

# Angular Fundamentals

---

## GETTING STARTED WITH ANGULAR



**Jim Cooper**

SOFTWARE CRAFTSMAN

@jimthecoop



# Required Prerequisites



**Basic JavaScript**  
[app.pluralsight.com/paths/skills/javascript](https://app.pluralsight.com/paths/skills/javascript)

**Basic HTML**  
[app.pluralsight.com/paths/skills/html5](https://app.pluralsight.com/paths/skills/html5)



# Helpful Prerequisites



## **Basic Node and Npm**

[app.pluralsight.com/courses/npm-playbook](http://app.pluralsight.com/courses/npm-playbook)

## **Modules and Module Loaders**

[app.pluralsight.com/courses/javascript-module-fundamentals](http://app.pluralsight.com/courses/javascript-module-fundamentals)

## **ES2015**

[app.pluralsight.com/courses/javascript-fundamentals-es6](http://app.pluralsight.com/courses/javascript-fundamentals-es6)

## **TypeScript**

[app.pluralsight.com/courses/typescript](http://app.pluralsight.com/courses/typescript)



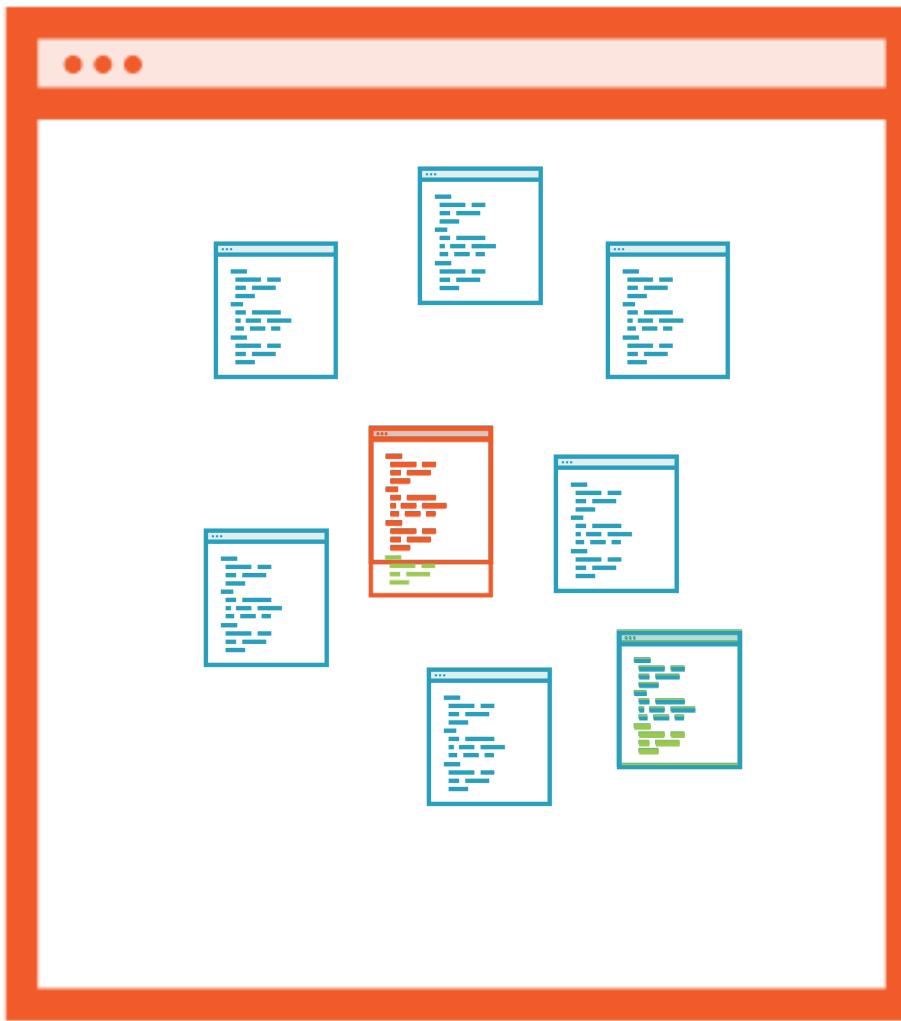
# Why Modules are Important



# Why Modules are Important



# Why Modules are Important



```
import { foo } from  
'..//folder/some-file.js'
```

```
export {  
  foo: someFunction()  
}
```



# What is SystemJs?

```
index.html  
  
<script src="file1.js">...  
<script src="file2.js">...  
<script src="file3.js">...  
<script src="file4.js">...  
<script src="file5.js">...  
<script src="file6.js">...  
.  
.
```



