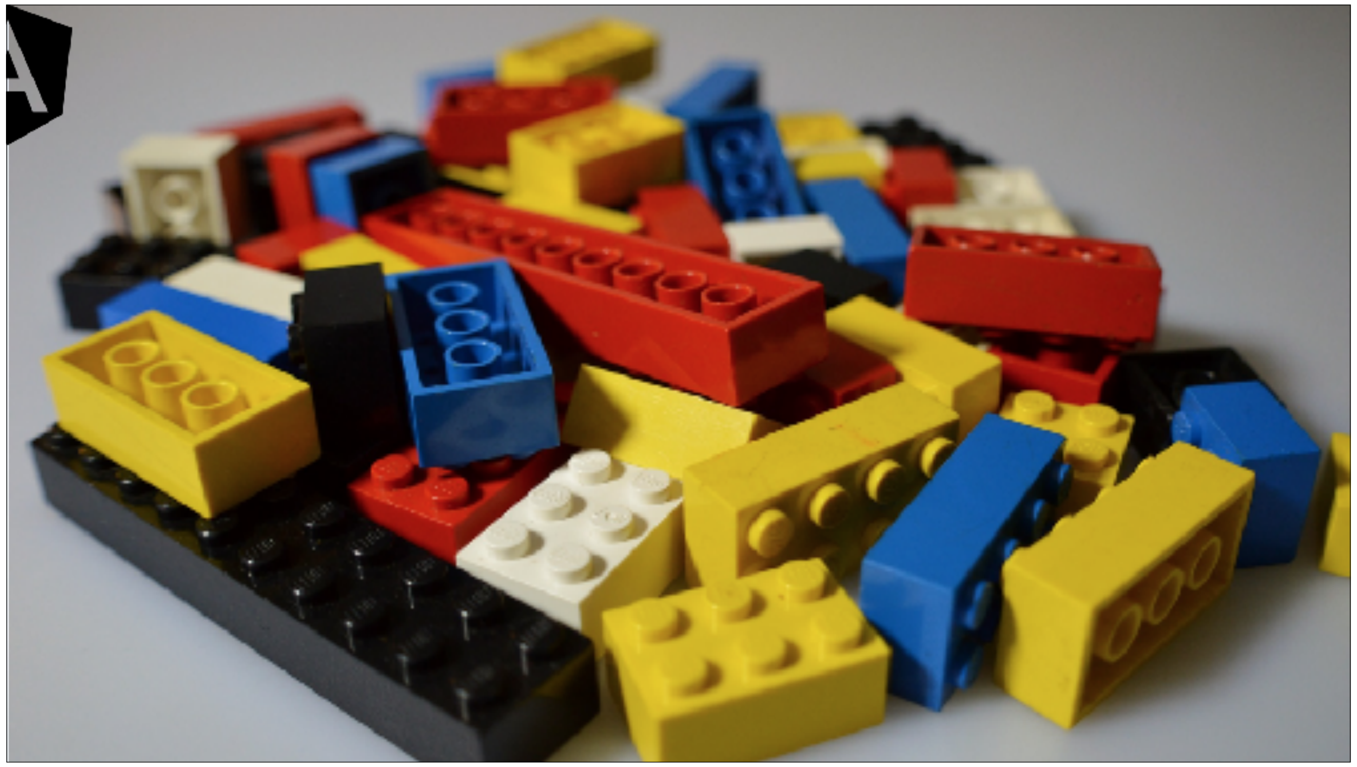
ANGULAR

THE COOL PARTS™

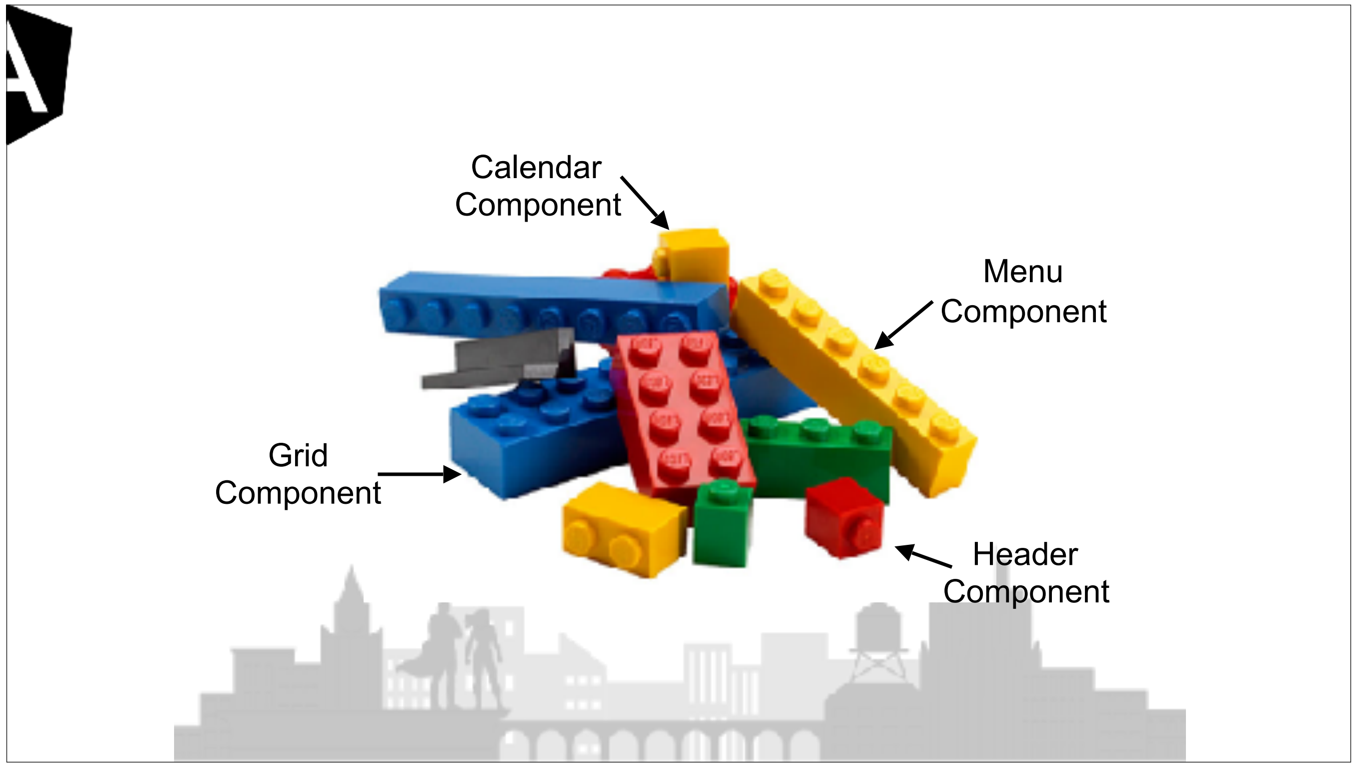
Hello Birmingham. I’m Alex Lakatos, a JavaScript Developer Advocate at Nexmo. I want to talk to you today about Angular.



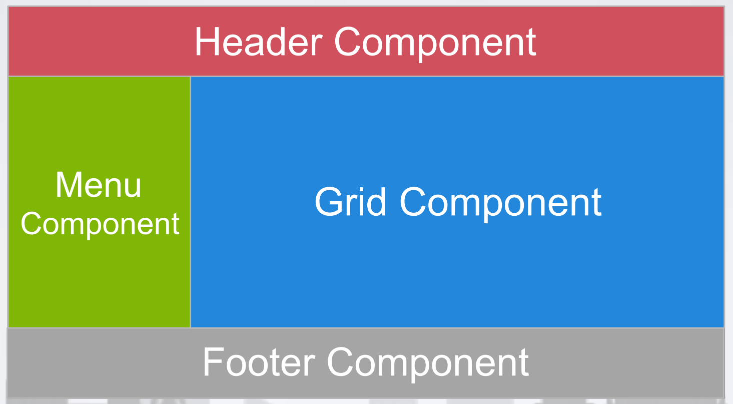
Now, these aren’t just any type of building block. These legos can be re-used in various ways and have the power to build applications that can target multiple devices and browsers.

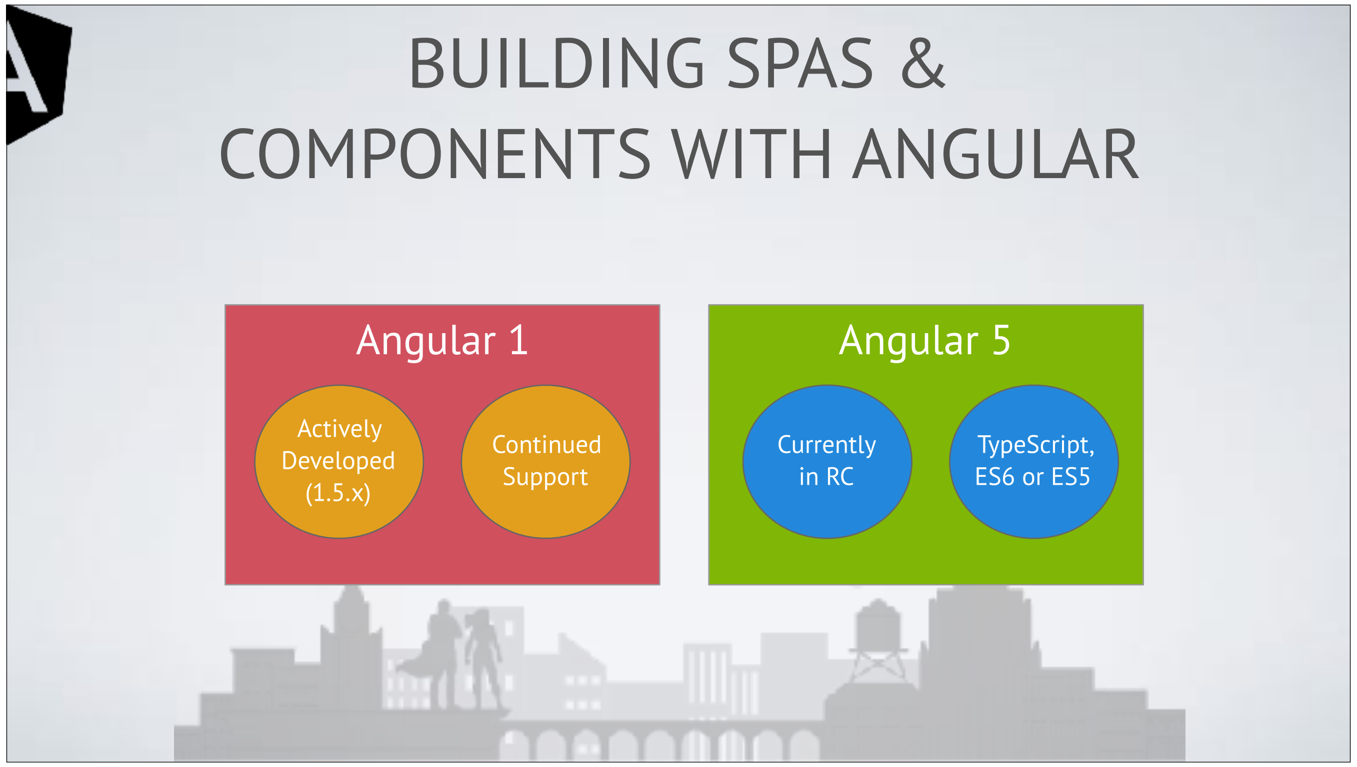
When the building blocks are combined you can build some cool things! Being recruited by the dark side? Build the next enterprise-scale app for the death star (since I’m sure they’ll build another one).

Being recruited by the rebel alliance to build an app? You can do that too of course.


