i kapitel 4.3 beskrivs businessobjeket och de olika komponenter som används för att konstruera funktionaliteten. Här introduceras begrepp såsom **DataSet** och **DataGrid** som är komponenter i Microsofts .net ramverk. För att sedan avsluta kapitlet beskrivs användargränssnittet i kapitel 4.4. Det är när användargränssnittet och dess komponenter beskrivs som man får en uppfattning om hur bokningssystemets utseende tagit form och hur det fungerar.

5 Slutsatser

Det arbete jag har utfört har varit mycket lärorikt. Inte bara när det gäller de rent faktiska kunskaperna utan även när det gäller hur saker och ting fungerar ute i arbetslivet. Det är nämligen ganska stor skillnad på miljön på ett företag och den i skolan. Jag fick ganska snabbt lära mig att man inte kan förlita sig på att saker och är som de sägs vara och att om man vill ha något gjort så får man göra det själv. På företaget där jag gjorde mitt arbete fanns det även brist på kvalificerad hjälp att få eftersom det jag skulle arbeta med var ganska nytt. Detta kanske kan verka negativt men jag tycker att det har haft en positiv inverkan både på mig som person och på mitt arbete. Främst har det faktiskt främjat mina kunskaper eftersom jag själv har fått söka informationen då det inte funnits någon att fråga så har jag fått en djupare och bredare kunskap än om man bara blivit serverad ett svar.

När det gäller själva arbetet så är jag ganska nöjd med resultatet. Dock hade jag gärna tillbringat mer tid för att göra det bättre då det finns ett par förbättringar som jag skulle vilja införa. Ett exempel på en förbättring är att man när man försöker boka en resurs som redan är bokad vid den aktuella tiden inte bara får ett felmeddelande utan får fram en tabell där man kan se när resursen är bokad eller inte. Detta är bara en av de saker som jag skulle vilja ha gjort. Arbetet skulle innehålla tio veckors heltidsjobb och det är inte så mycket när man väl kommer i gång. Jag har för avsikt att införa dessa förändringar i systemet fastän de överstiger den tid som var utsatt för arbetet. Detta främst för att jag vill leverera en bra produkt till min uppdragsgivare.

Referenser

[1] Microsoft Corporation, Global XML Web Service Architecture White Paper, http://gotdotnet.com/team/xmlwebservices/gxa_overview.aspx

[2] Deitel et al, C# How To Program, Prentice Hall 2002

[3] Microsoft Developer Network, C# Introduction and Overview, http://msdn.microsoft.com/vstudio/techinfo/articles/upgrade/Csharpintro.asp

[4] Microsoft Developer Network, .Net Framework Class Library, DataSet Class, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemdatadatasetclasstopic.asp

[5] Microsoft Developer Network, .Net Framework Class Library, SqlConnection Class, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemdatasqlclientsqlconnectionclasstopic.asp

[6] Microsoft Developer Network, .Net Framework Class Library, SqlDataAdapter Class, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemdatasqlclientsqldataadapterclasstopic.asp

[7] Microsoft Developer Network, .Net Framework Class Library, SqlCommand Class, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemdatasqlclientsqlcommandclasstopic.asp

[8] Microsoft Developer Network, .Net Framework Class Library, DataGrid Class, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemwebuiwebcontrolsdatagridclasstopic.asp

[9] Microsoft Developer Network, .Net Framework Class Library, DropDownList Class, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemwebuiwebcontrolsdropdownlistclasstopic.asp

[10] Microsoft Developer Network, .Net Framework Class Library, Button Class, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemwebuiwebcontrolsbuttonclasstopic.asp

[11] Microsoft Developer Network, .Net Framework Class Library, Calendar Class, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemwebuiwebcontrolscalendarclasstopic.asp

[12] Microsoft Developer Network, .Net Framework Class Library, Label Class, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemwebuiwebcontrolslabelclasstopic.asp

[13] World Wide Web Consortium, Web Service Description Language 1.1, http://www.w3.org/TR/wsdl

[14] Universal Description, Discovery and Integration, http://www.uddi.org

[15] Microsoft Developer Network, .NET Framework, http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000451

[16] Microsoft Developer Network, Active Server Pages, http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000522

A Enkätsvar

Enkätsvaren har som synes delats in i tre grupper. Detta för att enklare kunna se vad de olika typerna av anställda behöver för information. Enkätens frågor framgår genom svaren.

A.1 Konsulter

Vilken information inom företaget behöver du ofta få tillgång till?

? Adressinformation
? Standardmallar.
? Information om var medarbetarna befinner sig
? Mail
? Källkod med versionshantering till olika projekt/kunder
? Projektorganisationer
? Pågående aktiviteter
? telefonitjänst, någon form av intern telefonkatalog

Vilken information utanför företaget söker du efter?

? Information om nya produkter och lösningar.
? Kunders hemsidor.
? Felrapporter från Jeeves.
? Exempel och hjälp på diverse program och programmeringstekniker.

Vilken allmän information om företaget skulle du vilja ha tillgång till?

? Debiteringsläget
? Prospekt
? Ekonomisk information.
? Kontakt information m.m

Övriga önskemål på portalens innehåll?

? Att den fungerar på pocketpc
? Portalen skall ha en gemensam plats för dokument (internt, externt), program (Windows, Notes m.m.), kunddokument

A.2 Säljare

Vilken information inom företaget behöver du ofta få tillgång till?

? Kundinformation med korrbrev och offerter etc, kundekonomi och kontaktpersoner
? Företagsinformation med namn, adresser, ekonomi etc
? Personalinfo etc
? Leverantörsinfo kring broschyrer, prislistor etc
? Presentationer om företaget och våra produkter och tjänster
? Ekonomi, order etc i Jeeves
? Vissa gemensamma filer kring marknad, försäljning

Vilken information utanför företaget söker du efter?

? Konkurrentinfo, branchinfo på webben
? Leverantörsinfo på webben
? Bokningar av konferensrum, projektorer etc

Vilken allmän information om företaget skulle du vilja ha tillgång till?

? En samlad bild av det som finns under ”gemensam” idag

Övriga önskemål på portalens innehåll?

? Den tredje hörnstenen i vårt informationssystem där Super Office och Jeeves är de två andra. Portalen måste fungera som ett bibliotek dvs där skall jag söka efter det som inte finns i de två andra systemen exempelvis weblänkar etc.

A.3 Ledningsgrupp

Vilken information inom företaget behöver du ofta få tillgång till?

? Standardmallar tex Projektplaner, projektdirektiv, standardagendor, standardprotokoll.
? Planerade kundaktiviteter.
? Procpects.
? Beläggningsstatistik (prognos)
? Rapporterad tid (jeeves)
? Information kring kunder
? Medarbetarnas kalendrar

Vilken information utanför företaget söker du efter?

? Information från Jeeves, Siebel (marknadssystem), Hogia (främst lönesystemet).
? Jag letar ibland efter externa kurser på olika institut.
? Jobbar mot lev./Distri websajter så som Scribona, Pedag, Microsoft
? Andra bolag inom koncernens aktiviteter
? Nyhetslänkar
? Börsen

Vilken allmän information om företaget skulle du vilja ha tillgång till?

? Resultatet den senaste måndaden/månaderna.
? Beläggningsgraden den senaste månaden.
? "senaste nytt", dvs någon kan rapportera nåt kul/tråkigt från tex ett projekt
? Kommande aktiviteter på företaget
? Senast gjorda affärer

Övriga önskemål på portalens innehåll?

- Enkelhet, inte för många strukturer som man måste gräva ner sig i för att finna information
- Beläggnings /resultatbarometer