B

Vincent Farah

Abstract

<https://app.pluralsight.com/library/courses/angularjs-line-of-business-applications/table-of-contents>

Angular front to back with web api

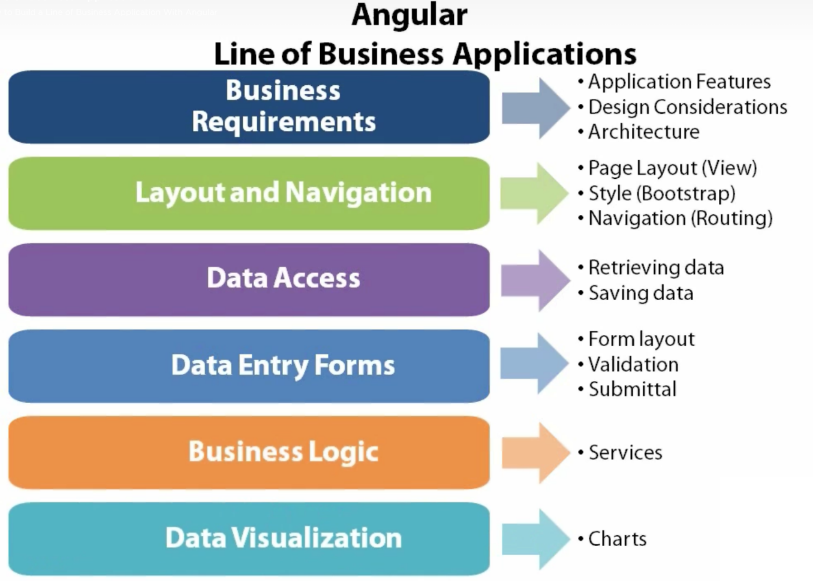
Help keep track of basics and git sample to fork and play with and to use tracking stuff done and keeping a nice summary of important information

# Angular line of business

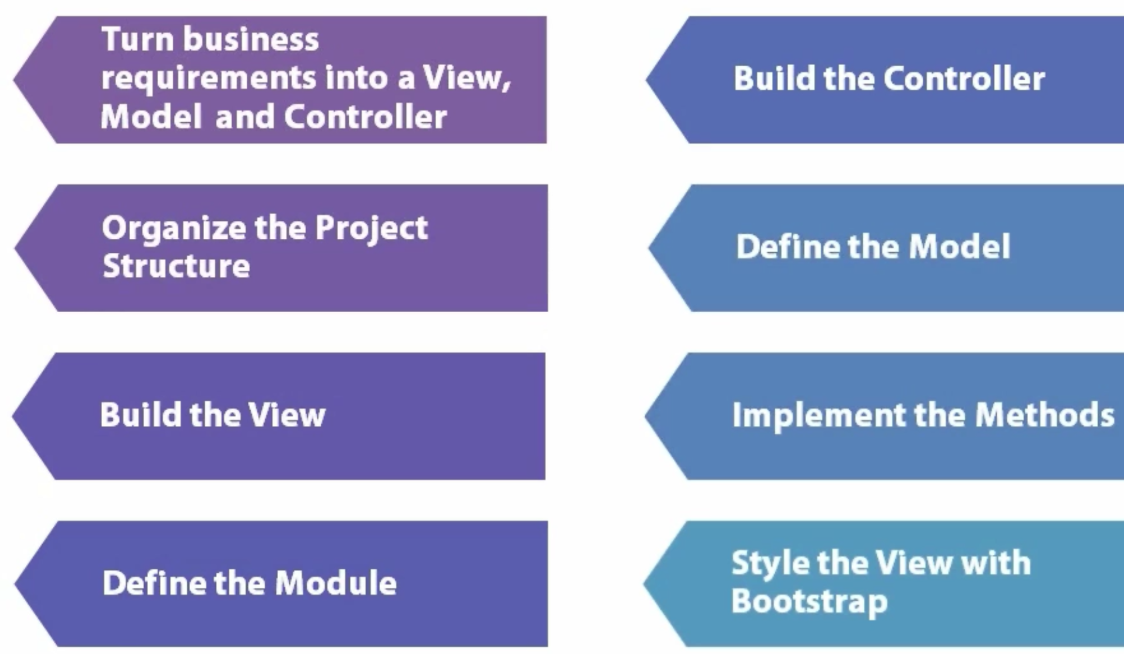
<https://app.pluralsight.com/library/courses/angularjs-line-of-business-applications/table-of-contents>

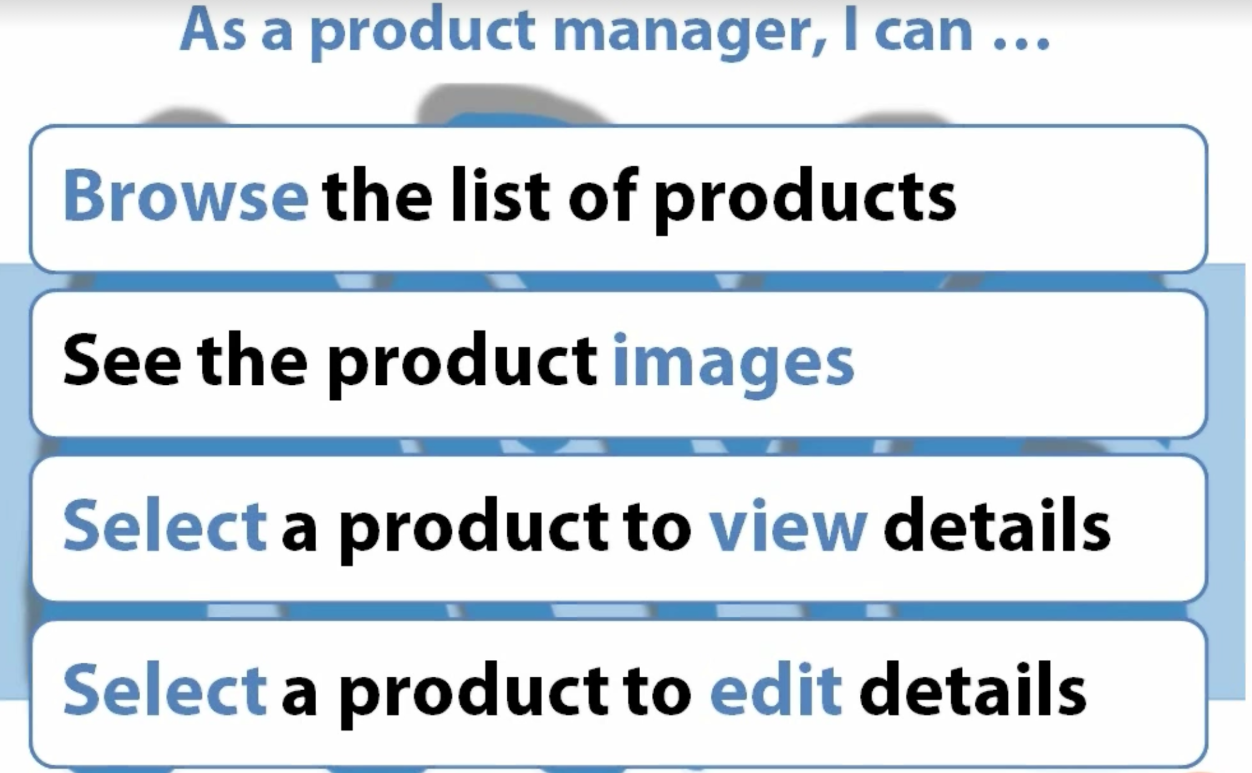
### Why use Angular

* A client side js for building interactive web sites
* Brings simple and clean back to the web site
* Originally developed by google but now open source
* Html is more expressive, easier to read
* Angular is modular, broken up into small units
* Rule based Navigation
* Powerful databinding
* Angular is testable and a focus on separation of concerns means it makes it easier to testing it
* Because it is so popular, it is easy to find help or any resource to help doing this



