**Guida passo passo alla risoluzione dell'esercizio 1 – laboratorio 4 (AngularJS)**

Esercizio 1 (guidato): Partendo dallo starter-kit disponibile all'indirizzo https://github.com/SoNet-2017/starter-kit.git migrare l'esercitazione n.4 in AngularJS.
La struttura dati deve essere acquisita tramite una chiamata http che restituirà i dati in formato JSON

**ISTRUZIONI**

1. Cloniamo il repository starter-kit da https://github.com/SoNet-2017/starter-kit (o scarica lo zip ed estrailo in una cartella a tuo piacimento

2. Apriamo il progetto con WebStorm

3. A partire dallo starter-kit noi vogliamo realizzare una single page application, cioè vogliamo caricare differenti view nella stessa pagina.
   Per farlo, come spiegato a lezione, dobbiamo utilizzare il meccanismo di routing.
   AngularJS supporta il routing tra viste tramite il servizio $route

   Guardando bene lo starter-kit possiamo notare che un meccanismo di routing è già implementato e gestisce 2 differenti view: view1 e view2.
   Capiamo quindi quali sono le componenti essenziali per implementare il routing in AngularJS:

   3a. Nella pagina index.html è presente una chiamata allo script angular-route.js
   Il file angular-route.js definisce il modulo Angular ngRoute che mette a disposizione tutto l'occorrente per gestire il routing, quindi, andando in app.js controlliamo che sia stata dichiarata la dipendenza da questo modulo: controlliamo cioè che nella dichiarazione del modulo vi sia il modulo 'ngRoute' tra le dipendenze (nelle parentesi quadre)

   Esempio:

   angular.module("myApp", ["ngRoute"]);

   3b. controlliamo che nel modulo principale (app.js) sia configurato il servizio di routing.
   Aprendo app.js dovremmo quindi trovare qualcosa di questo tipo:

```
angular.module("myApp",    ["ngRoute"])
.config(function($routeProvider){
   $routeProvider.when("/utenti",{...})
            .when("/utenti/:userId",{...})
            .otherwise({redirectTo:"/utenti"});
});
```

   Nell'esempio possiamo notare l'oggetto $routeProvider: esso consente di specificare tramite il metodo when() la mappatura tra un URL relativo e un oggetto di configurazione. Questa mappatura è appunto chiamata route.
   Il metodo otherwise() consente di gestire gli eventuali URL per i quali non è stata definita una mappatura. Questo consente di catturare situazioni in cui l'utente inserisce un URL errato o, come nel caso dello starter-kit, di definire il comportamento di default.

4. Una volta che abbiamo definito come vanno mappati gli URL del nostro routing occorre dire ad Angular cosa deve essere visualizzato in ogni view.
   Modificando il file index.html andiamo a definire il template di base, cioè la grafica che farà da base ad ogni nostra vista.
   Se ricordate, nell'esercitazione n. 4 abbiamo già fatto qualcosa di simile: avevamo la index da cui abbiamo tolto gli elementi li.
   Usiamo lo stesso contenuto anche in questo caso: copiamo il contenuto della index della vecchia esercitazione nella nuova.

```
<body ng-app="myApp">
<header class="navbar navbar-default navbar-fixed-top">
 <div class="container-fluid">
  <div class="navbar-header">
   <h1 class="navbar-brand">Pizza++</h1>
  </div>
 </div>
</header>

<div class="container-fluid">
 <div class="row">
  <div class="col-xs-12">
   <ul class="media-list">

   </ul>
  </div>
 </div>
</div>

<footer class="navbar navbar-default navbar-fixed-bottom">
 <div class="container-fluid">
  <ul class="nav navbar-nav navbar-center">
   <li class="active"><a href="#"><span class="glyphicon glyphicon-home"></span></a></li>
   <li><a href="#" data-toggle="modal" data-target="#myModal"><span class="glyphicon glyphicon-
plus"></span></a></li>
   <li><a href="#"><span class="glyphicon glyphicon-user"></span></a></li>
  </ul>
 </div>
</footer>

</body>
```

5. ATTENZIONE: ho volutamente rimosso l'header e il modal (quello che si apriva quando cliccavo sul +)
   Tuttavia è indispensabile capire cosa va inserito nell'header!
   E' facile: bisogna importare tutti i file js che ci sono nel nostro progetto!

