INGENJÖRSHÖGSKOLAN
HÖGSKOLAN I JÖNKÖPING

Utveckling av moduler för DotNetNuke-baserade CMS/CRM-system

Robert Nygren

Martin Johansson

EXAMENSARBETE 2006
ÄMNE DATATEKNIK

INGENJÖRSHÖGSKOLAN

HÖGSKOLAN I JÖNKÖPING

Utveckling av moduler för DotNetNuke-baserade CMS/CRM-system

Developing modules for DotNetNuke-based CMS/CRM-systems

Robert Nygren

Martin Johansson

Detta examensarbete är utfört vid Ingenjörshögskolan i Jönköping inom ämnesområdet Datateknik. Arbetet är ett led i den treåriga högskoleingenjörsutbildningen. Författarna svarar själva för framförda åsikter, slutsatser och resultat.

Handledare: Anders Carstensen

Omfattning: 10 poäng (C-nivå)

Datum: 2006-06-07

Arkiveringsnummer:

Postadress:	Besöksadress:	Telefon:
Box 1026	Gjuterigatan 5	036-10 10 00 (vx)
551 11 Jönköping		

Abstract

INVID Jönköping AB is an IT-consultant company. They are a part of the INVID-group that employ about 140 people and are represented in ten different locations in Sweden. INVIDs business areas include development and customization of web applications.

The company has been developing a web publishing tool, INVID Publisher .NET. Small- and medium sized companies are offered hosting of a complete web portal/platform with Content Management System/Customer Relationship Management (CMS/CRM) functionality.

Customers can manage the content published on web pages. They can choose among a set of templates were there are modules for example text/html and news. To find the right template for the purpose can be a difficult task for the customer.

The purpose of our report is to an answer two main questions:

- How to develop modules for DotNetNuke?
- What is the foundation for the architecture and structure of DotNetNuke?

Our goal was to create modules with the programming language VB.NET and server technology ASP.NET. We would take advantage of the DotNetNuke web application framework for developing modules that offer the customer a way to easily and effectively build new web pages with INVID Publisher .NET.

The starting point for the module we have developed is based on analysis of former existing module that INVID AB was not comfortable with. We analysed and discussed the problems that the old module constituted. During the development of the module we also received feedback on our work from our tutor on INVID and other members of the development team.

Our work has lead to a template module were the customer can create new web pages based on templates and also create new templates.
INVID and we also consider our module to be graphically well designed.

In the future it's possible to further develop our module to a great extent because we have followed the principles of DotNetNuke and have written program code that is object-oriented, reusable, and extensible and is based on good programming principles and prominent design patterns.

Sammanfattning

INVID Jönköping AB är ett IT-konsult bolag. De ingår i INVID-gruppen som totalt sysselsätter drygt 140 personer och finns representerade på tio orter i Sverige. De är verksamma inom ett flertal IT-områden, däribland utveckling och anpassning av webbapplikationer.

Företaget har under en tid arbetat med att utveckla ett webbpubliceringsverktyg, INVID Publisher.NET. Små- och medelstora- företag erbjuds en komplett webbportalplattform med Content Management System/Customer Relationship Management (CMS/CRM) funktionalitet sammankopplat med webbhotelltjänster som företaget erbjuder.

Kunder har möjlighet att själva påverka det innehåll som finns publicerat på webbsidor. De kan välja bland ett antal mallar där det finns moduler för bl.a. text/bild och nyheter. Att hitta rätt mall för ändamålet kan dock vara svårt för kunden.

Vårt examensarbetes frågeställningar var följande:

- Hur utvecklar man en modul för DotNetNuke?
- Vad ligger till grund för DotNetNukes arkitektur och uppbyggnad?

Vårt mål var att skapa en eller flera moduler med programmeringsspråket VB.NET och servertekniken ASP.NET. Vi avsedde också att lära oss webbapplikationsramverket DotNetNuke för att bygga modul(er) som erbjuder kunden att enkelt och effektivt bygga nya sidor i INVID Publisher .NET.

Utgångspunkten för den modul som vi utvecklade bygger på analyser av tidigare existerande mallmodul som företaget (INVID AB) inte var nöjda med. Vi analyserade och diskuterade de problem som den gamla modulen medförde. Under utvecklingens gång fick vi feedback på vårt arbete från vår handledare på företaget och andra personer från utvecklingsavdelningen.

Examensarbetet har resulterat i en fungerande mallmodul där kunden kan med lätthet skapa nya sidor utifrån mallar och även skapa nya mallar. Uppdragsgivaren och vi anser att modulen som utvecklats är mycket grafiskt väldesignad.

I framtiden kan modulen vidareutvecklas framgångsfullt bl.a. tack vare att vi i vårt utvecklingsarbete har följt DotNetNuke-ramverket och dess principer strikt. Vi har även skrivit kod som bl.a. karakteriseras av att den är objektorienterad, återanvändbar, utbyggbar och bygger på goda programmeringsprinciper och framstående designmönster.

Nyckelord

DotNetNuke
ASP.NET
Stored Procedures
Blog
Providermodell

Innehållsförteckning

Figurförteckning

1 Inledning

1.1 Förutsättningar

1.1.1 Företagets bakgrund

INVID Jönköping AB har sjugosju anställda. De står på egna ben men ingår i INVID-gruppen som totalt sysselsätter 140 personer på tio orter i Sverige.
I Jönköping är man det andra största bolaget inom gruppen. De är verksamma inom ett flertal IT-relaterade affärsområden, däribland utveckling och anpassning av webbapplikationer.

1.1.2 Vår bakgrund

Vi är två studenter som är i slutskedet av en ingenjörsutbildning inom datateknik med inriktning på kommunikation och informationsteknik.
Under studierna har vi bl.a. läst om användbarhet, PC-teknik, datanät, webbdesign, systemutveckling, programmering i C++ och VB.NET av windowsapplikationer samt utveckling av webbapplikationer med ASP.NET.

1.2 Problemformulering

Företaget har under en tid jobbat med en lösning där små- och medelstora-företag erbjuds en komplett webbportalplattform med Content Management System/Customer Relationship Management (CMS/CRM) funktionalitet sammankopplat med webbhotelltjänster.

Kunder har möjlighet att själva påverka det innehåll som finns publicerat på sidor. De kan välja bland ett antal mallar där det finns moduler för bl.a. text/bild och nyheter. De kan även lägga till andra moduler ur en lista.

När kunden ska lägga till en sida kan det ibland vara svårt att hitta rätt mall för ändamålet. Detta komplicerar byggandet av sidor och gör att det tar längre tid än nödvändigt.

Det största problemet med detta system är när en kund ska skapa nya sidor. Idag är kunden tvungen att på ett omständligt sätt applicera modul efter modul på sidan som han skapar eller alternativt spendera onödigt mycket tid för att hitta rätt mall för ändamålet.
Därför önskar man från företagets sida att vi skapar moduler där kunden kan välja att på ett enkelt och effektivt sätt skapa egna mallar eller välja från förfabricerade sådana. Dessa moduler som erbjuder mallfunktionalitet skall kunna integreras med webbpubliceringsverktyget INVID Publisher .NET.

Därmed har vi funderat ut följande frågeställningar:

- Hur utvecklar man en modul för DotNetNuke?
- Vad ligger till grund för DotNetNukes arkitektur och uppbyggnad?

1.3 Syfte och mål

Som framgår under avsnittet "Problemformulering" är tidsåtgången stor då det för kunden är komplicerat att skapa och redigera sidor med det nuvarande mallhanteringssystemet i INVID Publisher .NET.

INVID Publisher .NET bygger helt på DotNetNuke (DNN) som utvecklats med servertekniken ASP.NET. Därför faller valet naturligt på Visual Studio.NET 2003 som utvecklingsmiljö. DNN 3.0 ramverket använder följande nyckeltekniker:

- Windows 2000 Server, Windows Server 2003
- Microsoft Internet Information Services (IIS) 5.0 eller högre
- Microsoft SQL Server 2000
- Active Server Pages.NET 1.1 (ASP.NET)
- Microsoft Visual Basic.NET (VB.NET)
- ActiveX Data Objects.NET (ADO.NET)
- Web Forms

DNN kan anpassas att använda t.ex. MySQL, PostgreSQL m.fl. som datakälla men vi har valt att använda Microsoft SQL Server 2000 dels eftersom DNN stödjer den utan ändringar och dels för att uppdragsgivaren använder sig av den sedan tidigare.

Vårt mål är att skapa en eller flera moduler med VB.NET som programmeringsspråk, ASP.NET version 1.1 som serverteknik och Microsoft Visual Studio .NET 2003 som utvecklingsmiljö. Vi kommer att utnyttja webbapplikationsramverket DNN för att skapa en modul som erbjuder kunden att på ett enkelt och effektivt sätt bygga nya sidor i webbpubliceringsverktyget INVID Publisher .NET.

För exekvering av frågor mot SQL servern rekommenderar DNN-teamet att stored procedures (lagrade SQL-procedurer) används. Fördelarna med stored procedures gentemot inline frågor är bl.a. högre prestanda och lågt underhåll. Inline frågor sänker prestanda och gör databaskod mindre översiktlig och därmed underhållskrävande.

DNN-modulen kommer att konstrueras så att den utnyttjar objektorienterade principer och sk. patterns (“mönster”) som bl.a. Microsoft har utvecklat för att designa och konstruera arkitektuellt goda .NET applikationer. DNN modulen kommer även att konstrueras så att den kan kommunicera med SQL Server 2000 och utnyttjar stored procedures för exekvering av SQL-frågor mot databasen.

1.4 Disposition

Denna rapport är skriven enligt den mall som Ingenjörshögskolan har för examensarbeten. Vissa koncept och tekniker som förklaras har inte översatts från engelska till svenska pga. det inte finns någon passande översättning.
Den teoretiska bakgrunden som besvarar vår frågeställning presenteras först. Vi behandlar sakligt DotNetNuke, moduler, bakomliggande designmönster och modeller, lagerakitektur dvs. Data Access Layer, Business Logic Layer, Presentation Layer och beskriver lagrens karakteristik och ingående beståndsdelar. Programmeringsexempel tas upp som beskriver hur man praktiskt kan utnyttja specifika namnutrymmen och klasser i webbapplikationsramverket DotNetNuke.
Därefter följer genomförandet där vi beskriver vårt tillvägagångssätt för att konstruera en s.k. DNN-modul. Vi beskriver även vilken utrustning som vi har använt. Efter genomförandet presenteras resultatet där vi beskriver de analyser som vi har utfört som ligger till grund för DNN-modulen.
Tillsist presenteras slutsatsen och diskussionen där vi beskriver bl.a. vilka teoretiska kunskaper vi har inhämtat.

2 Teoretisk bakgrund

2.1 Introduktion till DotNetNuke

DotNetNuke (DNN) är ett open-source (öppen källkod) ramverk för utveckling av bl.a. intranät och extranät för webben. Det är också möjligt att utveckla Content Management System vilket är ett webbsystem där man kan publicera och hantera webbsidor på webbplatser [1].
DNN är licensierat under en BSD licens (engelska för ”Berkeley Software Distribution”) vilket ger tredje part rätt att utan kostnad erhålla en kopia av mjukvaran samt relaterad dokumentation men med det enkla kravet att ursprungscopyright bevaras. En sådan licens ger också tredje part frihet att modifiera källkoden som distribueras med DNN för både kommersiella och icke-kommersiella syften [1].
DNN är byggt på Microsofts plattform för webbapplikationer vid namn ASP.NET och det programmeringsspråk som använts uteslutande för DNN är Microsoft Visual Basic.NET (VB.NET) [1].

Microsoft utvecklade tillsammans med Vertigo Software en webbapplikation eller starter- kit som kallades för IBuySpy Portal Solution Kit (IBS). Starter-kits visar hur programmerare kan lösa vanliga programmeringsproblem [2, pp.2-5].
För att IBS (som DNN bygger på) skulle attrahera så många som möjligt så släpptes den fri i två versioner, den ena en VB.NET-version och den andra en C#-version (uttalas ”c-sharp”) år 2001 [2, pp.2-5].
Syftet med IBS var att demonstrera hur man med det nya Active Server språket (ASP.NET) avsett för skarp webbutveckling, kunde utveckla en applikation enligt bästa tillvägagångssätt [2, pp.2-5].

En IBS-portal lät användare skapa nya och dynamiska webbsidor från grunden. Varje sida hade ett sidhuvud och tre innehållsramar. Portalens administratör kunde enkelt ladda in (injicera) nya moduler inom dessa ramar för att hantera webbinnehåll [2, pp.2-5].
En modul är i DNN eller IBS en slags miniapplikation för att hantera olika sorters innehåll som t.ex. text, bilder, länkar, nyheter, diskussionsforum etc. [2, pp. 2-5].
IBS erbjöd också funktionalitet för att administrera portalens användare, roller, rättigheter och sidor etc. Den fick bra genomslagskraft och antogs av Microsoftvärlden som en utmärkt referensapplikation [2, pp.2-5].

Figur 2-1 Nyinstallerad portal i DotNetNuke

Den 24:e december 2002 släppte Shaun Walker, grundaren till DNN, en modifierad VB.NET-version av IBS som var gratis att ladda ned på www.ibuyspyworkshop.com.
Det var från början ett försök att modifiera IBS så att den kunde användas av olika amatörmässiga sportorganisationer och till slut säljas till dem.
Det blev senare uppenbart enligt Walker att applikationen inte skulle kunna säljas kommersiellt bl.a. pga. det saknades finansiella medel.
Därför bestämde han sig för att utvärdera om det gick att driva sitt projekt icke-kommersiellt, han hade trots allt spenderat åtta till tio månader på att modifiera ursprungsversionen (IBS) [2, pp.2-5].
Efter ett antal releaser och förbättringar på IBS-kodbasen så döptes applikationen slutligen om till ”DotNetNuke” och en BSD licens antogs.
DNN har vuxit och enligt källor (www.dotnetnuke.com) så finns det idag mer än 225 000 registrerade användare. Den officiella community och utvecklingssajten för DNN är numera dotnetnuke.com (http://www.dotnetnuke.com) [3].