Calendar Component
Menu Component
Grid Component
Header Component

BUILDING SPAS WITH COMPONENTS





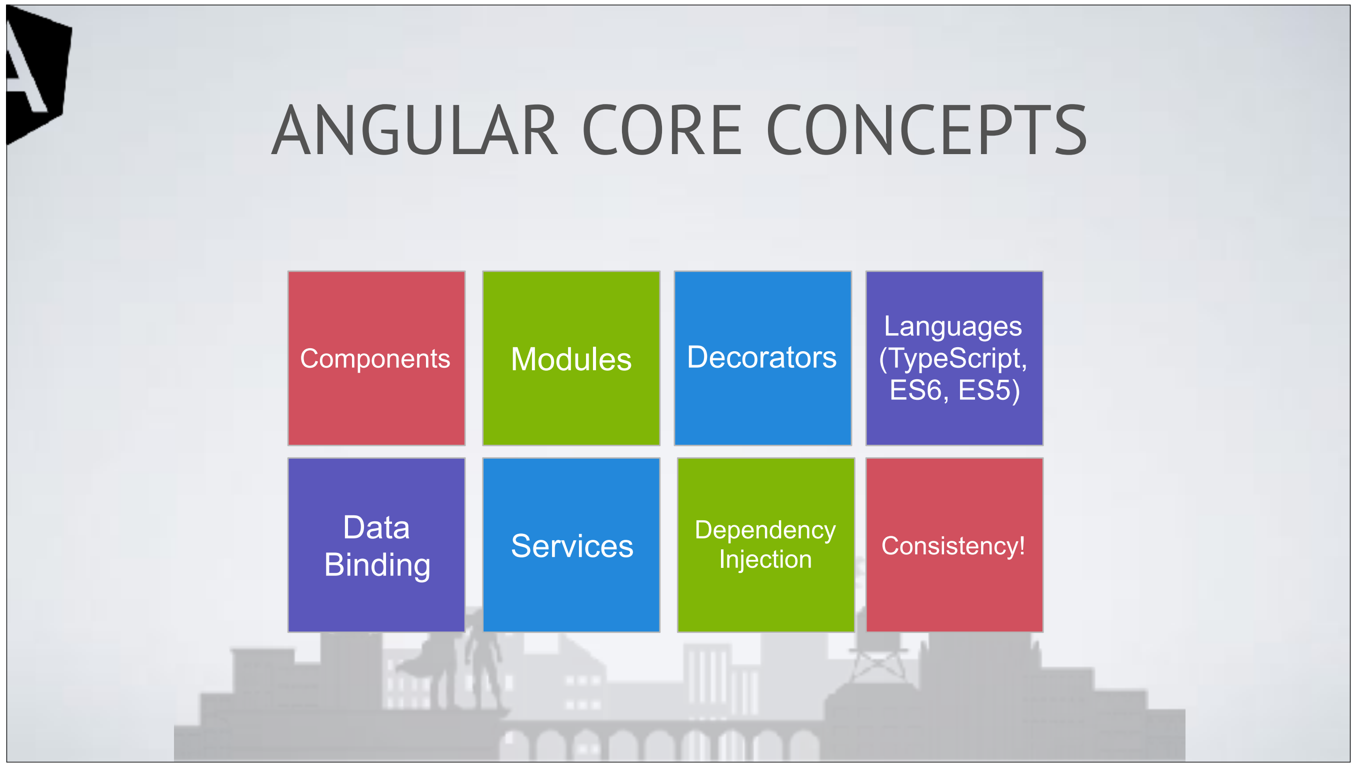
BUILDING SPAS & COMPONENTS WITH ANGULAR

Angular 1

- Actively Developed (1.5.x)
- Continued Support

Angular 5

- Currently in RC
- TypeScript, ES6 or ES5



ANGULAR CORE CONCEPTS

Components

Modules

Decorators

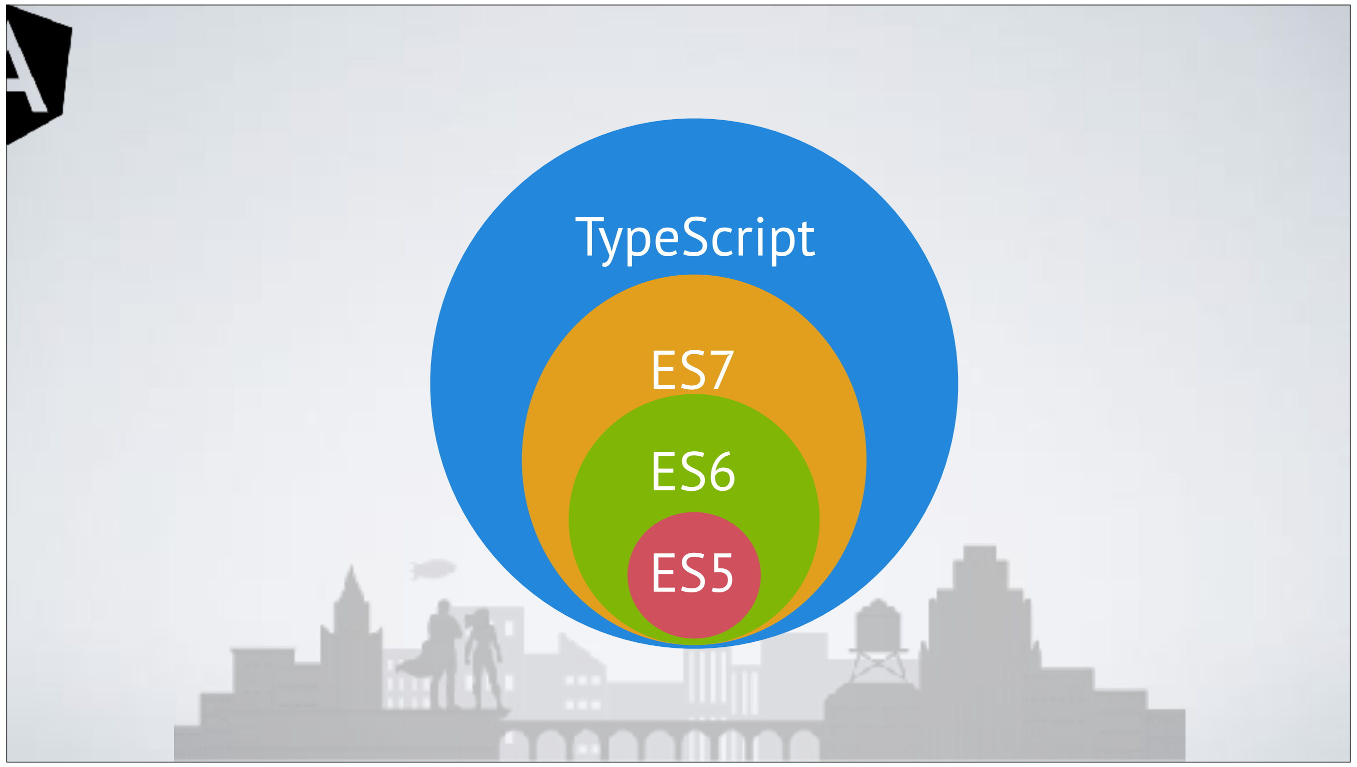
Languages (TypeScript, ES6, ES5)

Data Binding

Services

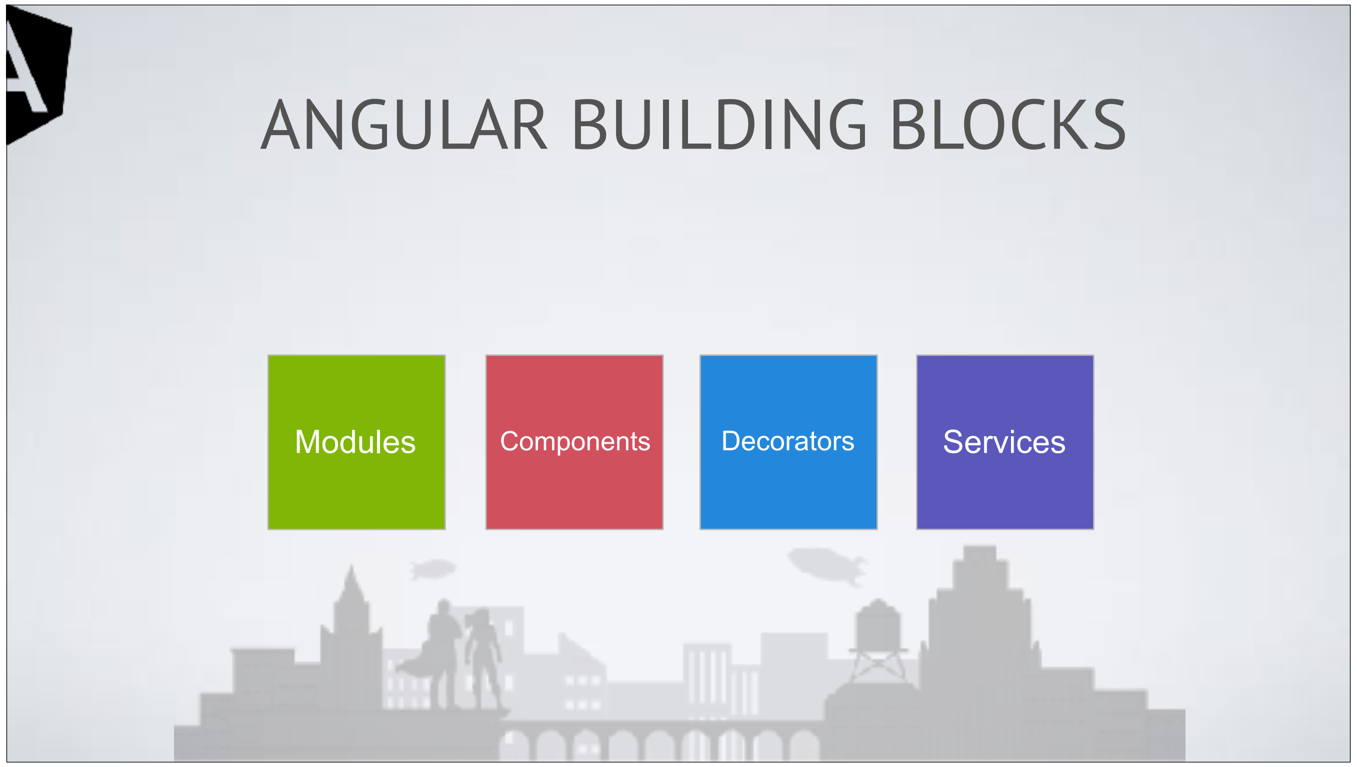
Dependency Injection

Consistency!


TypeScript
ES7
ES6
ES5

GETTING STARTED

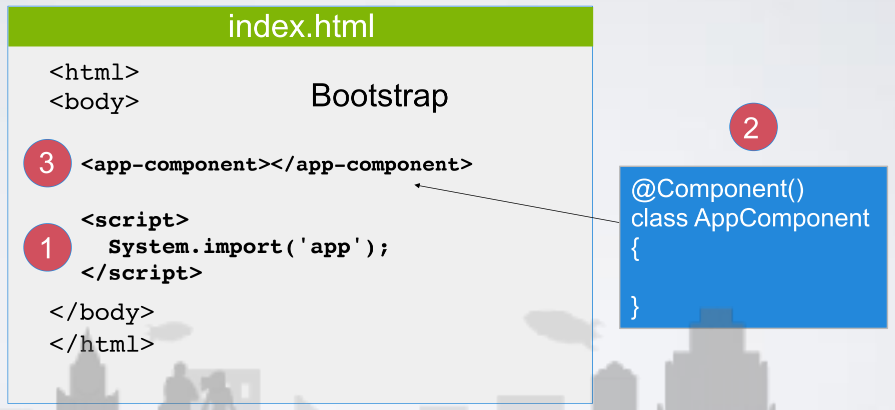
Angular is all about building blocks. So if you're someone who likes to build things this is a fun story to hear about!



ANGULAR BUILDING BLOCKS

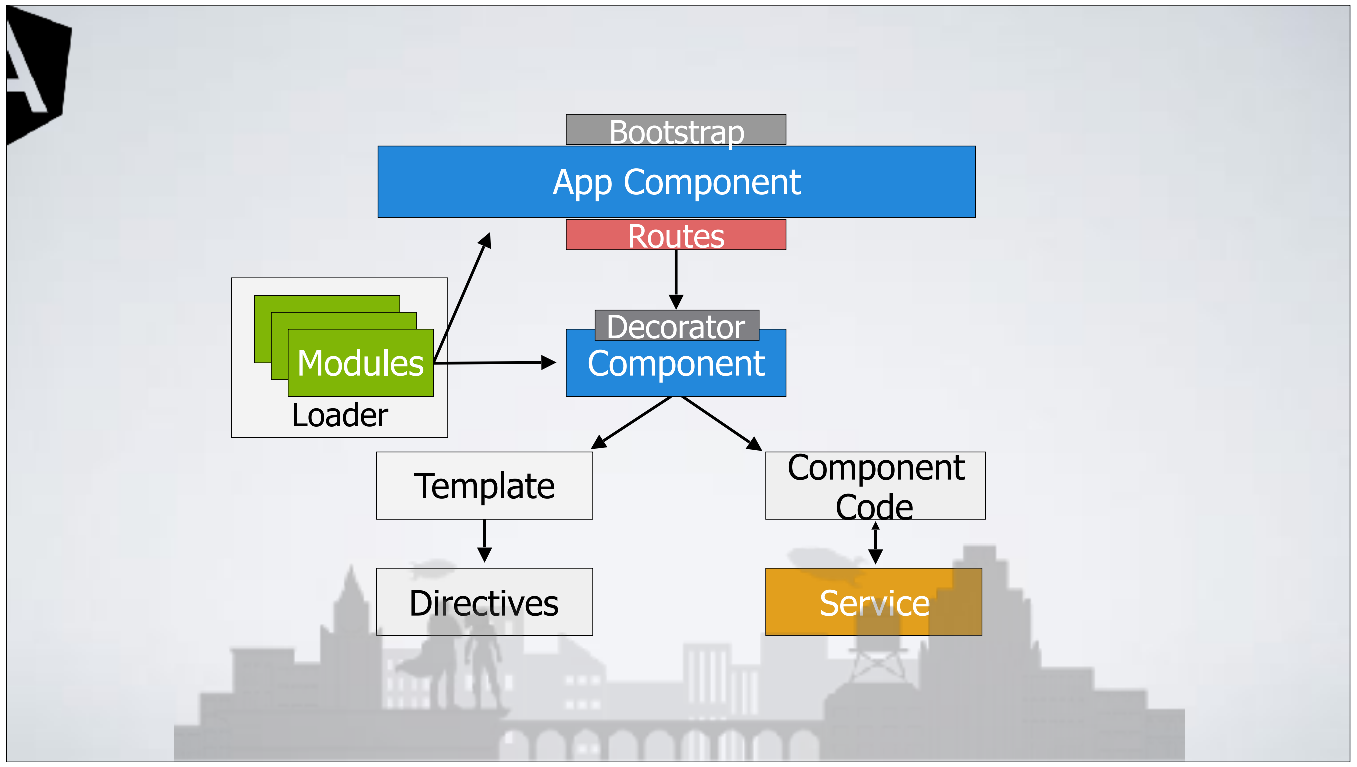
- Modules
- Components
- Decorators
- Services

