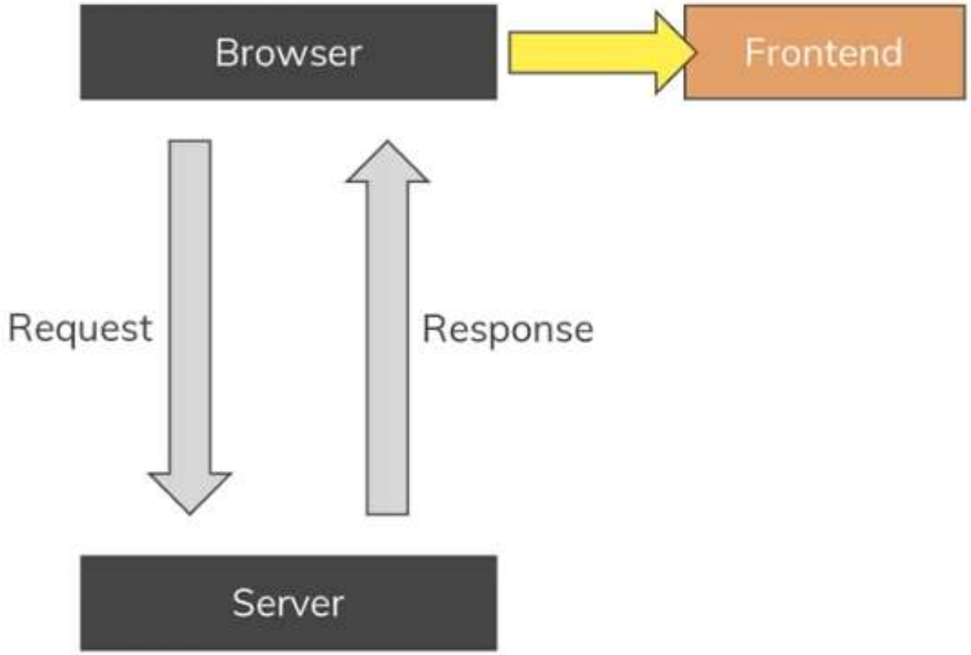
FULL STACK – WEB DEVELOPMENT

FRONTEND DEVELOPMENT

What?



Browser

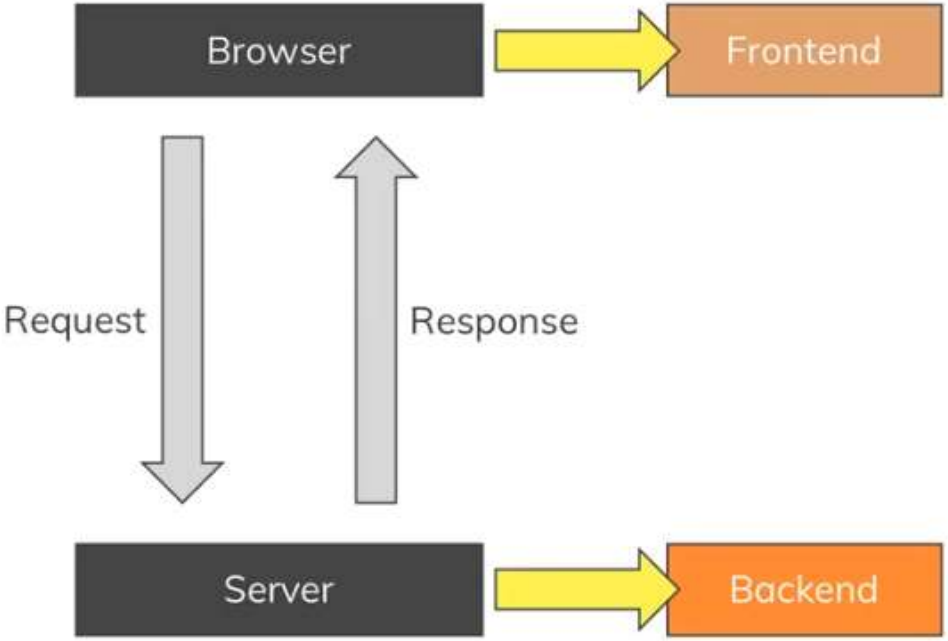
Frontend

Request

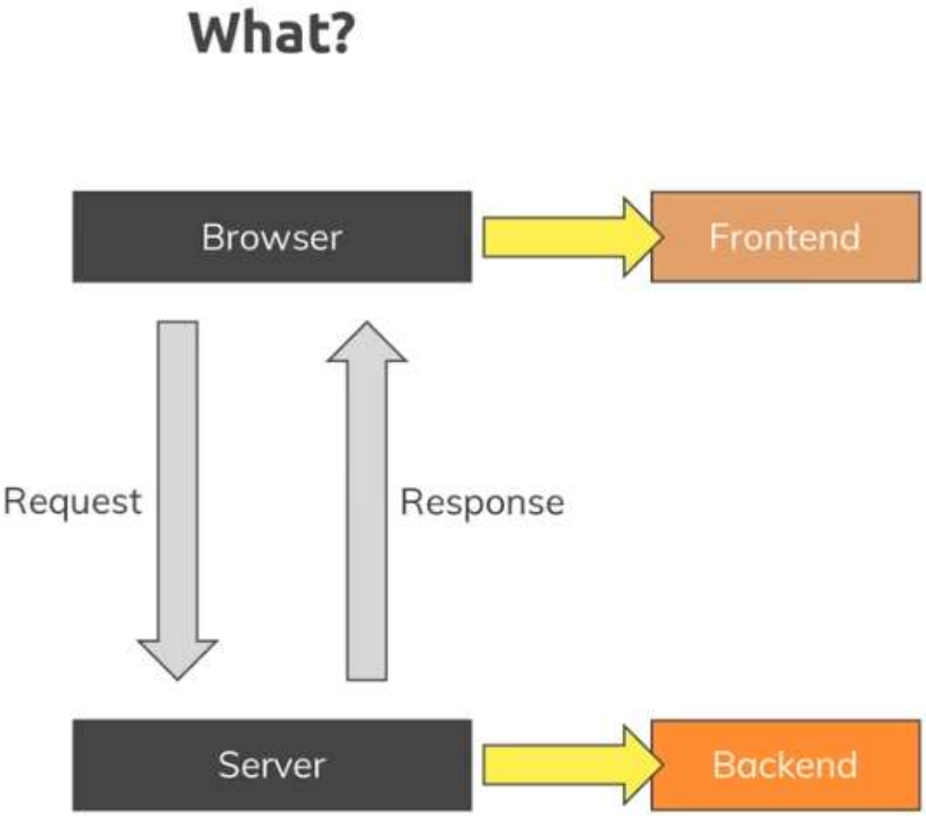
Response

Server

BACKEND DEVELOPMENT

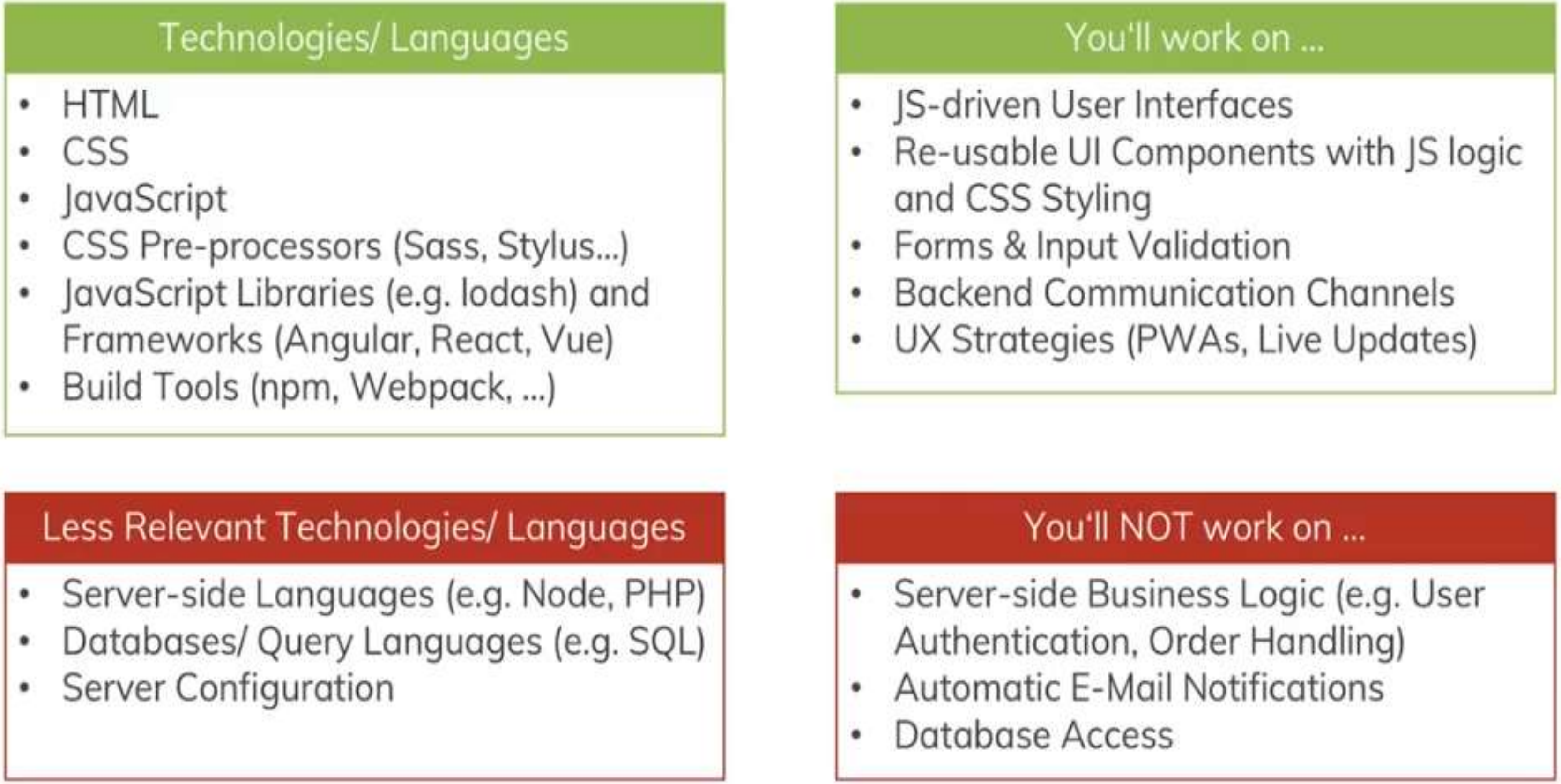


FULL STACK DEVELOPMENT



FULLSTACK

FRONTEND



Technologies/ Languages

- HTML
- CSS
- JavaScript
- CSS Pre-processors (Sass, Stylus...)
- JavaScript Libraries (e.g. lodash) and Frameworks (Angular, React, Vue)
- Build Tools (npm, Webpack, ...)

You'll work on ...