### Building the first page





### Organising Project Structure

* Create an empty solution
* Install
  + Install-package AngularJS.Core, AngularJS.Resource, Bootstrap, font-awesome, JQuery\*, bootswatch

**<?xml version="1.0" encoding="utf-8"?>**

**<packages>**

**<package id="AngularJS.Core" version="1.5.7" targetFramework="net452" />**

**<package id="AngularJS.Resource" version="1.5.7" targetFramework="net452" />**

**<package id="bootstrap" version="3.3.6" targetFramework="net452" />**

**<package id="Bootswatch" version="3.3.6" targetFramework="net452" />**

**<package id="font-awesome" version="4.6.1" targetFramework="net452" />**

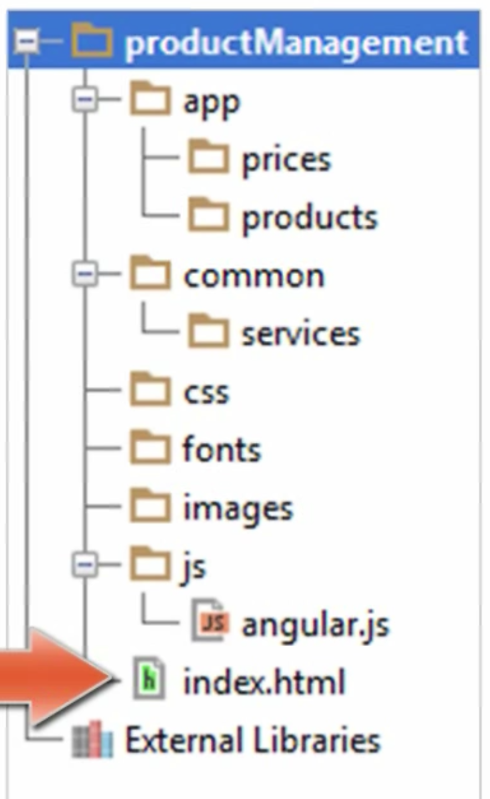
**<package id="jQuery" version="2.2.4" targetFramework="net452" />**

**<package id="jQuery.Validation" version="1.15.0" targetFramework="net452" />**

**<package id="Microsoft.jQuery.Unobtrusive.Validation" version="3.2.3" targetFramework="net452" />**

**<package id="Microsoft.Net.Compilers" version="1.3.0" targetFramework="net452" developmentDependency="true" />**

**</packages>**

* This will be designed by feature
* The app will have two areas, price and products
* Common services like data access will be setup in a common area to be shared
* The rest is self explanatory
* **ng-app** is the most important directive for defining what Angular utilises

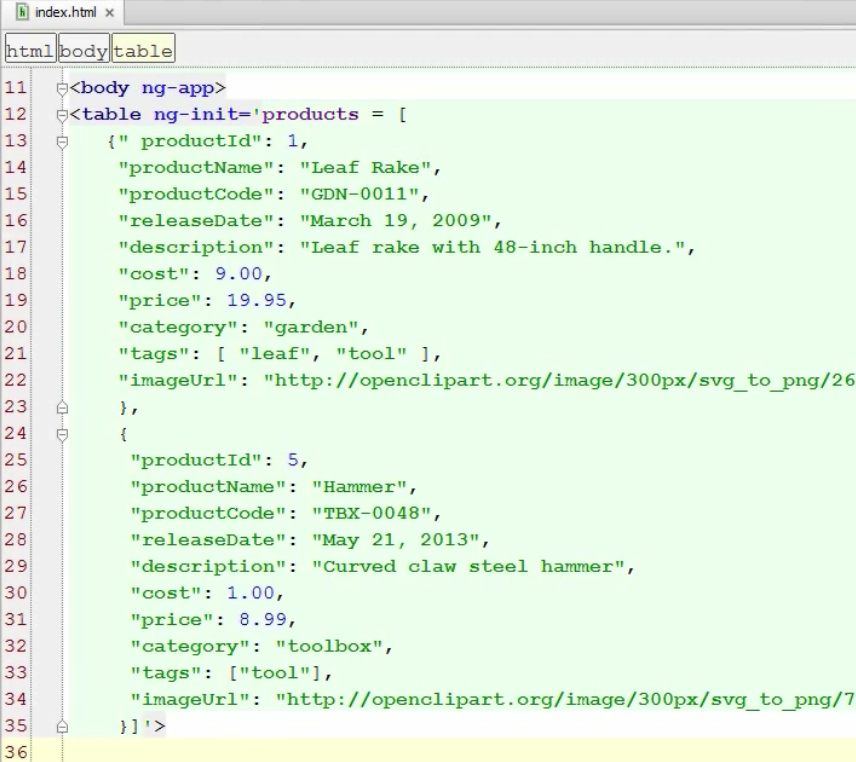
<!DOCTYPE html>

<html>

<head>…</head>

<body **ng-app="productManagement"**

* The **view** is the HTML that defines the visual elements and directives and is called the live template
* When building the view, it is easier to create the mark-up in the html direct and include an **ng-init** with test data

