Frontend Frameworks

SWE 432, Fall 2016

Design and Implementation of Software for the Web

GEORGE MASON UNIVERSITY

Today

- How do we build a single page app without dying?
  - MVC/MVVM
  - (AngularJS)

For further reading:

Book: Learning Javascript Design Patterns, Osmani (Safari books online)

Book: AngularJS in Action, Ruebbelke & Ford (Safari books online)

Book: Learning AngularJS, Williamson (Safari books online)

Only look at references for AngularJS (NOT Angular2)

Demos (source/clone git): https://github.com/gmu-swe432/lecture10demos

Source (run in browser): https://gmu-swe432.github.io/lecture10demos/

Single Page Application

Many challenges…

DOM Manipulation

Browser

events

HTML *HTML elements*

```
function helloWorld() {
      var message = "<h1>Hello, world!</h1>";
    $("body").html();
}
```

Javascript

History Tracking

HTTP Request

HTTP Response (JSON)

Caching

Data Binding

Routing

Web Server

Persistence tier

Loading Views

Handling a ton of async

Database

AngularJS — Superheroic Java

https://angularjs.org

ANGULARJS | Home | Learn | Develop | Discuss | Search

ANGULARJS by Google

HTML enhanced for web apps!

Download AngularJS 1 (1.5.6 / 1.2.30)

Try the new Angular 2

View on GitHub

Design Docs & Notes

Follow +AngularJS on

Follow @angularjs 173K followers

Tweet

Learn Angular in your browser for free!

Angular to the Rescue

- Full-featured SPA framework (can be used for non-SPA sites too!)
- It's full of buzzwords!
  - Data binding, MVC, MVVM, Routing, Testing, jqLite, Templates, History, Factories, Directives, Services, Dependency Injection, Validation, and all of their friends!
- There are other frameworks too, they work fine, but we're focusing on AngularJS:
  - Aurelia
  - Backbone.js
  - Ember.js

Keeping stuff organized… How do we break apart components?

My **V**ery **C**ool Drink Factory (MVC)

The MVC Drink Factory: A recipe

- Requires:
  - 3oz coconut milk
  - 3oz almond milk
  - 2 frozen bananas
  - 2 tbsp peanut butter
  - 1 tbsp agave nectar
- Place all ingredients in blender and blend for 45 seconds
- Serve in a pint glass, garnish with banana slice, cherry, and whipped cream (optional)

The MVC Drink Factory: Abstract

- What makes a drink?
  - Ingredients
  - Glasswear
  - Recipe
- Recipe *controls* the entire process
  - Ingredients make up the content of the drink
  - Glass changes how you see the drink, but not its contents

The MVC Drink Factory

- Can make *other* drinks by changing the ingredients, keeping the steps to follow and the glass
- Can make also keep the ingredients and steps, change the presentation

Same recipe, different presentation

Same recipe, different ingredients

The MVC Modular Drink Factory

- My Very Cool factory separates concerns between recipes, ingredients, and glasses
- Different people can pull out ingredients, follow the recipe, and pour the result into the correct glass without knowing exactly what the other person does
- Could even completely replace how the ingredients are gathered (maybe use pre-portioned), and it doesn't effect the rest of the process

My **V**ery **C**ool Drink Factory

- Wow, this separation of concerns is just what we want in our web apps!
- Because it's so modular, we named an entire design pattern based on this recipe, ingredient, presentation pattern (MVC)
- Alternatively, we might call it **M**odel-**V**iew-**C**ontroller
  - Model: Ingredients
  - Controller: Recipe
  - View: Glass/presentation

