|  |
| --- |
| Laborationsanvisning |

Äventyrliga kontakter

Steg 1, laborationsuppgift 2

|  |
| --- |
| *Författare:* Mats Loock  *Kurs:* ASP.NET MVC  *Kurskod:*1DV409 |

**Innehåll**

[Problem 4](#_Toc309197858)

[Databas 6](#_Toc309197859)

[Tabellen Contact 6](#_Toc309197860)

[Lagrade procedurer 6](#_Toc309197861)

[Modell 7](#_Toc309197862)

[Klassen Contact och Contact\_Metadata 7](#_Toc309197863)

[Klassen Repository 7](#_Toc309197864)

[Interfacet IRepository 7](#_Toc309197865)

[Klassen ContactController 8](#_Toc309197866)

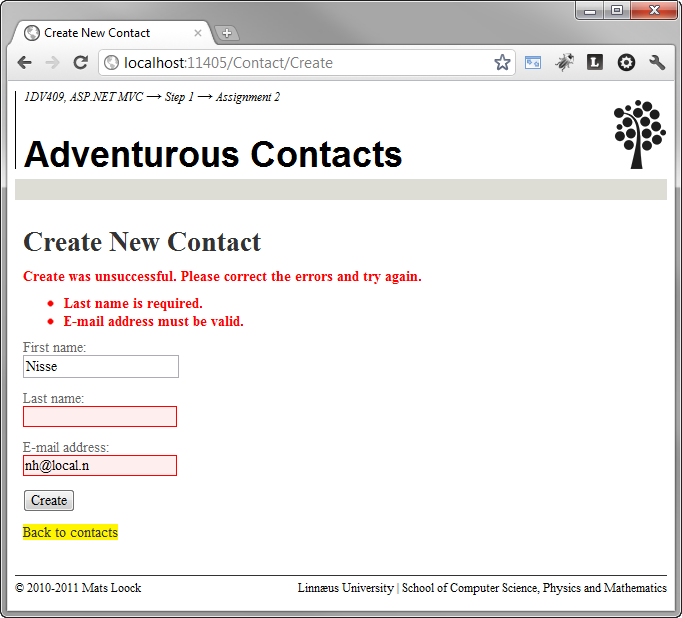
[Klassen HomeController 8](#_Toc309197867)

[Vyer 9](#_Toc309197868)

[Körexempel 10](#_Toc309197869)

# Problem

Skriv en webbapplikation med hjälp av ASP.NET MVC och C# där användaren ska kunna lista och redigera innehållet i tabellen Contact i databasen 1dv409\_AdventureWorksAssignment.



Figur 1. Webbsida efter klientvalidering.

Du får i princip fritt utforma applikationen. Följande krav måste dock uppfyllas:

* Applikationen ska ha CRUD*-*funktionalitet, där CRUD är en akronym för Create (INSERT), Read (SELECT), Update (UPDATE) och Delete (DELETE), d.v.s. den måste minst låta användaren kunna:
  + Skapa nya kontakter.
  + Läsa kontakter.
  + Uppdatera kontakter.
  + Ta bort kontakter.

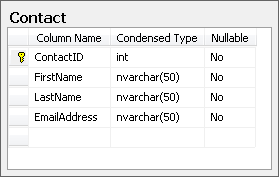
OBS! Det är bara kontakter du själv, eller någon annan kursdeltagare, skapat som kan uppdateras eller tas bort