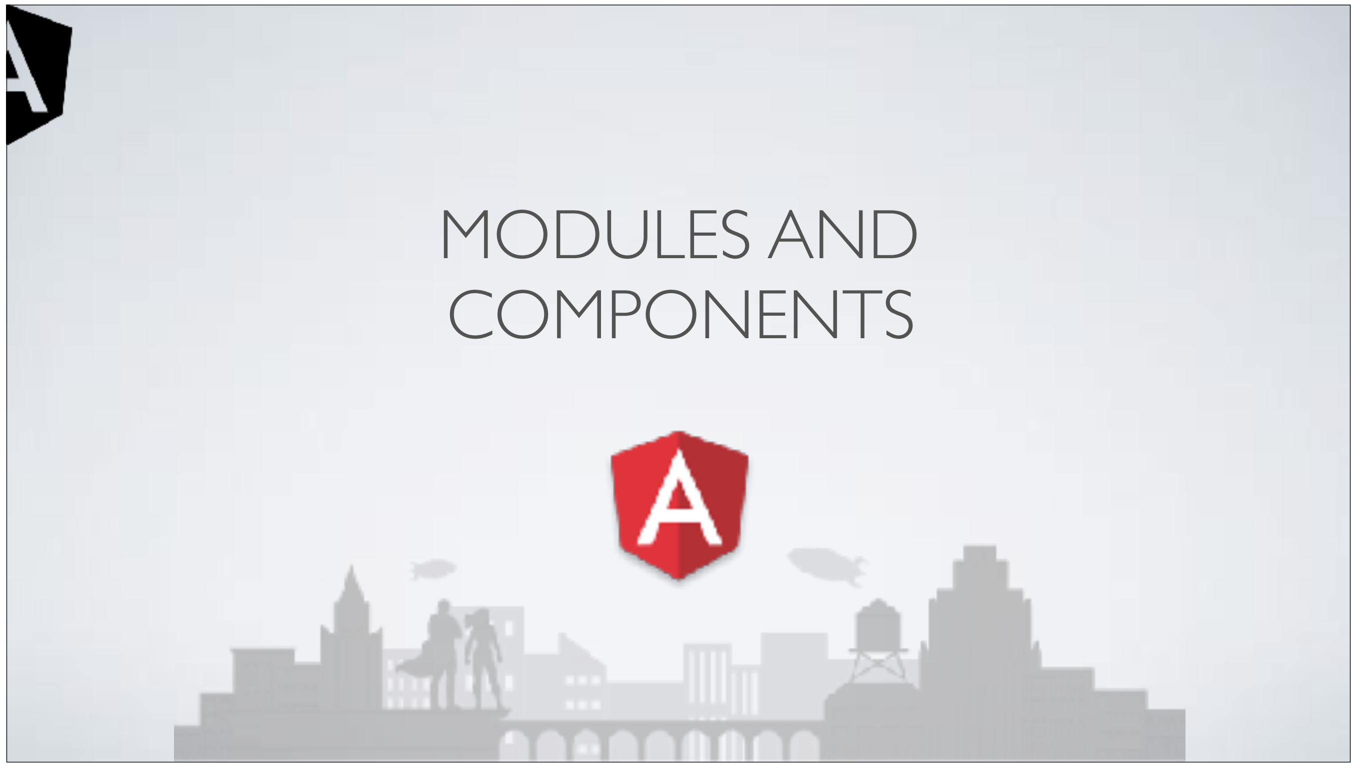
STEPS TO GET STARTED



```
<html>
<body>

<app-component></app-component>

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

</body>
</html>
```

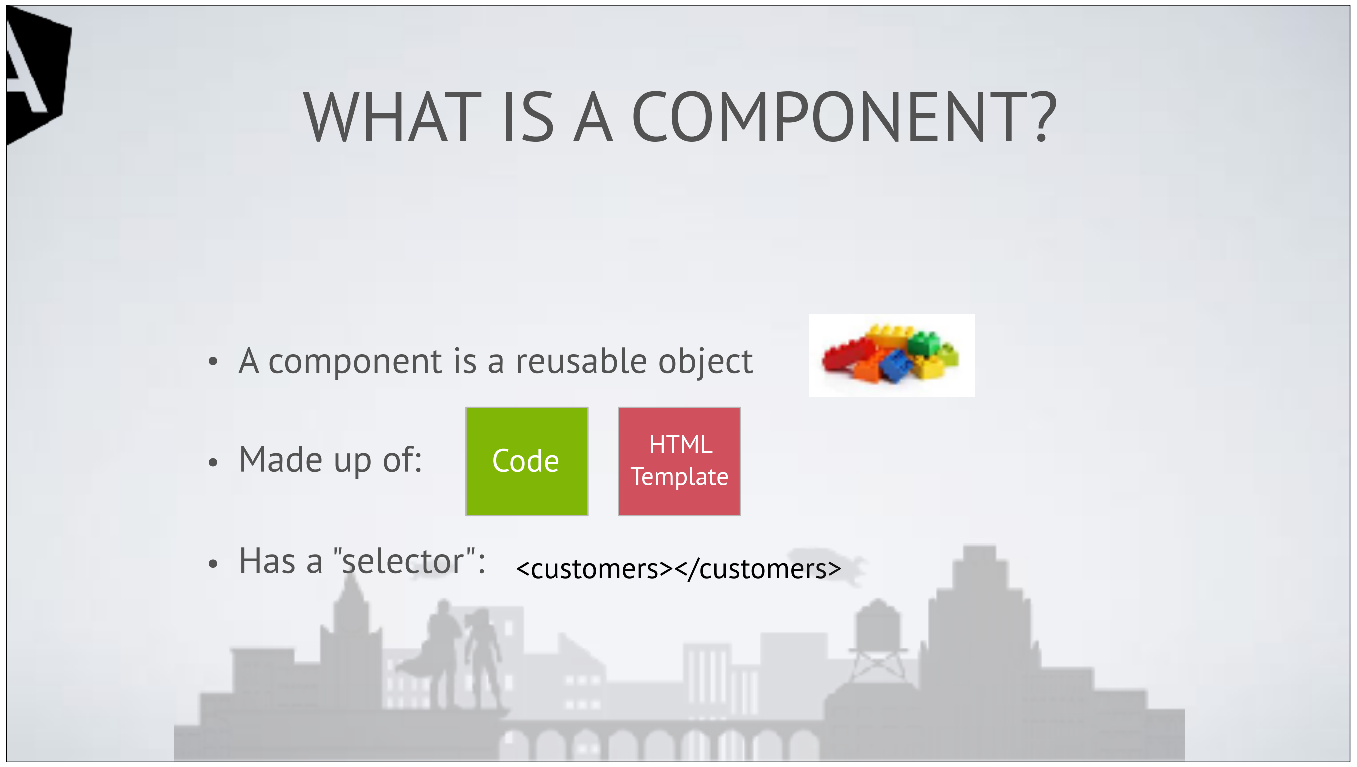
```
@Component()
class AppComponent
{

}
```


Bootstrap
App Component
Routes
Modules
Loader
Decorator
Component
Template
Component Code
Directives
Service



MODULES AND COMPONENTS

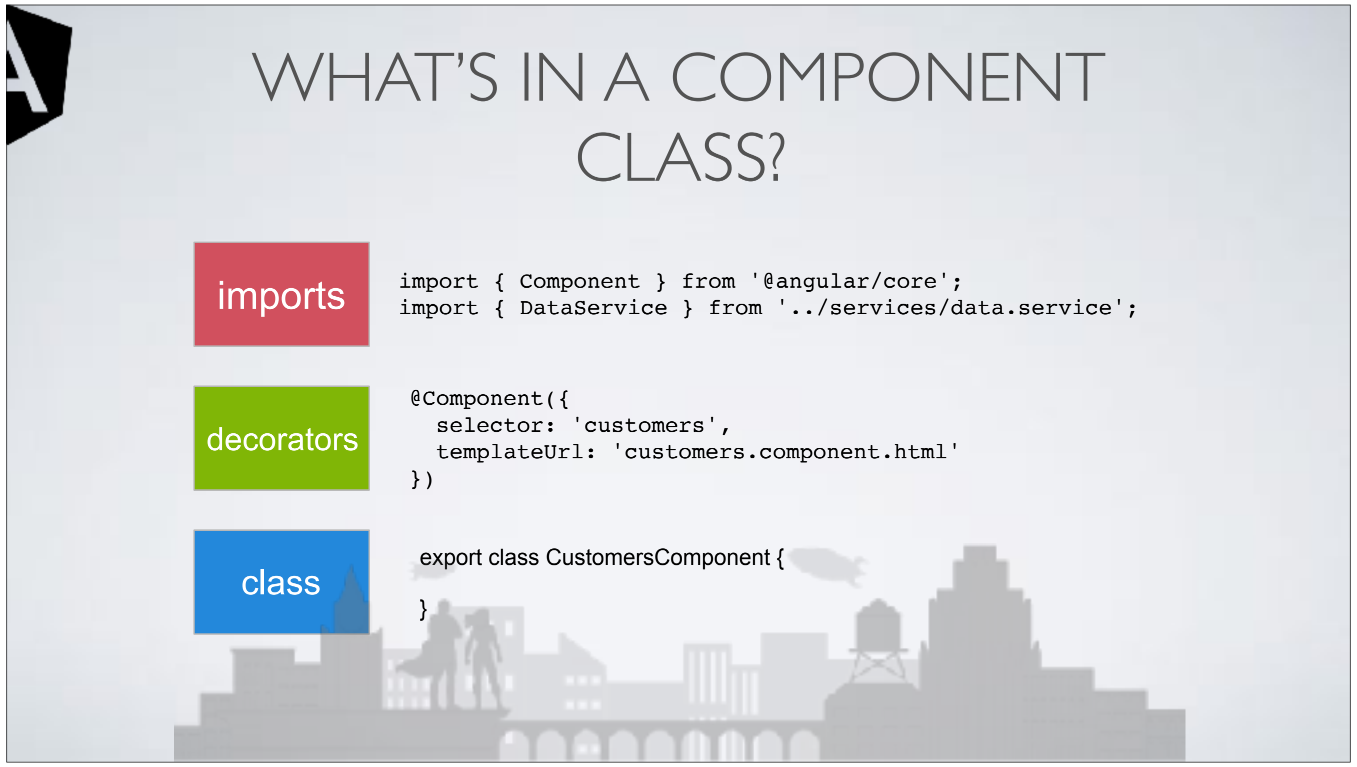
Angular is all about building blocks. So if you're someone who likes to build things this is a fun story to hear about!



WHAT IS A COMPONENT?

- A component is a reusable object

- Made up of: Code | HTML Template
- Has a "selector": `<customers></customers>`



WHAT'S IN A COMPONENT CLASS?

imports

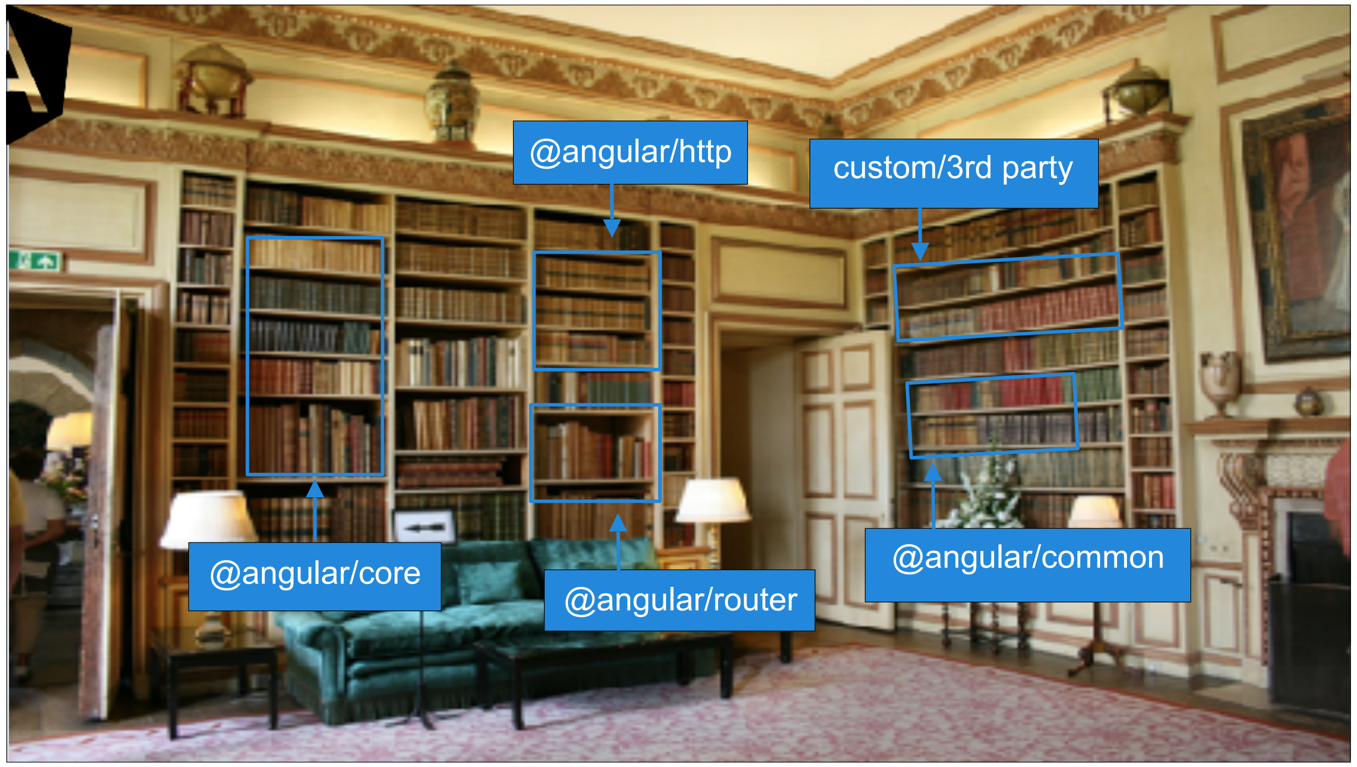
```
import { Component } from '@angular/core';
import { DataService } from '../services/data.service';
```

