Study Point Exercise-6

October 14. 2016 Please observe the deadline for these exercises

You can earn a maximum of **7** points for these exercises as outlined below:

• Problem-1:	1 point
• Problem-2:	1 point
• Problem-3:	1 point
• Problem-4:	1 point
• Problem-5:	1 point
• Bonus:	2 Point

Show your Course Completion Badge to demonstrate that you have completed all five levels in this course: https://www.codeschool.com/courses/shaping-up-with-angular-js

If you hand in via mail you can get the additional "attendance point" if your score is **four** or above. As usual, if it's not, you should probably have attended the class to get help ;-)

When to hand in

If you hand in via mail send a mail as described below no later than **Saturday**, October 15. **24.00**

Note that you have 1-2 extra days, if needed for this CA (Thursday + Saturday)

If you attend the class, you can demo your solution up until **12.00**

How to hand in:

Either demonstrate what you did in the class, or send a mail to iwantstudypoints@gmail.com including the following:

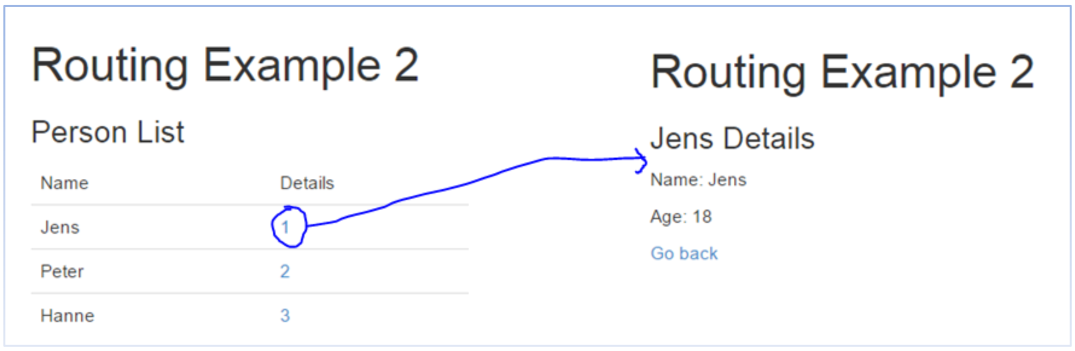
A CLEAR description of how to verify and test the code handed in. This description must also explain which part of the exercises that are implemented.

Topic: Study Point Exercise-6

Content:
First line should be your full name,
Next line: the link to your Git-hub repository for ex-1
Next line: the link to your Git-hub repository for ex-2
Next line: the link to your Git-hub repository for ex-3
Next line: the link to your Git-hub repository for ex-4
Next line: the link to your Git-hub repository for ex-5

Problem 1 Routing - passing in parameters via Route Urls

In this exercise you should create two routes and their corresponding templates as sketched below:



Name	Details
Jens	1
Peter	2
Hanne	3

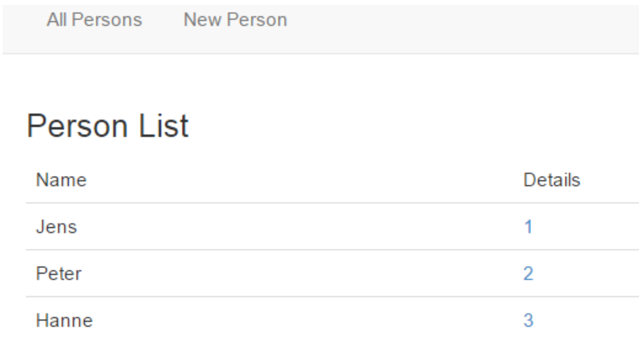
The first, and default template, should show a table with users, built with `ng-repeat` from a simple array in the Controller. The second column should be a link, which when pressed, should navigate to the second template with "details" for the person.

Complete the exercise following these steps:

1. Create a new project and add the necessary infrastructure to support angular and angular routing (don't forget angular-route: http://fall2015.azurewebsites.net/angularRouting/angularRouting.html#8)
2. Add an index.html to provide the starting point for the exercise, including a div with an `ng-view` directive to hold the partials.
3. Create the app-module for the exercise
4. Create a controller to supply the necessary data.
   Just use hardcoded values for this exercise, as sketched below:

```
var persons = [
   {id: 1,name: "Jens",age : 18}
  ,{id: 2,name: "Peter",age : 23}
  ,{id: 3,name: "Hanne",age : 23}
];
```

5. Create the routeprovider code to navigate the views[1].
6. Create the partials (templates) for the two routes
7. Provide the index.html file with a menu as sketched to the right:
   a) When "All persons" is pressed, it should show the list as before.
   b) When "New Person" is pressed it should show a Form that allows us to create new persons[2].



