Linnéuniversitetet
Kalmar Växjö

Laborationsanvisning

Äventyrliga kontakter

Steg 1, laborationsuppgift 2

Författare: Mats Loock

Kurs: ASP.NET MVC

*Kurskod:*1DV409

Linnéuniversitetet
Kalmar Växjö

Upphovsrätt för detta verk

Detta verk är framtaget i anslutning till kursen ASP.NET MVC (1DV409) vid Linnéuniversitetet.

Du får använda detta verk så här:

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!

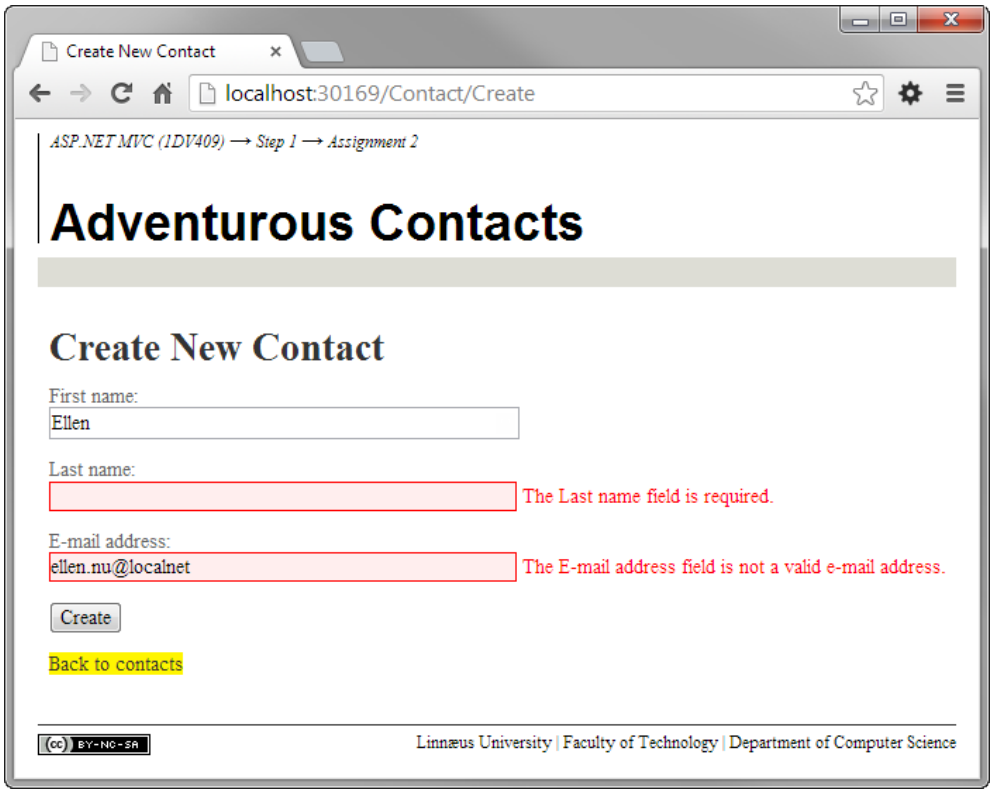
Vid all användning måste du ange källan: ”Linnéuniversitetet – ASP.NET MVC” och en länk till https://coursepress.lnu.se/kurs/aspnet-mvc och till Creative Common-licensen här ovan.

Linnéuniversitetet
Kalmar Växjö

Innehåll

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:

1. 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:
    a. Skapa nya kontakter.
    b. Läsa kontakter.
    c. Uppdatera kontakter.
    d. Ta bort kontakter.

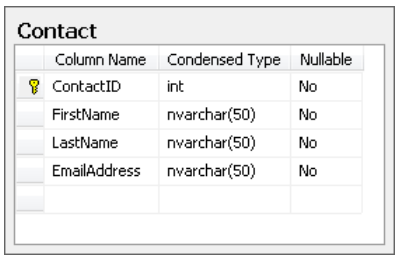
    OBS! Det är bara kontakter du själv, eller någon annan kursdeltagare, skapat som kan uppdateras eller tas bort.
2. Applikationen ska använda sig av *ADO.NET Entity Framework* för all hantering av persistent data.
3. Applikationen ska ansluta till databasen 1dv409_AdventureWorksAssignment på servern FALKEN 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`.
4. Data måste valideras på såväl klient som server innan det lagras persistent.
    a. Validering ska ske med *data annotations*, som placeras i en metadataklass.