* Applikationen ska använda sig av*ADO.NET Entity Framework*, eller *LINQ to SQL*, för att skapa datamodellklassen Contact och all kommunikation med databasen.
* Applikationen ska ansluta till databasen 1dv409\_AdventureWorksAssignment på server FALKEN med IP-numret 172.16.214.1. Användaren, som har rättighet att exekvera lagrade procedurer och ställa SELECT-frågor, är appUser och har lösenordet 1Br@Lösen=rd?. Anslutningssträngen med nämnd information måste vara placerad i filen web.config.
* Data måste valideras på såväl klient som server innan det lagras i databasen.
  + Validering ska ske med *data annotations* som placeras i en metadataklass som associeras med datamodellklassen.
  + Förnamn, efternamn och e-postadress måste finnas och får inte bestå av mer än 50 tecken. Tänk på att textfälten med fördel ska vara så pass stora att 50 tecken får plats. Attributen maxlength och size bör komma till användning.
  + E-postadressen måste vara korrekt formaterad. Du hittar ett reguljärt uttryck du kan använda för att validera en e-postadress på <http://www.regular-expressions.info/email.html>.
* Rätt- respektive felmeddelande ska visas för användaren då användaren utför operationer vars syfte är att förändra innehållet i databasen.
* Flera vyer ska applikationen ha, t.ex. en för att lista kontakter, en för att skapa kontakter och en för att redigera kontakter, varför minst en *layout*-fil ska användas.
* Försöker användaren redigera en kontakt som inte finns ska en vy visas som meddelar att kontakten inte finns.
* En allmän felsida ska visas om användaren exempelvis anger en URL som inte finns eller ett undantag kastas som inte hanteras.

## Databas

### Tabellen Contact

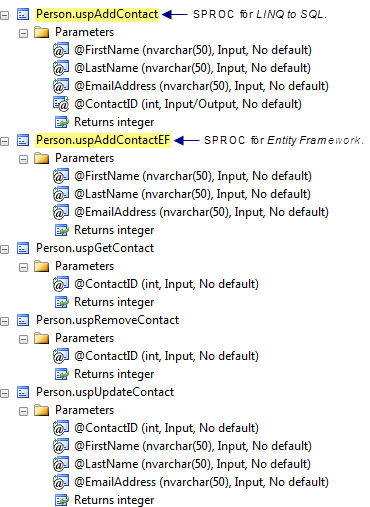
Webbapplikationen ska arbeta med tabellen Contact (starkt förenklad då flertalet fält tagits bort) i databasen 1dv409\_AdventureWorksAssignment, som finns på laborationsservern FALKEN (172.16.214.1).



Figur 2. Tabellen Contact med kolumnnamn och typer.

### Lagrade procedurer

Användaren appUser har rättighet att göra SELECT-frågor direkt mot tabellen. appUser saknar däremot rättigheter att direkt göra INSERT, UPDATE eller DELTET utan gränssnittet mot tabellen utgörs av ett antal lagrade procedurer.



Figur 3. Lagrade procedurer med namn och parametrar.

**OBS!** Lägg märke till att det finns två lagrade procedurer för att lägga till en ny kontakt.

Person.uspAddContact använder en parameter av typen OUTPUT för att göra primärnyckelns värde för den nya posten tillgänglig varför *Entity Framework* inte kan anropa den vilket *LINQ to SQL* däremot kan.

Person.uspAddContactEF är skriven så att *Entity Framework* kan anropa den och primärnyckelns värde görs tillgängligt med en SELECT-sats via NewContactId.

## Modell

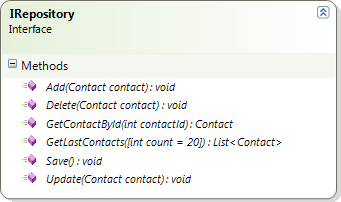
### Klassen Contact och Contact\_Metadata

I klassdiagrammet nedan ser du ett förslag på hur du kan utforma datamodellklassen, och den associerade metadataklassen, som representerar tabellen Contact. Metadataklassen kan vara en helt separat klass enligt Figur 4 eller en nästlad klass enligt Figur 5.

|  |  |
| --- | --- |
| C:\Users\mats\AppData\Local\Temp\SNAGHTMLcf4d7fa.PNG | C:\Users\mats\AppData\Local\Temp\SNAGHTMLcf773fb.PNG |
| Figur 4. Datamodellklassen Contact (alla medlemmar visas inte) med separat metadataklass. | Figur 5. Datamodellklassen Contact (alla medlemmar visas inte) med nästlad metadataklass. |

### Interfacet IRepository

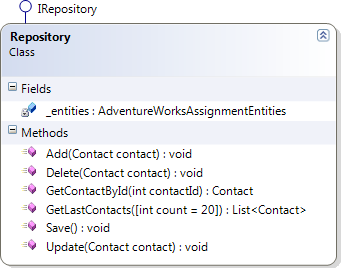
Klassen Repository i Figur 7 implementerar ett interface, IRepository, som definierar vilka medlemmar klassen måste ha.



