Building Single Page Applications with Angular 2

John Papa

Dan Wahlin

About Us

Dan Wahlin

@DanWahlin

http://blog.codewithdan.com

John Papa

@John_Papa

http://johnpapa.net

Get the Content

http://tinyurl.com/ngConfMay16

Quickstart

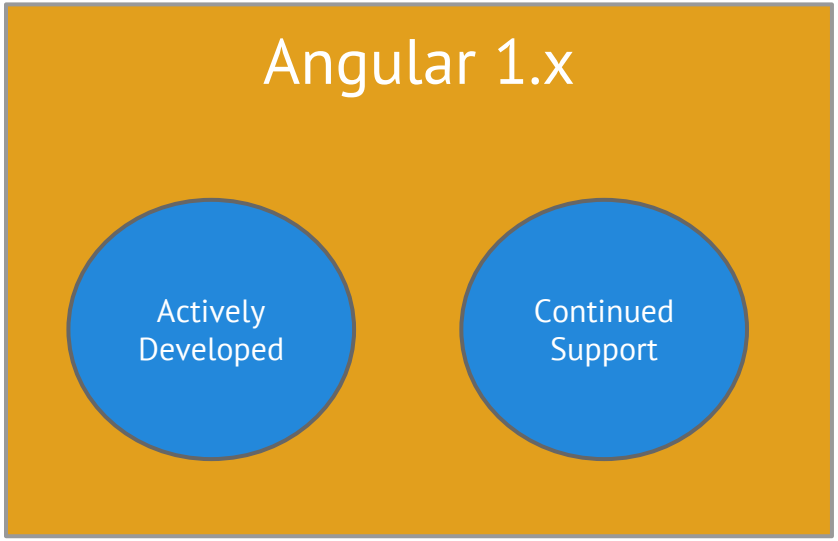
```
git clone  https://github.com/angular/quickstart ab16
cd ab16
npm i
npm start
```

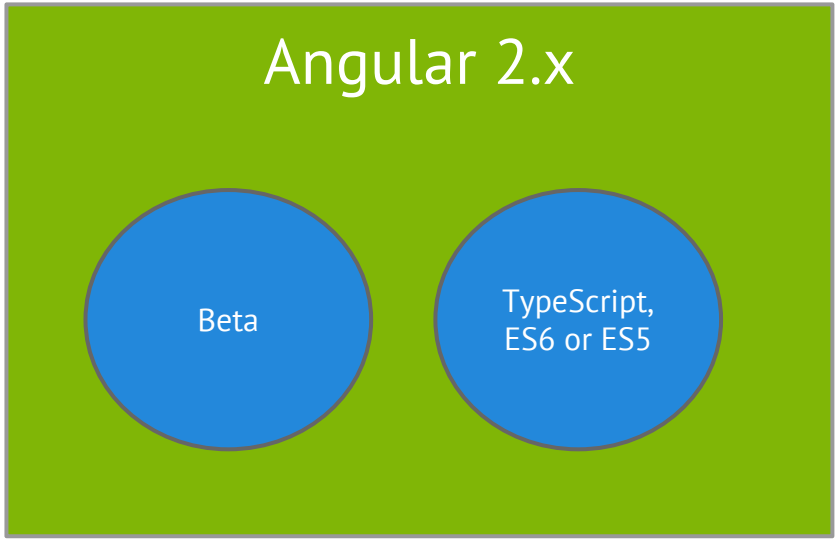
Agenda

- Introduction to Angular 2
- Angular 1 to Angular 2
- Modules, Components and Templates
- Binding and Directives
- Services and DI
- Http
- Routing