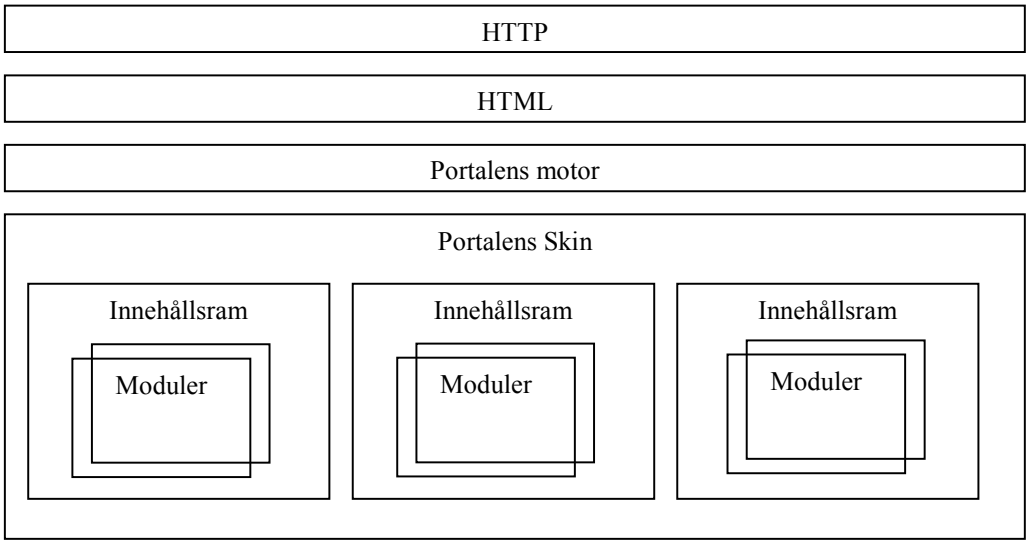
För närvarande har version 4.0.2 av DNN släppts vilket är avsett för ASP.NET 2.0 men versioner för ASP.NET 1.1 finns också att ladda ned som tex. 3.2.0, 3.2.1 och 3.2.2.
Figur 2-1 visar en nyinstallerad portal i DNN 3.2.2. Det är den version som rapporten främst utgår från men koncept och tekniker som presenteras kan också nyttjas för andra revisioner inom DNN version 3.
För anvisningar om installation av DNN 3.2.2 se ”Bilaga 1”.

2.2 Introduktion till Moduler

En modul är en användargränssnittskomponent som utför ”requests” (begäran) från en klient och genererar dynamiskt innehåll. En modul kan endast laddas in (injiceras) på ASP.NET-sidor och en specifik sida kan innehålla hur många instanser av modulen som helst. En viktig karakteristik för moduler är att de kan identifieras av dess typ. De predefinierade modulerna bestämmer vilken funktionalitet som kan erbjudas [2, pp.151].

En portal kan ses som en webbapplikation som sammanfogar innehåll från olika källor och förser klienter med presentationslagret (bestående av moduler) för informationssystemet [2, pp.151].

Figur 2-2 visar den grundläggande arkitekturen för en portal i DNN.
DNN måste utföra ett antal steg för att slutföra en sidbegäran [2, pp.152-154].

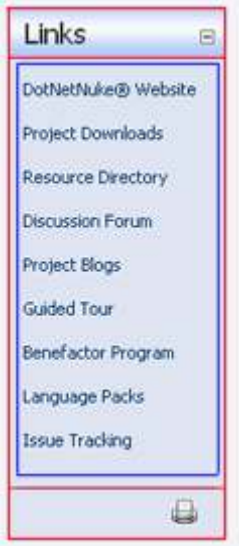


Figur 2-2 Portalens grundläggande arkitektur

Det första som sker vid en sidbegäran (från en klient) är att hämta modulerna för den specifika sidan. Det gäller alltså att bestämma var modulerna skall placeras på sidan och de säkerhetsroller som associeras med varje modul [2, pp.152-154].
Det andra steget är att undersöka vilken säkerhetsroll klienten har på den aktuella sidan och de säkerhetsroller som varje modul associeras med. En lista med moduler som har samma eller lägre säkerhetsnivå som klienten framställs [2, pp.152-154].
Det tredje steget innebär att de tillåtna modulerna injiceras till de motsvarande innehållsramarna för sidan [2, pp.152-154].

En sida är i DNN ett markup-dokument som består av ett antal innehållsramar som agerar värd för ett antal olika typer av moduler. Varje modul består av en titel, dekorationer och innehåll som produceras av modulen.
Vad gäller dekorationer så kan det inkludera olika grafiska ”Web forms”-komponenter som kan utföra specifik funktionalitet för den modulen [2, pp.152-154].

Web forms kan t.ex. vara kombinationer av HTML, kod och serverkomponenter som exekverar på en webbserver som kör Internet Information Services. Webbservern genererar till största delen det grafiska gränssnittet med HTML-kod som sänds ut till webbläsaren [7].



Dekorationer som omger en modul är modulens container. Genom containern kan webbklienter interagera med modulen för att t.ex. minimera/maximera den eller utföra mer eller mindre avancerade handlingar [2, pp.152-154].

Moduler kan placeras på en specifik sida genom att välja ur en lista med typer av moduler och specificera var de skall placeras på sidan. Figur 2-3 visar en links-modul och dess innehåll vilket är det blåa innersta inringade området och det röda yttre området är modulens container.

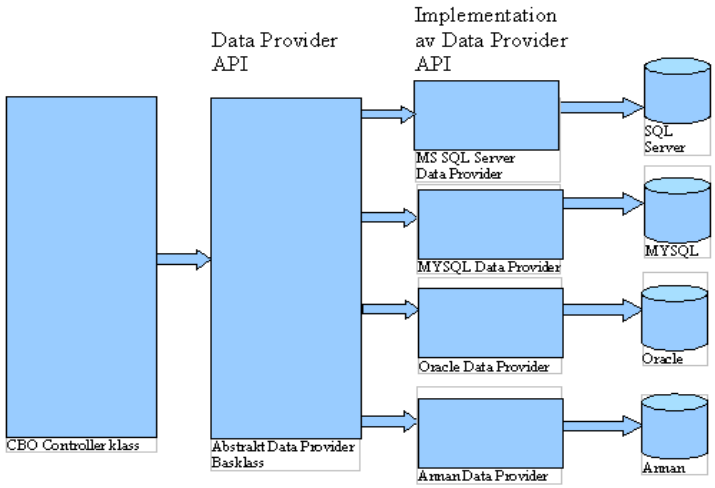
Figur 2-3 Exempel på modul

2.3 Active Server Pages.NET

Active Server Pages.NET eller ASP.NET är en integrerad webbplattform som låter designers och utvecklare skapa avancerade webbapplikationer med utmärkt stöd för bl.a. cachning, säkerhet och prestanda. ASP.NET är byggt på Microsofts .NET-ramverk vilket innebär att hela ramverket är tillgängligt för ASP.NET-utvecklaren. Det som gör ASP.NET ”bättre” än ASP är att det erbjuder utvecklaren att välja vilket programmeringsspråk som han önskar utveckla med. Valmöjligheterna inkluderar bl.a. VB.NET och C#. Det som gör att ASP.NET exekverar kod snabbare än skriptspråket ASP är att koden kompileras ned till en dll-fil bestående av MSIL-kod och cachas [5, pp.719-720,723].
MSIL är förkortning för Microsoft Intermediate Language vilket är en uppsättning av CPU-oberoende instruktioner som konverteras till ren binär kod av en Just-In-Time-Kompilator [6].

2.4 Providermodellen

DNN använder ett designmönster kallat providermodellen [4] för att tillåta ersättning av funktionalitet i applikationskärnan utan att ändra koden i kärnan. Genom att följa providermodellen blir det lättare att stödja olika typer av datakällor. Providermodellen som designmönster är utformad av Microsoft i ASP.NET Whidbey-projektet och har legat till grund för utveckling av flera av Microsofts starter-kits. Eftersom DNN bygger på Microsofts IBuySpy portal som följer providermodellen föll det sig naturligt för utvecklargruppen bakom DNN att anamma providermodellen.



Figur 2-4 Oberoende av datakälla

Figur 2-4 visar att DataProviderAPIn inte är beroende på detaljer hos implementationen av APIn.
En provider är ett kontrakt mellan API (Application Programming Interface) och BLL (Business Logic Layer). APIn är delad från implementationen av APIn. Vad detta vill säga är att själva koden som hör till en viss metod (t.ex. funktion eller procedur i VB.NET) separeras från definitionen eller deklarationen av metoden. I en särskild klass, `Dataprovider`, finns då endast definitioner av metoder medan kod som hör till dessa definitioner skrivs i en annan klass som t.ex. klassen `SqlDataProvider` (om användning av sql-datakälla). Tack vare denna abstraktion är APIn (t.ex. klassen `DataProvider`) i providermodellen inte beroende på detaljer hos implementationen (t.ex. klassen `SqlDataProvider`) av APIn [2, pp.187-189].

Enkelt kan syftet för modellen beskrivas som ”Utveckla objekt så att de inte beror på detaljer hos andra objekt.” [2, pp.188].

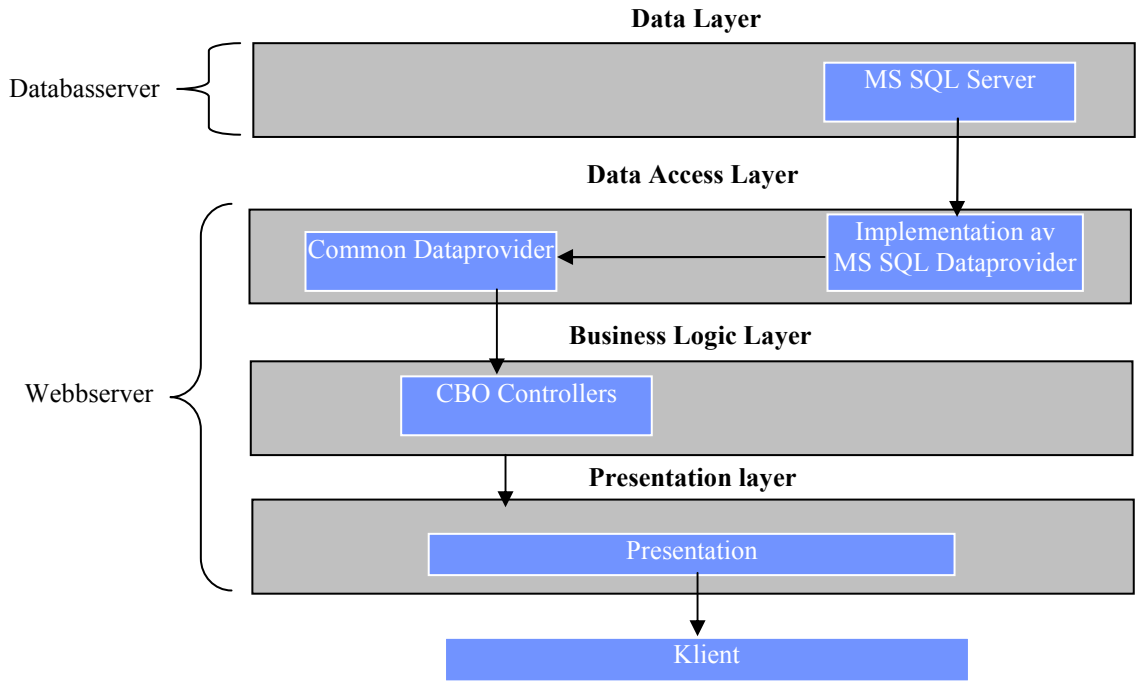
Flera områden inom DNN använder providermodellen, nämligen:

- DataProvider
- SchedulingProvider
- LoggingProvider
- HTMLEditorProvider
- SökProvider
- FriendlyUrlProvider

Första providern att använda providermodellen var dataprovidern.
Från början fanns bara stöd för Microsoft SQL Server och applikationskärnan var tätt knuten till dataskiktet [2, pp.189].

2.5 DotNetNukes Lagerakitektur

En webbklient som besöker en DNN-portal möts av ett grafiskt gränssnitt – ofta bestående av HTML- och JavaScript-kod. Det är webbservern som förser webbklienten med information och layout som presenteras. Data hämtas i en datakälla. Data och presentation är alltså skilda och fördelade på två separata servrar. Servrarna kan köras på samma fysiska maskin men välbesökta portaler använder ofta separata maskiner [2, pp.192-195].



Figur 2-5 DNNs lagerarkitektur

DNN delas in i tre applikationslager, vilket kan ses i Figur 2-5, dvs. applikationskod fördelas över tre distinkta lager som har tydligt definierade syften och klart avgränsade uppgifter. Applikationskod exekveras på en webbserver som levererar HTML- och JavaScript-kod till webbklientens webbläsare. Datalagret är det lager (databasen) som sköts av en databasserver.

2.6 Data Access Layer och Data Layer

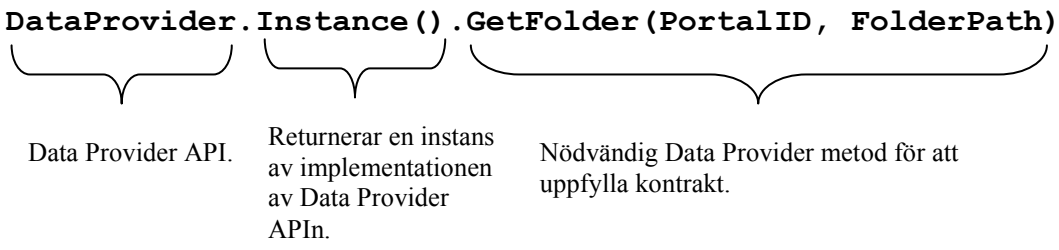
2.6.1 Data Access Layer

Data Access Layer (DAL) även kallat dataåtkomstskiktet förser Business Logic Layer (BLL) med datatjänster. Med hjälp av metoder som hör till klasser i DAL kan ett flöde av data hämtas och lagras i datakällan. Genom användandet av providermodellen ges teoretiskt stöd för ett oändligt antal datakällor [2, pp.198-199].

DAL delas in i två komponenter: [2, pp.198-199]

- Data Provider API. En abstrakt klass (t.ex. klassen `DataProvider`) som upprättar det kontrakt som implementationen av APIn ska uppfylla.
- Implementation (tex. klassen `SqlDataProvider`) av DataProviderAPI. Denna klass ärver från DataProviderAPI-klassen och uppfyller kontraktet genom att åsidosätta metoder som är nödvändiga för kontraktet. Detta för att få en giltig implementation av dataprovidern. Kod skrivs för denna implementation så att det är möjligt att kommunicera med den databas som skall användas av portalen.

