EXERCISE GUIDE

ANGULAR WORKSHOP

v1.0.e

TABLE OF CONTENTS

This page intentionally left blank

Exercise 1: Setting Up a New Project in VS Code

Objectives

You'll create your first web development project in **Visual Studio Code** (**VS Code** for short) using a template.

Tags: #vscode #npm

Steps

1. Create a new project as copy of the provided template. Follow the steps below:
   a. Open **File Explorer** on the **Training Resources** folder (e.g. `C:\Training\AngularWorkshop`). Change into the `projects` subfolder.
   b. Create a copy of the `template` folder and rename it "`worldapp`".
   c. To open the `worldapp` project in **VS Code** do either of the following:
      - If VS Code is not yet open you can, from File Explorer, right-click on the `worldapp` folder and select "Open with Code".
      - If VS Code is open you select "File" > "Open Folder…" from the menu.
2. Open the `package.json` file in the **VS Code Editor** and review its content.

   It contains *meta information* about your project and lists the *dependencies* (initially *Bootstrap* and *jQuery*) for your project.

   Note that it also includes a "start" *script* to run the app on the server.

   Optionally, you can change the *name* to "worldapp" and the *description* to something like "This app displays information about countries in the world.".
3. If not already done, open the **VS Code Terminal** view by selecting "View" > "Terminal" from the menu.

   In the Terminal view run the following command to install the required libraries into your project:

   ```
   > npm install
   ```

   Review the content of the newly added `node_modules` subfolder. Notice that the libraries for *Bootstrap* and *jQuery* have been installed.
4. Open the `index.html` file in the **VS Code Editor** and review its content.

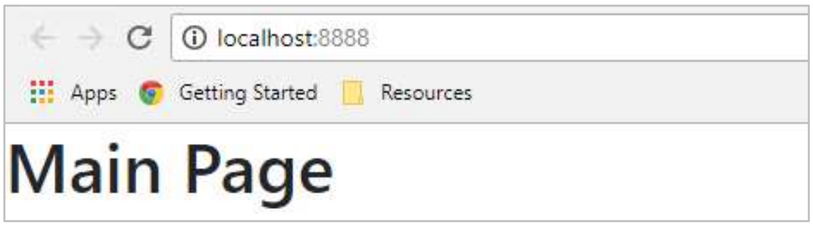
   It imports the dependencies (*jQuery* and *Bootstrap*) from the `node_modules` folder using `<link>` and `<script>` tags.

5. Finally, to start **http-server** and run your application *worldapp* on it perform the following command from the **Terminal** view:

```
> npm start
```

Notice that **http-server** will now run on localhost:8888 in the Terminal view. You can use the keyboard shortcut `CTRL-C` in this view to stop the server.

Open now **Chrome** on http://localhost:8888 to test your page:



Main Page

Exercise 2:
Displaying Static Data in a List View

Objectives

You will structure the main page of the application. You'll then use an HTML **table** to display static data and use Bootstrap for styling.

Tags: #html5 #semantic #bootstrap #table

Steps

1. Locate the `images` folder inside of the `resources` subfolder of the **Training Resources** folder.

   Now copy the `images` folder into your `worldapp` project folder.

   > ***Note**: The images folder comes with flags of all countries in the world. The images are provided in two sizes:* **flags-mini** *(height=20px and varying width=16px...51px) and* **flags-normal** *(width=550px and varying height=216px...672px).*
   > *(Source: flagpedia.net)*

2. Use semantic HTML wrappers to provide a basic structure for your page. Follow the steps below:

   a. Open the files `index.html` and `app.css` in the **VS Code Editor**.

   b. For a basic page structure, in `index.html`, use `<header>`, `<main>` and `<footer>` sections inside of the page's `<body>`. Place the text "The World App" in the `<header>` and an attribution for the used flag images in the `<footer>`:

```
<body>
  <header class="header">
    <h1>The World App</h1>
  </header>
  <main>
    <!-- Main content comes here -->
  </main>
  <footer class="footer">
    Flags of all countries by <a href="http://flagpedia.net/">flagpedia.net</a>
  </footer>
</body>
```

   c. Define the two used CSS classes `.header` and `.footer` in the project's `app.css` file. This is basically to pin the footer to the bottom of the page.

```
.header {
    width: 100%;
}

.footer {
    position: fixed;
    bottom: 0;
    width: 100%;
    background-color: #E9ECEF;
    font-size: smaller;
}
```

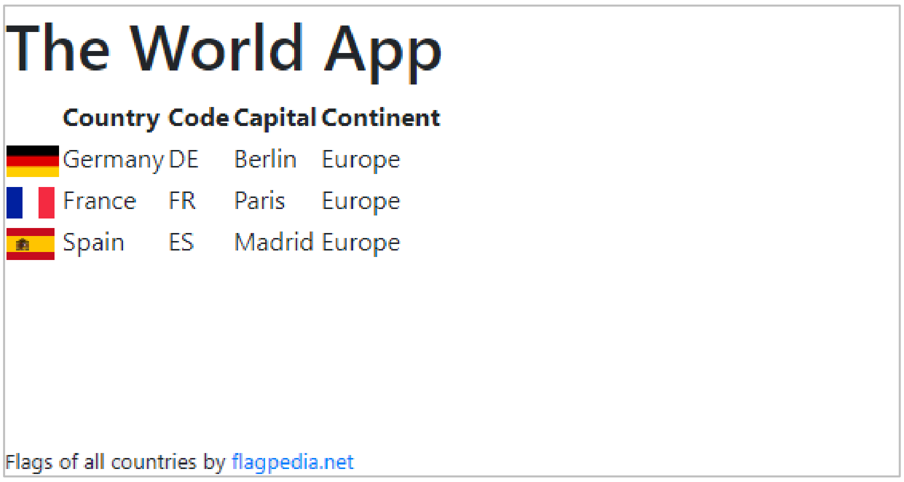
   d. Place an HTML table in the `<main>` section of the page to display information about countries.

   Initially, use just hard-coded data for three countries: Germany, France, and Spain. Let the table consist of 5 columns: *Flag* (omit column header), *Country*, *Code*, *Capital*, *Continent*. Display country flag images from `flags-mini` in the Flag column.

```
<table>
  <thead>
    <tr>
      <th></th>
      <th>Country</th>
      <th>Code</th>
      <th>Capital</th>
      <th>Continent</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td><img src="images/flags-mini/de.png" alt="DE"></td>
      <td>Germany</td>
      <td>DE</td>
      <td>Berlin</td>
      <td>Europe</td>
    </tr>
    <tr>
      <td><img src="images/flags-mini/fr.png" alt="FR"></td>
      <td>France</td>
      <td>FR</td>
      <td>Paris</td>
      <td>Europe</td>
    </tr>
    <tr>
      <td><img src="images/flags-mini/es.png" alt="ES"></td>
      <td>Spain</td>
      <td>ES</td>
      <td>Madrid</td>
      <td>Europe</td>
    </tr>
  </tbody>
</table>
```

3. Test your page in **Chrome**.

   If you stopped **http-server** in the meantime, run again the command `> npm start` from VS Code's **Terminal** view.



The World App

	Country	Code	Capital	Continent
🇩🇪	Germany	DE	Berlin	Europe
🇫🇷	France	FR	Paris	Europe
🇪🇸	Spain	ES	Madrid	Europe

Flags of all countries by flagpedia.net

***Note**: To disable caching in Chrome while testing make always sure **DevTools** is open (press F12).*

4. Finally, apply some **Bootstrap** styles to the table. Follow the steps below:

    a. Back to **VS Code**, assign the `.table` style to the `<table>` element. Check in **Chrome** the effect of applying this single style!

    Additionally assign the `.table-sm` (small table) style to make the table more compact.

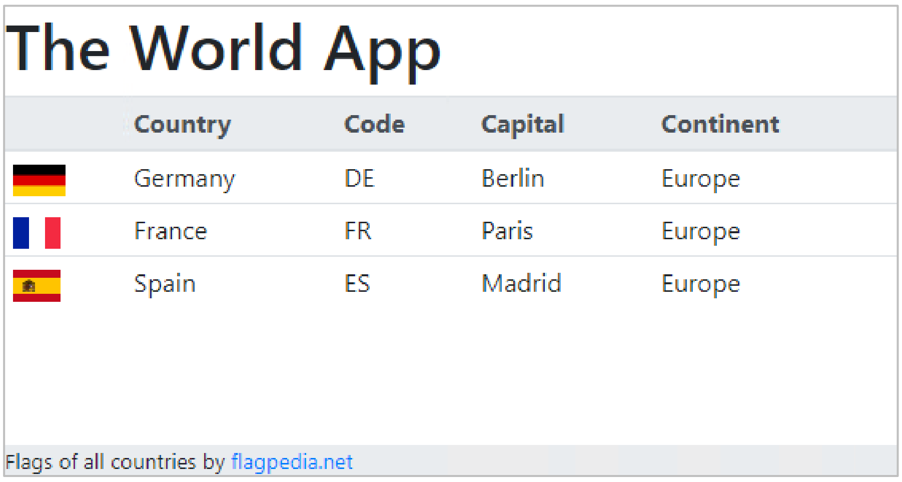
    ```
    <table class="table table-sm">
    ```

    (*Optional*) Experiment with additional table styles: `.table-sm`, `.table-bordered`, `.table-striped`, `.table-dark`, `.table-light`.

    b. Assign the `.head-light` style to the `<thead>` element to make the table head appear light gray.

