**Learn AngularJS With These 5 Practical Examples**

Martin Angelov August 30th, 2013

[**Download**](http://demo.tutorialzine.com/2013/08/learn-angularjs-5-examples/angularjs-examples.zip)

By now you’ve probably heard of [AngularJS](http://angularjs.org/) – the exciting open source framework, developed by Google, that changes the way you think about web apps. There has been much written about it, but I have yet to find something that is written for developers who prefer quick and practical examples. This changes today. Below you will find the basic building blocks of Angular apps - Models, Views, Controllers, Services and Filters - explained in 5 practical examples that you can edit directly in your browser. If you prefer to open them up in your favorite code editor, grab the zip above.

**What is AngularJS?**

On a high level, [AngularJS](http://angularjs.org/" \t "_blank) is a framework that binds your HTML (views) to JavaScript objects (models). When your models change, the page updates automatically. The opposite is also true – a model, associated with a text field, is updated when the content of the field is changed. Angular handles all the glue code, so you don’t have to update HTML manually or listen for events, like you do with jQuery. As a matter of fact, none of the examples here even include jQuery!

To use AngularJS, you have to include it in your page before the closing <body> tag. [Google’s CDN](https://developers.google.com/speed/libraries/devguide#angularjs) is recommended for a faster load time:

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.0.7/angular.min.js"></script>

AngularJS gives you a large number of [directives](http://docs.angularjs.org/guide/directive) that let you associate HTML elements to models. They are attributes that start with **ng-** and can be added to any element. The most important attribute that you have to include in any page, if you wish to use Angular, is ng-app:

<body ng-app>

It should be added to an element that encloses the rest of the page, like the body element or an outermost div. Angular looks for it when the page loads and automatically evaluates all directives it sees on its child elements.

Enough with the theory! Now let’s see some code.

**1. Navigation Menu**

As a first example, we will build a navigation menu that highlights the selected entry. The example uses only Angular’s [directives](http://docs.angularjs.org/guide/directive), and is the simplest app possible using the framework. Click the “**Edit**” button to see the source code. It is ready for experimentation!

<!-- Adding the ng-app declaration to initialize AngularJS -->

<div id="main" ng-app>

<!-- The navigation menu will get the value of the "active" variable as a class.

The $event.preventDefault() stops the page from jumping when a link is clicked. -->

<nav class="{{active}}" ng-click="$event.preventDefault()">

<!-- When a link in the menu is clicked, we set the active variable -->

<a href="#" class="home" ng-click="active='home'">Home</a>

<a href="#" class="projects" ng-click="active='projects'">Projects</a>

<a href="#" class="services" ng-click="active='services'">Services</a>

<a href="#" class="contact" ng-click="active='contact'">Contact</a>

</nav>

<!-- ng-show will show an element if the value in the quotes is truthful,

while ng-hide does the opposite. Because the active variable is not set

initially, this will cause the first paragraph to be visible. -->

<p ng-hide="active">Please click a menu item</p>

<p ng-show="active">You chose <b>{{active}}</b></p>

</div>

\*{

margin:0;

padding:0;

}

body{

font:15px/1.3 'Open Sans', sans-serif;

color: #5e5b64;

text-align:center;

}

a, a:visited {

outline:none;

color:#389dc1;

}

a:hover{

text-decoration:none;

}

section, footer, header, aside, nav{

display: block;

}

/\*-------------------------

The menu

--------------------------\*/

nav{

display:inline-block;

margin:60px auto 45px;

background-color:#5597b4;

box-shadow:0 1px 1px #ccc;

border-radius:2px;

}

nav a{

display:inline-block;

padding: 18px 30px;

color:#fff !important;

font-weight:bold;

font-size:16px;

text-decoration:none !important;

line-height:1;

text-transform: uppercase;

background-color:transparent;

-webkit-transition:background-color 0.25s;

-moz-transition:background-color 0.25s;

transition:background-color 0.25s;

}

nav a:first-child{

border-radius:2px 0 0 2px;

}

nav a:last-child{

border-radius:0 2px 2px 0;

}

nav.home .home,

nav.projects .projects,

nav.services .services,

nav.contact .contact{

background-color:#e35885;

}

p{

font-size:22px;

font-weight:bold;

color:#7d9098;

}

p b{

color:#ffffff;

display:inline-block;

padding:5px 10px;

background-color:#c4d7e0;

border-radius:2px;

text-transform:uppercase;

font-size:18px;

}

htmlcss



1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

<!-- Adding the ng-app declaration to initialize AngularJS -->

<div id="main" ng-app>

    <!-- The navigation menu will get the value of the "active" variable as a class.

