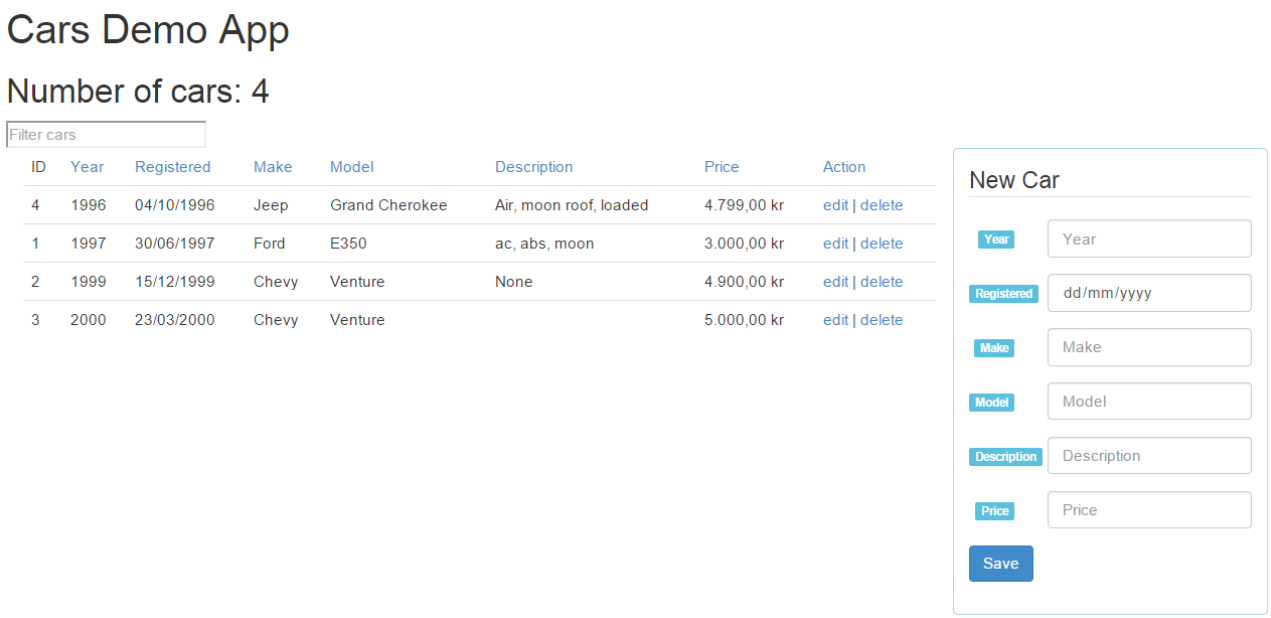
Angular Continued – Controllers, Data Binding and Routing

Adding CRUD to the "car" application

Yesterday you hopefully completed the "cars" exercise. Today we will continue with this exercise, but expand it to support all the four CRUD operations as sketched below.



As yesterday, it is still a client only exercise, but don't worry; we will soon add a backend ☺

The UI above would probably have gained from a separate view for the "new cars" stuff (or would it?). When we introduce Angular routing, you will see that this is an "easy" thing to do.

Use the start code carsV1Solution.zip, even if you completed the exercise yesterday. This code changes things a little bit, for example each car is provided with a unique id, and the JavaScript code is placed in a separate file.

Hints:

- If you think this sounds complicated then, don't worry, we are using angular ☺
- Use the section "End to end application using AngularJS Controller" in this document for an almost identical example: http://viralpatel.net/blogs/angularjs-controller-tutorial/
- I have provided an up to date version of the example above which you will find here: http://fall2015.azurewebsites.net/angularControllers/angularControllers.html#13

Two Way Data Binding and Form Controls

Download the file FormControlsStart.zip, unzip and open the project.

The project includes an html file with a "complex" form with a lot of hardcoded values.

The task is to investigate two way bindings and how to set up check box groups, radio buttons and select controls dynamically.

Some of the tasks below can be pretty tricky, but they all cover important aspects regarding Form design.

1)

Run or view the index.html file. Observe the two way binding in action when you type something in the name or gender fields.

2) Bind the email field to the corresponding field in the Controller

3)

The Form contains a number of check boxes, but they are all hardcoded into the html. Often we would like to set up these options via data provided from, for example, a database.

Change the Form to set up the Checkboxes dynamically using the values in the `emotions` array in the controller.

The actual selected values should bind to the controllers: `feedback.selectedEmotions` array

Hint: This is a tricky one that requires you to hook up to both the checked (ng-checked) event and clicked (ng-click). See here for an example: http://jsbin.com/ImAqUC/1/

4)

The form also contains a number of hard coded radio buttons. Remove them all and set up the radio buttons using the `satisfactionValues` array in the controller.

The actual selected value (remember, there can be only one) should bind to the controllers feedback.`satisfactionValue` property

5) Bind the comments input field to the corresponding value in the controller.

6)

Remove the hardcoded values from the "Location visited" select box and initialize it with values from the controllers `locations` array.

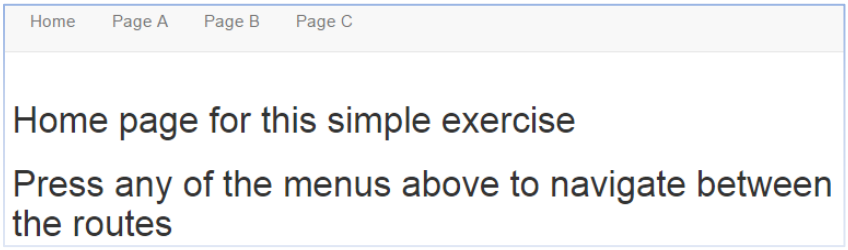
Bind the selected value to `feedback.location`.

Hint: https://docs.angularjs.org/api/ng/directive/ngOptions

Routing, the basics

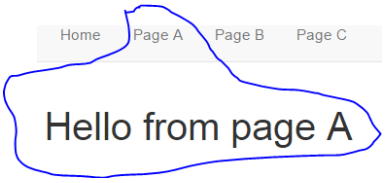
In this first exercise you should not use any seed, but create an empty WebStorm project to familiarize yourself with; what is necessary when we do routing with angular.

1) Create an index.html with a menu as sketched below:



2) Provide the necessary JavaScript to navigate between four routes.

a) For the Home link and for any non-existing routes it should display something similar to above.
b) For the A-C links, it should show a corresponding template, with a message obtained from the controller as sketched below:



Hints: If you add this reference to bootstrap in your head section:

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.0/css/bootstrap.min.css">
```

This code will produce the menu used above:

```
<nav class="navbar navbar-default" role="navigation">
  <div class="container">
    <div  >
      <ul class="nav navbar-nav">
        <li><a href="#">Home</a></li>
        <li><a href="#a">Page A</a></li>
        <li><a href="#b">Page B</a></li>
        <li><a href="#c">Page C</a></li>
      </ul>
    </div>
  </div>
</nav>
```

Routing - 2, loading local views

In the exercise above, templates where initially fetched from the server when pressed (but only once, convince yourself, using Chrome Developer Tools). Change this exercise to include all templates within the original *index.html* page. This will provide a larger initial load on the server, but also a SPA, which after this, only need the server to fetch data[1].

[1] Use the section "How to load local views" in this reference: http://viralpatel.net/blogs/angularjs-routing-and-views-tutorial-with-example/