- JS-driven User Interfaces
- Re-usable UI Components with JS logic and CSS Styling
- Forms & Input Validation
- Backend Communication Channels
- UX Strategies (PWAs, Live Updates)

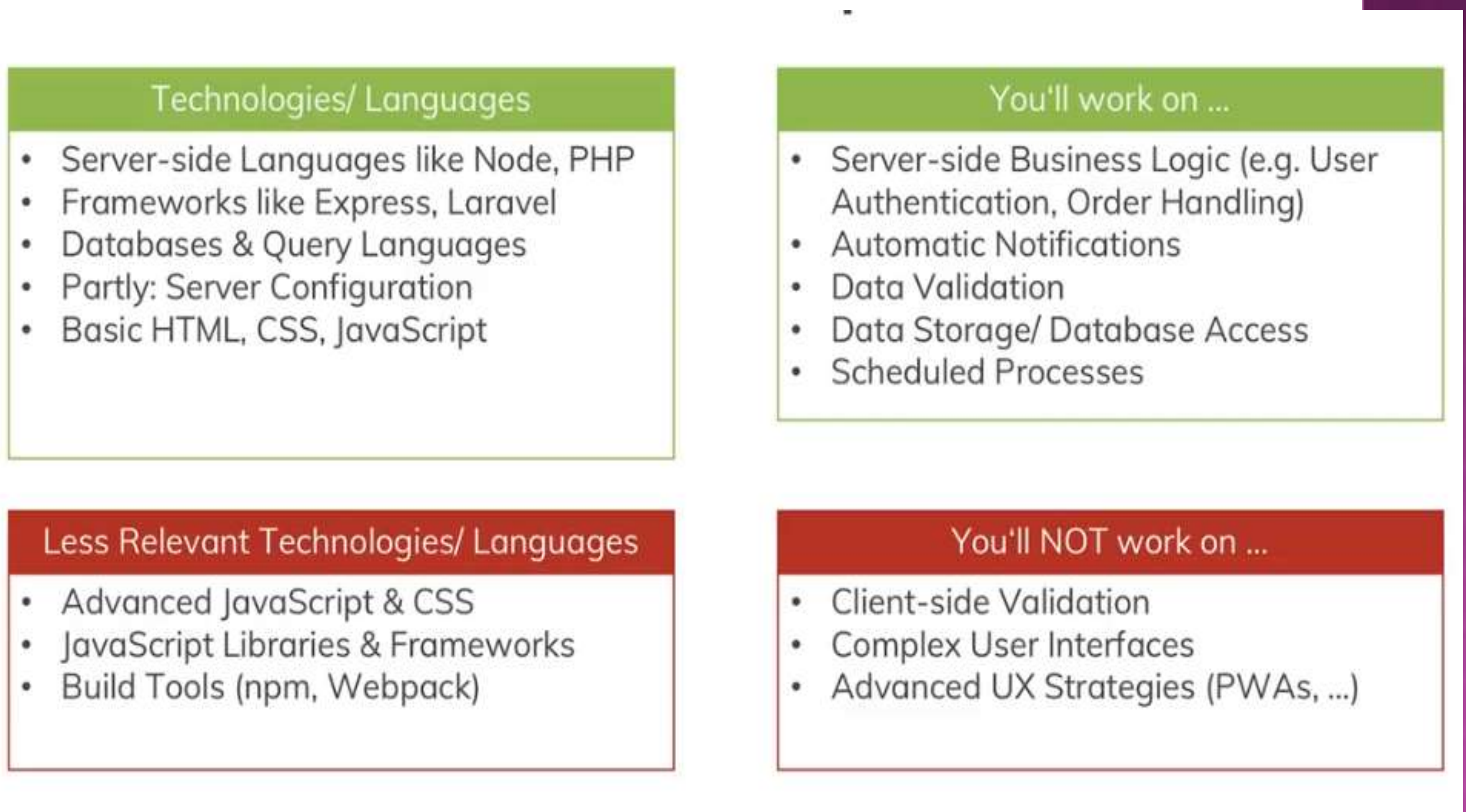
Less Relevant Technologies/ Languages

- Server-side Languages (e.g. Node, PHP)
- Databases/ Query Languages (e.g. SQL)
- Server Configuration

You'll NOT work on ...

- Server-side Business Logic (e.g. User Authentication, Order Handling)
- Automatic E-Mail Notifications
- Database Access

BACKEND



Technologies/ Languages

- Server-side Languages like Node, PHP
- Frameworks like Express, Laravel
- Databases & Query Languages
- Partly: Server Configuration
- Basic HTML, CSS, JavaScript

You'll work on ...

- Server-side Business Logic (e.g. User Authentication, Order Handling)
- Automatic Notifications
- Data Validation
- Data Storage/ Database Access
- Scheduled Processes

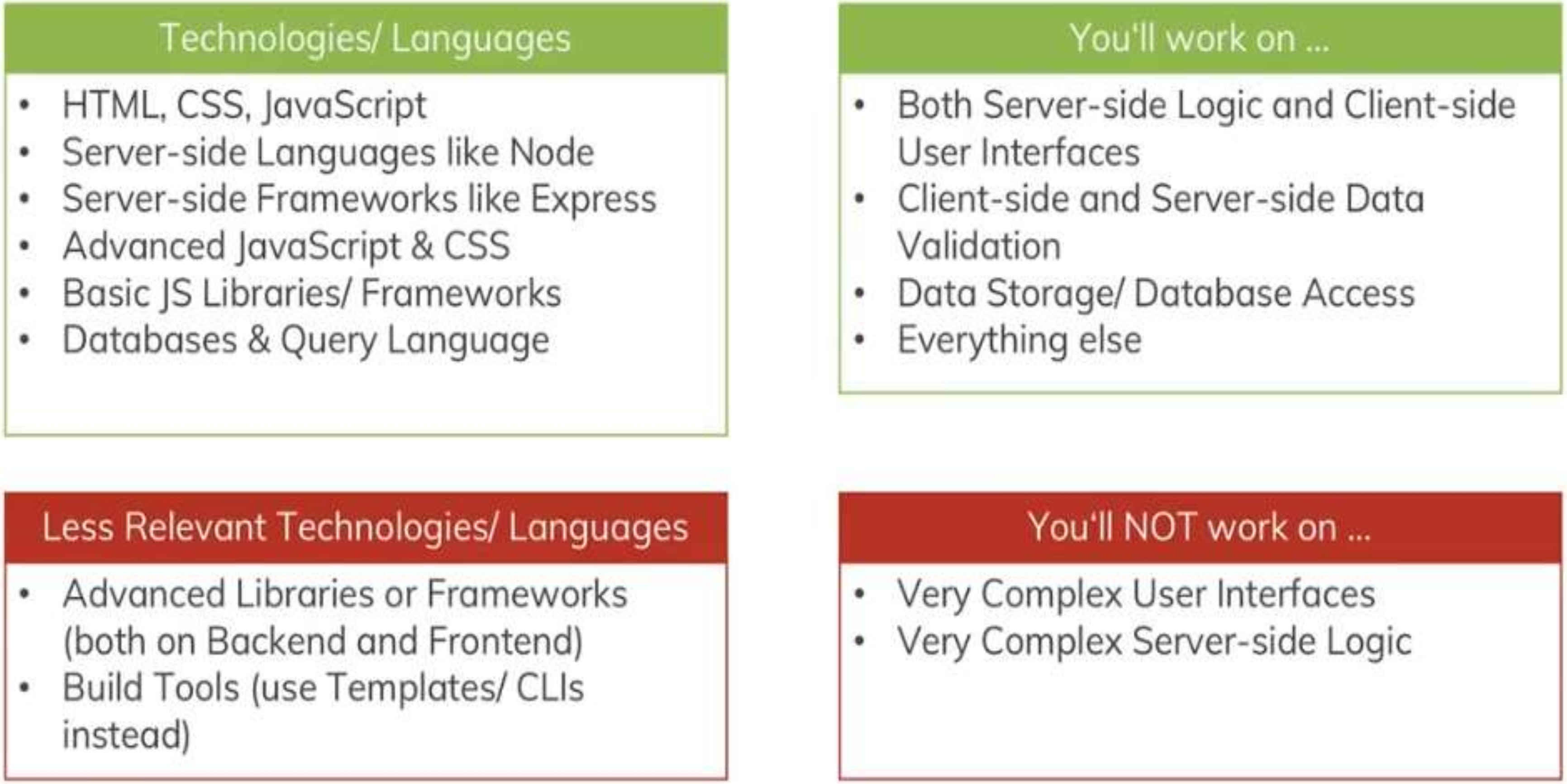
Less Relevant Technologies/ Languages

- Advanced JavaScript & CSS
- JavaScript Libraries & Frameworks
- Build Tools (npm, Webpack)

You'll NOT work on ...

- Client-side Validation
- Complex User Interfaces
- Advanced UX Strategies (PWAs, ...)

FULL STACK



Technologies/ Languages

- HTML, CSS, JavaScript
- Server-side Languages like Node
- Server-side Frameworks like Express
- Advanced JavaScript & CSS
- Basic JS Libraries/ Frameworks
- Databases & Query Language

You'll work on ...

- Both Server-side Logic and Client-side User Interfaces
- Client-side and Server-side Data Validation
- Data Storage/ Database Access
- Everything else

Less Relevant Technologies/ Languages

- Advanced Libraries or Frameworks (both on Backend and Frontend)
- Build Tools (use Templates/ CLIs instead)

You'll NOT work on ...

- Very Complex User Interfaces
- Very Complex Server-side Logic

POPULAR STACKS

- **LAMP stack**: JavaScript - Linux - Apache - MySQL - PHP
- **LEMP stack:** JavaScript - Linux - Nginx - MySQL - PHP
- **MEAN stack:** JavaScript - MongoDB - Express - AngularJS - Node.js
- **Django stack:** JavaScript - Python - Django - MySQL
- **Ruby on Rails**: JavaScript - Ruby - SQLite - PHP

ADVANTAGES

- You can make a prototype very rapidly
- You can provide help to all the team members
- You can reduce the cost of the project
- You can reduce the time used for team communication
- You can switch between front and back end development based on requirements
- You can better understand all aspects of new and upcoming technologies

DISADVANTAGES

- The solution chosen can be wrong for the project
- The solution chosen can be dependent on developer skills
- The solution can generate a key person risk
- Being a full stack developer is increasingly complex

FULL STACK JAVASCRIPT

- JavaScript has been around for over 20 years.
- It is the dominant programming language in web development.
- In the beginning JavaScript was a language for the web client (browser).
- Then came the ability to use JavaScript on the web server (with Node.js).

FULL STACK JAVASCRIPT

- Today the hottest buzzword is "Full Stack JavaScript".
- The idea of "Full Stack JavaScript" is that all software in a web application, both client side and server side, should be written using JavaScript only.

FULL STACK JS

- A full stack JavaScript developer is a person who can develop both **client** and **server** software.
- In addition to mastering HTML and CSS, he/she also knows how to:
- Program a **browser** (like using JavaScript, jQuery, Angular, or Vue)
- Program a **server** (like using Node.js)
- Program a **database** (like using MongoDB)

BACK END LANGUAGES

- ~~PHP~~
- ~~ASP~~
- ~~C++~~
- ~~C#~~
- ~~Java~~
- ~~Python~~

- Node.js
- Ruby
- REST
- GO
- SQL
- MongoDB

FULL STACK JAVASCRIPT BENEFITS

- Code reuse. Shared libraries, templates, and models.
- Best practice accumulated by 20 years of JavaScript.
- JavaScript is an evolving standard with a bright future.
- Easy to learn.
- No compilation. Faster development.
- Great distribution: npm.

EXAMPLE

- MEAN STACK
- M- Mongo DB
- E- Express js
- A- Angular js
- N- Node js.

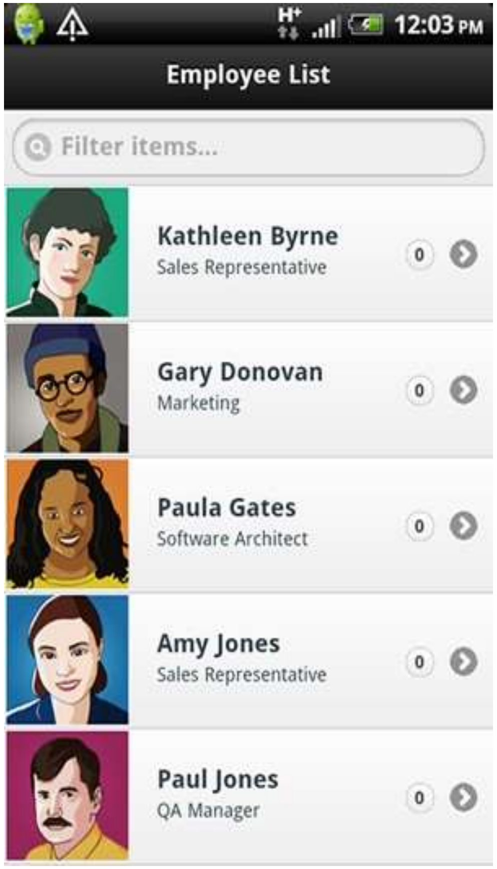
ANGULARJS
by Google

RISE OF THE RESPONSIVE SINGLE PAGE APP

Image: http://johnpolacek.github.io/scrolldeck.js/decks/responsive/

RESPONSIVE

- Unified across experiences
- Can be embedded as mobile app
- Better deployment and & maintanence
- Mobile users need to get access to everything



12:03 PM

Employee List

Filter items...