   Nell'esempio fatto in aula abbiamo importato questi (sia quelli del modulo principale (app.js), sia tutti quelli dei moduli e delle componenti secondarie (view1.js, …, version.js,…):

```
<script src="../lib/angular/angular.min.js"></script>
<script src="../lib/angular/angular-route.min.js"></script>
<script src="../lib/angular/angular-resource.min.js"></script>
<script src="../lib/bootstrap/ui-bootstrap-tpls-2.5.0.min.js"></script>
<script src="app.js"></script>
<script src="view1/view1.js"></script>
<script src="view2/view2.js"></script>
<script src="components/version/version.js"></script>
<script src="components/version/version-directive.js"></script>
```

```
<script src="components/version/interpolate-filter.js"></script>
```

6. Ora bisogna dire ad AngularJS in quale punto della pagina vogliamo visualizzare le viste che verranno caricate di volta in volta dinamicamente.
   A questo scopo utilizziamo la direttiva *ng-view*.
   Apriamo quindi index.html e aggiungiamo la direttiva ng-view a *<ul class="media-list">.*
   Qui dentro verrà "stampato" l'elenco delle pizze.

7. La parte che ancora manca per avere lo stesso output ottenuto nell'esercitazione n.4 è il file css, Importiamo, quindi, il foglio di stile definito nell'esercitazione n.4: copiamo il file custom.css nella cartella view1 (possiamo copiarlo anche nella cartella principale, basta poi importarlo dal giusto url). Dopo di che apriamo il file app.css e in alto scriviamo <<@import "pizzaView/custom.css"; >> per importare il file custom.css

8. A questo punto se proviamo a lanciare l'app vedremo il template (con scritta Pizza++ e pulsanti in basso) ma senza la lista delle pizze.

9. Ora è arrivato il momento di modificare view1: la view in cui verrà stampata la lista.

10. Rinominiamo la view1 in pizzaView e poi eliminiamo la view2 che non ci serve (ATTENZIONE: controllare che dopo aver rinominato ed eliminato i file le modifiche siano state apportate sia in index.html (togliamo gli script che non ci sono più) che in app.js (togliamo le dipendenze dai moduli non più esistenti))

11. Modifichiamo il template di pizzaView

    copiando il codice con li ecc dall'ultima esercitazione:

```
<ul class="media-list">
  <li class="media">
    <a class="plain-link" href="#">
      <div class="media-left">
        <img class="media-object" src="url_immagine/immagine.png" alt="alt_immagine">
      </div>
      <div class="media-body">
        <h4 class="media-heading"><strong>Nome pizza</strong></h4>
        <p>Nome Pizzeria</p>
      </div>
    </a>
    <hr/>
  </li>
</ul>
```

ATTENZIONE: Nel codice sopra non c'è ancora niente di dinamico, ma a questo pensiamo dopo

PASSIAMO ALL'IMPLEMENTAZIONE DELLA LOGICA

12. Apriamo pizzaView.js

13. Guardiamo il codice

    - In config è definita la route principale: quando viene richiamato l'url con pizzaView lui apre il

template che abbiamo appena finito di modificare

- Poi è definito un controller che rinominiamo in pizzaViewCtrl e che, per ora non fa nulla. A noi serve che questo controllore prenda i dati sulle pizze e crei l'elenco puntato

COMINCIAMO quindi dalla definizione della struttura dati

14. Creiamo una cartella "data" nella cartella principale

15. All'interno della cartella appena creata creiamo un file "pizzas.json"

16. Copiamo la struttura dati elenco_pizze dal vecchio gestionePizze.js in "pizzas.json"
    [{"nome_pizza":"Diavola", …},{...},...]

