AngularJS

- Introduction
- Expression
- Modules
- Directives
- Model
- Data Binding
- Controllers
- Scopes
- Filters
- Services
- HTTP
- Tables
- Select
- SQL
- DOM
- Events
- Forms
- Validation
- API
- Application

Angular

Web and desktop application development has become more accessible and safer as different approaches and new frameworks have come on the market, making application development much more sophisticated. In this tutorial series, you will learn about the Angular framework that brings revolutionary changes in developing software applications.

What Is Angular?

Angular (formerly called Angular JS) is a typescript-based web application framework that supports full-stack development for building all types of web applications. It helps in creating reactive single page application (SPA) and is completely based on the concept of components. Google owns Angular, and its stable version was released on September 14, 2016. Angular's official website is https://angular.io/. Google makes sure that they release a major version of Angular every six months.

Angular has nowadays become the most popular framework for developing mobile and desktop-based web applications. It comes with a variety of features. The version over 2.0 is called Angular. Angular 1.0 was named Angular JS. The latest version of Angular comes with all the possible features you may need to build a complex and

sophisticated web application for desktop and mobile. Some of the key features of Angular are components, forms, directives, dependency injection, HTTP services, pipes, etc.

What Is Single Page Application (SPA)

Single-page applications are web applications or a particular type of website that provide users with a very intuitive, responsive, and fast user experience. It is enriched with menus, multiple blocks, tiles, and interactive buttons on one page, helping users easily navigate the application. It helps to load a portion of the current page dynamically instead of reloading the entire page from the server. This is why Angular-based applications are called reactive fast-speed loading pages.

What Is Angular CLI

Angular CLI is a command-line interface tool that automates the application development process by initializing new Angular applications and maintaining them directly from a command shell. As you plan to set up your project using Angular CLI, it will show all its built-in features. You can learn about the various features that Angular provides for web application development.

Features of Angular

1. Multiple platforms: Angular helps develop desktop applications for different operating systems. Native applications can also be built using Angular with Cordova, NativeScript, or Ionic.
2. High performance and speed: Angular's performance is very high, and the reasons behind this high performance are:
    - Angular is used as a front-end web tool and can work in conjunction with PHP, Node.js, Struts of Java, and has the ability for near-instant rendering using only CSS and HTML.
    - It optimizes the web application for improved SEO
    - Angular provides its applications with the ability to load faster with new component routers and single page application (SPA) support.
    - Creating templates in Angular is also done with highly customized code.
3. Full-stack development: This framework also provides testing (Jasmine and Karma for unit testing), accessibility, automation, and supports full-stack development with Express JS, Node.JS, and MongoDB.
4. High Productivity: The angular framework provides a better yet, simple and powerful syntax for templates, CLIs, and IDEs to increase the productivity of any application development.
    - UI views can be developed easily and rapidly using Angular's powerful templates.
    - Intellisense and intelligent code completion IDEs and error detection editors make development smart and efficient.
    - Angular's CLI can rapidly build and test components and deploy them immediately.

Overall, it can be said that Angular is a powerful tool to work with and an efficient one from the developer's perspective.

Difference Between Angular and Angular JS

Angular	Angular JS
The name Angular became popular after Angular 2.0.	This was the common name for the first version of Angular (Angular 1.0).
It is a TypeScript-based framework.	It is a JavaScript-based framework.
It provides the feature of dynamic loading of web pages.	It does not provide such dynamic loading of pages.
Angular uses the concept of components as its primary building entity rather than scopes and controllers.	The older version of Angular (Angular JS) uses scope and controller as its primary application developing entity.
It uses TypeScript (a language of Microsoft) that provides features like OOPs, static typing, and the concept of generic programming for doing certain functions.	It uses a simple JS file that is merged with HTML pages to do certain functions.
It supports the feature of server-side programming.	It does not support the feature of server-side programming.
It uses different forms of expression syntaxes, such as [] and ().	It uses simple syntaxes that get embedded with HTML pages.

Angular -Development Environment Setup

Learn how to set up an Angular development environment, including installing Node.js and the Angular CLI, creating a new project, and running the development server. With this tutorial, you'll be able to start building Angular applications in no time.

To work with Angular (any version), you will need to install a few applications on your computer:

- Setup a Node.js development environment with npm (node package manager)
- Setup the Angular CLI (command-line interface)
- Setup an Editor/IDE

The Reason Why Angular Needs node.js

Node.js is a server-side backend, which makes it important but not mandatory for AngularJS. However, you will need node.js for the following purposes:

- npm is a package manager that comes with node.js by default; it allows you to manage your project dependencies. Therefore, you do not have to manually add dependencies, remove some, and update your package. Most Angular libraries are assigned as different NPM packages.

- npm provides the Angular CLI (command-line interface), a great tool for building Angular applications efficiently.
- TypeScript is a primary language for building angular applications, which is not supported directly by web browsers. To compile them in JavaScript, node.js is required.
- Angular works on the client side, while on the server side, you will need a node.js server environment for processing.

node.js Development Environment Setup

To get the best development experience, and if you do not yet have the node pre-installed on your System: Please go to the download page of the nodej.org website and install the latest version of the node. To check whether 'Node.js' is already installed in your system, you need to type the `node -v` command in your terminal prompt. This will immediately show you the Node.js version installed on your system. Furthermore, to check the Node Package Manager, you can use the `npm -v` command in your terminal prompt.

What is Angular CLI?

The Angular CLI (command-line interface) is a great tool to efficiently build Angular applications by automating operations in Angular projects rather than manually.

Angular CLI (command-line interface) Setup

If you have installed node.js on your system, the next step is to install Angular CLI by using the following command:

```
npm install -g @angular/cli
```

Here in the above command, `-g` is used for global installation. If you write this, you can use the CLI directly in any Angular project on your system.

Type the `ng version` or `ng v` command at your terminal prompt to verify your angular version.

Create a New Angular Project

After installing Angular CLI, you can now use it to create a new Angular project using the following command:

```
ng new my-app
```

The above command creates a new angular project(my-app) with all the required dependencies. Replace my-app with the desired name for your project. This will create a new directory with the same name and initialize it as an Angular project.

Navigate to the project directory:

```
cd my-app
```

Run the development server:

```
ng serve
```
This will compile and serve the application and will be available at http://localhost:4200/. The development server will also automatically reload the application whenever you make changes to the code.

That's it! You now have a working Angular development environment. You can start building your application by modifying the files in the `src` directory.

To build a production version of your application, run the following command:

```
ng build --prod
```
The code above will create a production-ready build of your application in the `dist` directory.

AngularJS History

AngularJS version 1.0 was released in 2012.

Miško Hevery, a Google employee, started to work with AngularJS in 2009.

The idea turned out very well, and the project is now officially supported by Google.

AngularJS is a **JavaScript framework**. It can be added to an HTML page with a <script> tag.

AngularJS extends HTML attributes with **Directives**, and binds data to HTML with **Expressions**.

AngularJS is a JavaScript Framework

AngularJS is a JavaScript framework written in JavaScript.

AngularJS is distributed as a JavaScript file, and can be added to a web page with a script tag:

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
```

AngularJS Extends HTML

AngularJS extends HTML with **ng-directives**.

The **ng-app** directive defines an AngularJS application.

The **ng-model** directive binds the value of HTML controls (input, select, textarea) to application data.

The **ng-bind** directive binds application data to the HTML view.

AngularJS Example

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="">
  <p>Name: <input type="text" ng-model="name"></p>
  <p ng-bind="name"></p>
</div>

</body>
</html>
```

Example explained:

AngularJS starts automatically when the web page has loaded.

The **ng-app** directive tells AngularJS that the element is the "owner" of an AngularJS **application**.

The **ng-model** directive binds the value of the input field to the application variable **name**.

The **ng-bind** directive binds the content of the <p> element to the application variable **name**.

AngularJS Directives

As you have already seen, AngularJS directives are HTML attributes with an **ng** prefix.

The **ng-init** directive initializes AngularJS application variables.

AngularJS Example

```
<div ng-app="" ng-init="firstName='JSPM'">

<p>The name is <span ng-bind="firstName"></span></p>

</div>
```

Alternatively with valid HTML:

AngularJS Example

```
<div data-ng-app="" data-ng-init="firstName='JSPM'">

<p>The name is <span data-ng-bind="firstName"></span></p>

</div>
```

You can use **data-ng-**, instead of **ng-**, if you want to make your page HTML valid.

You will learn a lot more about directives later in the classes.

AngularJS Expressions

AngularJS expressions are written inside double braces: **{{ expression }}**.

AngularJS will "output" data exactly where the expression is written:

AngularJS Example

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angu
lar.min.js"></script>
<body>

<div ng-app="">
  <p>My first expression: {{ 5 + 5 }}</p>
</div>
```

```
</body>
</html>
```

AngularJS expressions bind AngularJS data to HTML the same way as the **ng-bind** directive.

AngularJS Example

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="">
  <p>Name: <input type="text" ng-model="name"></p>
  <p>{{name}}</p>
</div>

</body>
</html>
```

You will learn more about expressions later in the class.