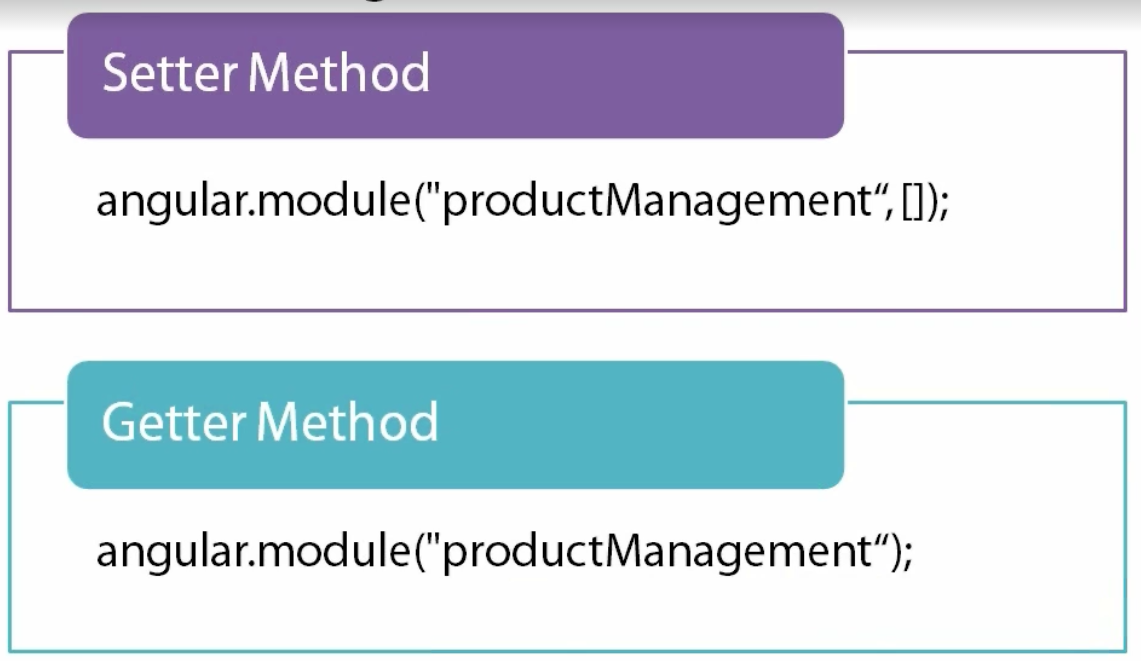




* So the table data can iterate through the data to test the data binding
* Each td is bound to the product data

# Defining the Module

* The Angular Module Method helps to setup the reference to a module



* Assigning this in an IIFE creates this as a self-executing anonymous function and all declarations will not be placed in the global space.

**(function () {**

**'use strict';**

**var app = angular.module('productManagement', [**

**// Angular modules**

**// Custom modules**

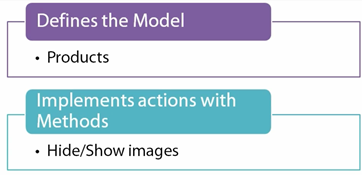
**// 3rd Party Modules**

**]);**

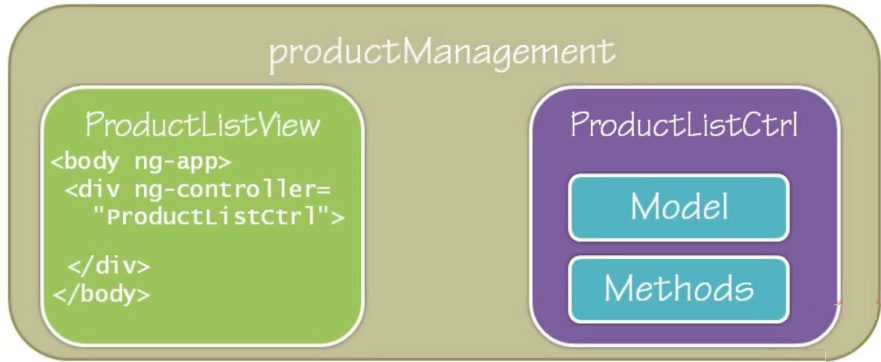
**})();**

# Defining the Controller

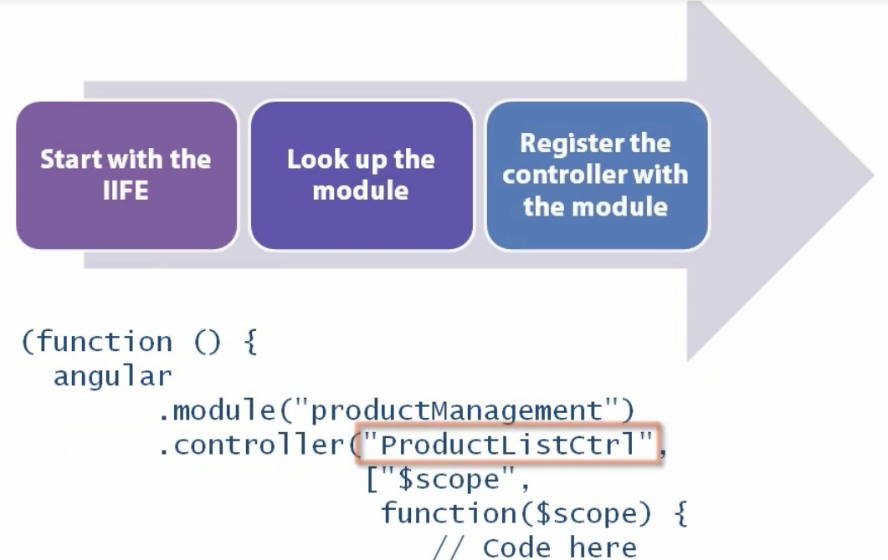
* Defines the model
* Implements any actions



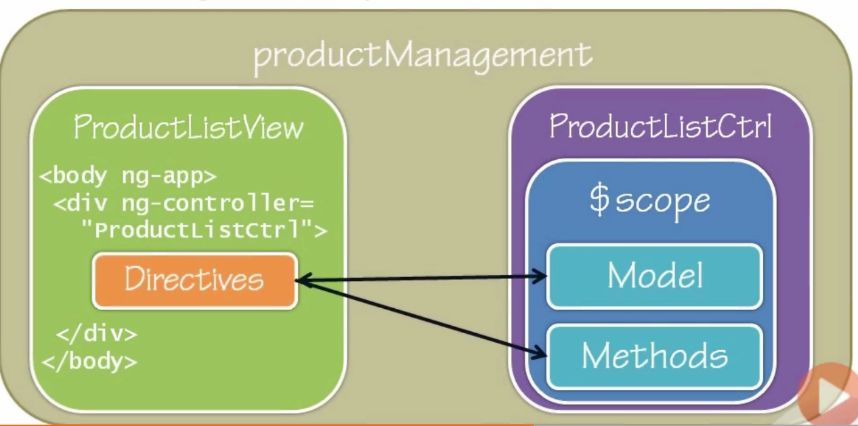
* Controller functions are registered with a module using the **ng-controller** directive