Kathleen Byrne
Sales Representative
0

Gary Donovan
Marketing
0

Paula Gates
Software Architect
0

Amy Jones
Sales Representative
0

Paul Jones
QA Manager
0

Image: http://coenraets.org/blog/wp--
-content/uploads/2011/10/directory11.png

SINGLE--PAGE APPLICATIONS (SPA)

- Web app that fits on **a single web page**
  - Fluid UX, like desktop app
  - Examples like Gmail, Google maps
- Html page contains **mini--views** *(*HTML Fragments*)* that can be loaded in the background
- **No reloading** of the page,
- Requires handling of **browser history, navigation and bookmarks**

JAVASCRIPT

- SPAs are implemented using **JavaScript** and **HTML**

CHALLENGES IN SPA

- **DOM Manipulation**
  - How to manipulate the view efficiently?
- **History**
  - What happens when pressing back button?
- **Routing**
  - Readable URLs?
- **Data Binding**
  - How bind data from model to view?
- **View Loading**
  - How to load the view?
- Lot of coding! You could **use a framework instead …**

SINGLE-PAGE APPLICATION

- Single page apps typically have
  - “application like” interaction
  - dynamic data loading from the server-side API
  - fluid transitions between page states
  - more JavaScript than actual HTML
- They typically do not have
  - support for crawlers (not for sites relying on search traffic)
  - support for legacy browsers (IE7 or older, dumbphone browsers)

SPAS ARE GOOD FOR …

- “App-like user experience”
- Binding to your own (or 3rd party) RESTful API
- Replacement for Flash or Java in your web pages
- Hybrid (native) HTML5 applications
- Mobile version of your web site

The SPA sweet spot is likely not on web sites,
but on content-rich cross-platform mobile apps

PJAX

- Pjax is a technique that allows you to progressively enhance normal links on a page so that clicks result in the linked content being loaded via Ajax and the URL being updated using HTML5 pushState, avoiding a full page load.
- In browsers that don't support pushState or that have JavaScript disabled, link clicks will result in a normal full page load. The Pjax Utility makes it easy to add this functionality to existing pages.

http://yuilibrary.com/yui/docs/pjax/

SPAS AND OTHER WEB APP ARCHITECTURES

	Server-side	Server-side + AJAX	PJAX	SPA
What	Server round-trip on every app state change	Render initial page on server, state changes on the client	Render initial page on server, state changes on server, inject into DOM on client-side	Serve static page skeleton from server; render every change on client-side
How	UI code on server; links & form posting	UI code on both ends; AJAX calls, ugly server API	UI code on server, client to inject HTTP, server API if you like	UI code on client, server API
Ease of development	🟢	🔴	🟢	🟢
UX & responsiveness	🔴	🟡	🟡	🟢
Robots & old browsers	🟢	🟡	🟢	🔴
Who's using it?	Amazon, Wikipedia; banks, media sites etc.	Facebook?; widgets, search	Twitter, Basecamp, GitHub	Google+, Gmail, FT; mobile sites, startups

ANGULAR_JS

ANGULAR JS

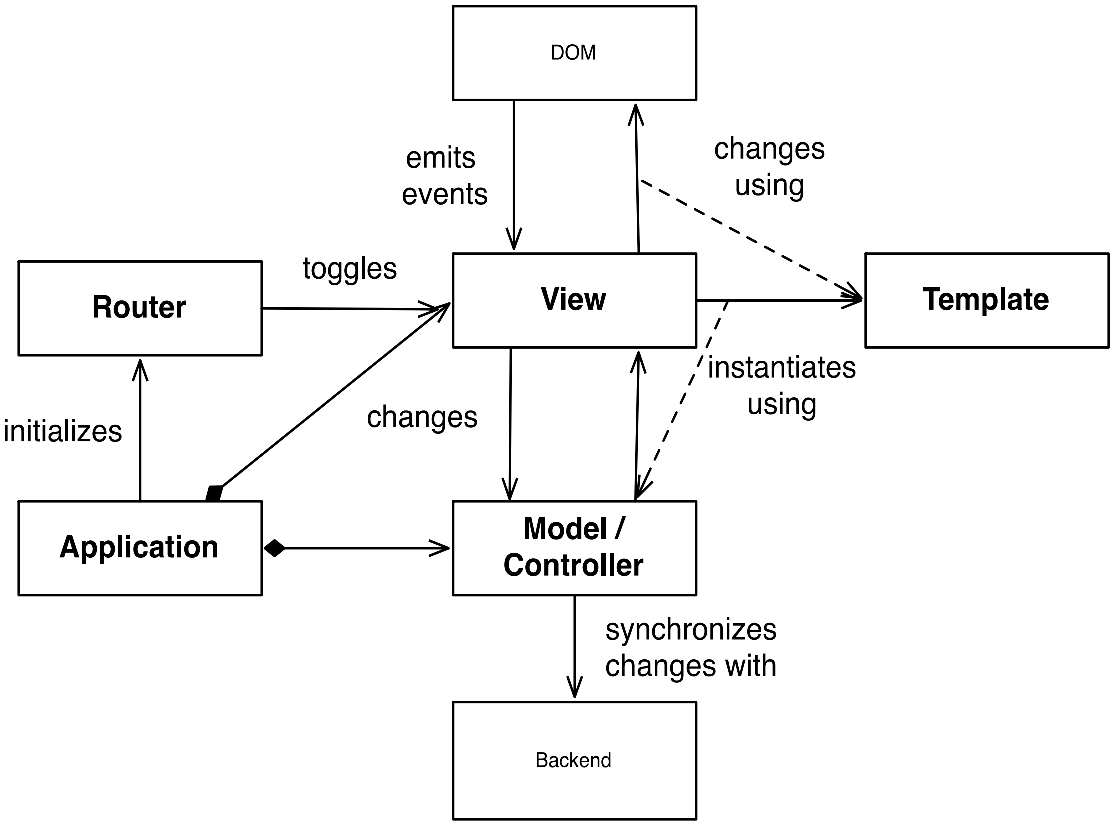
- **Single Page App Framework** for JavaScript
- Implements client--side **MVC** pattern
  - Separation of presentation from business logic and presentation state
- **No direct DOM** manipulation, less code
- Support for all major browsers
- Supported by Google
- Large and fast growing community

ANGULARJS – MAIN CONCEPTS

- Templates
- Directives
- Expressions
- Data binding
- Scope

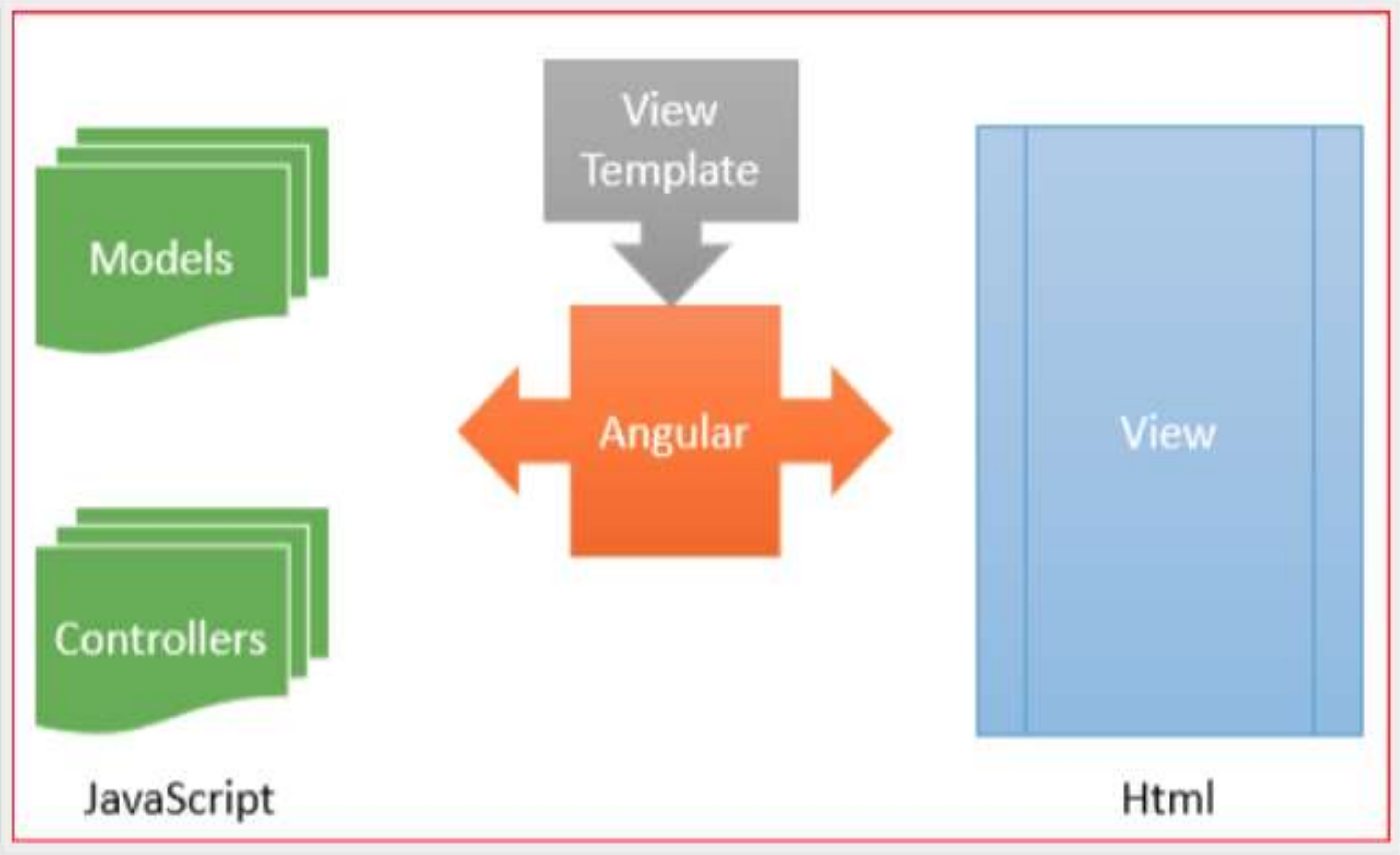
- Controllers
- Modules
- Filters
- Services
- Routing

ANATOMY OF A BACKBONE SPA



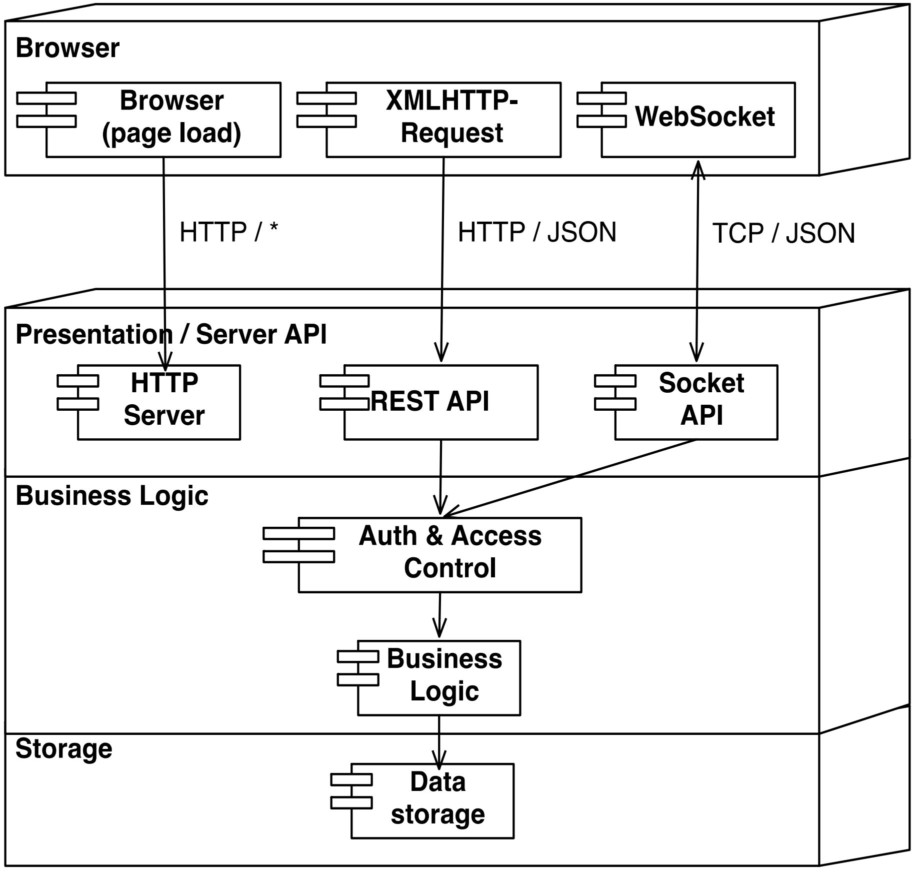
- Application as a ‘singleton’ reference holder
- Router handles the navigation and toggles between views
- Models synchronize with Server API
- Bulk of the code in views
- All HTML in templates