Figur 6. Interfacet IRespository.

### Klassen Repository

Med fördel använder du dig av designmönstret *Repository* så en controller inte behöver känna till om applikationen använder sig av *Entity Framework* eller *LINQ to SQL*.



Figur 7. Förslag på hur en *Repository*-klass kan utformas.

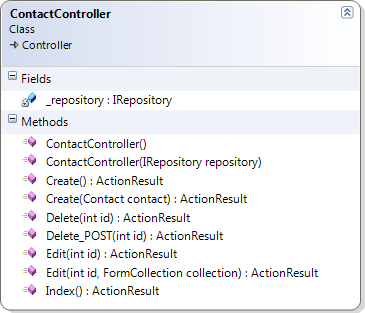
I och med att klassen Repository implementerar *IRepository* medför det att en controller inte behöver hårdkodas att använda enbart den konkreta *Repository*-klassen utan kan med hjälp av designmönstret *Dependency Injection* använda vilken klass som helst så länge som klassen implementerar interfacet.

Metoden GetLastContacts() hämtar de sista poster i tabellen Contact. I den konkreta klassen, men även i interfacet, definieras att parametern count har standardvärdet 20 vilket gör att metoden kan anropas utan att ange ett argument.

## Controller

### Klassen ContactController

Lämpligen skapar du en controllerklass som samordnar all hantering av kontakter. Figur 8 visar hur en sådan klass skulle kunna se ut.

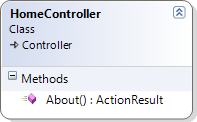


Figur 8. Exempel på controllerklass med funktionalitet som har med kontakter att göra.

Klassen ContactController använder sig av interfacet IRepository och designmönstret *Dependency Injection* (kan delvis ses på att det finns två konstruktorer, en standardkonstruktor, och en konstruktor som tar en parameter av typen IRepository).

### Klassen HomeController

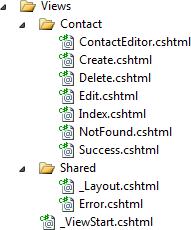
Det kan även vad en god idé att ha en controllerklass som tar hand om övrig funktionalitet som applikationen kan tänkas erbjuda användaren. Figur 9 visar ett exempel på en sådan klass.



Figur 9. Exempel på controllerklass med funktionalitet som inte har med kontakter att göra.

## Vyer

I figuren nedan hittar du ett förslag på vilka vyer som matchar controllerklasserna ContactController och HomeController.



Figur 10. Förslag på vyer.

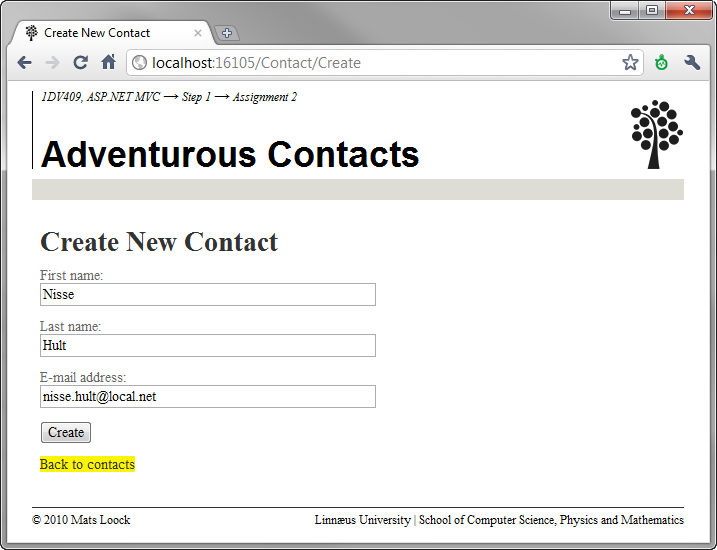
Då du använder flera vyer och behöver visa samma data i flera kan det vara en god idé att placera gemensam kod i en eller flera ”*partial views*” så att du bara har koden på ett ställe (bryt inte mot principen DRY, *Don’t Repeat Yourself*). Den partiella vyn ContactEditor i Figur 10 används av så väl Edit som Create.