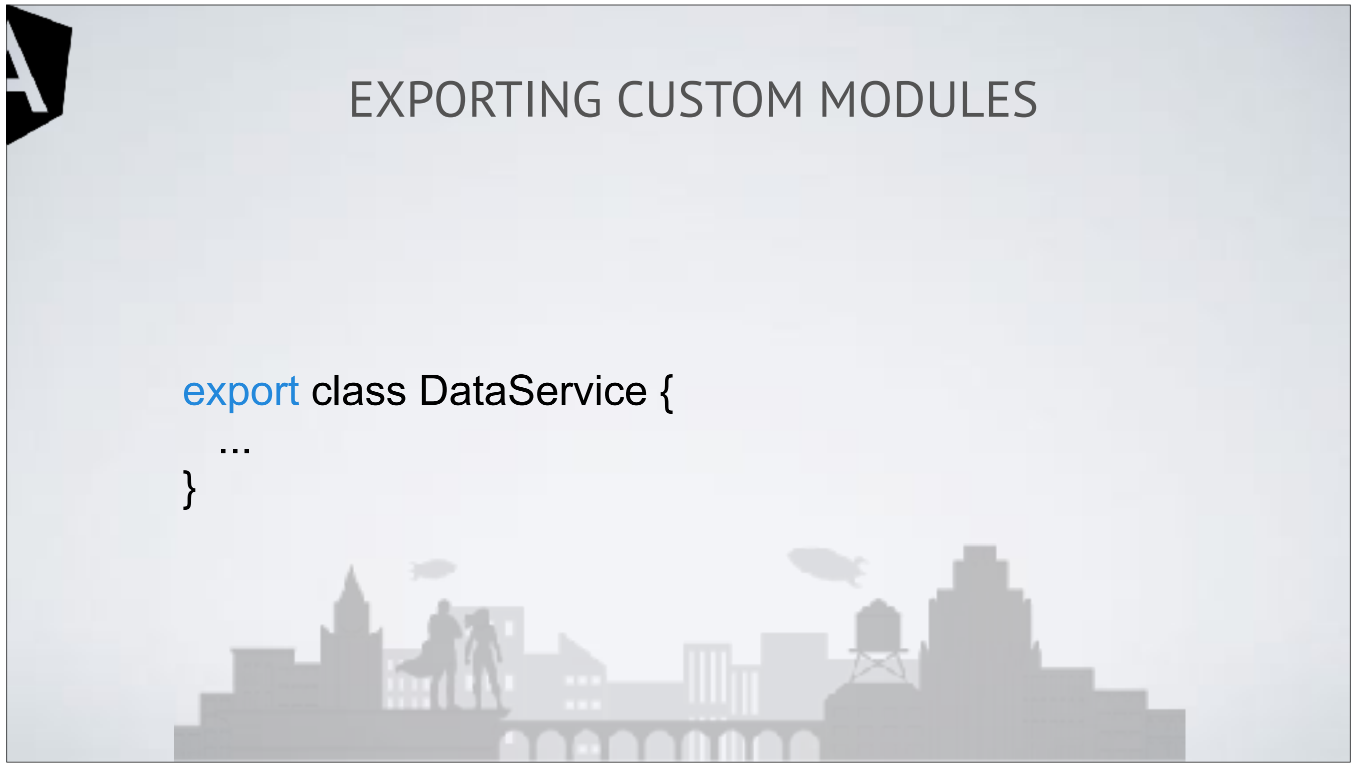
decorators

```
@Component({
  selector: 'customers',
  templateUrl: 'customers.component.html'
})
```

class

```
export class CustomersComponent {

}
```


@angular/http
custom/3rd party
@angular/core
@angular/router
@angular/common



EXPORTING CUSTOM MODULES

```
export class DataService {
  ...
}
```

IMPORTING MODULES

customers.component.ts

Imported module is relative to current file.

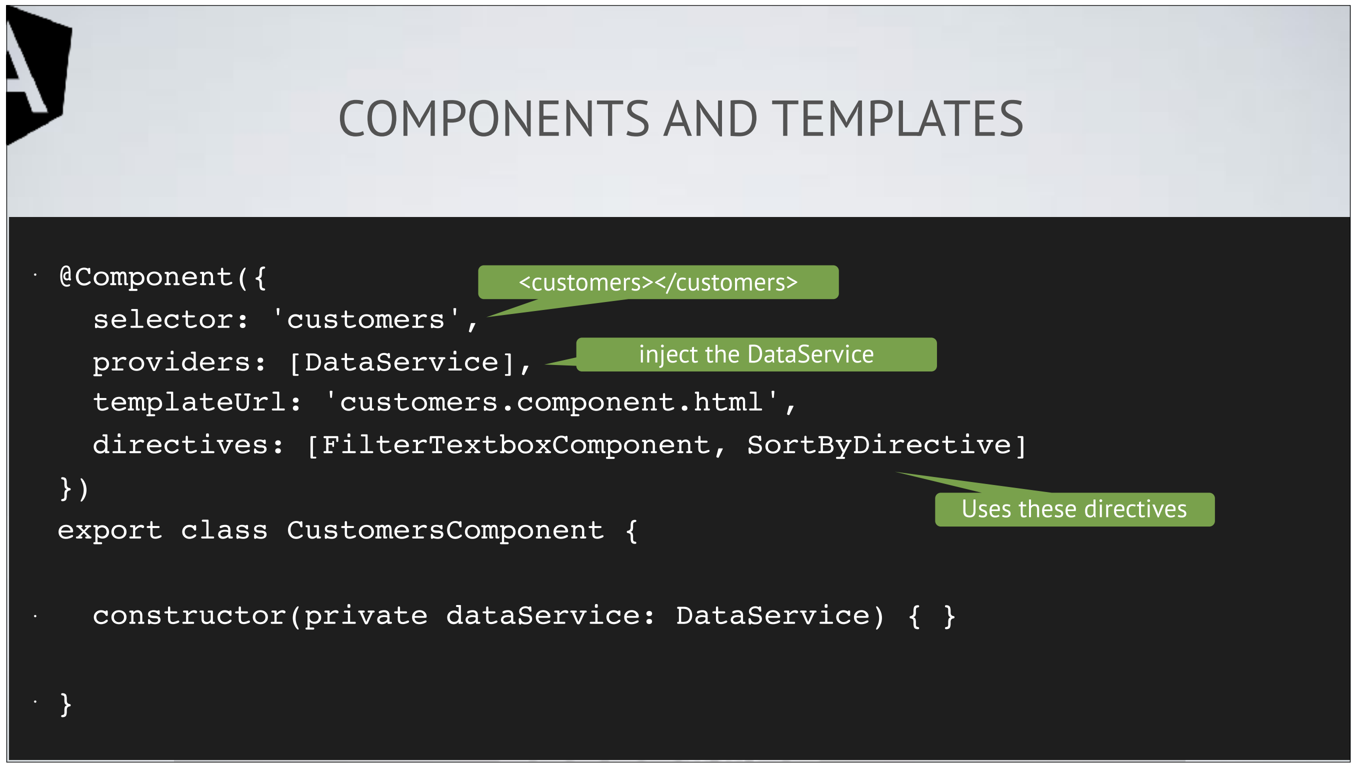
```
import { Component  } from '@angular/core';
import { DataService } from '../shared/services/data.service';
...
```

SYSTEM.JS: ENABLING MODULE LOADING

```
<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="systemjs.config.js"></script>

<script>
   System.import('app').catch(function(err){
      console.error(err);
   });

</script>
```

Import this module as a starting point

DECLARATIVE TEMPLATE SYNTAX

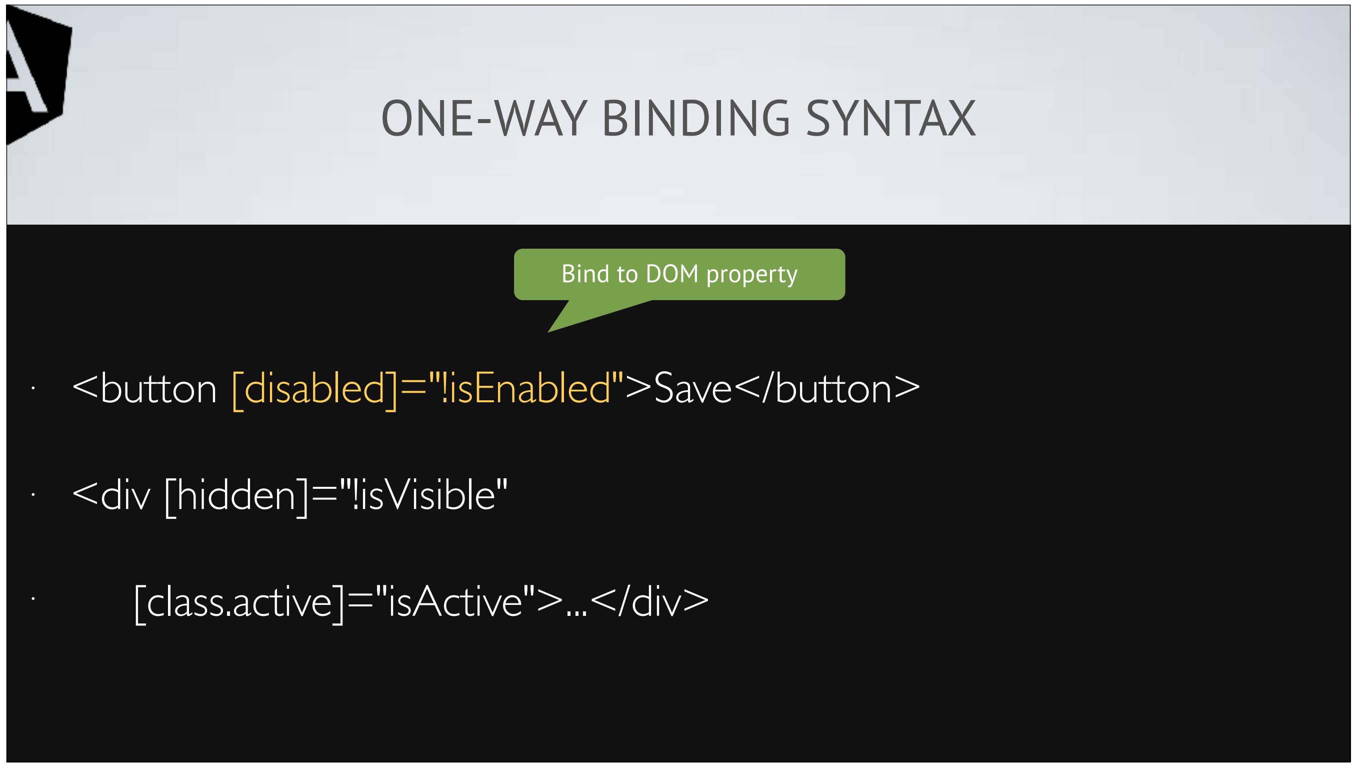


COMPONENTS AND TEMPLATES

```
@Component({
  selector: 'customers',
  providers: [DataService],
  templateUrl: 'customers.component.html',
  directives: [FilterTextboxComponent, SortByDirective]
})
export class CustomersComponent {

  constructor(private dataService: DataService) { }

}
```

<customers></customers>
inject the DataService
Uses these directives

Angular 2 apps are built using components
Provides reuse and consistency
Components rely on decorators to define metadata
Can use directives for rendering in component templates