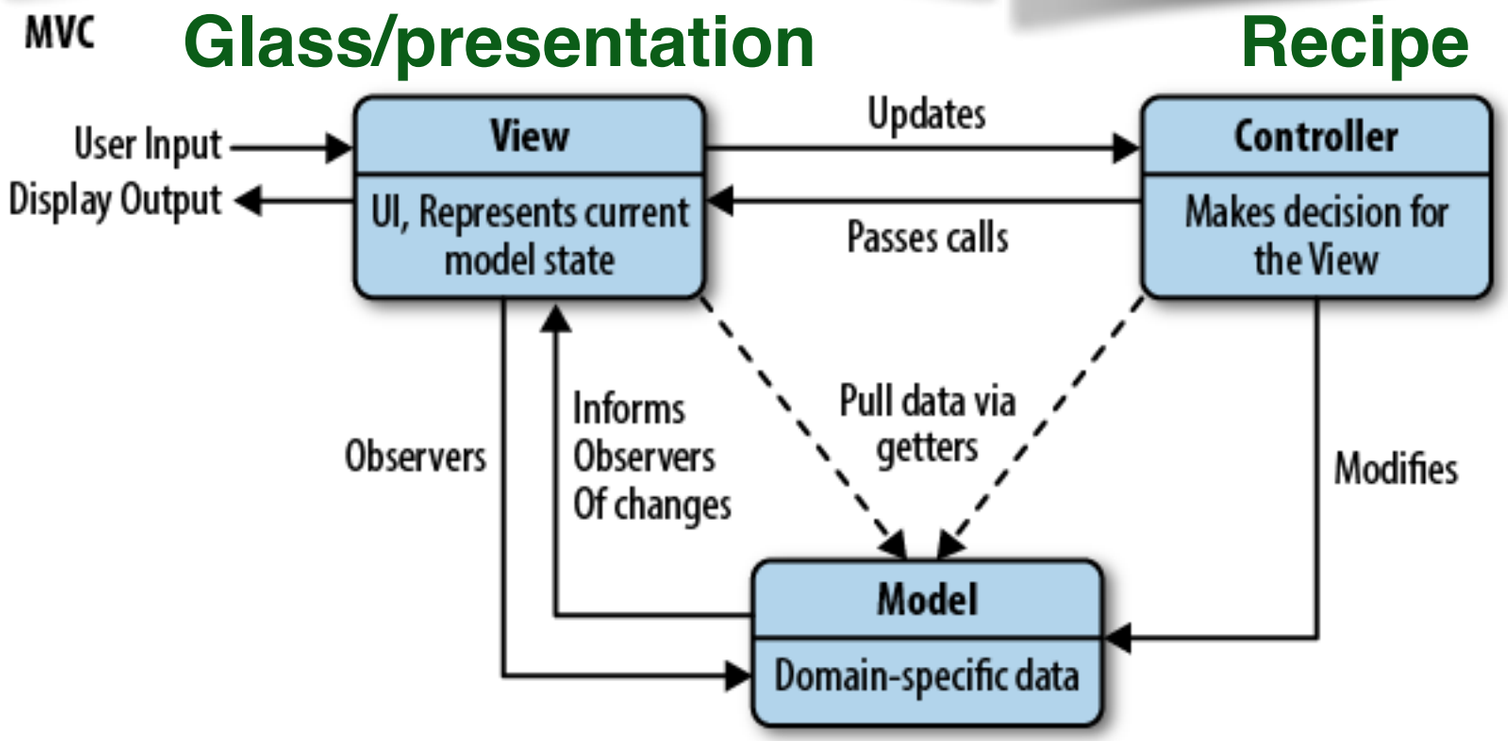
MV* Patterns

- The mother of them all: MVC
  - Originally from 70's: UIs were just becoming possible... how to separate presentation from data and logic?
  - Model: domain-specific data, doesn't matter how it's interacted with
  - View: visual representation of current state of model
  - Controller: Moderates user interactions, makes business decisions
  - *Separation of concerns*

MVC & JavaScript

DOM templates

JS that receives input from DOM, deploys spinner, etc.



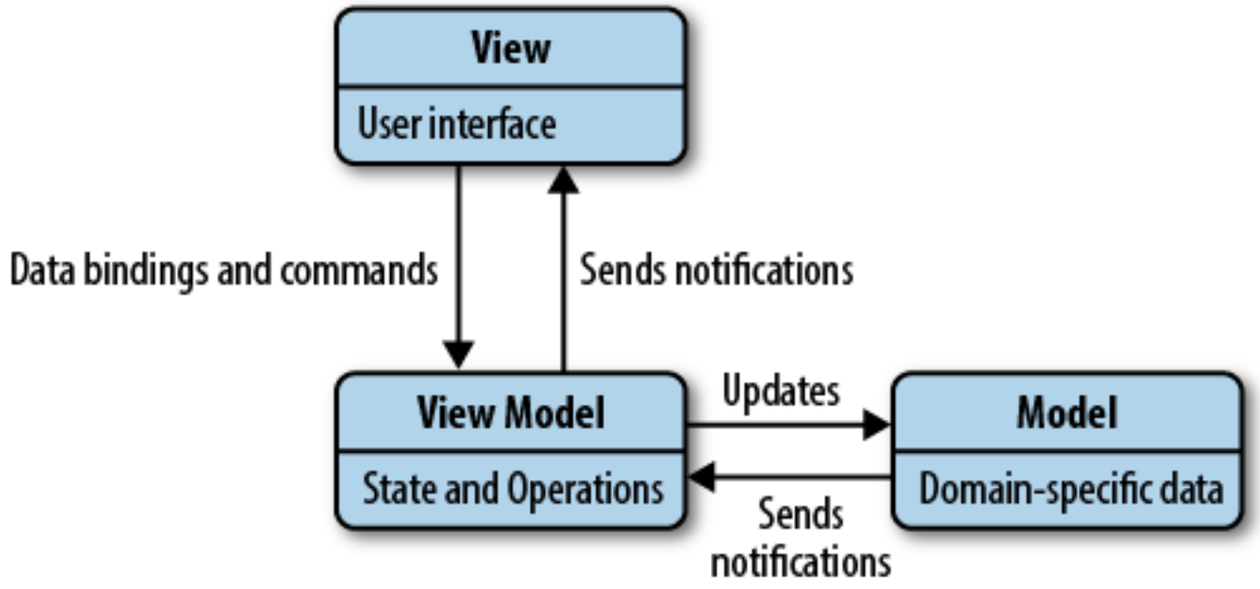
Ingredients

Firebase todoRef list

Firebase callbacks that update view directly

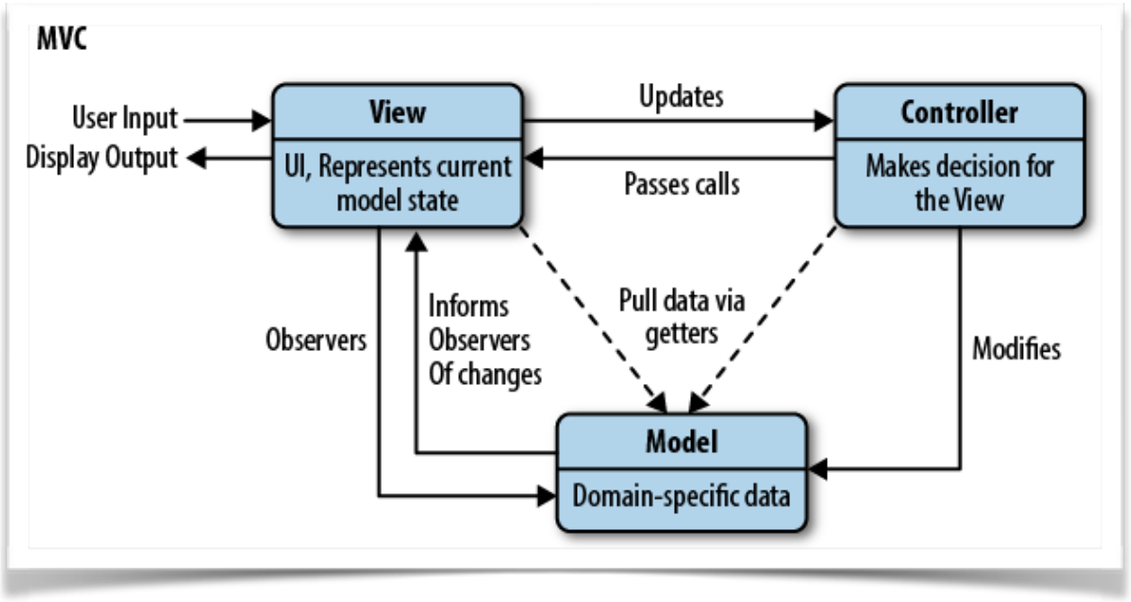
***Note that in drink factory, the glass doesn't care about the ingredients**