    ```
    <thead class="thead-light">
    ```

5. Test your page in **Chrome**. Make sure **DevTools** is open (press F12).



The World App

	Country	Code	Capital	Continent
	Germany	DE	Berlin	Europe
	France	FR	Paris	Europe
	Spain	ES	Madrid	Europe

Flags of all countries by flagpedia.net

6. Check the Bootstrap documentation for more options with tables: https://getbootstrap.com/docs/4.1/content/tables/

This page intentionally left blank

Exercise 3: Displaying Static Data in a Cards View

Objectives

You will create a second page "cards.html" with an alternative way to display the static data. This time you'll use Bootstrap **cards** instead of the table. From here on let's call `index.html` the *list* view and `cards.html` the *cards* view of the application.

Tags: #bootstrap #cards

Steps

1. Create a copy of the `index.html` file and name it "cards.html".

   ***Note**: You can do this using the Explorer inside of **VS Code**.*

2. To create an alternative view of the data using Bootstrap "cards", follow the steps below:

   a. Open `cards.html` in the **VS Code** editor and remove the whole content (`<table>`) of the `<main>` section.

   b. In **Chrome**, open the Bootstrap documentation for cards: https://getbootstrap.com/docs/4.1/components/card/

      Scroll down to the section titled "Card decks". Find there sample HTML code to display three cards side-by-side. Copy this code into the clipboard.

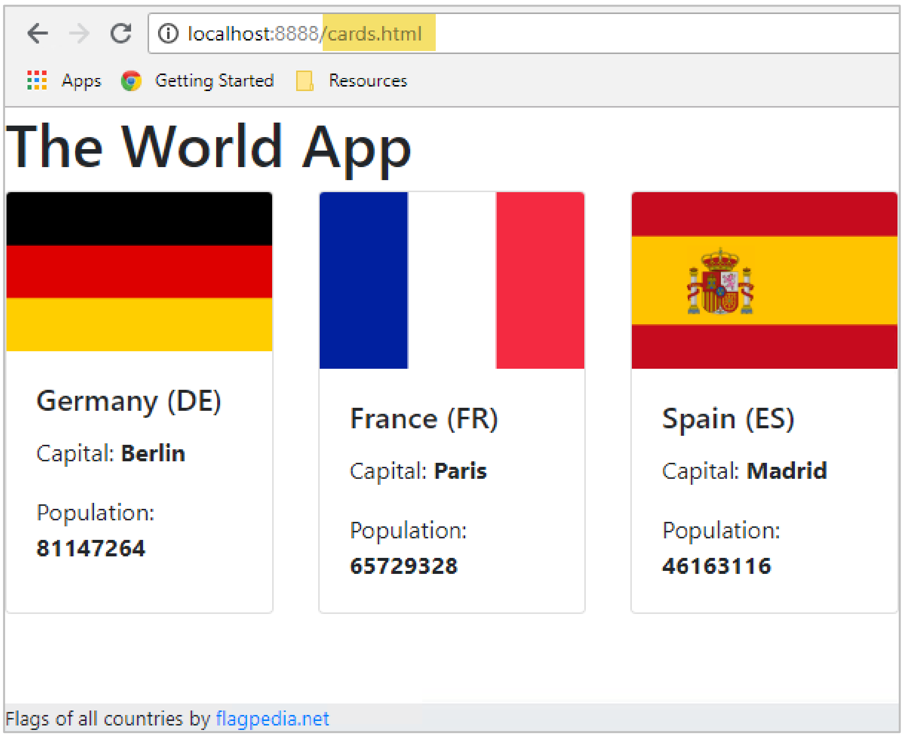
   c. Back to **VS Code**, paste the copied code into the (empty) `<main>` section.

      Adapt the copied sample code to display (hardcoded) data about the three countries we used in the previous exercise (Germany, France, and Spain).

      Display country flag images from `flags-normal`.

```
<div class="card-deck">
  <div class="card">
    <img class="card-img-top" src="images/flags-normal/de.png" alt="DE">
    <div class="card-body">
      <h5 class="card-title">Germany (DE)</h5>
      <p class="card-text">Capital: <strong>Berlin</strong></p>
      <p class="card-text">Population: <strong>81147264</strong></p>
    </div>
  </div>
  <div class="card">
    <img class="card-img-top" src="images/flags-normal/fr.png" alt="FR">
    <div class="card-body">
      <h5 class="card-title">France (FR)</h5>
      <p class="card-text">Capital: <strong>Paris</strong></p>
      <p class="card-text">Population: <strong>65729328</strong></p>
    </div>
  </div>
  <div class="card">
    <img class="card-img-top" src="images/flags-normal/es.png" alt="ES">
    <div class="card-body">
      <h5 class="card-title">Spain (ES)</h5>
      <p class="card-text">Capital: <strong>Madrid</strong></p>
      <p class="card-text">Population: <strong>46163116</strong></p>
    </div>
  </div>
</div>
```

3. Test the *cards* view in **Chrome**. Make sure **DevTools** is open (press F12).



localhost:8888/cards.html

Apps Getting Started Resources

The World App

Germany (DE)

Capital: **Berlin**

Population: **81147264**

France (FR)

Capital: **Paris**

Population: **65729328**

Spain (ES)

Capital: **Madrid**

Population: **46163116**

Flags of all countries by flagpedia.net

Exercise 4: Making Both Views Responsive

Objectives

You will make your two views index.html and cards.html responsive. While it turns out to be easy for the first one, you'll need to resort to a "flexbox" grid in the latter.

Tags: #bootstrap #responsive #gridsystem #flexbox

Steps

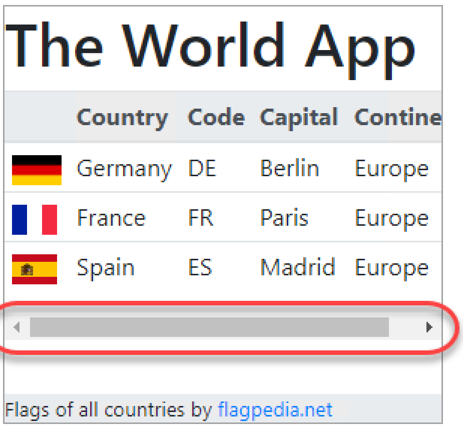
1. Start with *list* view. Making this view responsive across all viewports can easily be achieved by just wrapping the table with a `.table-responsive` container. Follow the steps below:

   a. Open `index.html` in the **VS Code Editor**.

   b. Add a new `<div>` with class `.table-responsive` as direct child of `<main>`.

   ```
   <div class="table-responsive">
     <table class="table table-sm">
       ...
     </table>
   </div>
   ```

   c. Test the *list* view in **Chrome** with narrow viewport. Make sure **DevTools** is open (press F12). If DevTools opens on the right-hand side you can easily use the separator slider to reduce the viewport width.



The World App

	Country	Code	Capital	Contine
	Germany	DE	Berlin	Europe
	France	FR	Paris	Europe
	Spain	ES	Madrid	Europe

Flags of all countries by flagpedia.net

> ***Note**: If the viewport's width gets too narrow, the scrollbar is now for the table only. Without the `.table-responsive` wrapper, the scrollbar would be for the whole viewport which would <u>not</u> be responsive.*

2. Next, get to work on *cards* view. According to the Bootstrap v4.1 documentation, "*Bootstrap includes a few options for laying out series of cards* [You used `.card-deck` in the previous exercise]. *For the time being, these layout options are not yet responsive.*"

   Therefore, you will replace the Card Deck by a flexible box (flexbox) grid now. Follow the steps below:

   a. Open `cards.html` in the **VS Code Editor**.

   b. Create a Flexbox Grid as direct child of the `<main>` section and consisting of one Fluid Container (`.container-fluid`), in turn containing one Grid Row (`.row`), and in turn containing three Grid Columns (`.col`), one for each of your Cards.