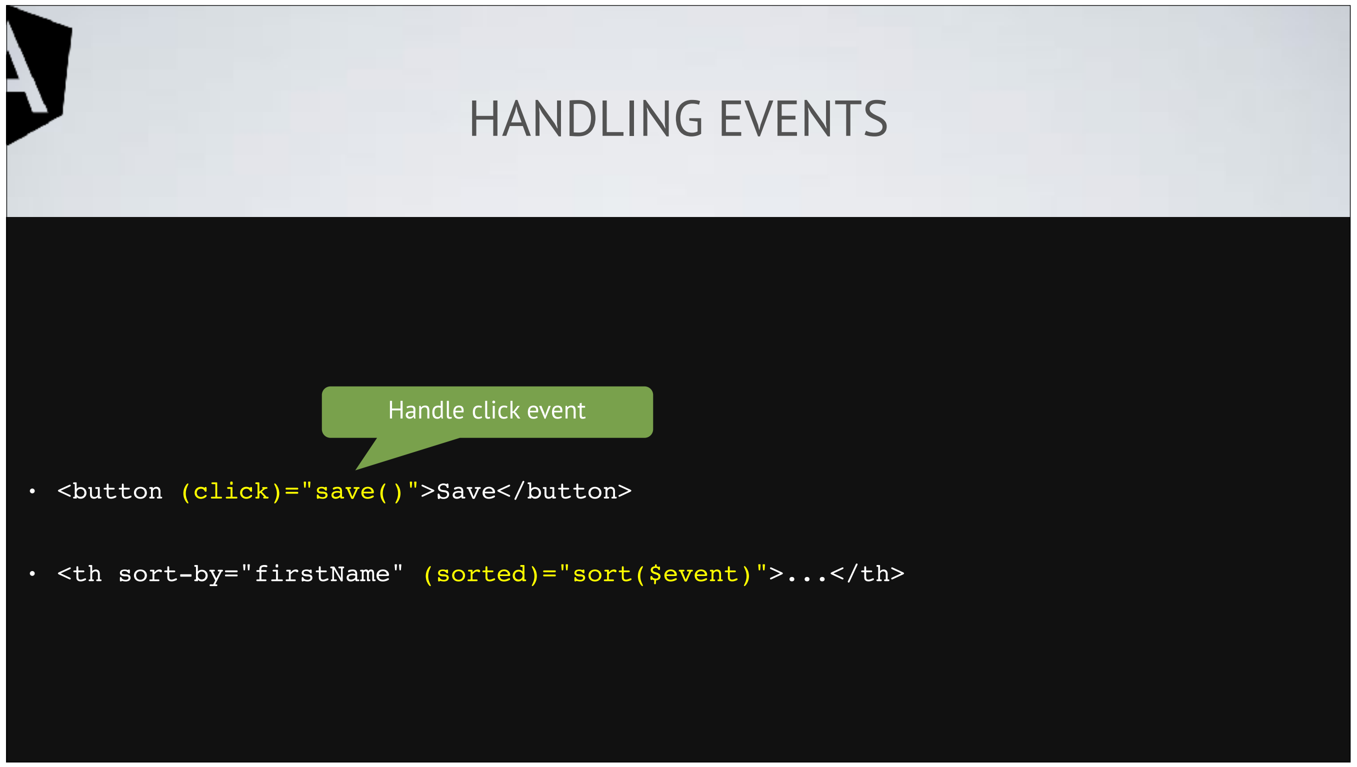


ONE-WAY BINDING SYNTAX

Bind to DOM property

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

<div [hidden]="!isVisible"

     [class.active]="isActive">...</div>
```

Binding and change detection relies on a tree of components (no cycles)
Create one-way bindings using [property] or bind-property syntax

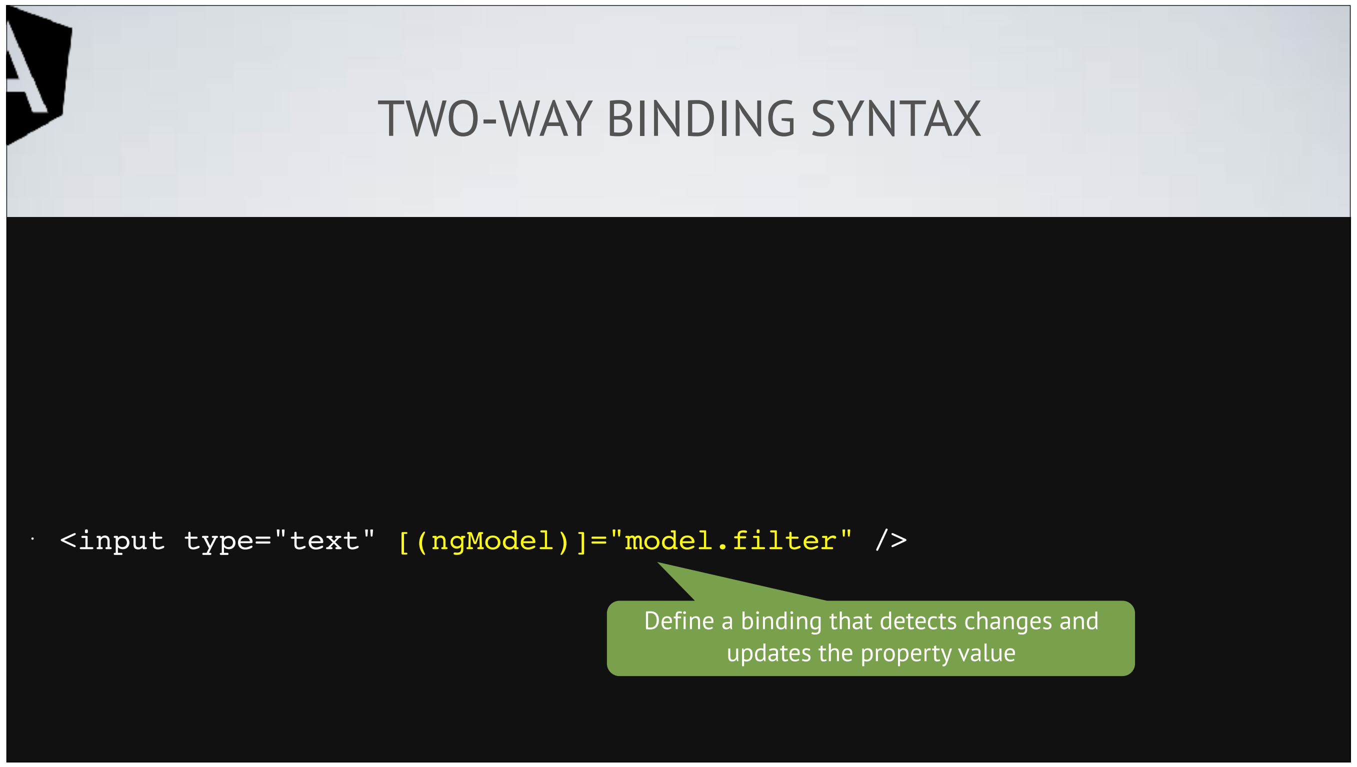


HANDLING EVENTS

Handle click event

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

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

banana in a box
Angular 2 simplifies working with events and eliminates the need for many directives
Handle an event using the (event) or on-event syntax

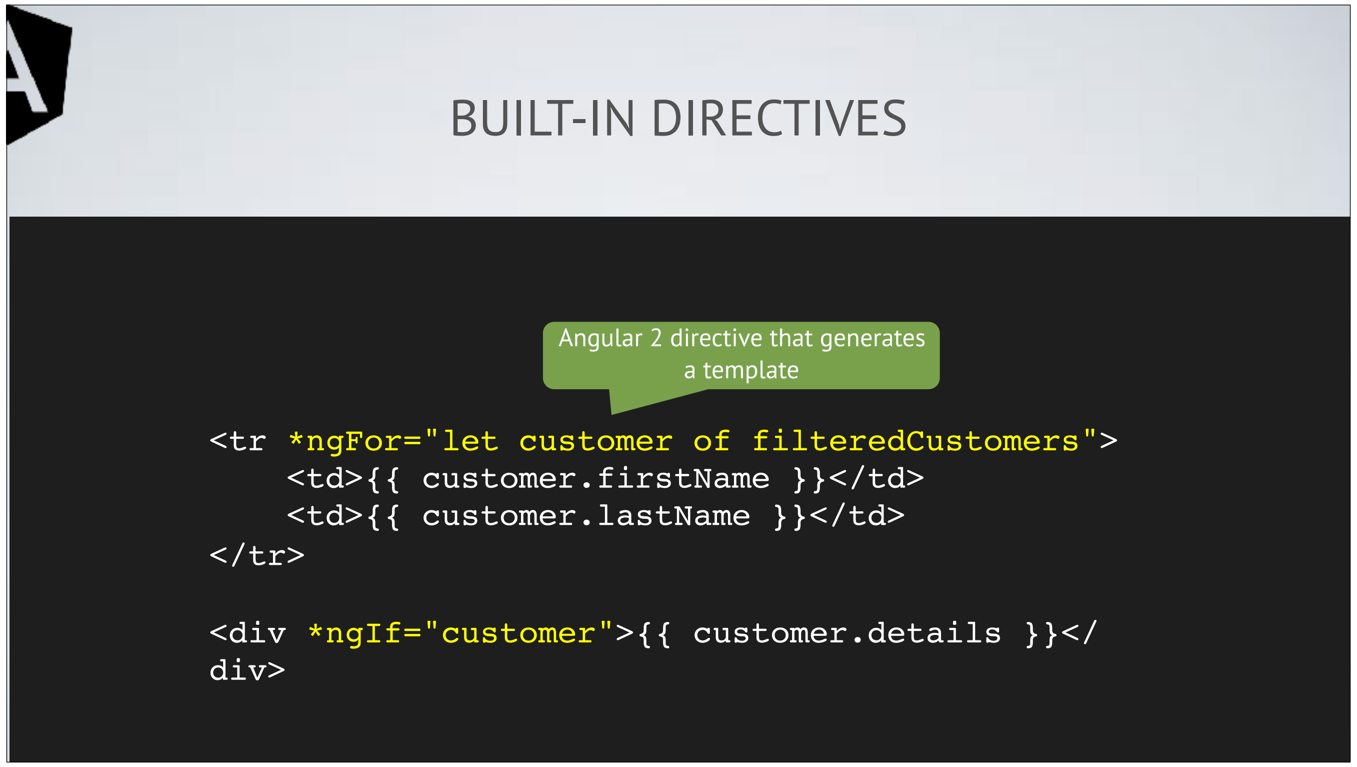


TWO-WAY BINDING SYNTAX

```
<input type="text" [(ngModel)]="model.filter" />
```

Define a binding that detects changes and updates the property value

Create two-way bindings using [(property)] or bindon-property syntax
Change triggers an event that Angular uses to update the property value



BUILT-IN DIRECTIVES

Angular 2 directive that generates a template

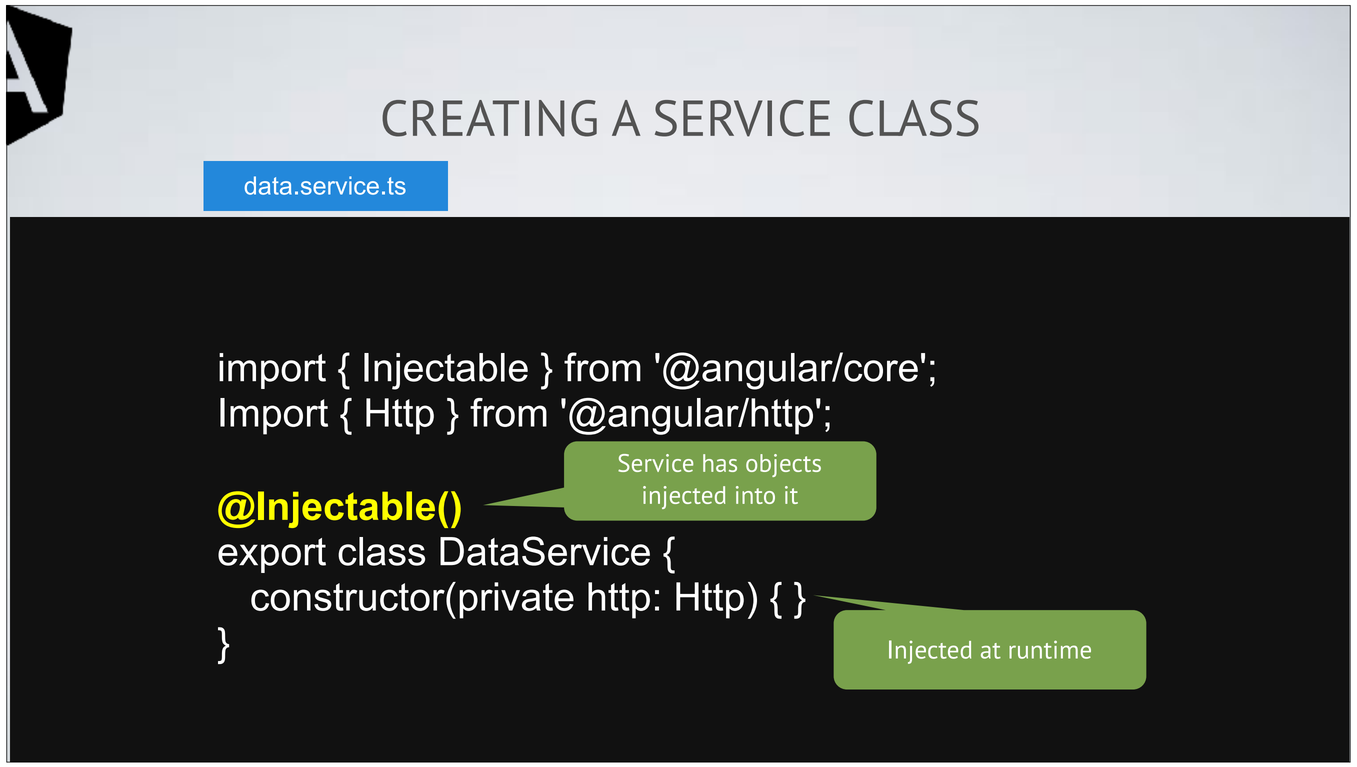
```
<tr *ngFor="let customer of filteredCustomers">
    <td>{{ customer.firstName }}</td>
    <td>{{ customer.lastName }}</td>
</tr>

