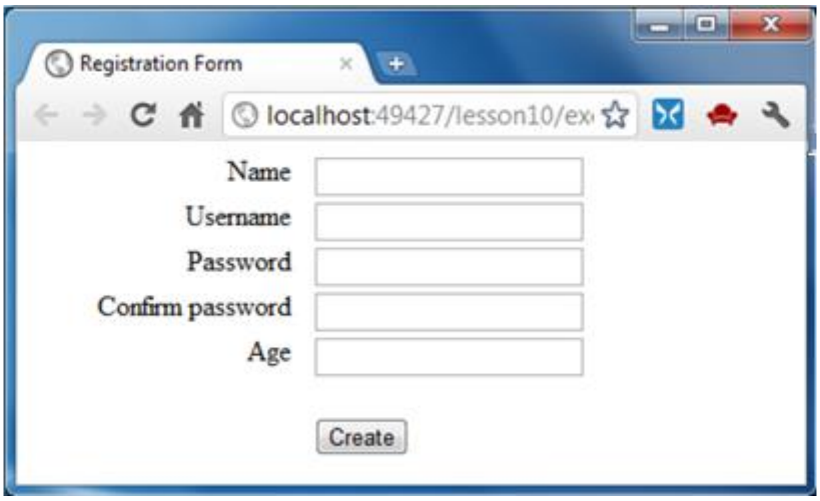
Lesson 10

Exercise 1

In this exercise, you must build a registration form with the fields you see here:



Create a `User` class with properties `Fullname`, `Username`, `Password`, `ConfirmPassword`, and Age. All properties must be strings except Age that is an `int`.

You must use in a strongly typed view for the form, and you must construct it in accordance to these guidelines:

Mandatory fields

- Name
- Username
- Password
- Confirm password
- Age

Additional requirements:

- Name must be more than four characters
- The values in the two fields "Password" and "Confirm password" has to be the same
- Passwords must be at least six characters long
- Age must be in the range between 1 and 120

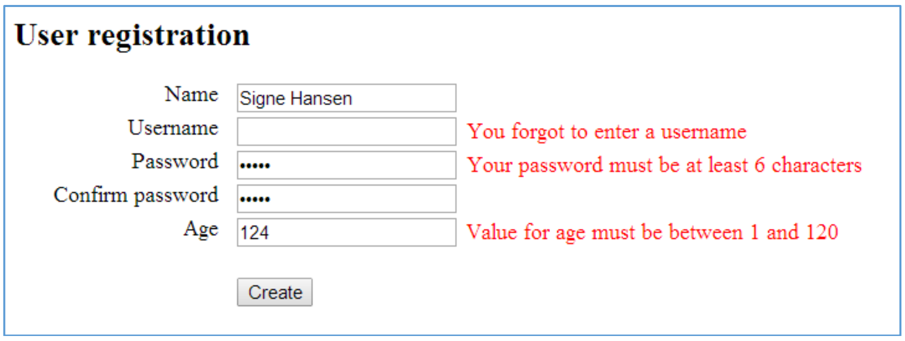
To implement that, you must:

1. Specify validation rules by using metadata for your model.

2. Add a controller `CreateBooking` that returns the form from the view.

3. Create a strongly typed view based on your `User` class and use the "Create" scaffold template when you add the view. It generates a basic form for you.

4. Add a controller with the same name (`CreateBooking`) that acts on `HttpPost`, and receives the model as a parameter, and then returns it to the view with error messages if the model is not valid:



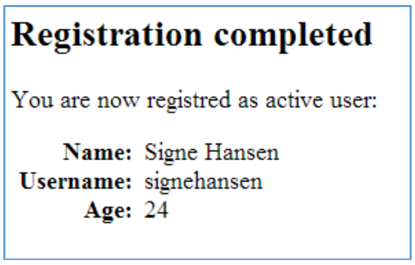
User registration

Name	Signe Hansen	
Username		You forgot to enter a username
Password	•••••	Your password must be at least 6 characters
Confirm password	•••••	
Age	124	Value for age must be between 1 and 120

5. Test the form and make sure the validation meets the requirements.

6. Enable client side validation.

7. Send model data to a completion page if the model is valid:



Registration completed

You are now registred as active user:

