|  |
| --- |
| Laborationsanvisning |

Äventyrliga kontakter

Steg 2, laborationsuppgift 1

|  |
| --- |
| *Författare:* Mats Loock  *Kurs:* ASP.NET Web Forms  *Kurskod:*1DV406 |

**Upphovsrätt för detta verk**

Detta verk är framtaget i anslutning till kursen ASP.NET Web Forms (1DV406) vid Linnéuniversitetet.

**Du får använda detta verk så här:**

Allt innehåll i detta verk av Mats Loock, förutom Linnéuniversitetets logotyp, symbol och kopparstick, är licensierad under:

|  |  |
| --- | --- |
|  | Creative Commons Erkännande-IckeKommersiell-DelaLika 2.5 Sverige licens. <http://creativecommons.org/licenses/by-nc-sa/2.5/se/> |

**Det betyder att du i icke-kommersiella syften får:**

* kopiera hela eller delar av innehållet
* sprida hela eller delar av innehållet
* visa hela eller delar av innehållet offentligt och digitalt
* konvertera innehållet till annat format
* du får även göra om innehållet

Om du förändrar innehållet så ta inte med Linnéuniversitetets logotyp, symbol och/eller kopparstick i din nya version!

**Innehåll**

[Uppgift 5](#_Toc376863947)

[Inledning 5](#_Toc376863948)

[Presentationslogiklagret 5](#_Toc376863949)

[Textfälten och validering 6](#_Toc376863950)

[Alternativknapparna 6](#_Toc376863951)

[Kommandoknappen 6](#_Toc376863952)

[Temperaturtabellen 7](#_Toc376863953)

[Stilmallar 7](#_Toc376863954)

[Affärslogiklagret 8](#_Toc376863955)

[Hantering av fel på servern 8](#_Toc376863956)

[Mål 8](#_Toc376863957)

[Tips 9](#_Toc376863958)

# 

# Uppgift

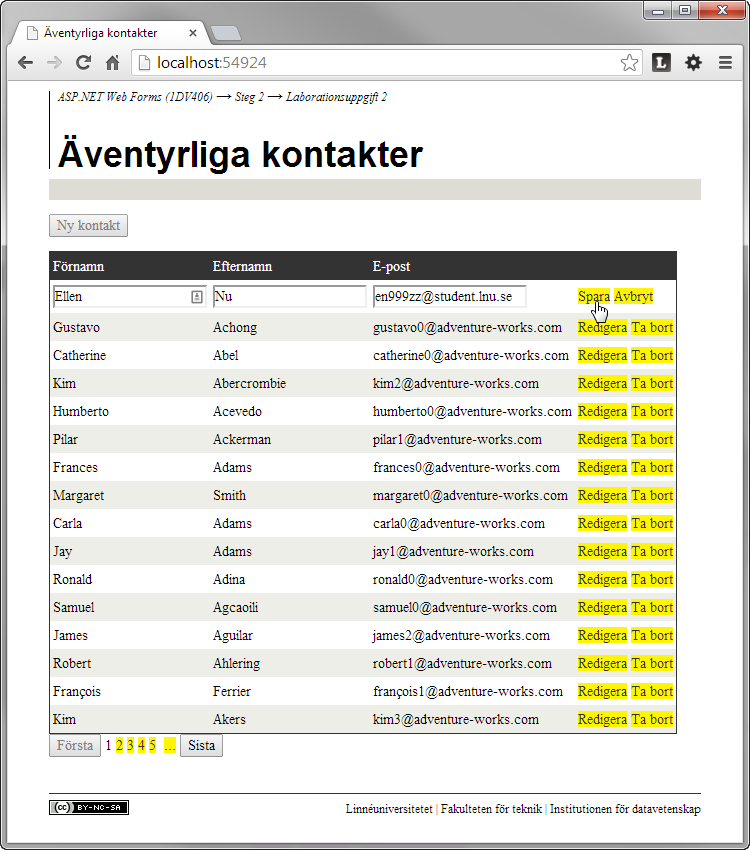
## Inledning

## Inledning

Målet med laborationsuppgiften är att du ska skapa en webbapplikation, uppdelade i flera lager, där användaren kan ta del av och redigera innehållet i tabellen Person.Contact i databasen 1dv406\_AdventureWorksAssignment.