FROM: LAURI SVAN



From Gary Arora

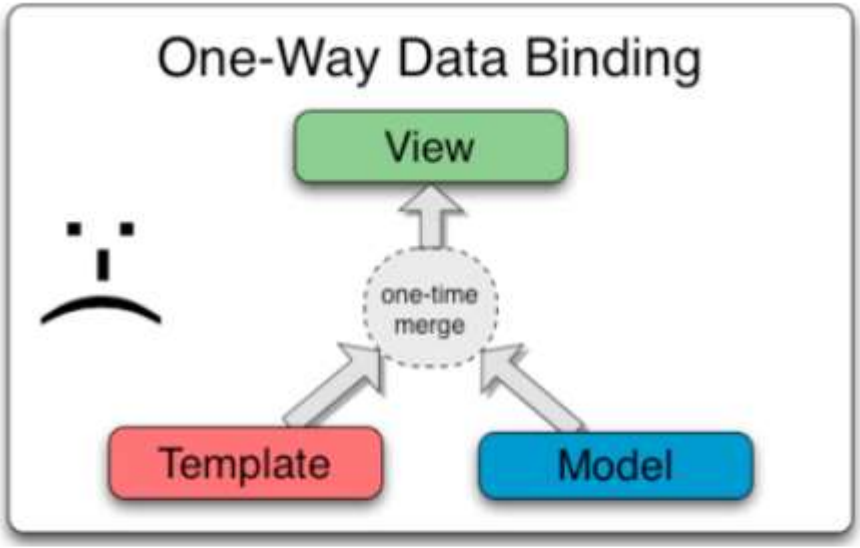
SPA CLIENT-SERVER COMMUNICATION

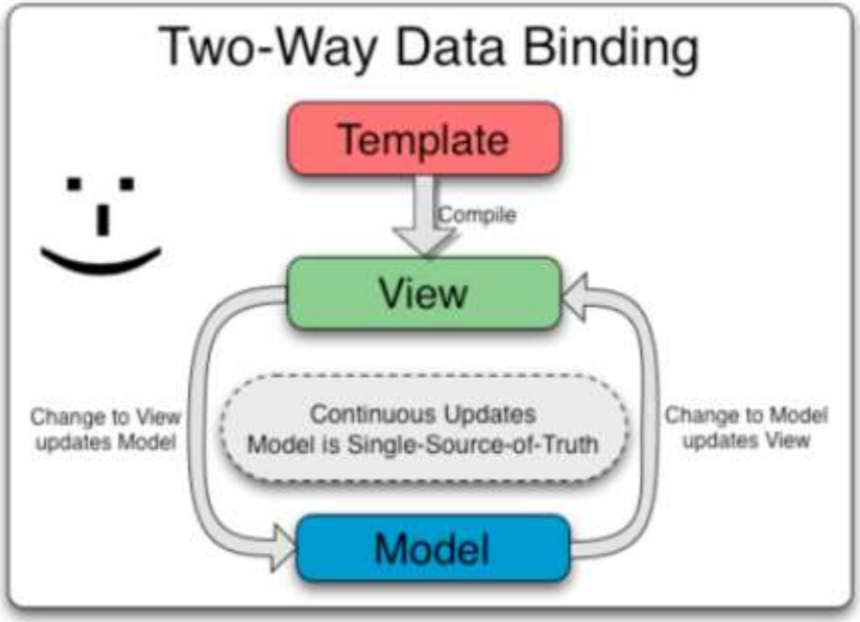


- HTML and all the assets are loaded in first request
- Additional data is fetched over XMLHTTPRequest
- If you want to go real-time, WebSockets (socket.io) can help you
- When it gets slow, cluster the backend behind a caching reverse proxy like Varnish

FROM: LAURI SVAN

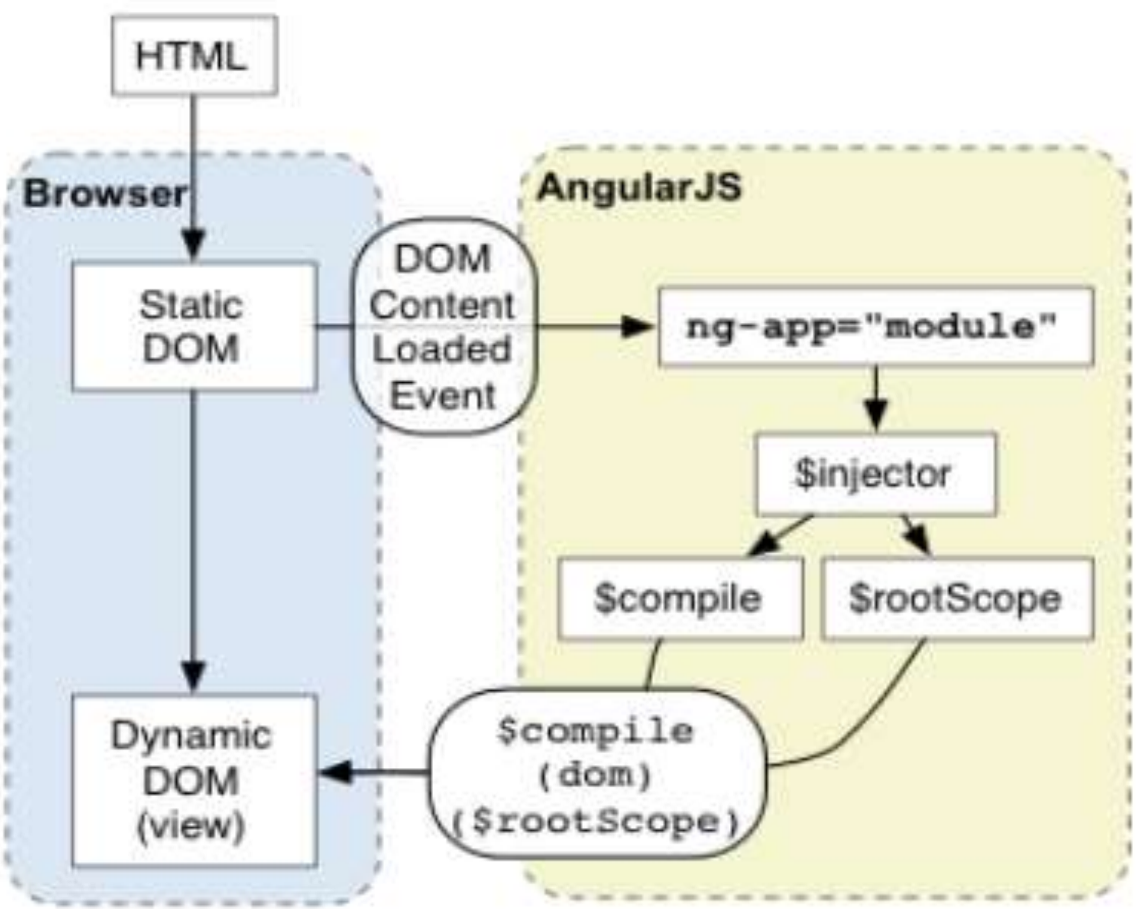
HOW IT WORKS?




Two-Way Data Binding
Template
Compile
View
Change to View updates Model
Continuous Updates
Model is Single-Source-of-Truth
Change to Model updates View
Model

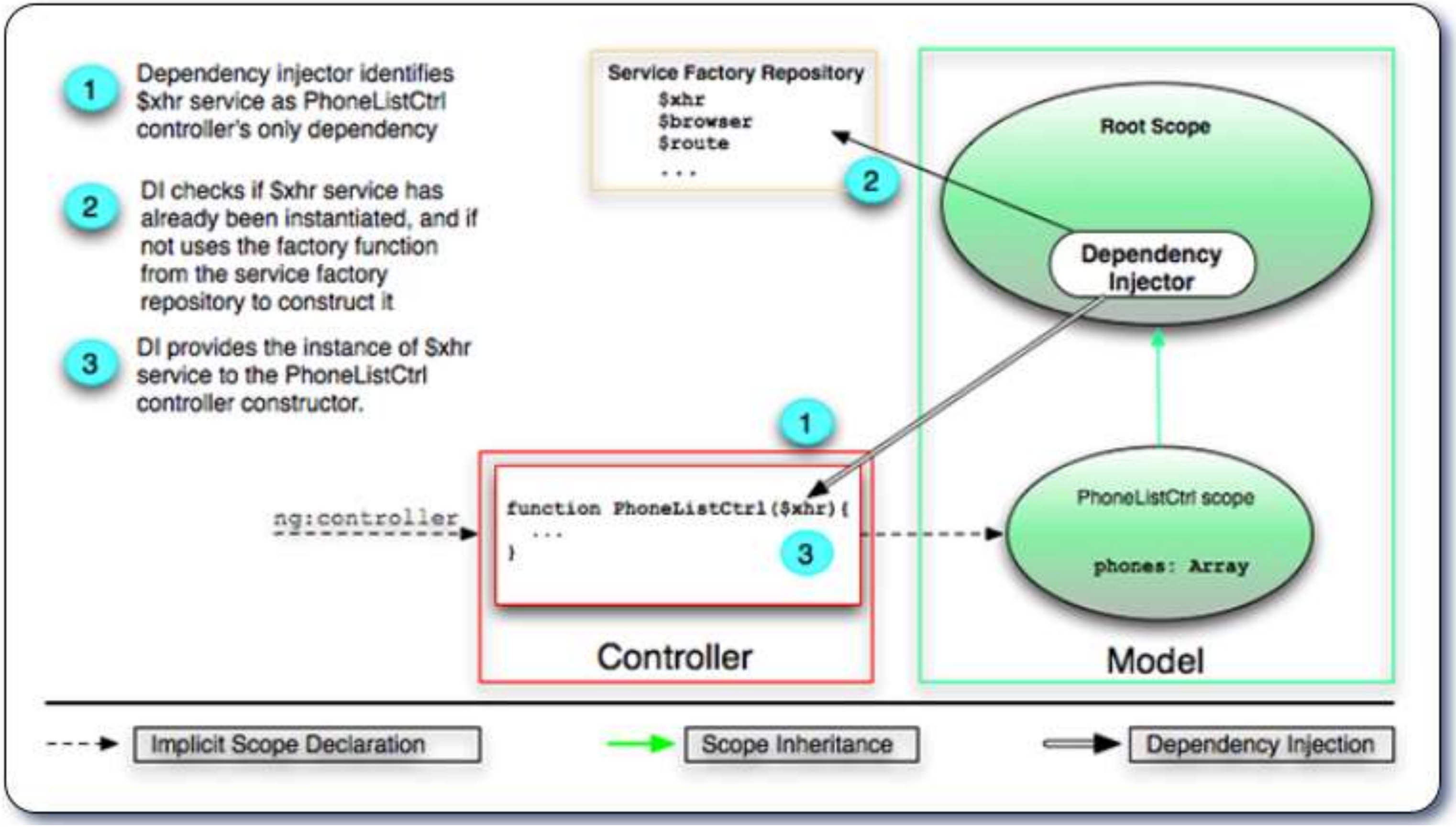
From Rouson

HOW IT WORKS?



From Rouson

HOW IT WORKS?