Name:	Signe Hansen
Username:	signehansen
Age:	24

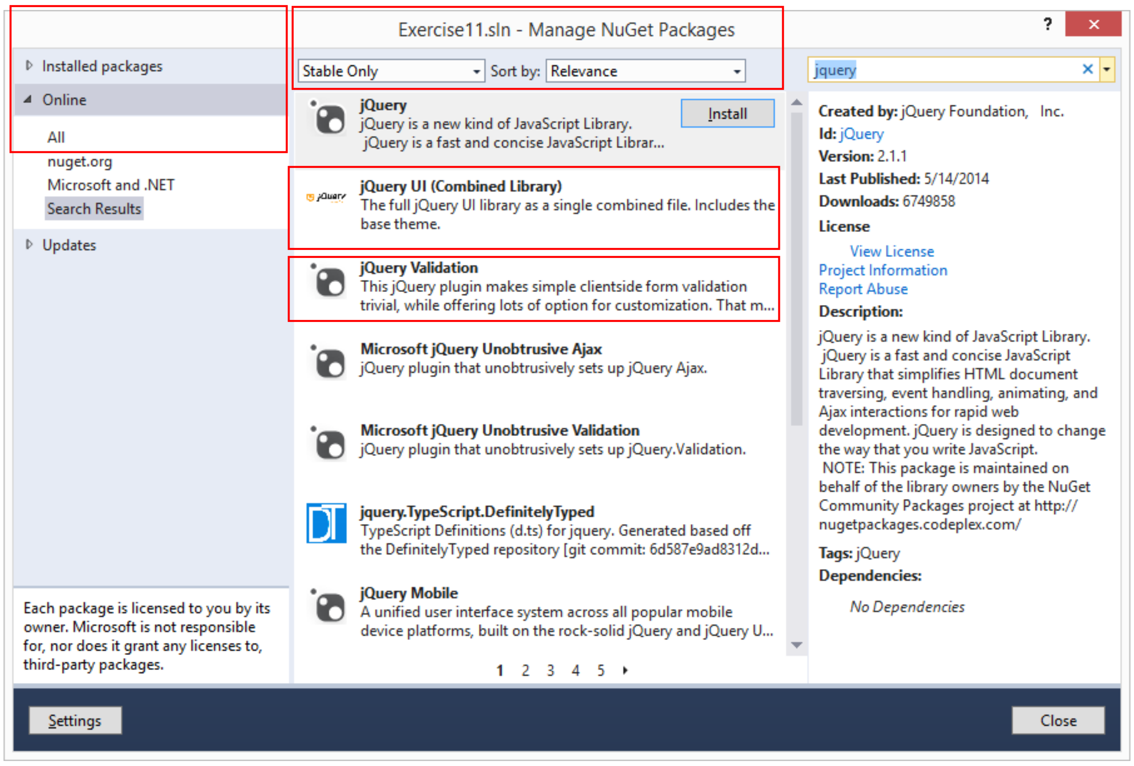
Tip: To download and installed the required JavaScript libraries, you can use the package manager console:

```
PM> Install-Package jQuery
```

```
PM> Install-Package jQuery.Validation
```

```
PM> Install-Package
Microsoft.jQuery.Unobtrusive.Validation
```

or alternatively, the “Manage NuGet Packages” interface:



Exercise11.sln - Manage NuGet Packages

Installed packages
Online
All
nuget.org
Microsoft and .NET
Search Results
Updates

Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

Stable Only | Sort by: Relevance | jquery

jQuery — jQuery is a new kind of JavaScript Library. jQuery is a fast and concise JavaScript Librar... [Install]

jQuery UI (Combined Library) — The full jQuery UI library as a single combined file. Includes the base theme.

jQuery Validation — This jQuery plugin makes simple clientside form validation trivial, while offering lots of option for customization. That m...

Microsoft jQuery Unobtrusive Ajax — jQuery plugin that unobtrusively sets up jQuery Ajax.

Microsoft jQuery Unobtrusive Validation — jQuery plugin that unobtrusively sets up jQuery.Validation.

jquery.TypeScript.DefinitelyTyped — TypeScript Definitions (d.ts) for jquery. Generated based off the DefinitelyTyped repository [git commit: 6d587e9ad8312d...

jQuery Mobile — A unified user interface system across all popular mobile device platforms, built on the rock-solid jQuery and jQuery U...