Du kommer under laborationsuppgiften använda ADO.NET för att exekvera lagrade procedurer i databasen. Du kommer även att fördjupa dig mer i databundna kontroller, men framför allt få en insikt om vad en flerlagerapplikation är.

Du har stor frihet att utforma webbapplikationen på det sätt du önskar. Figur 1 är ett exempel på hur användargränssnittet kan utformas, men du är fri att välja ett helt annat upplägg – under förutsättning att kraven uppfylls.



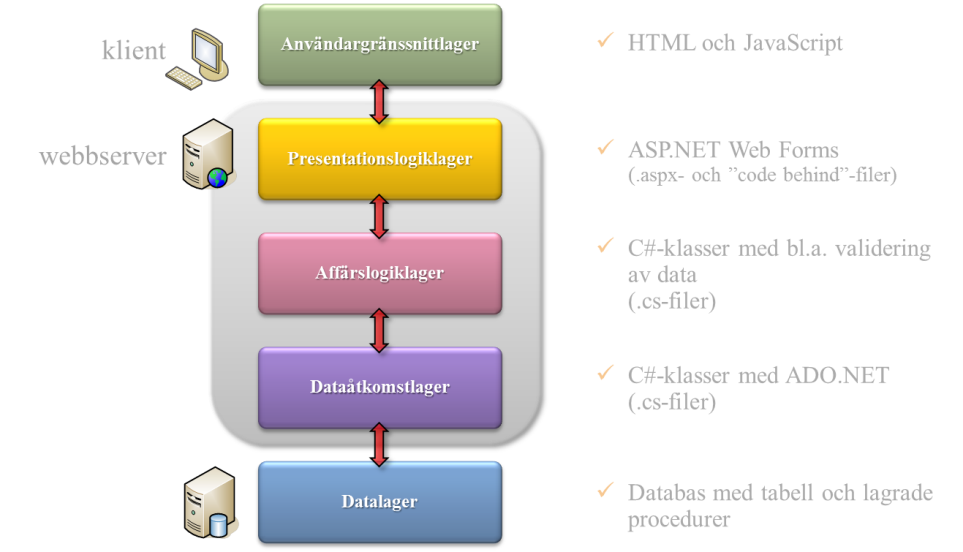
Figur 1. Webbsida då användaren står i begrepp att lägga till en ny kontakt.

## Flerlagerapplikation

Det mest centrala kravet är att webbapplikationen ska vara utformad som en femlagerapplikation. Figur 2 visar de fem lager applikationen ska delas upp i användargränssnittlager, presentationslogiklager, affärslogiklager, dataåtkomstlager samt datalager.

I laborationsuppgiften ska fyra av fem lager implenteras. Datalagret, som utgörs av en databas med en tabell samt ett antal lagrade procedurer, är redan färdigt att användas. Övriga lager måste impelementeras.

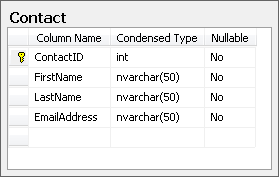
Användargränssnittlagret består av HTML och JavaScript som exekveras på klienten. Presentationslogiklagret utgörs av ett (eller flera) webbformulär (aspx-sida med tillhörande”code-behind”-fil). Affärslogiklagret är ”enkla” C#-klasser, t.ex. innehållande kod för validering av datat affärslogikobjekt innehåller. Dataåtkomstlagret är de klasser som använder ADO.NET för hantering av persistent data.



Figur 2. Applikationens fem lager.

Ett lager får inte hoppas över, t.ex. får inte presentationslogiklagret inte innehålla kod som direkt kommunicerar med datalagret.

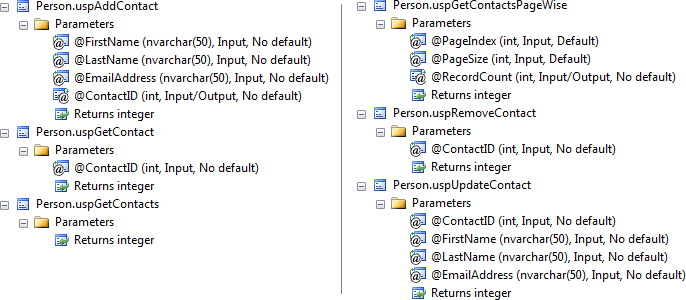
Webbapplikationen ska arbeta med tabellen Contact (starkt förenklad då flertalet fält negligeras) i databasen 1dv406\_AdventureWorksAssignment, som finns på laborationsservern FALKEN (172.16.214.1).