Ett metodanrop till DataProviderAPI kan brytas ned i dessa element: [2, pp.198-199]



Anropet till `Instance()`-metoden returnerar en instans (ett objekt) av datatjänst-implementationen. En abstrakt metod är en metod som saknar implementation dvs. tillhörande kod. Då abstrakta metoder definieras för en klass blir klassen abstrakt och saknar implementation.
Abstrakta metoder, vilket markeras med nyckelordet `MustOverride`, är en del i kontraktet mellan API och implementationen av API och definieras enligt Figur 2-6. [2, pp.198-199]

```
Abstrakt funktion
Public MustOverride Function MetodNamn(ByVal variabel1 As [datatyp], _
    ByVal variabel2 As [datatyp], ByVal variabel...n As [datatyp]) As [datatyp]
Abstrakt procedur
Public MustOverride Sub MetodNamn(ByVal variabel1 As [datatyp], _
    ByVal variabel2 As [datatyp], ByVal variabel...n As [datatyp])
```

Figur 2-6 Abstrakta metoder

Implementationen (t.ex. klassen `SqlDataProvider`) av API förser abstrakta metoder med en implementation (`Overrides`) genom att ärva från klassen som representerar DataProviderAPIn (t.ex. klassen `DataProvider`) [2, pp.198-199].

Principen för implementation av abstrakta metoder visas av Figur 2-7.
De abstrakta metoderna måste bli ”overridden” (ersättas med en implementation), vilket markeras med nyckelordet `Overrides`. I detta fallet är även metoderna överlagrade, vilket markeras med nyckelordet `Overloads`.
En överlagrad metod används för att få olika versioner på en metod med samma namn, men accepterar olika antal argument eller argument med olika datatyper.

```
Implementation av abstrakt överlagrad funktion
Public Overloads Overrides Function MetodNamn(ByVal Variabel1 As [datatyp], _
    ByVal Variabel2 As [datatyp], ByVal variabel...n As [datatyp]) As [datatyp]
        ' Kod som kommunicerar med databas
End Function
Implementation av abstrakt överlagrad procedur
Public Overloads Overrides Sub MetodNamn(ByVal Variabel1 As [datatyp], _
    ByVal Variabel2 As [datatyp], ByVal variabel...n As [datatyp])
        ' Kod som kommunicerar med databas
End Sub
```

Figur 2-7 Implementation av abstrakta metoder

2.6.2 Data Layer

Leverantör av data, till datatjänst-implementationen instanserad av metoden, är det underliggande lagret kallat Data Layer (Dataskikt) som sköter kommunikationen med den datakälla man definierat enligt providermodellen. [2, pp.198-199]

2.6.3 Microsoft Data Access Application Block

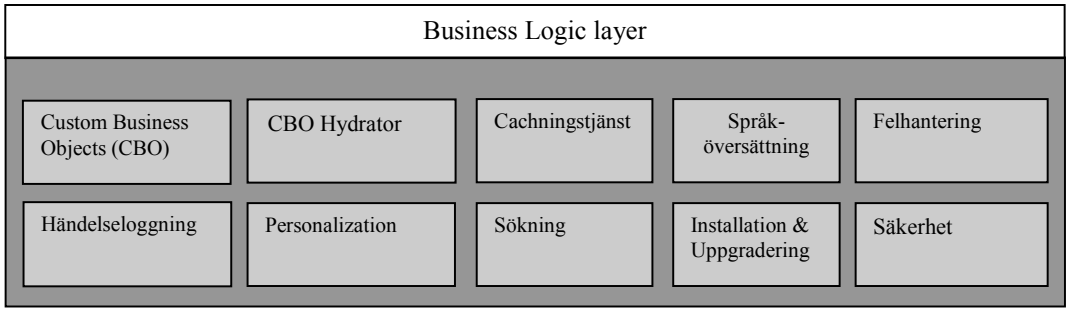
För att optimera prestanda och minimera specialanpassad kod använder DNN-projektet **Microsoft Data Access Application Block** (MSDAAB) som är en .NET-komponent som med hjälp av ADO.NET anropar stored procedures i databasen [2, pp.198-199].

2.7 Affärslogiklagret – Business Logic layer

2.7.1 Introduktion

Business Logic Layer (BLL) eller affärslogiklagret är ett av totalt tre lager eller skikt i DNNs arkitektur. Det erbjuder presentationslagret bl.a. åtkomst till datakälla, felhantering, händelseloggning, sökfunktionalitet, installation och uppgraderingar, säkerhet och möjlighet till språköversättning. Applikationskod som tillhör BLL körs på en webbserver som förser webbklienter med presentation av data.

BLL kommunicerar med dataåtkomstlagret eller DAL som det benämns och kan på så sätt få åtkomst till datakälla. Dock skall det påpekas att. pga. DNNs arkitektur så abstraheras den fysiska datakällan bort vilket gör att BLL inte är bunden till en specifik datakälla och behöver inte ha kännedom om vilken datakälla som används. Figur 2-8 beskriver vad BLL består av.



Figur 2-8 Ingående beståndsdelar i Business Logic layer

2.7.2 Custom Business Objects

Custom business objects eller CBOer som de benämns är en representation av ett objekt som utnyttjas av tillhörande CBO-kontroller-klass. Ett objekt är något som kan modelleras efter verkligheten och så är även en klass vilket är en mall för något. Ett objekt är således en instans av en specifik klass.

SearchResultsModuleInfo.vb

```
Imports System
Imports System.Configuration
Imports System.Data

Namespace DotNetNuke.Modules.SearchInput
    Public Class SearchResultsModuleInfo
        #Region "Private Members"
            Private _tabID As Integer
            Private _searchTabName As String
        #End Region
        #Region "Public Properties"
            Public Property TabID() As Integer
                Get
                    Return _tabID
                End Get
                Set(ByVal Value As Integer)
                    _tabID = Value
                End Set
            End Property
            Public Property SearchTabName() As String
                Get
                    Return _searchTabName
                End Get
                Set(ByVal Value As String)
                    _searchTabName = Value
                End Set
            End Property
        #End Region
    End Class
End Namespace
```

Figur 2-9 CBO- Info- klassen SearchResultsModuleInfo

Det karakteristiska för en CBO är att den bara innehåller egenskaper. För att kunna hantera en CBO (Info) finns en tillhörande CBO-kontroller-klass. Den innehåller metoder som är specifika för tillhörande CBO-Info-klass [2, pp.190-192].

Figur 2-9 är taget från en CBO-Info-klass som residerar i kärnan av DNN. Den visar hur en CBO Info klass programmerad i VB.NET kan se ut. Egenskaper skrivs med nyckelordet `Property` och enligt figuren finns det två publika egenskaper, `TabId` och `SearchTabName`. Vad som är viktigt att nämna är att CBO-Info-klasser endast används för att lagra värden (t.ex. 1,2,3, abc) [2, pp.190-192].

Figur 2-10 beskriver den tillhörande CBO-kontroller-klassen.
Vad som är specifikt för denna klass är att den innehåller affärslogiken dvs. metoder för att hantera ovanstående CBO-Info-klass.

SearchInputController.vb

```
Imports System
Imports System.Configuration
Imports System.Data


Namespace DotNetNuke.Modules.SearchInput
    Public Class SearchInputController
            Public Function GetSearchResultModules(ByVal PortalID As Integer) As ArrayList
                Return CBO.FillCollection(DataProvider.Instance().GetSearchResultModules(PortalID),
                        GetType(SearchResultsModuleInfo))
            End Function
    End Class
End Namespace
```

Figur 2-10 CBO-kontroller-klassen SearchInputController

2.7.3 CBO Hydrator

Om ett CBO Info objekt har många egenskaper kan det bli många rader kod om man skulle fylla det med data enligt traditionella förfaringssätt.
Traditionellt sett för att fylla ett objekt med data så skriver man `objektNamn.Egenskap=värde(n)`. Men detta har man tänkt på när man konstruerade DNN vilket innebär att man istället kan använda sig av något som kallas för CBO Hydrator. CBO Hydrator är en kraftfull tjänst som DNN erbjuder. CBO Hydrator är en samling metoder som man kan använda för att upptäcka egenskaper som ett CBO-Info-objekt har samt fylla det med korresponderande data. Detta kan åstadkommas med ett enda CBO-Info-objekt eller en samling av CBO-Info-objekt. För att fylla ett CBO-Info-objekt med data som returnerats från dataåtkomstlagret kan man således skriva såsom Figur 2-11 visar [2, pp.192-195].

```
CType(CBO.FillObject(DataProvider.Instance().GetFolder(PortalId,FolderPath ) _
        ,GetType(Services.FileSystem.FolderInfo),FolderInfo)
```

Figur 2-11 CBO Hydrator tillvägagångssättet

Enlig Figur 2-11 anropas en metod vid namn `FillObject()` i klassen `CBO` som tar två argument vilka är `IDataReader` och objektets typ (”objType As Type”).

`IDataReader` är ett interface som erbjuder utvecklaren att läsa strömmar som returnerats från exekvering av kommandon vid en datakälla [13] [14].
En `IDataReader` returneras från metoden `GetFolder()` vilket residerar i en instans av implementationen av dataprovidern, `DataProvider`.
Returtypen för metoden `FillObject()` är `Object` vilket innebär att vi måste konvertera objektet som returneras till typen `FolderInfo` (vilket görs med `CType()`). Presentationslagret förväntar sig nämligen ett CBO-Info-objekt av typen `FolderInfo` [2, pp.192-195].
Fördelen med detta tillvägagångssätt är att man bl.a. minskar antalet rader kod och det blir ett bra underlag för prestanda eftersom CBO Hydrator cachar CBO- Info-objektets egenskaper. Nästa gång man gör samma anrop enligt Figur 2-11 behöver inte `FolderInfo` -objektets egenskaper upptäckas eftersom de kan hämtas från cachen [2, pp.192-195].
Själva upptäckten av egenskaper sker med hjälp av bl.a. klasser i `System.Reflection`-namnutrymmet i .NET-ramverket [2, pp.192-195].

```
Objekt.Egenskap1=Värde(n)
Objekt.Egenskap2=Värden(n)
Objekt.Egenskap3=Värden(n)
Objekt.Egenskap4=Värden(n)
Objekt.Egenskap5=Värden(n)
```

Figur 2-12 Fylla egenskaper med värden enligt traditionell metod

Det standardmässiga tillvägagångssättet visas enligt Figur 2-12 ovan och har naturligtvis inte de fördelar som nämts om CBO Hydrator tillvägagångssättet. Här innebär t.ex. ”Egenskap1” namnet på egenskapen som ska fyllas med ett värde eller en samling av värden.
Dock måste det förtydligas att CBO-Info-objektets egenskaper i termer av namn och datatyp måste matcha de namn och datatyper som finns i datakällan för att CBO Hydrator tillvägagångssättet skall fungera korrekt [9, pp.30].
En egenskap vid namn förnamn av typen sträng i ett CBO-Info-objekt måste alltså kunna matchas till ett fält i datakällan.

2.7.4 Cachningstjänst

DNN erbjuder en ”tjänst” för att cacha objekt som användas ofta bl.a. för att minska antal anrop till datakälla och på så sätt öka prestanda. Det finns en specialiserad `DataCache` klass i DNN vilket erbjuder utvecklaren ett antal metoder för att hantera applikationens cache. Den använder sig bl.a. av `System.Web.Caching`-namnutrymmet för att åstadkomma detta. Vid behov kan man placera, hämta, ta bort objekt och tömma host-, portal- eller tab (sida) cachen enligt tolkning av källkoden i DNN [9, pp.33].

Objekt som placeras i cachen blir förr eller senare utgångna vilket betyder att man inte har någon nytta av dem längre. Därför använder DNN sig av något som kallas för sliding expiration vilket innebär att om ett objekt inte har hämtats inom en viss tidsperiod så kasseras det. Vad som också är viktigt att notera är att om applikationen (DNN) startas om så återskapas cachen till dess tidigare tillstånd [9, pp.33].
Exempel på när cachen används är i samband med bl.a. filuppladdning, modulinställningar och språköversättning enligt källkoden i DNN [9, pp.33].

```
Placering av ett objekt av typen ArrayList i Cachen
Dim list As New ArrayList
DataCache.SetCache(“strKeyList”, list)

Hämtning av ett objekt av typen ArrayList från cachen
Dim list As New ArrayList
list=CType(DataCache.GetCache(“strKeyList”), ArrayList)

Borttagning av ett objekt av typen ArrayList från cachen
DataCache.RemoveCache(”strKeyList”)
```

Figur 2-13 Exempel på användning av DataCache klassen

Figur 2-13 beskriver hur objekt kan placeras, hämtas och tas bort från cachen. Varje objekt som ska cachas associeras med en nyckel (”strKeyList” i Figur 2-13) för att kunna veta vilket/vilka objekt som syftas på när de ska hämtas ur cachen. När man önskar tömma cachen väljer man mellan att tömma host-, portal- eller tab-cachen enligt källkoden. Host-cachen berör objekt som associeras med administrationsläget, portal – objekt som associeras med portalen t.ex. portalinställningar och tab – vilket berör objekt i samband med sidor.

2.7.5 Språköversättning

Att stödja flera språk vid utformning av gränssnitt är viktigt – särskilt om det kommer att användas av personer från olika länder som talar vitt skilda språk. DNN har blivit ganska spritt och används av organisationer och företag världen om. Det är med andra ord alltså nödvändigt att DNN måste stödja fler språk pga. dess internationella inblandning. För att göra gränssnittet oberoende av språk har man därför vid utvecklingen av DNN bl.a. utformat en API för språköversättning [2, pp.211].

För att kunna visa olika information för olika kulturer måste man kunna lagra översättningarna på ett smidigt men kraftfullt sätt. Det lagringssätt som är passande för detta ändamål är Windows Resource files (RESX) formatet vilket DNN använder i hög grad. Viktigt att notera är att översättning av språk endast gäller för statiska strängar och tecken och inte för dynamiskt innehåll som t.ex. skulle kunna lagras i en databas [10].