AngularJS Applications

AngularJS **modules** define AngularJS applications.

AngularJS **controllers** control AngularJS applications.

The **ng-app** directive defines the application, the **ng-controller** directive defines the controller.

AngularJS Example

```
<div ng-app="myApp" ng-controller="myCtrl">

University Name: <input type="text" ng-model="UniName"><br>
Location: <input type="text" ng-model="location"><br>
<br>
University: {{UniName + " " + location}}

</div>
```

```
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.UniName= "JSPM";
  $scope.location= "Pune";
});
</script>
```

AngularJS modules define applications:

AngularJS Module

```
var app = angular.module('myApp', []);
```

AngularJS controllers control applications:

AngularJS Controller

```
app.controller('myCtrl', function($scope) {
  $scope.UniName= "JSPM";
  $scope.location= "Pune";
});
```

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="">

<p>Input something in the input box:</p>
<p>Name : <input type="text" ng-model="name" placeholder="Enter name here"></p>
<h1>Hello {{name}}</h1>

</div>

</body>
</html>
```

AngularJS Expressions

AngularJS binds data to HTML using **Expressions**.

AngularJS Expressions

AngularJS expressions can be written inside double braces: {{ *expression* }}.

AngularJS expressions can also be written inside a directive: ng-bind="*expression*".

AngularJS will resolve the expression, and return the result exactly where the expression is written.

AngularJS expressions are much like **JavaScript expressions:** They can contain literals, operators, and variables.

Example {{ 5 + 5 }} or {{ firstName + " " + lastName }}

Example

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="">
  <p>My first expression: {{ 15 + 15 }}</p>
</div>

</body>
</html>
```

If you remove the ng-app directive, HTML will display the expression as it is, without solving it:

Example

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div>
  <p>My expression: {{ 51 + 51 }}</p>
</div>
```

```
</body>
</html>
```

You can write expressions wherever you like, AngularJS will simply resolve the expression and return the result.

Example: Let AngularJS change the value of CSS properties.

Change the color of the input box below, by changing its value:

Example

```
<div ng-app="" ng-init="myCol='lightblue'">

<input style="background-color:{{myCol}}" ng-model="myCol">

</div>
```

AngularJS Numbers

AngularJS numbers are like JavaScript numbers:

Example

```
<div ng-app="" ng-init="quantity=1;cost=5">

<p>Total in dollar: {{ quantity * cost }}</p>

</div>
```

Same example using `ng-bind`:

Example

```
<div ng-app="" ng-init="quantity=1;cost=5">

<p>Total in dollar: <span ng-bind="quantity * cost"></span></p>

</div>
```

Using ng-init is not very common. You will learn a better way to initialize data in the chapter about controllers.

AngularJS Strings

AngularJS strings are like JavaScript strings:

Example

```
<div ng-app="" ng-init="UniName='JSPM';location='Pune'">

<p>The name is {{ UniName + " " + location }}</p>

</div>
```

Same example using ng-bind:

Example

```
<div ng-app="" ng-init="UniName='JSPM';location='Pune'">

<p>The University is <span ng-bind="UniName + ' , ' +
location"></span></p>

</div>
```

AngularJS Objects

AngularJS objects are like JavaScript objects:

Example

```
<div ng-app="" ng-init="university={UniName:'JSPM',location:'Pune'}">

<p>The University city is {{ university.location }}</p>

</div>
```

Same example using ng-bind:

Example

```
<div ng-app="" ng-init="university={UniName:'JSPM',location:'Pune'}">

<p>The University city is <span ng-
bind="university.location"></span></p>

</div>
```

AngularJS Arrays

AngularJS arrays are like JavaScript arrays:

Example

```
<div ng-app="" ng-init="points=[1,15,19,2,40]">

<p>The third result is {{ points[2] }}</p>

</div>
```

Same example using ng-bind:

Example

```
<div ng-app="" ng-init="points=[1,15,19,2,40]">

<p>The third result is <span ng-bind="points[2]"></span></p>

</div>
```

AngularJS Expressions vs. JavaScript Expressions

Like JavaScript expressions, AngularJS expressions can contain literals, operators, and variables.

Unlike JavaScript expressions, AngularJS expressions can be written inside HTML.

AngularJS expressions do not support conditionals, loops, and exceptions, while JavaScript expressions do.

AngularJS expressions support filters, while JavaScript expressions do not.

AngularJS Modules

An AngularJS module defines an application.

The module is a container for the different parts of an application.

The module is a container for the application controllers.

Controllers always belong to a module.

Creating a Module

A module is created by using the AngularJS function `angular.module`

```
<div ng-app="myApp">...</div>

<script>

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

</script>
```

The "myApp" parameter refers to an HTML element in which the application will run.

Now you can add controllers, directives, filters, and more, to your AngularJS application.

Adding a Controller

Add a controller to your application, and refer to the controller with the `ng-controller` directive:

Example

```
<div ng-app="myApp" ng-controller="myCtrl">
{{ firstName + " " + lastName }}
</div>

<script>

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

app.controller("myCtrl", function($scope) {
  $scope.firstName = "John";
  $scope.lastName = "Doe";
});

</script>
```

You will learn more about controllers later in this tutorial.

Adding a Directive

AngularJS has a set of built-in directives which you can use to add functionality to your application.

For a full reference, visit our AngularJS directive reference.

In addition you can use the module to add your own directives to your applications:

Example

```
<div ng-app="myApp" w3-test-directive></div>

<script>
var app = angular.module("myApp", []);

app.directive("w3TestDirective", function() {
  return {
    template : "I was made in a directive constructor!"
  };
});
</script>
```

You will learn more about directives later in this tutorial.

Modules and Controllers in Files

It is common in AngularJS applications to put the module and the controllers in JavaScript files.

In this example, "myApp.js" contains an application module definition, while "myCtrl.js" contains the controller:

Example

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="myApp" ng-controller="myCtrl">
{{ firstName + " " + lastName }}
</div>

<script src="myApp.js"></script>
<script src="myCtrl.js"></script>

</body>
</html>
```

myApp.js

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

The [] parameter in the module definition can be used to define dependent modules.

Without the [] parameter, you are not *creating* a new module,
but *retrieving* an existing one.

myCtrl.js

```
app.controller("myCtrl", function($scope) {
  $scope.firstName = "John";
  $scope.lastName= "Doe";
});
```

Functions can Pollute the Global Namespace

Global functions should be avoided in JavaScript. They can easily be overwritten or destroyed by other scripts.

AngularJS modules reduces this problem, by keeping all functions local to the module.

When to Load the Library

While it is common in HTML applications to place scripts at the end of the `<body>` element, it is recommended that you load the AngularJS library either in the `<head>` or at the start of the `<body>`.

This is because calls to `angular.module` can only be compiled after the library has been loaded.

Example

```
<!DOCTYPE html>
<html>
<body>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>

<div ng-app="myApp" ng-controller="myCtrl">
{{ firstName + " " + lastName }}
</div>

<script>
var app = angular.module("myApp", []);
app.controller("myCtrl", function($scope) {
  $scope.firstName = "John";
  $scope.lastName = "Doe";
});
</script>

</body>
</html>
```

AngularJS Directives

AngularJS lets you extend HTML with new attributes called **Directives**.

AngularJS has a set of built-in directives which offers functionality to your applications.

AngularJS also lets you define your own directives.

AngularJS Directives

AngularJS directives are extended HTML attributes with the prefix `ng-`.

The `ng-app` directive initializes an AngularJS application.

The `ng-init` directive initializes application data.

The `ng-model` directive binds the value of HTML controls (input, select, textarea) to application data.

Read about all AngularJS directives in our AngularJS directive reference.

Example

```
<div ng-app="" ng-init="firstName='John'">

<p>Name: <input type="text" ng-model="firstName"></p>
<p>You wrote: {{ firstName }}</p>

</div>
```

The `ng-app` directive also tells AngularJS that the element is the "owner" of the AngularJS application.

Data Binding

The `{{ firstName }}` expression, in the example above, is an AngularJS data binding expression.

Data binding in AngularJS binds AngularJS expressions with AngularJS data.

`{{ firstName }}` is bound with `ng-model="firstName"`.

In the next example two text fields are bound together with two ng-model directives:

Example

```
<div ng-app="" ng-init="quantity=1;price=5">

Quantity: <input type="number" ng-model="quantity">
Costs:    <input type="number" ng-model="price">

Total in dollar: {{ quantity * price }}

</div>
```

Using `ng-init` is not very common. You will learn how to initialize data in the chapter about controllers.

Repeating HTML Elements

The `ng-repeat` directive repeats an HTML element:

Example

```
<div ng-app="" ng-init="names=['Jani','Hege','Kai']">
  <ul>
    <li ng-repeat="x in names">
      {{ x }}
    </li>
  </ul>
</div>
```

The `ng-repeat` directive actually **clones HTML elements** once for each item in a collection.