MVVM

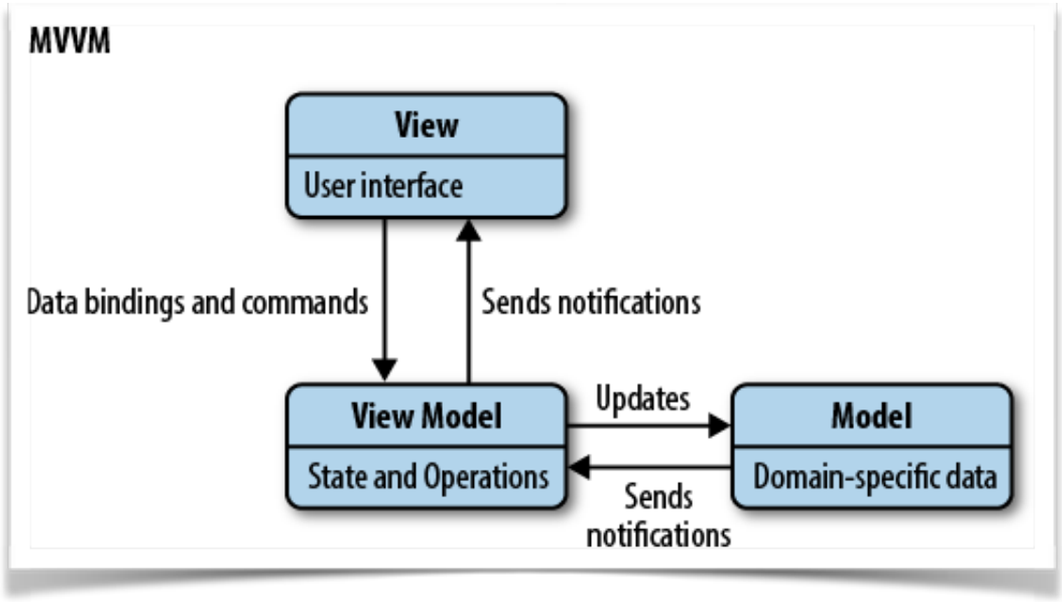


- View does not communicate with model directly
- Models are much more dumb: no formatting, etc
- ViewModel: like a controller from MVC, but only does data translation/formatting between M-V
- More directly maps to MyVeryCool Drink Factory than MVC does

MVC vs MVVM



Low level controller/model code can be easily shared (especially in server apps)
Views can have direct access to model