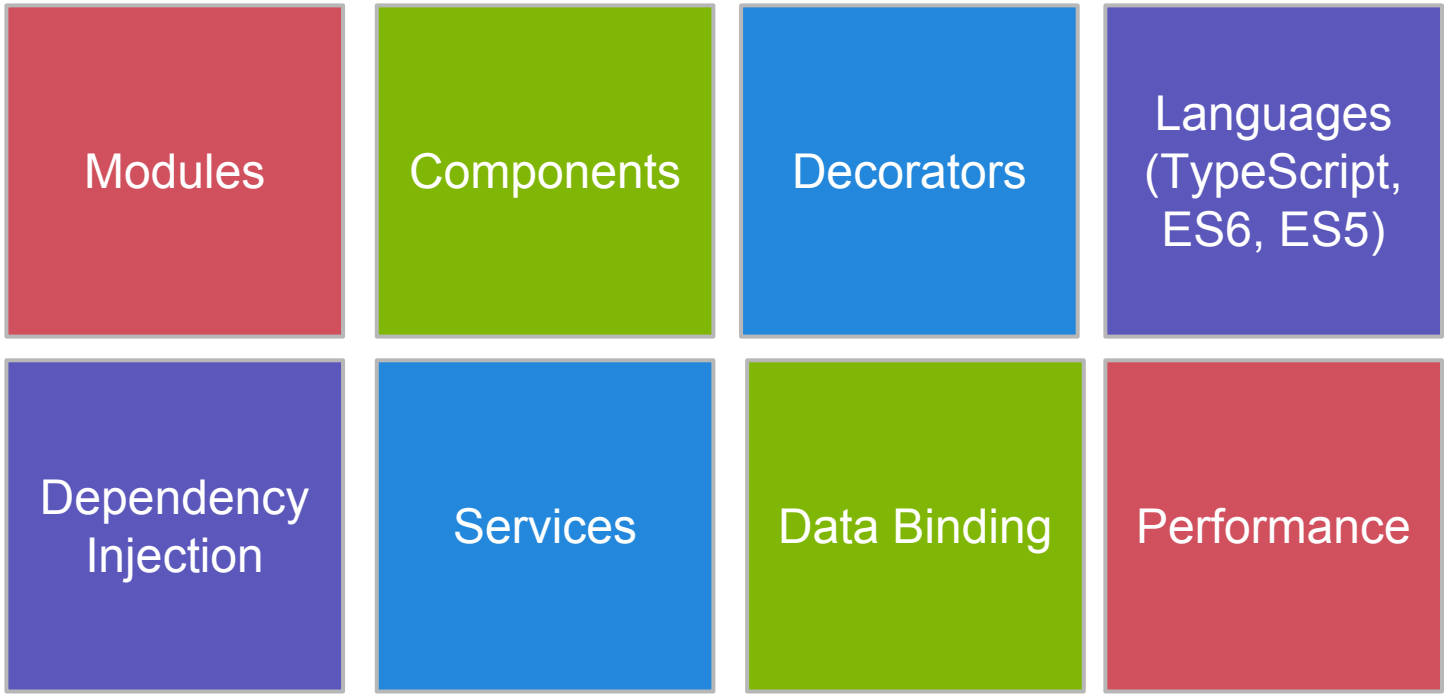
Introduction to Angular 2

Angular Versions

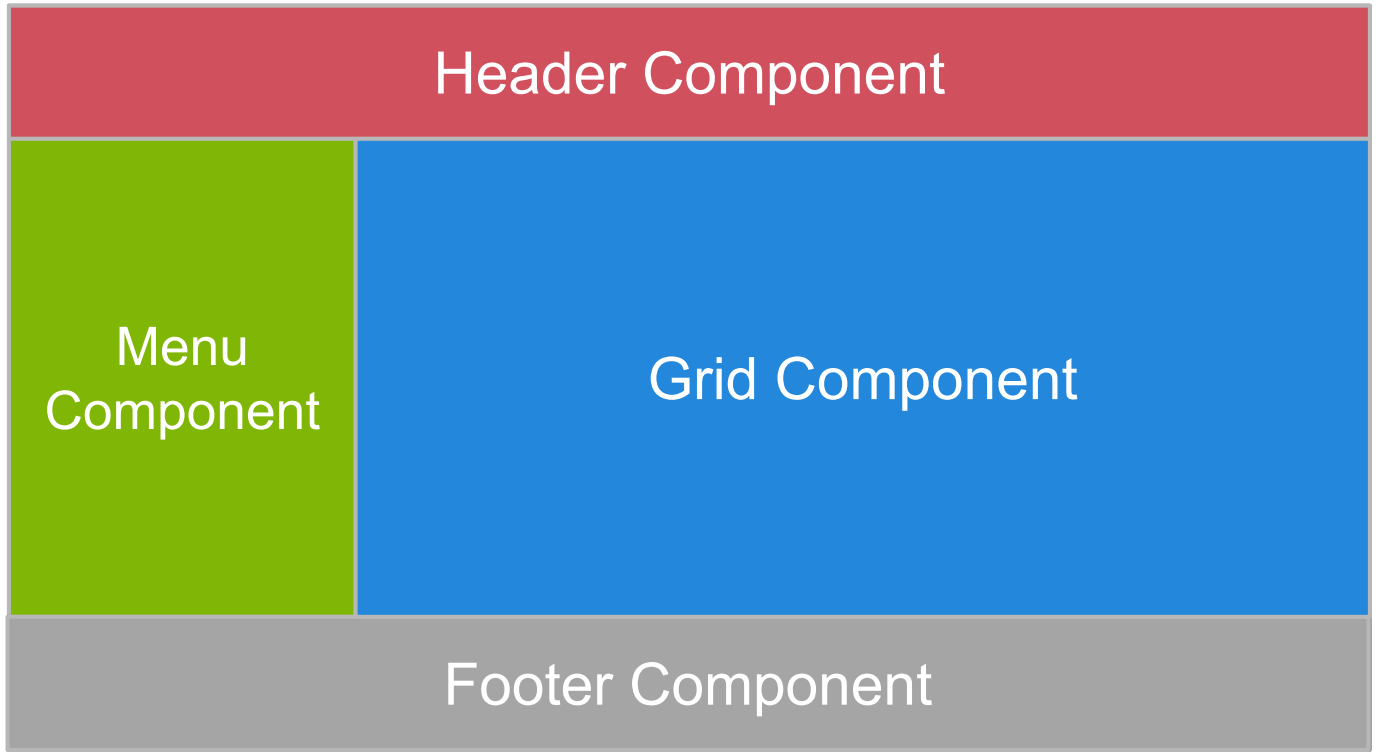




Angular 2 Overview



The Big Picture



https://github.com/DanWahlin/Angular2-JumpStart

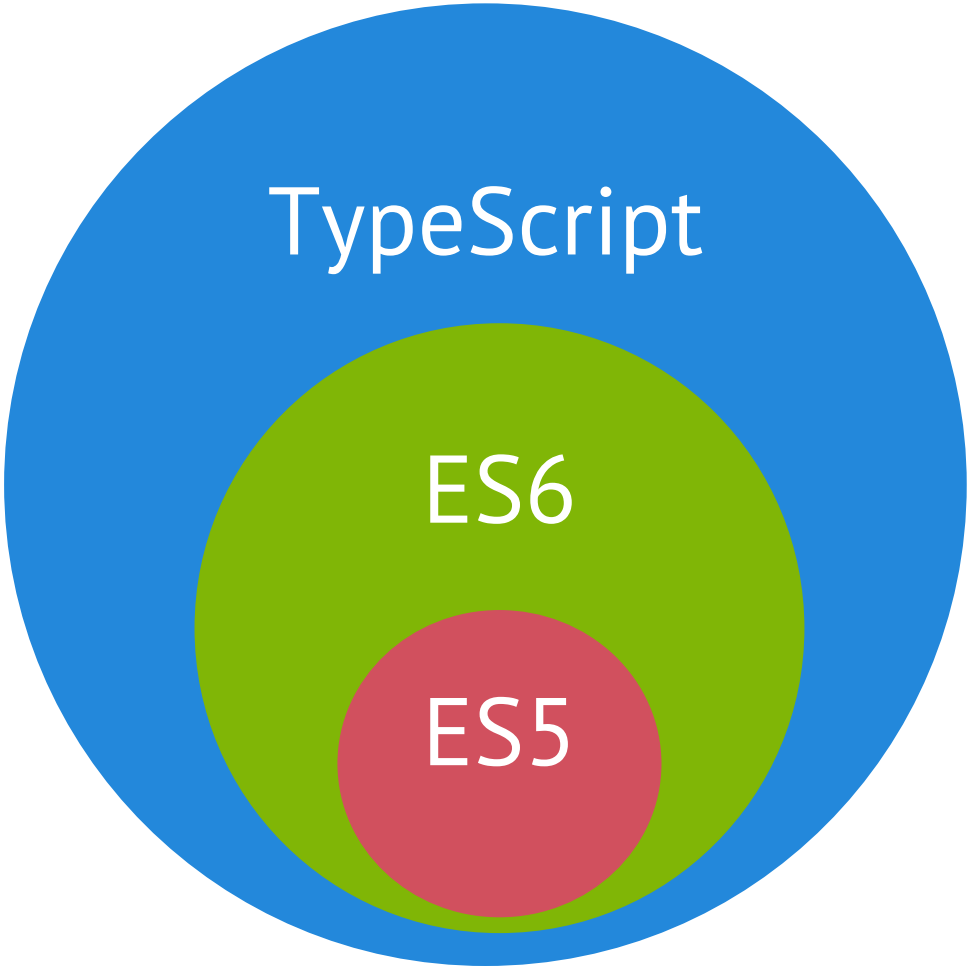
Demo

Angular 2 JumpStart

https://github.com/johnpapa/event-view/

Demo

Angular 2 Overview


TypeScript
ES6
ES5

Using TypeScript

- The future of JavaScript is ES6/ES2015
  - Major update to the JavaScript language
  - Modules, classes and more
  - Will help you align with Angular 2