The `ng-repeat` directive used on an array of objects:

Example

```
<div ng-app="" ng-init="names=[
{name:'Jani',country:'Norway'},
{name:'Hege',country:'Sweden'},
{name:'Kai',country:'Denmark'}]">

<ul>
  <li ng-repeat="x in names">
    {{ x.name + ', ' + x.country }}
  </li>
</ul>

</div>
```

AngularJS is perfect for database CRUD (Create Read Update Delete) applications.
Just imagine if these objects were records from a database.

The ng-app Directive

The `ng-app` directive defines the **root element** of an AngularJS application.

The `ng-app` directive will **auto-bootstrap** (automatically initialize) the application when a web page is loaded.

The ng-init Directive

The `ng-init` directive defines **initial values** for an AngularJS application.

Normally, you will not use ng-init. You will use a controller or module instead.

You will learn more about controllers and modules later.

The ng-model Directive

The `ng-model` directive binds the value of HTML controls (input, select, textarea) to application data.

The ng-model directive can also:

- Provide type validation for application data (number, email, required).
- Provide status for application data (invalid, dirty, touched, error).
- Provide CSS classes for HTML elements.
- Bind HTML elements to HTML forms.

Read more about the ng-model directive in the next chapter.

Create New Directives

In addition to all the built-in AngularJS directives, you can create your own directives.

New directives are created by using the .directive function.

To invoke the new directive, make an HTML element with the same tag name as the new directive.

When naming a directive, you must use a camel case name, w3TestDirective, but when invoking it, you must use - separated name, w3-test-directive:

Example

```
<body ng-app="myApp">

<w3-test-directive></w3-test-directive>

<script>
var app = angular.module("myApp", []);
app.directive("w3TestDirective", function() {
  return {
    template : "<h1>Made by a directive!</h1>"
  };
});
</script>

</body>
```

You can invoke a directive by using:

- Element name
- Attribute
- Class

- Comment

The examples below will all produce the same result:

Element name

```
<w3-test-directive></w3-test-directive>
```

Attribute

```
<div w3-test-directive></div>
```

Class

```
<div class="w3-test-directive"></div>
```

Comment

```
<!-- directive: w3-test-directive -->
```

Restrictions

You can restrict your directives to only be invoked by some of the methods.

Example

By adding a `restrict` property with the value `"A"`, the directive can only be invoked by attributes:

```
var app = angular.module("myApp", []);
app.directive("w3TestDirective", function() {
  return {
    restrict : "A",
    template : "<h1>Made by a directive!</h1>"
  };
});
```

The legal restrict values are:

- `E` for Element name
- `A` for Attribute

- C for Class
- M for Comment

By default the value is EA, meaning that both Element names and attribute names can invoke the directive.

AngularJS ng-model Directive

The ng-model directive binds the value of HTML controls (input, select, textarea) to application data.

The ng-model Directive

With the ng-model directive you can bind the value of an input field to a variable created in AngularJS.

Example

```
<div ng-app="myApp" ng-controller="myCtrl">
  Name: <input ng-model="name">
</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.name = "John Doe";
});
</script>
```

Two-Way Binding

The binding goes both ways. If the user changes the value inside the input field, the AngularJS property will also change its value:

Example

```
<div ng-app="myApp" ng-controller="myCtrl">
  Name: <input ng-model="name">
```

```
  <h1>You entered: {{name}}</h1>
</div>
```

Validate User Input

The `ng-model` directive can provide type validation for application data (number, e-mail, required):

Example

```
<form ng-app="" name="myForm">
  Email:
  <input type="email" name="myAddress" ng-model="text">
  <span ng-show="myForm.myAddress.$error.email">Not a valid e-mail
address</span>
</form>
```

In the example above, the span will be displayed only if the expression in the `ng-show` attribute returns `true`.

If the property in the `ng-model` attribute does not exist, AngularJS will create one for you.

Application Status

The `ng-model` directive can provide status for application data (valid, dirty, touched, error):

Example

```
<form ng-app="" name="myForm" ng-init="myText = 'post@myweb.com'">
  Email:
  <input type="email" name="myAddress" ng-model="myText" required>
  <h1>Status</h1>
  {{myForm.myAddress.$valid}}
  {{myForm.myAddress.$dirty}}
  {{myForm.myAddress.$touched}}
</form>
```

CSS Classes

The `ng-model` directive provides CSS classes for HTML elements, depending on their status:

Example

```
<style>
input.ng-invalid {
  background-color: lightblue;
}

</style>
<body>

<form ng-app="" name="myForm">
  Enter your name:
  <input name="myName" ng-model="myText" required>
</form>
```

The `ng-model` directive adds/removes the following classes, according to the status of the form field:

- ng-empty
- ng-not-empty
- ng-touched
- ng-untouched
- ng-valid
- ng-invalid
- ng-dirty
- ng-pending
- ng-pristine

AngularJS Data Binding

Data binding in AngularJS is the synchronization between the model and the view.

Data Model

AngularJS applications usually have a data model. The data model is a collection of data available for the application.

Example

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.firstname = "John";
  $scope.lastname = "Doe";
});
```

HTML View

The HTML container where the AngularJS application is displayed, is called the view.

The view has access to the model, and there are several ways of displaying model data in the view.

You can use the `ng-bind` directive, which will bind the innerHTML of the element to the specified model property:

Example

```
<p ng-bind="firstname"></p>
```

You can also use double braces `{{ }}` to display content from the model:

Example

```
<p>First name: {{firstname}}</p>
```

Or you can use the `ng-model` directive on HTML controls to bind the model to the view.

The ng-model Directive

Use the ng-model directive to bind data from the model to the view on HTML controls (input, select, textarea)

Example

```
<input ng-model="firstname">
```

The ng-model directive provides a two-way binding between the model and the view.

Two-way Binding

Data binding in AngularJS is the synchronization between the model and the view.

When data in the *model* changes, the *view* reflects the change, and when data in the *view* changes, the *model* is updated as well. This happens immediately and automatically, which makes sure that the model and the view is updated at all times.

Example

```
<div ng-app="myApp" ng-controller="myCtrl">
  Name: <input ng-model="firstname">
  <h1>{{firstname}}</h1>
</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.firstname = "Mickey";
  $scope.lastname = "Doe";
});
</script>
```

AngularJS Controller

Applications in AngularJS are controlled by controllers. Read about controllers in the AngularJS Controllers chapter.

Because of the immediate synchronization of the model and the view, the controller can be completely separated from the view, and simply concentrate on the model data. Thanks to the data binding in AngularJS, the view will reflect any changes made in the controller.

Example

```
<div ng-app="myApp" ng-controller="myCtrl">
  <h1 ng-click="changeName()">{{firstname}}</h1>
</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.firstname = "Mini";
  $scope.changeName = function() {
    $scope.firstname = "Nelly";
  }
});

</script>
```

AngularJS Controllers

AngularJS controllers **control the data** of AngularJS applications.

AngularJS controllers are regular **JavaScript Objects**.

AngularJS Controllers

AngularJS applications are controlled by controllers.

The **ng-controller** directive defines the application controller.

A controller is a **JavaScript Object**, created by a standard JavaScript **object constructor**.

AngularJS Example

```
<div ng-app="myApp" ng-controller="myCtrl">

First Name: <input type="text" ng-model="firstName"><br>
```

```
Last Name: <input type="text" ng-model="lastName"><br>
<br>
Full Name: {{firstName + " " + lastName}}

</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.firstName = "Joseph";
  $scope.lastName = "Deny";
});
</script>
```

Application explained:

The AngularJS application is defined by **ng-app="myApp"**. The application runs inside the .

The **ng-controller="myCtrl"** attribute is an AngularJS directive. It defines a controller.

The **myCtrl** function is a JavaScript function.

AngularJS will invoke the controller with a **$scope** object.

In AngularJS, $scope is the application object (the owner of application variables and functions).

The controller creates two properties (variables) in the scope (**firstName** and **lastName**).

The **ng-model** directives bind the input fields to the controller properties (firstName and lastName).

Controller Methods

The example above demonstrated a controller object with two properties: lastName and firstName.

A controller can also have methods (variables as functions):

AngularJS Example

```
<div ng-app="myApp" ng-controller="personCtrl">

First Name: <input type="text" ng-model="firstName"><br>
Last Name: <input type="text" ng-model="lastName"><br>
<br>
Full Name: {{fullName()}}

</div>

<script>
var app = angular.module('myApp', []);
app.controller('personCtrl', function($scope) {
  $scope.firstName = "Mini";
  $scope.lastName = "Deni";
  $scope.fullName = function() {
    return $scope.firstName + " " + $scope.lastName;
  };
});
</script>
```

Controllers In External Files

In larger applications, it is common to store controllers in external files.

Just copy the code between the <script> tags into an external file named personController.js:

AngularJS Example

```
<div ng-app="myApp" ng-controller="personCtrl">

First Name: <input type="text" ng-model="firstName"><br>
Last Name: <input type="text" ng-model="lastName"><br>
<br>
Full Name: {{fullName()}}

</div>

<script src="personController.js"></script>
```

Another Example

For the next example we will create a new controller file:

```
angular.module('myApp', []).controller('namesCtrl', function($scope) {
  $scope.names = [
    {name:'Jini',country:'India'},
    {name:'Hero',country:'Nepal'},
    {name:'iak',country:'UK'}
  ];
});
```

Save the file as namedController.js:

And then use the controller file in an application:

AngularJS Example

```
<div ng-app="myApp" ng-controller="namesCtrl">

<ul>
  <li ng-repeat="x in names">
    {{ x.name + ', ' + x.country }}
  </li>
</ul>

</div>

<script src="namedController.js"></script>
```

AngularJS Scope

The scope is the binding part between the HTML (view) and the JavaScript (controller).

The scope is an object with the available properties and methods.

The scope is available for both the view and the controller.

How to Use the Scope?

When you make a controller in AngularJS, you pass the `$scope` object as an argument:

Example

Properties made in the controller, can be referred to in the view:

```
<div ng-app="myApp" ng-controller="myCtrl">

<h1>{{carname}}</h1>

</div>

<script>
var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {
  $scope.carname = "Volvo";
});

</script>
```

When adding properties to the `$scope` object in the controller, the view (HTML) gets access to these properties.

In the view, you do not use the prefix `$scope`, you just refer to a property name, like `{{carname}}`.

Understanding the Scope

If we consider an AngularJS application to consist of:

- View, which is the HTML.
- Model, which is the data available for the current view.
- Controller, which is the JavaScript function that makes/changes/removes/controls the data.

Then the scope is the Model.

The scope is a JavaScript object with properties and methods, which are available for both the view and the controller.

Example

If you make changes in the view, the model and the controller will be updated:

```
<div ng-app="myApp" ng-controller="myCtrl">

<input ng-model="name">

<h1>My name is {{name}}</h1>

</div>

<script>
var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {
  $scope.name = "Johe Dawid";
});

</script>
```

Know Your Scope

It is important to know which scope you are dealing with, at any time.

In the two examples above there is only one scope, so knowing your scope is not an issue, but for larger applications there can be sections in the HTML DOM which can only access certain scopes.

Example

When dealing with the `ng-repeat` directive, each repetition has access to the current repetition object:

```
<div ng-app="myApp" ng-controller="myCtrl">
<ul>
  <li ng-repeat="x in names">{{x}}</li>
</ul>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.names = ["Emil", "Tobias", "Linus"];
});
</script>
```

Each `<li>` element has access to the current repetition object, in this case a string, which is referred to by using `x`.

Root Scope

All applications have a `$rootScope` which is the scope created on the HTML element that contains the `ng-app` directive.

The rootScope is available in the entire application.

If a variable has the same name in both the current scope and in the rootScope, the application uses the one in the current scope.

Example

A variable named "color" exists in both the controller's scope and in the rootScope:

```
<body ng-app="myApp">
<p>The rootScope's favorite color:</p>
<h1>{{color}}</h1>
<div ng-controller="myCtrl">
  <p>The scope of the controller's favorite color:</p>
  <h1>{{color}}</h1>
</div>
<p>The rootScope's favorite color is still:</p>
<h1>{{color}}</h1>
<script>
var app = angular.module('myApp', []);
app.run(function($rootScope) {
  $rootScope.color = 'blue';
});
app.controller('myCtrl', function($scope) {
  $scope.color = "red";
});
</script>
</body>
```

AngularJS Filters

Filters can be added in AngularJS to format data.

AngularJS Filters

AngularJS provides filters to transform data:

- `currency` Format a number to a currency format.
- `date` Format a date to a specified format.
- `filter` Select a subset of items from an array.
- `json` Format an object to a JSON string.
- `limitTo` Limits an array/string, into a specified number of elements/characters.
- `lowercase` Format a string to lower case.
- `number` Format a number to a string.
- `orderBy` Orders an array by an expression.
- `uppercase` Format a string to upper case.

Adding Filters to Expressions

Filters can be added to expressions by using the pipe character `|`, followed by a filter.

The `uppercase` filter format strings to upper case:

Example

```
<div ng-app="myApp" ng-controller="personCtrl">
<p>The name is {{ lastName | uppercase }}</p>
</div>
```

The `lowercase` filter format strings to lower case:

Example

```
<div ng-app="myApp" ng-controller="personCtrl">
<p>The name is {{ lastName | lowercase }}</p>
</div>
```

Adding Filters to Directives

Filters are added to directives, like `ng-repeat`, by using the pipe character `|`, followed by a filter:

Example

The `orderBy` filter sorts an array:

```
<div ng-app="myApp" ng-controller="namesCtrl">
<ul>
  <li ng-repeat="x in names | orderBy:'country'">
    {{ x.name + ', ' + x.country }}
  </li>
</ul>
</div>
```

The currency Filter

The `currency` filter formats a number as currency:

Example

```
<div ng-app="myApp" ng-controller="costCtrl">
<h1>Price: {{ price | currency }}</h1>
</div>
```

The filter Filter

The `filter` filter selects a subset of an array.

The `filter` filter can only be used on arrays, and it returns an array containing only the matching items.

Example

Return the names that contains the letter "i":

```
<div ng-app="myApp" ng-controller="namesCtrl">
<ul>
  <li ng-repeat="x in names | filter : 'i'">
    {{ x }}
```

```
  </li>
</ul>
</div>
```

Filter an Array Based on User Input

By setting the `ng-model` directive on an input field, we can use the value of the input field as an expression in a filter.

Type a letter in the input field, and the list will shrink/grow depending on the match:

- Jani
- Carl
- Margareth
- Hege
- Joe
- Gustav
- Birgit
- Mary
- Kai

Example

```
<div ng-app="myApp" ng-controller="namesCtrl">

<p><input type="text" ng-model="test"></p>

<ul>
  <li ng-repeat="x in names | filter : test">
    {{ x }}
  </li>
</ul>

</div>
```

Sort an Array Based on User Input

Click the table headers to change the sort order::

Name	Country
Jani	India
Carl	Sweden
Margareth	England
Hege	Norway
Joe	Denmark
Gustav	Sweden
Birgit	Denmark
Mary	England
Kai	Norway

By adding the `ng-click` directive on the table headers, we can run a function that changes the sorting order of the array:

Example

```
<div ng-app="myApp" ng-controller="namesCtrl">

<table border="1" width="100%">
```

```
  <tr>
    <th ng-click="orderByMe('name')">Name</th>
    <th ng-click="orderByMe('country')">Country</th>
  </tr>
  <tr ng-repeat="x in names | orderBy:myOrderBy">
    <td>{{x.name}}</td>
    <td>{{x.country}}</td>
  </tr>
</table>

</div>

<script>
angular.module('myApp', []).controller('namesCtrl', function($scope) {
  $scope.names = [
    {name:'Jani',country:'India'},
    {name:'Carl',country:'Sweden'},
    {name:'Margareth',country:'England'},
    {name:'Hege',country:'Norway'},
    {name:'Joe',country:'Denmark'},
    {name:'Gustav',country:'Sweden'},
    {name:'Birgit',country:'Denmark'},
    {name:'Mary',country:'England'},
    {name:'Kai',country:'Norway'}
  ];
  $scope.orderByMe = function(x) {
    $scope.myOrderBy = x;
  }
});
</script>
```

Custom Filters

You can make your own filters by registering a new filter factory function with your module:

Example

Make a custom filter called "myFormat":

```
<ul ng-app="myApp" ng-controller="namesCtrl">
  <li ng-repeat="x in names">
    {{x | myFormat}}
  </li>
</ul>
```

```
<script>
var app = angular.module('myApp', []);
app.filter('myFormat', function() {
  return function(x) {
    var i, c, txt = "";
    for (i = 0; i < x.length; i++) {
      c = x[i];
      if (i % 2 == 0) {
        c = c.toUpperCase();
      }
      txt += c;
    }
    return txt;
  };
});
app.controller('namesCtrl', function($scope) {
  $scope.names =
['Jani', 'Carl', 'Margareth', 'Hege', 'Joe', 'Gustav', 'Birgit', 'Mary'
, 'Kai'];
});
</script>
```

The `myFormat` filter will format every other character to uppercase.

AngularJS Services

In AngularJS you can make your own service, or use one of the many built-in services.

What is a Service?

In AngularJS, a service is a function, or object, that is available for, and limited to, your AngularJS application.

AngularJS has about 30 built-in services. One of them is the `$location` service.

The `$location` service has methods which return information about the location of the current web page:

Example

Use the `$location` service in a controller:

```
var app = angular.module('myApp', []);
app.controller('customersCtrl', function($scope, $location) {
    $scope.myUrl = $location.absUrl();
});
```

Note that the `$location` service is passed in to the controller as an argument. In order to use the service in the controller, it must be defined as a dependency.

Why use Services?

For many services, like the `$location` service, it seems like you could use objects that are already in the DOM, like the `window.location` object, and you could, but it would have some limitations, at least for your AngularJS application.

AngularJS constantly supervises your application, and for it to handle changes and events properly, AngularJS prefers that you use the `$location` service instead of the `window.location` object.

The $http Service

The `$http` service is one of the most common used services in AngularJS applications. The service makes a request to the server, and lets your application handle the response.

Example

Use the `$http` service to request data from the server:

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
  $http.get("welcome.htm").then(function (response) {
    $scope.myWelcome = response.data;
  });
});
```

This example demonstrates a very simple use of the `$http` service.