RESX-formatet använder sig av XML-Extensible Markup Language för att lagra översättningar. Dessa översättningar lagras som XML-taggar där varje attribut associeras med ett värde [2, pp.222-223].

```
<data name="Email.Header">
        <value>Email</value>
</data>
<data name="Telephone.Header">
        <value>Telephone</value>
</data>
```

Figur 2-14 XML från contacts-modulen

Figur 2-14 beskriver hur XML från en kontakt-modul i DNN kan se ut. Namn attributet i figuren benämns som resursnyckel. Denna nyckel används för att identifiera vilken text som hör ihop med vilket värde. I figuren tillhör exempelvis texten Email resursnyckeln Email.Header. Ändelsen Header står för `headertext` egenskapen för en s.k. `DataGrid`. Det grafiska gränssnittet för kontaktmodulen har en `DataGrid` med ett antal kolumner. En av dessa kolumner har en rubrik som har `headertext`- egenskapen namngiven till Email. För att DNN skall kunna veta att det är texten Email som skall placeras i denna kolumn eftersöks resursnyckeln Email.Header i RESX-filen (XML-filen). Denna typ av översättning är en översättning av lokala resurser [2, pp.222-223].

DNN kan utföra tre olika typer av översättningar. Den första typen är för resurser på applikationsbasis. Där ingår översättning av text som delas mellan många moduler t.ex. ord som ”ja”, ”nej”, ”uppdatera” eller ”avbryt”. Dessa resurser lagras i en RESX-fil i en katalog vid namn App_GlobalResources som befinner sig direkt under roten för en typisk installation av DNN [10] [2, pp.222-223].
Den andra typen av översättning är för lokala resurser. Dessa resurser är direkt relaterade till moduler vilket behöver översättning pga. dess unika funktionalitet. Även dessa resurser lagras i en RESX-fil. Exempel på detta ges av kontakt-modulen som beskrivits ovan. För att specificera vilket språk och land som den lokala resursfilen gäller för används notationen ”ModulNamn.språkkod-landskod.resx”. Namnet på en lokal resursfil för språket engelska i landet USA skulle då kunna bli ”ModulNamn.en-US.resx”. Enligt DNN måste den lokala resursfilen alltid placeras under modulens App_LocalResources katalog [10] [2, pp.222-223].
Den tredje och sista typen av översättning är för globala resurser.
Dessa resurser används av komponenter (t.ex. klasser) som inte har lokala resursfiler. Även globala resurser använder en RESX-fil för lagring [10] [2, pp.222-223].

```
Dim strHelp As String

strHelp=Services.Localization.Localization.GetString(ModuleActionType.ModuleHelp,
                Services.Localization.Localization.GlobalResourceFile)
```

Figur 2-15 Exempel på användning av globala resurser

Figur 2-15 visar hur ordet ”Help” hämtas för den aktuella språkinställningen för portalen. Genom att göra ett anrop till metoden `GetString()` i klassen `Localization` och specificera argumentet `ModuleActionType.ModuleHelp` och argument två som är `Services.Localization.Localization.GlobalResourceFile` hämtas ordet ”Help” från filen GlobalResources.resx som är lagringsplatsen för globala resurser.

2.7.6 Felhantering

Det finns många olika typer av fel som kan uppstå i programmerings-sammanhang vad gäller utveckling av såväl windows som webbapplikationer. Ett scenario som kan tänkas medföra att fel uppstår är t.ex. när användaren matar in värden som är utav fel typ, t.ex. bokstäver istället för siffror.
Men fel kan också bero på ”buggar” i programkoden som gör att en viss uppgift misslyckas.

DNN erbjuder fyra metoder för att hantera fel som uppstår.
Dessa är `ProcessModuleLoadException()`, `ProcessPageLoadException()`, `LogException()` samt `ProcessSchedulerException()`. Syftet med dessa är bl.a. att kunna vidta åtgärder när fel uppstår, spara egenskaper för felet till en fellogg och visa vänliga felmeddelanden för användaren. Dessa metoder har flera överlagrade metoder vilket betyder att man kan anropa dessa med olika argument. Syftet med metoden `ProcessModuleLoadException()` är att logga fel som uppstår inom en modul och att visa vänliga felmeddelanden där modulen befinner sig på sidan [2, pp.216-221].

```
Try
      Dim strFriendlyErrorMessage As String
      strFriendlyErrorMessage=”Ett fel inträffade, försök ladda sidan igen.”
      Dim visaFel As Boolean = True


      ' Kod som orsakar fel…
      programinstruktion


Catch ex As Exception
      ProcessModuleLoadException(strFriendlyErrorMessage, Me, ex, visaFel)
End Try
```

Figur 2-16 Strukturerad felhantering samt användning av metoden ProcessModuleLoadException

Figur 2-16 beskriver hur man kan göra för att hantera ett fel som uppstår pga. en felaktig programinstruktion. Instruktionen är omgärdad av ett kodblock som kallas `Try Catch End Try` block vilket är ett exempel på hur man använder strukturerad felhantering i VB.NET [5, pp.256].
När en eller flera programinstruktioner orsakar ett fel inom detta block kommer instruktioner inom `Catch ex As Exception…End Try` att exekveras [5, pp.256].
När fel uppkommer inom Try…Catch-blocket kommer därför metoden `ProcessModuleLoadException()` att anropas med fyra argument.
Det första argumentet `strFriendlyErrorMessage` specificerar det vänliga felmeddelande som ska visas för användaren. Det andra argumentet specificerar vilket kontrollobjekt som felet berör. Det tredje argumentet skickas med så att man i felloggen kan se detaljer om felets natur (egenskaper).
Det sista argumentet som ska specificeras är om det vänliga felmeddelandet skall visas för användaren eller inte [2, pp. 216-221].

Syftet med metoden `ProcessPageLoadException()` är samma som `ProcessModuleLoadException()`.
Det som särskiljer `ProcessPageLoadException()` från `ProcessModuleLoadException()` är att man kan använda den på ASP.NET-sidor istället för inom DNN-moduler [2, pp. 216-221].
Metoden `LogException()` används endast för att spara ned fel till felloggen. `ProcessSchedulerException()` används för att logga fel som inträffar inom schemalagda uppgifter [2, pp. 216-221].

2.7.7 Händelseloggning

DNN erbjuder en tjänst som gör det möjlighet att logga olika typer av händelser. Händelser som inkluderas är loggning av felmeddelanden, systemhändelser och säkerhetsmässiga aspekter. Händelseloggningstjänsten använder en loggningsprovider vilket bygger på providermodellen. Händelser lagras i ett XML-dokument. Eftersom tjänsten är konstruerad med tanke på providermodellen så kan man byta typ av datakälla smidigare [2, pp.208-210].

När det gäller loggningen så kan meddelanden klassificeras till två typer, systemhändelser och felmeddelanden. Systemhändelser relaterar till något specifikt som har inträffat inom DNN och som inte är ett fel utan t.ex. att en sida har skapats. Loggningstjänsten kan konfigureras från logg-granskningssidan där ett antal inställningar kan göras för varje klassificering [2, pp.208-210].

Det finns tolv klasser i namnutrymmet `DotNetNuke.Services.Log.EventLog` som kan användas för att hantera loggningen. Men det är två klasser av de tolv som erbjuder mest funktionalitet och de är `EventLogController` och `ExceptionLogController`.
`EventLogController` används för att logga systemhändelser, rensa loggen, ta bort vissa händelser m.m. Den har flera överlagrade metoder.
`ExceptionLogController` används för att logga fel som har inträffat [2, pp.208-210].

```
objEventLog.AddLog(objTabInfo, PortalSettings, UserInfo.UserID, UserInfo.Username, _
       Services.Log.EventLog.EventLogController.EventLogType.TAB_CREATED)
```

Figur 2-17 Exempel på loggning av systemhändelse med metoden AddLog

Figur 2-17 beskriver tillvägagångssättet för att logga systemhändelsen att sida har skapats. Metoden `AddLog()` anropas i klassen `EventLogController` (objektet `objEventLog`) och tar fem argument. Det första argumentet specificerar detaljer om sidan som skapats (t.ex. sidnamn) som residerar i objektet `objTabInfo`. Det andra är egenskapen `PortalSettings` som är av typen `DotNetNuke.Entities.Portals.PortalSettings` och som erbjuder metoder och egenskaper för olika inställningar för portalen. Det tredje är ett identifikationsnummer för användaren och det fjärde argumentet specificerar användarnamnet så att man i loggen kan se vem som skapade sidan.
Det sista argumentet specificerar vilken typ av händelse – i detta fallet att en sida har skapats [2, pp.208-210].

2.7.8 Personalization

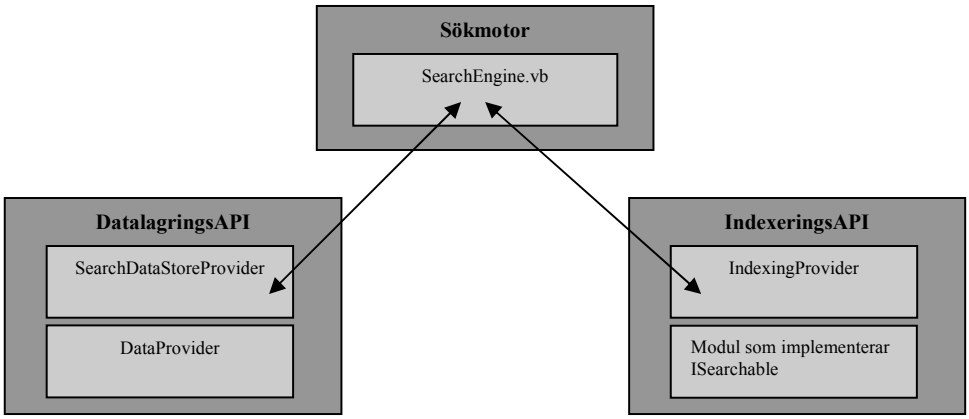
HTTPmodulen personalization laddar en användarprofil och skapar ett XML objekt i början på en sidbegäran och sparar profilen i slutet av en sidbegäran. För att lagra personlig information om användaren kan man använda sig av personalizationklasserna under Personalization-katalogen i DNN [2, pp.240].

2.7.9 Sökning

Att finna den information användare söker har naturligtvis stor betydelse. Därför har man i DNN utformat en sök-API för att söka och indexera information i portalen. SökningsAPIn är uppdelad i tre delar; sökmotor, datakälla och indexering. Sök-APIn implementerar providermodellen och mediatormönstret [2, pp.254-256].
Ett mönster (designmönster) är en beprövad lösning för återkommande problem inom mjukvaruutveckling i specifika situationer [12].

Mediatormönstret är ett mönster som definierar ett objekt som används för att styra hur en uppsättning eller grupp av objekt kommunicerar med varandra [11].

Ett exempel på detta är hur flygplan kan kommunicera med varandra genom ett flygledartorn. Flygledartornet samordnar flygplanens rörelser på marken och i luften. Utan denna samordning skulle kommunikationen mellan flygplanen bli överflödig, svårbegriplig och ineffektiv och sannolikt leda till flygkatastrofer.



Figur 2-18 Sökmotorramverket, samt beskrivning av hur "objekten" kommunicerar med varandra

Figur 2-18 beskriver hur de olika "objekten" kommunicerar med varandra. Själva sökmotorn finns implementerad i klassen `SearchEngine` i namnutrymmet `DotNetNuke.Services.Search`. Den erbjuder metoder för att anropa indexeringsprovidern - tjänsten som indexerar innehåll, - som sedan lagrar resultatet med hjälp av SearchDataStoreProvider – tjänsten som används för att lagra resultatet i datakälla [2, pp.254-256].

Vad som inträffar när en sökning skall göras är att metoden `IndexContent()` anropas i klassen `SearchEngine`. Denna metod anropar sedan en hjälpmetod vid namn `GetContent()` i samma klass som itererar genom alla portaler som webbapplikationen förfogar över. För varje portal som hittas anropas metoden `GetSearchIndexItems()` med argumentet portalid som är ett heltal. Metoden `GetSearchIndexItems()` är en del av indexeringsprovidern – vilken bara är en abstrakt klass [13].

Själva implementationen av metoden utförs i klassen `ModuleIndexer`. Vad metoden `GetSearchIndexItems()` i klassen `ModuleIndexer` sedan gör är att den för en viss portal hämtar en lista över moduler som residerar i den och som implementerar interfacet `ISearchable`. För varje modul anropas sedan metoden `GetSearchItems()` – vilket själva modulen måste ha en giltig implementation för [13].

Om en modul implementerar interfacet `ISearchable` korrekt så kan indexeringsprovidern anropa modulen och få tillbaka sökresultat. Sökresultatet returneras i form av ett objekt av typen `DotNetNuke.Services.Search.SearchItemInfoCollection` som helt enkelt är en samling (”vektor, matris”) av `DotNetNuke.Services.Search.SearchItemInfo`-objekt.
Varje `SearchItemInfo` objekt innehåller egenskaper som kan användas för att lagra t.ex. titel, författare, publiceringsdatum etc. för sökningen. Det är alltså med andra ord modulens ansvar att förse indexeringsprovidern med sökresultat [2, pp.254-256].

2.7.10 Installation och uppgradering

Vid ny installation av DNN eller uppgradering av detsamma används metoder i en klass vid namn `Upgrade`. Vid ny installation anropas metoden `InstallDNN()` som exekverar tre sql-skript som skapar tabellerna, stored procedures och fyller tabellerna med lämplig data i databasen. Dessa tre är DotNetNuke.SetUp.SqlDataProvider, DotNetNuke.Schema.SqlDataProvider och DotNetNuke.Data.SqlDataProvider.
För uppgradering av DNN efter ny installation används metoden `UpgradeDNN()` som exekverar en samling sql-skript som uppgraderar den databas som webbapplikationen (DNN) använder [2, pp.199-200].

Vad som bör påpekas är att anrop av metoder och exekvering av sql-skript sköts automatiskt vid installation/uppgradering av DNN och behöver inte installatören tänka på [2, pp.199-200].