Figur 3. Den förenklade tabellen Contact med kolumnnamn och typer.

Du har inte direkt åtkomst till tabellen utan ditt gränssnitt mot tabellen utgörs av ett antal lagrade procedurer som användaren appUser, med lösenordet 1Br@Lösen=rd?, har rättighet att exekvera.

Genom att på lämpligt sätt använda de lagrade procedurer ska användaren av applikationen kunna visa kontakterna, skapa nya kontakter, ändra befintliga kontakter och ta bort kontakter. Det är bara poster du själv, eller någon annan kursdeltagare, lagt till som du kan på något sätt ändra på. Försöker du ta en annan post kommer ett fel att inträffa, varför felhantering är en viktig del (som alltid) att ta hänsyn till. Användaren ska bli informerad om då ett fel inträffat likväl som då en operation lyckats.

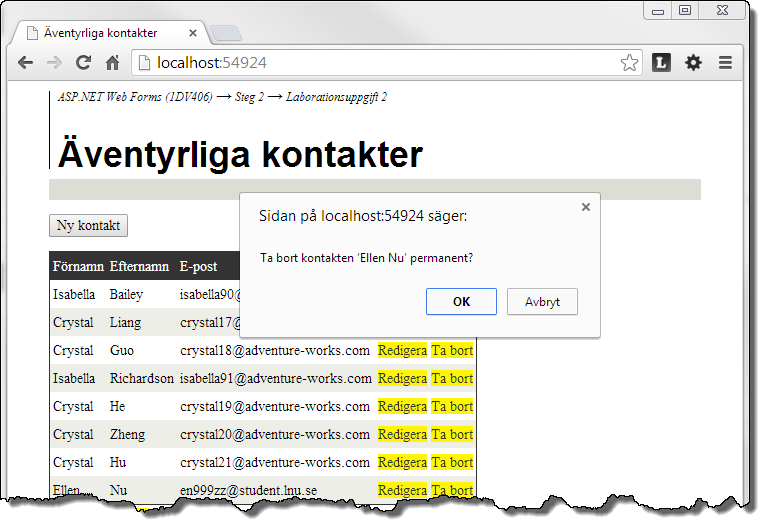


Figur 4. Lagrade procedurer med namn och parametrar.

## Användargränssnittlagret

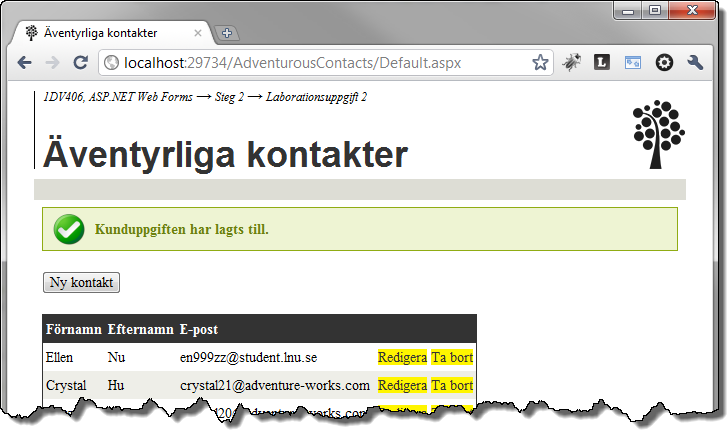
Viss del av funktionaliteten måste eventuellt implementeras med JavaScript. Du väljer själv om du vill använda dig av ett JavaScript-bibliotek, som till exempel jQuery, eller inte.

Innan en kontakt tas bort måste användaren bekräfta det. Utformningen av bekräftelsen är du fri att utforma. Figur 5 visar ett exempel där JavaScript används för att visa en bekräftelsedialogruta.



Figur 5. Bekräftelse krävs innan en kontaktuppgift tas bort.

Användaren ska på lämpligt sätt informeras om hanteringen av en kontakt lyckats. Väljer du att göra detta med hjälp av ett eller flera HTML-element ska meddelandet kunna tas bort av användaren. Rättmeddelanden kan alternativt vara placerat i en helt separat sida.