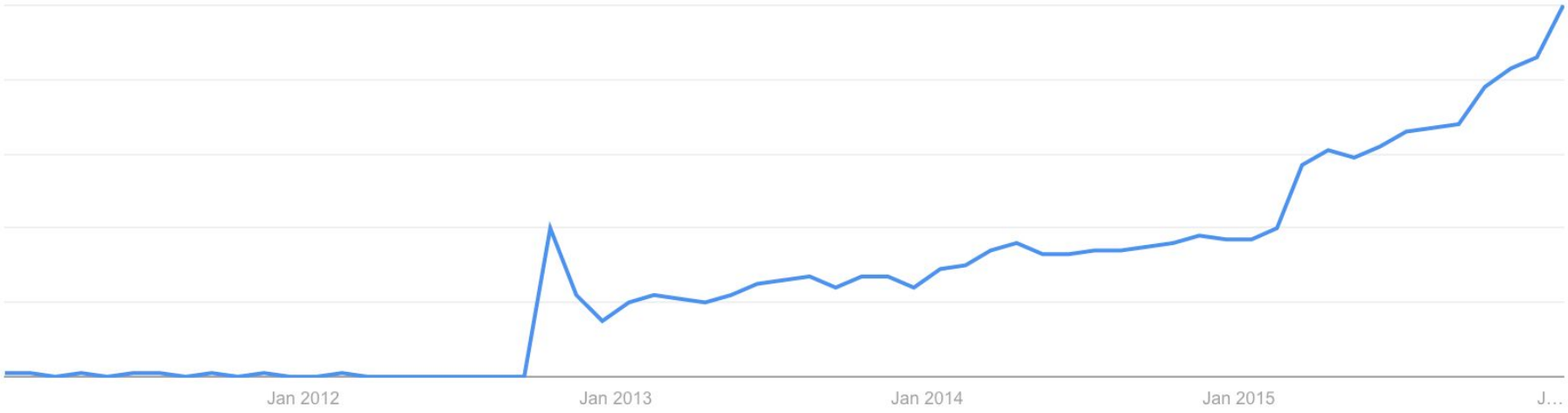
- TypeScript builds on top of ES6

typescript
Search term

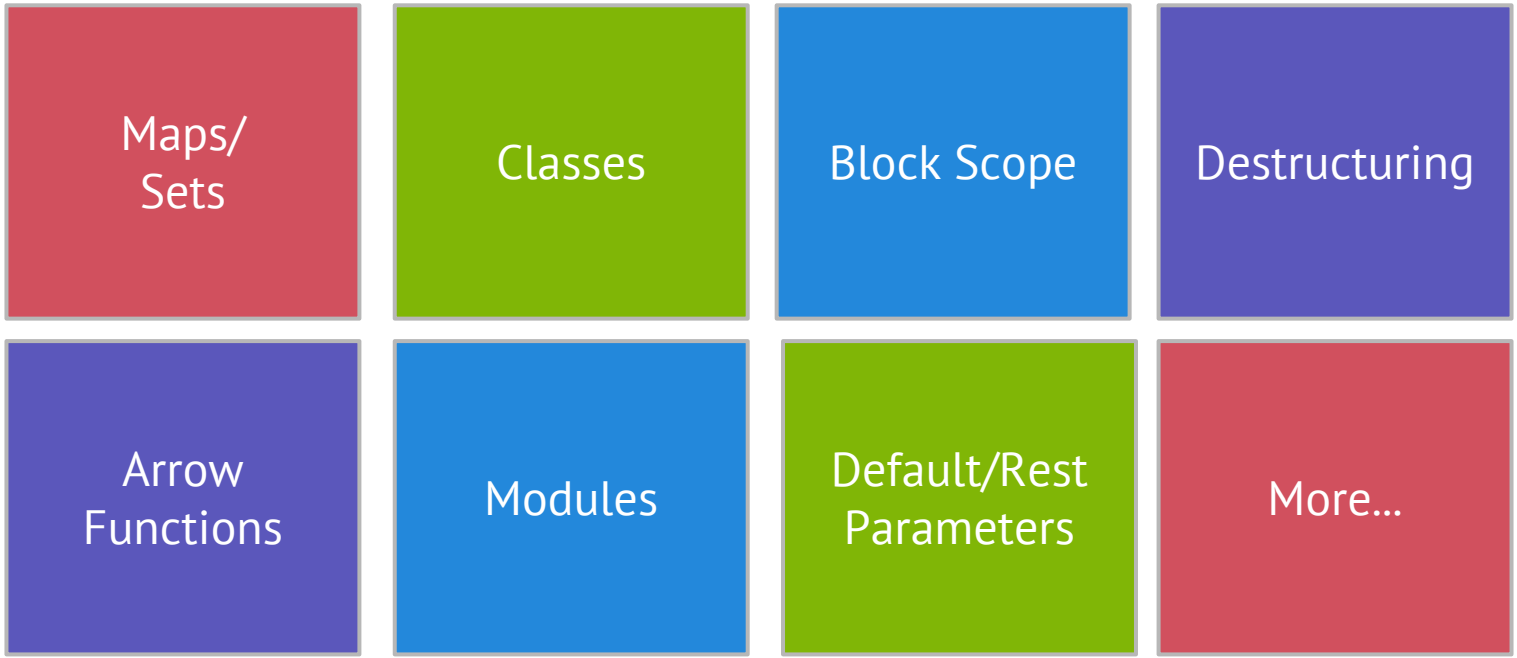
+ Add term

Interest over time

Compare to category News headlines Forecast

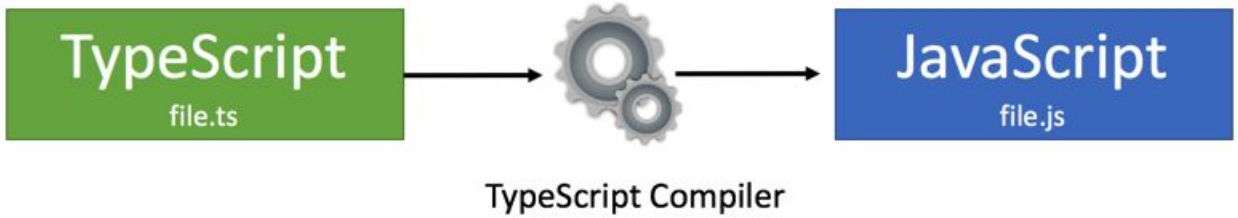


Key ES6 Features



https://github.com/DanWahlin/ES6Samples

TypeScript Compilation



http://www.typescriptlang.org/Playground

Demo

Getting Started with TypeScript

http://jpapa.me/a2firstlook

Angular 1 to Angular 2

1 Controllers to Components

Angular 1

```
<body ng-controller="StoryController as vm">
  <h3>{{ vm.story.name }}</h3>
</body>

(function() {
  angular
  .module('app')
  .controller('StoryController',StoryController);

  function StoryController() {
    var vm = this;
    vm.story = { id:100, name:'Star Wars' };
  }
})();
```