2.7.11 Säkerhet

Säkerhet i DNN 3.0 har implementerats med tanke på framtiden och ASP.NET 2.0. Forms-baserad inloggning stöds och tillägg från tredje part utvecklare finns som gör att windowsautentisering kan användas i äldre versioner än DNN 3.0. När DNN 3.0 utvecklades hade Microsoft inte ännu släppt ASP.NET 2.0 vilket innebar att man utvecklade en likartad version på den medlemskap/roll provider som finns i ASP.NET 2.0 men för användning med ASP.NET 1.1. Den medlemskap/roll provider som finns i DNN 3.0 och dagens version hanterar all interaktion mot databas för hantering av användare och säkerhetsroller. Fördelen med denna provider är bl.a. att andra applikationer kan dela användare- och roll- information med DNN [2, pp.201-202,206].

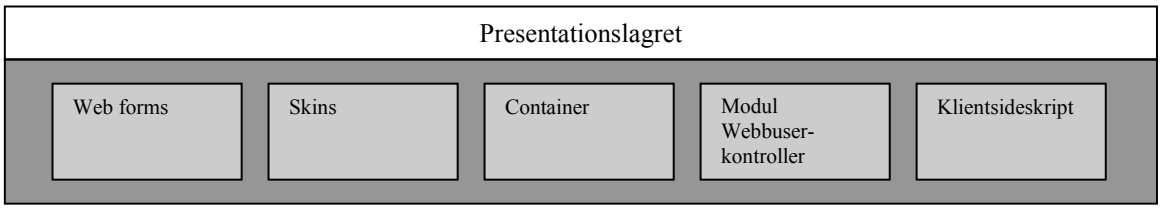
I namnutrymmet `DotNetNuke.Security` finns det klasser som hanterar auktorisering och autentisering för bl.a. sidor, moduler, roller och kataloger. Man kan bl.a. hämta information om rättigheter för sidor – vem får se dem, vilket bl.a. kan åstadkommas med klasserna `TabPermissionController`, `TabPermissionInfo` och `TabPermissionCollection`.

För att hämta, ändra, ta bort eller lägga till modulrättigheter används klasserna `ModulePermissionController`, `ModulePermissionInfo` och `ModulePermissionCollection`.
Ovanstående klasser borde i sammanhanget vara några av de mest använda klasserna i namnutrymmet `DotNetNuke.Security`. Det finns dock åtskilliga andra klasser som kan användas för säkerhetsrelaterade uppgifter.
Gemensamt för ovannämnda klasser är även att de bygger starkt på provider-modellen vilket ju som tidigare sagt underlättar byte av datakälla vid behov [2, pp.201-202,206].

2.8 Presentationslagret

2.8.1 Introduktion

Presentationslagret (PL) är det lager som erbjuder ett grafiskt gränssnitt gentemot användaren. De element eller komponenter som det består av är bl.a. Web forms, skins, container, modul webbuser-kontroller samt klientsideskript. Figur 2-19 visar indelningen av presentationslagret.



Figur 2-19 Presentationslagrets ingående beståndsdelar

2.8.2 Web forms

Web forms är utvecklat av Microsoft (Ms) och bygger på .NET ramverket och ASP.NET och erbjuder utvecklare att designa och programmera interaktiva webbsidor [14].

Den utvecklingsmiljö som Microsoft lanserat för bl.a. utveckling av aktiva webbsidor är Visual Studio, som erbjuder RAD-verktyg (Rapid Application Development), vilket underlättar och ökar utvecklingstakten av webbapplikationer [14].

ASP.NET är en teknologi där kod körs på en webbserver som genererar HTML-sidor som skickas till en klient vid sidbegäran. Källkod kompileras ned till MSIL (Microsoft Intermediate Language) [6] [14].

Web forms-sidor kan programmeras med de språk som stödjer .NET common language runtime (CLR). I dagsläget stöds bl.a. Ms Visual Basic.NET, Ms Visual C# .NET och Ms JScript.NET [14].

Vanliga grafiska komponenter som t.ex. knappar, listor och kryssrutor kan enkelt placeras på Web forms-sidor (vanligtvis uttryckt ASP.NET-sidor) genom att skriva koden för respektive komponent eller med hjälp av de verktyg som finns till hands i Visual Studio dra in komponenten till design-ytan.
Det finns också ett stort utbud av komponenter från tredje part utvecklare som kan användas på Web forms-sidor [14].
Web forms sidor är alltså det grafiska gränssnittet som användaren möts av vid sidbegäran. En Web forms sida är uppdelad i två komponenter; den visuella komponenten vilket är en fil med HTML och programmerbara grafiska komponenter och den logiska komponenten vilket är en fil med programmeringslogiken för sidan. Den logiska komponenten refereras till som ”code-behind”-filen och har filändelsen aspx.vb (VB.NET) eller aspx.cs (C#) beroende på vilket programmeringsspråk som valts. Den visuella komponenten har filändelsen aspx [14].
Varje Code-behind-fil som hör ihop med respektive aspx-fil i ett projekt kompileras ned till ett dynamisk länkbibliotek, dll-fil, bestående av MSIL-kod. Aspx-filer kompileras också men på ett något annorlunda sätt. Första gången en användare begär en aspx-sida så genererar ASP.NET en klass-fil som representerar sidan och kompilerar den till en dll-fil. Vid en sidbegäran så exekveras dll-filerna på webbservern som producerar HTML för en specifik sida [14].

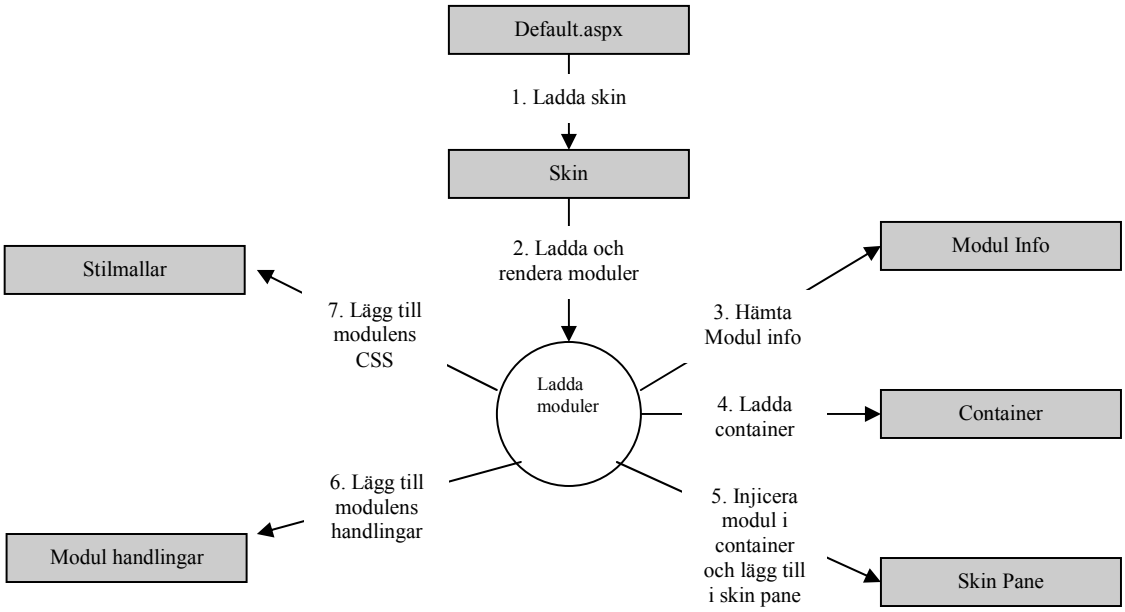
Code-behind filen (klassen `CDefault`) för Default.aspx är huvudsida i DNN och ärver från klassen `DotNetNuke.Framework.PageBase` istället för den sedvanliga klassen `System.Web.UI.Page`. Detta innebär att klassen `CDefault` får ytterligare metoder och egenskaper som är specifika för DNN.
Vad som också karakteriserar huvudsidan är att den är deklarerad med nyckelordet `MustInherit` vilket innebär att instanser av klassen `CDefault` inte kan skapas, med nyckelordet `New`, utan endast referenser till den kan göras.

2.8.3 Skins

Skins används av DNN för att på ett smidigt men kraftfullt sätt ändra den grafiska designen på sidor. Skins kan appliceras för sidor på host-, portal -nivå eller för alla sidor. I ett skin sätts identifierade attribut som gör att DNN dynamiskt kan ladda in data på de sektioner på sidan där attributen definieras [2, pp.331-336].

För att skapa sidor med tre innehållsramar kan man således skapa ett skin, antingen med hjälp av HTML eller med webbuser-kontroller, där man sedan definierar tabeller för dessa innehållsramar. Även statiska data som t.ex. bilder kan inkluderas i ett skin [2, pp.331-336].

Skins kan användas mycket tack vare den treskiktade arkitektur som DNN baseras på. Därmed kan man enklare separera applikationslogik från användargränssnittet. För att ändra designen på sidor kan man således ändra skin genom att göra en inställning i portalen. Det är möjligt att skapa egna skins i DNN version 2.0 eller högre [2, pp.331-336].



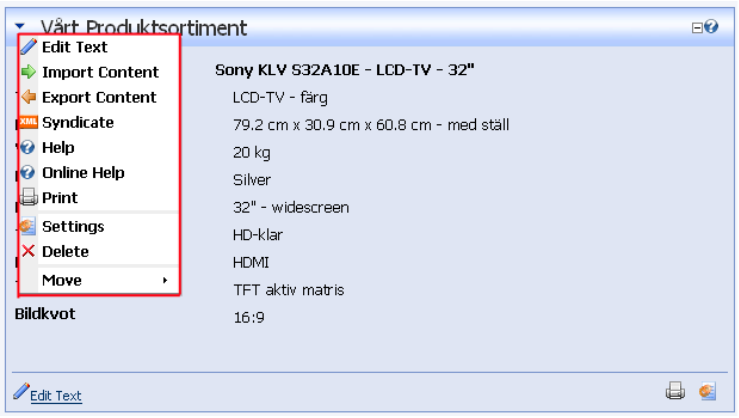
Figur 2-20 Översiktlig beskrivning av sidladdningsprocessen i DNN

Vad som inträffar när en sidbegäran tas emot är att Web forms-sidan Default.aspx i DNN laddar den specifika sidan (Figur 2-20 beskriver översiktligt sidladdningsprocessen i DNN). Den undersöker vilket skin som är valt och laddar sedan skin-webbuserkontrollen. Skin-kontrollen måste ärva från klassen `DotNetNuke.UI.Skins.Skin` för att en sida skall kunna visas. Denna klass hanterar i princip hela sidladdningsprocessen [2, pp.196-197].

Vad `Skin` klassen först utför är att den itererar igenom alla moduler som är associerade med den valda sidan (sida som man önskar visa).
Sedan undersöker `Skin` klassen om en specifik modul implementerar interfacet `DotNetNuke.Entities.Modules.IActionable`. Om modulen gör det så kontrollerar `Skin` klassen vilka handlingar som definieras av modulen och lägger till dem till containern [2, pp.196-197].
Sedan undersöker `Skin`-klassen om det finns en fil som heter module.css i modulens installationskatalog på disk. Om den filen existerar så adderas en `HtmlGenericControl` till den valda sidan för att referera till modulens stilmall (CSS). Ovanstående steg inträffar inom `Init`-händelsen i `Skin`-klassen.
Den slutgiltiga renderingen av innehållet i modulen ansvarar modulen själv för. Slutligen renderar Default.aspx stilmallens länkar baserat på portalens konfiguration och det valda skinnet [2, pp.196-197].

2.8.4 Container

En container är en skin-definition som kan appliceras på modulnivå.
Därmed kan den grafiska designen för moduler ändras enkelt men kraftfullt.
Att skapa en container påminner mycket om att skapa ett skin.
Attribut definieras på liknande sätt som för skin och grafiska komponenter som är gemensamma för alla moduler definieras. När en sida besöks i DNN injiceras varje modul i dess container vilket betyder att när man skapar en container måste attributet ”CONTENTPANE” specificeras [2, pp.331,336].
Det finns dock ett särskilt krav som måste uppfyllas vid konstruktion av en container vilket är att en ”actions”-kontroll måste finnas med i containern. Genom ”actions”-kontrollen kan användare administrera modulens funktionalitet och därmed agerar den som en länk mellan modulen och DNN-ramverket, för exempel på ”actions”-kontroll se Figur 2-21 (röd markering) [2, pp.348-349].



Figur 2-21 Exempel på ”Actions”-kontroll för Text/Html-modul

Med ”actions”-kontrollen (meny) kan användare bl.a. administrera modulens innehåll, få specifik hjälp för modulens funktionalitet, skriva ut modulens innehåll, ändra modulens inställningar, ta bort modulen från sidan eller flytta modulen till en annan sektion (innehållsram) på sidan. Dessa handlingar är några av de standardhandlingar som en modul normalt kan utföra från ”actions”-kontrollen. Det finns även möjlighet att addera fler handlingar till denna kontroll vilket kan vara ännu mer specifika för en moduls funktionalitet [2, pp.348-349].

När man skapar en container kan man också bl.a. bestämma var modulens titel, modulikon, utskriftsikon skall positioneras och om de överhuvudtaget skall visas [2, pp.348-349].

För att designa sin portal för företaget, organisationen eller för hemmabruk med DNN kan man således skapa en egen grafisk design och profil med hjälp av skin- och container- konceptet. Design av skin och container bör också ske i samklang med varandra eftersom de bör komplettera varandra.

2.8.5 Modulers webbuser-kontroller

Moduler är byggblock som kan användas för att hantera information på sidor. Varje modul har designats med en specifik funktionalitet i åtanke. Det finns t.ex. moduler som erbjuder funktionalitet för att hantera text och bild, länkar, kontaktinformation, annonser m.m. [2, pp.299-300].

DNN version 3.2.2 erbjuder totalt 21 moduler, bortsett från de moduler som finns i admin- och host- gränssnittet, som kan hantera olika sorters information och ”lösa” olika uppgifter [2, pp.299-300].

Varje modul består av ett antal webbuser-kontroller, om åtminstone en webbuser-kontroll, vilket en användare interagerar med för att utföra specifika uppgifter. Webbuser-kontrollerna är med andra ord det grafiska gränssnitt som användaren kommer i kontakt med, och de kommunicerar med underliggande DNN-skikt (BLL och DAL) för att bl.a. hämta, uppdatera, ta bort och lägga till ny information i databasen [2, pp.299-300].