The $timeout Service

The `$timeout` service is AngularJS' version of the `window.setTimeout` function.

Example

Display a new message after two seconds:

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $timeout) {
  $scope.myHeader = "Hello World!";
  $timeout(function () {
    $scope.myHeader = "How are you today?";
  }, 2000);
});
```

The $interval Service

The `$interval` service is AngularJS' version of the `window.setInterval` function.

Example

Display the time every second:

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $interval) {
  $scope.theTime = new Date().toLocaleTimeString();
  $interval(function () {
    $scope.theTime = new Date().toLocaleTimeString();
  }, 1000);
});
```

Create Your Own Service

To create your own service, connect your service to the module:

Create a service named `hexafy`:

```
app.service('hexafy', function() {
  this.myFunc = function (x) {
    return x.toString(16);
  }
});
```

To use your custom made service, add it as a dependency when defining the controller:

Example

Use the custom made service named `hexafy` to convert a number into a hexadecimal number:

```
app.controller('myCtrl', function($scope, hexafy) {
  $scope.hex = hexafy.myFunc(255);
});
```

Use a Custom Service Inside a Filter

Once you have created a service, and connected it to your application, you can use the service in any controller, directive, filter, or even inside other services.

To use the service inside a filter, add it as a dependency when defining the filter:

The service `hexafy` used in the filter `myFormat`:

```
app.filter('myFormat',['hexafy', function(hexafy) {
  return function(x) {
    return hexafy.myFunc(x);
  };
}]);
```

You can use the filter when displaying values from an object, or an array:

```
<ul>
  <li ng-repeat="x in counts">{{x | myFormat}}</li>
</ul>
```

AngularJS AJAX - $http

$http is an AngularJS service for reading data from remote servers.

AngularJS $http

The AngularJS `$http` service makes a request to the server, and returns a response.

Example

Make a simple request to the server, and display the result in a header:

```
<div ng-app="myApp" ng-controller="myCtrl">

<p>Today's welcome message is:</p>
<h1>{{myWelcome}}</h1>

</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
  $http.get("welcome.htm")
  .then(function(response) {
    $scope.myWelcome = response.data;
  });
});

</script>
```

Methods

The example above uses the `.get` method of the `$http` service.

The .get method is a shortcut method of the $http service. There are several shortcut methods:

- .delete()
- .get()
- .head()
- .jsonp()
- .patch()
- .post()
- .put()

The methods above are all shortcuts of calling the $http service:

Example

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
  $http({
    method : "GET",
      url : "welcome.htm"
  }).then(function mySuccess(response) {
    $scope.myWelcome = response.data;
  }, function myError(response) {
    $scope.myWelcome = response.statusText;
  });
});
```

The example above executes the $http service with an object as an argument. The object is specifying the HTTP method, the url, what to do on success, and what to do on failure.

Properties

The response from the server is an object with these properties:

- .config the object used to generate the request.
- .data a string, or an object, carrying the response from the server.
- .headers a function to use to get header information.
- .status a number defining the HTTP status.
- .statusText a string defining the HTTP status.

Example

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
  $http.get("welcome.htm")
```

```
  .then(function(response) {
    $scope.content = response.data;
    $scope.statuscode = response.status;
    $scope.statustext = response.statusText;
  });
});
```

To handle errors, add one more functions to the `.then` method:

Example

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
  $http.get("wrongfilename.htm")
  .then(function(response) {
    // First function handles success
    $scope.content = response.data;
  }, function(response) {
    // Second function handles error
    $scope.content = "Something went wrong";
  });
});
```

JSON

The data you get from the response is expected to be in JSON format.

JSON is a great way of transporting data, and it is easy to use within AngularJS, or any other JavaScript.

Example: On the server we have a file that returns a JSON object containing 15 customers, all wrapped in array called `records`.

Click here to take a look at the JSON object.

Example

The `ng-repeat` directive is perfect for looping through an array:

```
<div ng-app="myApp" ng-controller="customersCtrl">

<ul>
  <li ng-repeat="x in myData">
    {{ x.Name + ', ' + x.Country }}
```

```
  </li>
</ul>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('customersCtrl', function($scope, $http) {
  $http.get("customers.php").then(function(response) {
    $scope.myData = response.data.records;
  });
});
</script>
```

Application explained:

The application defines the `customersCtrl` controller, with a `$scope` and `$http` object.

`$http` is an **XMLHttpRequest object** for requesting external data.

`$http.get()` reads **JSON data** from https://www.w3schools.com/angular/customers.php.

On success, the controller creates a property, `myData`, in the scope, with JSON data from the server.

AngularJS Tables

The ng-repeat directive is perfect for displaying tables.

Displaying Data in a Table

Displaying tables with angular is very simple:

AngularJS Example

```
<div ng-app="myApp" ng-controller="customersCtrl">
<table>
  <tr ng-repeat="x in names">
    <td>{{ x.Name }}</td>
    <td>{{ x.Country }}</td>
  </tr>
</table>
```

```
</div>
<script>
var app = angular.module('myApp', []);
app.controller('customersCtrl', function($scope, $http) {
  $http.get("customers.php")
  .then(function (response) {$scope.names = response.data.records;});
});
</script>
```

Displaying with CSS Style

To make it nice, add some CSS to the page:

CSS Style

```
<style>
table, th , td {
  border: 1px solid grey;
  border-collapse: collapse;
  padding: 5px;
}
table tr:nth-child(odd) {
  background-color: #f1f1f1;
}
table tr:nth-child(even) {
  background-color: #ffffff;
}
</style>
```

Display with orderBy Filter

To sort the table, add an **orderBy** filter:

AngularJS Example

```
<table>
  <tr ng-repeat="x in names | orderBy : 'Country'">
    <td>{{ x.Name }}</td>
    <td>{{ x.Country }}</td>
  </tr>
</table>
```

Display with uppercase Filter

To display uppercase, add an **uppercase** filter:

AngularJS Example

```
<table>
  <tr ng-repeat="x in names">
    <td>{{ x.Name }}</td>
    <td>{{ x.Country | uppercase }}</td>
  </tr>
</table>
```

Display the Table Index ($index)

To display the table index, add a <td> with **$index**:

AngularJS Example

```
<table>
  <tr ng-repeat="x in names">
    <td>{{ $index + 1 }}</td>
    <td>{{ x.Name }}</td>
    <td>{{ x.Country }}</td>
  </tr>
</table>
```

Using $even and $odd

AngularJS Example

```
<table>
  <tr ng-repeat="x in names">
    <td ng-if="$odd" style="background-color:#f1f1f1">{{ x.Name }}</td>
    <td ng-if="$even">{{ x.Name }}</td>
```

```
    <td ng-if="$odd" style="background-color:#f1f1f1">{{ x.Country
}}</td>
    <td ng-if="$even">{{ x.Country }}</td>
  </tr>
</table>
```

```
<!DOCTYPE html>
<html>
<style>
table, td  {
  border: 1px solid grey;
  border-collapse: collapse;
  padding: 5px;
}
</style>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="myApp" ng-controller="customersCtrl">

<table>
  <tr ng-repeat="x in names">
    <td ng-if="$odd" style="background-color:#f1f1f1">
    {{ x.Name }}</td>
    <td ng-if="$even">
    {{ x.Name }}</td>
    <td ng-if="$odd" style="background-color:#f1f1f1">
    {{ x.Country }}</td>
    <td ng-if="$even">
    {{ x.Country }}</td>
  </tr>
</table>

</div>

<script>
var app = angular.module('myApp', []);
app.controller('customersCtrl', function($scope, $http) {
    $http.get("customers.php")
    .then(function (response) {$scope.names = response.data.records;});
});
</script>

</body>
</html>
```

AngularJS Select Boxes

AngularJS lets you create dropdown lists based on items in an array, or an object.

Creating a Select Box Using ng-options

If you want to create a dropdown list, based on an object or an array in AngularJS, you should use the `ng-options` directive:

Example

```
<div ng-app="myApp" ng-controller="myCtrl">

<select ng-model="selectedName" ng-options="x for x in names">
</select>

</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.names = ["Emil", "Tobias", "Linus"];
});

</script>
```

ng-options vs ng-repeat

You can also use the `ng-repeat` directive to make the same dropdown list:

Example

```
<select>
  <option ng-repeat="x in names">{{x}}</option>
</select>
```

Because the ng-repeat directive repeats a block of HTML code for each item in an array, it can be used to create options in a dropdown list, but the ng-options directive was made especially for filling a dropdown list with options.

What Do I Use?

You can use both the ng-repeat directive and the ng-options directive:

Assume you have an array of objects:

```
$scope.cars = [
  {model : "Ford Mustang", color : "red"},
  {model : "Fiat 500", color : "white"},
  {model : "Volvo XC90", color : "black"}
];
```

Example

Using ng-repeat:

```
<select ng-model="selectedCar">
  <option ng-repeat="x in
cars" value="{{x.model}}">{{x.model}}</option>
</select>