Angular 2

```
<my-story></my-story>



import { Component } from '@angular/core';

@Component({
  selector: 'my-story',
  template: '<h3>{{story.name}}</h3>'
})
export class StoryComponent {
  story = {id: 100, name: 'Star Wars' };
}
```

2

Structural Built-In Directives

Angular 1

```
<ul>
  <li ng-repeat="vehicle in vm.vehicles">
    {{ vehicle.name }}
  </li>
</ul>

<div ng-if="vm.vehicles.length">
  {{ vm.vehicles.length}} vehicles
</div>
```

Angular 2

```
<ul>
  <li *ngFor="let vehicle of vehicles">
    {{ vehicle.name }}
  </li>
</ul>

<div *ngIf="vehicle.length">
  {{ vehicles.length}} vehicles
</div>
```

3

Data Binding

DOM

Interpolation

One Way Binding

Event Binding

Two Way Binding

Component

Interpolation

Angular 1

```
<h3>{{vm.story.name}}</h3>
```

Context

Angular 2

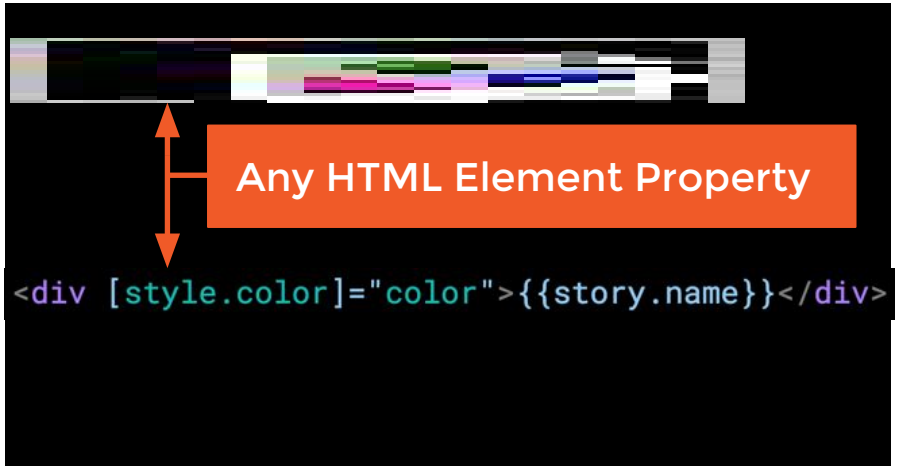
```
<h3>{{story.name}}</h3>
```

1 Way Binding

Angular 1

```
<h3 ng-bind="vm.story.name"></h3>
```

Angular 2



Event Binding

Angular 1

```
<button
  ng-click="vm.log('click')"
  ng-blur="vm.log('blur')">OK</button>
```

Angular 2

```
<button
  (click)="log('click')"
  (blur)="log('blur')">OK</button>
```

2 Way Binding on an <INPUT>

Angular 1

```
<input ng-model="vm.story.name">
```

Angular 2

```
<input [(ngModel)]="story.name">
```

4 Removes the Need for Many Directives

Angular 1

```
<div ng-style=
  "vm.story ?
    {visibility: 'visible'}
    : {visibility: 'hidden'}">

  <img ng-src="{{vm.imagePath}}">
  <br/>
  <a ng-href="{{vm.link}}">
    {{vm.story}}
  </a>

</div>
```

Angular 2

```
<div [style.visibility]=
  "story ? 'visible' : 'hidden'">

  <img [src]="imagePath">
  <br/>
  <a [href]="link">{{story}}</a>

</div>
```

No Longer Need these Directives Either

Angular 1	Angular 2
ng-click="saveVehicle(vehicle)”	(click)="saveVehicle(vehicle)"
ng-focus="log('focus')”	(focus)="log('focus')"
ng-blur="log('blur')”	(blur)="log('blur')"
ng-keyup="checkValue()"	(keyup)="checkValue()"

Angular 2 Template Concepts Remove 40+ Angular 1 Built-In Directives

5

Services

Angular 1	Angular 2
Factories	
Services	
Providers	Class
Constants	
Values	

Angular 2 Application Walkthrough Exercise

http://plnkr.co/edit/iHEkgOLj4E7lR9urOz2M

Modules, Components and Templates

Angular 2 provides a robust library of modules:

https://angular.io/docs/ts/latest/api

http://github.com/danwahlin/angular2-barebones

Demo

Angular 2 Bare Bones

Steps to Build Components

1. Import/export required modules
2. Define component class
3. Add @Component decorator to class
4. Create a template

Steps to Build Components

1. Import/export required modules
2. Define component class
3. Add @Component decorator to class
4. Create a template

The Role of Modules

- Modules separate code into separate "buckets"
- Rely on **export** and **import** keywords
- Browsers need help with modules
- System.js (and others) load modules

http://www.2ality.com/2014/09/es6-modules-final.html

Exporting Modules

Classes, functions and variables can be exported using the **export** keyword

my.component.ts

```
export class CustomersComponent {
    ...
}
```

Importing Modules

Modules can be imported using the **import** keyword

customers.component.ts

```
import { Component  } from '@angular/core';
import { DataService } from '../services/data-service';

...
export class CustomersComponent {
   ...
}
```

Module Loader: System.js

Loading ES6 modules in the browser

index.html

```
<script src="node_modules/zone.js/dist/zone.js"></script>
<script src="node_modules/reflect-metadata/Reflect.js"></script>
<script src="node_modules/systemjs/dist/system.js"></script>

<script src="system.config.js"></script>
<script>
    System.import('app').catch(function(err){
        console.error(err);
    });
</script>
```

Import this module as a starting point

Steps to Build Components

1. Import/export required modules
2. Define component class
3. Add @Component decorator to class
4. Create a template

What's a Component?

- Components are reusable objects
- A component consists of:

- Has a "selector": <customers></customers>

Components Overview

- Building blocks of Angular apps
- A portion of the screen (a "view")
- Import functionality from modules
- Use a @Component decorator to define metadata
- Accept input and handle template events
- Have a life-cycle
- Delegate to services (more on this later!)