En modul kan ha webbuser-kontroller som används som någon utav följande fem typer:

- View-kontroll
- Edit-kontroll
- Settings-kontroll
- Anonymous-kontroll
- Admin-kontroll
- Host-kontroll

View-kontrollen är den webbuser-kontroll som användare i första hand kan se och interagera med och som kommunicerar med databasen för att bl.a. visa information som modulen hanterar. Det är möjligt att ha flera view-kontroller i en modul [2, pp.299-300].

Med edit-kontrollen kan användare redigera den information som modulen är tänkt att hantera och visa. Man kan t.ex. uppdatera, ta bort eller lägga till ny information [2, pp.299-300].

Settings-kontrollen används för att göra modulspecifika inställningar som t.ex. påverkar hur information visas i view-kontrollen [2, pp.299-300].

Anonymous-kontrollen kan användas för anonym granskning av den information modulen hanterar [2, pp.299-300].

Admin-kontrollen är tänkt att användas av portalens administratör för att hantera information [2, pp.299-300].

Host-kontrollen används av superanvändaren, host, vilket hanterar t.ex. information och portalinställningar på en hög och någorlunda kritisk nivå. Om modulen t.ex. hanterar kritiska filer på webbservern kan det vara bra om inte en användare med låg säkerhetsklassning utför detta [2, pp.299-300].

Från en modulutvecklares perspektiv är det dock brukligt att använda webbuser-kontroller som används som typerna view, edit och settings.

Vad som är gemensam karakteristik för codebehind-klasserna för view-, edit- och settings- kontrollerna är att de är deklarerade med nyckelordet `MustInherit`, vilket tidigare har förklarats. En moduls view och edit kontrolls codebehind-klass ärver från klassen `Entities.Modules.PortalModuleBase`. Detta innebär att view- och edit- kontrollen får tillgång till en stor samling egenskaper och metoder specifika för moduler.

Egenskaper som view- och edit- kontrollen får tillgång till inkluderar bl.a. modulens id (identifikationsnummer), autentiserad användares id, sidans id, portalens id och portalinställningar [2, pp.300,302-303].

Det som särskiljer codebehind-klassen för settings-kontrollen från klasserna för view och edit är att den ärver från klassen `Entities.Modules.ModuleSettingsBase`. Detta innebär att settings-kontrollen får egenskaper och metoder avsedda för att hantera modulinställningar [2, pp.300,302-303].

Ytterligare en karakteristik för codebehind-klassen för view-kontrollen är att den kan implementera följande tre interface, vilket är valbart: [2, pp.205,304]

- `IActionable`
- `IPortable`
- `ISearchable`

För att kunna lägga till handlingar till ”actions”-kontrollen för en container för en specifik modul måste modulen implementera interfacet `IActionable` korrekt. Den skrivskyddade egenskapen `ModuleActions` i codebehind-klassen för view-kontrollen brukar implementera detta interface. Där finns bl.a. skriven kod för att man med ”actions”-kontrollen skall kunna navigera till edit - kontrollen [2, pp.205,304].
För att kunna exportera och importera information från en modul med ”actions”- kontrollen måste man implementera interfacet `IPortable`.
Två metoder, en för export och en för import skrivs för implementering av detta interface [2, pp.205,304].
Interfacet `ISearchable` används för att nyttja den sökmotor som DNN har till förfogande för att söka information i moduler [2, pp.205,304].

Det ska också förtydligas att ovanstående interface är valbara vilket betyder att de inte behöver användas i en modul [2, pp.205,304].

2.8.6 Klientsideskript

Med klientsideskript i form av JavaScript som körs på klientsidan kan en mer interaktiv och precisare upplevelse av det grafiska gränssnittet erbjudas. Detta innebär också att webbservern avlastas då kod körs hos klienten. Ett typexempel på när JavaScript passar ändamålet utmärkt är när man önskar validera indata. Men JavaScript kan också användas för uppgifter av mer grafisk natur t.ex. animation.

Grafiska komponenter i DNN som t.ex. kalenderkomponenten använder sig i hög grad av JavaScript. JavaScript som används av DNN lagras i skriptfiler under ”js” katalogen i roten för en typisk DNN-installation. En del skins använder sig av JavaScript och då befinner sig dessa skript under skinnets installationskatalog. Samma gäller för moduler som använder sig av JavaScript, de skript finns då under modulens installationskatalog [2, pp.196-197].

DNN använder en kombination av JavaScript och serversidekod för att bl.a. göra det möjligt att dra-och-släppa moduler från en innehållsram till en annan på sidor. Den visuella renderingen av flyttningen av en modul från en innehållsram till en annan görs på klientsidan medans anrop görs till serversidan för att slutligen uppdatera databasen med information om den nya sektionen som modulen flyttades till [15].

DNNs klient-API består av både klientsideskript och serversidekod som samverkar för att t.ex. möjliggöra dra-och-släpp funktionalitet av moduler på klientsidan [16]. Vad det innebär är alltså att klientsideskript kan kommunicera med servern utan att en s.k. ”postback” behöver inträffa. Detta leder till en bättre och mer interaktiv användarupplevelse som tidigare nämnts.

3 Genomförande

3.1 Inlärningsfas

Att utveckla moduler för DNN kräver speciella kunskaper och färdigheter. Grundläggande kunskaper inhämtades med hjälp av instruktionsvideos producerade av Seabury Design (www.seaburydesign.com). För att få fördjupad förståelse om DNN och moduler studerade vi flera böcker däribland *Professional DotNetNuke ASP.NET Portals* och *Building Websites with VB.NET and DotNetNuke 3.0*. Vid flertalet tillfällen var vi dock tvungna att ha tillgång till direkta referenser för programmeringsuppgifter och vände oss därför till DNN-dokumentationen som medföljer källkoden. Vi märkte även vid flera tillfällen att det inte gick att få direkt hjälp av DNN-dokumentationen då vi istället var tvungna att söka svar på våra frågor hos andra källor. Vi gjorde efterforskningar på webbsajter som bland annat *http://forums.asp.net/90/ShowForum.aspx* vilket är en community-sajt för DotNetNuke-utvecklare som ”postar” svar på DNN-relaterade lösningar och problem.
Det har krävts ett stort mått av problemlösningsförmåga och kreativitet att finna lösningar på de problem som vi har haft under arbetets gång.

3.2 Utrustning

Förutom en dator med Windows XP/2000 krävs följande för att utveckla moduler till DNN:

- Microsoft Visual Studio 2002/2003/2005
- Microsoft SQL Server 2000/2005
- Microsoft Internet Information Services (IIS)
- DNN valfri version

Vi rekommenderar även att man använder:

- SQL Enterprise Manager
- Verktyget CodeSmith med mallar för DNN

3.3 Modulen Mallar

Eftersom den modul som är under utveckling åt uppdragsgivaren, INVID Jönköping AB, innehåller skyddad kod kommer en modul av liknande karaktär och funktion att beskrivas istället, se avsnittet ”Modulen Blog”.

Vi analyserade och diskuterade de problem som den gamla modulen utgjorde. Efterhand som vi utvecklade mallmodulen fick vi bra idéer och synpunkter på förbättringar från vår handledare och andra personer från utvecklings-avdelningen. Vi hade också regelbundna träffar med utvecklarna på företaget för att få feedback och synpunkter på vårt arbete.

För att kunna skapa sidor och mallar med Mallarmodulen diskuterade vi tillsammans med uppdragsgivaren fram att det måste finnas två olika användarroller. En roll som endast kan skapa, redigera och ta bort sidor, Sidanvändare, och en roll som används för att skapa nya mallar, redigera dem och hantera dem, malladministratör (MallAdmin).
På så vis kan t.ex. inte användare som tillhör rollen sidanvändare av misstag ta bort mallar eftersom det inte är deras primära roll.

Sidrättigheter, dvs. vem som får granska sidor, var i viss omfattning ett betydligt problem men dock överkomligt. Vi löste det så att när man skapar nya sidor med vår modul så ärver den nya sidan rättigheter från en rättighetssida. Rättighetssidan är en helt vanlig sida som skapas i INVID Publisher.NET. Detta är dock inget som Sidanvändaren behöver tänka på. Det är en inställning som görs endast en gång – nämligen när man injicerar (lägger till) Mallarmodulen på en sida.

Att skapa nya sidor utifrån mallar var det problem som förmodligen har varit det svåraste. Det handlar om att ha kunskap om hur DNNs-sidskapningsklasser fungerar och hur databasen som används av en DNN-portal är uppbyggd.
Vi konsulterade DNN-dokumentationen ett flertal gånger och även webbplatser som presenterar lösningar och svar på DNN-relaterade problem (däribland *http://forums.asp.net/90/ShowForum.aspx*) för att få svar på våra frågor.

3.4 Modulen Blog

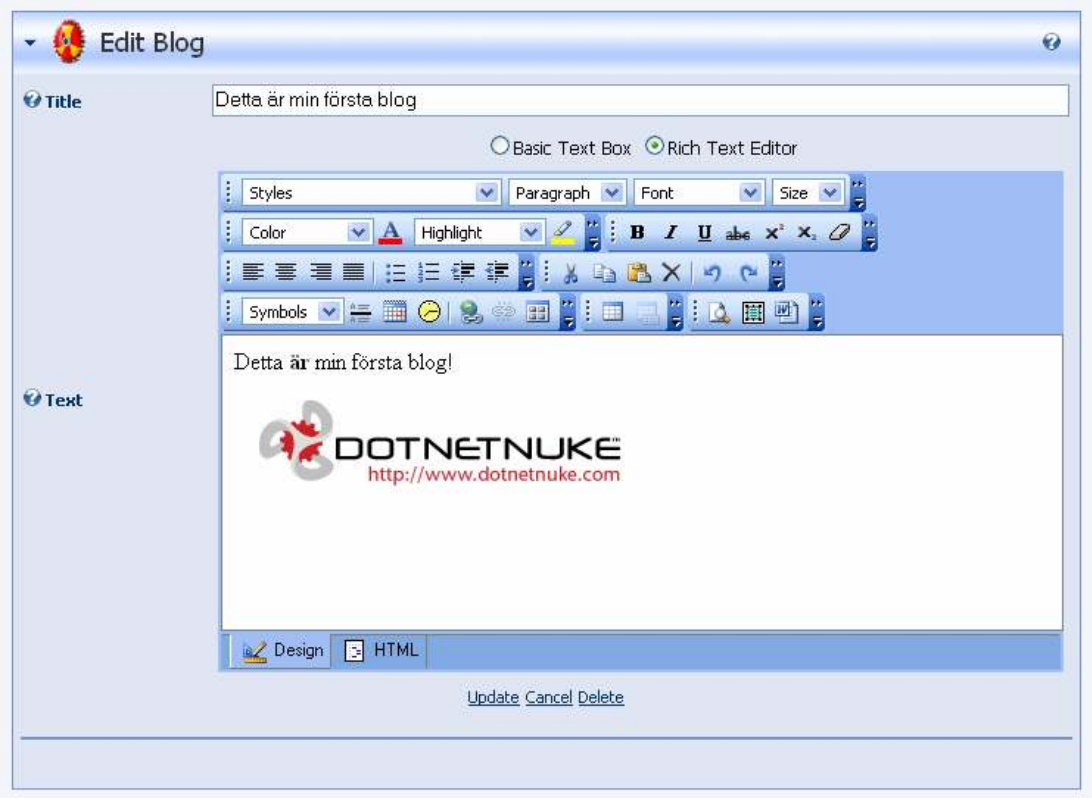
Modulen, fortsättningsvis refererad till som blog, se Figur 3-1, är uppbyggd enligt DNNs ramverk och använder därigenom de olika lager som finns specificerade för DNN-moduler.

Kortfattat sker lagringen av text och datum i ett `BlogInfo`-objekt. Data lagrad i ett `BlogInfo`-objekt visas dels för användaren på skärmen men används också för att en instans av `BlogController` (i Business Logic Layer) ska kunna skicka informationen vidare till DataProvider-klassen som står i kontakt med SQLDataProvider som i sin tur skriver data fysiskt till databasen.

Hämtning av information görs med ett `BlogController`-objekt som lagrar information i ett `BlogInfo`-objekt. `BlogController`-objektet gör en förfrågan till DataProvider-klassen som anropar SQLDataProvider där implementationen av providern fysiskt hämtar data ur databasen.
`BlogInfo`-objektet fylls med information som presentationslagret visar i webbläsaren. Figur 3-3 visar blogmodulen i sitt redigeringsläge, där kan nya bloginlägg skrivas. Figur 3-2 visar blogmodulen i inställningsläget.

Figur 3-1 Modulen blog i visningsläge

Figur 3-2 Modulen blog i inställningsläge



Figur 3-3 Modulen blog i redigeringsläge

3.5 Design av databastabell och stored procedures för blogmodul

Tabellen som ska lagra data åt blog skapas med hjälp av verktyget SQL Enterprise Manager och består av sju fält:

- BlogId. Modulens primärnyckel inkrementeras för varje inlägg.
- PortalId. Vilken portal modulen hör till.
- ModuleId. Modulens identifikationsnummer.
- Title. Ämne för blog.
- Text. Vilken text som skall presenteras och som hör ihop med resp. titel.
- DateAdd. Datum för bloginlägg.
- DateMod. Datum för ändring av bloginlägg.

Vidare skapade vi fem stored procedures som kan lägga till/ta bort/hämta och uppdatera bloginformation. De fem är följande:

- dbo.blog_BlogAdd. Lägger till nytt bloginlägg för en särskild modulinstans (ModuleId) i en specifik portal (PortalId).
- dbo.blog_BlogDelete. Tar bort ett specifikt bloginlägg (BlogId).
- dbo.blog_BlogGet. Hämtar specificerat bloginlägg (BlogId).
- dbo.blog_BlogList. Hämtar alla bloginlägg som tillhör en viss modulinstans (ModuleId) för en specificerad portal (PortalId).
- dbo.blog_BlogUpdate. Uppdaterar ett specifikt bloginlägg (BlogId) för en specificerad modulinstans (ModuleId) i en särskild portal (PortalId).

3.6 Implementation av Blogmodul

Modulen blog består av ett antal filer som utgör presentationslagret, affärslogiklagret och dataåtkomstlagret. Dessa filer har producerats enligt de principer som bl.a. har förklarats i teoriavsnittet (t.ex. `Controller` och `Info`).