Figur 6. Rättmeddelande, innan det försvinner automatiskt, då användaren lyckats lägga till en ny kontaktuppgift.

Försöker användaren spara kontaktuppgifter som inte uppfyller uppställda valideringsregler, eller om annan typ av fel inträffar, ska ett felmeddelande visas.

## Presentationslogiklagret

Presentationslogiklagret utgörs av de aspx- och ”code behind”-filer du tycker dig behöva för att lösa uppgiften. I detta lager får du bara använda dig av typen/typerna i affärslogiklagret.

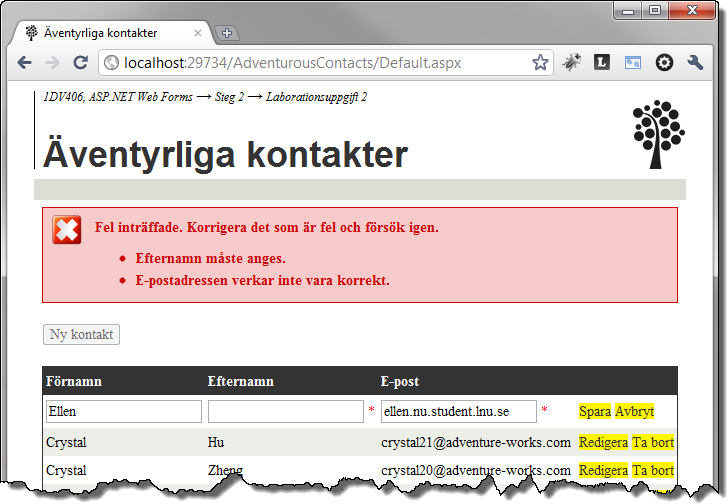
Tänk på att använda valideringskontroller så du kan erbjuda användaren en snabb återkoppling på om något felaktigt matats in i formulärs textfält. *OBS! Valideringskontrollerna kan på inget sätt ersätta valideringen av datat som måste ske i affärslogiklagret.*

### Validering

Då en kontaktuppgift skapas eller redigeras ska följande villkor vara uppfyllda:

* Det måste finna ett förnamn, som mest får bestå av 50 tecken.
* Det måste finna ett efternamn, som mest får bestå av 50 tecken.
* Det måste finna en e-postadress, som mest får bestå av 50 tecken.
* E-postadressen måste vara korrekt formaterad.

Med hjälp av kontrollen RequiredFieldValidator kan du säkerställa att ett textfält innehåller något. Kontrollen RegularExpressionValidator är lämplig att använda då du vill undersöka om ett textfälts innehåll kan tolkas som en e-postadress. Antalet tecken ett textfält kan innehålla begränsar du enklast med hjälp av egenskapen MaxLength som renderas som attributet maxlength varför det inte finns något behov av en valideringskontroll som hanterar detta.

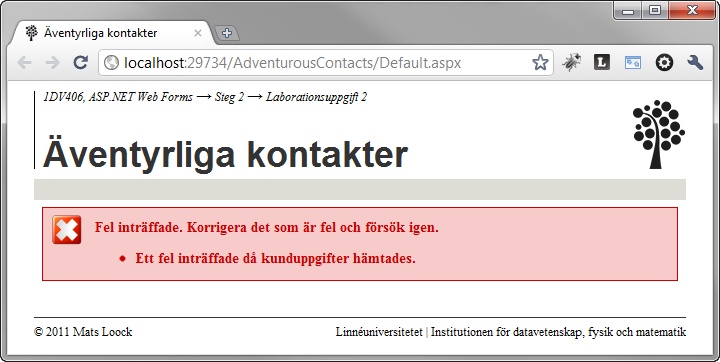


Figur 7. Felmeddelande då användaren försöker lägga till en ny kontakt med ofullständiga uppgifter.

### Hantering av undantag

I största möjligaste mån ska kastade undantag inte leda till att en allmän felsida presenteras för användaren. Ta därför på lämpligt sätt hand om undantag så relevanta felmeddelande kan presenteras vid behov.

Misslyckas hanteringen av någon orsak kan det vara lämpligt att använda sig av samma ValidationSummery-kontroll som används av andra valideringskontroller för att visa ett felmeddelande.