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

BUILT-IN PIPES

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

Format as US currency

SERVICES



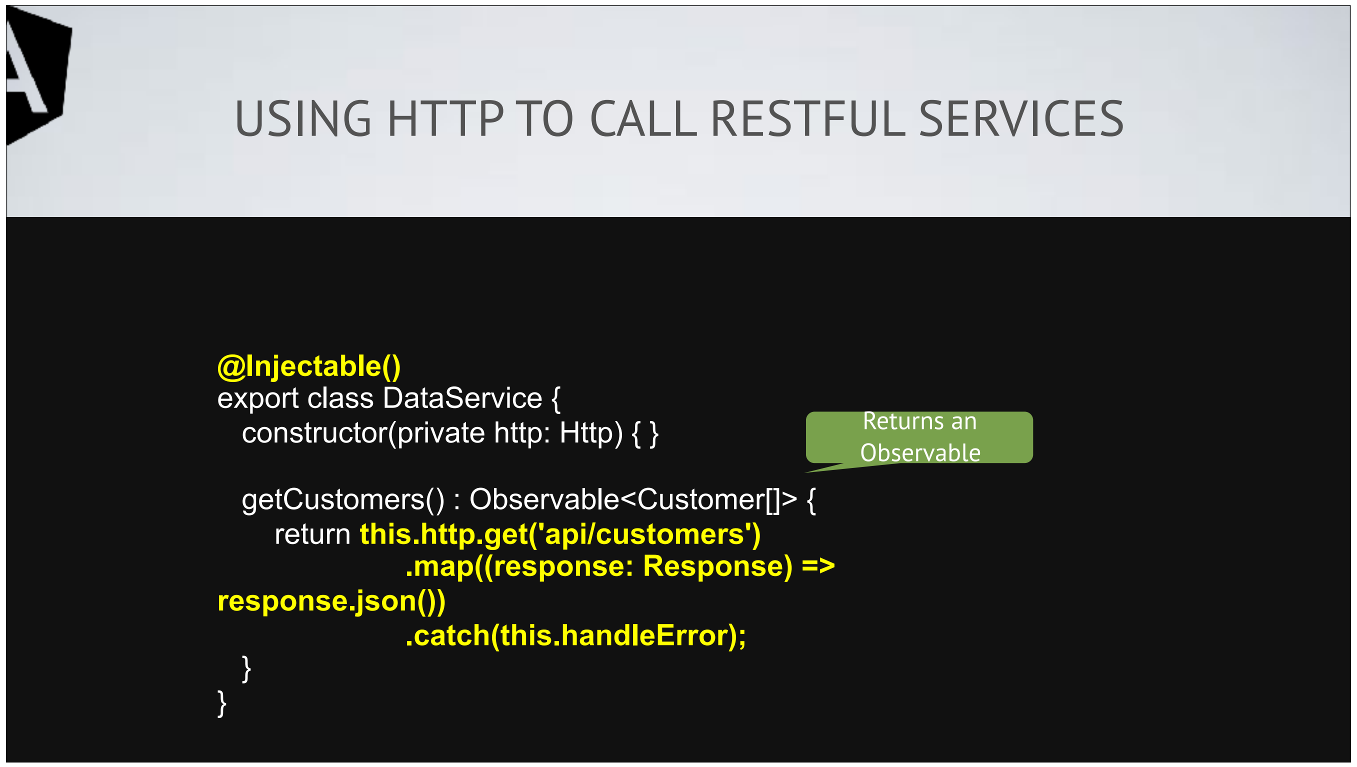
CREATING A SERVICE CLASS

data.service.ts

```
import { Injectable } from '@angular/core';
Import { Http } from '@angular/http';

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

Service has objects injected into it

Injected at runtime

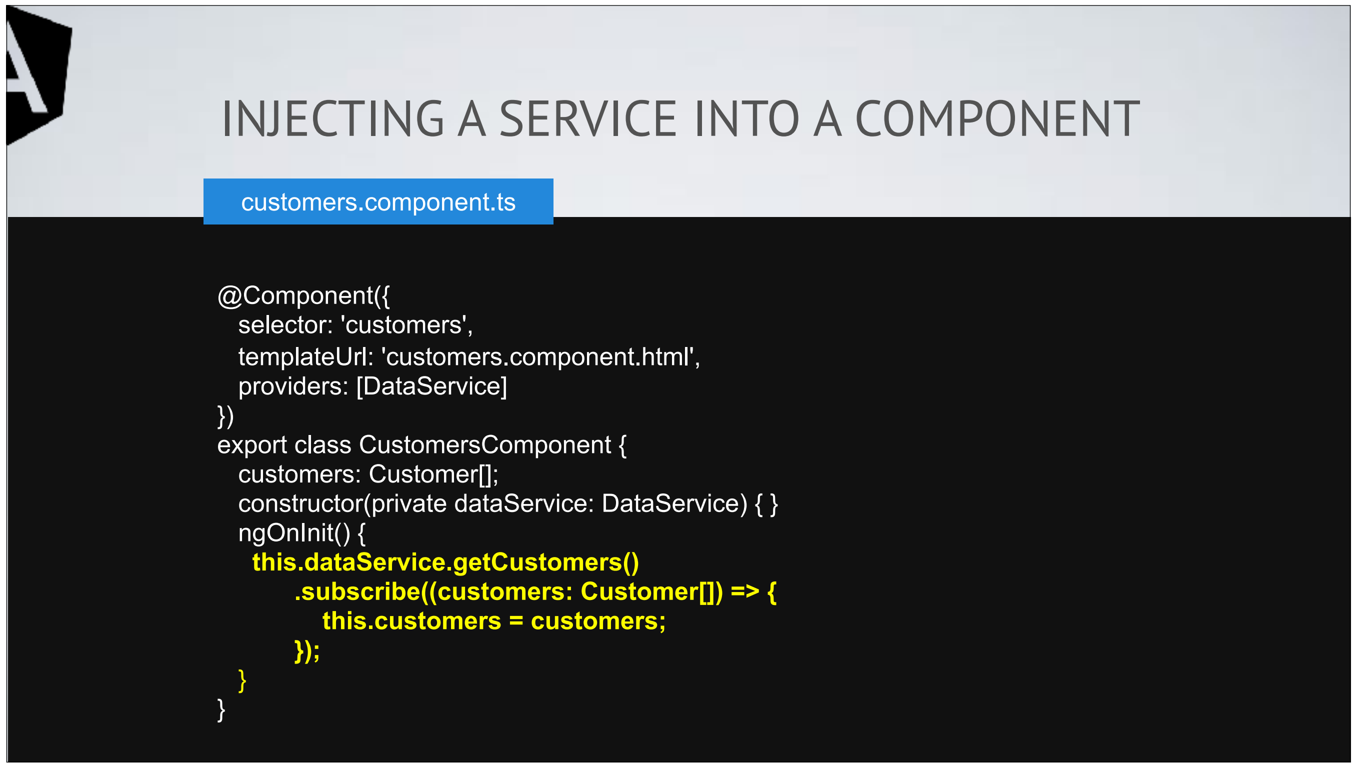


USING HTTP TO CALL RESTFUL SERVICES

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

   getCustomers() : Observable<Customer[]> {
       return this.http.get('api/customers')
                      .map((response: Response) =>
response.json())
                      .catch(this.handleError);
   }
}
```

Returns an Observable

RESTful services can be called using Http (@angular/http module)
Need to add HTTP_PROVIDERS
Can use Observables or Promises for async operations
Standard get, put, post, delete (and more) functions are supported



INJECTING A SERVICE INTO A COMPONENT

customers.component.ts

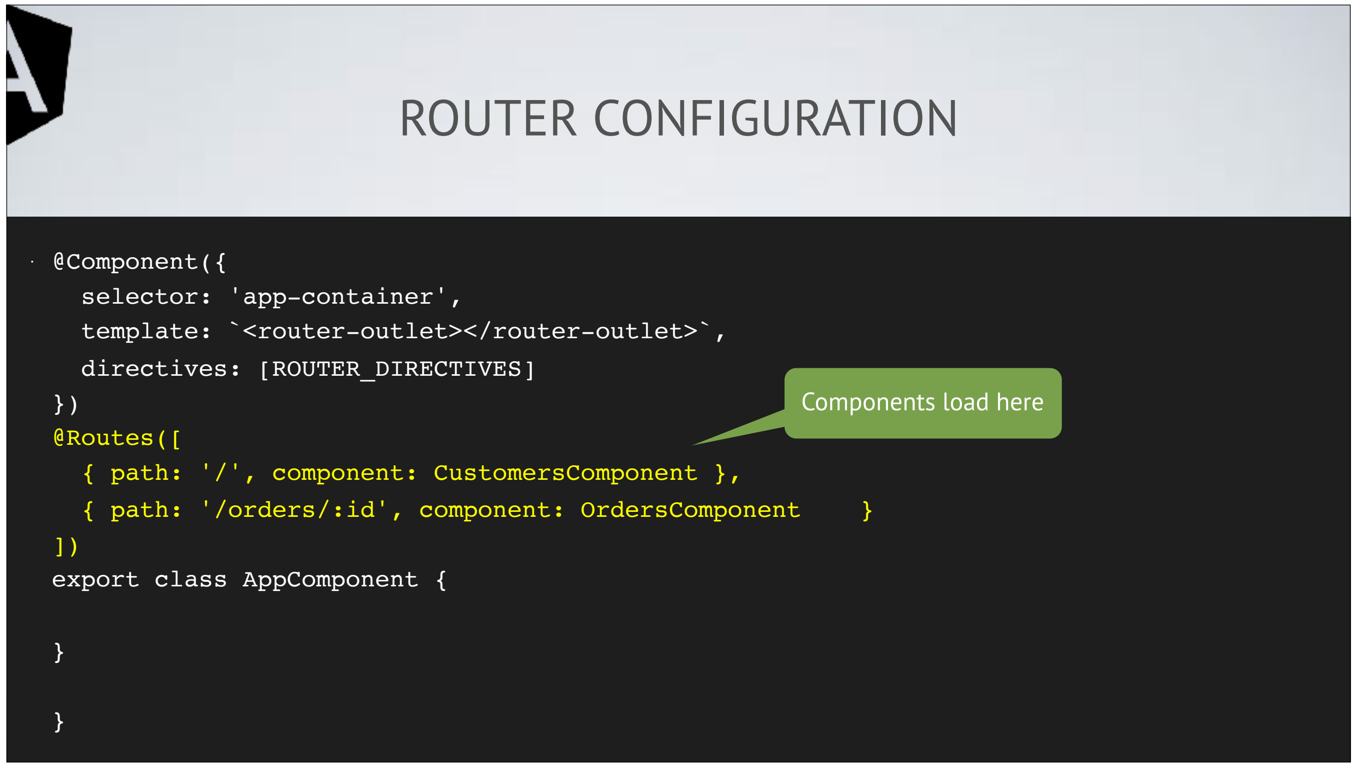
```
@Component({
   selector: 'customers',
   templateUrl: 'customers.component.html',
   providers: [DataService]
})
export class CustomersComponent {
   customers: Customer[];
   constructor(private dataService: DataService) { }
   ngOnInit() {
     this.dataService.getCustomers()
           .subscribe((customers: Customer[]) => {
               this.customers = customers;
           });
   }
}
```

Services can be "provided" to components

The following component has DataService injected into it at runtime

ROUTING

Angular is all about building blocks. So if you're someone who likes to build things this is a fun story to hear about!