1. Dependency injector identifies $xhr service as PhoneListCtrl controller's only dependency

2. DI checks if $xhr service has already been instantiated, and if not uses the factory function from the service factory repository to construct it

3. DI provides the instance of $xhr service to the PhoneListCtrl controller constructor.

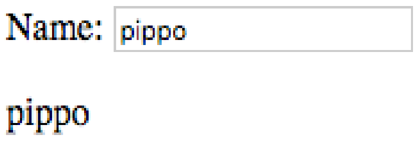
```
function PhoneListCtrl($xhr){
  ...
}
```

From Rouson

GETTING STARTED WITH ANGULAR_JS

BASIC CONCEPTS

- **1) Templates**
  - HTML with additional markup, directives, expressions, filters ...
- **2) Directives**
  - Extend HTML using `ng-app`, `ng-bind`, `ng-model`
- **3) Filters**
  - Filter the output: `filter`, `orderBy`, `uppercase`
- **4) Data Binding**
  - Bind model to view using expressions `{{ }}`



Name: pippo

pippo

FIRST EXAMPLE - TEMPLATE

Template

```
<!DOCTYPE html>
<html>
 <head>
 <title>Title</title>
 <meta charset="UTF-8" />
 <style media="screen"></style>
 <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/an
gular.min.js"></script>
 </head>
 <body>
 <div ng-app>
<!-- store the value of input field into a variable name -->
<p>Name: <input type="text" ng-model="name"></p>
<!-- display the variable name inside (innerHTML) of p -->
<p ng-bind="name"></p>
</div>
</body>
</html>
```

2) DIRECTIVES

- **Directives** apply special behavior to attributes or elements in HTML
  - Attach behaviour, transform the DOM
- Some directives
  - **ng-app**
    - Initializes the app
  - **ng-model**
    - Stores/updates the value of the input field into a variable
  - **ng-bind**
    - Replace the text content of the specified HTML with the value of given expression

ABOUT NAMING

- AngularJS HTML Compiler supports multiple formats
  - ng-bind
    - Recommended Format
  - data-ng-bind
    - Recommended Format to support HTML validation
  - ng_bind, ng:bind, x-ng-bind
    - Legacy, don't use

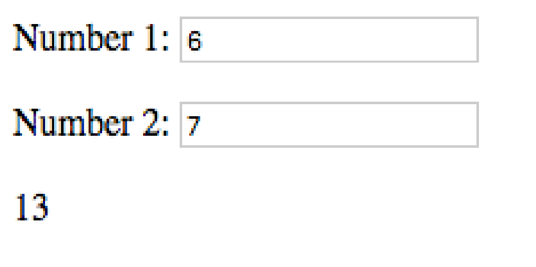
LOT OF BUILT IN DIRECTIVES

- ngApp
- ngClick
- ngController
- ngModel
- ngRepeat
- ngSubmit
- ngDblClick
- ngMouseEnter
- ngMouseMove
- ngMouseLeave
- ngKeyDown
- ngForm

2) EXPRESSIONS

- Angular **expressions** are JavaScript--like code snippets that are usually placed in bindings
  - `{{ expression }}.`
- Valid Expressions
  - `{{ 1 + 2 }}`
  - `{{ a + b }}`
  - `{{ items[index] }}`
- Control flow (loops, if) are not supported!
- You can use **filters** to format or filter data

EXAMPLE



Number 1: 6

Number 2: 7

13

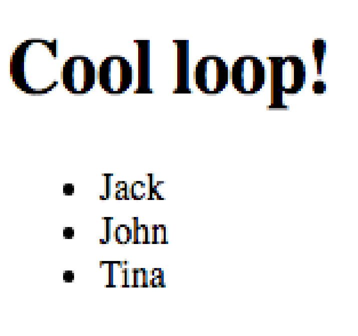
```
<!DOCTYPE html>
<html>
 <head>
 <title>Title</title>
 <meta charset="UTF-8" />
 <style media="screen"></style>
 <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js">
</script>
 </head>
 <body>
 <div ng-app>
 <p>Number 1: <input type="number" ng-model="number1"></p>
 <p>Number 2: <input type="number" ng-model="number2"></p>
 <!-- expression -->
 <p>{{ number1 + number2 }}</p>
 </div>
 </body>
</html>
```

Directive

Directive

Expression

NG-INIT AND NG-REPEAT DIRECTIVES



Cool loop!

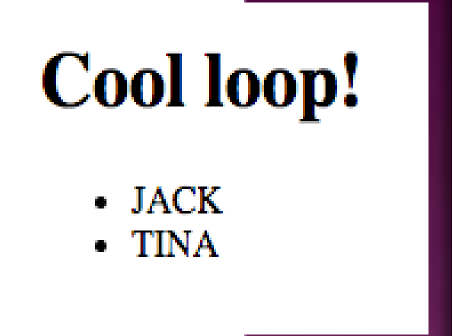
- Jack
- John
- Tina

```
<!DOCTYPE html>
<html data-ng-app="">
 <head>
 <title>Title</title>
 <meta charset="UTF-8" />
 <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/a
ngular.min.js"></script>
 </head>
<body>
<div data-ng-init="names = ['Jack', 'John', 'Tina']">
<h1>Cool loop!</h1>
<ul>
<li  data-ng-repeat="name in names">{{ name }}</li>
</ul>
</div>
</body>

</html>
```

3) FILTER

- With **filter,** you can **format or filter** the output
- **Formatting**
  - currency, number, date, lowercase, uppercase
- **Filtering**
  - filter, limitTo
- **Other**
  - orderBy, json

USING FILTERS – EXAMPLE



Cool loop!

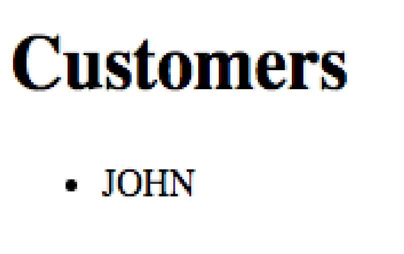
- JACK
- TINA

```
<!DOCTYPE html>
<html data-ng-app="">
 <head>
 <title>Title</title>
 <meta charset="UTF-8" />
 <style media="screen"></style>
 <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js”>
</script>
 </head>
 <body>
<div data-ng-init="customers = [{name:’tina'}, {name:’jack'}]">
<h1>Cool loop!</h1>
<ul>
<li data-ng-repeat="customer in customers | orderBy:'name'">
{{ customer.name | uppercase }}</li>
</ul>
</div>
</body>

</html>
```

Filter

Filter

USING FILTERS – EXAMPLE

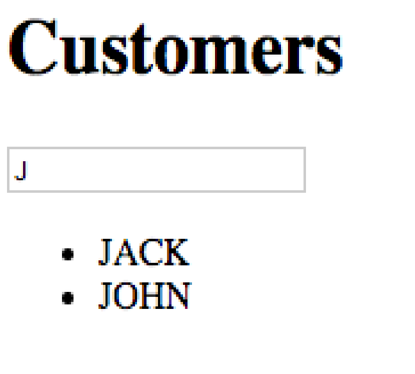


Customers

- JOHN

```
<!DOCTYPE html>
<html data-ng-app="">
 <head>
 <title>Title</title>
 <meta charset="UTF-8" />
 <style media="screen"></style>
 <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script
>
 </head>
 <body>
<div data-ng-init=
"customers = [{name:'jack'}, {name:'tina'}, {name:'john'}, {name:'donald'}]">
<h1>Customers</h1>
<ul>
<li data-ng-repeat="customer in customers | orderBy:'name' | filter:'john'">{{
customer.name | uppercase }}</li>
</ul>
</div>
</body>

</html>
```

USING FILTERS - USER INPUT FILTERS THE DATA



Customers

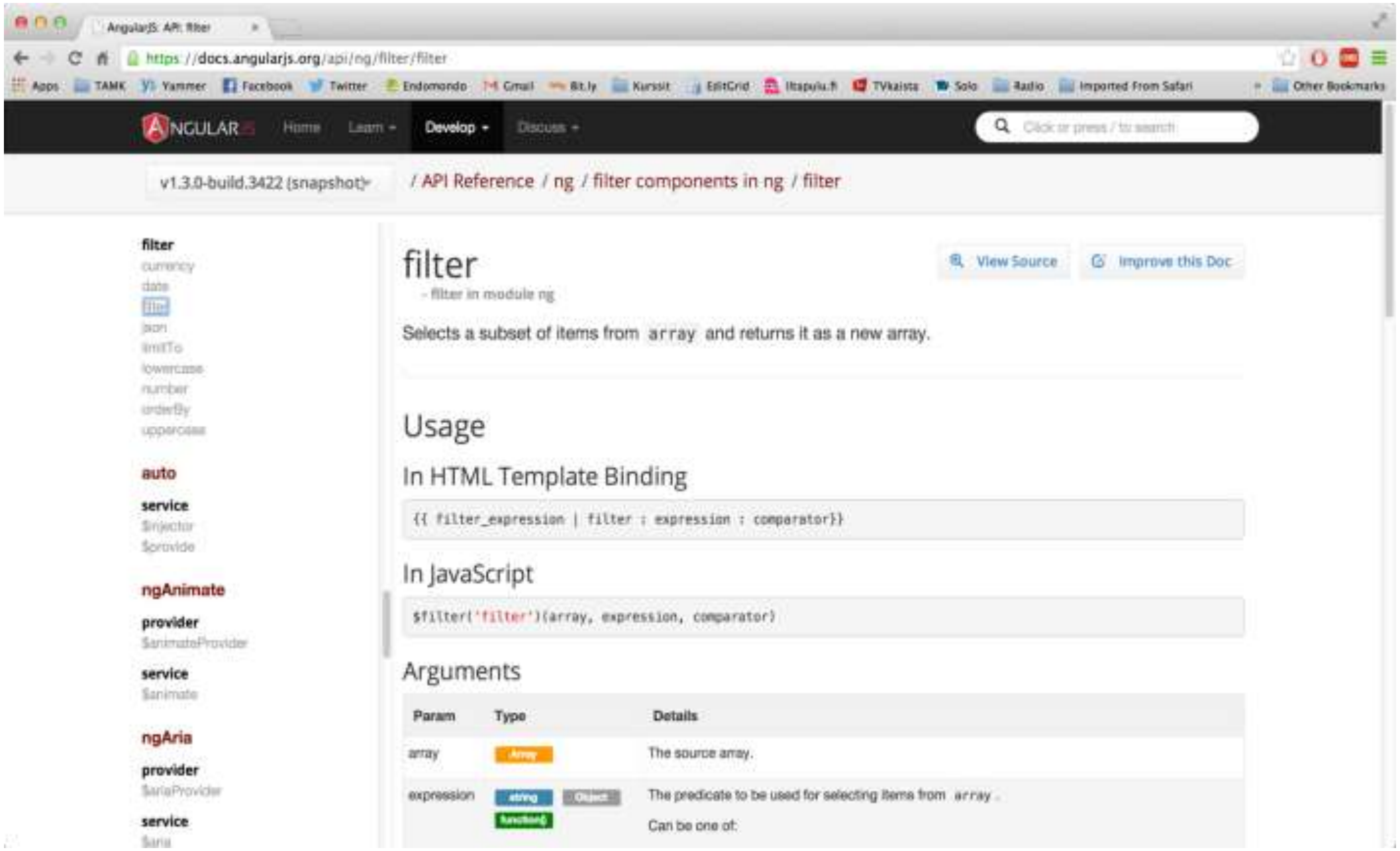
J

- JACK
- JOHN

```
<!DOCTYPE html>
<html data-ng-app="">
 <head>
 <title>Title</title>
 <meta charset="UTF-8" />
 <style media="screen"></style>
 <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js">
</script>
 </head>
 <body>
<div data-ng-init=
"customers = [{name:'jack'}, {name:'tina'}, {name:'john'},
{name:'donald'}]">
<h1>Customers</h1>

<input type="text" data-ng-model="userInput" />
<ul>
<li data-ng-repeat="customer in customers | orderBy:'name' |
filter:userInput">{{  customer.name | uppercase }}</li>
</ul>
</div>
</body>

</html>
```

API REFERENCE

https://docs.angularjs.org/api/ng/filter/filter



/ API Reference / ng / filter components in ng / filter

filter
currency
date
filter
json
limitTo
lowercase
number
orderBy
uppercase

auto

service
$injector
$provide

ngAnimate

provider
$animateProvider

service
$animate

ngAria

provider
$ariaProvider

service
$aria

filter

- filter in module ng

Selects a subset of items from `array` and returns it as a new array.