<h1>You selected: {{selectedCar}}</h1>
```

When using the value as an object, use ng-value insead of value:

Example

Using ng-repeat as an object:

```
<select ng-model="selectedCar">
  <option ng-repeat="x in cars" ng-value="{{x}}">{{x.model}}</option>
</select>
<h1>You selected a {{selectedCar.color}} {{selectedCar.model}}</h1>
```

Example

Using ng-options:

```
<select ng-model="selectedCar" ng-options="x.model for x in cars">
</select>
```

```
<h1>You selected: {{selectedCar.model}}</h1>
<p>Its color is: {{selectedCar.color}}</p>
```

When the selected value is an object, it can hold more information, and your application can be more flexible.

The Data Source as an Object

In the previous examples the data source was an array, but we can also use an object.

Assume you have an object with key-value pairs:

```
$scope.cars = {
  car01 : "Ford",
  car02 : "Fiat",
  car03 : "Hero"
};
```

The expression in the `ng-options` attribute is a bit different for objects:

Example

Using an object as the data source, `x` represents the key, and `y` represents the value:

```
<select ng-model="selectedCar" ng-options="x for (x, y) in cars">
</select>

<h1>You selected: {{selectedCar}}</h1>
```

The selected value will always be the **value** in a key-**value** pair.

The **value** in a key-**value** pair can also be an object:

Example

The selected value will still be the **value** in a key-**value** pair, only this time it is an object:

```
$scope.cars = {
  car01 : {brand : "Ford", model : "Mustang", color : "red"},
  car02 : {brand : "Fiat", model : "500", color : "white"},
```

```
  car03 : {brand : "Hero", model : "YC90", color : "Pink"}
};
```

The options in the dropdown list does not have to be the **key** in a **key**-value pair, it can also be the value, or a property of the value object:

Example

```
<select ng-model="selectedCar" ng-options="y.brand for (x, y) in cars">
</select>
```

AngularJS SQL

AngularJS is perfect for displaying data from a Database. Just make sure the data is in JSON format.

Fetching Data From a PHP Server Running MySQL

AngularJS Example

```
<div ng-app="myApp" ng-controller="customersCtrl">

<table>
  <tr ng-repeat="x in names">
    <td>{{ x.Name }}</td>
    <td>{{ x.Country }}</td>
  </tr>
</table>

</div>

<script>
var app = angular.module('myApp', []);
app.controller('customersCtrl', function($scope, $http) {
  $http.get("customers_mysql.php")
  .then(function (response) {$scope.names = response.data.records;});
});
</script>
```

Server Code Examples

The following section is a listing of the server code used to fetch SQL data.

1. Using PHP and MySQL. Returning JSON.
2. Using PHP and MS Access. Returning JSON.

Cross-Site HTTP Requests

A request for data from a different server (other than the requesting page), are called **cross-site** HTTP requests.

Cross-site requests are common on the web. Many pages load CSS, images, and scripts from different servers.

In modern browsers, cross-site HTTP requests **from scripts** are restricted to **same site** for security reasons.

The following line, in our PHP examples, has been added to allow cross-site access.

```
header("Access-Control-Allow-Origin: *");
```

1. Server Code PHP and MySQL

```
<?php
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");

$conn = new mysqli("myServer", "myUser", "myPassword", "Northwind");

$result = $conn->query("SELECT CompanyName, City, Country FROM
Customers");

$outp = "";
while($rs = $result->fetch_array(MYSQLI_ASSOC)) {
  if ($outp != "") {$outp .= ",";}
  $outp .= '{"Name":"'  . $rs["CompanyName"] . '",';
  $outp .= '"City":"'   . $rs["City"]        . '",';
  $outp .= '"Country":"'. $rs["Country"]     . '"}';
```

```
}
$outp ='{"records":['.$outp.']}';
$conn->close();

echo($outp);
?>
```

2. Server Code PHP and MS Access

```
<?php
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=ISO-8859-1");

$conn = new COM("ADODB.Connection");
$conn->open("PROVIDER=Microsoft.Jet.OLEDB.4.0;Data
Source=Northwind.mdb");

$rs = $conn->execute("SELECT CompanyName, City, Country FROM
Customers");

$outp = "";
while (!$rs->EOF) {
  if ($outp != "") {$outp .= ",";}
  $outp .= '{"Name":"'  . $rs["CompanyName"] . '",';
  $outp .= '"City":"'   . $rs["City"]        . '",';
  $outp .= '"Country":"'. $rs["Country"]     . '"}';
  $rs->MoveNext();
}
$outp ='{"records":['.$outp.']}';

$conn->close();

echo ($outp);
?>
```

AngularJS HTML DOM

AngularJS has directives for binding application data to the attributes of HTML DOM elements.

The ng-disabled Directive

The **ng-disabled** directive binds AngularJS application data to the disabled attribute of HTML elements.

AngularJS Example

```
<div ng-app="" ng-init="mySwitch=true">

<p>
<button ng-disabled="mySwitch">Click Me!</button>
</p>

<p>
<input type="checkbox" ng-model="mySwitch">Button
</p>

<p>
{{ mySwitch }}
</p>

</div>
```

Application explained:

The **ng-disabled** directive binds the application data **mySwitch** to the HTML button's **disabled** attribute.

The **ng-model** directive binds the value of the HTML checkbox element to the value of **mySwitch**.

If the value of **mySwitch** evaluates to **true**, the button will be disabled:

```
<p>
<button disabled>Click Me!</button>
</p>
```

If the value of **mySwitch** evaluates to **false**, the button will not be disabled:

```
<p>
<button>Click Me!</button>
</p>
```

The ng-show Directive

The **ng-show** directive shows or hides an HTML element.

AngularJS Example

```
<div ng-app="">

<p ng-show="true">I am visible.</p>

<p ng-show="false">I am not visible.</p>

</div>
```

The ng-show directive shows (or hides) an HTML element based on the **value** of ng-show.

You can use any expression that evaluates to true or false:

AngularJS Example

```
<div ng-app="" ng-init="hour=13">

<p ng-show="hour > 12">I am visible.</p>

</div>
```

In the next chapter, there are more examples, using the click of a button to hide HTML elements.

The ng-hide Directive

The **ng-hide** directive hides or shows an HTML element.

AngularJS Example

```
<div ng-app="">

<p ng-hide="true">I am not visible.</p>

<p ng-hide="false">I am visible.</p>

</div>
```

AngularJS Events

AngularJS has its own HTML events directives.

AngularJS Events

You can add AngularJS event listeners to your HTML elements by using one or more of these directives:

- `ng-blur`
- `ng-change`
- `ng-click`
- `ng-copy`
- `ng-cut`
- `ng-dblclick`
- `ng-focus`
- `ng-keydown`
- `ng-keypress`
- `ng-keyup`
- `ng-mousedown`
- `ng-mouseenter`
- `ng-mouseleave`
- `ng-mousemove`
- `ng-mouseover`
- `ng-mouseup`
- `ng-paste`

The event directives allows us to run AngularJS functions at certain user events.

An AngularJS event will not overwrite an HTML event, both events will be executed.

Mouse Events

Mouse events occur when the cursor moves over an element, in this order:

1. ng-mouseover
2. ng-mouseenter

3. ng-mousemove
4. ng-mouseleave

Or when a mouse button is clicked on an element, in this order:

1. ng-mousedown
2. ng-mouseup
3. ng-click

You can add mouse events on any HTML element.

Example

Increase the count variable when the mouse moves over the H1 element:

```
<div ng-app="myApp" ng-controller="myCtrl">

<h1 ng-mousemove="count = count + 1">Mouse over me!</h1>

<h2>{{ count }}</h2>

</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.count = 0;
});
</script>
```

The ng-click Directive

The `ng-click` directive defines AngularJS code that will be executed when the element is being clicked.

Example

```
<div ng-app="myApp" ng-controller="myCtrl">

<button ng-click="count = count + 1">Click me!</button>

<p>{{ count }}</p>

</div>
```

```
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.count = 0;
});
</script>
```

You can also refer to a function:

Example

```
<div ng-app="myApp" ng-controller="myCtrl">

<button ng-click="myFunction()">Click me!</button>

<p>{{ count }}</p>

</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.count = 0;
  $scope.myFunction = function() {
    $scope.count++;
  }
});
</script>
```

Toggle, True/False

If you want to show a section of HTML code when a button is clicked, and hide when the button is clicked again, like a dropdown menu, make the button behave like a toggle switch:

Click Me

Example

```
<div ng-app="myApp" ng-controller="myCtrl">

<button ng-click="myFunc()">Click Me!</button>
```

```
<div ng-show="showMe">
  <h1>Menu:</h1>
  <div>Pizza</div>
  <div>Pasta</div>
  <div>Pesce</div>
</div>

</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.showMe = false;
  $scope.myFunc = function() {
    $scope.showMe = !$scope.showMe;
  }
});
</script>
```

The `showMe` variable starts out as the Boolean value `false`.

The `myFunc` function sets the `showMe` variable to the opposite of what it is, by using the `!` (not) operator.

$event Object

You can pass the `$event` object as an argument when calling the function.

The `$event` object contains the browser's event object:

Example

```
<div ng-app="myApp" ng-controller="myCtrl">

<h1 ng-mousemove="myFunc($event)">Mouse Over Me!</h1>

<p>Coordinates: {{x + ', ' + y}}</p>

</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.myFunc = function(myE) {
    $scope.x = myE.clientX;
    $scope.y = myE.clientY;
  }
```

```
});
</script>
```

AngularJS Forms

Forms in AngularJS provides data-binding and validation of input controls.

Input Controls

Input controls are the HTML input elements:

- input elements
- select elements
- button elements
- textarea elements

Data-Binding

Input controls provides data-binding by using the `ng-model` directive.

