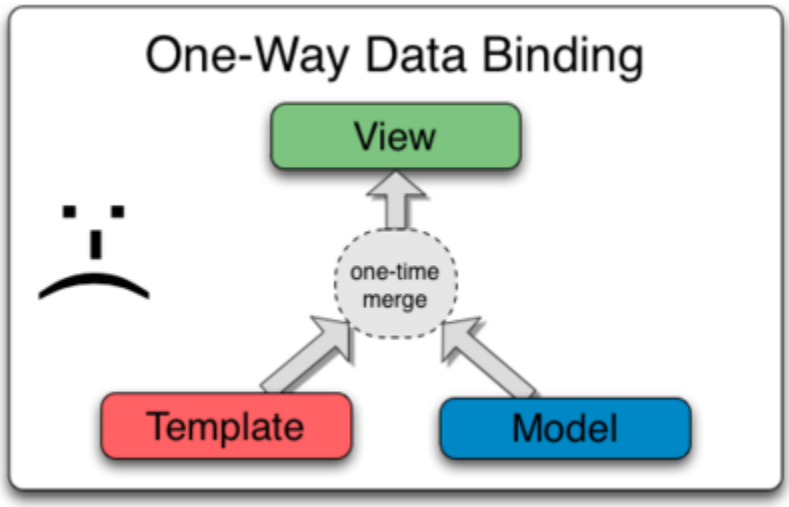
AngularJs Directives:

https://www.w3schools.com/angular/angular_directives.asp

Data Binding

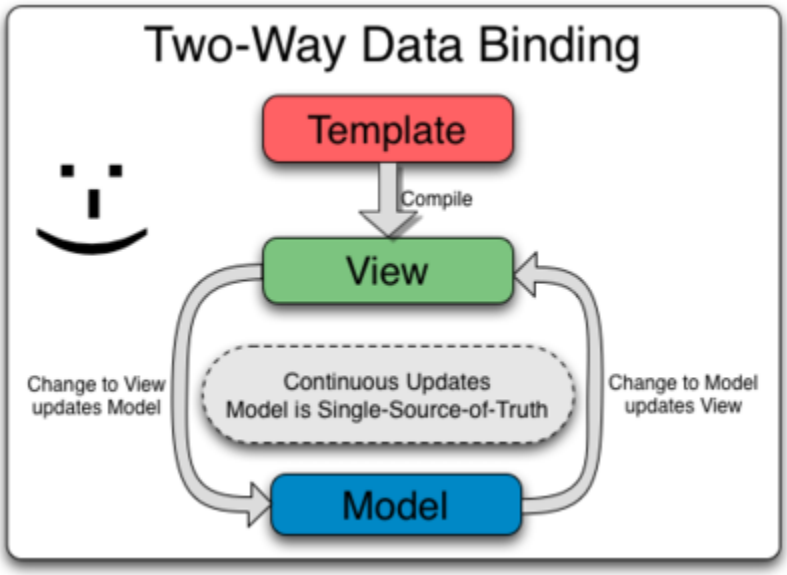
Data-binding in AngularJS apps is the automatic synchronization of data between the model and view components. The way that AngularJS implements data-binding lets you treat the model as the single-source-of-truth in your application. The view is a projection of the model at all times. When the model changes, the view reflects the change, and vice versa.

Data Binding in Classical Template Systems



Most templating systems bind data in only one direction: they merge template and model components together into a view. After the merge occurs, changes to the model or related sections of the view are NOT automatically reflected in the view. Worse, any changes that the user makes to the view are not reflected in the model. This means that the developer has to write code that constantly syncs the view with the model and the model with the view.

Data Binding in AngularJS Templates



AngularJS templates work differently. First the template (which is the uncompiled HTML along with any additional markup or directives) is compiled on the browser. The compilation step produces a live view. Any changes to the view are immediately reflected in the model, and any changes in the model are propagated to the view. The model is the single-source-of-truth for the application state, greatly simplifying the programming model for the developer. You can think of the view as simply an instant projection of your model.

Because the view is just a projection of the model, the controller is completely separated from the view and unaware of it. This makes testing a snap because it is easy to test your controller in isolation without the view and the related DOM/browser dependency.