# Körexempel

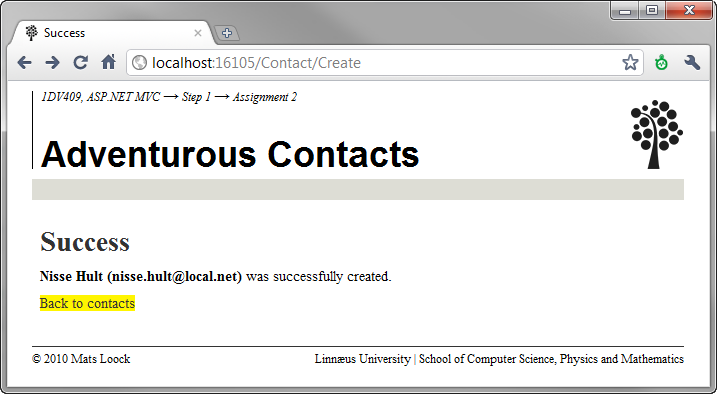
Figurerna nedan är ett förslag på hur gränssnittet kan utformas. Det står dig fritt att utforma det som du finner lämpligt.



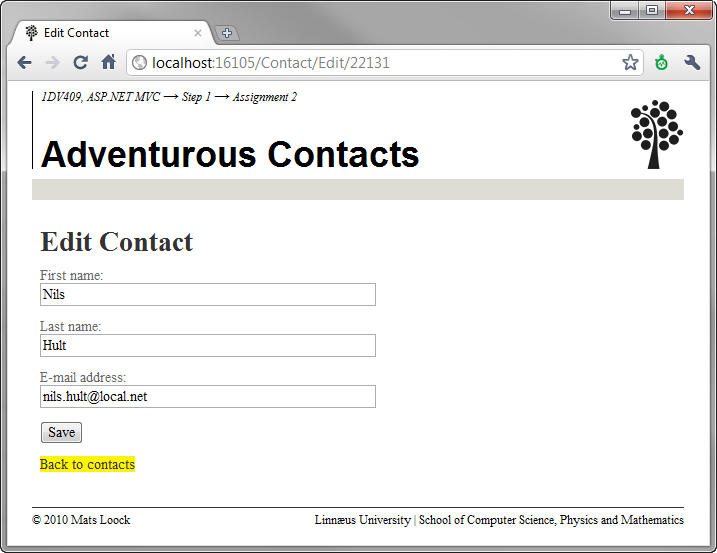
Figur 11. Lista med de sista 20 kontakterna i tabellen Contacts.



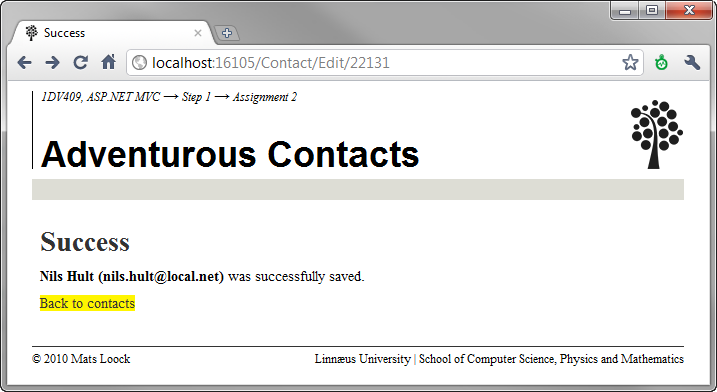
Figur 12. Formulär för att skapa en ny kontakt.



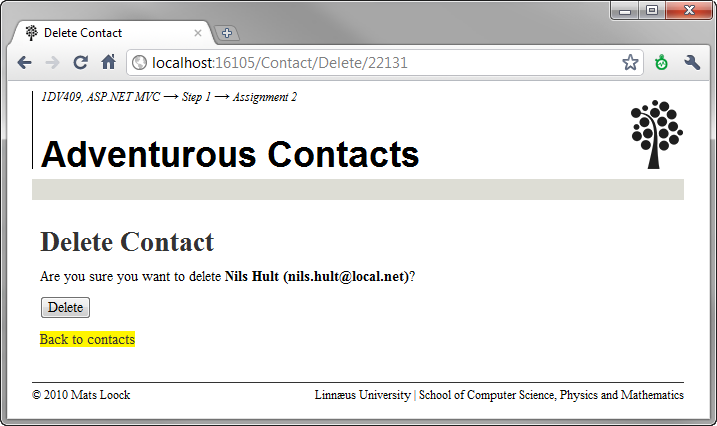
Figur 13. Rättmeddelande då en ny kontakt skapats.