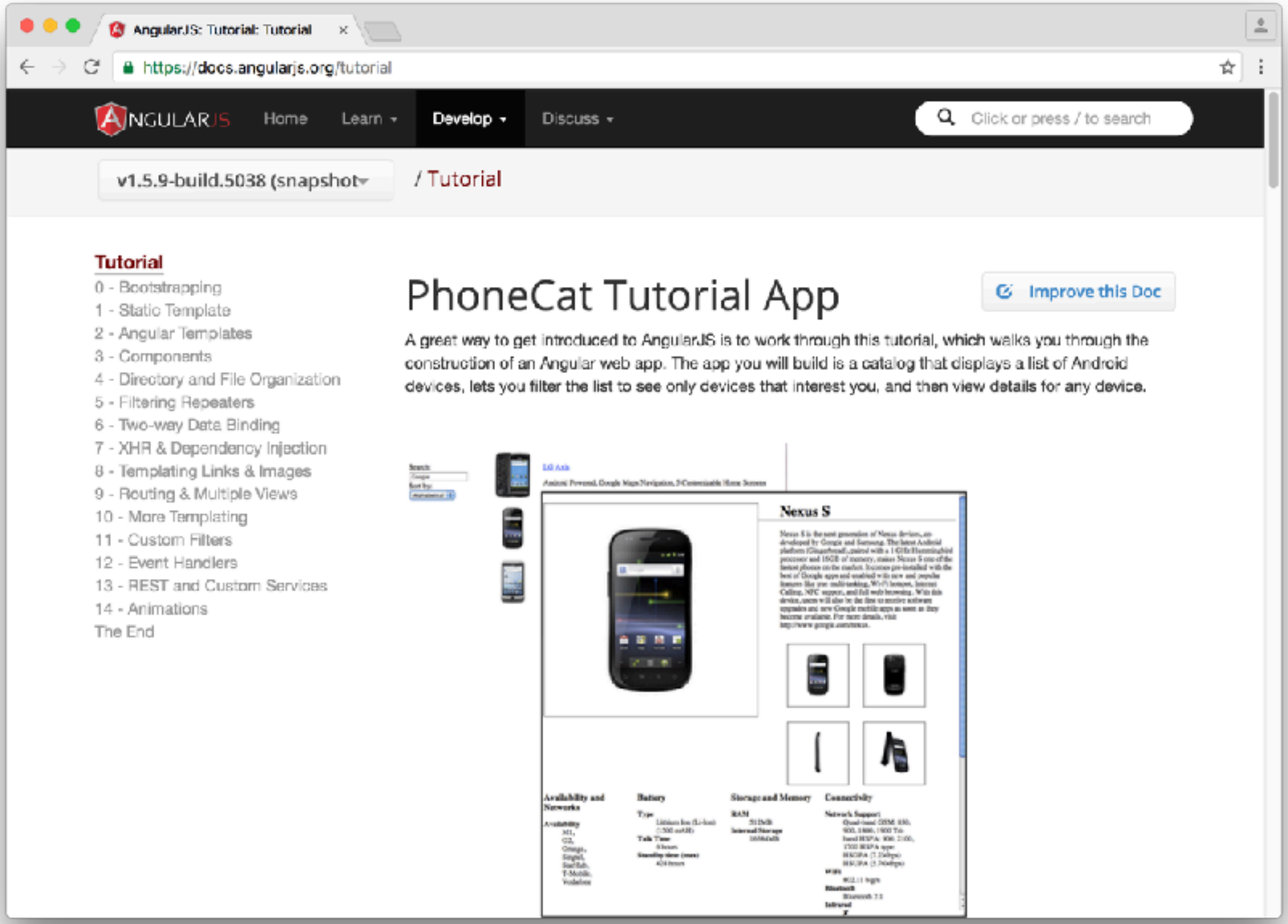


Easier to develop in parallel (V only talks to VM)
View is completely “dumb” and just needs data bindings

AngularJS

- Supports MVC/MVVM
- Provides structure to organize your code
- Two-way data binding
- Uses plain old objects for your data - no fancy structures needed
- HTML templating (like react)
- Designed for SPAs

Angular Documentation: Great



Directives & Data Binding

- Core feature of Angular
- Unlike React (add HTML to code), Angular lets us *direct* the html to have some code in it too
- Lets us add code into HTML
- Angular example:

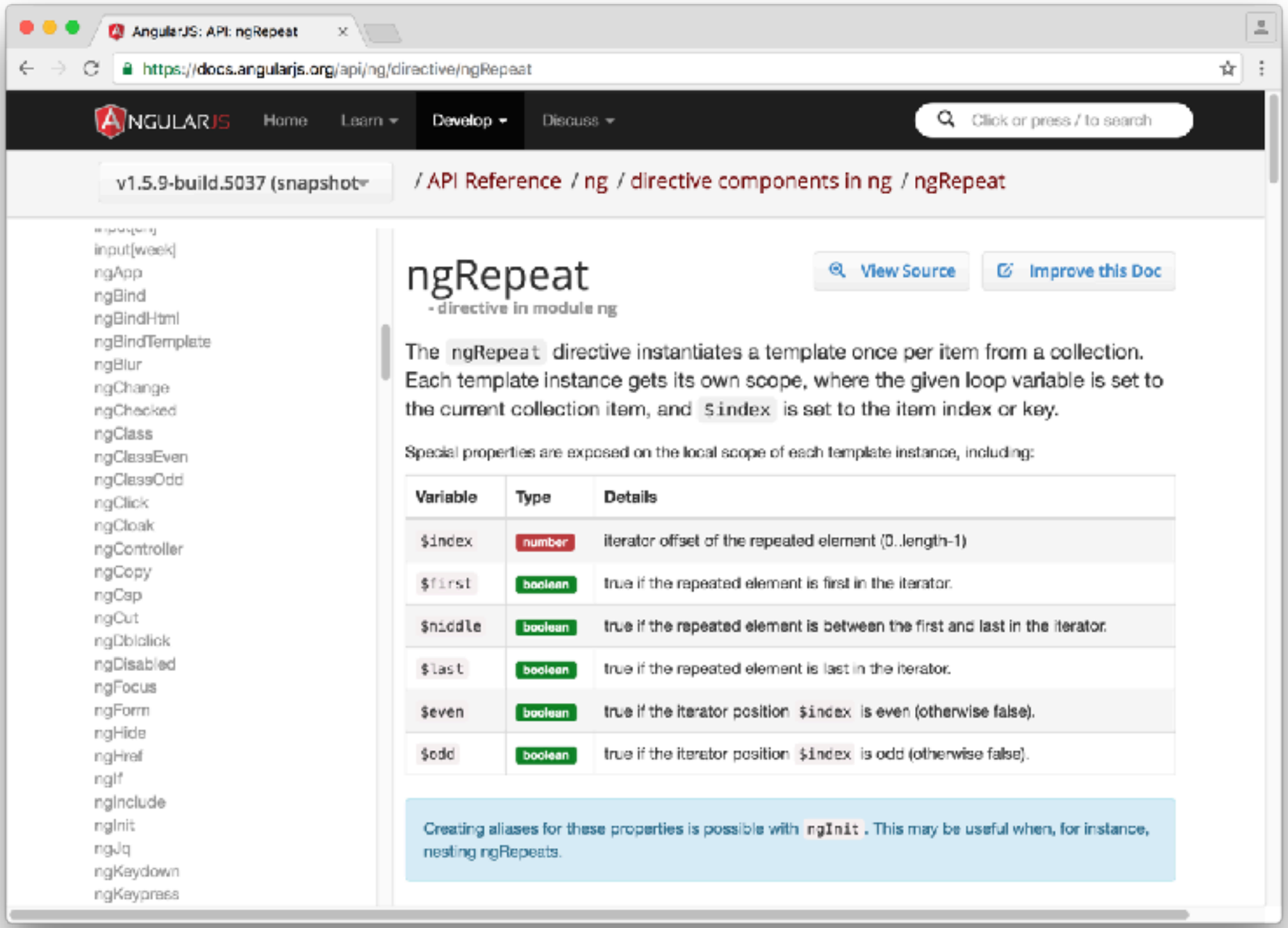
```
<!DOCTYPE html>
<html lang="en" ng-app>
<head>
    <meta charset="UTF-8">
    <title>My Angular Demo</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/
angular.js"></script>

</head>
<body>
What's your name? <input type="text" data-ng-model="name" /> {{name}}
</body>
</html>
```