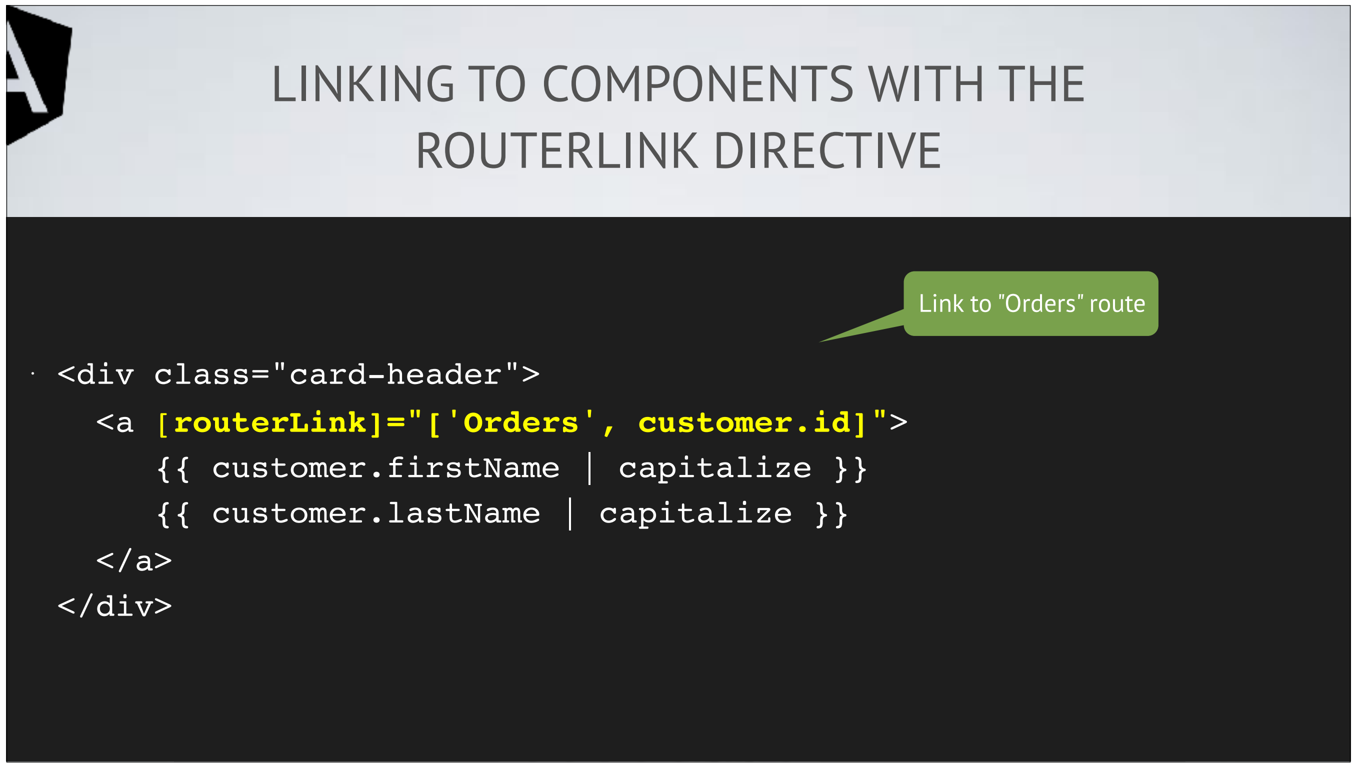


ROUTER CONFIGURATION

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

}

}
```

Define routes using the @Routes() decorator

Components are loaded into the <router-outlet></router-outlet> area

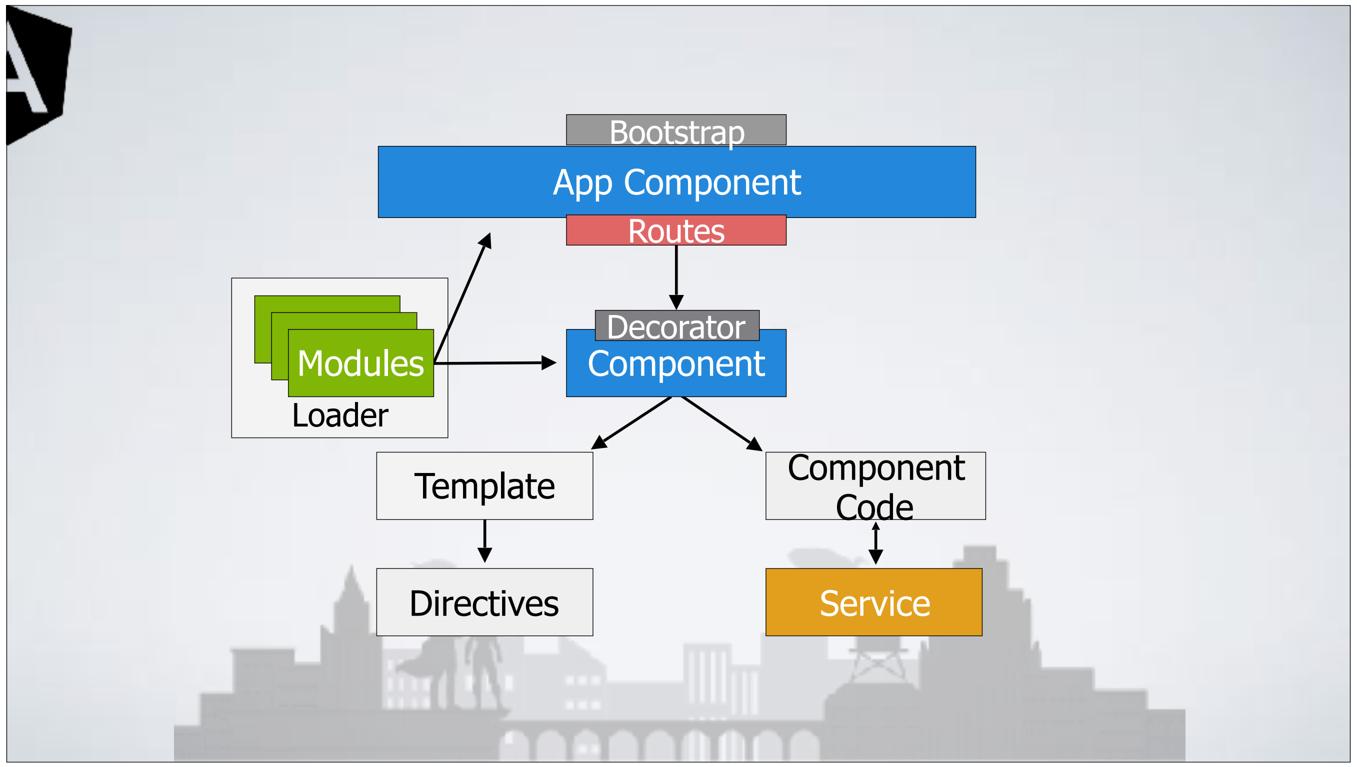


LINKING TO COMPONENTS WITH THE ROUTERLINK DIRECTIVE

Link to "Orders" route