What's in a Component?

imports

```
import { Component } from '@angular/core';
import { DataService } from '../services/data-service';
```

decorators

```
@Component({
  ...
})
```

class

```
export class CustomersComponent {

}
```

Steps to Build Components

1. Import/export required modules
2. Define component class
3. Add @Component decorator to class
4. Create a template

The @Component Decorator

- Decorators provide metadata for a component class
- @Component imported from **@angular/core** module
- Key properties:

Property	Description
selector	Defines the selector that triggers instantiation of the component (ex: 'customers' = <customers></customers>
template & templateUrl	Defines the template used by the component
providers	Defines objects injected into the component
directives	Defines custom directives used in the component template
pipes	Defines custom pipes used in the component template

Using @Component Properties

Defining metadata for providers and directives using @Component decorator

```
@Component({
  selector: 'customers',
  providers: [DataService],
  templateUrl: 'app/customers/customers.component.html',
  directives: [RouterLink, SortByDirective]
})
export class CustomersComponent {
  constructor(private dataService: DataService) { }
}
```

<customers></customers>

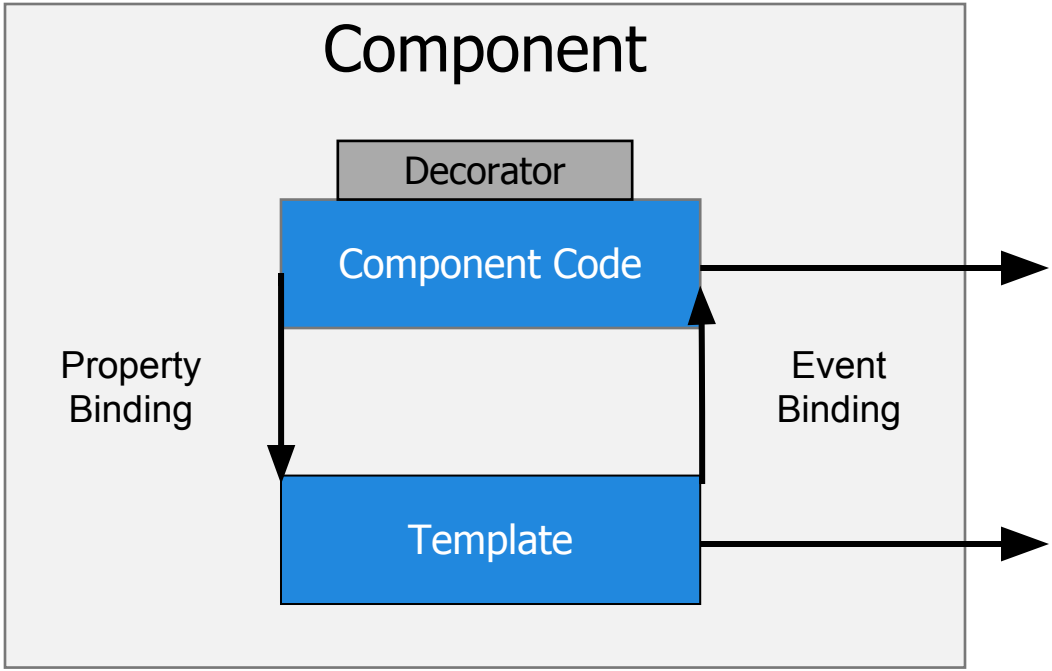
Injects DataService

Uses these directives

Steps to Build Components

1. Import/export required modules
2. Define component class
3. Add @Component decorator to class
4. Create a template

Component Code and Templates




```
@Component({
   templateUrl: 'mytemplate.html'
})
export class MyComponent {
   isHidden: true;
   clickMe() { … }
}
```

```
<div [hidden]="isHidden">
    <span (click)="clickMe()">
      Section 3
    </span>
</div>
```

Creating a Template

- Templates are HTML files
- Components are linked to templates using one of the following properties:
  - template
  - templateUrl

customer.component.ts

```
@Component({
   templateUrl: 'customer.component.html'
})
export class MyComponent {
   isHidden: true;
   clickMe() { … }
}
```

customer.component.html

```
<div [hidden]="isHidden">
    <span (click)="clickMe()">
      Section 3
    </span>
</div>
```

Bootstrapping a Component

index.html

```
<html>
<body>

  <app-component></app-component>

  <script>
    System.import('app');
  </script>

</body>
</html>
```

Bootstrap

```
@Component()
export class AppComponent
{
  …
}
```

Bootstrapping "App" Component

Applications must bootstrap a root component that is used to load other components

Import the bootstrap function and the root app component

main.ts

```
import { bootstrap } from '@angular/platform-browser-dynamic';
import { AppComponent } from './app.component';

bootstrap(AppComponent)
  .then(
      success => console.log('AppComponent bootstrapped!'),
      error => console.log(error)
);
```

Building a Component

http://plnkr.co/edit/velqr3mk6a5titNQBQz0

Binding and Directives

Template Syntax

- Components bind data to templates and handle events that pass data back to the component
- Template use "expressions"

Syntax Example	Description
{{ propertyName }}	Bind to and display property value
{{ 2 + 2 }}	Template expression
[target]="expression"	Property binding
(target)="statement"	Event binding
[(target)]="expression"	Two-way binding

One-Way Interpolation

Evaluate an expression that is between the {{ }} brackets

{{ expression }}

```
<img src="{{ customer.imagePath }}" />

{{ customer.firstName }} {{ customer.lastName }}

<br />

{{ customer.city }}, {{ customer.state.name }}
```

One-Way Property Bindings

One-way property bindings bind a DOM property to a value or expression

Use . syntax for nested properties and attr to bind to attributes

[target]="expression" or bind-target="expression"

```
<img [src]="customer.imagePath" />

<button [disabled]="!isEnabled">Save</button>

<div [hidden]="!isVisible" [class.active]="isActive">...</div>

<div [style.color]="textColor" [attr.aria-label]="text">..</div>

<div class="btn" [ngClass]="{active:isActive, disabled: isDisabled}">...</div>
```

Event Bindings

Event bindings are used to execute an expression when an event occurs

(target)="expression" or on-target="expression"

```
<button (click)="save()">Save</button>

<th sort-by="firstName" (sorted)="sort($event)">...</th>
```

Two-Way Binding

The ngModel directive can be used to create a "two-way" binding between a property and a control

Can also use bindon-ngModel="..." syntax

"Banana in a Box" syntax

```
<input type="text" [(ngModel)]="customer.firstName" />
```

Structural Directives

Angular 2 has built-in "structural" directives such as ***ngFor** and ***ngIf**
Manipulate the DOM structure

http://victorsavkin.com/post/119943127151/angular-2-template-syntax

Angular 2 directive that generates a template

```
<tr *ngFor="let customer of filteredCustomers">
    <td>{{ customer.firstName }}</td>
    <td>{{ customer.lastName }}</td>
</tr>

<div *ngIf="customer">{{ customer.details }}</div>
```