         The $event.preventDefault() stops the page from jumping when a link is clicked. -->

    <nav class="{{active}}" ng-click="$event.preventDefault()">

        <!-- When a link in the menu is clicked, we set the active variable -->

        <a href="#" class="home" ng-click="active='home'">Home</a>

        <a href="#" class="projects" ng-click="active='projects'">Projects</a>

        <a href="#" class="services" ng-click="active='services'">Services</a>

        <a href="#" class="contact" ng-click="active='contact'">Contact</a>

    </nav>

    <!-- ng-show will show an element if the value in the quotes is truthful,

         while ng-hide does the opposite. Because the active variable is not set

         initially, this will cause the first paragraph to be visible. -->

    <p ng-hide="active">Please click a menu item</p>

    <p ng-show="active">You chose <b>{{active}}</b></p>

</div>

X

Edit

Run

In the code above, we are using Angular’s directives to set and read the **active** variable. When it changes, it causes the HTML that uses it to be updated automatically. In Angular’s terminology, this variable is called a model. It is available to all directives in the current scope, and can be accessed in your controllers (more on that in the next example).

If you have used JavaScript templates before, you are familiar with the {{var}} syntax. When the framework sees such a string, it replaces it with the contents of the variable. This operation is repeated every time var is changed.

**2. Inline Editor**

For the second example, we will create a simple inline editor – clicking a paragraph will show a tooltip with a text field. We will use a controller that will initialize the models and declare two methods for toggling the visibility of the tooltip. Controllers are regular JavaScript functions which are executed automatically by Angular, and which are associated with your page using the ng-controller directive:

<!-- When this element is clicked, hide the tooltip -->

<div id="main" ng-app ng-controller="InlineEditorController" ng-click="hideTooltip()">

<!-- This is the tooltip. It is shown only when the showtooltip variable is truthful -->

<div class="tooltip" ng-click="$event.stopPropagation()" ng-show="showtooltip">

<!-- ng-model binds the contents of the text field with the "value" model.

Any changes to the text field will automatically update the value, and

all other bindings on the page that depend on it. -->

<input type="text" ng-model="value" />

</div>

<!-- Call a method defined in the InlineEditorController that toggles

the showtooltip variable -->

<p ng-click="toggleTooltip($event)">{{value}}</p>

</div>

// The controller is a regular JavaScript function. It is called

// once when AngularJS runs into the ng-controller declaration.

function InlineEditorController($scope){

// $scope is a special object that makes

// its properties available to the view as

// variables. Here we set some default values:

$scope.showtooltip = false;

$scope.value = 'Edit me.';

// Some helper functions that will be

// available in the angular declarations

$scope.hideTooltip = function(){

// When a model is changed, the view will be automatically

// updated by by AngularJS. In this case it will hide the tooltip.

$scope.showtooltip = false;

}

$scope.toggleTooltip = function(e){

e.stopPropagation();

$scope.showtooltip = !$scope.showtooltip;

}

}

\*{

margin:0;

padding:0;

}

body{

font:15px/1.3 'Open Sans', sans-serif;

color: #5e5b64;

text-align:center;

}

a, a:visited {

outline:none;

color:#389dc1;

}

a:hover{

text-decoration:none;

}

section, footer, header, aside, nav{

display: block;

}

/\*-------------------------

The edit tooltip

--------------------------\*/

.tooltip{

background-color:#5c9bb7;

background-image:-webkit-linear-gradient(top, #5c9bb7, #5392ad);

background-image:-moz-linear-gradient(top, #5c9bb7, #5392ad);

background-image:linear-gradient(top, #5c9bb7, #5392ad);

box-shadow: 0 1px 1px #ccc;

border-radius:3px;

width: 290px;

padding: 10px;

position: absolute;

left:50%;

margin-left:-150px;

top: 80px;