Usage

In HTML Template Binding

```
{{ filter_expression | filter : expression : comparator}}
```

In JavaScript

```
$filter('filter')(array, expression, comparator)
```

Arguments

Param	Type	Details
array	Array	The source array.
expression	string Object function()	The predicate to be used for selecting items from `array`. Can be one of:

VIEWS, CONTROLLERS, SCOPE

MODEL - VIEW – CONTROLLERS

- **Controllers** provide the **logic** behind your app.
  - – So use controller when you need logic behind your UI
- AngularJS apps are controller by controllers
- Use **ng--controller** to define the controller
- Controller **is a JavaScript Object, created by** standard **JS object constructor**

MODEL – VIEW – CONTROLLERS

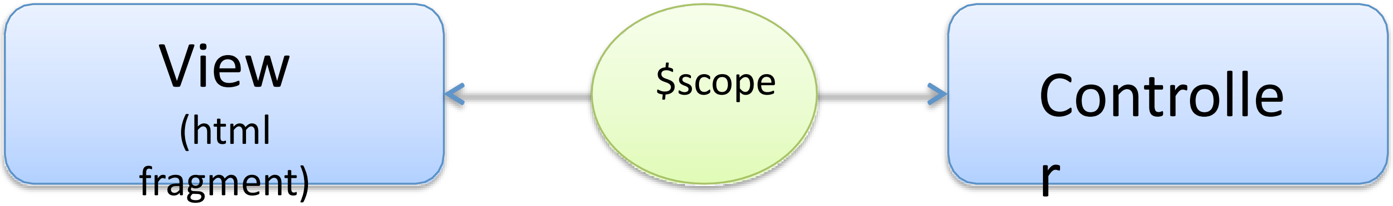
a controller is a JavaScript function

○ It contains data

○ It specifies the behavior

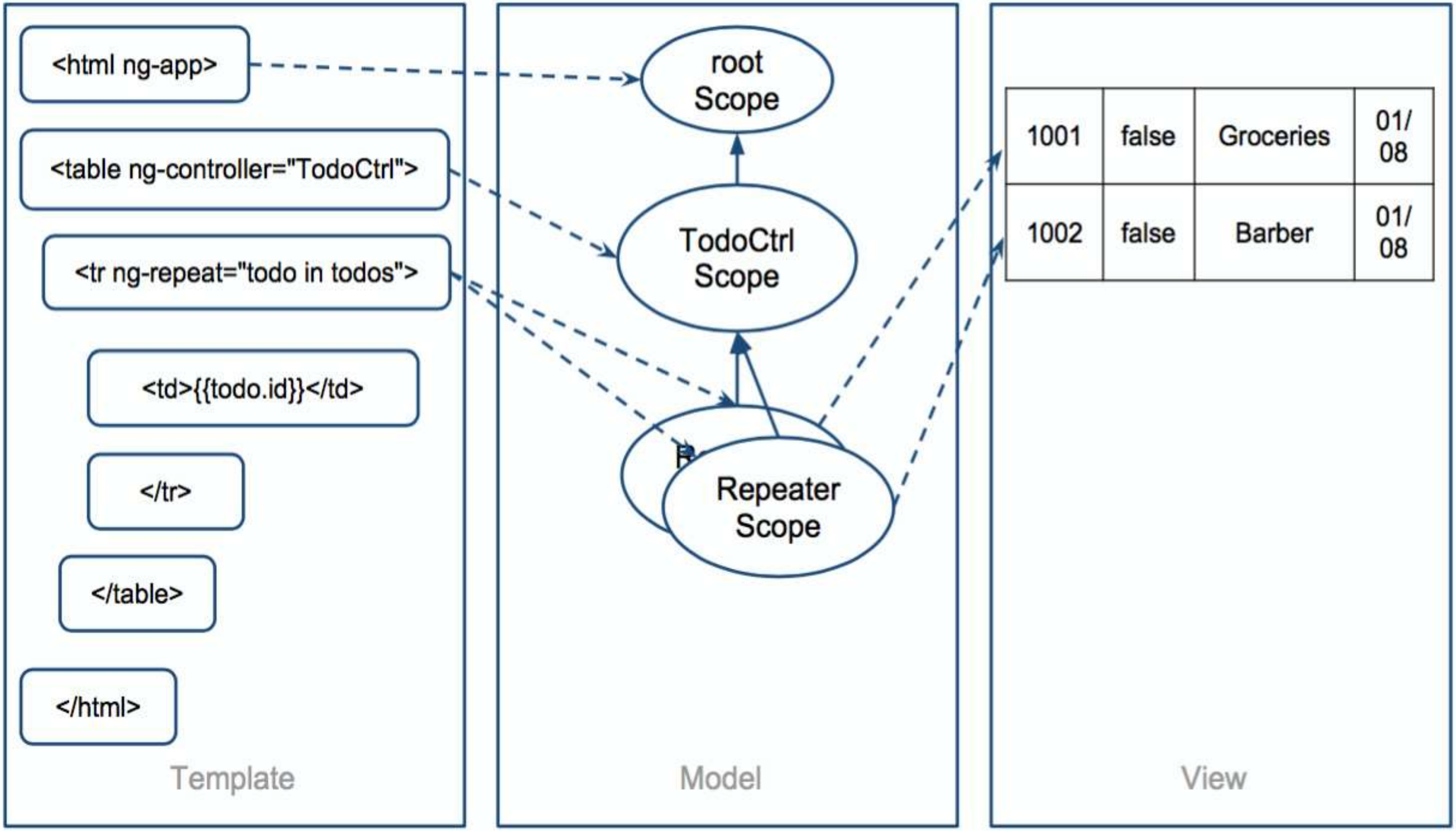
○ It should contain only the business logic needed for a single view.

VIEW, CONTROLLER AND SCOPE



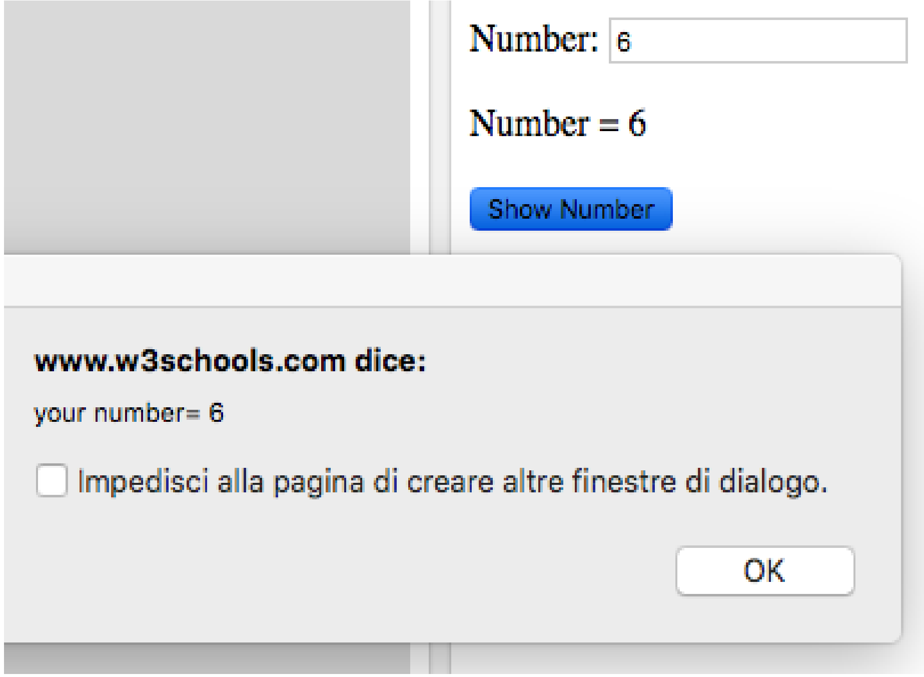
$scope is an object that can *be used to communicate* between View and Controller

SCOPE



1001	false	Groceries	01/ 08
1002	false	Barber	01/ 08

```
<!DOCTYPE html>
<html>
 <head>
 <title>Title</title>
 <meta charset="UTF-8" />
 <style media="screen"></style>
 <script src="https://ajax.googleapis.com/
ajax/libs/angularjs/1.4.8/angular.min.js">
</script>

 </head>
 <body>
<div data-ng-app="myApp" data-ng-controller="NumberCtrl">
<p>Number: <input type="number" ng-model="number"></p>
<p>Number = {{ number }}</p>
<button ng-click="showNumber()">Show Number</button>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('NumberCtrl', function($scope) {
    $scope.number = 1;
    $scope.showNumber = function(){
       window.alert( "your number= " + $scope.number );
    };
});
</script>
</body>
</html>
```



Number: 6

Number = 6

www.w3schools.com dice:

your number= 6

□ Impedisci alla pagina di creare altre finestre di dialogo.

MODULES

- **Module** is a reusable container for different features of your app
  - **Controllers**, services, filters, directives...
- If you have a lot of controllers, you are **polluting JS namespace**
- Modules can be loaded in any order
- We can build our **own filters** and **directives**!

When to use controllers

- Use controllers
  - set up the initial state of $scope object
  - add behavior to the $scope object
- Do not
  - Manipulate DOM (use **databinding**, **directives**)
  - Format input (use **form controls**)
  - Filter output (use **filters**)
  - Share code or state (use **services**)

APP EXPLAINED

- App runs inside **ng-app** (div)
- AngularJS will invoke the constructor with a $scope – object
- $scope is an object that links controller to the view

MODULES, ROUTES, SERVICES

EXAMPLE: OWN FILTER

```
// declarea module


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

 // configure the module.
 // in this example we will create agreeting filter
myAppModule.filter('greet', function() {
 return function(name) {
      return          ' + name +
             'Hello    '!';
      ,
    };
  });
```

HTML USING THE FILTER

```
<div ng-app="myApp">
 <div>
   {{ 'World'| greet }}
    </div>
  </div>
```

TEMPLATE FOR CONTROLLERS

```
// Create new module 'myApp' using angular.module
                                  method.
// The module is not dependent on any other module
var myModule = angular.module('myModule',
                                  []);

myModule.controller('MyCtrl', function ($scope) {
    // Your controller code here!
});
```

CREATING A CONTROLLER IN MODULE

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

myModule.controller('MyCtrl', function ($scope) {

    var model = { "firstname": "Jack",
                  "lastname": "Smith" };

    $scope.model = model;
    $scope.click = function() {
                        alert($scope.model.firstname
                        );
                  };
});
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <style
    media="screen"></style>

    <script  src="../angular.min.js"></script
    <script  >   src="mymodule.js"></script>
   </head>
   <body>
     <div ng-app="myModule"
       <div ng-
       controller="MyCtrl">
         <p>Firstname: <input type="text" ng-
         model="model.firstname"></p>
         <p>Lastname: <input type="text" ng-model="model.lastname"></p>
         <p>{{model.firstname + " " + model.lastname}}</p>


         <button ng-click="click()">Show Number</button>


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

This is now the model object from MyCtrl. Model object is shared with view and controller

ROUTING

ROUTING

- Since **we are building a SPA** app, everything happens in **one page**
  - How should **back--button** work?
  - How should **linking** between "pages" work?
  - How about **URLs?**
- **Routing** comes to rescue!

```
<html data-ng-app="myApp">
<head>
  <title>Demonstration of Routing -
  index</title>
  <meta charset="UTF-8" />
  <script src="../angular.min.js" type="text/javascript"></script>
  <script src="angular-route.min.js" type="text/javascript"></script>
  <script src="myapp.js" type="text/javascript">
</script>
</head>

<body>
  <div data-ng-
  view=""></div>
</body>
</html>
```

We will have to load additional module

The content of this will change dynamically

```
// This module is dependent on ngRoute. Load
ngRoute
// before this.
var myApp = angular.module('myApp', ['ngRoute']);
// Configure routing.
myApp.config(function($routeProvider) {
    // Usually we have different controllers for different views.
    // In this demonstration, the controller does nothing.
    $routeProvider.when('/', {
                    templateUrl: 'view1.html',
                    controller: 'MySimpleCtrl' });

    $routeProvider.when('/view2', {
                    templateUrl: 'view2.html',
                    controller: 'MySimpleCtrl'
                    });

    $routeProvider.otherwise({ redirectTo: '/' });
});

// Let's add a new controller to MyApp
myApp.controller('MySimpleCtrl', function ($scope)
{
});
```

VIEWS

- **view1.html:**

```
<h1>View 1</h1>
<p><a href="#/view2">To View 2</a></p>
```

- **view2.html:**

```
<h1>View 2</h1>
<p><a href="#/view1">To View 1</a></p>
```

WORKING IN LOCAL ENVIRONMENT

- If you get "cross origin requests are only supported for HTTP" ..
- Either
  - 1) Disable web security in your browser
  - 2) Use some web server and access files http://..
- To **disable** web security in chrome
  - `taskkill /F /IM chrome.exe`
  - `"C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --disable-web-security --allow-file-access- from-files`

SERVICES

- Controller should be very thin;
- Meaning, most of the business logic and persistent data in your application should be taken care of or stored in a services.
- For memory purposes, controllers are instantiated only when they are needed and discarded when they are not.
- Because of this, every time you switch a route or reload a page, Angular cleans up the current controller.
- Services however provide a means for keeping data around for the lifetime of an application while they also can be used across different controllers in a consistent manner.