Figur 8. Felmeddelande då dataåtkomstlagret kastat ett undantag.

Figur 8 visar ett anpassat felmeddelande som orsakats av att dataåtkomstlagret inte kunde skapa en anslutning till databasen.

### Resultat av en lyckad hantering av kontaktuppgift

Då en kontaktuppgift lyckats läggas till, uppdaterats eller tagits bort ska ett rättmeddelande visas.

## Affärslogiklagret

Affärslogikklassen ansvara för all hantering av kontaktuppgifterna. I affärslogiklagret placerar du den klass som krävs för datat för en post i tabellen Contact.

Fälten i affärslogiklassen måste vara privata; undantagsfall kan de vara ”protected”.

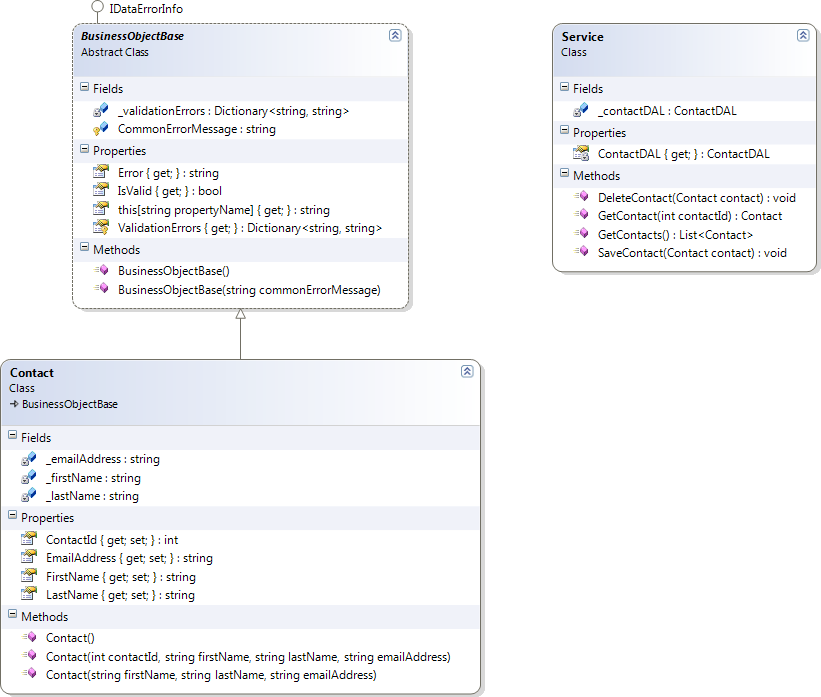
Egenskaperna knutna till fälten (förutom medlemmen för primärnyckeln) måste kunna valideras. Bryter något av fältens data valideringsvillkoren betraktas objektet i in helhet att inte klara valideringen. Förnamn, efternamn och e-postadress får inte ha värdet null eller vara en tom sträng. Strängen får inte heller överskrida antalet tecken som är möjligt att lagra i databasen.

Förutom nämnd klass inklusive medlemmar behövs ett antal metoder för att kunna hämta, spara och radera kontaktuppgifter. Det kan vara lämpligt att placera kod som har med detta att göra i en separat klass. Tänk på att det är lämpligt att skicka fullständiga och validerade objekt mellan lagren, men det är inte alltid som det krävs.

### Förslag på klasser

I figur 9 nedan ser du ett klassdiagram med förslag på hur du kan utforma klasserna i affärslogiklagret som ska hantera kontaktuppgifterna. Du behöver inte utforma dina klasser på samma sätt utan kan skapa egna om du så vill. Lämpliga medlemmar för klassen som representerar tabellen Contact ges av figur 3.

Genom att omsorgsfullt utforma klassen kommer det att bli enklare att för dig att skriva koden som tillhör användargränssnitt- och presentationslogiklagren. Tänk på att klasserna i möjligaste mån inte ska använda typer tillhörande namnutrymmen för webbtjänster. Finns behov av att använda typer tillhörande namnutrymmen för webbtjänster ska dessa i största möjligaste utsträckning koncentreras till en enskild klass (exempelvis klassen Service men inte i klassen Contact i figur 9).



Figur 9. Förslag på klasser och dess medlemmar för hanteraring av kontaktuppgifter i affärslogiklagret.