   b. 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.
   c. E-postadressen måste vara korrekt formaterad.
5. Rätt- respektive felmeddelande ska visas för användaren då användaren utför operationer vars syfte är att på något sätt förändra persistent data.
6. Applikationen ska ha flera vyer, 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.
7. Försöker användaren redigera en kontakt som inte finns ska en vy visas som meddelar att kontakten inte finns.
8. En allmän felsida ska visas om användaren exempelvis anger en URL som inte finns eller ett undantag kastas som inte hanteras.
9. Efterfrågas en resurs som inte finns eller kan hittas ska applikationen visa en egen felsida istället för den som webbserver tillhandahåller.

Databas

Tabellen Contact

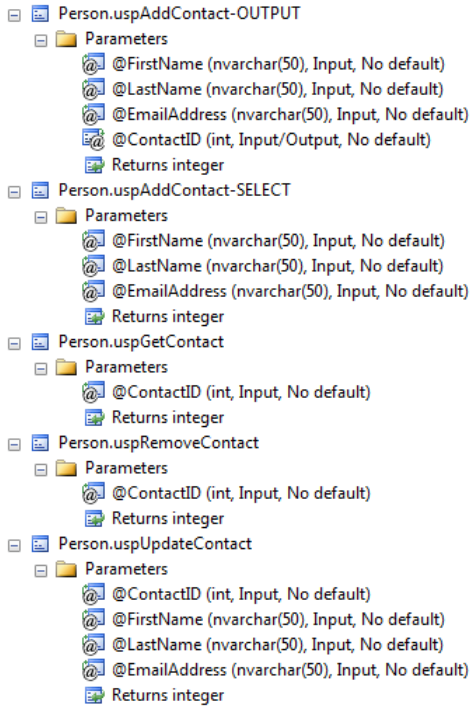
Webbapplikationen ska arbeta med tabellen Contact (starkt förenklad då flertalet fält tagits bort) i databasen 1dv409_AdventureWorksAssignment, som finns på laborationsservern FALKEN (172.16.214.1).



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.

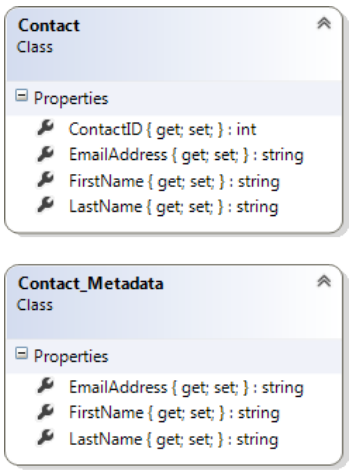
OBS! Lägg märke till att det finns två lagrade procedurer för att lägga till en ny kontakt. `Person.uspAddContact-OUTPUT` använder en parameter av typen OUTPUT för att göra primärnyckelns värde för den nya posten tillgänglig varför du inte kan använda den om du i datamodellen mappar ”insert”, ”update” och ”delete” mot lagrade procedurer.

`Person.uspAddContact-SELECT` är skriven så att primärnyckelns värde görs tillgängligt med en SELECT-sats via NewContactId.

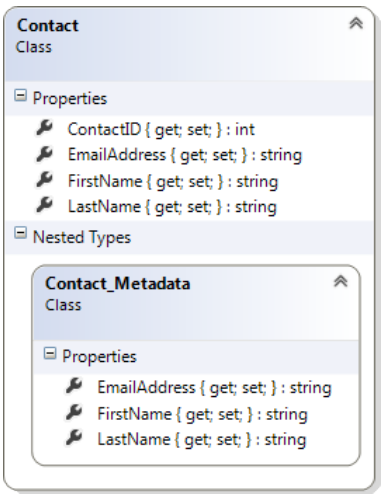
Modell

Klassen Contact och Contact_Metadata

I klassdiagrammet nedan ser du ett förslag på hur du kan utforma POCO-klassen, 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.