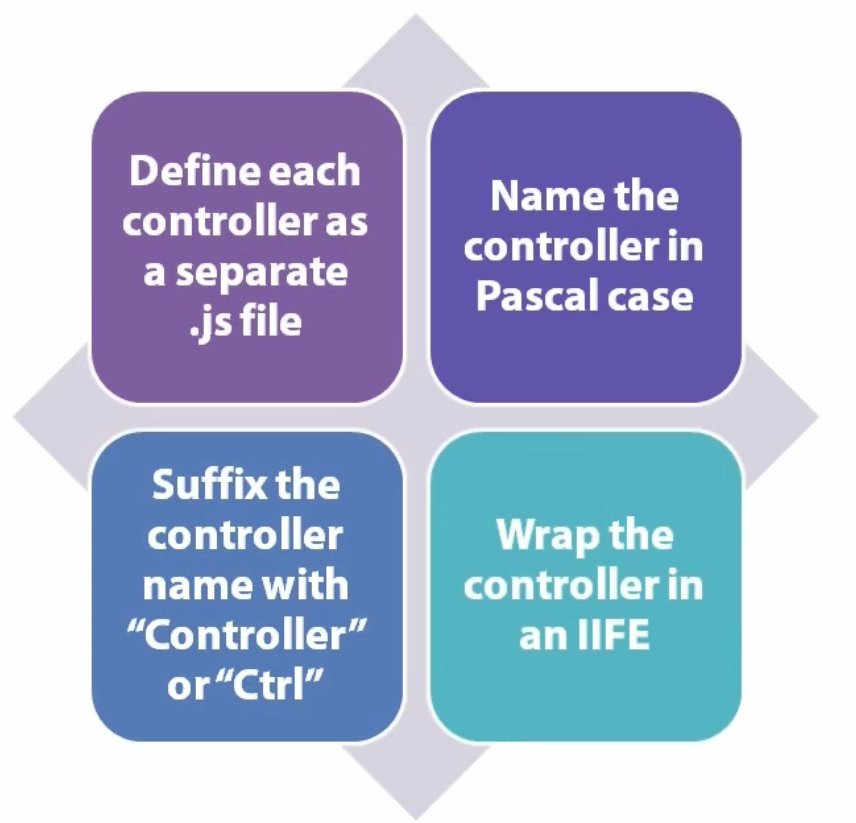
* So the logical order is



* **$scope** is a communication mechanism between the View and the Controller
* Angular injects the scope adding the scope to the list ctrl

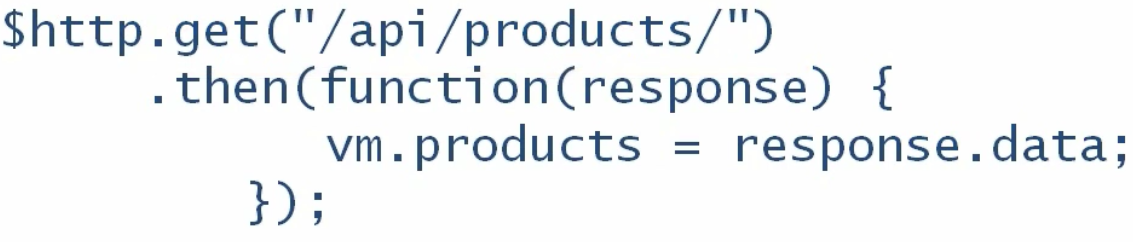


* “Controller As” syntax simplified the $scope handling making it not required as a parameter e.g. ProductListCtrl As vm/product
* The model methods are defined in the controller itself
* The views and the model methods are referenced using an alias defined in the ng-controller
* $scope still exists but it lies behind the scenes
* Controller best practises

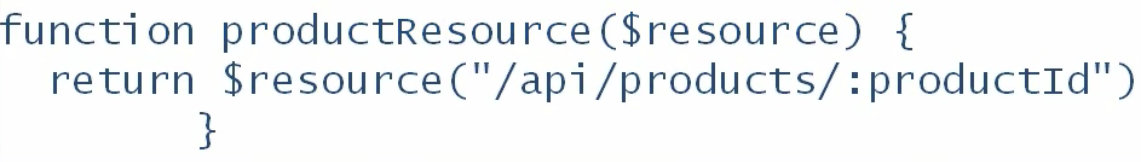


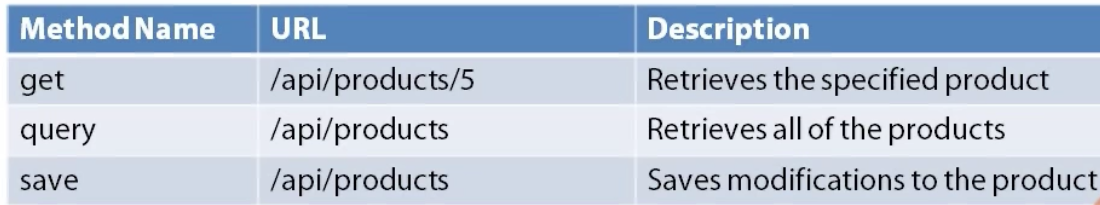
# Built in Angular Services for calling web services

* **$http** facilitates asynchronous communication with remote web servers through a promise



* **$resource** is an *angular factory* that creates a resource or REST object abstracting complexity