Presentationslagret	Affärslogiklagret	Dataåtkomstlagret
• Blog.ascx • BlogEdit.ascx • Settings.ascx • Module.css • Blog.ascx.resx • BlogEdit.ascx.resx • Settings.ascx.resx • icon_Blog_32px.gif	• BlogController.vb • BlogInfo.vb	• DataProvider.vb • SqlDataProvider.vb

3.6.1 Blog.ascx

Blog.ascx är webbuser-kontrollen för visningsläget av bloginformation. Den visar de bloginlägg som har gjorts, se Figur 3-1.

Metoden `HtmlDecode`, som finns i filen Blog.ascx, (i Server objektet i Figur 3-4) avkodar en htmlkodad sträng eftersom själva blog-meddelandet lagras i databasen som kodade html-taggar(ex: ”&lt” måste kodas om till ”<”) så måste de kodas om till rena html-taggar så att de kan visas i blogmodulen på ett korrekt sätt.

```
Public Function HtmlDecode(ByVal strValue As String) As String
    Try
        Return Server.HtmlDecode(strValue)
    Catch exc As Exception 'Module failed to load
        ProcessModuleLoadException(Me, exc)
    End Try
End Function
```

Figur 3-4 Kodexempel för avkodning av html-kodad sträng

3.6.2 BlogEdit.ascx

WYSIWYG-texteditorn (se Figur 3-3) i modulens redigeringsläge används för att skapa bloginlägg. Den är förprogrammerad och medföljer DNN och kan enkelt placeras på webbuserkontrollen BlogEdit.ascx genom att referera till kontrollen i HTML-koden.
För att posta ett bloginlägg skickas datum och andra data in i objektet `objBlogInfo` (t.ex. `objBlogInfo.Title = txtTitle.Text`), som är deklarerad som typen `BlogInfo`, se Figur 3-5. När alla egenskaper för objektet är satta skickas `objBlogInfo` vidare till `objBlogController` som tar hand om objektet och vidarebefordrar objektets data till dataåtkomstlagret.

```
objBlogInfo.BlogId = blogId
objBlogInfo.PortalId = PortalId
objBlogInfo.ModuleId = ModuleId
objBlogInfo.Title = txtTitle.Text
objBlogInfo.Text = teText.Text
objBlogInfo.DateMod = Convert.ToDateTime(Now.ToString)

If Null.IsNull(blogId) Then
     objBlogInfo.DateAdd = Convert.ToDateTime(Now.ToString)

     objBlogController.Add(objBlogInfo)
Else
     objBlogController.Update(objBlogInfo)
End If
```

Figur 3-5 Kodexempel för att lägga till/uppdatera ett Bloginlägg

3.6.3 Settings.ascx

I modulens inställningsvy kan man välja om datum skall visas eller inte.
Figur 3-6 visar inladdning av inställning från databasen. Värdet som returneras från `CType(TabModuleSettings("BlogShowDate"), String)` är antingen `Nothing`, `True` eller `False` beroende på det värde som finns i databasen.
Om `True` (vilket innebär att man sedan tidigare har kryssat kryssrutan) returneras kommer kryssrutan `chkShowDate` att kryssas i.

```
Dim ShowDate As String = CType(TabModuleSettings("BlogShowDate"), String)
If Not ShowDate Is Nothing Then
     chkShowDate.Checked = Convert.ToBoolean(ShowDate)
End If
```

Figur 3-6 Kodexempel för inladdning av inställning

3.6.4 Module.css och icon_Blog_32px.gif

Det är naturligtvis vanligt (och av betydelse) att man i ett ASP.NET-projekt strukturerar och skriver ned de teckensnitt, storlekar, färger m.m. som ska användas för tex. rubriker och textmassor. I filen Module.css kan modulspecifika formateringsinställningar lagras. Blog använder sig endast av systemspecifika stilmallar som är fördefinierade av DNN.

Bilden icon_Blog_32px.gif används endast för att visa en ikon bredvid titeln i modulens redigeringsläge, se Figur 3-3.

3.6.5 Blog.ascx.resx, BlogEdit.ascx.resx och Settings.ascx.resx

Dessa tre filer används för språköversättning av det grafiska gränssnittet och är avsedda för respektive Blog.ascx, BlogEdit.ascx och Settings.ascx. De statiska tecken som ska visas i modulens redigeringsläge (BlogEdit.ascx) lagras alltså i filen BlogEdit.ascx.resx och i modulens visningsläge lagras tecken i filen Blog.ascx.resx och så vidare.

3.6.6 BlogController.vb

Denna klass har metoder för att hämta, ta bort, lägga till och uppdatera bloginformation. Vi har skrivit följande fem metoder för det:

- `Get`
- `List`
- `Add`
- `Update`
- `Delete`

Dessa metoder fungerar i princip på samma sätt. De anropar ett objekt av typen DataProvider – som står i kontakt med implementationen av de abstrakta metoderna, dessa principer har förklarats i teoriavsnittet.

3.6.7 BlogInfo.vb

Denna klass har programmerats med egenskaper som matchar fältnamn och typ i datakällan, se avsnitt ”Design av databastabell och stored procedures för blogmodul”. För varje egenskap som finns i `BlogInfo`-klassen används interna variabler som håller värden.

3.6.8 DataProvider.vb

Metoder som residerar i denna klass är abstrakta dvs. implementationen av dem finns inte skriven här utan de är bara definierade. Fem metoder har definierats vilka är `GetBlog`, `ListBlog`, `AddBlog`, `UpdateBlog` och `DeleteBlog`.
Som man även märker ”mappar” dessa till metoder i `BlogController` klassen.

```
Public MustOverride Function ListBlog( _
    ByVal portalId As Integer, ByVal moduleId As Integer) As IDataReader
```

Figur 3-7 Exempel på en av de abstrakta metoder som residerar i DataProvider- klassen

Figur 3-7 är ett exempel på en av de abstrakta metoder som används av blogmodulen. Metoden `ListBlog` hämtar en lista med bloginlägg för en specifik instans (moduleId) av blogmodulen i en särskild DNNportal (portalId). För en utförlig förklaring av abstrakta metoder och deras karakteristik se avsnittet ”Data Access Layer”.

3.6.9 SqlDataProvider.vb

SqlDataProvider har själva implementationen av de abstrakta metoderna såsom de har definierats i klassen `DataProvider`. Eftersom `SqlDataProvider` ärver från `Dataprovider` måste den erbjuda en komplett implementation och har därmed samma metodnamn, signatur (vilka argument som krävs) och returvärden som `DataProvider`, se avsnittet ”DataProvider.vb” för metodnamn. Den programmeringslogik som finns i varje dataåtkomstmetod i klassen `SqlDataProvider` förlitar sig på metoder i Microsofts Data Access Application Block.

```
Public Overrides Function ListBlog( _
    ByVal portalId As Integer, ByVal moduleId As Integer) As IDataReader

    Return CType(SqlHelper.ExecuteReader( _
        ConnectionString, DatabaseOwner & ObjectQualifier & "blog_BlogList", _
        portalId, moduleId), IDataReader)

End Function
```

Figur 3-8 Exempel på en implementation av en abstrakt metod

Figur 3-8 beskriver den metod (`ListBlog`) som används av blogmodulen för att hämta en lista med bloginlägg för en specifik instans (moduleId) av blogmodulen i en specifik DNN-portal (portalId). Koden mellan metodhuvud och metodens slut är ett exempel på användning av Microsofts Data Access Application Block.

Klassen `SqlHelper` anropar en metod, `ExecuteReader`, som tar emot ett antal argument. Det första argumentet specificerar en `ConnectionString` som bestämmer bl.a. vilken databas och server som anslutningen skall gälla för. Argument nummer två specificerar det prefix som används för databasen, `DatabaseOwner`, och prefix för stored procedures, `ObjectQualifier`.
Det tredje argumentet bestämmer vilken stored procedure som skall exekveras, i detta fallet ”blog_BlogList”. De två nästkommande argumenten är `portalId` (portalinstans) och `moduleId` (modulinstans). Slutligen returneras resultatet av exekveringen av metoden `ExecuteReader` (och därigenom ”blog_BlogList”) och konverteras till en `IDataReader` med `CType()`.

4 Resultat

Examensarbetet har utmynnat i en välfungerande mallmodul.
Utgångspunkten för vår modul bygger på analyser av tidigare existerande mallmodul som företaget (INVID AB) inte var nöjda med.

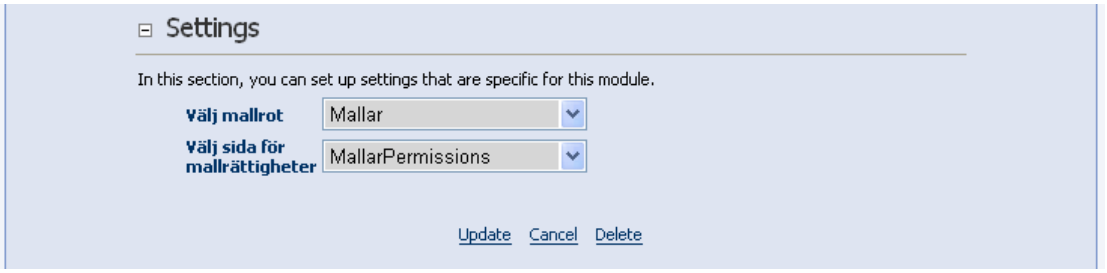
Uppdragsgivaren anser att modulen som utvecklats är grafiskt tilltalande och vi anser att den håller en hög programmeringsteknisk nivå. Mallmodulen är konstruerad enligt den treskiktade lagerprincip som DNN bygger på.
Modulen går att installera i vilken DNN version 3 baserad portal som helst.
Mallarmodulen är konstruerad med avsikt att utnyttja två olika användarroller, Sidanvändaren och MallAdmin. Sidanvändaren har möjlighet att använda mallar som MallAdmin skapat med hjälp av Mallarmodulen. Med mallar (mallsidor) som förlaga kan nya sidor skapas. Sidanvändaren har dock ingen möjlighet att lägga till mallar i systemet.
Malladministratören har större möjligheter, bl.a. kan han skapa mallar åt Sidanvändaren och har större möjligheter att påverka inställningar än Sidanvändaren. Det krävs att MallAdmin har större vana vid DotNetNuke i allmänhet och INVID Publisher .NET i synnerhet för att kunna administrera.

Mallar sorteras under kategorier. Både mallar och kategorier är vanliga s.k. DNN-sidor. Vi har inte bestämt någon specifik gräns utan det går att skapa hur många kategorier och mallar som helst, det är DNN-portalen och databasen som begränsar.

Att skapa nya sidor med mallmodulen är enkelt. En kund behöver bara ange ett antal uppgifter (t.ex. titel etc.) för att skapa en ny sida baserat på en mall.
När sidan har skapats kan kunden lägga till nya moduler genom att utnyttja INVID Publisher.NETs kontrollpanel som finns längst upp på varje nyskapad sida.

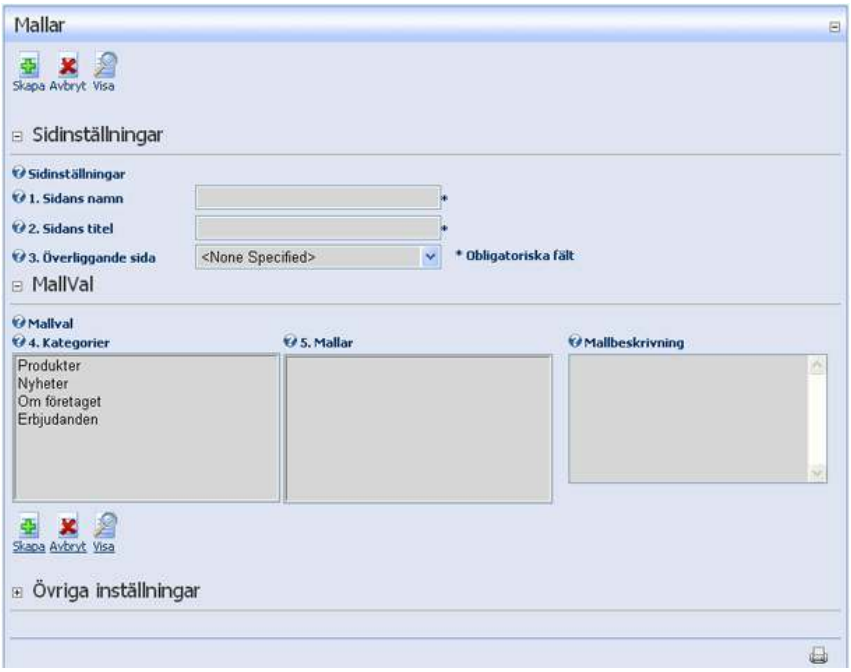
Mallar (mallsidor) som skapas med mallmodulen är vanliga DNN-sidor som nämnts tidigare. Varje mall kan ha ett antal olika moduler t.ex. text/html, länkar, annonser etc. Varje modul (t.ex. text/html, annonser etc.) som är placerad på en specifik mall hanterar och presenterar olika sorters information (t.ex. text, bild etc.). När en ny sida skall skapas utifrån en mall kopieras de moduler som finns på mallen till den nya sidan. Innehållet i varje modul kopieras dock inte till den nya sidan, med undantag för text/html-modulen, bl.a. pga. man inte har någon direkt nytta av det och dels att någon ny och generell funktion för kopiering inte enkelt kan skapas.

Kunder har alltid möjlighet att lägga till text/html-moduler på sina sidor då denna modul ingår i deras standard-moduluppsättnig. Nyttan med kopiering av innehåll för en text/html-modul är dock störst jämfört med kopiering av innehåll för någon annan modul som ingår i kundens standard-moduluppsättning.



Figur 4-1 Inställningsvy

Det första som måste göras när man lägger till vår modul till en sida är att ange var roten för alla mallar finns dvs. ”föräldern” till mallarna. Den andra inställningen som krävs är att ange vilken sida som ska utnyttjas som rättighetssida. Dessa två inställningar görs från inställningsvyn, se Figur 4-1. Text som ses ovanför ”Välj mallrot” i Figur 4-1 är på Engelska pga. vi valde den språkinställningen när vi utvecklade modulen. Denna text är dock inte knuten till vår modul eller till något specifikt språk (språkoberoende) utan läggs till automatiskt av DNN.