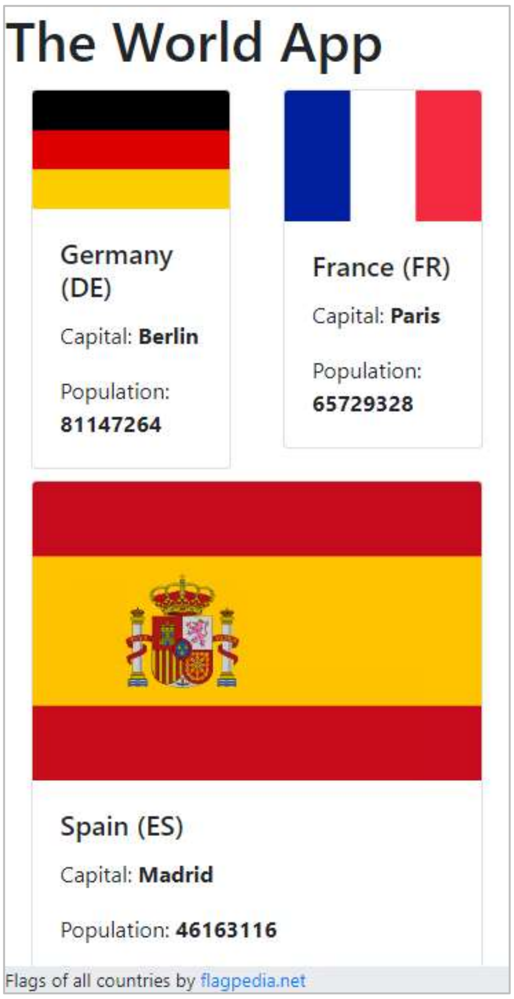
The Grid Columns additionally use `.m-1` to ensure the Cards appear with enough distance (margin) to each other.

```
<div class="container-fluid">
  <div class="row">
    <div class="col m-1">
    </div>
    <div class="col m-1">
    </div>
    <div class="col m-1">
    </div>
  </div>
</div>
```

c. Move now the three Cards (`.card`) out of the Card Deck and into the three Grid Columns.

```
<div class="col m-1">
  <div class="card">
    <img class="card-img-top" src="images/flags-normal/de.png" alt="DE">
    <div class="card-body">
      <h5 class="card-title">Germany (DE)</h5>
      <p class="card-text">Capital: <strong>Berlin</strong></p>
      <p class="card-text">Population: <strong>81147264</strong></p>
    </div>
  </div>
</div>
...
```

d. Test the *cards* view in **Chrome** with narrow viewport. Make sure **DevTools** is open (press F12). Use the separator slider to reduce the viewport width.


The World App

Germany (DE)
Capital: **Berlin**
Population: **81147264**

France (FR)
Capital: **Paris**
Population: **65729328**

Spain (ES)
Capital: **Madrid**
Population: **46163116**

Flags of all countries by flagpedia.net

Exercise 5:
First Steps into AngularJS

Objectives

You will be taking your first steps with AngularJS in this exercise. For the time being you'll see what you can do with AngularJS using just HTML5 … no JavaScript coding for now.

Tags: #angular #built-in #directives

Steps

1. First thing to do is to install Angular in your project, and to add it as new dependency. To do so, follow the steps below:

   a. In **VS Code**, open the **Terminal** view if not already open: from the menu select "View" > "Terminal".

   b. In the **Terminal** view run the following command to install Angular into your project:

   ```
   > npm install angular
   ```

   Refresh VS Code's **Explorer** and check the `node_modules` subfolder. Notice that besides *Bootstrap* and *jQuery* now also *Angular* has been installed.

   You can also check the `package.json` file. You should see a new entry under "dependencies".

   c. Open both views in the **Editor**, `index.html` and `cards.html`. In the `<head>` section of each of the two files, add a new `<script>` import for `angular.min.js`:

   ```
   <script src="node_modules/jquery/dist/jquery.min.js"></script>
   <script src="node_modules/bootstrap/dist/js/bootstrap.min.js"></script>
   <script src="node_modules/angular/angular.min.js"></script>
   <script src="app.js"></script>
   ```

   ***Note**: You must make sure that the imports are in the right order: first jQuery, then Bootstrap, then Angular, then your project's JS.*

   d. (*Optional*) You can check which version of Angular you are actually using by refreshing either of your two pages (`index.html` or `cards.html`) in **Chrome**, open **DevTools** (press F12), select the "Console" tab of DevTools and type:

   ```
   > angular.version.full
   ```

2. Next thing to do is to use the *ngApp* directive (i.e. the `ng-app` attribute) to tell AngularJS what the root element of the application is. Follow the steps below:

   a. Have both of your `.html` files open in the **VS Code Editor**.

   b. In each one, add the `ng-app` attribute to `<html>` element.

   ```
   <html lang="en" ng-app>
   ```

***Note**: It had also been correct to add the `ng-app` attribute to a different element, for example to the `<body>` element.*

3. Now focus on the *list* view. You will use the *ngInit* directive to provide the data (an array of country objects) that will be displayed in the table. Each country object has five attributes: *ISO_2* (the country's 2-letter code), *Continent*, *Country* (the country's name), *Capital*, and *Population*.

   Then you will replace the three hard-coded table `<tr>` elements by a single `<tr>` element that uses the *ngRepeat* directive and expressions. The repeater tells AngularJS to create a `<tr>` element for each item in the array.

   Follow the steps below:

   a. Using **File Explorer** locate the file `ten-countries.min.json` in the resources subfolder of the **Training Resources** folder. Open it in **Notepad++** and copy (CTRL+C) its entire content into the *clipboard*.

   b. Open `index.html` in the **VS Code Editor**. Use the `ng-init` attribute on the `<div>` wrapper of the `<table>` to initialize a "countries" variable with the copied data array.

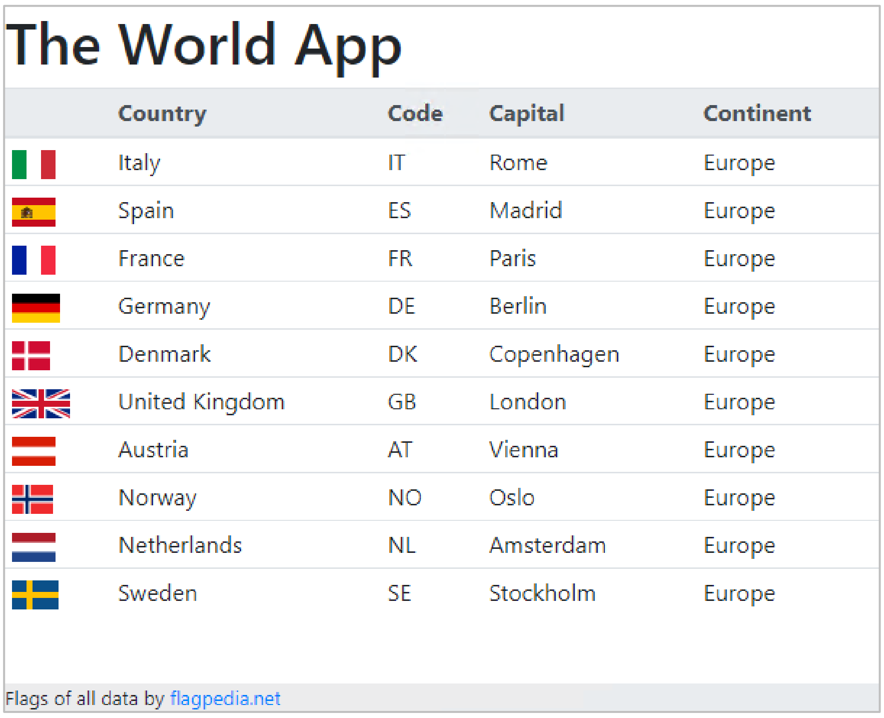
```
<div class="table-responsive"
ng-init="countries=[{'ISO_2':'IT','Continent':'Europe','Country':'Italy', ...}, ...
]">
```

   c. Next replace the three `<tr>` elements inside of `<tbody>` with a single one using `ng-repeat`. Inside of the `<td>` elements replace all hard-coded values by appropriate `{{…}}` expressions:

```
<tr ng-repeat="item in countries">
  <td><img ng-src="images/flags-mini/{{item.ISO_2}}.png" alt="{{item.ISO_2}}"></td>
  <td>{{item.Country}}</td>
  <td>{{item.ISO_2}}</td>
  <td>{{item.Capital}}</td>
  <td>{{item.Continent}}</td>
</tr>
```

   ***Note**: You should also change the `<img>` tag to use now `ng-src` instead of `src`.*

   d. Test the *list* view in **Chrome**. Make sure **DevTools** is open (press F12).