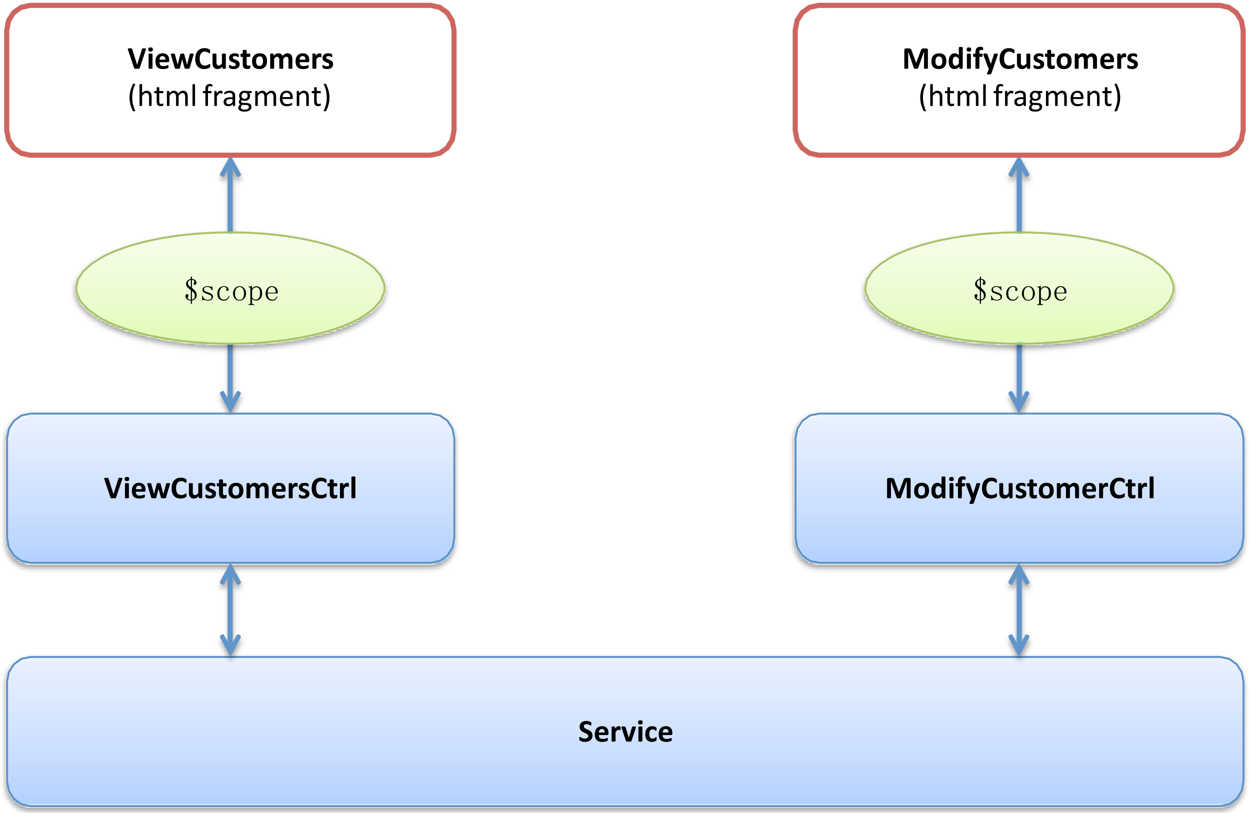
SERVICES

- View--independent **business logic** should **not be** in a controller
  - Logic should be in **a service component**
- **Controllers** are **view specific**, **services** are **app--spesific**
  - We can move from view to view and service is still alive
- Controller's responsibility is to bind model to view. Model can be fetched from service!
  - Controller is not responsible for manipulating (create, destroy, update) the data. **Use Services instead!**
- AngularJS **has many built--in services**, see
  - http://docs.angularjs.org/api/ng/service
  - Example: `$http`

SERVICES

- Angular provides us with three ways to create and register our own service.
  - Factory
  - Service
  - Provider

SERVICES


ViewCustomers
(html fragment)
ModifyCustomers
(html fragment)
$scope
$scope
ViewCustomersCtrl
ModifyCustomerCtrl
Service

FACTORY

- When you’re using a **Factory** you create an object, add properties to it, then return that same object.
- When you pass this service into your controller, those properties on the object will now be available in that controller through your factory.

ANGULARJS CUSTOM SERVICES USING FACTORY

```
// Let's add a new controller to MyApp. This controller uses
Service!  myApp.controller('ViewCtrl', function ($scope,
CustomerService) {
    $scope.contacts = CustomerService.contacts;
});

// Let's add a new controller to MyApp. This controller uses
Service!  myApp.controller('ModifyCtrl', function ($scope,
CustomerService) {
    $scope.contacts = CustomerService.contacts;
});

// Creating a factory object that contains services for the
// controllers.
myApp.factory('CustomerService', function()
{
    var factory = {};
    factory.contacts = [{name: "Jack", salary: 3000}, {name:
"Tina",  salary: 5000}, {name: "John", salary: 4000}];
    return factory;
});
```

SERVICE

- When you’re using **Service**, it’s instantiated with the ‘new’ keyword.
- Because of that, you’ll add properties to ‘this’ and the service will return ‘this’.
- When you pass the service into your controller, those properties on ‘this’ will now be available on that controller through your service.

ALSO SERVICE

```
// Service is instantiated with new - keyword.
// Service  function can use "this" and the return
//  value is this.



     myApp.service('CustomerService', function()
        {   this.contacts              =

              [{name:  "Jack",  salary:  3000},

               {name:  "Tina",  salary:  5000},
               {name:  "John",  salary:  4000}];

       });
```

PROVIDERS

- **Providers** are the only service you can pass into your .config() function. Use a provider when you want to provide module-wide configuration for your service object before making it available.

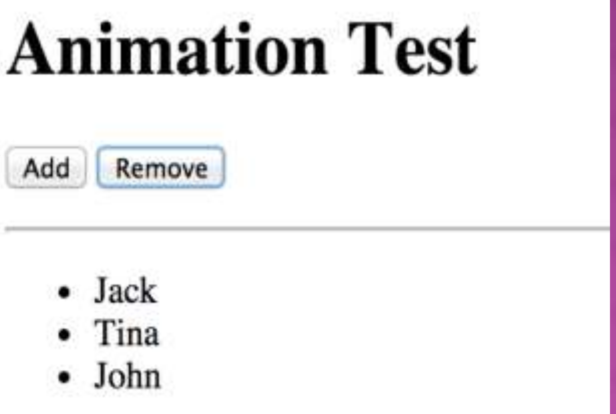
ANIMATIONS AND UNIT TESTING

ANGULARJS ANIMATIONS

- Include ngAnimate module as dependency
- Hook animations for common directives such as ngRepeat, ngSwitch, ngView
- Based on CSS classes
  - If HTML element has class, you can animate it
- AngularJS adds special classes to your html- - elements

EXAMPLE FORM

```
<body ng-controller="AnimateCtrl">
  <button ng-click="add()">Add</button>
  <button ng-
  click="remove()">Remove</button></p>
  <ul>
    <li ng-repeat="customer in
customers">{{customer.name}}</l
i>
  </ul>
</body>
```



Animation Test

- Jack
- Tina
- John

Adds and Removes names

ANIMATION CLASSES

- When adding a new name to the model, ng– repeat knows the item that is either added or deleted
- CSS classes are added at runtime to the repeated element (<li>)
- When adding new element:
  - <li class=″... ng-enter ng-enter-active″>New Name</li>
- When removing element
  - <li class=″... ng-leave ng-leave-active″>New Name</li>

DIRECTIVES AND CSS

Event	Starting CSS	Ending CSS	Directives
enter	.ng--enter	.ng--enter--active	ngRepeat, ngInclude, ngIf, ngView
leave	.ng--leave	.ng--leave--active	ngRepeat, ngInclude, ngIf, ngView
move	.ng--move	.ng--move.active	ngRepeat

EXAMPLE CSS

```
/* starting animation
*/
.ng-enter {
  -webkit-transition:
  1s;  transition: 1s;
  margin-left: 100%;
}

/* ending animation */
.ng-enter-active {
  margin-left: 0;
}

/* starting animation
*/
.ng-leave {
  -webkit-transition:
  1s;  transition: 1s;
  margin-left: 0;
}

/* ending animation */
.ng-leave-active {
  margin-left: 100%;
}
```

TEST DRIVEN DESIGN

- Write tests firsts, then your code
- AngularJS emphasizes modularity, so it can be easy to test your code
- Code can be tested using several unit testing frameworks, like QUnit, Jasmine, Mocha ...

QUNIT

- Download `qunit.js` and `qunit.css`
- Write a simple HTML page to run the tests
- Write the tests

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>QUnit Example</title>
  <link rel="stylesheet" href="qunit-
  1.10.0.css">
  <script src="qunit-1.10.0.js"></script>
</head>
<body>
  <div id="qunit"></div>
  <script type="text/javascript">