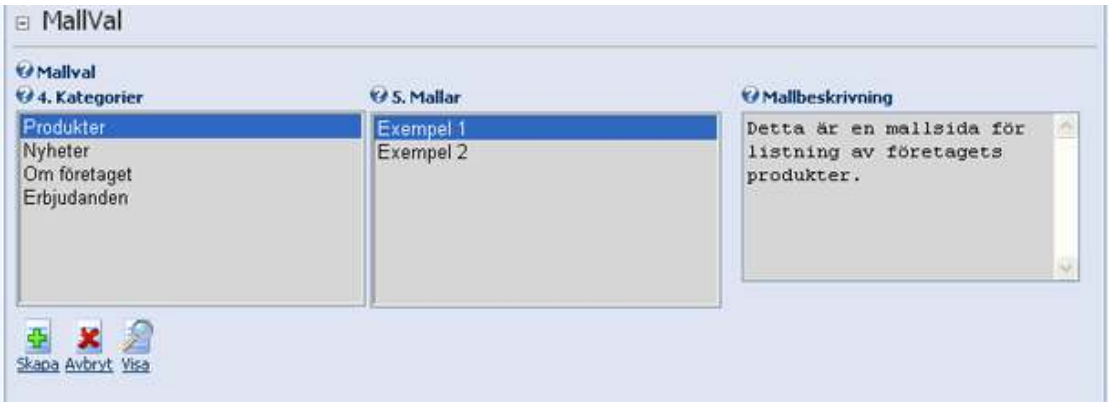
Figur 4-2 Mallhanteringsmodulen i visningsläge

Figur 4-2 visar vår mallhanteringsmodul installerad i vår test-och-utvecklingsportal med standardskin valt. Eftersom det är viktigt att dela upp det grafiska gränssnittet i logiska delar har vi använt oss av minimerings/maximerings-kontroller (plus/minus-tecken bredvid rubriktext). Därmed kan användaren bestämma själv vad denne önskar ägna uppmärksamhet åt och dessutom tar det mindre plats.
Vad man även kan se i Figur 4-2 är att själva sidskapningsfunktionaliteten är uppdelad i tre distinkta sektioner: sidinställningar, mallval och övriga inställningar.

Figur 4-3 Sektion sidinställningar

Att skapa en sida kräver några obligatoriska steg som man måste följa i vår modul. Det är att fördra att användaren utför vissa steg i en viss ordning varför dessa är utmärkta med siffror som man upptäcker i Figur 4-2.
Vår hypotes är att den ovane användaren skall lättare kunna skapa sidor.
Vid minsta osäkerhet kan man klicka på frågetecknet bredvid uppgift som ska anges, då vi förtydligar mera vad som avses, se Figur 4-3.

För att skapa en sida måste man först ange sidans namn – vilket är namnet som den kommer att få i menyn. Sedan anger man sidans titel vilket är titeln i användarens webbläsare. Om man anger en överliggande sida kommer den nya sidan att placeras hierarkiskt under en annan sida i huvudmenyn i DNN.
Både sidans namn och titel är obligatoriska fält som måste anges.
De valideras även vid ett försök att skapa en sida utan att fylla i dem korrekt.



Figur 4-4 Sektion mallval

Nästa sektion att rikta uppmärksamheten mot är mallval, se Figur 4-4.
Här skall man först välja vilken mallkategori och sedan mall. Vid klick på mall visas också beskrivning av mallen vilket är tänkt att underlätta syftet med mallen som sådant.

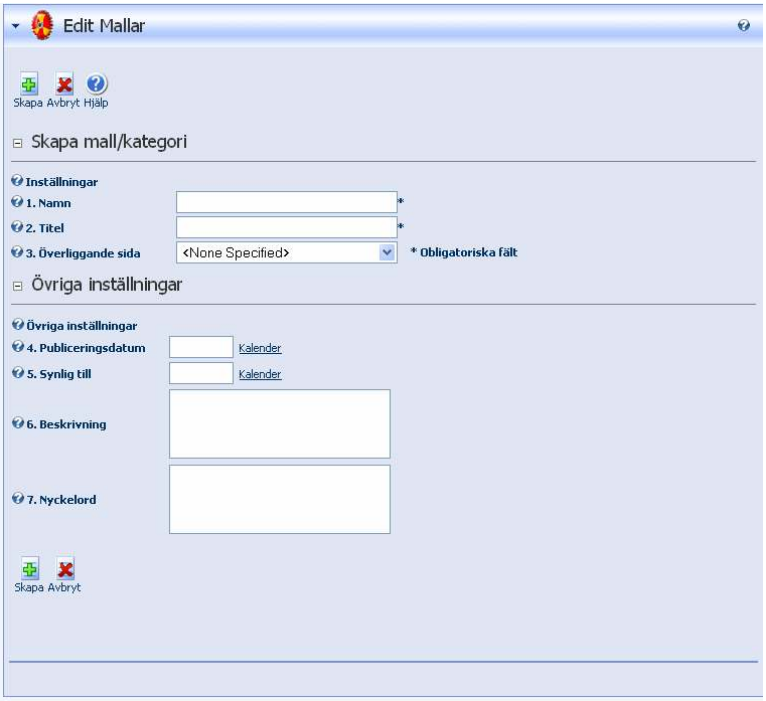
Figur 4-5 Förhandsgranskning av vald mall

För att förhandsgranska vald mall (mallsida) kan man trycka på ”visa”-länken eller förstoringsglaset, se Figur 4-4. Då öppnas ett popup-fönster som visar vald mall, vilket man kan se i Figur 4-5.

Figur 4-6 Sektion övriga inställningar

Om användaren önskar ange ytterligare information som t.ex. datum för publicering av sidan, sidbeskrivning etc. är det lämpligt att rikta uppmärksamheten mot sektionen övriga inställningar, se Figur 4-6.
Här kan man ange när en sida skall publiceras dvs. vilket datum den skall vara synlig från och när den inte skall visas längre. Nyckelord kan även anges som bl.a. underlättar för sökmotorer (t.ex. Google) vid indexering av sidan. Sidbeskrivning kan även anges som är tänkt som en kort och koncis beskrivning av syftet med sidan.

När användaren är nöjd med sitt val av mall, sidnamn etc. kan denne trycka på ”skapa”-länken eller skapa-ikonen, se Figur 4-4, då en sida skapas enligt de uppgifter som har angetts och innehållet bestäms utifrån den mall som valts.



Figur 4-7 Mallmodul i redigeringsläge

Mallar och kategorier kan skapas från mallmodulens redigeringsläge, se Figur 4-7. Först skapar malladministratören kategorisidor som fungerar som behållare för mallsidorna. Dessa skall placeras under roten för mallarna (förälder till kategorier) vilket väljs med komboboxen ”överliggande sida”.
När mallsidor sedan ska skapas placeras dessa som barnsida till den kategori som de skall tillhöra (vilket väljs med samma kombobox som nämnts ovan). Inställningar såsom namn, titel för mallar och kategorier görs under sektionen ”Skapa mall/kategori”. Om publiceringsdatum, beskrivning och nyckelord önskas anges för mall eller kategori kan detta göras under sektionen ”Övriga inställningar”.

5 Slutsats och diskussion

DNN är byggt på en trelagersprincip och inkorporerar även Microsofts providermodell och mediatormönster. Moduler kan konstrueras enligt DNNs ramverk. Båda modulerna som visas i denna rapport följer providermodellen såsom den har beskrivits i teorikapitlet.

När vi fick förslaget att bygga moduler för det DNN-baserade webbpubliceringsverktyget INVID Publisher.NET hade vi ingen kunskap om DNN och hur det var uppbyggt.

Vi har under arbetets gång fått forska mycket och djupt i DNN-ramverket för att få förståelse för hur applikationens lager är sammankopplade.
Kunskapen om de metoder som används i ramverket har varit helt nödvändiga för vår fortsatta utveckling av mallhanteringsmodulen.
Kunskaper inhämtades från programmeringslitteratur, medföljande DNN-dokumentation och artiklar på webben. Vi anser att det krävs en god insikt i bl.a. webbprogrammering och naturligtvis DNN för att överhuvudtaget kunna utveckla moduler för DNN.

Vi anser att modulen Mallar blev välkonstruerad och relativt grafiskt tilltalande. Det finns alltid saker som kan finputsas och göras bättre.
En funktion med förbättringspotential är att i nuvarande version kopieras innehåll (information) i mallar (mallsidor) endast för modulen text/html (när den finns på en mallsida). I framtiden skulle funktionalitet för innehållskopiering med andra moduler kunna utvecklas.

Det finns mycket litteratur att tillgå. Problemet är att mycket av den litteratur som skrivits ligger på en hög nivå och koncept och tekniker som presenteras kan i början vara svåra att förstå.

I framtiden kan modulen vidareutvecklas framgångsfullt bl.a. tack vare att vi i vårt utvecklingsarbete har följt DNNs ramverk och principer strikt och även anammat goda programmeringsprinciper.

6 Referenser

[1] *What is DotNetNuke?*
http://www.dotnetnuke.com/About/WhatIsDotNetNuke/Introduction/tabid/781/Default.aspx
(Acc. 2006-01-25)

[2] Shaun, Walker; Patrick J. Santry et al (2005) *Professional DotNetNuke ASP.NET Portals*
Wiley Publishing, Indianapolis, ISBN 0-7645-9563-6.

[3] *What is the history of dotnetnuke.*
http://www.dotnetnuke.com/About/WhatIsDotNetNuke/Background/tabid/779/Default.aspx
(Acc. 2006-01-25)

[4] *Provider Model Design Pattern and Specification, Part 1* (2004)
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnaspnet/html/asp02182004.asp
(Acc. 2006-03-11)

[5] Evjen, Bill; Beres, Jason et. al (2001) *Visual Basic .NET Bible*
Wiley Publishing, Indianapolis, ISBN 0-7645-4826-3.

[6] *Compiling to MSIL*(2002)
http://msdn.microsoft.com/library/default.asp?url=/library/en- us/cpguide/html/cpconMicrosoftIntermediateLanguageMSIL.asp
(Acc. 2006-02-17)

[7] *ASP .NET: Web Forms Let You Drag and Drop Your Way to Powerful Web Apps*.(2001)
http://msdn.microsoft.com/msdnmag/issues/01/05/webforms/default.aspx
(Acc. 06-02-17)

[8] *.NET Framework Class Library*(2002)
http://msdn2.microsoft.com/en-us/library/system.data.idatareader.aspx
(Acc. 06-02-20)

[9] DotNetNuke Data Access.pdf
http://www.dotnetnuke.com

[10] DotNetNuke Localization.pdf
http://www.dotnetnuke.com

[11] *Mediator Design Pattern in C# and VB.NET* (?)
http://www.dofactory.com/Patterns/PatternMediator.aspx#UML
(Acc. 06-02-27)

[12] *Design pattern* (2006)
http://en.wikipedia.org/wiki/Design_pattern_(computer_science)
(Acc. 06-02-27)

[13] DotNetNuke SearchEngine.pdf
http://www.dotnetnuke.com

[14] Introduction to Web Forms Pages (2002)
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vbconintroductiontowebforms.asp
(Acc. 06-02-27)

[15] DotNetNuke Client API DragDrop.pdf
http://www.dotnetnuke.com

[16] DotNetNuke Client API Client Callback.pdf
http://www.dotnetnuke.com

7 Sökord

8 Bilagor

Bilaga 1 Installationsanvisning för DNN 3.2.2

Bilaga 1

Installation av DNN version 3.2.2

Att installera DotNetNuke kräver ett flertal nödvändiga steg. DotNetNuke har inga systemkrav i sig men kräver andra programvaror och tjänster. Det första som behöver undersökas är huruvida systemkrav för nödvändiga program och tjänster uppfylls. Enligt figuren nedan visas minimala krav för installation.

	.NET 1.1	*SQL Server 2000*	*Windows XP pro*	*Internet Information Services*
RAM	128MB*	64MB*	64MB	64MB
CPU	Pentium 133MHz*	Pentium 166MHz*	Pentium 233MHz	Pentium 233MHz
Hårddisk	160MB	95-270MB	1,5GB	-
OS version	Win2000 Sp3, Win2000 Server Sp3, Win XP Pro, Windows 2003 Familjen	Windows XP Pro, 2000 Server, 2003 Server	-	Win XP Pro
Andra krav	800x600x8 upplösning eller högre	-	800x600x8 upplösning, CDRom/DVD	-
Mjukvarukrav	IIS 5.0 eller senare	IIS 5.0 eller senare, ASP.NET 1.1	-	-

Det bör påpekas att man inte behöver köra IIS och SQL på samma dator utan kan använda dedicerade webb och databas servrar. Att använda separata datorer kan ge bättre prestanda genom fördelning av arbetslasten.

Det första som behöver göras för att installera en version av DotNetNuke från grunden är att registrera sig och ladda ned källkoden. Följ nedanstående steg för installation.

- Källkoden packas lämpligtvis upp i C:\DotNetNuke.
- Programmet kräver att användaren ASPNET har åtminstonde läsa och köra, skriva och ändra -rättigheter på mappen för att kunna exekveras. Högerklicka på katalogen ”DotNetNuke” under C:\ och sätt dessa attribut.
- Gå till kontrollpanelen och öppna Datorhantering. Under tjänster och program finns en instans av SQL server 2000. Markera den och högerklicka och välj ”Ny databas”. I dialogrutan som visas ska namnet på databasen anges. Skriv ”DotNetNuke”. Låt alla andra inställningar vara satta till standard och tryck ”OK”.
- Välj ”Internet Information Services” i datorhantering. För att kunna besöka webbsidan skall vi sedan skapa en virtuell katalog med detta verktyg. Högerklicka på ”Standardwebbplats” och välj ”Nytt” och sedan ”Virtuell katalog”. Aliaset kallas lämpligtvis för DotNetNuke.

- I katalogen ”C:\DotNetNuke” finns en fil som heter release.config. Denna fil innehåller applikationsspecifika inställningar.bl.a. finns här databaskonfiguration. Redigera filen med en texteditor. Ändra databas och inloggningsdata så att de möter Sql server konfigurationen. Ändra slutligen namnet på filen till ”web.config”.
- Navigera till url:en http://localhost/DotNetNuke. Det som inträffar sedan är att tabeller och stored procedures skapas.