}

.tooltip:after{

/\* The tip of the tooltip \*/

content:'';

position:absolute;

border:6px solid #5190ac;

border-color:#5190ac transparent transparent;

width:0;

height:0;

bottom:-12px;

left:50%;

margin-left:-6px;

}

.tooltip input{

border: none;

width: 100%;

line-height: 34px;

border-radius: 3px;

box-shadow: 0 2px 6px #bbb inset;

text-align: center;

font-size: 16px;

font-family: inherit;

color: #8d9395;

font-weight: bold;

outline: none;

}

p{

font-size:22px;

font-weight:bold;

color:#6d8088;

height: 30px;

cursor:default;

}

p b{

color:#ffffff;

display:inline-block;

padding:5px 10px;

background-color:#c4d7e0;

border-radius:2px;

text-transform:uppercase;

font-size:18px;

}

p:before{

content:'✎';

display:inline-block;

margin-right:5px;

font-weight:normal;

vertical-align: text-bottom;

}

#main{

height:300px;

position:relative;

padding-top: 150px;

}

htmljscss



1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

<!-- When this element is clicked, hide the tooltip -->

<div id="main" ng-app ng-controller="InlineEditorController" ng-click="hideTooltip()">

    <!-- This is the tooltip. It is shown only when the showtooltip variable is truthful -->

    <div class="tooltip" ng-click="$event.stopPropagation()" ng-show="showtooltip">

        <!-- ng-model binds the contents of the text field with the "value" model.

         Any changes to the text field will automatically update the value, and

         all other bindings on the page that depend on it.  -->

        <input type="text" ng-model="value" />

    </div>

    <!-- Call a method defined in the InlineEditorController that toggles

     the showtooltip variable -->

    <p ng-click="toggleTooltip($event)">{{value}}</p>

</div>

X

Edit

Run

When the controller function is executed, it gets the special $scope object as a parameter. Adding properties or functions to it makes them available to the view. Using the ng-model binding on the text field tells Angular to update that variable when the value of the field changes (this in turn re-renders the paragraph with the value).

**3. Order Form**

In this example, we will code an order form with a total price updated in real time, using another one of Angular’s useful features – [filters](http://docs.angularjs.org/guide/dev_guide.templates.filters). Filters let you modify models and can be chained together using the pipe character |. In the example below, I am using the [currency filter](http://docs.angularjs.org/api/ng.filter:currency), to turn a number into a properly formatted price, complete with a dollar sign and cents. You can easily make your own filters, as you will see in example #4.

<!-- Declare a new AngularJS app and associate the controller -->

<form ng-app ng-controller="OrderFormController">

<h1>Services</h1>

<ul>

<!-- Loop through the services array, assign a click handler, and set or

remove the "active" css class if needed -->

<li ng-repeat="service in services" ng-click="toggleActive(service)" ng-class="{active:service.active}">

<!-- Notice the use of the currency filter, it will format the price -->

{{service.name}} <span>{{service.price | currency}}</span>

</li>

</ul>

<div class="total">

<!-- Calculate the total price of all chosen services. Format it as currency. -->

Total: <span>{{total() | currency}}</span>

</div>

</form>

function OrderFormController($scope){

// Define the model properties. The view will loop

// through the services array and genreate a li

// element for every one of its items.

$scope.services = [

{

name: 'Web Development',

price: 300,

active:true

},{

name: 'Design',

price: 400,

active:false

},{

name: 'Integration',

price: 250,

active:false

},{

name: 'Training',

price: 220,

active:false

}

];

$scope.toggleActive = function(s){

s.active = !s.active;

};

// Helper method for calculating the total price

$scope.total = function(){

var total = 0;

// Use the angular forEach helper method to

// loop through the services array:

angular.forEach($scope.services, function(s){

if (s.active){

total+= s.price;

}

});

return total;

};

}

\*{

margin:0;

padding:0;

}

body{

font:15px/1.3 'Open Sans', sans-serif;

color: #5e5b64;

text-align:center;

}

a, a:visited {

outline:none;

color:#389dc1;

