MVC RECAP

Model - Data

View - Presentation

Controller - Application Logic

ANGULAR JS

SETUP

CS498RK SUMMER 2017

UNIVERSITY OF ILLINOIS @ URBANA-CHAMPAIGN

SETUP

https://github.com/krishnadusad/angular_starter

SETUP

partials

routes

controllers

services

SETUP

controller - logic defined for a particular view

SETUP

routes - define templates and controllers for different routes

SETUP

partials - templates you define for different views

SETUP

services/factory - logic/state
shared across the app

SETUP

services/factory - logic/state
shared across the app

SETUP

Example

ANGULAR JS

CONCEPTS

Template	HTML with additional markup
Directives	extend HTML with custom attributes and elements
Model	the data shown to the user in the view and with which the user interacts
Scope	context where the model is stored so that controllers, directives and expressions can access it
Expressions	access variables and functions from the scope
Compiler	parses the template and instantiates directives and expressions
Filter	formats the value of an expression for display to the user
View	what the user sees (the DOM)
Data Binding	sync data between the model and the view
Controller	the business logic behind views
Dependency Injection	Creates and wires objects and functions
Injector	dependency injection container
Module	a container for the different parts of an app including controllers, services, filters, directives which configures the Injector
Service	reusable business logic independent of views

ANGULAR JS

CONCEPTS - PRACTICAL

TEMPLATES

```
<h1> {{heading}} </h1>
<div>
        <h2> Enter your username </h2>
        <input type="text" ng-model="username" >
        <h3> Welcome {{username}} </h3>
</div>
```

EXPRESSIONS

```
{{ expression | filter }}
```

An expression in a template is a JavaScript-like code snippet that allows AngularJS to read and write variables. Note that those variables are not global variables.

SCOPE

```
$scope.myVariable = “”
```

Just like variables in a JavaScript function live in a scope, AngularJS provides a scope for the variables accessible to expressions.

DIRECTIVES

DOM transformation engine that lets you extend HTML's syntax to create your own custom tags, attributes, etc.

{{}}, ng-*: predefined directives that come with Angular

DIRECTIVES

ng-change
ng-model
ng-repeat
ng-click
ng-src
ng-href
ng-hide/ng-show
ng-style
ng-class

FILTERS

```
{{ expression | filter }}
```

A filter formats the value of an expression for display to the user

FILTERS

```
{{ expression | filter }}
```

```
<div>
    <p>The name is {{ lastName | uppercase }}</p>
</div>
```

`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.

SERVICES

logic/state shared across the app

BUILT IN SERVICES

$http – allows you to make http requests

$routeParams – allows you to share route parameters

$rootScope – allows you to share scope

…

Example

ANGULAR JS

CONCEPTS - THEORY

MODULE

Container for the different parts of your app – controllers, services, filters, directives

MODULE

declaration dependencies

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

CONTROLLER

- Constructor function that is used to augment the `$scope`

- Attached to the DOM.

- On instantiation a **child scope** will be created and made available as an injectable parameter to the controller's constructor function

CONTROLLER

injections

```
app.controller(‘myCont’,[‘$scope',function($scope)
{
          console.log("On page 1”);
}]);
```

SERVICES & FACTORIES

- Reusable business logic independent of views.

- Substitutable objects that are wired together using dependency injection.

- Used to organize and share code across your app.

SERVICES

Use to organize and share code across your app.

Think of services as constructors that we run “new” on.

SERVICES

```
app.service(‘myService’,function()
{
        this.sayHello = function (name) {
            return “Hi” + name + “!”;
        }
});
```

FACTORIES

- Very similar to services

- Provides shared code across app

- Think of factories as functions we run and return from

FACTORIES

```
app.factory('myFactory',function()
{
    return {
         sayHello: function (name) {
              return "Hi" + name + "!";
         }
        }
});
```

Example

DEPENDENCY INJECTION OVERVIEW