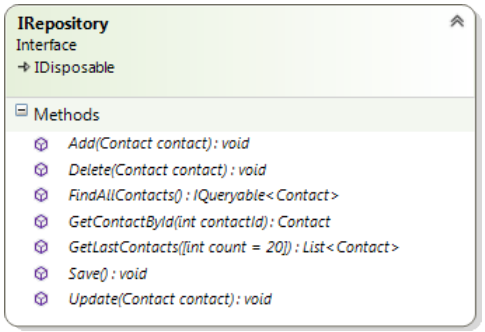
Figur 4. POCO-klassen Contact med separat metadataklass.



Figur 5. POCO-klassen Contact med nästlad metadataklass.

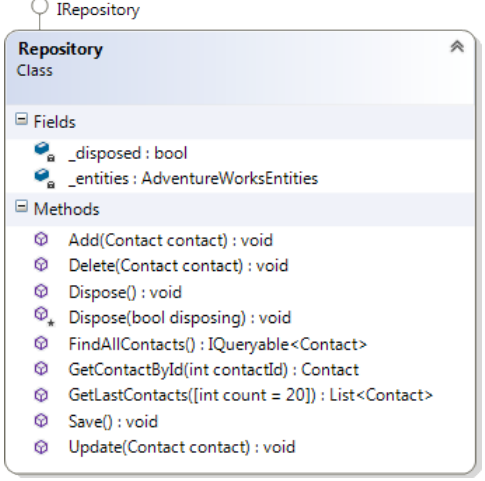
Interfacet IRepository

Klassen `Repository` i figur 7 implementerar ett interface, `IRepository`, som definierar vilka medlemmar klassen måste ha. Uppmärksamma att interfacet ärver från interfacet `IDisposable`.



Figur 6. Interfacet `IRespository`.

Klassen Repository



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

Med fördel använder du dig av designmönstret *Repository* så en controller inte behöver känna till vilken typ av persistent ramverk applikationen använder sig av.

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.

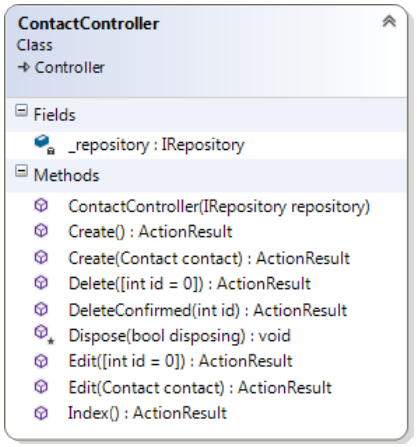
Metoden `GetLastContacts()` hämtar de sista poster i tabellen Contact. I den konkreta klassen, men även i interfacet, definieras att parametern `count` har standardvärdet `20` vilket gör att metoden kan anropas utan att ange ett argument.

Klassen måste även implementera interface `IDisposable` som ju `IRepsoitory` ärver från. Klassen kan då erbjuda funktionalitet för att återlämna systemresurser som t.ex. databasanslutningar.

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.

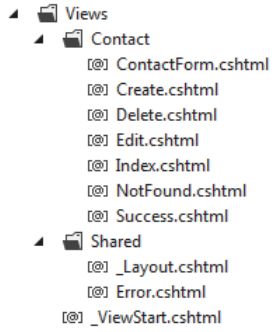


Figur 8. Exempel på controllerklass med funktionalitet som har med kontakter att göra.

Klassen `ContactController` använder sig av interfacet `IRepository` och *Dependency Injection* (kan delvis ses på att det finns en konstruktor som tar en parameter av typen `IRepository` medan standardkonstruktor saknas). Klassen överskuggar metoden `Dispose` för att kunna anropa metoden `Dispose` för `Repository`-objektet.

Vyer

I figuren nedan hittar du ett förslag på vilka vyer som matchar controllerklassen `ContactController`.