Directives

Data Binding

Simple Data Binding Example

Other Directives



Other Directives

- ng-init
  - Initialize variables within the scope of a DOM element
- ng-repeat
  - Replicate a DOM element over an array

```
<div class="container"
   data-ng-init="names=['Dave','Napur','Heedy','Shriva']">

    <h3>Looping with the ng-repeat Directive</h3>
    <ul>
        <li data-ng-repeat="name in names">{{ name }}</li>
    </ul>
</div>
```

Filters

- Allow you to modify the text going into data bindings
- Only want to make simple modifications here
- Syntax:

```
{{todo.text | uppercase}}
```

  - (Converts the todo to uppercase)

```
<div ng-repeat="todo in todos | orderBy:'-priority' ">
```

  - (Shows all todos ordered by key)
- Other uses:
  - Select only some values in a list
  - Order a list

Partials:

- A "Partial" HTML document
- Can be included into another with **<ngInclude>**
- Example:

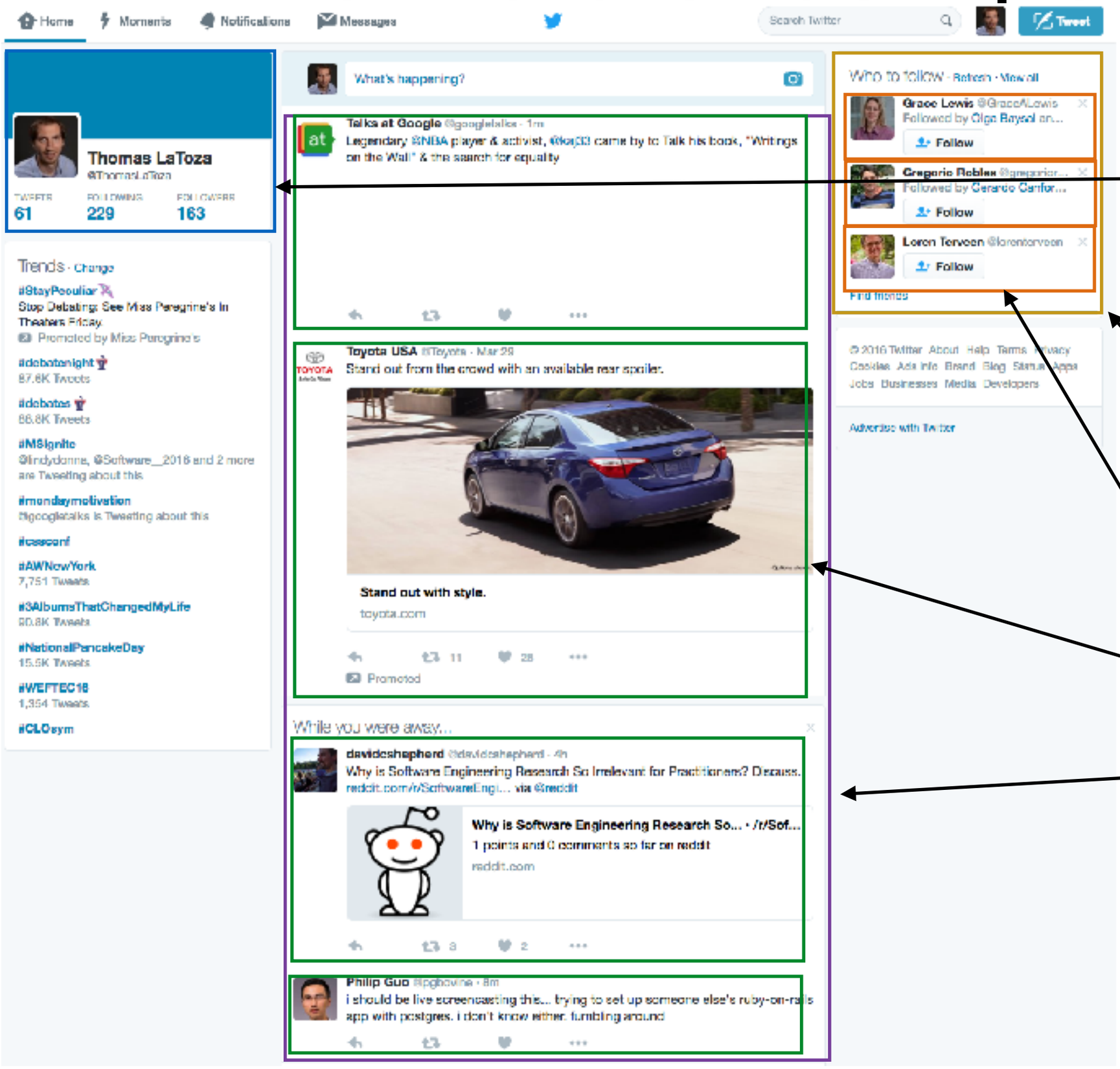
```
<div>
    What's your name? <input type="text"
data-ng-model="name"/> Hello, {{name}}!
</div>
```

partials/hello.html

```
<ng-include src="'partials/hello.html'"></ng-include>
```