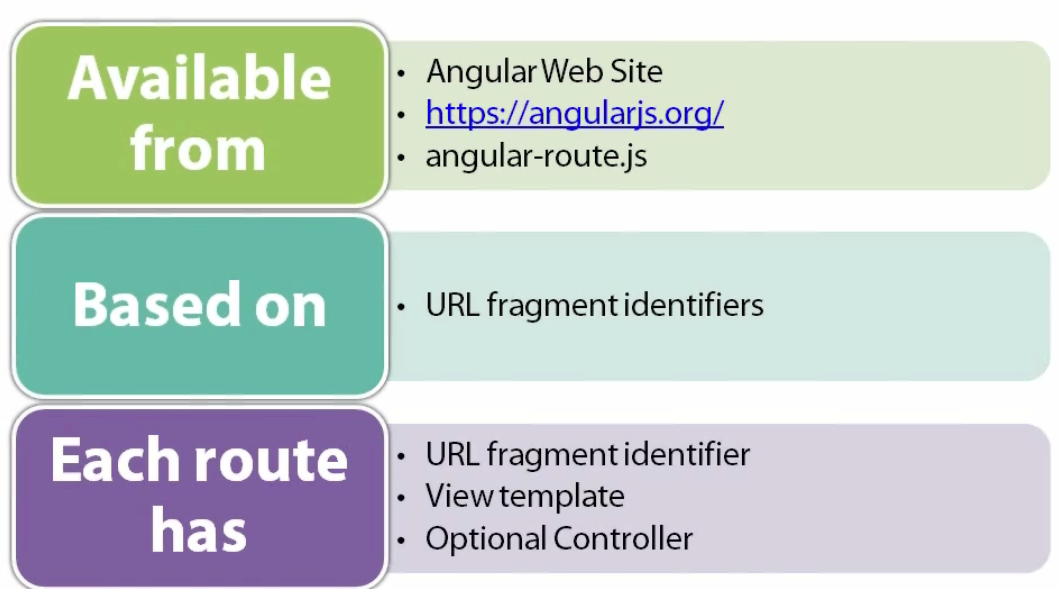




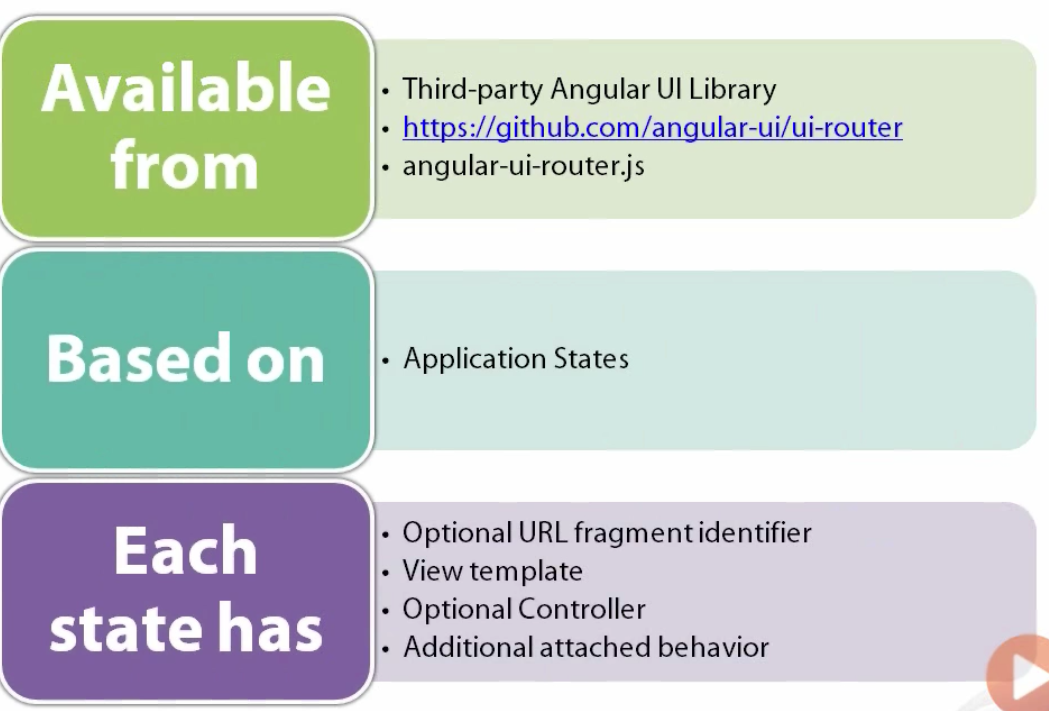
* $**httpBackend** is used to mock this
* Create a **common.services** module to utilise the $resource logic
* Faking the Web Service with **$httpBackend mocking the calls to the webservice**
* Two implementation:
  + **ngMock**: for unit testing applications
  + **ngMockE2E**: for end-to-end testing or backend-less development
* use ngMockE2E [**https://docs.angularjs.org/api/ngMockE2E**](https://docs.angularjs.org/api/ngMockE2E)
* **Download** <https://angularjs.org> or in or case just use Nuget ☺

# Routing to multiple views

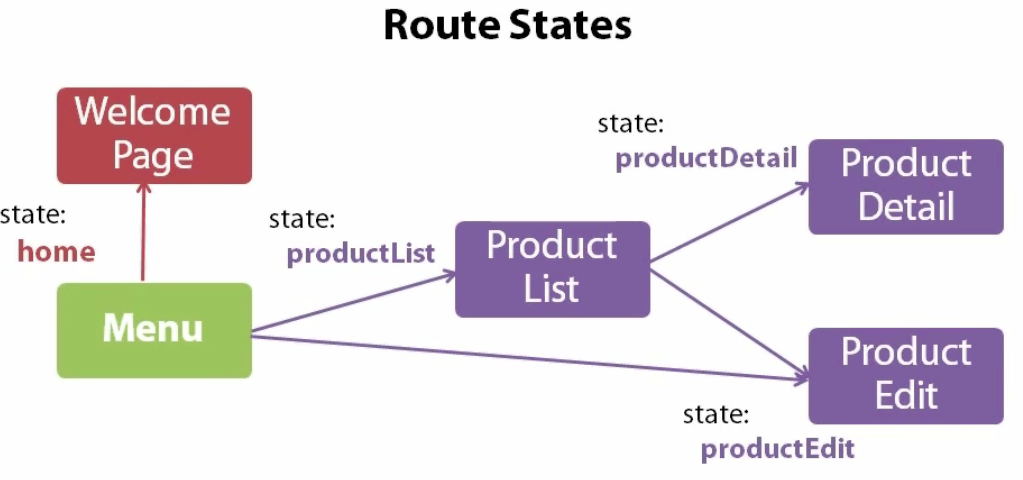
* Angular has two frameworks for doing this.
  + ngRoute



* + UI Router

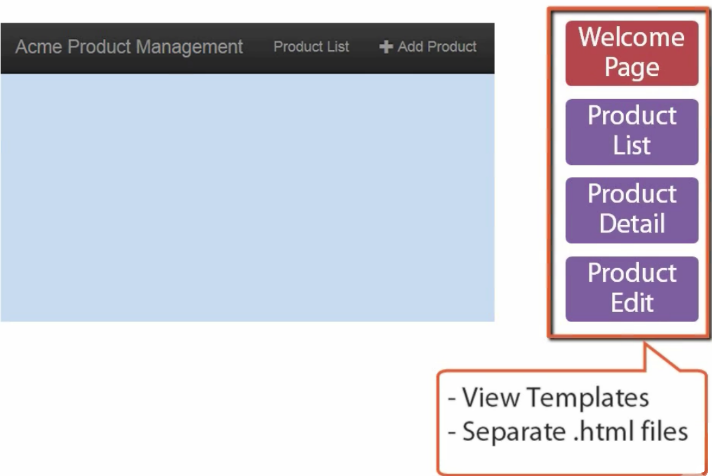


### SiteMap

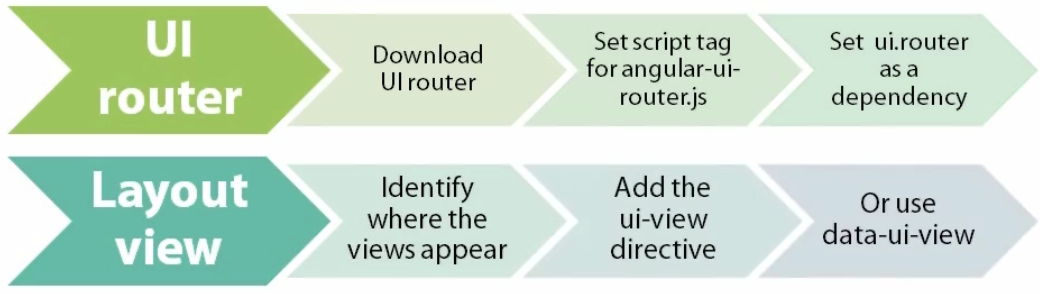


### Layout Template

* The main page of the application is the single page of the SPA
* Each menu option at the top will engage in bringing in something below



* Setting up the route needs some thought and prep



* <https://github.com/angular-ui/ui-router> or <http://angular-ui.github.io/ui-router/release/angular-ui-router.min.js>
  + Add to the app.js all the desired states

**app.config(["$stateProvider",**

**function($stateProvider) {**

**$stateProvider**

**// Products**

**.state("productList", {**

**url: "/products",**

**templateUrl: "app/products/productListView.html",**

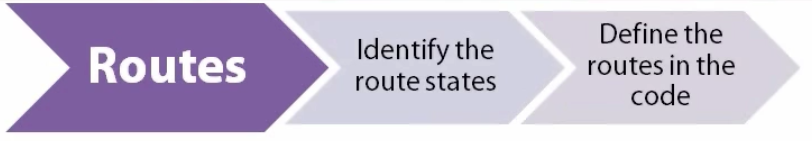
**controller: "ProductListCtl as vm"**

**});**

**}**

**]);**

* Doing the route then



### Defining a default state

* **$urlProvider** Service
* Watches **$location** for changes to the URL
* When **$location** changes, it finds a matching state and then **activates** that **state**
* Used behind the scenes
* Activating a route



### Building a menu with Routing

* Add JQuery and bootstrap.js for menu functionality to work, JQuery first in the list
* To any anchor, add the attribute ui-sref=”STATE”

<a class="navbar-brand" **ui-sref="home"**>Acme Product Management</a>

**.state("home", {**url: "/", templateUrl:"app/welcomeView.html"

* If passing any variables be sure to add the state with a parameter value e.g.