Add to DOM if customer property has a value

Built-in Pipes

Angular 2 apps can use Pipes to filter and format data

Several built-in pipes

uppercase, lowercase, slice, date, currency, json

```
{{ customer.orderTotal | currency:'USD':true }}
```

Format as currency

Custom Pipes

Custom pipes can be created using the Pipe annotation

Pipe classes have a transform method

```
import { Pipe } from '@angular/core';

@Pipe({ name: 'capitalize' })
export class CapitalizePipe {
    transform(value: any) {
        if (value: any) {
            return value.charAt(0).toUpperCase() + value.slice(1);
        }
        return value;
    }
}
```

Data Binding Hands-On Exercise

http://plnkr.co/edit/KA6DzGJRVd7izNfhe1y9

Services and DI

Services

A Service provides anything our application needs.
It often shares data or functions between other Angular features

vehicle.service.ts

```
@Injectable()
export class VehicleService {
  getVehicles() {
    return [
      new Vehicle(10, 'Millenium Falcon'),
      new Vehicle(12, 'X-Wing Fighter'),
      new Vehicle(14, 'TIE Fighter')
    ];
  }
}
```

A Service is just a class

Service

Provides something of value

Shared data or logic

e.g. Data, logger, exception handler, or message service

Dependency Injection

Dependency Injection is how we provide an instance of a class to another Angular feature

Dependency Injection

vehicle.component.ts

```
export class VehicleListComponent {
  vehicles: Vehicle[];

  constructor(private _vehicleService: VehicleService) {
    this._vehicleService.getVehicles()
      .subscribe(vehicles => this.vehicles = vehicles);
  }
}
```

Injecting VehicleService

Injecting a Service into a Component

Locates the service in the Angular injector

Injects into the constructor

Injecting a Service into a Service

Same concept as injecting into a Component

@Injectable() is similar to Angular 1's $inject

Provides metadata about injectables

```
@Injectable()
export class SpeakerService {
  constructor(private http: Http) { }

  getSpeakers() {
    return this.http
      .get(speakersUrl)
      .map(res => <Speaker[]>res.json().data);
  }
}
```

Injecting Http

Providers

Register these Services with Angular's injector

Injection

Inject a Service into another object

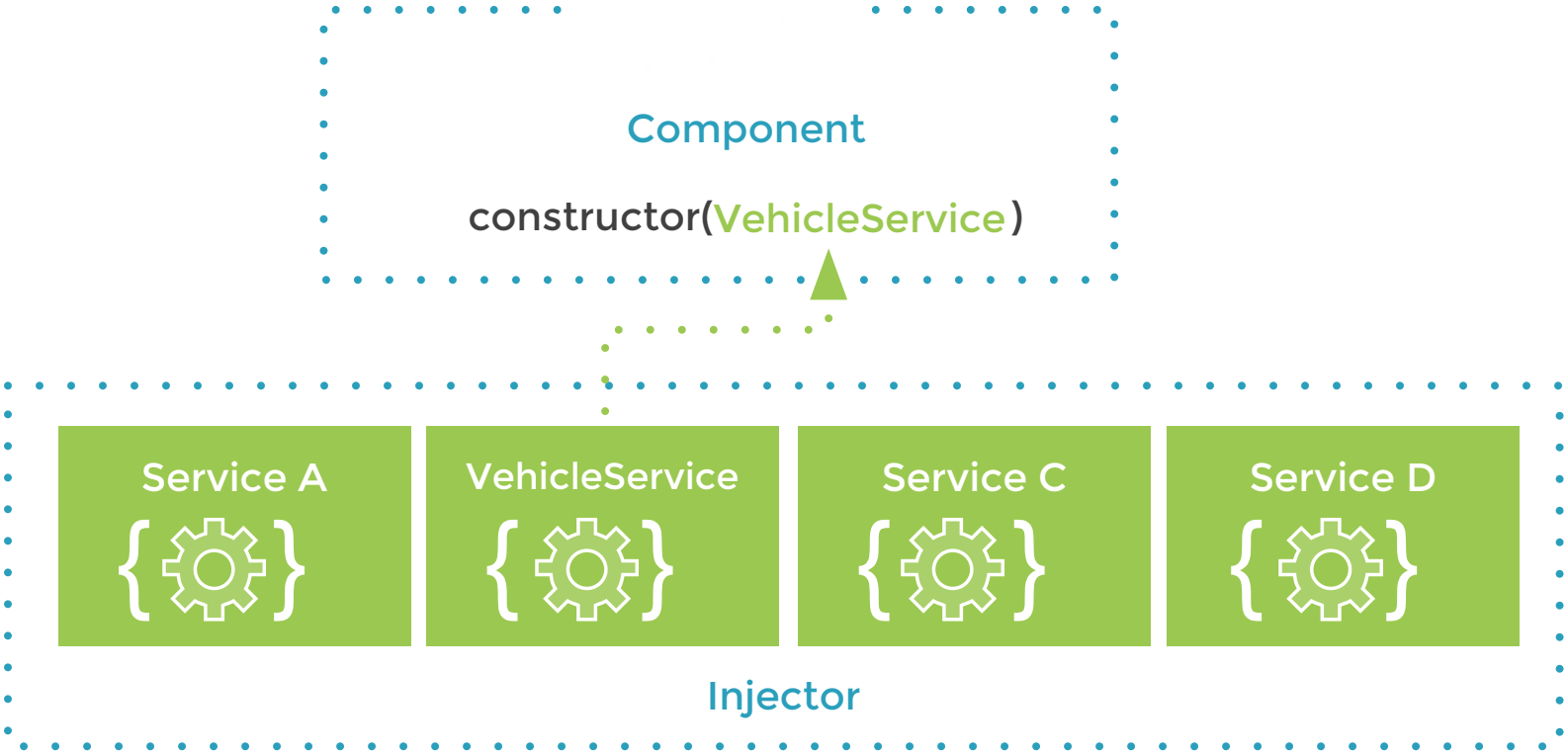
```
@Component({
  selector: 'story-characters',
  templateUrl: './app/characters.component.html',
  styleUrls: ['./app/characters.component.css'],
  directives: [CharacterDetailComponent],
  providers: [HTTP_PROVIDERS, CharacterService]
})
export class CharactersComponent implements OnInit {
  @Output() changed = new EventEmitter<Character>();
  @Input() storyId: number;
  characters: Observable<Character[]>;
  selectedCharacter: Character;

  constructor(private _characterService: CharacterService) { }

  ngOnInit() {
    this.characters = this._characterService
      .getCharacters(this.storyId);
  }

  select(selectedCharacter: Character) {
    this.selectedCharacter = selectedCharacter;
    this.changed.emit(selectedCharacter);
  }
}
```

Register the service with the injector at the parent that contains all components that require the service


Component
constructor(VehicleService)
Service A
VehicleService
Service C
Service D
Injector

Http

Http

We use Http to get and save data with Promises or Observables. We isolate the http calls in a shared Service.