index.html

Partials & Components



User profile partial

Who to follow partial

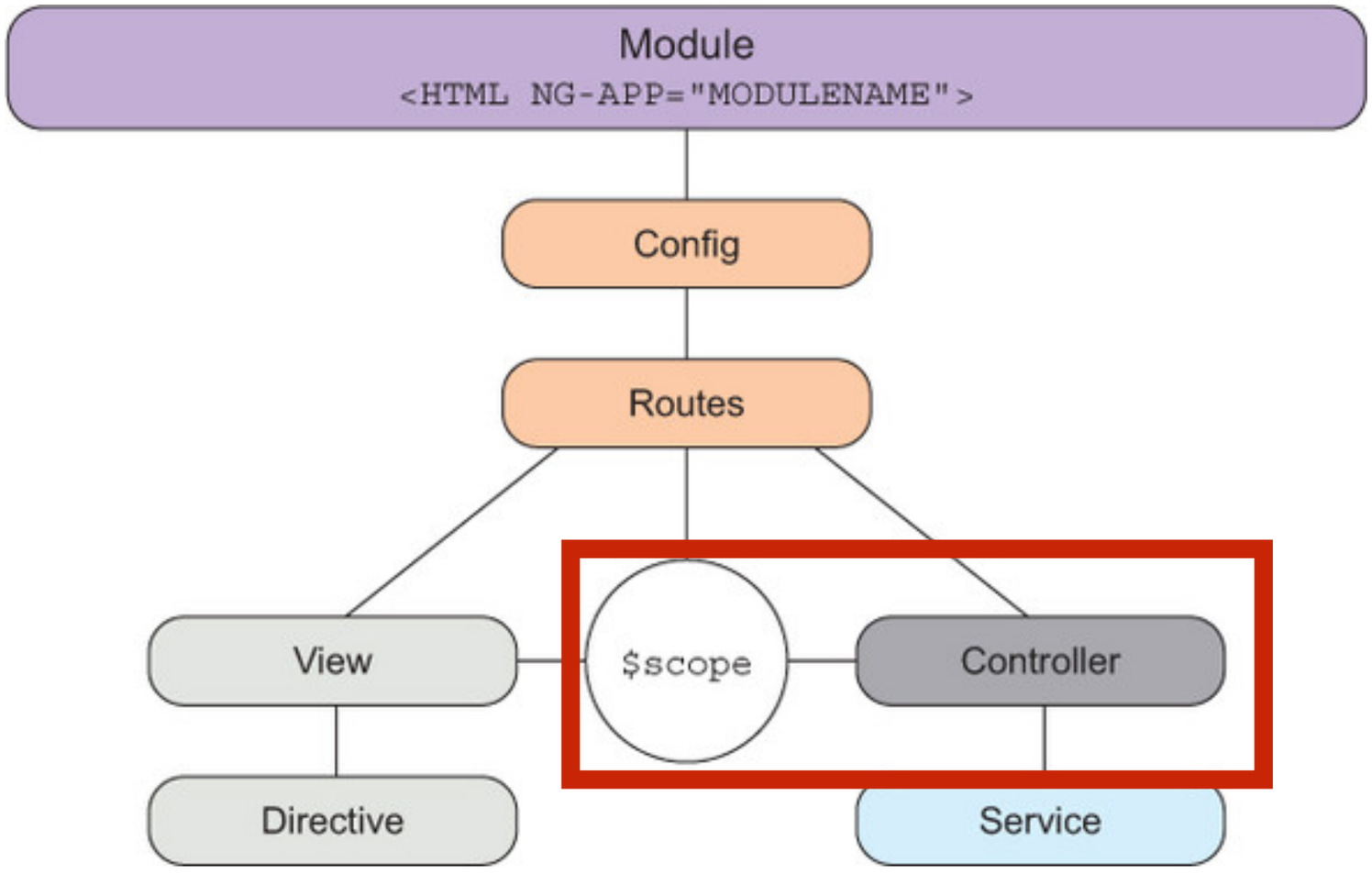
Follow partial

Feed partial

Feed item partial

Partial Demo

Views, Controllers, Scopes



- Angular has a lot more than just views and directives
- Let's focus on controllers and scope

Angular Controllers

- Each controller is a function that gets passed `$scope`
- `$scope` is the bridge between the controller and view
- `$scope` is initially empty when the controller is called, and then it sets some properties on it
- When a view uses a controller, it inherits its `$scope`

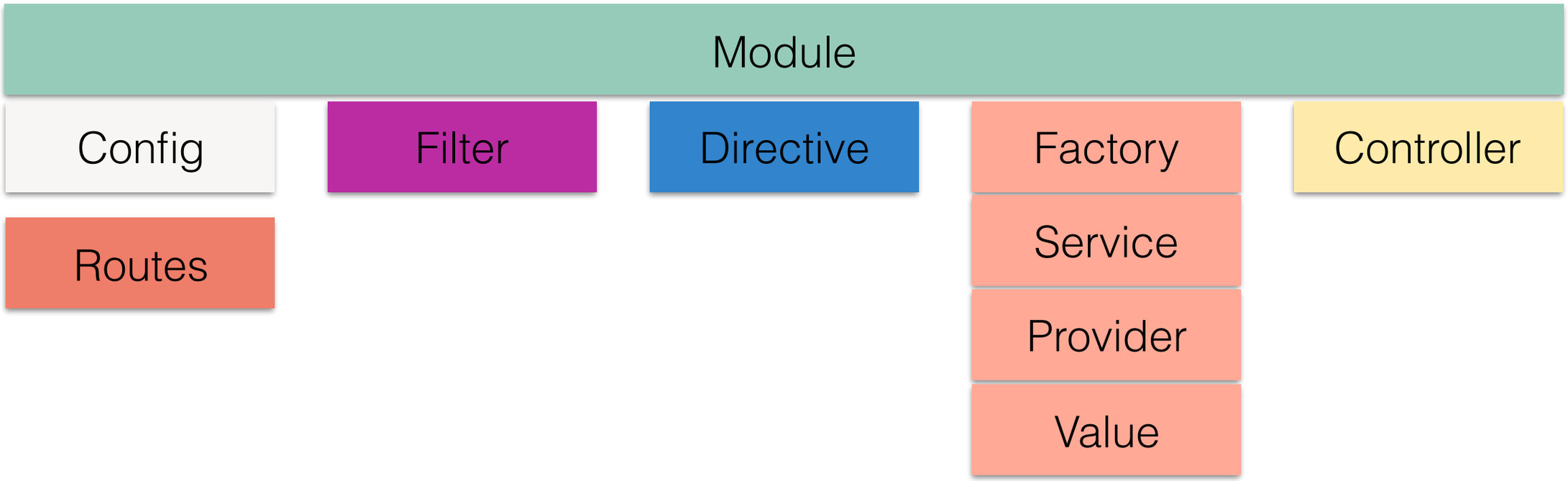
```
function TodoController($scope)
{
    $scope.todos = [
        { text: "Write more demos", priority: 5},
        { text : "Add some gifs", priority: 10}
    ];
}
```