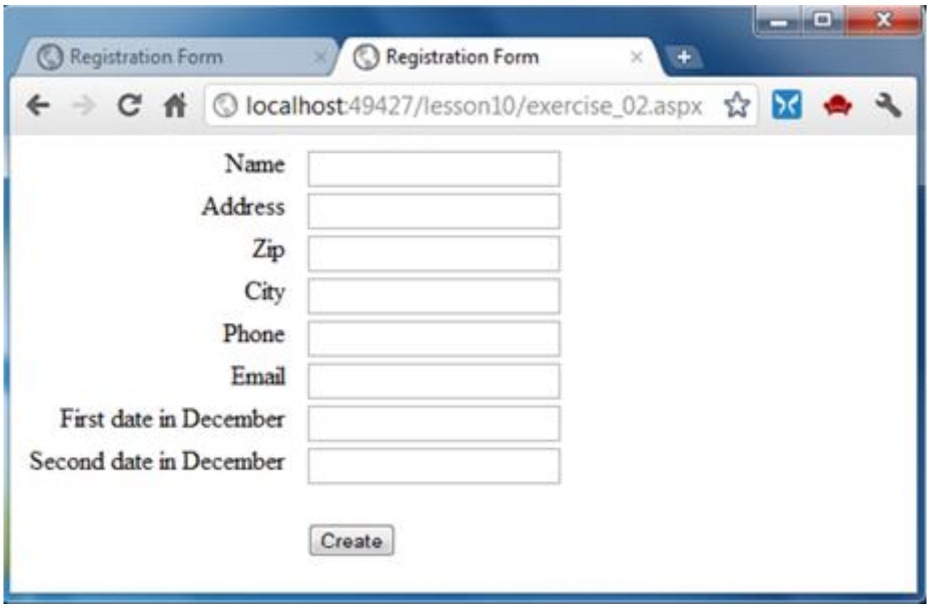
1 2 3 4 5

Created by: jQuery Foundation, Inc.
Id: jQuery
Version: 2.1.1
Last Published: 5/14/2014
Downloads: 6749858
License
View License
Project Information
Report Abuse
Description:
jQuery is a new kind of JavaScript Library. jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is designed to change the way that you write JavaScript. NOTE: This package is maintained on behalf of the library owners by the NuGet Community Packages project at http://nugetpackages.codeplex.com/
Tags: jQuery
Dependencies:
No Dependencies

Settings | Close

Exercise 2

You must build a registration form for a telephone interview:



Registration Form
localhost:49427/lesson10/exercise_02.aspx

Name	
Address	
Zip	
City	
Phone	
Email	
First date in December	
Second date in December	

Create

1. Create an `Interviewee` class with properties that corresponds to the form fields

2. The two date fields indicate the dates on which the registrant wants to be called for the first and the second telephone interview respectively.

3. The validation rules for the form are:

   Mandatory fields:

   - Name
   - Address
   - Zip
   - City
   - Phone
   - E-mail
   - Date for the first interview
   - Date for the second interview

   Additional requirements

   1. ZIP code and telephone number must be valid according to Danish standard (use a regular expression).

   2. E-mail address must be formally validated as an email address (use a regular expression)

   3. Both dates should be after the current date (add a custom validation attribute for that).

      Tip: See *Pro ASP.NET MVC 5* for instructions of how to write a custom validation attribute (pp. 724-27)

   4. The final date must be after the first date (extend the model to be a self-validating model for that rule).

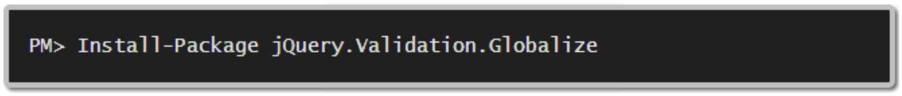
      Tip: See *Pro ASP.NET MVC 5* for instructions of how to write a self-validating model (pp. 729-30)

Create a strongly typed view based on your `Interviewee` class and use the “Create” scaffold template when you add the view. It drastically speeds up the process of creating the form.

Globalize

You might have a problem with dates because jQuery Validate reads dates in US format (mm/dd/yy). You can patch the jQuery Validation date and number validation to be locale specific by using the jquery.validate.globalize.js file.

You can download and install it from Nuget:



```
PM> Install-Package jQuery.Validation.Globalize
```

See http://jqueryvalidationunobtrusivenative.azurewebsites.net/AdvancedDemo/Globalize for further instructions.

After including the globalize files you must also include a small JavaScript to enable jQuery validate to accept the "-" date part separator, like in "dd-mm-yyyy":

```
Globalize.culture("da-DK");

$.validator.methods.date = function (value, element) {
    return this.optional(element) ||
        Globalize.parseDate(value) ||
        Globalize.parseDate(value, "dd-MM-yyyy") ||
        Globalize.parseDate(value, "yyyy-MM-dd");
}
```

Now you have client side validation set for the local culture, you should do the same for server side validation. To do that, you must configure ASP.NET MVC for local culture setting. Open the `Web.config` file and add this line:

```
<globalization uiCulture="da-DK" culture="da-DK" />
```

to the `<system.web>` node.