}

a:hover{

text-decoration:none;

}

section, footer, header, aside, nav{

display: block;

}

/\*-------------------------

The order form

--------------------------\*/

form{

background-color: #61a1bc;

border-radius: 2px;

box-shadow: 0 1px 1px #ccc;

width: 400px;

padding: 35px 60px;

margin: 50px auto;

}

form h1{

color:#fff;

font-size:64px;

font-family:'Cookie', cursive;

font-weight: normal;

line-height:1;

text-shadow:0 3px 0 rgba(0,0,0,0.1);

}

form ul{

list-style:none;

color:#fff;

font-size:20px;

font-weight:bold;

text-align: left;

margin:20px 0 15px;

}

form ul li{

padding:20px 30px;

background-color:#e35885;

margin-bottom:8px;

box-shadow:0 1px 1px rgba(0,0,0,0.1);

cursor:pointer;

}

form ul li span{

float:right;

}

form ul li.active{

background-color:#8ec16d;

}

div.total{

border-top:1px solid rgba(255,255,255,0.5);

padding:15px 30px;

font-size:20px;

font-weight:bold;

text-align: left;

color:#fff;

}

div.total span{

float:right;

}

htmljscss



1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

<!-- Declare a new AngularJS app and associate the controller -->

<form ng-app ng-controller="OrderFormController">

    <h1>Services</h1>

    <ul>

        <!-- Loop through the services array, assign a click handler, and set or

            remove the "active" css class if needed -->

        <li ng-repeat="service in services" ng-click="toggleActive(service)" ng-class="{active:service.active}">

            <!-- Notice the use of the currency filter, it will format the price -->

            {{service.name}} <span>{{service.price | currency}}</span>

        </li>

    </ul>

    <div class="total">

        <!-- Calculate the total price of all chosen services. Format it as currency. -->

        Total: <span>{{total() | currency}}</span>

    </div>

</form>

X

Edit

Run

The ng-repeat binding ([docs](http://docs.angularjs.org/api/ng.directive:ngRepeat)) is another useful feature of the framework. It lets you loop through an array of items and generate markup for  them. It is intelligently updated when an item is changed or deleted.

**Note:** For a more complete version, see [this tutorial](http://tutorialzine.com/2013/05/quick-tip-convert-backbone-to-angularjs/), which is based on [this one](http://tutorialzine.com/2013/04/services-chooser-backbone-js/), written with Backbone.js.

**4. Instant Search**

This example will allow users to filter a list of items by typing into a text field. This is another place where Angular shines, and is the perfect use case for writing a custom filter. To do this though, we first have to turn our application into a module.

Modules are a way of organizing JavaScript applications into self-contained components that can be combined in new and interesting ways. Angular relies on this technique for code isolation and requires that your application follows it before you can create a filter. There are only two things that you need to do to turn your app into a module:

1. Use the angular.module("name",[]) function call in your JS. This will instantiate and return a new module;
2. Pass the name of the module as the value of the ng-app directive.

Creating a filter then is as simple as calling the filter() method on the module object returned by angular.module("name", []).

<!-- Initialize a new AngularJS app and associate it with a module named "instantSearch"-->

<div ng-app="instantSearch" ng-controller="InstantSearchController">

<div class="bar">

<!-- Create a binding between the searchString model and the text field -->

<input type="text" ng-model="searchString" placeholder="Enter your search terms" />

</div>

<ul>

<!-- Render a li element for every entry in the items array. Notice

the custom search filter "searchFor". It takes the value of the

searchString model as an argument.

-->

<li ng-repeat="i in items | searchFor:searchString">

<a href="{{i.url}}"><img ng-src="{{i.image}}" /></a>

<p>{{i.title}}</p>

</li>

</ul>

</div>

// Define a new module for our app. The array holds the names of dependencies if any.

var app = angular.module("instantSearch", []);

// Create the instant search filter

app.filter('searchFor', function(){

// All filters must return a function. The first parameter

// is the data that is to be filtered, and the second is an

// argument that may be passed with a colon (searchFor:searchString)

return function(arr, searchString){

if(!searchString){

return arr;

}

var result = [];

searchString = searchString.toLowerCase();

// Using the forEach helper method to loop through the array

angular.forEach(arr, function(item){

if(item.title.toLowerCase().indexOf(searchString) !== -1){

result.push(item);

}

});

return result;

};

});

// The controller

function InstantSearchController($scope){

// The data model. These items would normally be requested via AJAX,

// but are hardcoded here for simplicity. See the next example for

// tips on using AJAX.