Views & Controllers

- Select a controller for a DOM element, and it will provide variables for everything contained in it
- Can have multiple controllers on one page

```
<div class="todoList" data-ng-controller='TodoController'>
    <div ng-repeat='todo in todos | orderBy:'-priority' '>
    {{todo.text | uppercase}}
    </div>
</div>
<script>
    function TodoController($scope)
    {
        $scope.todos = [
            { text: "Write more demos", priority: 5},
            { text : "Add some gifs", priority: 10}
        ];
    }
</script>
```

Modules as Containers



- Modules contain everything that we need for a single component
- Organize views, controllers, etc.
- How do we make and use them?

Creating a Module

- Create a module and add a controller:

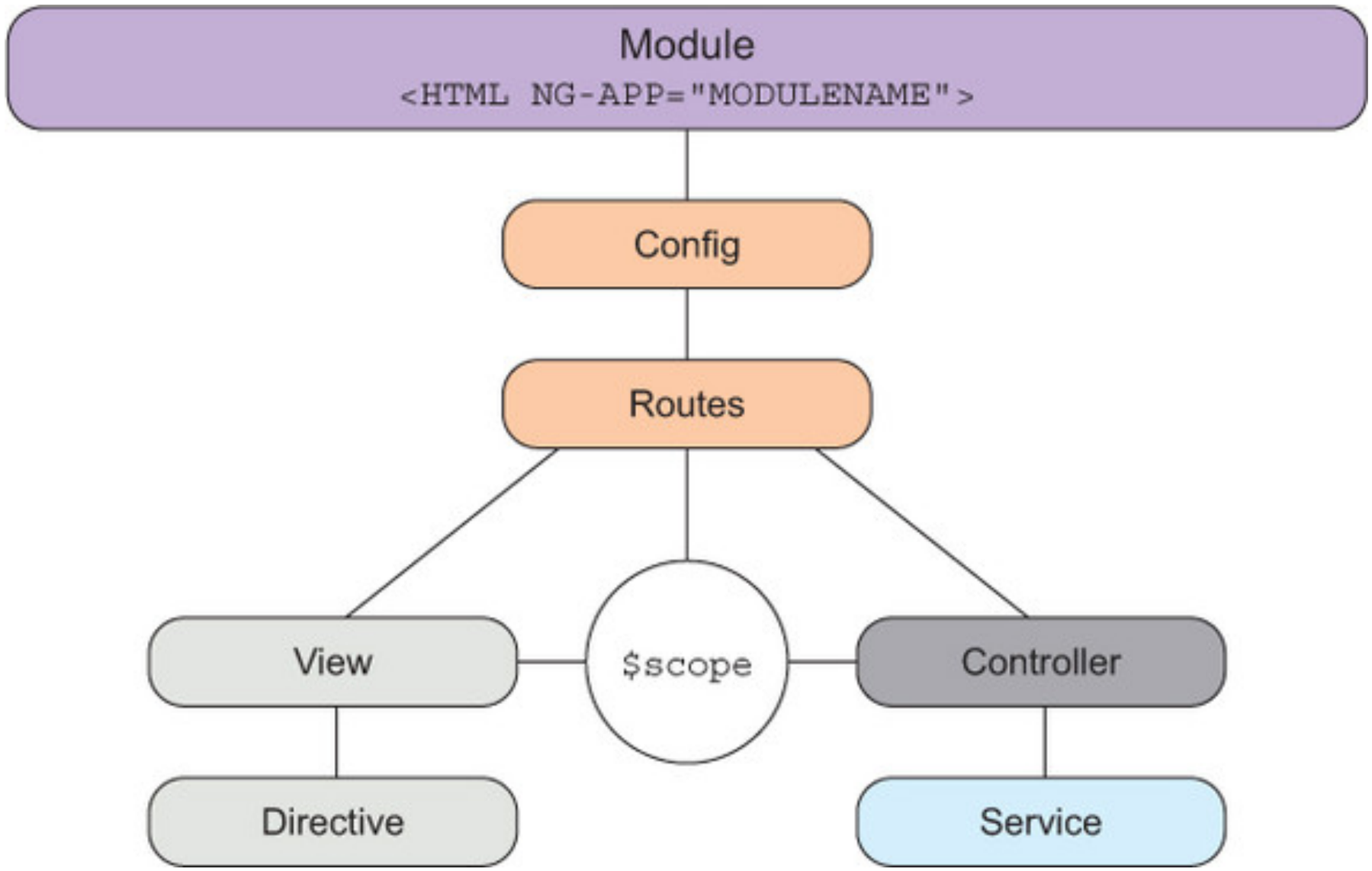
```
var myApp = angular.module('demoApp', []);
myApp.controller("TodoController", TodoController);
```