The World App

	Country	Code	Capital	Continent
	Italy	IT	Rome	Europe
	Spain	ES	Madrid	Europe
	France	FR	Paris	Europe
	Germany	DE	Berlin	Europe
	Denmark	DK	Copenhagen	Europe
	United Kingdom	GB	London	Europe
	Austria	AT	Vienna	Europe
	Norway	NO	Oslo	Europe
	Netherlands	NL	Amsterdam	Europe
	Sweden	SE	Stockholm	Europe

Flags of all data by flagpedia.net

4. Next, do something similar on the second view, the `cards.html` file. Follow the steps below:

   a. Have `cards.html` open in the **VS Code Editor**.

b. Using the `ng-init` attribute in the Grid Row, initialize a variable named "countries":

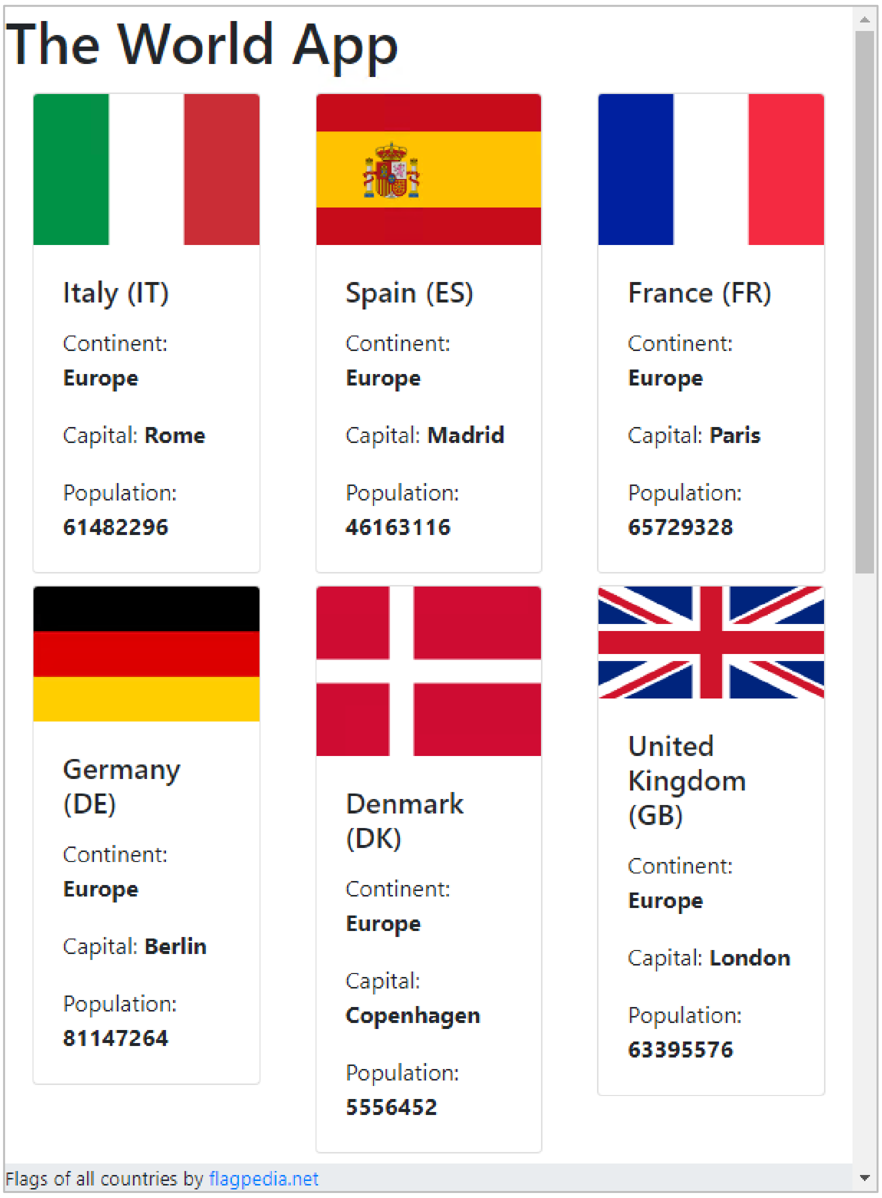
```
<div class="row" ng-init="countries=[{'ISO_2':'IT','Continent':'Europe', …}, …]">
```

c. Next, replace the <u>three</u> Grid Columns with a <u>single</u> one using `ng-repeat`. Inside of the contained Card, replace all hard-coded values with appropriate `{{…}}` expressions.

```
<div class="col m-1" ng-repeat="item in countries">
  <div class="card">
    <img class="card-img-top" ng-src="images/flags-normal/{{item.ISO_2}}.png"
      alt="{{item.ISO_2}}">
    <div class="card-body">
      <h5 class="card-title">{{item.Country}} ({{item.ISO_2}})</h5>
      <p class="card-text">Continent: <strong>{{item.Continent}}</strong></p>
      <p class="card-text">Capital: <strong>{{item.Capital}}</strong></p>
      <p class="card-text">Population: <strong>{{item.Population}}</strong></p>
    </div>
  </div>
</div>
```

***Note**: You should also change the `<img>` tag to use now `ng-src` instead of `src`.*

d. Test the *cards* view in **Chrome**. Make sure **DevTools** is open (press F12).



The World App

Italy (IT)
Continent: **Europe**
Capital: **Rome**
Population: **61482296**

Spain (ES)
Continent: **Europe**
Capital: **Madrid**
Population: **46163116**

France (FR)
Continent: **Europe**
Capital: **Paris**
Population: **65729328**

Germany (DE)
Continent: **Europe**
Capital: **Berlin**
Population: **81147264**

Denmark (DK)
Continent: **Europe**
Capital: **Copenhagen**
Population: **5556452**

United Kingdom (GB)
Continent: **Europe**
Capital: **London**
Population: **63395576**

Flags of all countries by flagpedia.net

This page intentionally left blank

Exercise 6: Sorting and Filtering the Data

Objectives

You can still do more with AngularJS using just HTML5 … JavaScript coding still not needed. In both of your views, you will sort the data by the *Country* name and filter the data by the *Capital* name.

Tags: #angular #built-in #orderby #filter

Steps

1. To allow users of the *worldapp* to search for countries where the capital name matches certain criteria, add an *input* field in both of your views. To do so, follow the steps below:

   a. Have both of your `.html` files open in the **VS Code Editor**.

   b. In each one, add an `<input>` element to the `<header>` section.

   ```
   <input class="form-control" placeholder="Search for capital..." ng-model="search">
   ```

   ***Note**: The text entered by the user in this field will be available in an implicitly defined variable "search".*

2. Now, implement <u>sorting</u> of the data by the *Country* name and <u>filtering</u> by the *Capital* name. Follow the steps below:

   a. In each of your `.html` files locate the element with the `ng-repeat` attribute.

   b. In each of these two places, append `orderBy:` and `filter:` terms as shown below:

   ```
   ng-repeat="item in countries | orderBy:'Country' | filter:{Capital:search}"
   ```

   ***Note**: Here, you are making use of two <u>built-in</u> filter components provided by AngularJS.*
   If you used `orderBy:'Country':true` instead of the above, the sorting would be in <u>reverse</u> order.
   If you used just `filter:search`, the search criteria would have applied to <u>any</u> of the attributes Country, Capital, and Continent. Equivalent notation: `filter:{$:search}`.

3. Finally, make use of one more built-in filter component which you can use in `Cards.html` to format the display of the *Population* numbers. Follow the steps below:

   a. In `cards.html`, locate the element which is using the expression `{{item.Population}}`.

   b. Append a `number:` term as shown below:

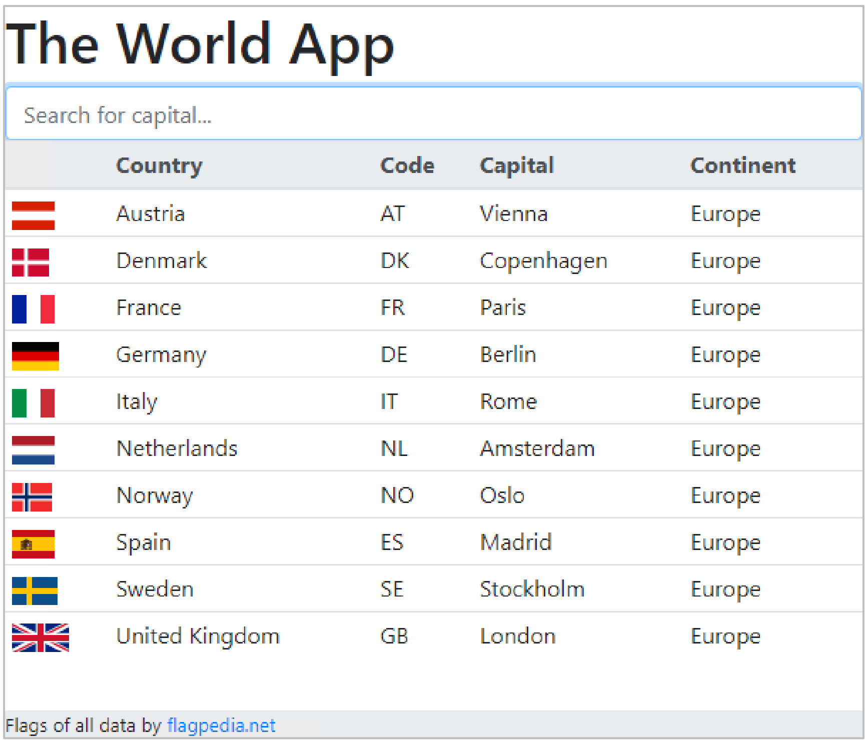
   ```
   {{item.Population | number:0}}
   ```

   ***Note**: The argument "0" specifies the number of decimal places. Then the number is appropriately formatted based on the current locale.*

4. (*Optional*) See the AngularJS documentation on built-in filter components in and find out how to display the *Continent* name in UPPER case (https://docs.angularjs.org/api/ng/filter).