# What is SystemJs?

index.html

```
<script src="system.js">  
<script src="config.js">
```

system.config.js

```
var config = {  
  map: {  
    'app': '/folder/app'  
  },  
  packages: {  
    'app': {main: 'main.js'}  
  }  
}
```



# ES2015 Features

**let and const**

**Arrow Functions**

**Array Methods**

**Classes**



```
function doSomething(x) {  
  var y = 10  
  ...  
}  
  
console.log(y) // logs undefined
```

let and const



```
function doSomething(x) {  
  if (x) {  
    var y = 10  
  }  
  console.log(y) // logs 10  
}
```

let and const



```
function doSomething(x) {  
  if (x) {  
    let y = 10  
  }  
  console.log(y) // logs undefined  
}
```

let and const



```
function doSomething() {  
  const y = 10  
  y = 20 // exception  
}
```

let and const



```
function(x) { (x) => {
    if (x) if (x)
        return 10 return 10
    else else
        return 20 return 20
}
```

## Arrow Functions



```
var cats = [ {name: 'Fluffy'}, {name: 'Muffin'} ]  
var muffin = cats.find(cat => cat.name === 'Muffin')  
var muffin = cats.find(cat => cat.name == 'Muffin')  
console.log(muffin.name) // logs 'Muffin'
```

## Find and Filter Array Methods



```
var cats = [ {name: 'Fluffy'}, {name: 'Muffin'} ]  
var cats = cats.filter(cat => cat.name.indexOf('u') > -1)  
console.log(cats[0].name) // logs 'Fluffy'  
console.log(cats[1].name) // logs 'Muffin'
```

## Find and Filter Array Methods



```
var cat = {name: 'Fluffy', color: 'White'}
```

## ES2015 Classes



```
var Cat = new function(name, color) {  
    this.name = name  
    this.color = color  
}
```

```
var fluffy = new Cat('Fluffy', 'White')
```

## ES2015 Classes



```
var Cat = new function(name, color) {  
    this.name = name  
    this.color = color  
}  
Cat.prototype.speak = function() {  
    console.log('meow')  
}  
var fluffy = new Cat('Fluffy', 'White')
```

## ES2015 Classes



```
class Cat {  
  constructor (name, color) {  
    this.name = name;  
    this.color = color;  
  }  
  speak() { console.log('meow') }  
}  
  
var fluffy = new Cat('Fluffy', 'White')
```