Http Step by Step

Import from @angular/http

Register the Http providers

Call Http.get in a Service and return the mapped the result

Subscribe to the Service's function in the Component

Http Requirements

HTTP_PROVIDERS is an array of service providers for Http

Located in module @angular/http

```
import { Component } from '@angular/core';
import { HTTP_PROVIDERS } from '@angular/http';

@Component({
  moduleId: module.id,
  selector: 'the-app',
  templateUrl: 'my.component.html',
  providers: [HTTP_PROVIDERS]
})
export class AppComponent { }
```

Declaring the providers

```
@Injectable()
export class SpeakerService {
  constructor(private http: Http) { }


  getSpeakers() {
    return this.http.get(speakersUrl)
      .map(res => <Speaker[]>res.json().data)
      .catch(this.handleError)
  }

  private handleError(err: Response) {
    console.error(err);
    Observable.throw( … );
  }
}
```

Make and return the async GET call

Map the response

Handle errors

Subscribing to the Observable

Component is handed an Observable

We Subscribe to it

```
getSpeakers() {
  this.speakers = [];

  this.speakerService.getSpeakers()
    .subscribe(
      speakers => this.speakers = speakers,
      error => this.errorMessage = <any>error
    );
}
```

Subscribe to the Observable

Handle error conditions

http://reactivex.io/

RxJs

RxJs (Reactive Js) implements the asynchronous observable pattern and is widely used in Angular 2

vehicle.service.ts

```
return this._http.get('api/vehicles')
  .map((response: Response) =>
    <Vehicle[]>response.json().data
  )
  .catch(this.handleError);
```

json() is defined by the http spec

data is what we defined on the server

Returning from Http

We do not return the response

Service does the dirty work

The consumers simply get the data

Catching Errors

```
getVehicles() {
  return this._http.get('api/vehicles')
    .map((response: Response) => <Vehicle[]>response.json().data)
    .catch(this.handleError);
}

private handleError(error: Response) {
  console.error(error);
  return Observable.throw(error.json().error || 'Server error');
}
```

Catch

Exception Handling

We catch errors in the Service

We sometimes pass error messages to the consumer for presentation

vehicle-list.component.ts

```
getHeroes() {
  this._vehicleService.getVehicles()
    .subscribe(
      vehicles => this.vehicles = vehicles,
      error =>  this.errorMessage = <any>error
  );
}
```

Subscribe to the observable

Success and failure cases

Subscribing to the Observable

Component is handed an Observable

We Subscribe to it

Async Pipe

The Async Pipe receives a Promise or Observable as input and subscribes to the input, eventually emitting the value(s) as changes arrive.

vehicle-list.component.ts

```
vehicles: Observable<Vehicle[]>;

getHeroes() {
  this.vehicles = this._vehicleService.getVehicles();
}
```

Property becomes Observable

Set the observable from the Service

Observable Properties

Component is simplified

Grab the Observable and set it to the property

Async Pipe in the Template

Apply the async Pipe

Subscribes to vehicles

```
<ul>
  <li *ngFor="let vehicle of vehicles | async">
    {{ vehicle.name }}
  </li>
</ul>
```

Services and Http Hands-On Exercise

http://plnkr.co/edit/TC1AGl44XUv48ORqYRZa

Routing

Angular Routing

- Components can be changed/swapped by using routing
- Use the **@angular/router** module

Steps to Use Routing

1. Add the <base href="/"> element
2. Register ROUTER_PROVIDERS
3. Add the @Routes decorator
4. Add a <router-outlet>
5. Use routerLink

Define the <base> Element

Router supports history.pushState which allows paths like http://yourdomain.com/customers to be used

The <base> element needs to be set for it to work properly

index.html

```
<html>
<head>
  <base href="/">
</head>
<body>

  ...

</body>
</html>
```

Steps to Use Routing

1. Add the <base> element
2. Register ROUTER_PROVIDERS
3. Add the @Routes decorator
4. Add a <router-outlet>
5. Use routerLink

Register ROUTER_PROVIDERS

Register ROUTER_PROVIDERS in order to use the router functionality

app.providers.ts

```
import { ROUTER_PROVIDERS } from '@angular/router';
…

export const APP_PROVIDERS = [
  ROUTER_PROVIDERS,
  ...
];
```

app.component.ts

```
import { APP_PROVIDERS }
 from './app.providers';

@Component({
  selector: 'app-container',
  template: `...`,
  providers: [APP_PROVIDERS]
  …
})
export class AppComponent {
  constructor() { }
}
```

Steps to Use Routing

1. Add the <base> element
2. Register ROUTER_PROVIDERS
3. Add the @Routes decorator
4. Add a <router-outlet>
5. Use routerLink

Add the @Routes Decorator

- Define route configuration on a component with @Routes
- Each component can have its own routes (more on this later!)

app.component.ts

```
@Component({
  selector: 'app-container',
  template: `...`,
  ...
})
@Routes([
  { path: '/',           component: CustomersComponent},
  { path: '/orders/:id', component: OrdersComponent }
])
export class AppComponent {
  constructor() { }
}
```

Steps to Use Routing

1. Add the <base> element
2. Register ROUTER_PROVIDERS
3. Add the @Routes decorator
4. Add a <router-outlet>
5. Use routerLink

The RouterOutlet Directive

- Angular places components into a "component container" called RouterOutlet
- Use **<router-outlet>** to define the location where components are loaded

app.component.ts

```
import { ROUTER_DIRECTIVES } from '@angular/router';
//Other imports here for APP_PROVIDERS, CustomersComponent, OrdersComponent

@Component({
  selector: 'app-container',
  template: `<router-outlet></router-outlet>`,
  directives: [ROUTER_DIRECTIVES],
  providers: [APP_PROVIDERS]
})
@Routes([
  { path: '/',           component: CustomersComponent },
  { path: '/orders/:id', component: OrdersComponent    }
])
export class AppComponent { }
```

Define the container

Make the router directives available to the template

Steps to Use Routing

1. Add the <base> element
2. Register ROUTER_PROVIDERS
3. Add the @Routes decorator
4. Add a <router-outlet>
5. Use routerLink