5. Test both views in **Chrome**. Make sure **DevTools** is open (press F12).

   Note how the data is now sorted by *Country* name and how searching by *Capital* name works in both views. Also, the *Population* numbers in `cards.html` now include "," group separators after each third digit.

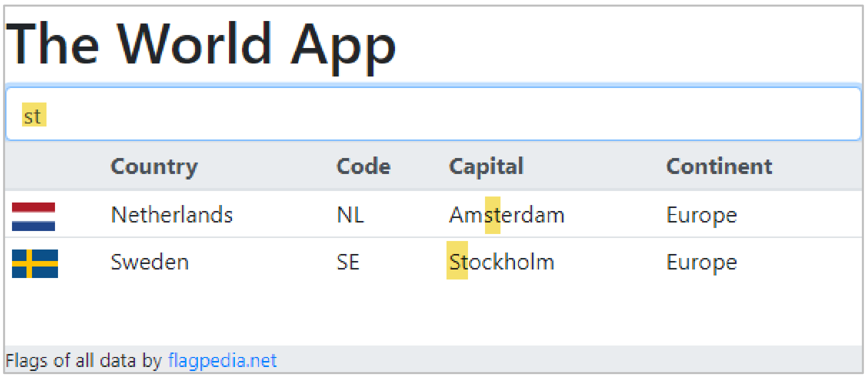


The World App

Search for capital...

	Country	Code	Capital	Continent
	Austria	AT	Vienna	Europe
	Denmark	DK	Copenhagen	Europe
	France	FR	Paris	Europe
	Germany	DE	Berlin	Europe
	Italy	IT	Rome	Europe
	Netherlands	NL	Amsterdam	Europe
	Norway	NO	Oslo	Europe
	Spain	ES	Madrid	Europe
	Sweden	SE	Stockholm	Europe
	United Kingdom	GB	London	Europe

Flags of all data by flagpedia.net

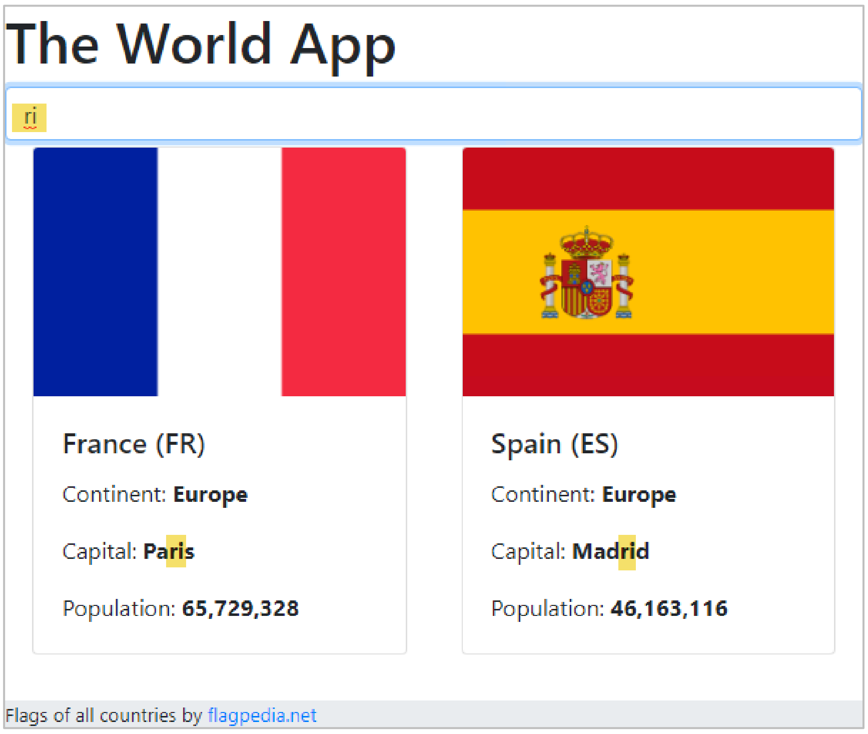


The World App

st

	Country	Code	Capital	Continent
	Netherlands	NL	Amsterdam	Europe
	Sweden	SE	Stockholm	Europe

Flags of all data by flagpedia.net



The World App

ri

France (FR)

Continent: **Europe**

Capital: **Paris**

Population: **65,729,328**

Spain (ES)

Continent: **Europe**

Capital: **Madrid**

Population: **46,163,116**

Flags of all countries by flagpedia.net

Exercise 7:
Defining the Module and a Controller

Objectives

Ok, it's time for some *JavaScript* coding now! You'll define the *Module* and a *Controller* in the project's main JavaScript file.

Tags: #angular #module #controller

Steps

1. First thing to do is to register the *Module* defining the application. The *Module* serves as a container for the different parts of the app including *Controllers*, *Services*, *Filters*, etc.

   Follow the steps below:

   a. Open the file `app.js` in the **VS Code Editor**.

   b. Use the `angular.module()` function to create and register a new *Module* named "worldApp" for your application. Define a *variable* to easy accessing the *Module* in your code:

   ```
   var worldapp = angular.module('worldApp', []);
   ```

   ***Note**: The 2nd argument can be used to define modules (as array of module names) your module depends on. For a new module you must specify the 2nd argument—even if it is an empty array—because otherwise you would <u>not</u> be creating a new module, but retrieving an existing one.*

2. Next, define a *Controller* and register it in the *Module*. To do so follow the steps below:

   a. Have the file `app.js` open in the **VS Code Editor**.

   b. Use the *Module's* `controller()` function to register a new *Controller* named "worldappCtrl":

   ```
   worldapp.controller('worldappCtrl', function($scope) {
     // Setup initial state and add behavior to the $scope object here
   });
   ```

   ***Note**: In the next step you'll attach this Controller to the `<body>` element of your view. Angular will create a new* Scope *object. The 2nd argument of the above `controller()` function is itself a function that obtains the* Scope *object as input parameter and is used to initialize data and add behavior in the Scope.*

   ***Note**: In larger projects it is good practice to define Controllers in separate `.js` files. For the sake of simplicity we stay with a single `.js` file for now.*

3. Instead of using the `ng-init` directive in the `.html` file (which is <u>not</u> good practice), you will initialize the `countries` array here in the *Controller*.

   Follow the steps below:

   a. Using **File Explorer** locate the file `world-data.min.json` in the `resources` subfolder of the **Training Resources** folder. Open it in **Notepad++** and copy (CTRL+C) its entire content (a single very long line) into the *clipboard*.

b. Back in the **VS Code Editor**, `app.js`, initialize a *scope property* "countries" with the copied (very long) data array inside of the *Controller*:

```
$scope.countries = VERY-LONG-DATA-ARRAY-COPIED-FROM-FILE;
```

> ***Note**: The array you are pasting here looks **very big**, but don't worry … it's only 32 KB! This is a temporary change anyway. Later, you'll remove the array here and get the data from a REST service call.*

4. To allow users to narrow the display of the countries to only one *continent*, initialize another *scope property* "continents". Also, add a *scope property* "selected_continent" for the continent that was selected by the user.

   Follow the steps below:

   a. In **VS Code Editor**, `app.js`, initialize a *scope property* "continents" with an array of *continent* objects, each having "label" and "value" properties. Include a selection "All Continents".

   ```
   $scope.continents = [{label:'All Continents',value:''},
   {label:'Africa',value:'Africa'}, {label:'Asia',value:'Asia'},
   {label:'Europe',value:'Europe'}, {label:'North America',value:'North America'},
   {label:'Oceania',value:'Oceania'}, {label:'South America',value:'South America'}];
   ```

   b. Add a *scope property* "selected_continent" and initialize it with the first entry ("All Continents") of the `continents` array:

   ```
   $scope.selected_continent = $scope.continents[0];
   ```

5. Now, change the two views (`index.html` and `cards.html`) to make use of the new *Controller* and to get rid of the `ng-init` directive.