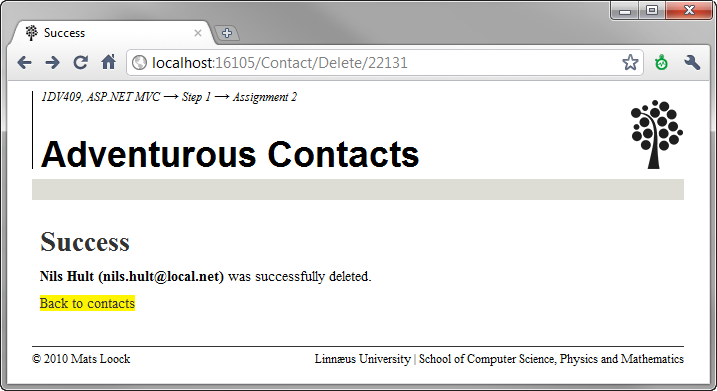
Figur 14. Formulär för redigering av befintlig kontakt.



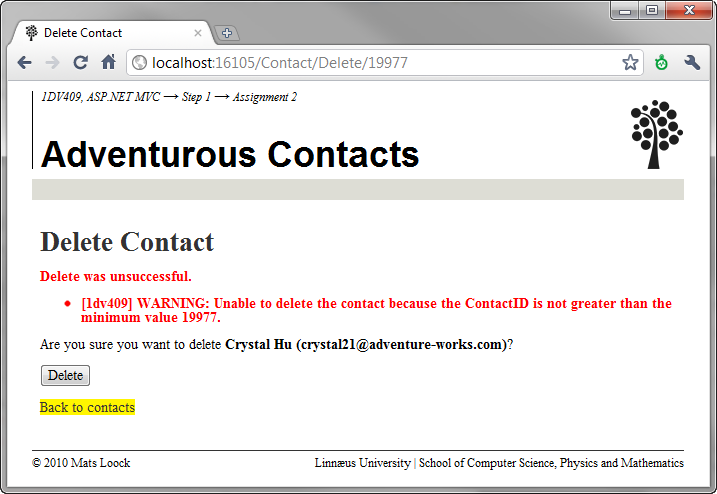
Figur 15. Rättmeddelande då ändringar av en kontakt sparats.



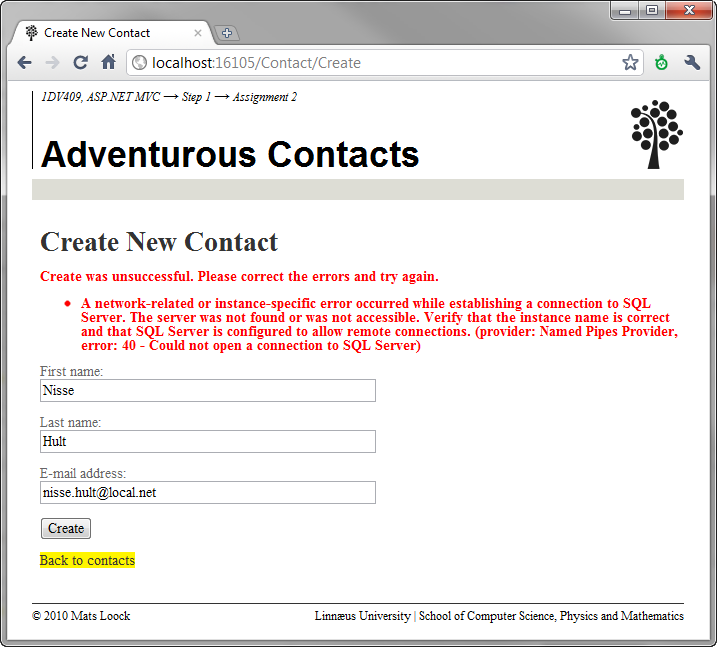
Figur 16. Formulär för bekräftelse av borttagning av en kontakt.