```
<input type="text" ng-model="firstname">
```

The application does now have a property named `firstname`.

The `ng-model` directive binds the input controller to the rest of your application.

The property `firstname`, can be referred to in a controller:

Example

```
<script>
var app = angular.module('myApp', []);
app.controller('formCtrl', function($scope) {
  $scope.firstname = "John";
});
</script>
```

It can also be referred to elsewhere in the application:

Example

```
<form>
  First Name: <input type="text" ng-model="firstname">
</form>

<h1>You entered: {{firstname}}</h1>
```

Checkbox

A checkbox has the value `true` or `false`. Apply the `ng-model` directive to a checkbox, and use its value in your application.

Example

Show the header if the checkbox is checked:

```
 <form>
  Check to show a header:
  <input type="checkbox" ng-model="myVar">
</form>

<h1 ng-show="myVar">My Header</h1>
```

Radiobuttons

Bind radio buttons to your application with the `ng-model` directive.

Radio buttons with the same `ng-model` can have different values, but only the selected one will be used.

Example

Display some text, based on the value of the selected radio button:

```
<form>
  Pick a topic:
  <input type="radio" ng-model="myVar" value="dogs">Dogs
```

```
  <input type="radio" ng-model="myVar" value="tuts">Tutorials
  <input type="radio" ng-model="myVar" value="cars">Cars
</form>
```

The value of myVar will be either `dogs`, `tuts`, or `cars`.

Selectbox

Bind select boxes to your application with the `ng-model` directive.

The property defined in the `ng-model` attribute will have the value of the selected option in the selectbox.

Example

Display some text, based on the value of the selected option:

```
<form>
  Select a topic:
  <select ng-model="myVar">
    <option value="">
    <option value="dogs">Dogs
    <option value="tuts">Tutorials
    <option value="cars">Cars
  </select>
</form>
```

The value of myVar will be either `dogs`, `tuts`, or `cars`.

An AngularJS Form Example

First Name:

Last Name:

RESET

form = {"firstName":"Joihn","lastName":"Dio"}

master = {"firstName":"Joihn","lastName":"Dio"}

Application Code

```
<div ng-app="myApp" ng-controller="formCtrl">
  <form novalidate>
    First Name:<br>
    <input type="text" ng-model="user.firstName"><br>
    Last Name:<br>
    <input type="text" ng-model="user.lastName">
    <br><br>
    <button ng-click="reset()">RESET</button>
  </form>
  <p>form = {{user}}</p>
  <p>master = {{master}}</p>
</div>

<script>
var app = angular.module('myApp', []);
app.controller('formCtrl', function($scope) {
  $scope.master = {firstName: "John", lastName: "Doe"};
  $scope.reset = function() {
    $scope.user = angular.copy($scope.master);
  };
  $scope.reset();
});
</script>
```

The **novalidate** attribute is new in HTML5. It disables any default browser validation.

Example Explained

The **ng-app** directive defines the AngularJS application.

The **ng-controller** directive defines the application controller.

The **ng-model** directive binds two input elements to the **user** object in the model.

The **formCtrl** controller sets initial values to the **master** object, and defines the **reset()** method.

The **reset()** method sets the **user** object equal to the **master** object.

The **ng-click** directive invokes the **reset()** method, only if the button is clicked.

The novalidate attribute is not needed for this application, but normally you will use it in AngularJS forms, to override standard HTML5 validation.

AngularJS Form Validation

AngularJS can validate input data.

Form Validation

AngularJS offers client-side form validation.

AngularJS monitors the state of the form and input fields (input, textarea, select), and lets you notify the user about the current state.

AngularJS also holds information about whether they have been touched, or modified, or not.

You can use standard HTML5 attributes to validate input, or you can make your own validation functions.

Client-side validation cannot alone secure user input. Server side validation is also necessary.

Required

Use the HTML5 attribute `required` to specify that the input field must be filled out:

Example

The input field is required:

```
<form name="myForm">
  <input name="myInput" ng-model="myInput" required>
</form>
```

```
<p>The input's valid state is:</p>
<h1>{{myForm.myInput.$valid}}</h1>
```

E-mail

Use the HTML5 type `email` to specify that the value must be an e-mail:

Example

The input field has to be an e-mail:

```
<form name="myForm">
  <input name="myInput" ng-model="myInput" type="email">
</form>

<p>The input's valid state is:</p>
<h1>{{myForm.myInput.$valid}}</h1>
```

Form State and Input State

AngularJS is constantly updating the state of both the form and the input fields.

Input fields have the following states:

- `$untouched` The field has not been touched yet
- `$touched` The field has been touched
- `$pristine` The field has not been modified yet
- `$dirty` The field has been modified
- `$invalid` The field content is not valid
- `$valid` The field content is valid

They are all properties of the input field, and are either `true` or `false`.

Forms have the following states:

- `$pristine` No fields have been modified yet
- `$dirty` One or more have been modified
- `$invalid` The form content is not valid
- `$valid` The form content is valid
- `$submitted` The form is submitted

They are all properties of the form, and are either `true` or `false`.

You can use these states to show meaningful messages to the user. Example, if a field is required, and the user leaves it blank, you should give the user a warning:

Example

Show an error message if the field has been touched AND is empty:

```
<input name="myName" ng-model="myName" required>
<span ng-show="myForm.myName.$touched && myForm.myName.$invalid">The
name is required.</span>
```

CSS Classes

AngularJS adds CSS classes to forms and input fields depending on their states.

The following classes are added to, or removed from, input fields:

- `ng-untouched` The field has not been touched yet
- `ng-touched` The field has been touched
- `ng-pristine` The field has not been modified yet
- `ng-dirty` The field has been modified
- `ng-valid` The field content is valid
- `ng-invalid` The field content is not valid
- `ng-valid-`*`key`* One *key* for each validation. Example: `ng-valid-required`, useful when there are more than one thing that must be validated
- `ng-invalid-`*`key`* Example: `ng-invalid-required`

The following classes are added to, or removed from, forms:

- `ng-pristine` No fields has not been modified yet
- `ng-dirty` One or more fields has been modified
- `ng-valid` The form content is valid
- `ng-invalid` The form content is not valid
- `ng-valid-`*`key`* One *key* for each validation. Example: `ng-valid-required`, useful when there are more than one thing that must be validated
- `ng-invalid-`*`key`* Example: `ng-invalid-required`

The classes are removed if the value they represent is `false`.

Add styles for these classes to give your application a better and more intuitive user interface.

Example

Apply styles, using standard CSS:

```
<style>
input.ng-invalid {
  background-color: pink;
}
input.ng-valid {
  background-color: lightgreen;
}

</style>
```

Forms can also be styled:

Example

Apply styles for unmodified (pristine) forms, and for modified forms:

```
<style>
form.ng-pristine {
  background-color: lightblue;
}
form.ng-dirty {
  background-color: pink;
}

</style>
```

Custom Validation

To create your own validation function is a bit more tricky; You have to add a new directive to your application, and deal with the validation inside a function with certain specified arguments.

Example

Create your own directive, containing a custom validation function, and refer to it by using `my-directive`.

The field will only be valid if the value contains the character "e":

```
<form name="myForm">
<input name="myInput" ng-model="myInput" required my-directive>
</form>
<script>
var app = angular.module('myApp', []);
app.directive('myDirective', function() {
  return {
    require: 'ngModel',
    link: function(scope, element, attr, mCtrl) {
      function myValidation(value) {
        if (value.indexOf("e") > -1) {
          mCtrl.$setValidity('charE', true);
        } else {
          mCtrl.$setValidity('charE', false);
        }
        return value;
      }
      mCtrl.$parsers.push(myValidation);
    }
  };
});