   Follow the steps below:

   a. Have both of your `.html` files open in the **VS Code Editor**.

   b. In each one, refer to the *Module* "worldApp" using the `ng-app` directive:

   ```
   <html lang="en" ng-app="worldApp">
   ```

   c. Moreover, use the `ng-controller` directive to attach the Controller "worldappCtrl" to the `<body>` tag in each of the two `.html` files:

   ```
   <body ng-controller="worldappCtrl">
   ```

   d. In each of the two `.html` files remove the `ng-init` attribute. In `index.html` is on the table wrapper:

   <div class="table-responsive" ~~ng-init="countries=..."~~>

   … and in `cards.html` it is on the *grid column*:

   <div class="row" ~~ng-init="countries=..."~~>

6. Test both views in **Chrome**. Make sure **DevTools** is open (press F12). You should now see much more countries!

> ***Note**:*
> *As the name of the `countries` scope property didn't change the bindings used so far still work:*
> `ng-repeat="item in countries ..."`

7. Finally, in both `.html` views change the `<header>` section such that it contains a dropdown to allow users to select a continent. Also, in both views add a *headline* telling the continent and the number of countries currently displayed.

   Follow the steps below:

   a. In both of your `.html` files change the `<header>` section to obtain something as shown below:

```
<header class="header">
  <h1>The World App</h1>
  <select class="form-control" ng-model="selected_continent"
    ng-options="continent.label for continent in continents"></select>
  <input class="form-control" placeholder="Search for capital..."
    ng-model="search">
  <br>
  <h5>Countries in {{selected_continent.label}}
    <span class="badge badge-secondary">{{filtered.length}}</span></h5>
</header>
```

   ***Note**: The expression `{{selected_continent.label}}` in the `<h5>` headline is used to display the selected continent. The other expressions in the headline `{{filtered.length}}` is to tell the number of countries displayed (see also next substep b.).*

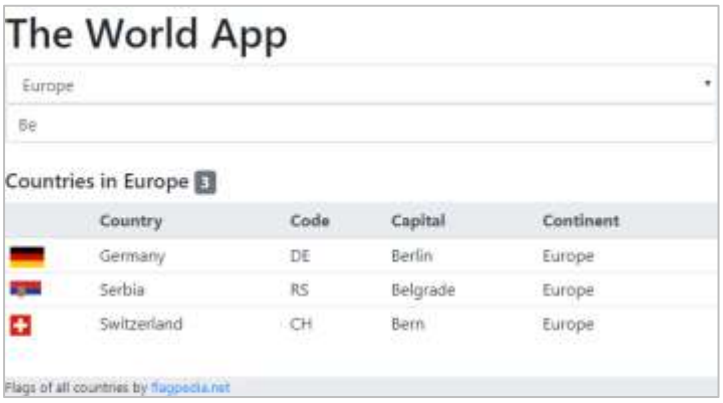
   ***Note**: The Bootstrap "Badge" component is used to display the number of countries.*

   b. In both files modify the `ng-repeat` attribute by chaining another filter expression:

```
ng-repeat="item in countries | orderBy:'Country' | filter:{Capital:search}
| filter:{Continent:selected_continent.value} as filtered"
```

   ***Note**: At the end of `ng-repeat` we used an alias expression "`as filtered`" for the final filtering result. The "filtered" alias was used in the `<header>` section to display the number of results.*

8. Test both views in **Chrome**. Make sure **DevTools** is open (press F12).



The World App

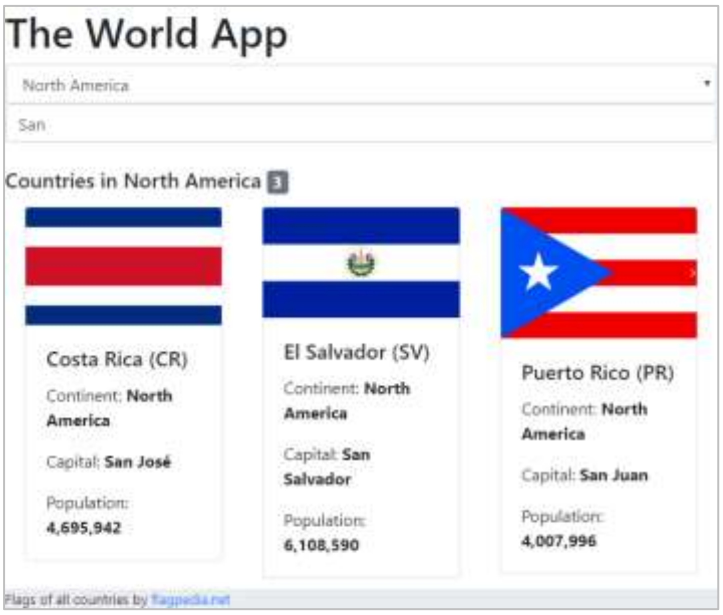
Europe

Be

Countries in Europe 3

	Country	Code	Capital	Continent
	Germany	DE	Berlin	Europe
	Serbia	RS	Belgrade	Europe
	Switzerland	CH	Bern	Europe

Flags of all countries by flagpedia.net



The World App

North America

San

Countries in North America 3

Costa Rica (CR)
Continent: **North America**
Capital: **San José**
Population: **4,695,942**

El Salvador (SV)
Continent: **North America**
Capital: **San Salvador**
Population: **6,108,590**

Puerto Rico (PR)
Continent: **North America**
Capital: **San Juan**
Population: **4,007,996**

Flags of all countries by flagpedia.net

Exercise 8:
Working with Services

Objectives

In this exercise has two parts. In **Part I**, you'll change your app such that the data now comes from a *REST service* instead of having its hard-coded initialization in the *Controller*. In **Part II**, you'll create a *custom service* to wrap the HTTP call, and so simplifying the code in the *Controller*.

Tags: #angular #built-in #custom #service #http #rest #factory #promise

Steps of Part I

1. To setup a *REST service* that returns exactly the same data as you used before you will now "cheat" a little bit ☺. Follow the steps below:

    a. Using **File Explorer**, locate the file `world-data.min.json` in the `resources` subfolder of the **Training Resources** folder and copy it into a new folder named "rest" inside of your `worldapp` project.

    b. Using **Chrome**, you can now test the URL http://localhost:8888/rest/world-data.min.json which should return the JSON data. This will be the URL of the *REST service*.

2. Next, change the `worldappCtrl` Controller by replacing the hard-coded initialization of the `countries` scope property by the *REST service* call. To invoke the *REST service* you'll use a built-in AngularJS service called `$http`.

    Follow the steps below:

    a. Open the file `app.js` in the **VS Code Editor**.

    b. Inside of the *Controller*, initialize the *scope property* `countries` now with an empty array (i.e. remove the very long list of country objects):

    ```
    $scope.countries = [];
    ```

    c. Inject the built-in `$http` service into the *Controller* by adding another input argument to the controller function:

    ```
    worldapp.controller('worldappCtrl', function($scope, $http) {
      ...
    ```

    d. Define a *scope method* "loadData" that uses the `$http` service to invoke the *REST service* and then uses the response data to initialize the `countries` *scope property*:

    ```
    $scope.loadData = function($scope, $http) {
      $http.get('/rest/world-data.min.json').then(
        function successCallback(response){
          $scope.countries = response.data;
          console.log('Success: ' + response.statusText);
        },
        function errorCallback(response) {
          console.log('Error: ' + response.statusText);
        }
      );
    };
    ```

    ***Note**: The `$http` built-in service returns a* Promise. *Its `then()` method expects one or more functions as input parameters:* successCallback, errorCalback, notifyCallback.

e. Invoke the `loadData` method at the end of the controller function to actually initialize the data:

```
$scope.loadData($scope, $http);
```

3. Test both views in **Chrome**. Make sure **DevTools** is open (press F12). Both views should still work and look the same as before. You can look for the "Success" message in *DevTools > Console*.

Steps of Part II

4. Now extract the `$http` call into a *custom service*.

   Follow the steps below:

   a. Open the file `app.js` in the **VS Code Editor**.

   b. Use the *Module's* `factory()` function to register a new *Service* named "worldappSvc":

```
worldapp.factory('worldappSvc', function ($http) {
  return {
    retrieveCountries: function () {
      return $http.get('/rest/world-data.min.json').then(
        function (response) {
          console.log('worldappSvc:Success: ' + response.statusText);
          return response.data;
        },
        function (response) {
          console.log('worldappSvc:Error: ' + response.statusText);
        }
      );
    }
  };
});
```

***Note**: Using the* `factory()` *function is the most typical of three ways in AngularJS to create a service. It creates and returns a reusable singleton object that can be shared across the app by injecting it into controllers, filters and directives.*
Your `worldappSvc` *custom service has one method* `retrieveCountries`*. This method, as* `$http` *does, returns a promise.*

   c. Inject the `worldappSvc` *custom service* (instead of the built-in `$http` service) into the *Controller* by replacing the second input parameter of the *controller function*:

```
worldapp.controller('worldappCtrl', function ($scope, worldappSvc) {
  ...
```

   d. Finally, remove the `loadData` *scope method* together with its invocation. Instead, invoke the `worldappSvc` *custom service* and use the result data to initialize the `countries` *scope property:*

```
worldappSvc.retrieveCountries().then(
  function (result) {
    $scope.countries = result;
  }
);
```

5. Again, test both views in **Chrome**. Make sure **DevTools** is open (press F12). Both views should still work and look the same as before. You can look this time for the " worldappSvc:Success " message in *DevTools > Console*.