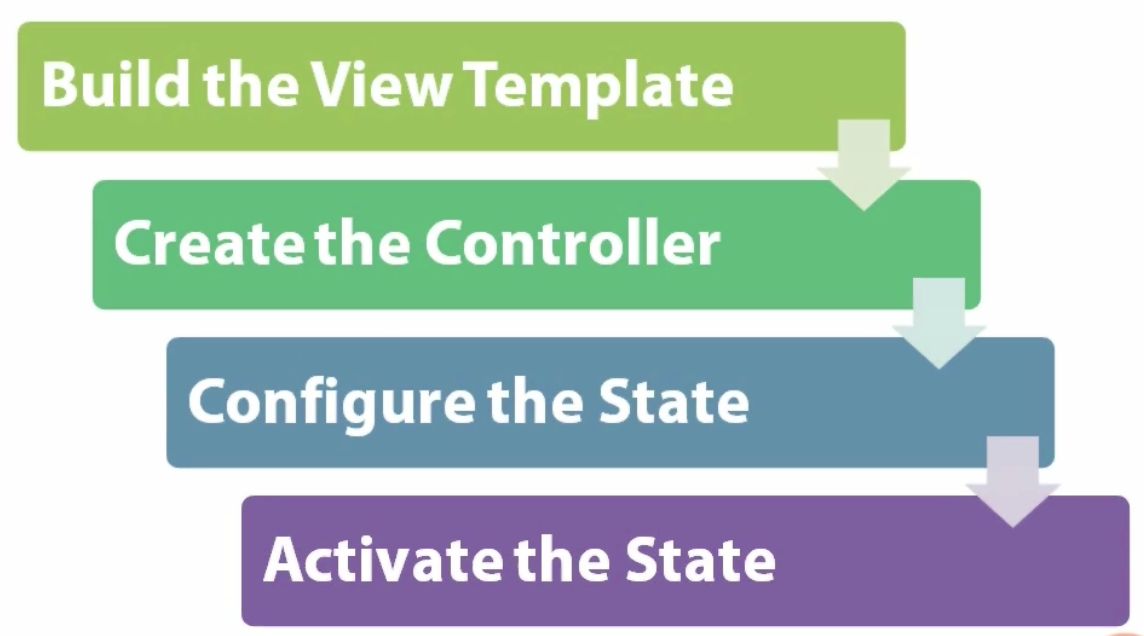
<a **ui-sref="productEdit({productId:0})"**><i class="glyphicon glyphicon-plus"></i>Add Product List</a>

.state("productEdit", {

                   url: "/products/edit/**:productId**",

                   templateUrl: "app/products/productEditView.html",

                   controller: "ProductEditCtl as vm"



* Resolve
  + Resolve is a property that can be attached to a route
  + Resolve can also provide a controller with data
  + Identifies dependencies and are defined wit key/value pairs

.state("productDetail", {

    url: "/products/:productId",

    templateUrl: "app/products/productDetailView.html",

    controller: "ProductDetailCtl as vm",

**resolve**: {

        productResource: "productResource",

        product: function (productResource, $stateParams) {

            var productId = $stateParams.productId;

            return productResource.get({ productId: productId }).$promise;