$scope.items = [

{

url: 'http://tutorialzine.com/2013/07/50-must-have-plugins-for-extending-twitter-bootstrap/',

title: '50 Must-have plugins for extending Twitter Bootstrap',

image: 'http://cdn.tutorialzine.com/wp-content/uploads/2013/07/featured\_4-100x100.jpg'

},

{

url: 'http://tutorialzine.com/2013/08/simple-registration-system-php-mysql/',

title: 'Making a Super Simple Registration System With PHP and MySQL',

image: 'http://cdn.tutorialzine.com/wp-content/uploads/2013/08/simple\_registration\_system-100x100.jpg'

},

{

url: 'http://tutorialzine.com/2013/08/slideout-footer-css/',

title: 'Create a slide-out footer with this neat z-index trick',

image: 'http://cdn.tutorialzine.com/wp-content/uploads/2013/08/slide-out-footer-100x100.jpg'

},

{

url: 'http://tutorialzine.com/2013/06/digital-clock/',

title: 'How to Make a Digital Clock with jQuery and CSS3',

image: 'http://cdn.tutorialzine.com/wp-content/uploads/2013/06/digital\_clock-100x100.jpg'

},

{

url: 'http://tutorialzine.com/2013/05/diagonal-fade-gallery/',

title: 'Smooth Diagonal Fade Gallery with CSS3 Transitions',

image: 'http://cdn.tutorialzine.com/wp-content/uploads/2013/05/featured-100x100.jpg'

},

{

url: 'http://tutorialzine.com/2013/05/mini-ajax-file-upload-form/',

title: 'Mini AJAX File Upload Form',

image: 'http://cdn.tutorialzine.com/wp-content/uploads/2013/05/ajax-file-upload-form-100x100.jpg'

},

{

url: 'http://tutorialzine.com/2013/04/services-chooser-backbone-js/',

title: 'Your First Backbone.js App – Service Chooser',

image: 'http://cdn.tutorialzine.com/wp-content/uploads/2013/04/service\_chooser\_form-100x100.jpg'

}

];

}

\*{

margin:0;

padding:0;

}

body{

font:15px/1.3 'Open Sans', sans-serif;

color: #5e5b64;

text-align:center;

}

a, a:visited {

outline:none;

color:#389dc1;

}

a:hover{

text-decoration:none;

}

section, footer, header, aside, nav{

display: block;

}

/\*-------------------------

The search input

--------------------------\*/

.bar{

background-color:#5c9bb7;

background-image:-webkit-linear-gradient(top, #5c9bb7, #5392ad);

background-image:-moz-linear-gradient(top, #5c9bb7, #5392ad);

background-image:linear-gradient(top, #5c9bb7, #5392ad);

box-shadow: 0 1px 1px #ccc;

border-radius: 2px;

width: 400px;

padding: 14px;

margin: 45px auto 20px;

position:relative;

}

.bar input{

background:#fff no-repeat 13px 13px;

background-image:url(data:image/png;base64,);

border: none;

width: 100%;

line-height: 19px;

padding: 11px 0;

border-radius: 2px;

box-shadow: 0 2px 8px #c4c4c4 inset;

text-align: left;

font-size: 14px;

font-family: inherit;

color: #738289;

font-weight: bold;

outline: none;

text-indent: 40px;

}

ul{

list-style: none;

width: 428px;

margin: 0 auto;

text-align: left;

}

ul li{

border-bottom: 1px solid #ddd;

padding: 10px;

overflow: hidden;

}

ul li img{

width:60px;

height:60px;

float:left;

border:none;

}

ul li p{

margin-left: 75px;

font-weight: bold;

padding-top: 12px;

color:#6e7a7f;

}

htmljscss



1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

<!-- Initialize a new AngularJS app and associate it with a module named "instantSearch"-->

<div ng-app="instantSearch" ng-controller="InstantSearchController">

    <div class="bar">

        <!-- Create a binding between the searchString model and the text field -->

        <input type="text" ng-model="searchString" placeholder="Enter your search terms" />

    </div>

    <ul>

        <!-- Render a li element for every entry in the items array. Notice

             the custom search filter "searchFor". It takes the value of the

             searchString model as an argument.