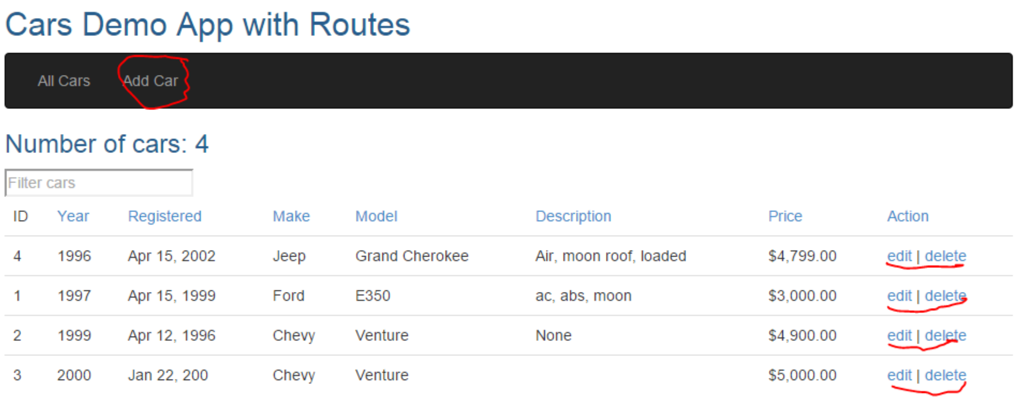
Name	Details
Jens	1
Peter	2
Hanne	3

[1] You need to use the parameter notation in your "when's" (/:id) and you need to pass in `$routeParams` to your controller
[2] Remember `ng-click` allows you to bind, for example a buttons click event, to a function in your controller.

Problem 2 Controllers and Routing

If you have not already done so, now is the time to complete the exercise from Angular-day2. This provided the full CRUD API to the simple list used for day-1's start exercise and should look similar to this: https://sem3exercises.herokuapp.com/carsCRUD/index.html

In this exercise you should change this application, where everything was available in a single view, into something like this:


Cars Demo App with Routes

All Cars Add Car

Number of cars: 4

Filter cars

ID	Year	Registered	Make	Model	Description	Price	Action
4	1996	Apr 15, 2002	Jeep	Grand Cherokee	Air, moon roof, loaded	$4,799.00	edit | delete
1	1997	Apr 15, 1999	Ford	E350	ac, abs, moon	$3,000.00	edit | delete
2	1999	Apr 12, 1996	Chevy	Venture	None	$4,900.00	edit | delete
3	2000	Jan 22, 200	Chevy	Venture		$5,000.00	edit | delete

- When "Add Car" or edit for a single car is pressed the View should change to a form where we can Add an new or Edit an existing car similar to the previous version of the exercise.
- You should have (at least) two controllers one for the "all Cars View" and one for The Add/Edit view.
- Use the factory below (we will come back to factories and services next week) to share data between the two Controllers.

```
carApp.factory('CarFactory', function () {
var cars =  [
 { id: 1, year: 1997,registered: new Date(1999,3,15), make: 'Ford',model: 'E350', description: 'ac, abs, moon', price: 3000 }
,{ id: 2, year: 1999,registered: new Date(1996,3,12), make: 'Chevy', model: 'Venture', description: 'None', price: 4900 }
,{ id: 3, year: 2000,registered: new Date(199,12,22), make: 'Chevy', model: 'Venture', description: '', price: 5000 }
,{ id: 4, year: 1996,registered: new Date(2002,3,15), make: 'Jeep', model: 'Grand Cherokee',description: 'Moon roof',price: 4799 }]
  var nextId = 5;

  var getCars = function () {return cars;}
  var deleteCar = function (id) {
    for (var i = 0; i < cars.length; i++) {
      if (cars[i].id === id) {
        cars.splice(i, 1);
        return;
      }
    }
  }
  var addEditCar = function(newcar){
    if (newcar.id == null) {
      newcar.id = nextId++;
      cars.push(newcar);
    }
    else {
      for (var i = 0; i < cars.length; i++) {
        if (cars[i].id === newcar.id) {
          cars[i] = newcar;
          break;
        }
      }
    }
  }
  return {
    getCars: getCars,
    deleteCar: deleteCar,
    addEditCar: addEditCar
  };
});
```

You can inject you factory into the controller like this (leave out $scope if you are using the Controller-as Syntax):

```
carApp.controller('ViewCarController', ['$scope', "CarFactory", function ($scope, CarFactory) {
```

Problem 3 Connecting our Car-app to a REST-backend

In this exercise you must rewrite the controller from exercise three to fetch data from a backend via a REST API.

- Implement a simple REST API, using JAX-RS which can add, edit, delete and fetch cars.
- Use the factory given below as a substitute for the factory used in exercise 3.

```
carApp.factory('CarFactory', function () {

  var getCarss = function () {} //Return Cars from the server
  var deleteCar = function (id) {};//Delete Car on the Server
  var addCar = function(newcar){};//Add Car on the Server
  var editCar = function(car){}//Edit Car on the Server;
  return {
    getCars: getCars,
    deleteCar: deleteCar,
    addCar: addCar,
    editCar: editCar
  };
});
```

Hints:

A small script to give you a few cars in the database:

```
insert into car(id,model_year,registered,make,model,description,price) values(null,1996,'1999-4-11','Jeep','Grand Cherokee','Air, loaded',4799);
insert into car(id,model_year,registered,make,model,description,price) values(null,2002,'2002-4-25','Ford','E350','ac, abs, moon',3000);
insert into car(id,model_year,registered,make,model,description,price) values(null,2005,'2005-4-25','Chevy','Venture','none',7600);
```

Gson and dates:

If you have date problems with Gson when de-serializing your car, you should create your gson instance as below:

```
private static final Gson gson = new GsonBuilder().
          setDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'").
          setPrettyPrinting().create();
```

Problem 4, JQuery versus Angularjs

exam-preparation_JqueryVsAngular.pdf

Problem 5 Angular, Controllers and Routing

Exam-preparation_Angular1.pdf