Exercise 9: Converting into a SPA using Routing

Objectives

In this exercise you'll provide navigation between your two views or "partials" (*list* and *cards*) without the need of reloading the entire application, i.e. you will convert your application into to a SPA.

Tags: #angular #di #route #spa

Steps

1. *Routing* is provided through the AngularJS *Router Module* (called `ngRoute`) which is distributed separately from the core AngularJS framework. So, first thing to do is to install the `angular-route` package in your project. To do so, follow the steps below:

   a. In **VS Code**, open the **Terminal** view if not already open: from the menu select "View" > "Terminal".

   b. In the **Terminal** view run the following command:

   ```
   > npm install angular-route
   ```

   Refresh VS Code's **Explorer** and check the `node_modules` subfolder. Notice that besides *Bootstrap*, *jQuery* and *Angular* now also AngularJS *Router Module* has been installed.

   You can also check the `package.json` file. You should see a new entry under "dependencies".

2. Next, add the *Router Module* as new a dependency. To do so, follow the steps below:

   a. Open `index.html` in the **VS Code Editor**. In the `<head>` section, add a new `<script>` import for `angular-route.min.js`:

   ```
   <script src="node_modules/jquery/dist/jquery.min.js"></script>
   <script src="node_modules/bootstrap/dist/js/bootstrap.min.js"></script>
   <script src="node_modules/angular/angular.min.js"></script>
   <script src="node_modules/angular-route/angular-route.min.js"></script>
   <script src="app.js"></script>
   ```

   ***Note**: You must make sure that the imports are in the right order: first jQuery, then Bootstrap, then Angular, then the Router Module, and then your project's JS.*

   b. Now, open `app.js` in **VS Code Editor** and add `'ngRoute'` as module on which your `worldapp` module depends on:

   ```
   var worldapp = angular.module('worldApp', ['ngRoute']);
   ```

3. To create the two "partials" for the *list* and *cards* views, follow the steps below:

   a. Create a new file `list.html` in your project.

   b. Open `index.html` in the **VS Code Editor**. Cut the entire content of the `<main>` section (without the `<main>` tag) and paste it as new entire content of `list.html`:

   ```
   <div class="table-responsive">
     <table class="table table-sm">
       ...
     </table>
   </div>
   ```

c. Now open `cards.html` in the **VS Code Editor**. Cut the entire content of the `<main>` section (again without the `<main>` tag) and paste/replace it as new entire content into the same file `cards.html`:

```
<div class="container-fluid">
  <div class="row">
    ...
  </div>
</div>
```

4. In the `index.html` template make now use of the *ngView* directive (`ng-view` attribute) that comes with the *Router Module*. The role of the *ngView* directive is to render the "partial" for the current route into the layout template page. Follow the steps below:

   a. Open `index.html` in the **VS Code Editor**.

   b. Inside of the `<header>` section, add links to allow the users to switch between the two views, list and cards. To do so add another `<div>` right above the existing `<h5>` header line:

```
  <div class="text-right">
    <a href="#!/list" class="mx-3">List</a>
    <a href="#!/cards" class="mx-3">Cards</a>
  </div>
  <h5>Countries in ... </h5>
```

> ***Note**: The `mx-3` class is used to have some space at the left and right sides of the links. See: Bootstrap documentation (http://getbootstrap.com/docs/4.1/utilities/spacing/)*

   c. Inside of the now empty `<main>` section add a *single* and *empty* `<div>` tag with an `ng-view` attribute:

```
<main>
  <div ng-view></div>
</main>
```

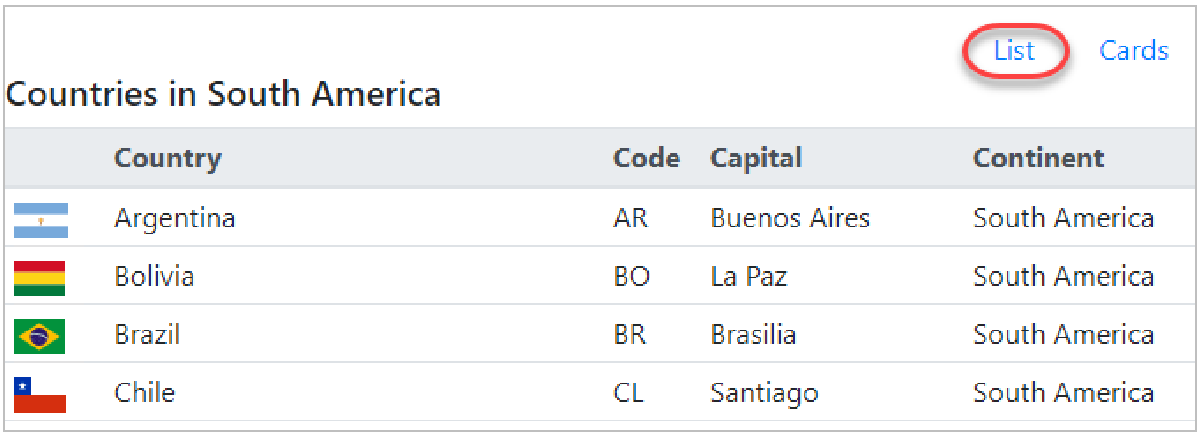
5. Last step consists in configuring the *Route Service* (`$route`). To do so we need the *Route Provider* (`$routeProvider`) which, as `$route`, comes with the *ngRoute* module. Follow the steps below:

   a. Verify that `'ngRoute'` has been added as dependency to your `worldapp` module (you should have done so in step #2).

   b. Open `app.js` in **VS Code Editor** and use the `config()` method of your `worldapp` module to add the following configuration:

```
worldapp.config(function($routeProvider) {
  $routeProvider
    .when('/list', {
      templateUrl: 'list.html'
    })
    .when('/cards', {
      templateUrl: 'cards.html'
    })
    .otherwise('/list');
});
```

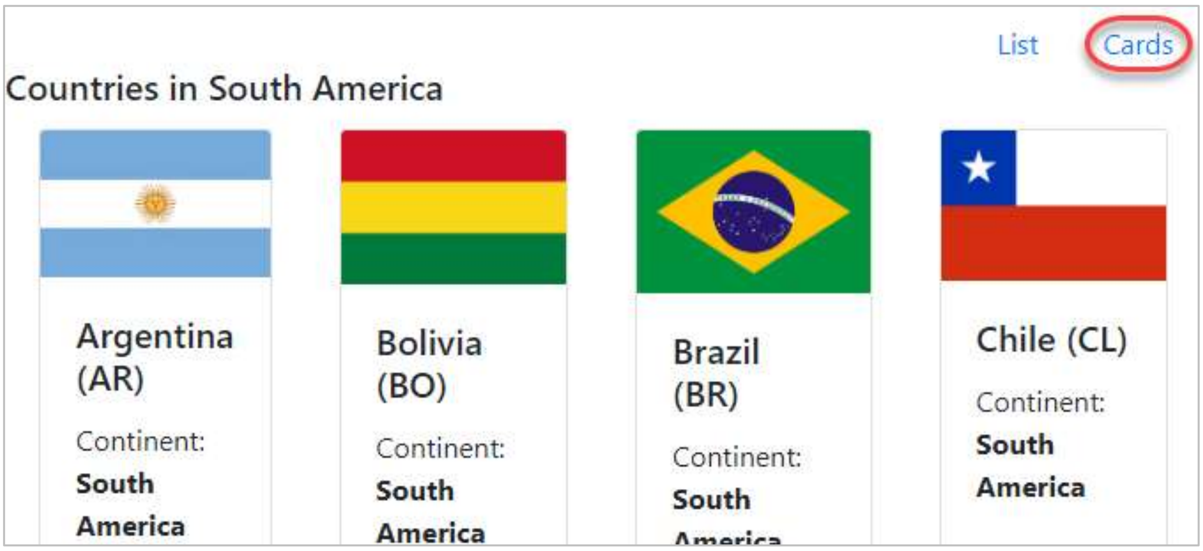
6. Test your application in **Chrome**. Make sure **DevTools** is open (press F12).