The routerLink Directive

- The routerLink directive can be used to add links to routes defined using @Routes
- Defines the route alias and any route parameter data

customer.component.ts

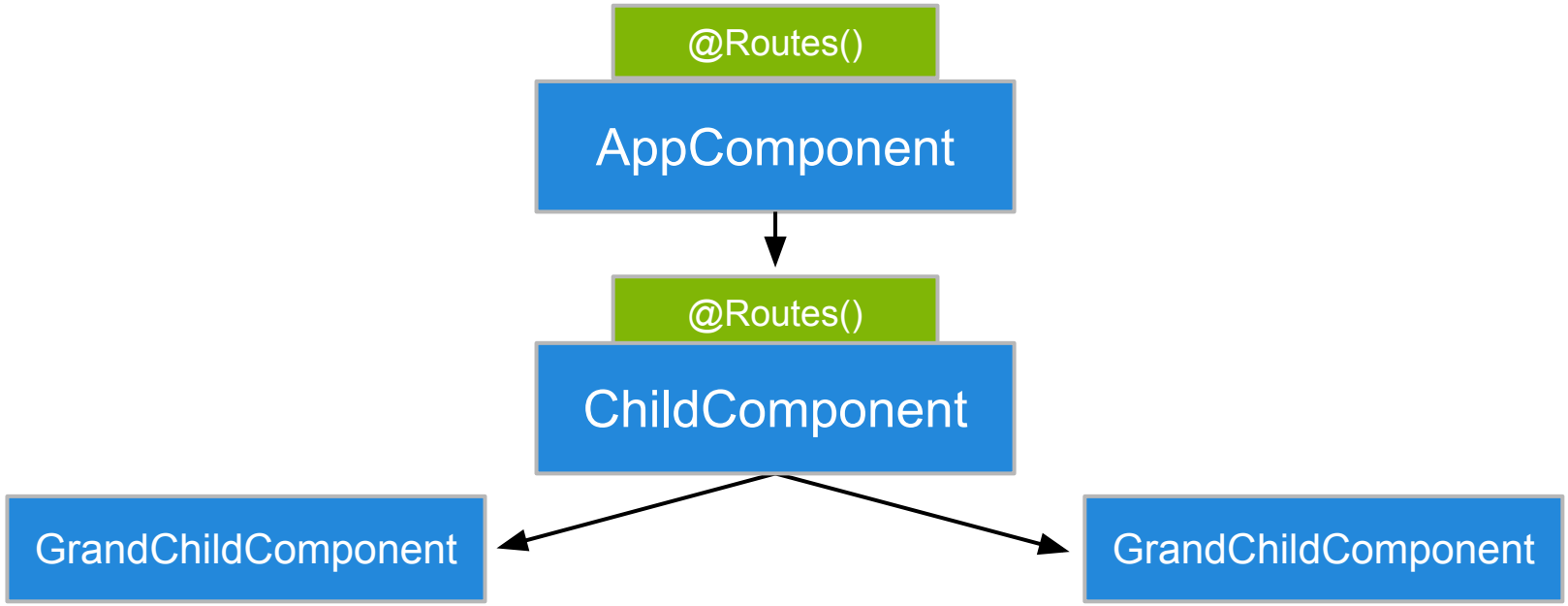
```
@Component({
  selector: 'customers',
  templateUrl: 'customers.component.html',
  directives: [ROUTER_DIRECTIVES]
})
export class CustomersComponent {
  …
}
```

customer.component.html

```
<a [routerLink]="['Orders', customer.id]">
  {{ customer.firstName | capitalize }}
</a>
```

Child Routes

Angular supports child routes on components



Child Routes

- Angular supports child routes
- Apply @Routes to a child component to enable

child.component.ts

```
@Component({
  selector: 'child',
  templateUrl: `<router-outlet></router-outlet>`,
  directives: [ROUTER_DIRECTIVES]
})
@Routes([
  { path:'/orders',  component: CustomerOrdersComponent },
  { path:'/details', component: CustomerDetailsComponent }
])
export class CustomersComponent {
  ...
}
```

angular-cli
Hands-On Exercise

https://github.com/angular/angular-cli

Thanks for Coming!

@DanWahlin
@John_Papa

http://tinyurl.com/ng2ABspring2016

VS Code Snippets for Angular 2

https://marketplace.visualstudio.com/items?itemName=danwahlin.angular2-snippets

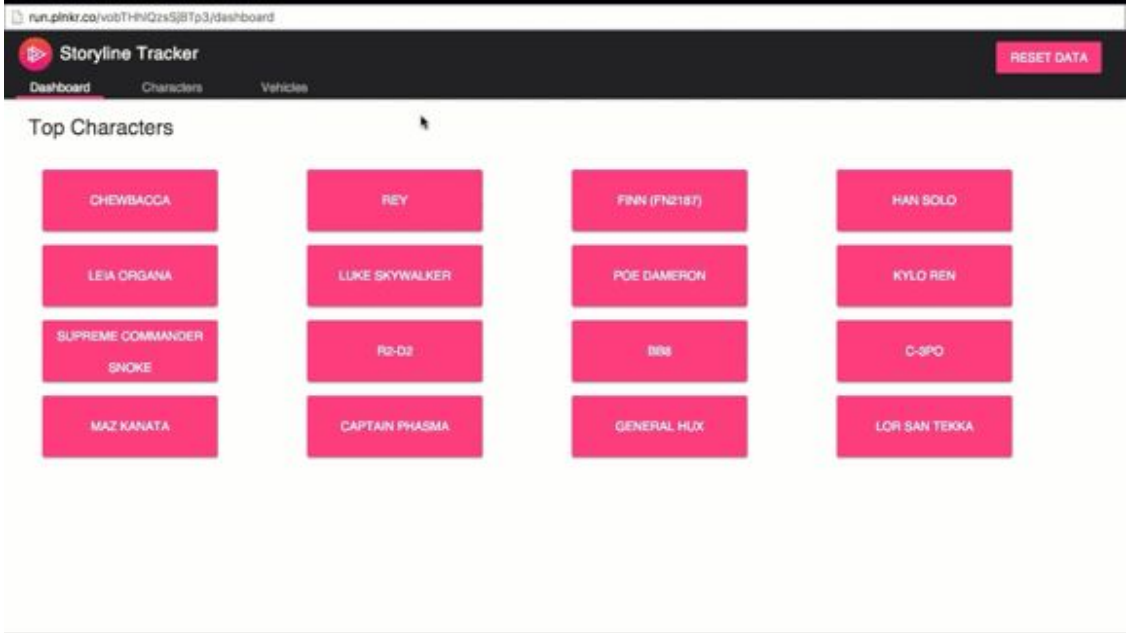
https://marketplace.visualstudio.com/items?itemName=johnpapa.Angular2

Angular 2: First Look

By John Papa

This course is a gentle introduction to the changes that Angular 2 brings, how they compare to Angular 1, and provides an understanding of the architecture and how the core concepts work together to build applications.

http://jpapa.me/a2ps1stlook



Onsite Training

Need Angular 2 onsite training for your team?

Contact us at training@codewithdan.com