Controllers in AngularJS:

In AngularJS, a Controller is defined by a JavaScript **constructor function** that is used to augment the AngularJS Scope.

Controllers can be attached to the DOM in different ways. For each of them, AngularJS will instantiate a new Controller object, using the specified Controller's **constructor function**:

- the ngController directive. A new **child scope** will be created and made available as an injectable parameter to the Controller's constructor function as $scope.
- a route controller in a $route definition.
- the controller of a regular directive, or a component directive.

If the controller has been attached using the controller as syntax then the controller instance will be assigned to a property on the scope.

Use controllers to:

- Set up the initial state of the $scope object.
- Add behavior to the $scope object.

Do not use controllers to:

- Manipulate DOM — Controllers should contain only business logic. Putting any presentation logic into Controllers significantly affects its testability. AngularJS has databinding for most cases and directives to encapsulate manual DOM manipulation.
- Format input — Use AngularJS form controls instead.
- Filter output — Use AngularJS filters instead.
- Share code or state across controllers — Use AngularJS services instead.
- Manage the life-cycle of other components (for example, to create service instances).

In general, a Controller shouldn't try to do too much. It should contain only the business logic needed for a single view.

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 = "John";
  $scope.lastName = "Doe";
});
</script>
```

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 = "John";

  $scope.lastName = "Doe";

  $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>
```

Forms in Angular:

Controls (input, select, textarea) are ways for a user to enter data. A Form is a collection of controls for the purpose of grouping related controls together.

Form and controls provide validation services, so that the user can be notified of invalid input before submitting a form. This provides a better user experience than server-side validation alone because the user gets instant feedback on how to correct the error. Keep in mind that while client-side validation plays an important role in providing good user experience, it can easily be circumvented and thus can not be trusted. Server-side validation is still necessary for a secure application.

The key directive in understanding two-way data-binding is ngModel. The ngModel directive provides the two-way data-binding by synchronizing the model to the view, as well as view to the model. In addition it provides an API for other directives to augment its behavior.

Index.html

```
<div ng-controller="ExampleController">
 <form novalidate class="simple-form">
  <label>Name: <input type="text" ng-model="user.name" /></label><br />
  <label>E-mail: <input type="email" ng-model="user.email" /></label><br />
  Best Editor: <label><input type="radio" ng-model="user.preference" value="vi" />vi</label>
  <label><input type="radio" ng-model="user.preference" value="emacs" />emacs</label><br
/>
  <input type="button" ng-click="reset()" value="Reset" />
  <input type="submit" ng-click="update(user)" value="Save" />
```

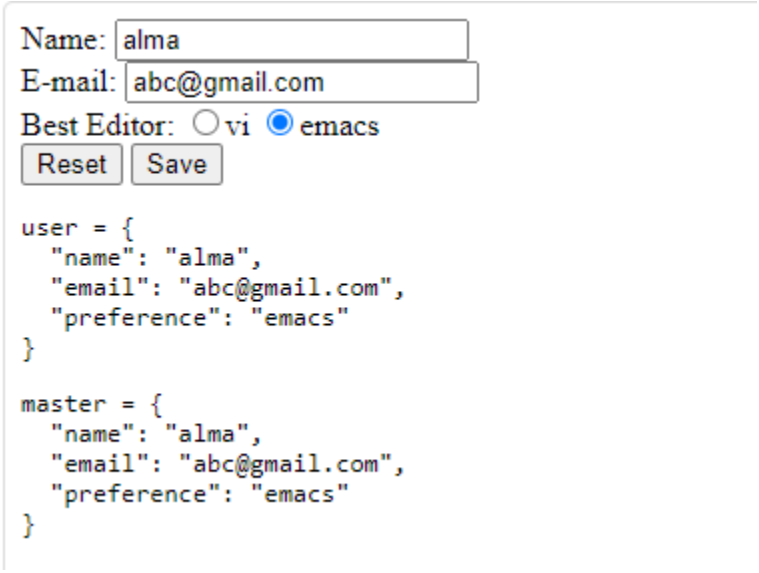
```
 </form>
 <pre>user = {{user | json}}</pre>
 <pre>master = {{master | json}}</pre>
</div>

<script>
 angular.module('formExample', [])
   .controller('ExampleController', ['$scope', function($scope) {
     $scope.master = {};

     $scope.update = function(user) {
       $scope.master = angular.copy(user);
     };

     $scope.reset = function() {
       $scope.user = angular.copy($scope.master);
     };

     $scope.reset();
   }]);
</script>
```