         -->

        <li ng-repeat="i in items | searchFor:searchString">

            <a href="{{i.url}}"><img ng-src="{{i.image}}" /></a>

            <p>{{i.title}}</p>

        </li>

    </ul>

</div>

X

Edit

Run

Filters follow the Angular.js philosophy – every piece of code that you write should be self-contained, testable and reusable. You can use this filter in all your views and even combine it with others through chaining.

**5. Switchable Grid**

Another popular UI interaction is switching between different layout modes (grid or list) with a click of a button. This is very easy to do in Angular. In addition, I will introduce another important concept – **Services**. They are objects that can be used by your application to communicate with a server, an API, or another data source. In our case, we will write a service that communicates with [Instagram’s API](http://instagram.com/developer/) and returns an array with the most popular photos at the moment.

Note that for this code to work, we will have to include one additional Angular.js file in the page:

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.0.7/angular-resource.min.js"></script>

This includes the [ngResource module](http://docs.angularjs.org/api/ngResource.$resource) for easily working with AJAX APIs (the module is exposed as the **$resource** variable in the code). This file is automatically included in the editor below.

<div ng-app="switchableGrid" ng-controller="SwitchableGridController">

<div class="bar">

<!-- These two buttons switch the layout variable,

which causes the correct UL to be shown. -->

<a class="list-icon" ng-class="{active: layout == 'list'}" ng-click="layout = 'list'"></a>

<a class="grid-icon" ng-class="{active: layout == 'grid'}" ng-click="layout = 'grid'"></a>

</div>

<!-- We have two layouts. We choose which one to show depending on the "layout" binding -->

<ul ng-show="layout == 'grid'" class="grid">

<!-- A view with big photos and no text -->

<li ng-repeat="p in pics">

<a href="{{p.link}}" target="\_blank"><img ng-src="{{p.images.low\_resolution.url}}" /></a>

</li>

</ul>

<ul ng-show="layout == 'list'" class="list">

<!-- A compact view smaller photos and titles -->

<li ng-repeat="p in pics">

<a href="{{p.link}}" target="\_blank"><img ng-src="{{p.images.thumbnail.url}}" /></a>

<p>{{p.caption.text}}</p>

</li>

</ul>

</div>

// Define a new module. This time we declare a dependency on

// the ngResource module, so we can work with the Instagram API

var app = angular.module("switchableGrid", ['ngResource']);

// Create and register the new "instagram" service

app.factory('instagram', function($resource){

return {

fetchPopular: function(callback){

// The ngResource module gives us the $resource service. It makes working with

// AJAX easy. Here I am using the client\_id of a test app. Replace it with yours.

var api = $resource('https://api.instagram.com/v1/media/popular?client\_id=:client\_id&callback=JSON\_CALLBACK',{

client\_id: '642176ece1e7445e99244cec26f4de1f'

},{

// This creates an action which we've chosen to name "fetch". It issues

// an JSONP request to the URL of the resource. JSONP requires that the

// callback=JSON\_CALLBACK part is added to the URL.

fetch:{method:'JSONP'}

});

api.fetch(function(response){

// Call the supplied callback function

callback(response.data);

});

}

}

});

// The controller. Notice that I've included our instagram service which we

// defined below. It will be available inside the function automatically.

function SwitchableGridController($scope, instagram){

// Default layout of the app. Clicking the buttons in the toolbar

// changes this value.

$scope.layout = 'grid';

$scope.pics = [];

// Use the instagram service and fetch a list of the popular pics

instagram.fetchPopular(function(data){

// Assigning the pics array will cause the view

// to be automatically redrawn by Angular.