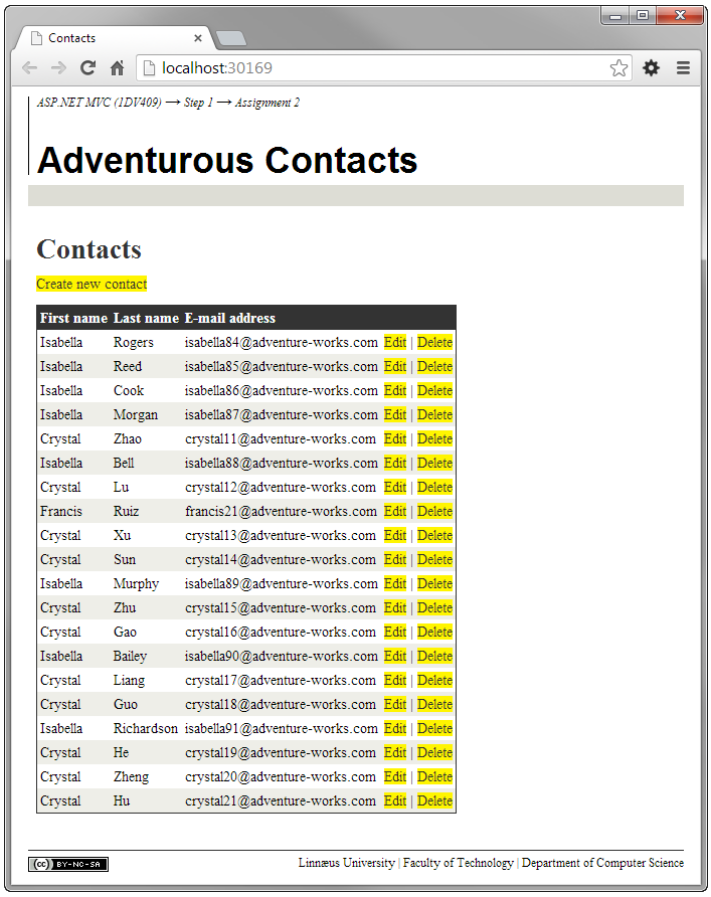


Figur 9. Förslag på vyer.

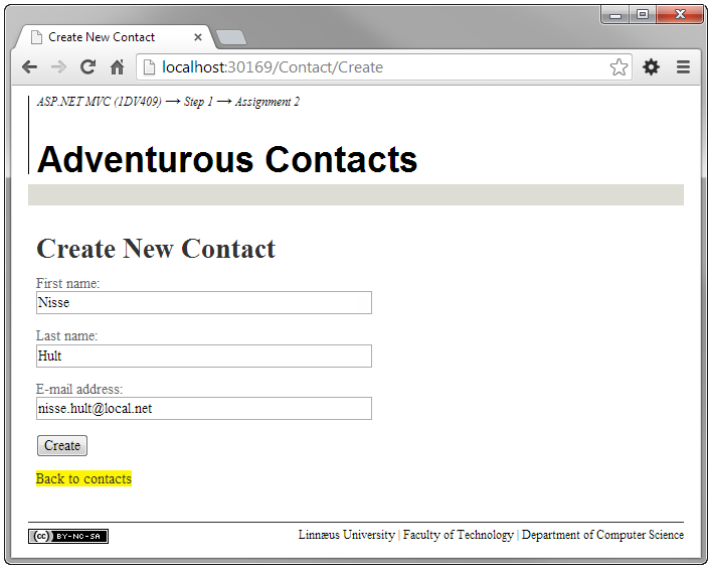
Då du använder flera vyer och behöver visa samma data i flera kan det vara en god idé att placera gemensam kod i en ”*partial view*” så att du bara har koden på ett ställe (bryt inte mot principen DRY, *Don’t Repeat Yourself*). Den partiella vyn `ContactForm` i figur 10 används av såväl `Edit` som `Create`.

Körexempel

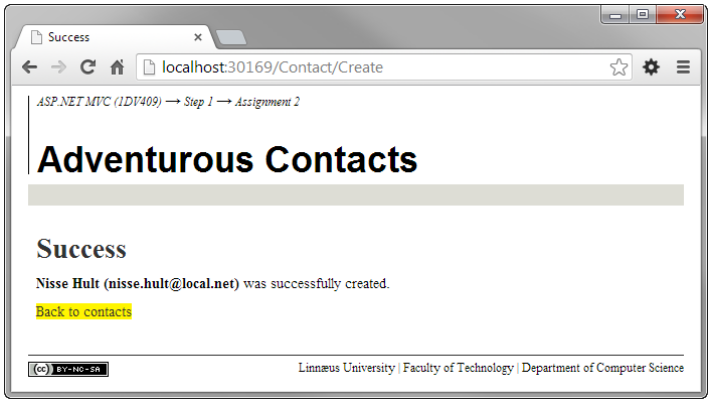
Figurerna nedan är ett förslag på hur gränssnittet kan utformas. Det står dig fritt att utforma det som du finner lämpligt.