```
<div class="card-header">
  <a [routerLink]="['Orders', customer.id]">
     {{ customer.firstName | capitalize }}
     {{ customer.lastName | capitalize }}
  </a>
</div>
```

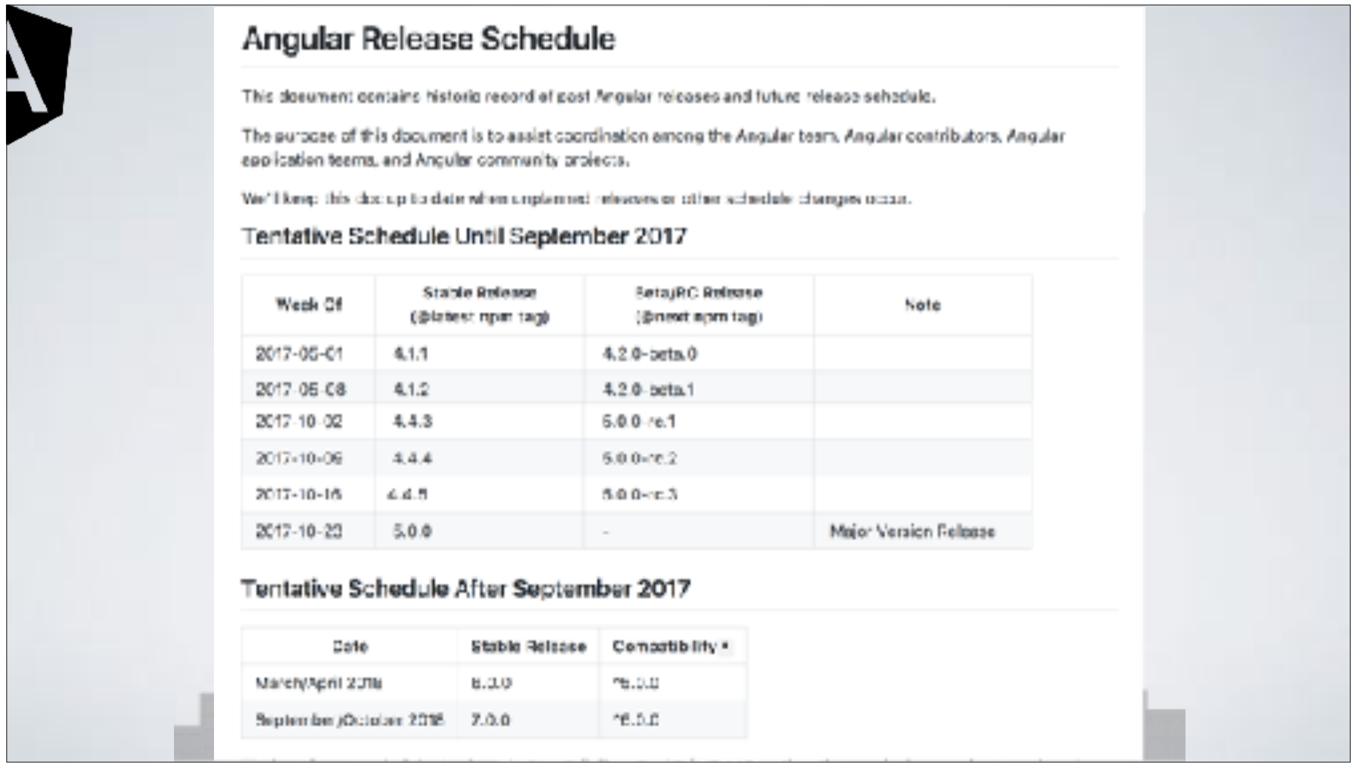
Navigate between Components using the RouterLink directive


Bootstrap
App Component
Routes
Modules
Loader
Decorator
Component
Template
Component Code
Directives
Service

ANGULAR 5

THE COOL PARTS™

Angular is all about building blocks. So if you're someone who likes to build things this is a fun story to hear about!



Angular Release Schedule

This document contains historic record of past Angular releases and future release schedule.

The purpose of this document is to assist coordination among the Angular team, Angular contributors, Angular application teams, and Angular community projects.

We'll keep this doc up to date when unplanned releases or other schedule changes occur.

Tentative Schedule Until September 2017

Week Of	Stable Release (@latest npm tag)	Beta/RC Release (@next npm tag)	Note
2017-05-01	4.1.1	4.2.0-beta.0	
2017-05-08	4.1.2	4.2.0-beta.1	
2017-10-02	4.4.3	5.0.0-rc.1	
2017-10-09	4.4.4	5.0.0-rc.2	
2017-10-16	4.4.5	5.0.0-rc.3	
2017-10-23	5.0.0	-	Major Version Release

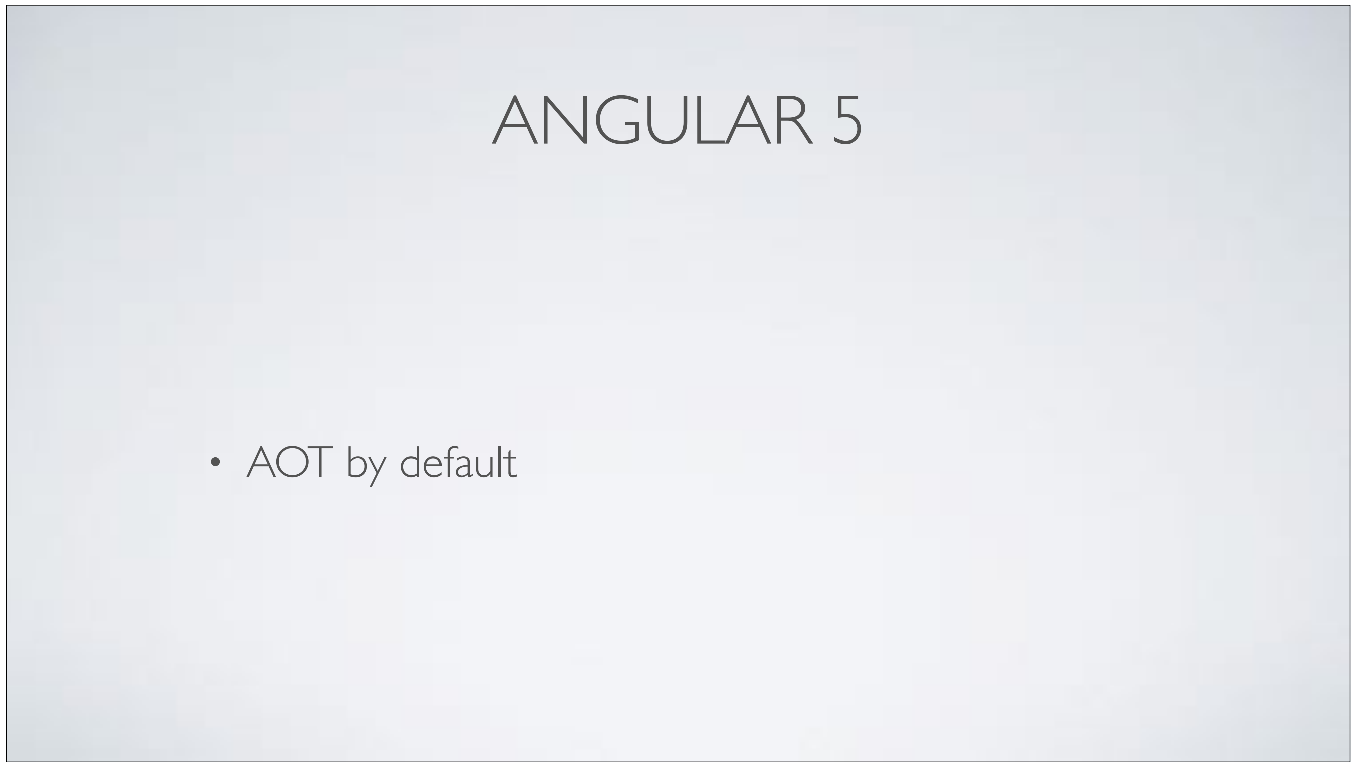
Tentative Schedule After September 2017

Date	Stable Release	Compatibility *
March/April 2018	6.0.0	^5.0.0
September/October 2018	7.0.0	^6.0.0

Release date got pushed back till Oct 23rd

ANGULAR 4

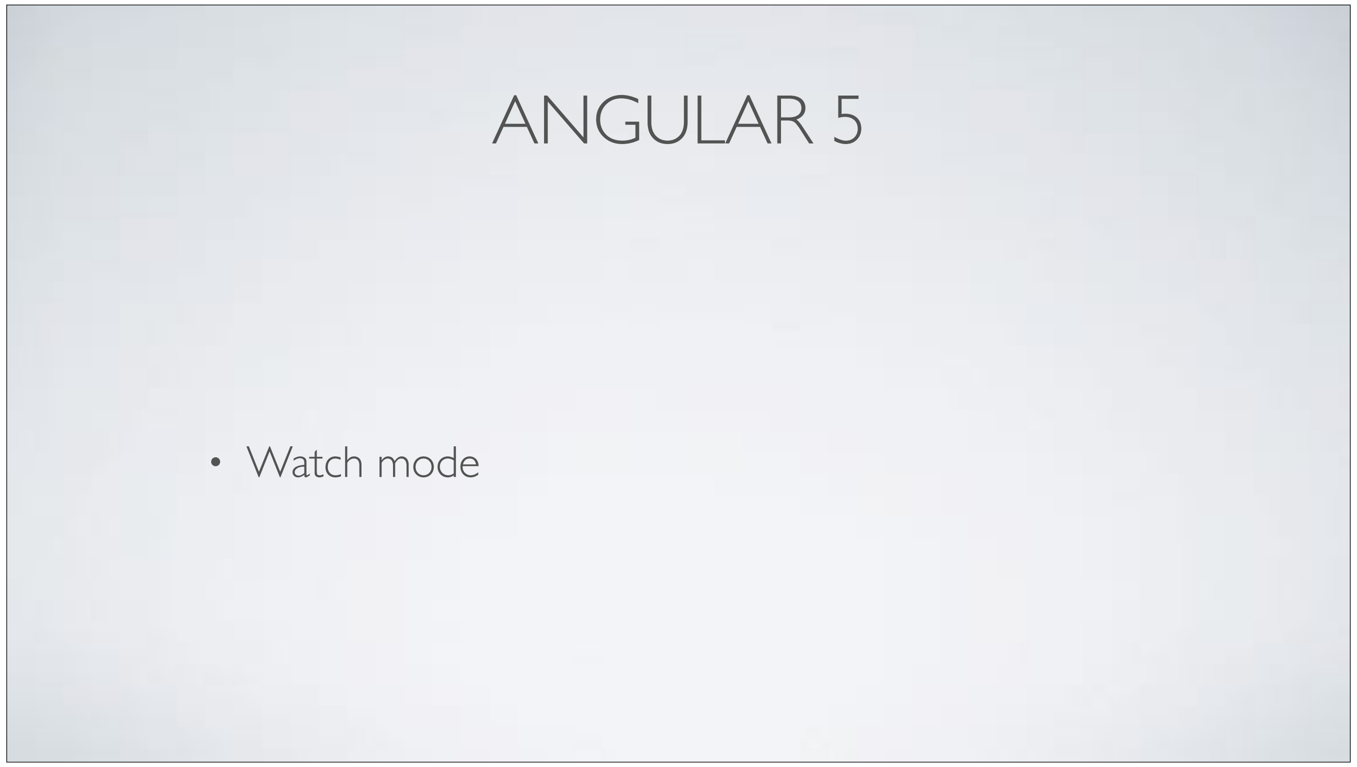
- TypeScript 2.x
- @angular/cli & @angular/universal
- @angular/platform-browser/animations



ANGULAR 5

- AOT by default

One of the Angular’s team goals is to simplify the way we compile the application. It’s very common that in the development mode we use what we call Just-In-Time (JIT) compilation. JIT is a very fast compilation process that helps with enabling the fast development lifecycle. Then, when you’re ready to go to production, use the optimizing compiler that we call the Ahead-Of-Time (AOT) compiler. Sometimes, there are differences between this two, which can be frustrating, because you work on an application and when you’re ready to ship you realize that there are things that you still need to do before you can actually build up an application and push it to production.
The team is going to make the AOT compilation the default. It will be done by making it much faster and by enabling incremental compilation. This means that we will have a single compilation process that we will use throughout the whole application development. The process will reduce the number of the friction in the development.
Another thing is focusing on size and speed. We have heard a lot about the “60% reduction of bundle size or the application size across the board”. This was a good achievement but the team realized that they can still do much more.

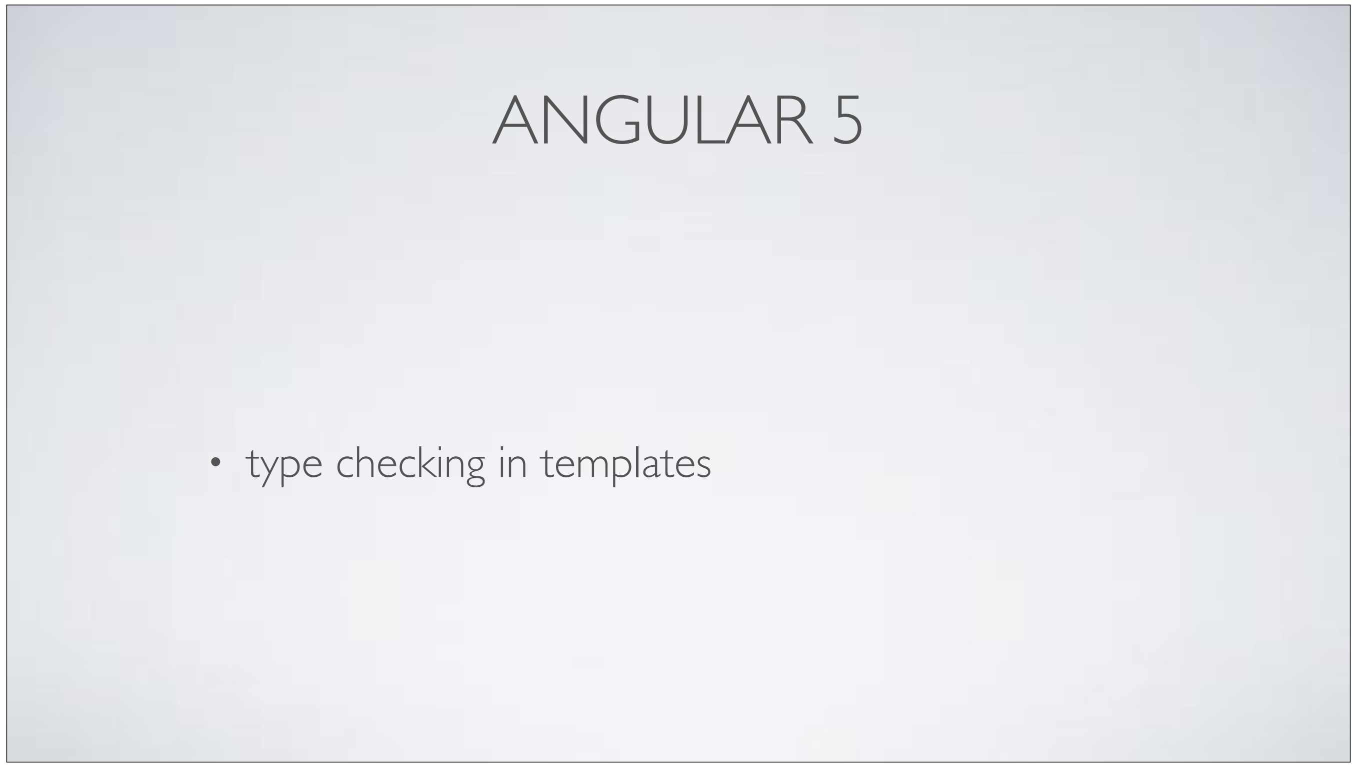


ANGULAR 5

- Watch mode

Watch mode

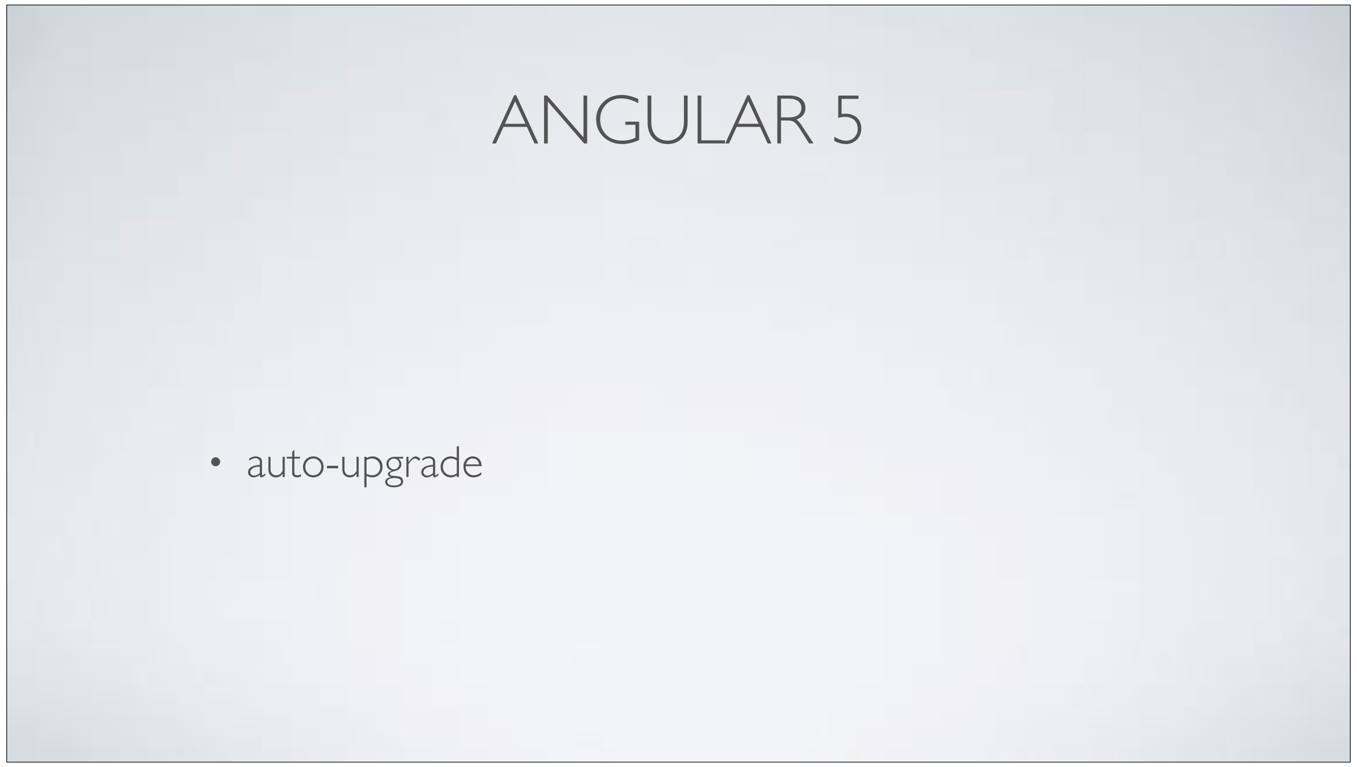
As we can run tsc —-watch, we want to have the same for the Angular's compiler (ngc) and integrate it with Angular-CLI. We want to realize as soon as possible if our code is good or not.



ANGULAR 5

- type checking in templates

Type checking in templates
Another thing they will actually work on is the type checking in templates. If we used ahead of time compilation and we misspelled the property, then there will be a type check error in our *.ngfactory files. For someone who is not an expert, this doesn't tell anything because when looking at these *.ngfactory files we don't know what Angular does under the hood. So, the team will change this to be a type checker that will report regarding our template, so it will say " line: 2, column: 3 you made a mistake in property x" and it will work, also, with Typescript strict-null-checks.
Remove *.ngfactory.ts files
Angular's offline compiler compiles templates into Typescript files with .ngfactory.ts extension. The Typescript compiler compiles this files into JavaScript. So, it (type-)checks and detects typo and type errors in our templates if any. The team will change the compiled pipeline to inline the *.ngfactory into our code, while we compile it.



ANGULAR 5

- auto-upgrade

Smooth upgrades, just like with version 4, are amazing. They want to make it super easy for us to jump on and use it. It can be done only by making sure that there are no obstacles and that there is no reason to stick to version 4 or 5. Does it sound like an automatic upgrade? 😱

Google invests big resources in the field of upgrading application. It is inevitable to supply an upgrade path to the developers that don't start from zero and coming to Angular with an AngularJS-based application. The team doesn't stop and will continue working on helping these developers to stay in front and enjoy new features. Version 5 will bring us the following improvements.

- Lazy loading for AngularJS/Angular
- Less boilerplate and better tooling
- Improve the performance of hybrid applications
- Better dual-router
- Improve the testing
- More documentation and examples

CHEERS!

@LAKATOS88

Alex Lakatos