See http://msdn.microsoft.com/en-us/goglobal/bb896001.aspx for a full list of Microsoft local languages and cultures.

Exercise 3

In this exercise, you must improve the registration form you did in exercise 1. Username is unique, and when users want to register you must examine whether the username is available or already take by another user. It is your job to write a *remote validation attribute* (ajax validation) and a *custom validation attribute* (server side validation) to ensure that.

To add the remote validation attribute, you must follow these steps:

1. For remote validation create a new Controller and name it `RemoteValidationController`.

2. Add a new ActionResult named `UniqueUserName` to the `RemoteValidationController` and specify the return type to be a `JsonResult`.

3. You need to get access to users from the `UniqueUserName` method. To do so, you can create a `Users` class as a repository class to hold a list of instances of `User`. The `Users.cs` file is located

inside a new `InfraStructure` folder.

4. In the `Users` class you must declare a static method `GetUsers` that returns a list of users:

```
List<User> users = new List<User>();
users.Add(new User { Fullname = "Peter Nielsen", Username = "peter", Password = "ptr3nls", Age = 24 });
users.Add(new User { Fullname = "Thomas Larsen", Username = "tmlar", Password = "thm14lar", Age = 32 });
users.Add(new User { Fullname = "Vibeke Hansen", Username = "vibe", Password = "vibe2hanse", Age = 22 });
users.Add(new User { Fullname = "Susan Olsen", Username = "suol", Password = "susanol", Age = 34 });
```

5. Also, you must create a static method `UsernameIsUnique(string username)` that returns false if `username` is already taken; otherwise true. Therefore, the return type must be a Boolean.

   To check if a username exists in the `users` collection, you can use LINQ:

```
      var queryResult  = from User in users
                           where User.Username == username
                           select users;
```

   The result (`queryResult`) is an `IEnumerable`. If you convert that to a `List,` you can use the method `Count` to check if there are any items in the list:

```
   queryResult.ToList().Count > 0
```

6. When you have completed `UsernameIsUnique`, you can call `Users.UsernameIsUnique(string username)` from the `RemoteValidationController`. The return type is a `Json` object (see *Pro ASP.NET MVC 5*, pp. 736-37).

7. Test the controller inside the browser by calling the action with username as query string, like for example (modify the port number to match your project):

   `http://localhost:56508/RemoteValidation/UniqueUserName?username=vibe`

8. Add the action method as a `Remote` attribute to `Username` in the `User` class.

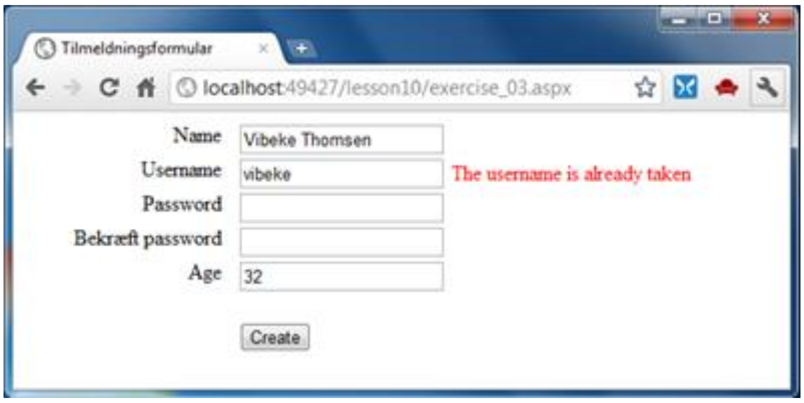
9. Test the user registration form and make sure that the *client side validation* for unique `Username` works as expected.

   Tip:

   1.
   For further instructions, see *Pro ASP.NET MVC 5*, pp. 735-38, which gives an example of how to create and use a custom remote validation attribute.

   2.
   If you want to disable access to the `RemoteValidationController` for all request expect Ajax, you can create your own user defined `ActionFilterAttribute` for that. See http://jdav.is/2015/08/10/ajaxonly-attribute-for-mvc-actions/ for an instruction of how to do that.

10. Now client side validation works. The next step is to add *server side validation*. You'll do that by adding a *custom validation attribute*. With the `Users` class in the `Infrastructure` folder and its `UsernameIsUnique` method, you can easily do that. See *Pro ASP.NET MVC 5* for further instructions how it is done pp. 724-27.



Name	Vibeke Thomsen
Username	vibeke — The username is already taken
Password	
Bekræft password	
Age	32

Disable JavaScript in the browser to test that the server side validation works as expected.

Exercise 4

Add validation to the MbmStore registration form – http://localhost:[portnumer]/Cart/Checkout.