## ES2015 Classes



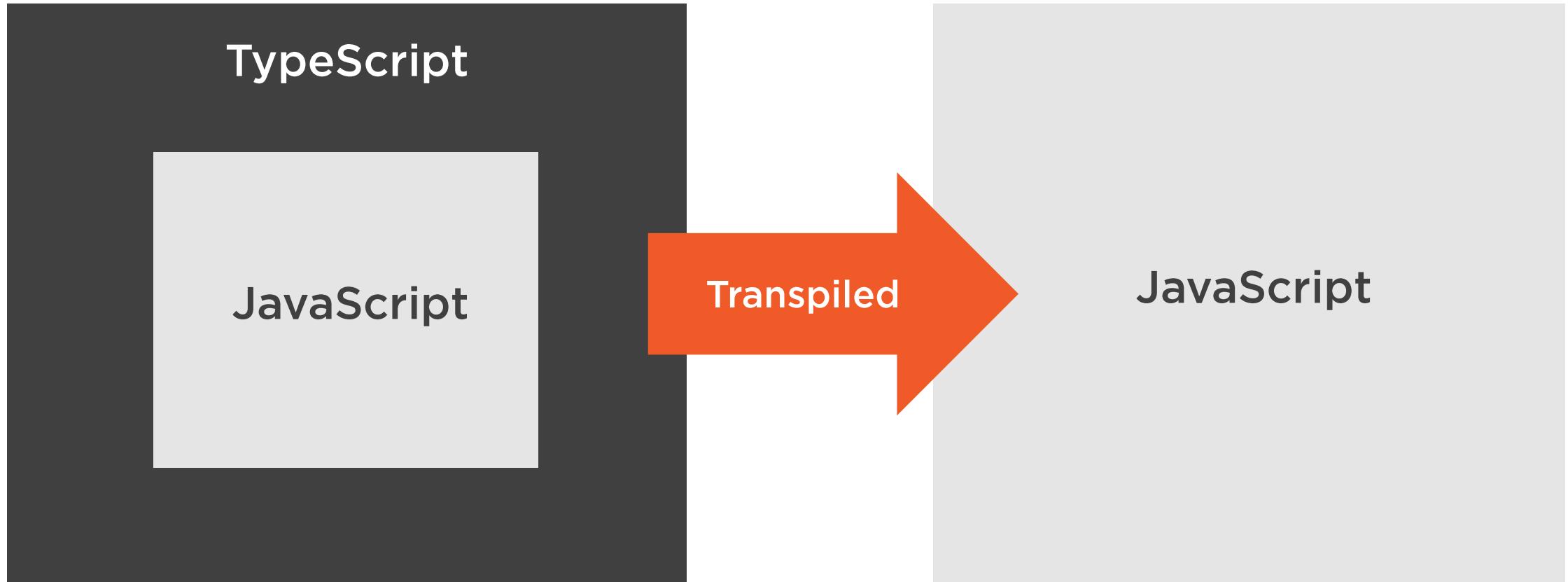
# TypeScript Overview

TypeScript

JavaScript



# TypeScript Overview



# ES2015 Features

Static Typing

Interfaces

Class Properties

Public/Private Accessibility



```
let name : string  
let age : number  
let birthDate : date
```

## Static Typing



```
interface ICat {  
    name:string  
    age:number  
}
```

# TypeScript Interfaces



```
interface ICat {  
    name:string  
    age:number  
}  
  
let fluffy:ICat = {  
    name: 'Fluffy',  
    age: 'seven'  
}
```

## TypeScript Interfaces



```
interface ICat {  
    name:string  
    age:number  
}  
  
let fluffy:ICat = {  
    name: 'Fluffy'  
}
```

## TypeScript Interfaces



```
interface ICat {  
    name:string  
    age?:number  
}  
  
let fluffy:ICat =  
{  
    name: 'Fluffy'  
}
```

## TypeScript Interfaces



```
class Cat {  
  constructor (name) {  
    this.name = name  
  }  
}
```

## TypeScript Class Properties



```
class Cat {  
    name:string  
    constructor (name) {  
        this.name = name;  
    }  
}
```

## TypeScript Class Properties



```
class Cat {  
    name:string  
    color:string  
    constructor (name) {  
        this.name = name;  
    }  
}
```

## TypeScript Class Properties



```
class Cat {  
    name  
    color  
    constructor (name) {  
        this.name = name;  
    }  
}
```

## TypeScript Class Properties



```
class Cat {  
    name:string  
    color:string  
    constructor (name) {  
        this.name = name;  
    }  
    speak() { console.log('meow') }  
}
```

## Public and Private Accessibility



```
class Cat {  
    name:string  
    color:string  
    constructor (name) {  
        this.name = name;  
    }  
    speak() { console.log('My name is: ' + this.name) }  
}
```

## Public and Private Accessibility



```
class Cat {  
    name:string  
    speak() { console.log('My name is: ' + this.name) }  
}
```

```
let fluffy = new Cat()  
console.log(fluffy.name)  
fluffy.speak()
```

## Public and Private Accessibility



```
class Cat {  
    private name:string  
    private speak() { console.log('My name is: ' + this.name) }  
}  
  
let fluffy = new Cat()  
console.log(fluffy.name)  
fluffy.speak()
```

## Public and Private Accessibility



```
class Cat {  
    private name:string  
    private speak() { console.log('My name is: ' + this.name) }  
}
```

```
let fluffy = new Cat()  
console.log(fluffy.name) //compile-time error  
fluffy.speak() // compile-time error
```

## Public and Private Accessibility



```
class Cat {  
    private name:string  
    private color:string  
    constructor(name, color) {  
        this.name = name  
        this.color = color  
    }  
}
```

```
class Cat {  
    constructor(private name, private color) {  
    }  
}  
let fluffy = new Cat('Fluffy', 'White')
```