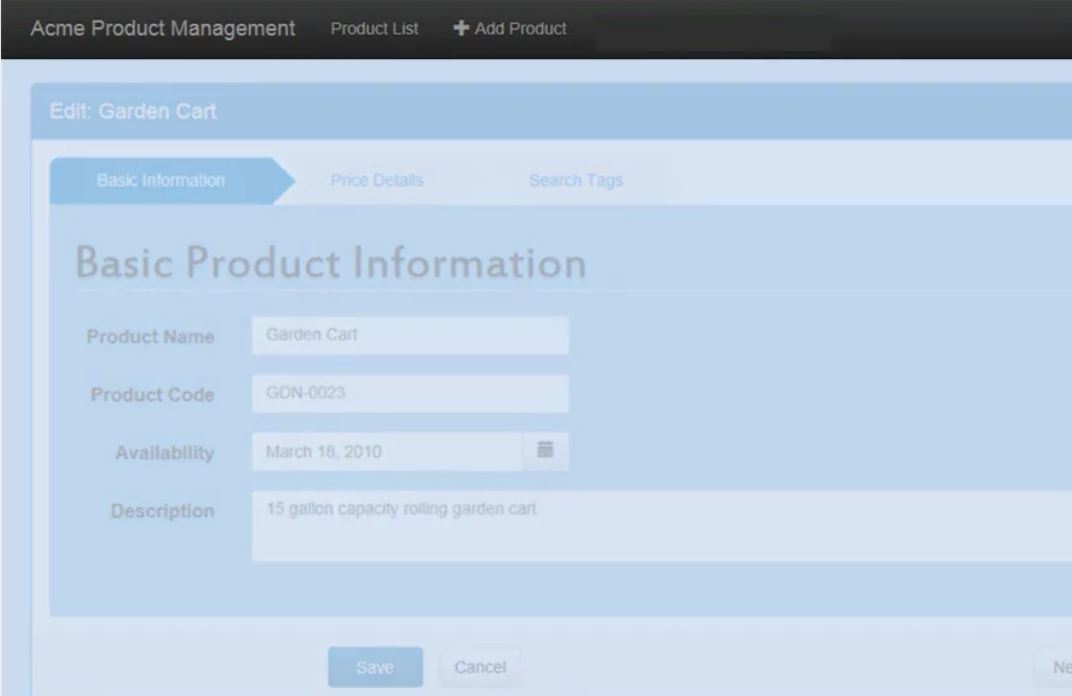
        }

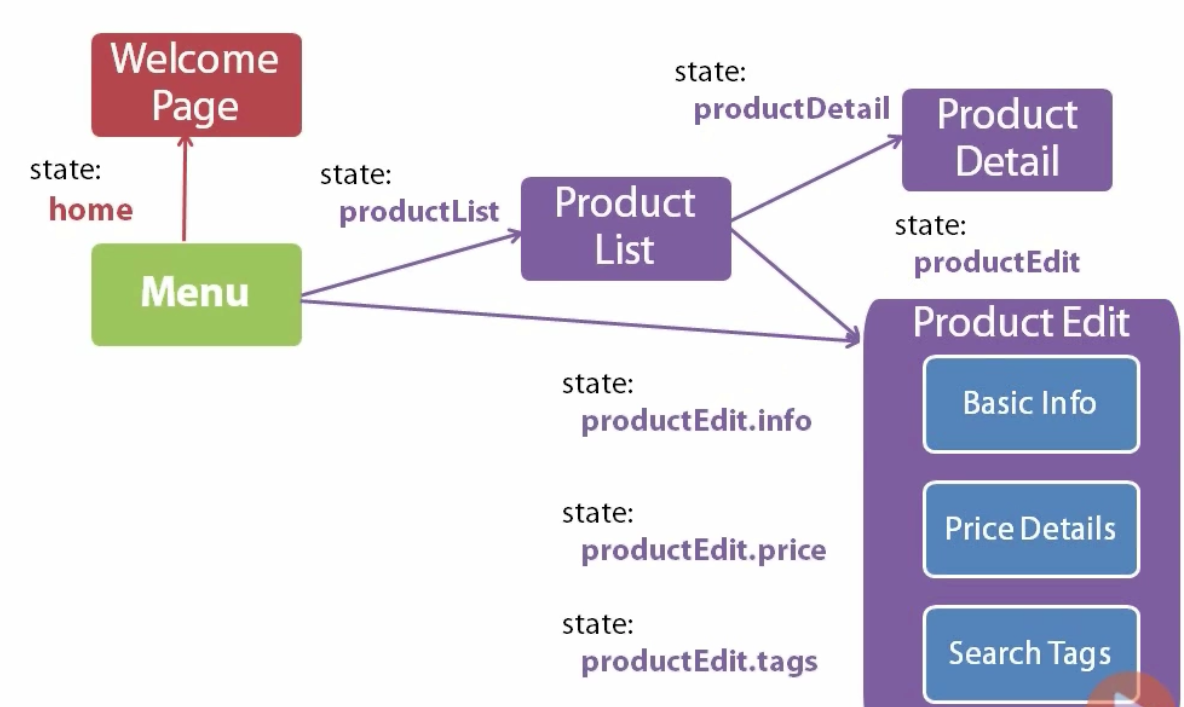
    }

})

### Defining Nested Routing States

* Creating views within views or tabs within the current view





* Define partial product edit substates

.state("productEdit.**info**", {

    url: "/info",

    templateUrl: "app/products/productEditInfoView.html"

})

.state("productEdit.**price**", {

    url: "/price",

    templateUrl: "app/products/productEditInfoView.html"

})

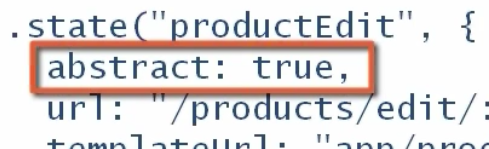
.state("productEdit.**tags**", {

    url: "/tags",

    templateUrl: "app/products/productEditInfoView.html"

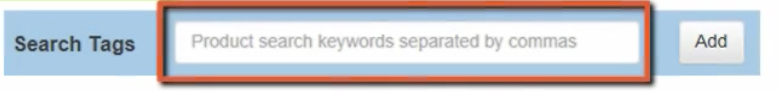
})

* Abstract States
  + Cannot be explicitly activated
  + Activation attempts to throw an exception
  + Activate implicitly when child state is activated



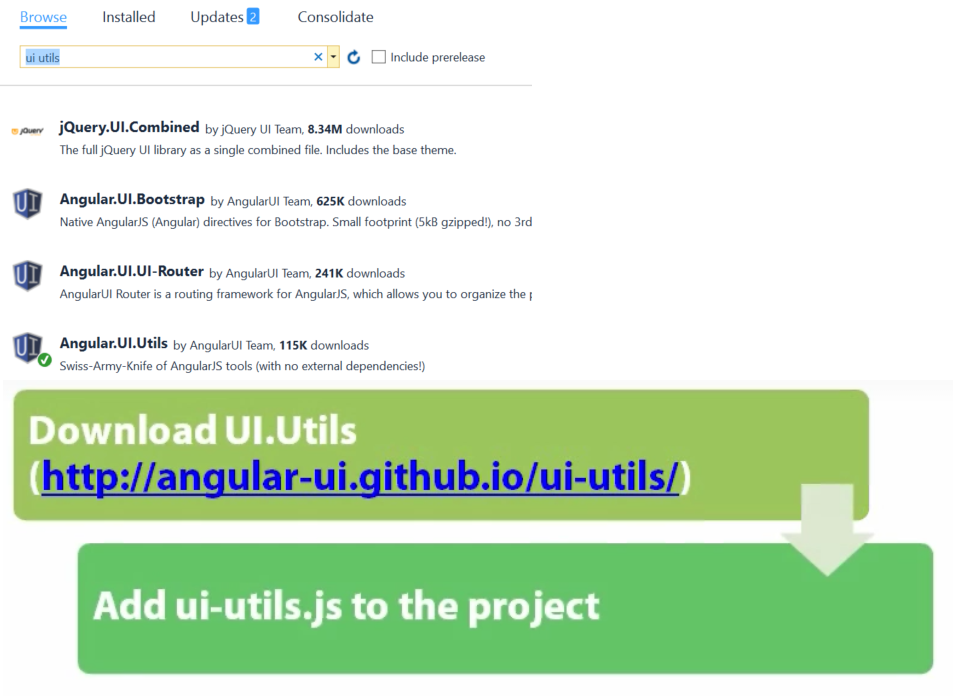
### Building Data Entry Forms

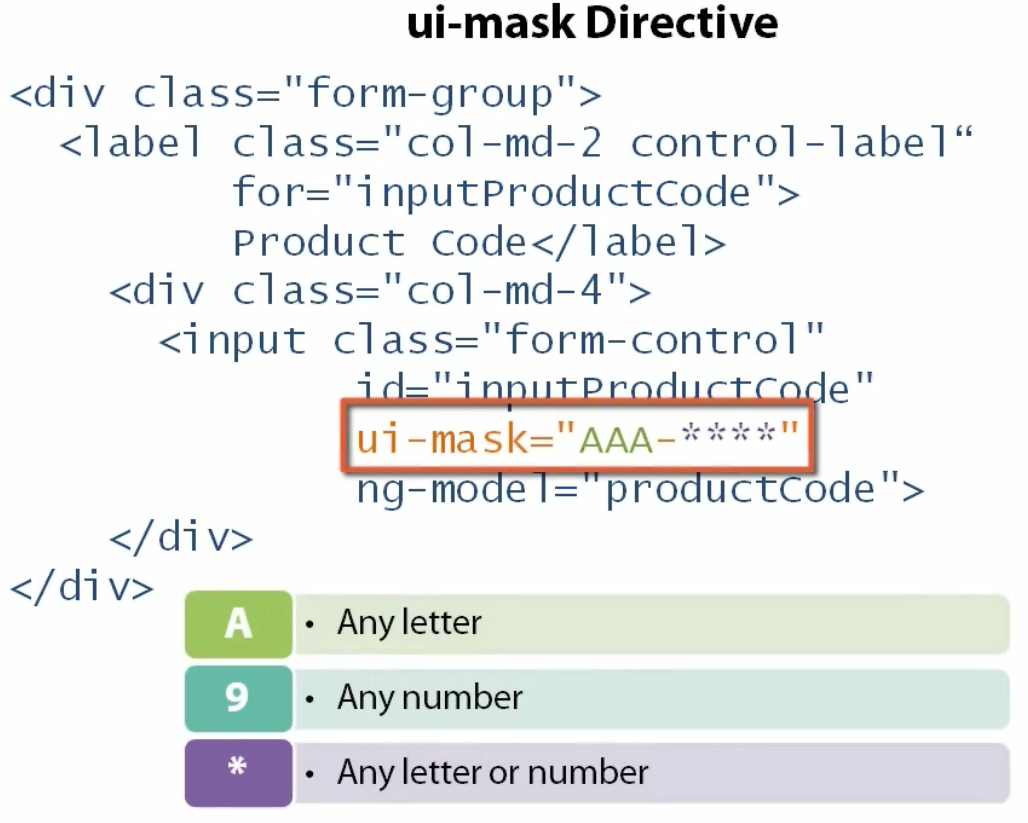
* Facilitate relevant accurate and valid data, and prevent inaccurate, invalid and erroneous data through helpful data entry **controls** and **validation**
* The intent of each label must be clear by using a **placeholder**
* Use the business terminology