    function calculate(a, b) {
        return a + b;
    }
    test( "calculate test", function()
      ok( {calculate(5,5) ===    "Ok!
      10,                        "     );
      ok( calculate(5,0) ===     "Ok!  );
      5,  ok( calculate(-5,5)    "     );
      === 0,                     "OK!
                                 "
    });
  </script
  >
</body>
</html>
```

THREE ASSERTIONS

- Basic
  - ok(boolean [, message]);
- If *actual* == *expected*
  - equal(actual, expected [, message]);
- if *actual* === *expected*
  - deepEqual(actual, expected [, message));
- Other
  - http://qunitjs.com/cookbook/#automating- unit-testing

WRAPPING UP

WRAPPING UP

- AngularJS is a modular JavaScript SPA framework
- Lot of great features, but learning curve can be hard
- Great for CRUD (create, read, update, delete) apps, but not suitable for every type of apps
- Works very well with some JS libraries (JQuery)

AJAX + REST

AJAX

- **Asynchronous JavaScript + XML**
  - XML not needed, **very oden JSON**
- Send data and retrieve asynchronously from server in background
- **Group of technologies**
  - HTML, CSS, DOM, XML/JSON, XMLHttpRequest object and JavaScript

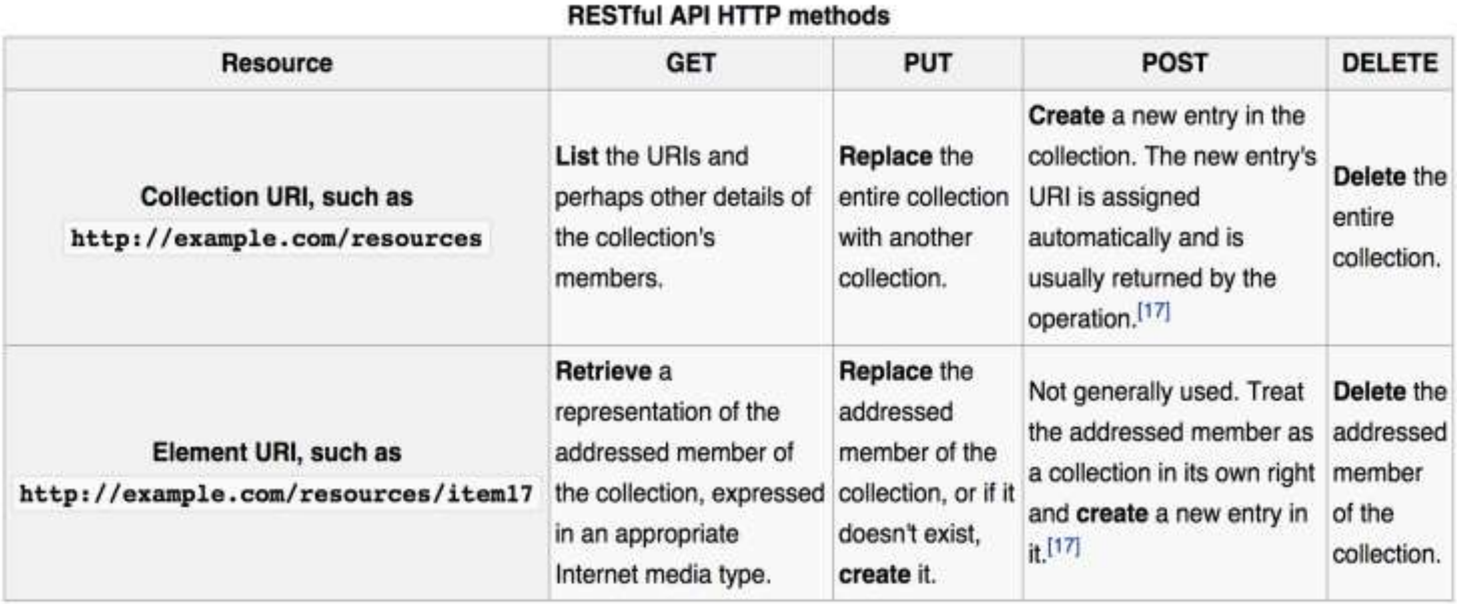
$HTTP - EXAMPLE (AJAX) AND ANGULARJS

```
<script type="text/javascript">
    var myapp = angular.module("myapp", []);

    myapp.controller("MyController", function($scope, $http) {
            $scope.myData = {};
            $scope.myData.doClick = function(item, event) {
                var responsePromise = $http.get("text.txt");

                responsePromise.success(function(data, status, headers, config) {
                  $scope.myData.fromServer = data;
                });
                responsePromise.error(function(data, status, headers,
                    config) {  alert("AJAX failed!");
                });
            }
        } );
  </script>
```

RESTFUL

- Web Service APIs that adhere to REST architectural constrains are called RESTful
- Constrains
  - – Base URI, such as http://www.example/resources
  - – Internet media type for data, such as JSON or XML
  - – Standard HTTP methods: GET, POST, PUT, DELETE
  - – Links to reference reference state and related resources

RESTFUL API HTTP METHODS (WIKIPEDIA)



RESTful API HTTP methods

Resource	GET	PUT	POST	DELETE
Collection URI, such as `http://example.com/resources`	**List** the URIs and perhaps other details of the collection's members.	**Replace** the entire collection with another collection.	**Create** a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.[17]	**Delete** the entire collection.
Element URI, such as `http://example.com/resources/item17`	**Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	**Replace** the addressed member of the collection, or if it doesn't exist, **create** it.	Not generally used. Treat the addressed member as a collection in its own right and **create** a new entry in it.[17]	**Delete** the addressed member of the collection.

AJAX + RESTFUL

- The web app can fetch using RESTful data from server
- Using AJAX this is done asynchronously in the background
- AJAX makes HTTP GET request using url ..
  - – http://example.com/resources/item17
- .. and receives data of item17 in JSON ...
- .. which can be displayed in view (web page)

EXAMPLE: WEATHER API

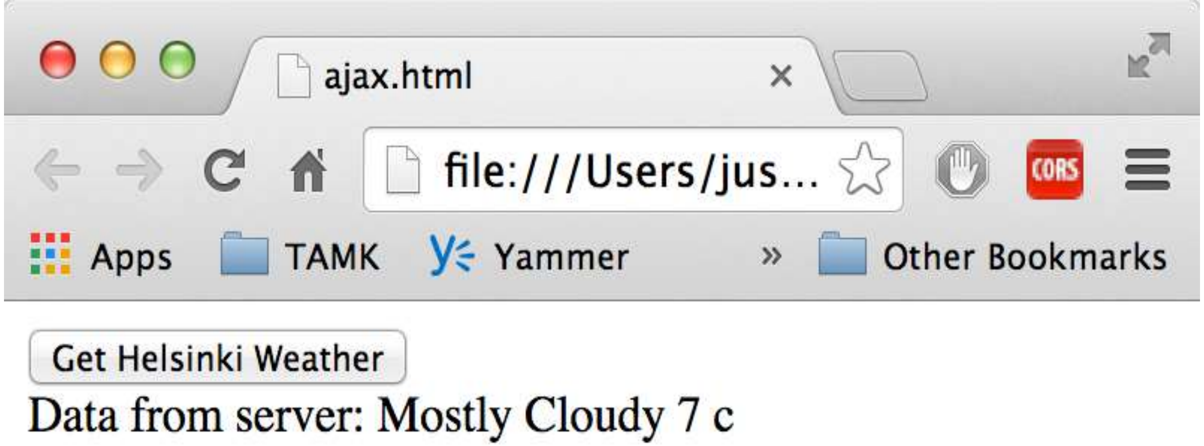
- Weather information available from wunderground.com
  - You have to **make account** and receive a **key**
- To get Helsinki weather in JSON
  - http://api.wunderground.com/api/***your-key***/ conditions/q/Helsinki.json

```
{
  "response": {
    "version":
    "0.1",
    "termsofService":
    "http:\/\/www.wunderground.com\/weather\/api\/d\/terms.html",
    "features": {
      "conditions"
      : 1
    }
  },
  "current_observation"
    : {  "image": {
      "url":
      "http:\/\/icons.wxug.com\/graphics\/wu2\/logo_130x80.png",
      "title": "Weather Underground",
      "link": "http:\/\/www.wunderground.com"
    },
    "display_location": {
      "full": "Helsinki,
      Finland",  "city":
      "Helsinki",
      "state": "",
      "state_name":
      "Finland",
      "country": "FI",
      "country_iso3166":
      "FI",  "zip":
      "00000",
      "magic": "1",
      "wmo": "02974",
      "latitude": "60.31999969",
      "longitude": "24.96999931",
      "elevation": "56.00000000"
    }
```

```
<!DOCTYPE html>
<html>
<head>
  <script src="../angular.min.js" type="text/javascript"></script>
  <title></title>
</head>

<body data-ng-app="myapp">
  <div data-ng-controller="MyController">
    <button data-ng-click="myData.doClick(item, $event)">Get Helsinki
    Weather</button><br />  Data from server: {{myData.fromServer}}
  </div>


<script type="text/javascript">
    var myapp = angular.module("myapp",
    []);
    myapp.controller("MyController", function($scope,
    $http) {
            $scope.myData = {};
            $scope.myData.doClick = function(item, event) {
                var responsePromise =
$http.get("http://api.wunderground.com/api/key/conditions/  q/Helsinki.json");

                responsePromise.success(function(data, status, headers, config) {
                    $scope.myData.fromServer = "" + data.current_observation.weather +
                                              " " + data.current_observation.temp_c + " c";
                });
                responsePromise.error(function(data, status, headers, config) {
                    alert("AJAX failed!");
                });
            }
        } );
  </script>
</body>
</html>
```

This is JSON object!

VIEW AFTER PRESSING THE BUTTON



Data from server: Mostly Cloudy 7 c

$RESOURCE

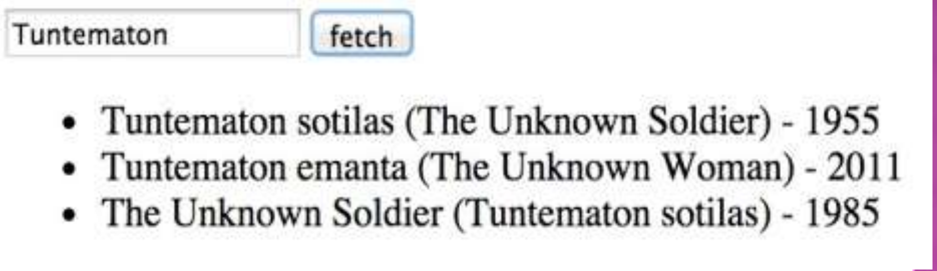
- Built on top of $http service, $resource is a factory that lets you interact with RESTful backends easily
- $resource does not come bundled with main Angular script, separately download
  - angular-resource.min.js
- Your main app should declare dependency on the ngResource module in order to use $resource

GETTING STARTED WITH $RESOURCE

- $resource expects classic RESTful backend
  - http://en.wikipedia.org/wiki/Representational_state_transfer#Applied_to_web_services
- You can create the backend by whatever technology. Even JavaScript, for example Node.js
- We are not concentrating now how to build the backend.

USING $RESOURCE ON GET

```
// Load ngResource before this
var restApp = angular.module('restApp',['ngResource']);

restApp.controller("RestCtrl", function($scope, $resource) {
    $scope.doClick = function() {
            var title = $scope.movietitle;
            var searchString =
'http://api.rottentomatoes.com/api/
public/v1.0/movies.json?apikey=key&q=' + title + '&page_limit=5';

            var result = $resource(searchString);

            var root = result.get(function()     // {method:'GET'
            {
                $scope.movies = root.movies;
        }   });
});
```



- Tuntematon sotilas (The Unknown Soldier) - 1955
- Tuntematon emanta (The Unknown Woman) - 2011
- The Unknown Soldier (Tuntematon sotilas) - 1985

$RESOURCE METHODS

- $resource contains convenient methods for
  - get （’GET’）
  - save （’POST’）
  - query （’GET’， isArray:true)
  - remove （’DELETE’）
- Calling these will invoke $http (ajax call) with the specified http method (GET, POST, DELETE), destination and parameters

PASSING PARAMETERS

```
// Load ngResource before this
var restApp =
angular.module('restApp',['ngResource']);

 restApp.controller("RestCtrl", function($scope, $resource)
                             {
    $scope.doClick = function() {
            var searchString =
'http://api.rottentomatoes.com/api/public/
v1.0/movies.json?apikey=key&q=:title&page_limit=5';
            var result = $resource(searchString);
            var root = result.get({title: $scope.movietitle}, function()
            {
        }       $scope.movies = root.movies;
});         });
```

:title -> parametrized URL template

Giving the parameter from $scope

USING SERVICES

Controller responsible for binding

```
// Load ngResource before this
var restApp =
angular.module('restApp',['ngResource']);
restApp.controller("RestCtrl", function($scope, MovieService) {
  $scope.doClick = function() {
           var root = MovieService.resource.get({title:
           $scope.movietitle},
           function() {
               $scope.movies = root.movies;
           });
})      }
;

restApp.factory('MovieService',
  function($resource) {  factory = {};
  factory.resource =
  $resource('http://api.rottentomatoes...&q=:title&page_limit=5');
  return factory;
});
```

Service responsible for the

SIMPLE VERSION

Just call get from MovieService

```
// Load ngResource before this
var restApp =
angular.module('restApp',['ngResource']);
restApp.controller("RestCtrl",               MovieService)
function($scope,                             {
   $scope.doClick = function() {
           var root =                        $scope.movietitle
           MovieService.get({title:          },
           function() {
           })
        }  ;  $scope.movies = root.movies;
})
;
```

Returns the resource

```
restApp.factory('MovieService', function($resource) {
     return
     $resource('http://api.rottentomatoes...&q=:title&page_limit=5');;
});
```