### BusinessObjectBase

Den abstrakta klassen BusinessObjectBase implementerar interfacet IDataErrorInfo och innehåller medlemmar för hantering av anpassade felmeddelanden.

Inga medlemmar i klassen BusinessObjectBase är abstrakta varför en klass som ärver från BusinessObjectBase inte behöver implementera några medlemmar. Klassen BusinessObjectBase har gjort abstrakt för att hindra att objekt instansieras av klassen.

Var uppmärksam på att fältet \_validationErrors är ”private”, medlemmarna CommonErrorMessage samt ValidationErrors är ”protected” och övriga medlemmar är ”public”.

### Contact

Den konkreta klassen Contact ärver från BusinessObjectBase och därmed all hantering av anpassade felmeddelanden.

Egenskaperna EmailAddress, FirstName och LastName validerar respektive värde då det tilldelas underliggande fält. Felmeddelanden hanteras med hjälp av medlemmar som ärvs från basklassen. Egenskapen ContactId är auto-implementerad och dess värde behöver inte valideras.

Fält och egenskaper som inte tilldelas värden då de instansieras får lämpligen sina standardvärden för typen.

### Service

Medlemmarna i klassen Service använder presentationslogiklagret vid implementation av CRUD-funktionaliteten (Create Read Update Delete).

Metoden Save används både då en ny kontaktuppgift ska läggas till i tabellen Contact och då en befintlig kontaktuppgift ska uppdateras. Genom att undersöka värdet egenskapen ContactId har för Contact-objektet kan det bestämmas om det är fråga om en helt ny post, eller en uppdatering. Har ContactId värdet 0 (standardvärdet för fält av typen int) är det en ny post. Är värdet större än 0 måste det vara en befintlig post som ska uppdateras.

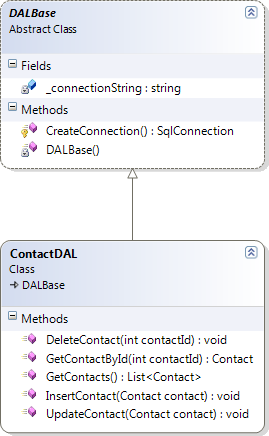
Innan en post skapas eller uppdateras måste Contact-objektet valideras. Misslyckas valideringen ska ett undantag av typen ApplicationException kastas. Genom egenskapen Data i klassen ApplicationException och metoden Add kan en referens till Contact-objektet som inte klarade valideringen skickas med undantaget, som tas omhand och behandlas i presentationslogiklagret.

## Dataåtkomstlagret

Dataåtkomstlagret låter du utgöras av den klass som ska hantera all kommunikation med datalagret. Använd lämpliga klasser och metoder i ADO.NET för att exekvera de lagrade procedurer, se figur 4, som innehåller SQL-satser som SELECT, INSERT, UPDATE och DELET.

### Förslag på klasser

I figur 10 nedan ser du ett klassdiagram med förslag på hur du kan utforma klasserna i dataåtkomstlagret. Du behöver inte utforma dina klasser på samma sätt utan kan skapa egna om du så vill.



Figur 10. Förslag på klasser och dess medlemmar för hantering av kontaktuppgifter i dataåtkomstlagret.

### DALBase

Klassen DALBase är en abstrakt basklass. Ingen av medlemmarna i klassen är abstrakt, utan klassen har gjorts abstrakt för att förhindra att objekt instansieras av klassen.

Fältet \_connectionString och konstruktorn är statiska, och ”private”. Den statiska konstruktorn initierar det statiska fältet genom att hämta anslutningssträngen från filen web.config.

Metoden CreateConnection är ”protected” och skapar och returnerar en referens till ett anslutningsobjekt.

### ContactDAL

Klassen har de medlemmar som krävs för att implementera CRUD-funktionalitet. Metoderna GetContacts och GetContactById används för att hämta alla kontaktuppgifter respektive en enskild kontaktuppgift. InsertContact skapar en ny post i tabellen Contact. UpdateContact uppdaterar en befintlig kontaktuppgift, och DeleteContact tar bort en. Samtliga metoder exekverar de lagrade procedurerna enligt figur 4.

## Hantering av fel på servern

Skulle ett oväntat fel av något slag inträffa ska användaren slippa se en ”gul-ful” sida med ett automatgenererat felmeddelande. Istället ska du visa en sida med ett användarvänligare(?) meddelande. Figur 11 visar ett exempel på en sådan sida.