List Cards

Countries in South America

	Country	Code	Capital	Continent
	Argentina	AR	Buenos Aires	South America
	Bolivia	BO	La Paz	South America
	Brazil	BR	Brasilia	South America
	Chile	CL	Santiago	South America



List Cards

Countries in South America

Argentina (AR)
Continent: **South America**

Bolivia (BO)
Continent: **South America**

Brazil (BR)
Continent: **South America**

Chile (CL)
Continent: **South America**

7. (Optional) As an optional step, you can use *icons* for the two links "List" and "Cards". *Bootstrap* doesn’t include an icon library by default, but there are external libraries that are recommended to use with Bootstrap.

   In this step you'll use icons from *Open Iconic* (http://useiconic.com/open/) which can easily be used as Bootstrap *font* in your page.

   Follow the (sketchy) steps below:

   a. Install the *Open Iconic* package

   ```
   npm install open-iconic
   ```

   b. Import *Open Iconic* styles in `index.html`

   ```
   <!-- Order of CSS Imports: first Bootstrap, then your project's CSS -->
   <link rel="stylesheet" href="node_modules/bootstrap/dist/css/bootstrap.min.css" />
   <link rel="stylesheet"
     href="node_modules/open-iconic/font/css/open-iconic-bootstrap.min.css" />
   <link rel="stylesheet" href="app.css" />
   ```

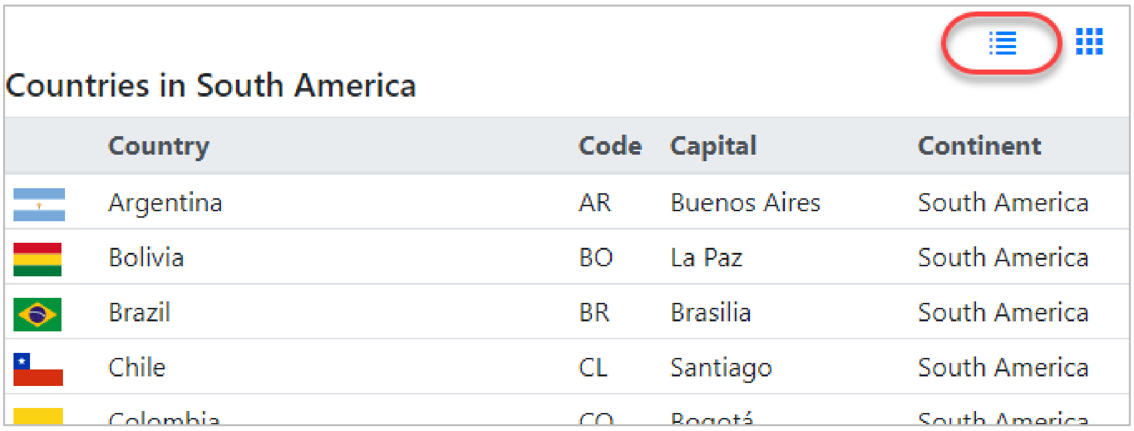
   c. Replace the two links you created in step #4.b by the following:

   ```
   <a href="#!/list" class="mx-3"><span class="oi oi-list"></span></a>
   <a href="#!/cards" class="mx-3"><span class="oi oi-grid-three-up"></span></a>
   ```

d. Add *Open Iconic* to the attributions in the `<footer>` section:

```
<footer class="footer">
  Flags of all countries by <a href="http://flagpedia.net/">flagpedia.net</a>
  | Icons by Open Iconic —
    <a href="http://www.useiconic.com/open/">www.useiconic.com/open</a>
</footer>
```

e. Test your application in **Chrome**.



Countries in South America

	Country	Code	Capital	Continent
	Argentina	AR	Buenos Aires	South America
	Bolivia	BO	La Paz	South America
	Brazil	BR	Brasilia	South America
	Chile	CL	Santiago	South America
	Colombia	CO	Bogotá	South America

Exercise 10: Creating Custom Filters and Directives

Objectives

You'll create a custom filter "startsWith" that takes two arguments: a *property* and a *search* value. You'll use it for searching by Capital names that start with the given search value.

Moreover, you'll create a custom directive "attribution" that renders the attributions in the footer of the page.

Tags: #angular #custom #filter #directive

Steps

1. To create the custom "startsWith" filter, follow the step below:

   a. Open `app.js` in the **VS Code Editor**.

   b. In the worldappCtrl controller initialize the search property that is bound to the search input field (we probably missed to do so in Exercise #6):

   ```
   $scope.search = '';
   ```

   c. Use the `filter()` method of your `worldapp` module to add the following custom filter:

   ```
   worldapp.filter('startsWith', function () {
     return function (items, property, search) {
       var filtered = [];
       angular.forEach(items, function (el) {
         if (el[property].startsWith(search)) {
           filtered.push(el);
         }
       });
       return filtered;
     }
   });
   ```

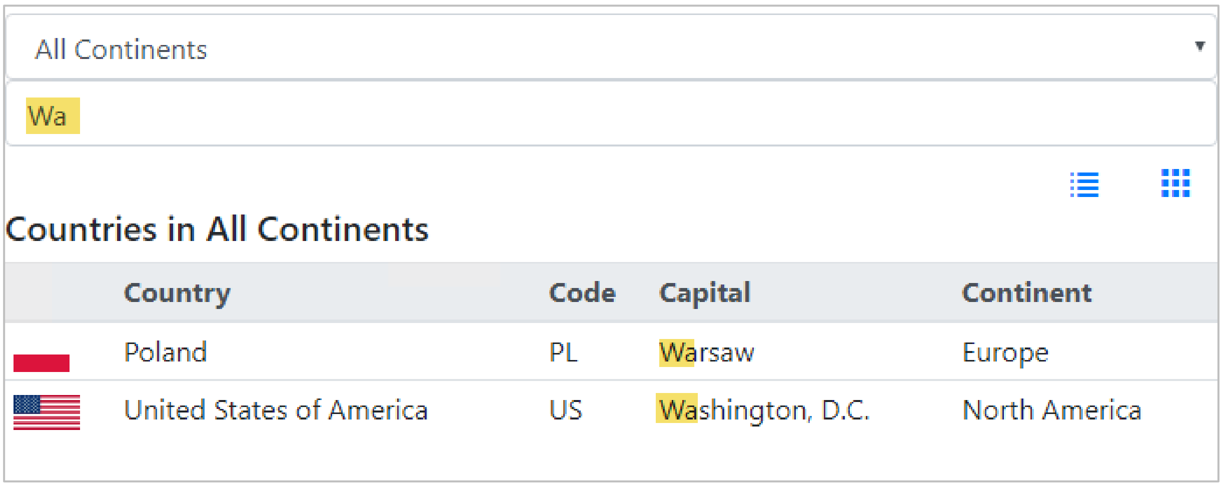
   ***Note**: You are using here the Angular API function `forEach()`. It invokes the specified function once for each item in the given collection.*
   You are also using the JavaScript method `String.startsWith()`. This method is case-sensitiv.

   d. Open `list.html` and `cards.html` in the **VS Code Editor**. In each of both files locate the `ng-repeat` attribute value and replace the existing filter expression by a new expression that uses the custom *startsWith* filter:

   ```
   ~~filter:{Capital:search}~~
   startsWith:'Capital':search
   ```

2. Test your application in **Chrome**.

   The behavior of the "Search for capital…" input field should have changed now. Before, for example, you could find *Washington* by entering "wash" or "ington". Now, Washington is only found if you enter something like "Was" or "Wash". Just "Wa" will return *Washington* and *Warsaw*.



All Continents

Wa

Countries in All Continents

	Country	Code	Capital	Continent
	Poland	PL	Warsaw	Europe
	United States of America	US	Washington, D.C.	North America

3. To create a custom "attribution" directive follow the steps below:

   a. Open `app.js` in the **VS Code Editor**.

   b. In the `worldappCtrl` controller initialize an array `attributions` of attribution objects:

```
$scope.attributions = [
  {label:'Flags of all countries by', url:'flagpedia.net'},
  {label:'Icons by Open Iconic -', url:'useiconic.com/open'}];
```

   c. Use the `directive()` method of your `worldapp` module to add the following custom directive:

```
worldapp.directive('attribution', function() {
  return {
    template: '| <span ng-repeat="a in attributions">{{a.label}} \
    <a href="http://{{a.url}}">{{a.url}}</a> | </span>'
  };
});
```

   d. Open `index.html` in the **VS Code Editor**. Replace the content of the `<footer>` section by a single `<div>` that has an `attribution` attribute:

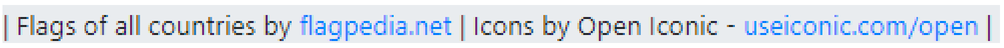
```
<footer class="footer">
  <div attribution></div>
</footer>
```

   (*Optional*) You can also try to use the directive as *element* (instead of as *attribute*):

```
<footer class="footer">
  <attribution/>
</footer>
```

4. Test your application in **Chrome**.

   The footer should look almost the same as before:



| Flags of all countries by flagpedia.net | Icons by Open Iconic - useiconic.com/open |