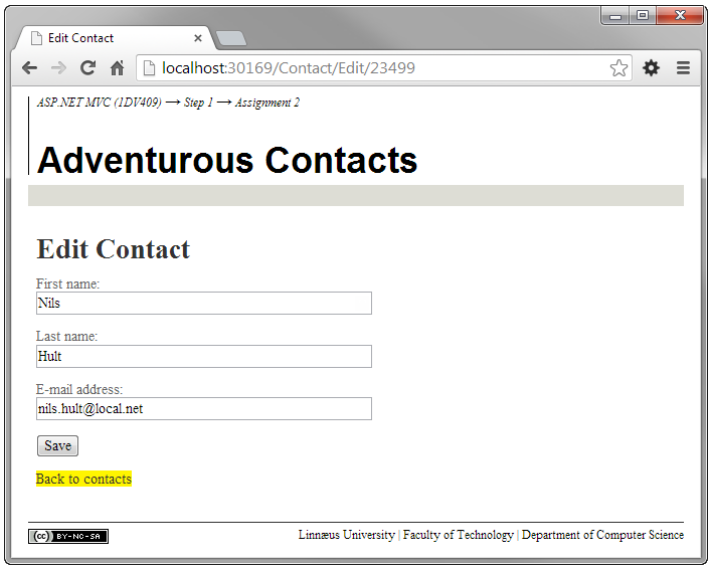
Figur 10. Lista med de sista 20 kontakterna i tabellen Contacts.



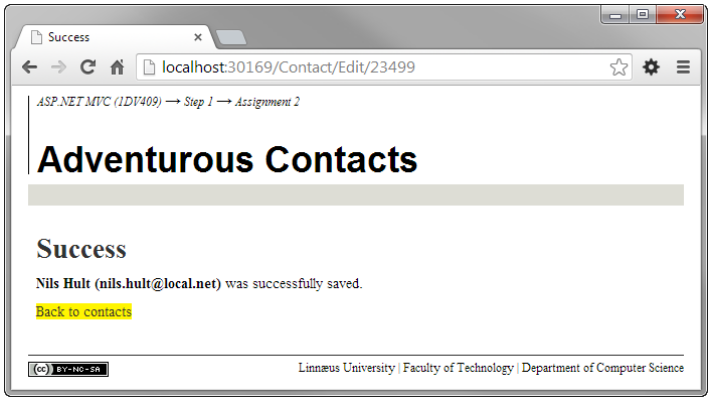
Figur 11. Formulär för att skapa en ny kontakt.



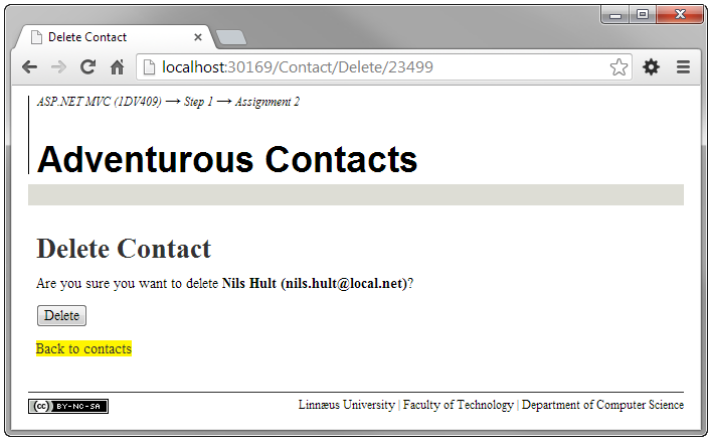
Figur 12. Rättmeddelande då en ny kontakt skapats.