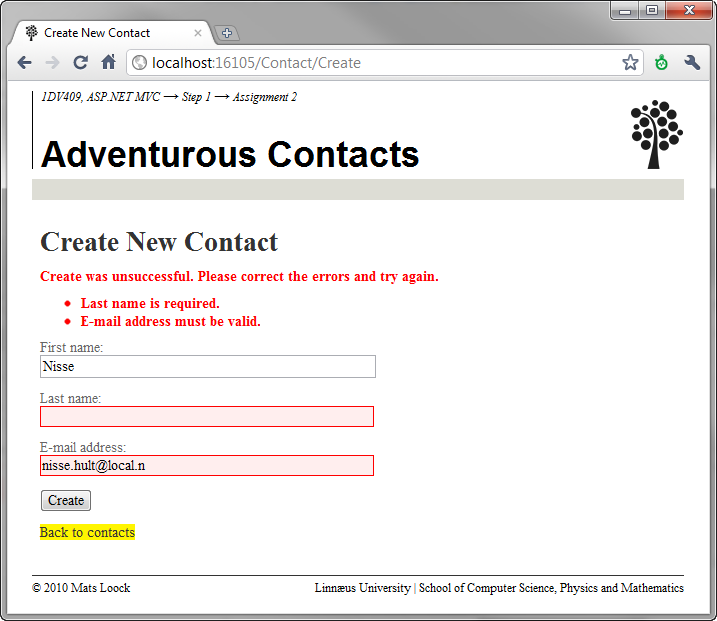
Figur 17. Rättmeddelande efter att en kontakt tagits bort.



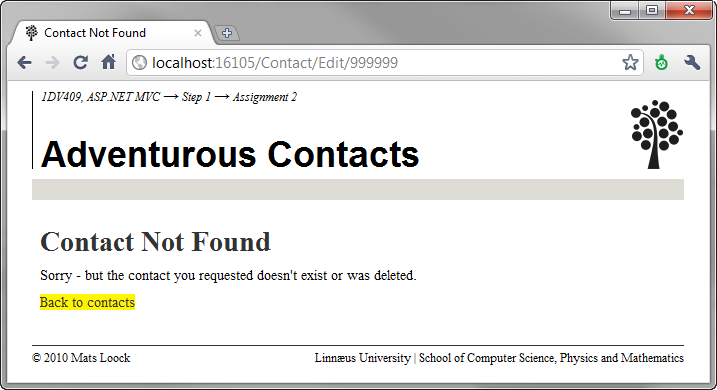
Figur 18. Felmeddelande vid försök att ta bort kontakt som inte går att ta bort.



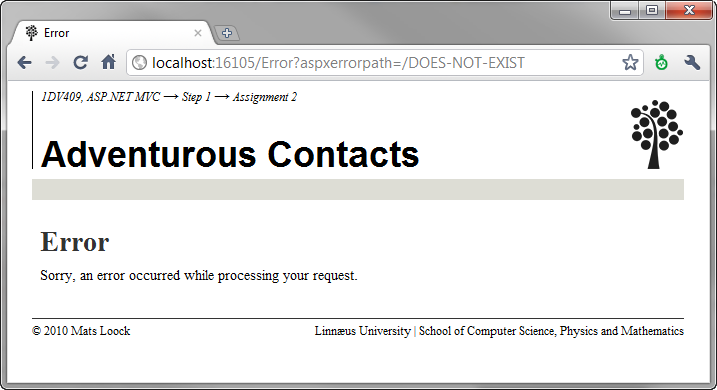
Figur 19. Ett väl detaljerat felmeddelande då en kontakt inte kunde skapas.



Figur 20. Formulär med kontaktuppgifter som inte klarar valideringen.



Figur 21. Vy då användaren försöker redigera en kontakt som inte finns.



Figur 22. Vy som exempelvis visas då användaren anger en URL som inte finns. Visas även då ett undantag kastas som inte hanteras.