17. Abbiamo bisogno di qualcosa che vada a prendere i dati e ce li passi.
    Questo qualcosa potrebbe essere un servizio (che può essere definito con i comandi service o factory).
    Per le differenze tra service e factory rimandiamo alla documentazione ufficiale di AngularJS, noi qui usiamo il metodo factory.
    Il servizio lo creiamo in un nuovo modulo così vediamo anche come fare ad usare servizi definiti in altri moduli

18. In components creiamo una cartella "pizza".

19. Come specificato nelle "best practices" (https://docs.google.com/document/d/1XXMvReO8-Awi1EZXAXS4PzDzdNvV6pGcuaF4Q9821Es/pub) è bene separare la definizione del modulo dalla definizione dei servizi, pertanto creeremo 2 script separati (quello principale/padre e poi uno secondario)

20. Creiamo lo script pizza.js

21. Definiamo il modulo myApp.pizza

```
'use strict';

angular.module('myApp.pizza', []);
```

22. Creiamo un nuovo script "pizza-service.js"

23. Definiamo il modulo myApp.pizza.service

```
'use strict'; angular.module('myApp.pizza.service',

[])
.factory('Pizza', function($http) { var
  pizzaService = {
    getData: function () {
      return $http.get('../data/pizzas.json').then(function (response) { return
        response.data;
      });
    }
  }
```

```
    return pizzaService;
})
```

24. Andiamo in pizza.js (creato al punto 20) e definiamo la dipendenza da pizza-service

```
'use strict';

angular.module('myApp.pizza', [
  'myApp.pizza.service'
]);
```

25. Poi andiamo in pizzaView.js ed aggiungiamo la dipendenza dal modulo pizza

```
angular.module('myApp.pizzaView',    ['ngRoute','myApp.pizza'])
```

26. aggiungiamo pizzza.js e pizza-service.js alla pagina index.html

27. definiamo il controller nell'inline array annotation che troviamo tra parentesi quadre.

    Nell'inline array annotation si definisce

    - lo scope (che consente la definizione del modello dei dati e la sua esposizione alla view)
    - le dipendenze
    - la funzione da eseguire

```
.controller('pizzaViewCtrl', ['$scope', 'Pizza',
  function($scope, Pizza) {
            Pizza.getData().then(function(data)   {
              $scope.dati = {};
        $scope.dati.pizzas = data;
    });
  }]);
```

    Nella funzione diciamo di assegnare alla variabile pizzas il contenuto di data (ottenuto con una chiamata http)

28. Adesso abbiamo i dati disponibili nello scope quindi bisogna fare in modo che vengano stampati

29. Apriamo il file pizzaView e ci inseriamo questo codice:

```
<ul class="media-list">
  <li class="media" ng-repeat="pizza in dati.pizzas">
    <a class="plain-link" href="#!/pizzas/{{pizza.id}}">
      <div class="media-left">
        <img class="media-object" ng-src="../{{pizza.img_url}}" alt="{{pizza.img_alt}}">
      </div>
      <div class="media-body">
        <h4     class="media-heading"><strong>{{pizza.nome_pizza}}</strong></h4>
        <p>{{pizza.nome_pizzeria}}</p>
      </div>
    </a>
    <hr/>
  </li>
</ul>
```

30. Dopo di che eseguiamo l'app e vedremo la nostra bella lista stampata al centro dell'app.

ATTENZIONE: Rispetto alla soluzione sviluppata passo passo in classe ho aggiunto

*<a class="plain-link" href="#!/detailsPizza/{{pizza.id}}">*

Cosa è? Vi ho preparato il link per svolgere l'esercizio 2. Quindi adesso dovete aggiungere una nuova vista (ad esempio pizzas) che, nel controller prenderà la variabile pizza.id (passata nell'URL) e stamperà i dettagli della pizza.

PS. Sicuramente non verranno visualizzate le immagini: il motivo è che bisogna importarle nella cartella images

**SITI UTILI**

- Guida base su AngularJS: http://www.html.it/guide/guida-angularjs/
- Video guida base su AngularJS: https://www.youtube.com/watch?v=i9MHigUZKEM