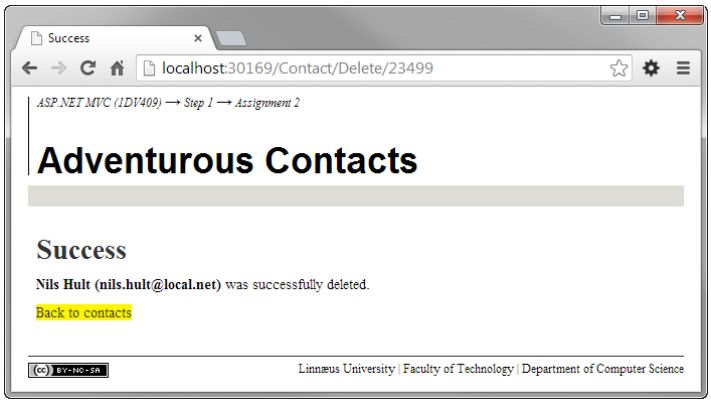
Figur 13. Formulär för redigering av befintlig kontakt.



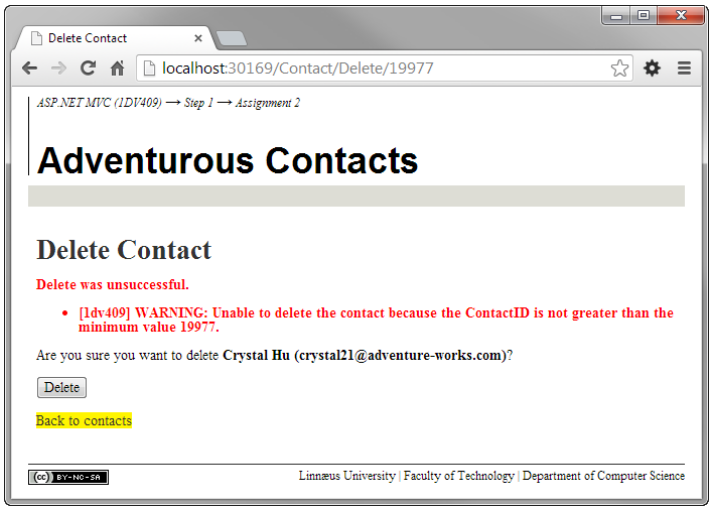
Figur 14. Rättmeddelande då ändringar av en kontakt sparats.



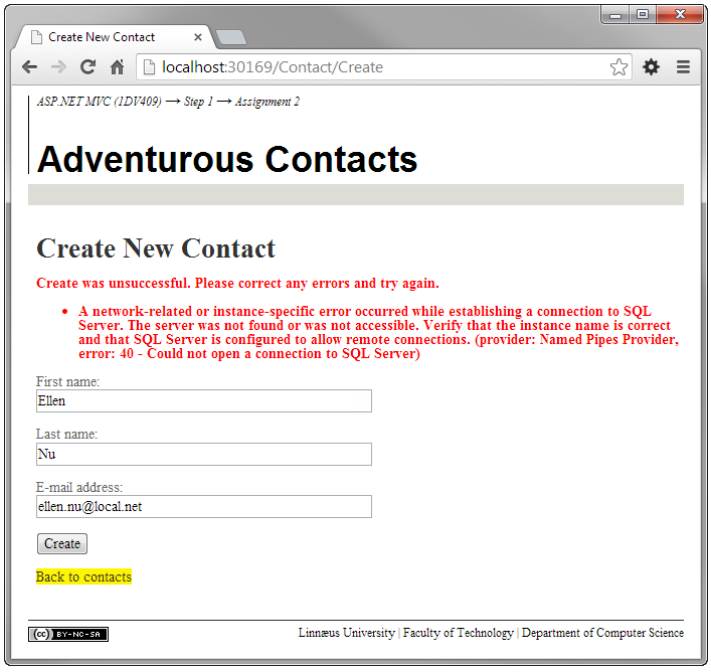
Figur 15. Formulär för bekräftelse av borttagning av en kontakt.