$scope.pics = data;

});

}

\*{

margin:0;

padding:0;

}

body{

font:15px/1.3 'Open Sans', sans-serif;

color: #5e5b64;

text-align:center;

}

a, a:visited {

outline:none;

color:#389dc1;

}

a:hover{

text-decoration:none;

}

section, footer, header, aside, nav{

display: block;

}

/\*-------------------------

The search input

--------------------------\*/

.bar{

background-color:#5c9bb7;

background-image:-webkit-linear-gradient(top, #5c9bb7, #5392ad);

background-image:-moz-linear-gradient(top, #5c9bb7, #5392ad);

background-image:linear-gradient(top, #5c9bb7, #5392ad);

box-shadow: 0 1px 1px #ccc;

border-radius: 2px;

width: 580px;

padding: 10px;

margin: 45px auto 25px;

position:relative;

text-align:right;

line-height: 1;

}

.bar a{

background:#4987a1 center center no-repeat;

width:32px;

height:32px;

display:inline-block;

text-decoration:none !important;

margin-right:5px;

border-radius:2px;

cursor:pointer;

}

.bar a.active{

background-color:#c14694;

}

.bar a.list-icon{

background-image:url(data:image/png;base64,);

}

.bar a.grid-icon{

background-image:url(data:image/png;base64,);

}

.bar input{

background:#fff no-repeat 13px 13px;

border: none;

width: 100%;

line-height: 19px;

padding: 11px 0;

border-radius: 2px;

box-shadow: 0 2px 8px #c4c4c4 inset;

text-align: left;

font-size: 14px;

font-family: inherit;

color: #738289;

font-weight: bold;

outline: none;

text-indent: 40px;

}

/\*-------------------------

List layout

--------------------------\*/

ul.list{

list-style: none;

width: 500px;

margin: 0 auto;

text-align: left;

}

ul.list li{

border-bottom: 1px solid #ddd;

padding: 10px;

overflow: hidden;

}

ul.list li img{

width:120px;

height:120px;

float:left;

border:none;

}

ul.list li p{

margin-left: 135px;

font-weight: bold;

color:#6e7a7f;

}

/\*-------------------------

Grid layout

--------------------------\*/

ul.grid{

list-style: none;

width: 570px;

margin: 0 auto;

text-align: left;

}

ul.grid li{

padding: 2px;

float:left;

}

ul.grid li img{

width:280px;

height:280px;

display:block;

border:none;

}

htmljscss



1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

<div ng-app="switchableGrid" ng-controller="SwitchableGridController">

    <div class="bar">

        <!-- These two buttons switch the layout variable,

             which causes the correct UL to be shown. -->

        <a class="list-icon" ng-class="{active: layout == 'list'}" ng-click="layout = 'list'"></a>

        <a class="grid-icon" ng-class="{active: layout == 'grid'}" ng-click="layout = 'grid'"></a>

    </div>

    <!-- We have two layouts. We choose which one to show depending on the "layout" binding -->

    <ul ng-show="layout == 'grid'" class="grid">

        <!-- A view with big photos and no text -->

        <li ng-repeat="p in pics">

            <a href="{{p.link}}" target="\_blank"><img ng-src="{{p.images.low\_resolution.url}}" /></a>

        </li>

    </ul>

    <ul ng-show="layout == 'list'" class="list">

        <!-- A compact view smaller photos and titles -->

        <li ng-repeat="p in pics">

            <a href="{{p.link}}" target="\_blank"><img ng-src="{{p.images.thumbnail.url}}" /></a>

            <p>{{p.caption.text}}</p>

        </li>

    </ul>

</div>

X

Edit

Run

Services are entirely self-contained, which makes it possible to write different implementations without affecting the rest of your code. For example, while testing, you might prefer to return a hard-coded array of photos which would speed up your tests.

**Further Reading**

If you’ve reached this point, you have already grasped the basics of developing with Angular. However, there is much to learn if you want to be a pro. Here is a list of resources which will help you in your quest:

* [The AngularJS Home Page](http://angularjs.org/)
* [The AngularJS Guide](http://docs.angularjs.org/guide/overview)
* [The Official AngularJS Tutorial](http://docs.angularjs.org/tutorial)
* [A list of lots and lots of more resources, videos and tutorials](https://github.com/jmcunningham/AngularJS-Learning)