</script>
```

Example Explained:

In HTML, the new directive will be referred to by using the attribute `my-directive`.

In the JavaScript we start by adding a new directive named `myDirective`.

Remember, when naming a directive, you must use a camel case name, `myDirective`, but when invoking it, you must use `-` separated name, `my-directive`.

Then, return an object where you specify that we require `ngModel`, which is the ngModelController.

Make a linking function which takes some arguments, where the fourth argument, `mCtrl`, is the `ngModelController`,

Then specify a function, in this case named `myValidation`, which takes one argument, this argument is the value of the input element.

Test if the value contains the letter "e", and set the validity of the model controller to either `true` or `false`.

At last, `mCtrl.$parsers.push(myValidation);` will add the `myValidation` function to an array of other functions, which will be executed every time the input value changes.

Validation Example

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angu
lar.min.js"></script>
<body>

<h2>Validation Example</h2>

<form  ng-app="myApp"  ng-controller="validateCtrl"
name="myForm" novalidate>

<p>Username:<br>
  <input type="text" name="user" ng-model="user" required>
  <span style="color:red" ng-show="myForm.user.$dirty &&
myForm.user.$invalid">
  <span ng-show="myForm.user.$error.required">Username is
required.</span>
  </span>
</p>

<p>Email:<br>
  <input type="email" name="email" ng-model="email" required>
  <span style="color:red" ng-show="myForm.email.$dirty &&
myForm.email.$invalid">
  <span ng-show="myForm.email.$error.required">Email is
required.</span>
  <span ng-show="myForm.email.$error.email">Invalid email
address.</span>
  </span>
</p>

<p>
  <input type="submit"
  ng-disabled="myForm.user.$dirty && myForm.user.$invalid ||
  myForm.email.$dirty && myForm.email.$invalid">
</p>

</form>

<script>
var app = angular.module('myApp', []);
app.controller('validateCtrl', function($scope) {
```

```
  $scope.user = 'Joshep Dio';
  $scope.email = 'joshep.dio@ymail.com';
});
</script>
</body>
</html>
```

The HTML form attribute **novalidate** is used to disable default browser validation.

Example Explained

The AngularJS directive **ng-model** binds the input elements to the model.

The model object has two properties: **user** and **email**.

Because of **ng-show**, the spans with color:red are displayed only when user or email is **$dirty** and **$invalid**.

AngularJS API

API stands for **A**pplication **P**rogramming **I**nterface.

AngularJS Global API

The AngularJS Global API is a set of global JavaScript functions for performing common tasks like:

- Comparing objects
- Iterating objects
- Converting data

The Global API functions are accessed using the angular object.

Below is a list of some common API functions:

API	Description
angular.lowercase()	Converts a string to lowercase
angular.uppercase()	Converts a string to uppercase
angular.isString()	Returns true if the reference is a string
angular.isNumber()	Returns true if the reference is a number

---

angular.lowercase()

Example

```
<div ng-app="myApp" ng-controller="myCtrl">
  <p>{{ x1 }}</p>
  <p>{{ x2 }}</p>
</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.x1 = "JOHN";
  $scope.x2 = angular.lowercase($scope.x1);
});
</script>
```

---

angular.uppercase()

Example

```
<div ng-app="myApp" ng-controller="myCtrl">
  <p>{{ x1 }}</p>
```

```
  <p>{{ x2 }}</p>
</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.x1 = "Joshep";
  $scope.x2 = angular.uppercase($scope.x1);
});
</script>
```

angular.isString()

Example

```
<div ng-app="myApp" ng-controller="myCtrl">
  <p>{{ x1 }}</p>
  <p>{{ x2 }}</p>
</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.x1 = "JOSHEP";
  $scope.x2 = angular.isString($scope.x1);
});
</script>
```

angular.isNumber()

Example

```
<div ng-app="myApp" ng-controller="myCtrl">
  <p>{{ x1 }}</p>
  <p>{{ x2 }}</p>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.x1 = "JOHN";
  $scope.x2 = angular.isNumber($scope.x1);
});
</script>
```

AngularJS Application

It is time to create a real AngularJS Application.

Make a Shopping List

Lets use some of the AngularJS features to make a shopping list, where one can add or remove items:

My Shopping List

- Milk×
- Bread×
- Cheese×

Add

Application Explained

Step 1. Getting Started:

Start by making an application called `myShoppingList`, and add a controller named `myCtrl` to it.

The controller adds an array named `products` to the current `$scope`.

In the HTML, we use the `ng-repeat` directive to display a list using the items in the array.

Example

So far we have made an HTML list based on the items of an array:

```
<script>
var app = angular.module("myShoppingList", []);
app.controller("myCtrl", function($scope) {
  $scope.products = ["Milk", "Bread", "Cheese"];
});
```

```
</script>
<div ng-app="myShoppingList" ng-controller="myCtrl">
  <ul>
    <li ng-repeat="x in products">{{x}}</li>
  </ul>
</div>
```

Step 2. Adding Items:

In the HTML, add a text field, and bind it to the application with the `ng-model` directive.

In the controller, make a function named `addItem`, and use the value of the `addMe` input field to add an item to the `products` array.

Add a button, and give it an `ng-click` directive that will run the `addItem` function when the button is clicked.

Example

Now we can add items to our shopping list:

```
<script>
var app = angular.module("myShoppingList", []);
app.controller("myCtrl", function($scope) {
  $scope.products = ["Milk", "Bread", "Cheese"];
  $scope.addItem = function () {
    $scope.products.push($scope.addMe);
  }
});
</script>
<div ng-app="myShoppingList" ng-controller="myCtrl">
  <ul>
    <li ng-repeat="x in products">{{x}}</li>
  </ul>
  <input ng-model="addMe">
  <button ng-click="addItem()">Add</button>
</div>
```

Step 3. Removing Items:

We also want to be able to remove items from the shopping list.

In the controller, make a function named `removeItem`, which takes the index of the item you want to remove, as a parameter.

In the HTML, make a `<span>` element for each item, and give them an `ng-click` directive which calls the `removeItem` function with the current `$index`.

Example

Now we can remove items from our shopping list:

```
<script>
var app = angular.module("myShoppingList", []);
app.controller("myCtrl", function($scope) {
  $scope.products = ["Milk", "Bread", "Cheese"];
  $scope.addItem = function () {
    $scope.products.push($scope.addMe);
  }
  $scope.removeItem = function (x) {
    $scope.products.splice(x, 1);
  }
});
</script>
<div ng-app="myShoppingList" ng-controller="myCtrl">
  <ul>
    <li ng-repeat="x in products">
      {{x}}<span ng-click="removeItem($index)">&times;</span>
    </li>
  </ul>
  <input ng-model="addMe">
  <button ng-click="addItem()">Add</button>
</div>
```

Step 4. Error Handling:

The application has some errors, like if you try to add the same item twice, the application crashes. Also, it should not be allowed to add empty items.

We will fix that by checking the value before adding new items.

In the HTML, we will add a container for error messages, and write an error message when someone tries to add an existing item.

Example

A shopping list, with the possibility to write error messages:

```
<script>
var app = angular.module("myShoppingList", []);
app.controller("myCtrl", function($scope) {
  $scope.products = ["Milk", "Bread", "Cheese"];
  $scope.addItem = function () {
    $scope.errortext = "";
    if (!$scope.addMe) {return;}
    if ($scope.products.indexOf($scope.addMe) == -1) {
      $scope.products.push($scope.addMe);
    } else {
      $scope.errortext = "The item is already in your shopping list.";
    }
  }
  $scope.removeItem = function (x) {
    $scope.errortext = "";
    $scope.products.splice(x, 1);
  }
});
</script>
<div ng-app="myShoppingList" ng-controller="myCtrl">
  <ul>
    <li ng-repeat="x in products">
      {{x}}<span ng-click="removeItem($index)">&times;</span>
    </li>
  </ul>
  <input ng-model="addMe">
  <button ng-click="addItem()">Add</button>
  <p>{{errortext}}</p>
</div>
```

Step 5. Design:

The application works, but could use a better design. We suggest to use the proper stylesheet to style the application.

Complete Application

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<link rel="stylesheet" href="style.css">
<body>
<script>
var app = angular.module("myShoppingList", []);
app.controller("myCtrl", function($scope) {
  $scope.products = ["Milk", "Bread", "Cheese"];
```

```
    $scope.addItem = function () {
        $scope.errortext = "";
        if (!$scope.addMe) {return;}
        if ($scope.products.indexOf($scope.addMe) == -1) {
            $scope.products.push($scope.addMe);
        } else {
            $scope.errortext = "The item is already in your shopping list.";
        }
    }
    $scope.removeItem = function (x) {
        $scope.errortext = "";
        $scope.products.splice(x, 1);
    }
});
</script>
<div ng-app="myShoppingList" ng-cloak ng-controller="myCtrl" class="w3-card-2 w3-
margin" style="max-width:400px;">
  <header class="w3-container w3-light-grey w3-padding-16">
    <h3>My Shopping List</h3>
  </header>
  <ul class="w3-ul">
    <li ng-repeat="x in products" class="w3-padding-16">{{x}}<span ng-
click="removeItem($index)" style="cursor:pointer;" class="w3-right w3-margin-
right">×</span></li>
  </ul>
  <div class="w3-container w3-light-grey w3-padding-16">
    <div class="w3-row w3-margin-top">
      <div class="w3-col s10">
        <input placeholder="Add shopping items here" ng-model="addMe" class="w3-input
w3-border w3-padding">
      </div>
      <div class="w3-col s2">
        <button ng-click="addItem()" class="w3-btn w3-padding w3-green">Add</button>
      </div>
    </div>
    <p class="w3-text-red">{{errortext}}</p>
  </div>
</div>
</body>
</html>
```