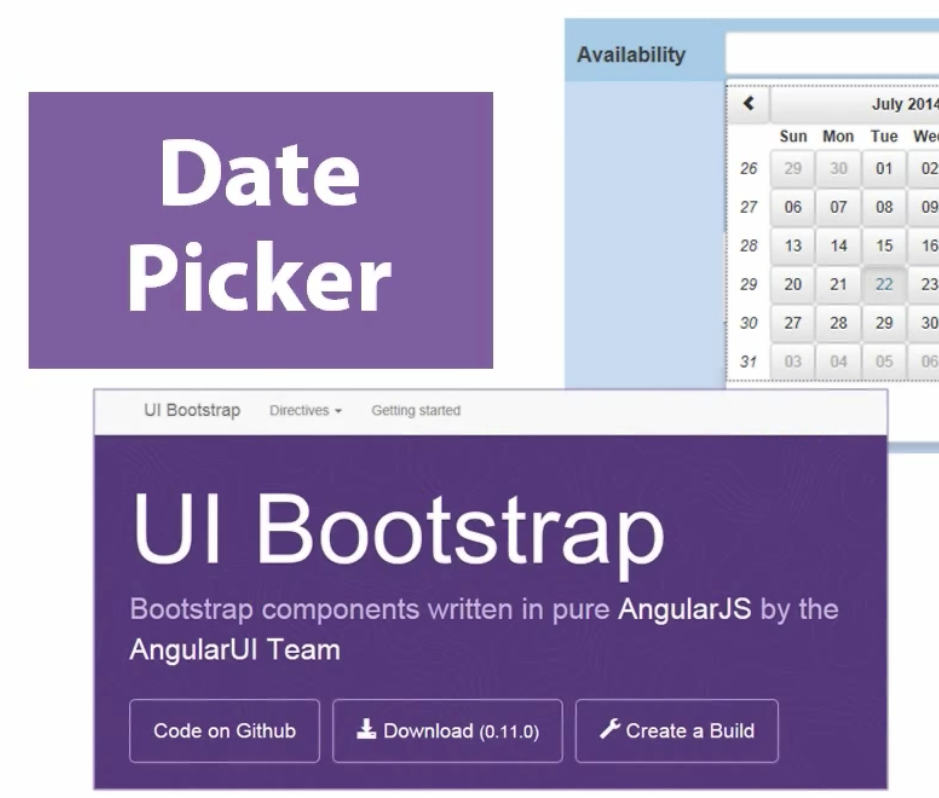
* **Organization** helps with grouping information through the toolbar or with tabs
* **Mask edits** help restrict the text that is added



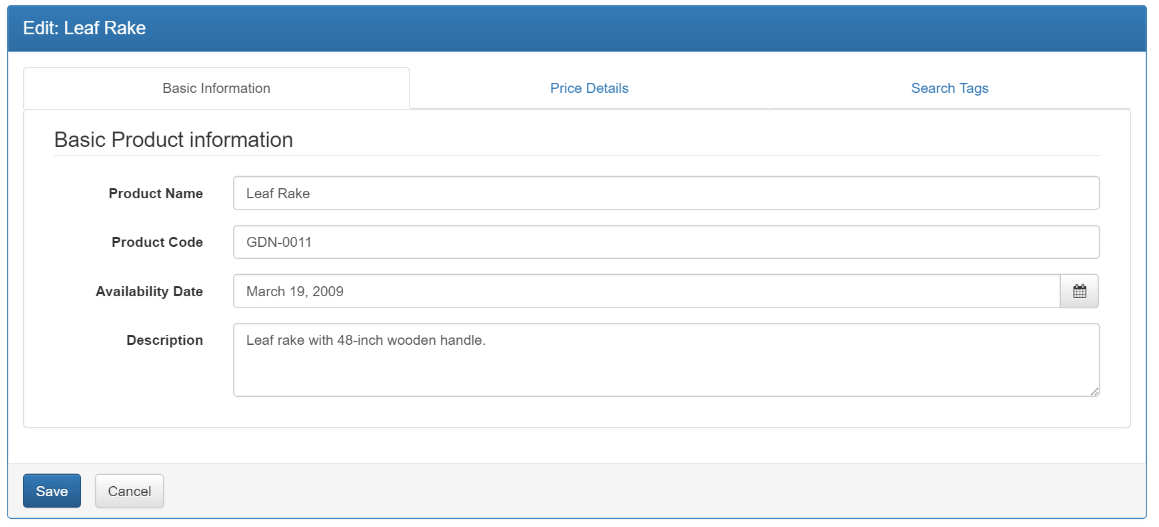




* Creating a form using the form tag and create two way binding with the **ng-model** directive
* Date pickers are great for formatting date controls or download @ <http://angular-ui.github.io/bootstrap/>



* Save a form within form tag

****

**<form class="form-horizontal" name="productForm" role="form">**

**<fieldset>**

**<legend>Basic Product information</legend>**

**</fieldset>**

**<div class="form-group">**

**<label for="inputProductName" class="col-sm-2 control-label">Product Name</label>**

**<div class="col-sm-10">**

**<input id="inputProductName"**

**class="form-control"**

**type="text"**

**placeholder="Product Name"**

**ng-model="vm.product.productName" />**

**</div>**

**</div>**

**<div class="form-group">**

**<label for="inputProductCode" class="col-sm-2 control-label">Product Code</label>**

* Save is an extension of the **$resource** paradigm

***vm.save = function () {***

***vm.product.$save(function(data) {***

***toastr.success("Save Successful");***

***},function(data) {***

***toastr.error("Unable to save because " + data);***

***});***

***};***

### Validation

TODO: <https://app.pluralsight.com/player?course=angularjs-line-of-business-applications&author=deborah-kurata&name=angularjs-lob-m3-data&clip=6&mode=live>