How to use JSON Array into Angular

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html>

<head>

  <title></title>

</head>

<body>

  <script type="text/javascript"
src="//ajax.googleapis.com/ajax/libs/angularjs/1.3.9/angular.min.js"></script>

  <script type="text/javascript">

    var app = angular.module('MyApp', <>)

        app.controller('MyController', function ($scope) {

        $scope.IsVisible = false;

        $scope.GenerateTable = function () {

        $scope.Customers = <

        { CustomerId: 1, Name: "John Hammond", Country: "United States" },

        { CustomerId: 2, Name: "Mudassar Khan", Country: "India" },

        { CustomerId: 3, Name: "Suzanne Mathews", Country: "France" },

        { CustomerId: 4, Name: "Robert Schidner", Country: "Russia" }
```

```
        >;
        $scope.IsVisible = true;
        };
        });
    </script>
    <div ng-app="MyApp" ng-controller="MyController">
      <input type="button" value="Generate Table" ng-click="GenerateTable()" />
      <hr />
      <table cellpadding="0" cellspacing="0" ng-show="IsVisible">
        <tr>
          <th>Customer Id</th>
          <th>Name</th>
          <th>Country</th>
        </tr>
        <tbody ng-repeat="m in Customers">
          <tr>
            <td>{{m.CustomerId}}</td>
            <td>{{m.Name}}</td>
            <td>{{m.Country}}</td>
          </tr>
        </tbody>
      </table>
    </div>
</body>
</html>
```

Handling events in AngularJS

1. Display the list of technologies in a table
2. Provide the ability to ike and dislike a technology
3. Increment the likes and dislikes when the respective buttons are clicked

Script.js : In the controller function we have 2 methods to increment likes and dislikes. Both the functions have the technology object that we want to like or dislike as a parameter.

```
var app = angular
        .module("myModule", <>)
        .controller("myController", function ($scope) {

            var technologies = [
                { name:"C#", likes:0, dislikes:0 },
                { name: "ASP.NET", likes: 0, dislikes: 0 },
                { name: "SQL", likes: 0, dislikes: 0 },
                { name: "AngularJS", likes: 0, dislikes: 0 }
            ];

            $scope.technologies = technologies;

            $scope.incrementLikes = function (technology) {
                technology.likes++;
            };

            $scope.incrementDislikes = function (technology) {
                technology.dislikes++;
            };
        });
```

HtmlPage1.html : Notice in the html below, we are associating incrementLikes() and incrementDislikes() functions with the respective button. When any of these buttons are clicked, the corresponsing technology object is automatically passed to the function, and the likes or dislikes property is incremented depending on which button is clicked.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title></title>
  <script src="Scripts/angular.min.js"></script>
  <script src="Scripts/Script.js"></script>
  <link href="Styles.css" rel="stylesheet" />
</head>
<body ng-app="myModule">
  <div ng-controller="myController">
    <table>
      <thead>
        <tr>
          <th>Name</th>
          <th>Likes</th>
          <th>Dislikes</th>
          <th>Like/Dislike</th>
        </tr>
      </thead>
      <tbody>
        <tr ng-repeat="technology in technologies">
          <td> {{ technology.name }} </td>
          <td style="text-align:center"> {{ technology.likes }} </td>
          <td style="text-align:center"> {{ technology.dislikes }} </td>
          <td>
            <input type="button" ng-click="incrementLikes(technology)" value="Like" />
            <input type="button" ng-click="incrementDislikes(technology)" value="Dislike" />
          </td>
        </tr>
      </tbody>
    </table>
  </div>
</body>
</html>
```

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.

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:

```
<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>
```