Figur 11. Anpassad sida som visas vid fel istället för en ”gul-ful” sida.

På sidan *customErrors Element (ASP.NET Settings Schema),* <http://msdn.microsoft.com/en-us/library/h0hfz6fc.aspx>, beskrivs vad du måste ändra på i web.config för att åstadkomma detta.

# Krav

Sammanställning av krav som måste uppfyllas:

1. Webbapplikationen ska vara en femlagerapplikation. Inget lager får ”hoppas över”.
2. Affärslogik- och dataåtkomstlager ska placera i skilda underkataloger i katalogen App\_Code.
3. Dataåtkomstlagret måste utgöras av egenhändigt skrivna klasser med metoder som använder ADO.NET.
4. All kommunikation med databasen måste ske genom användaren appUser som har lösenordet 1Br@Lösen=rd?.
5. Anslutningssträngen får bara finnas lagrad på ett ställe i webbapplikationen, i web.config.
6. Användaren måste bli informerad med ett meddelande då användaren på något sätt försökt/lyckats påverka datat i tabellen Contact. Både rätt- och felmeddelande måste visas.
7. Om rättmeddelande visas på samma sida som används för hantering av kontaktuppgifter ska det försvinna automatiskt efter en tid, t.ex. 10 sekunder.
8. Innan en kontaktuppgift tas bort måste användaren bekräfta det.
9. Samtliga kontaktuppgifter får inte presenteras på en och samma gång, ”paging” måste användas för att begränsa antalet kontaktuppgifter som visas samtidigt.
10. Anpassade felmeddelande ska visas då fel inträffar. Allmän felsida ska finnas men ska i princip inte användas.
11. Affärslogikklasser får inte använda sig av några typer som tillhör namnutrymmen för webbtjänster. Undantag från kravet gäller klass som eventuellt hanterar ”cachat” data.
12. Validering ska ske på klienten och i presentationslogiklagret av data som matas in i textfält.
13. Validering av Contact-objekt ska ske i affärslogiklagret innan objektet skickas vidare till dataåtkomstlagret för att skapa en ny kontaktuppgift eller uppdatera en befintlig. Misslyckas valideringen ska ett undantag kastas.

# Mål

Efter denna laboration kommer du bildat kunskap gällande hur du kan använda ADO.NET och skapa en applikation uppdelad i flera lager. Du ska:

* Kunna implementera en femlagerapplikation.
* Använda ADO.NET för att hämta data från en databas via lagrade procedurer.
* Förstå hantering av databundna kontroller, d.v.s. hur de används deklarativt och programmatiskt.
* Förstå vikten av att, och hur du, använder rätt- och felmeddelanden.
* Förstå vikten av att validera data innan det behandlas vidare, både på klienten och på servern samt vara väl förtrogen med hur du validerar innehållet i ett textfält med hjälp av valideringskontrollerna som ASP.NET erbjuder.
* Ha fått insikt om att data som skickas till affärslogiklagret måste valideras av affärslogiklagret då valideringen kan vara felaktigt implementerad, eller omöjligt att implementera, i användargränssnitt- och presentationslogiklager.
* Förstå att då användaren klickar på en knapp sker en ”postback” och händelser skapas som kan tas om hand i ”code-behind”-filen.

# Läsvärt

* ObjectDataSource-kontrollen
  + Pro ASP.NET 4 in C# 2010, Fourth Edition, 386-401.
  + “*ObjectDataSource Web Server Control Overview*”, <http://msdn.microsoft.com/en-us/library/9a4kyhcx.aspx>.
* ListView-kontrollen
  + Pro ASP.NET 4 in C# 2010, Fourth Edition, 447-453.
  + “*ListView Web Server Control Overview*”, <http://msdn.microsoft.com/en-us/library/bb398790.aspx>.
  + “*ListView Class*”, <http://msdn.microsoft.com/en-us/library/bb459902.aspx>.
  + “*The Only Data-binding Control You'll Ever Need*”, <http://msdn.microsoft.com/en-us/magazine/cc337898.aspx>.
* SqlConnection, SqlCommand och SqlDataReader
  + Pro ASP.NET 4 in C# 2010, Fourth Edition, 283-306.