- The empty array can instead specify dependencies
  - Example dependency (a great one!): `firebase`
  - Controllers should not stand on their own - must be part of module
  - Module name must be the name provided in ng-app

```
<html lang="en" data-ng-app=“demoApp">
```

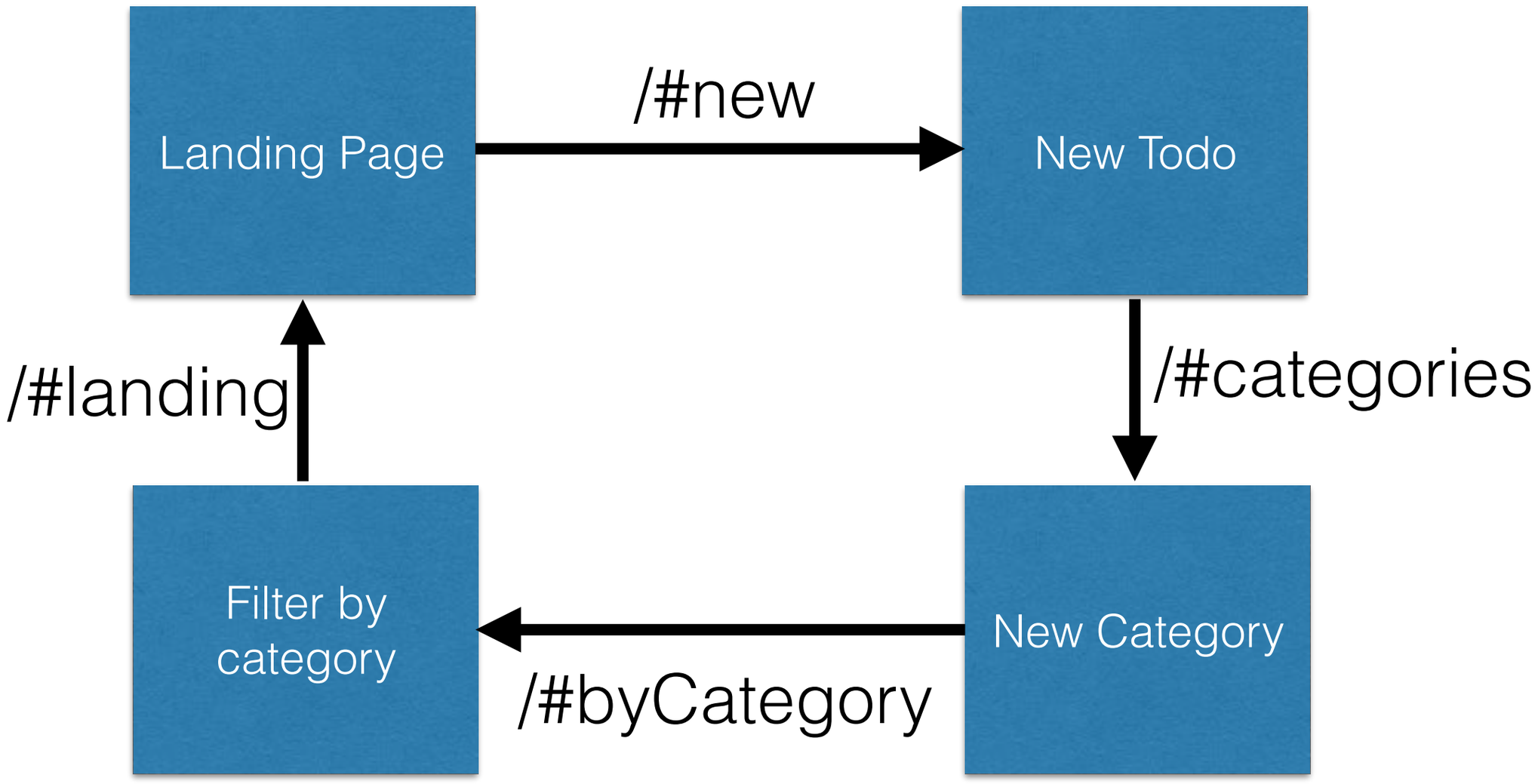
Demo: Modules, Controllers, Firebase

Modules, Routes, Services



Routes

Routes are paths from view/controllers to others



Routes

- AngularJS makes routes read like magic!

```
myApp.config(function($routeProvider){
    $routeProvider.when("/",{
        controller: "TodoController",
        templateUrl: "partials/editableTodos.html"
    }).when("/categories",{
        controller: "CategoryController",
        templateUrl: "partials/categories.html"
    }).otherwise({redirectTo: "/"});
});
```

Dependency injection magic!

- Reads like a sentence (chaining!)

Partials

- Easy way to have "partial" HTML documents and combine them, magically-dynamically into one!
- Will be included by the **route**, into the container labeled with the directive

```
<div data-ng-view></div>
...
```

```
myApp.config(function($routeProvider){
    $routeProvider.when("/",{
        controller: "TodoController",
        templateUrl: "partials/editableTodos.html"
    }).when("/categories",{
        controller: "CategoryController",
        templateUrl: "partials/categories.html"
    }).otherwise({redirectTo: "/"});
});
```

Demo: Routes + Partials

Exit-Ticket Activity

Go to socrative.com and select “Student Login”

Class: SWE432001 (Prof LaToza) or SWE432002 (Prof Bell)

ID is your @gmu.edu email

1: How well did you understand today's material
2: What did you learn in today's class?
For question 3: How do you think you will use React in your HW this week?