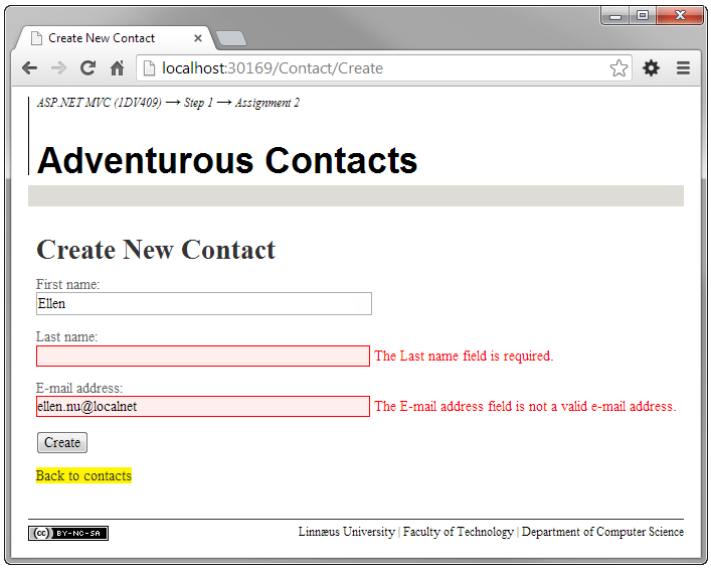
Figur 16. Rättmeddelande efter att en kontakt tagits bort.



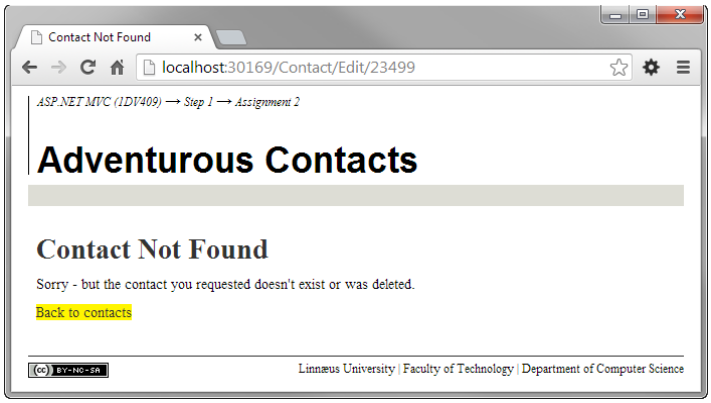
Figur 17. Felmeddelande vid försök att ta bort kontakt som inte går att ta bort.



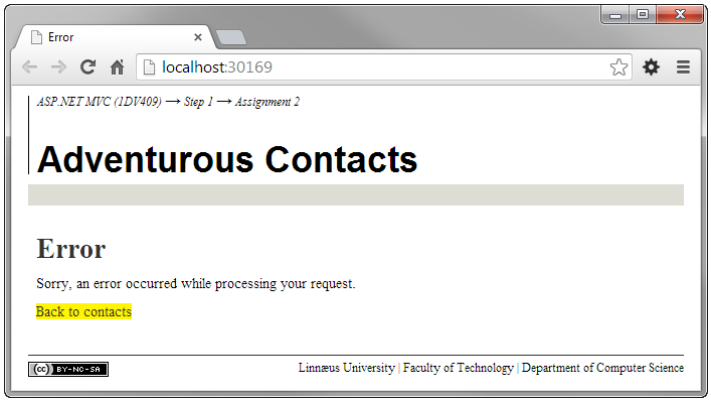
Figur 18. Ett väl detaljerat felmeddelande då en kontakt inte kunde skapas.



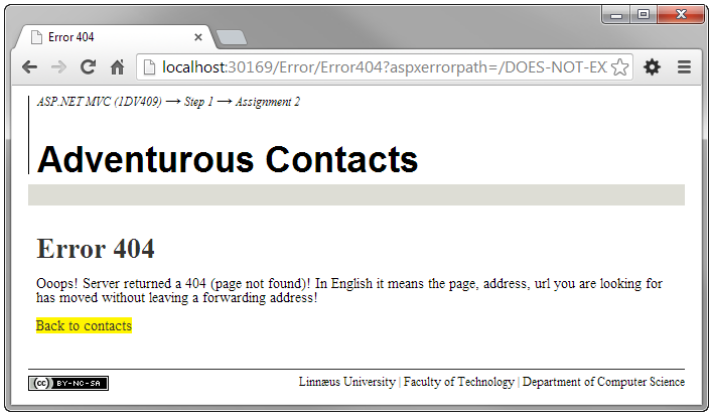
Figur 19. Formulär med kontaktuppgifter som inte klarar valideringen.



Figur 20. Vy då användaren försöker redigera en kontakt som inte finns.



Figur 21. Vy som visas då ett undantag kastas som inte hanteras.



Figur 22. Vy som visas då klienten efterfrågar en resurs som inte finns.