## Public and Private Accessibility



# Angular 2 Conceptual Overview

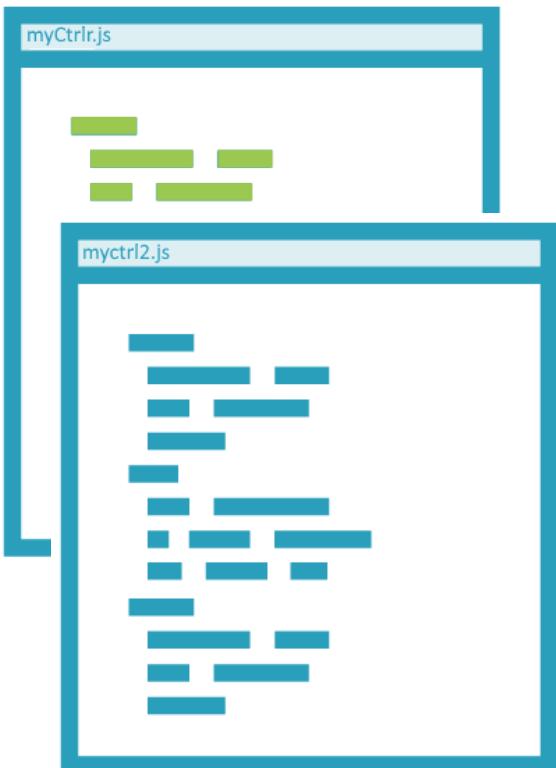
---



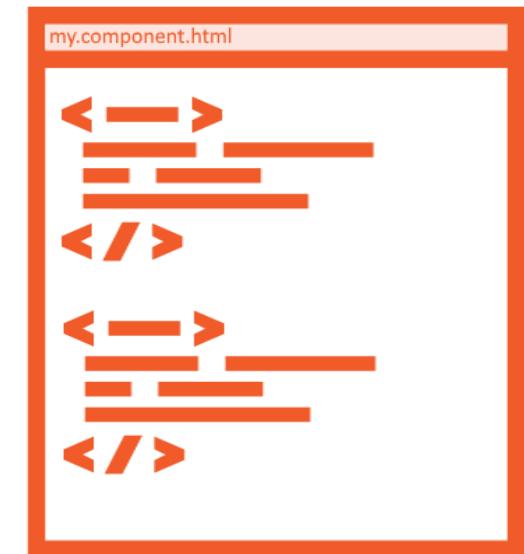
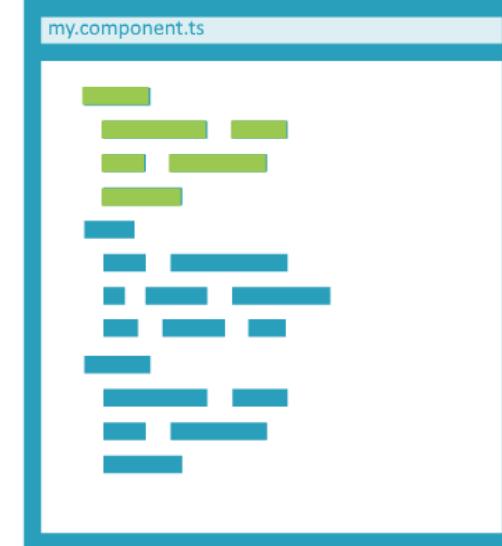
# MVC vs Components

Angular 1

```
index.html
<!-->
<!-->
</>
<ng-controller="myCtrlr">
<!-->
</>
<ng-controller="myCtrl12">
<!-->
</>
```

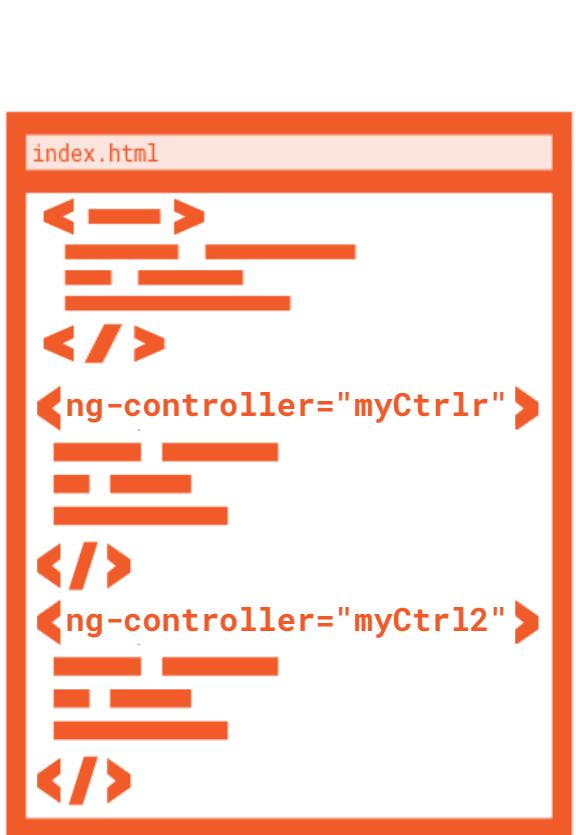


Angular 2

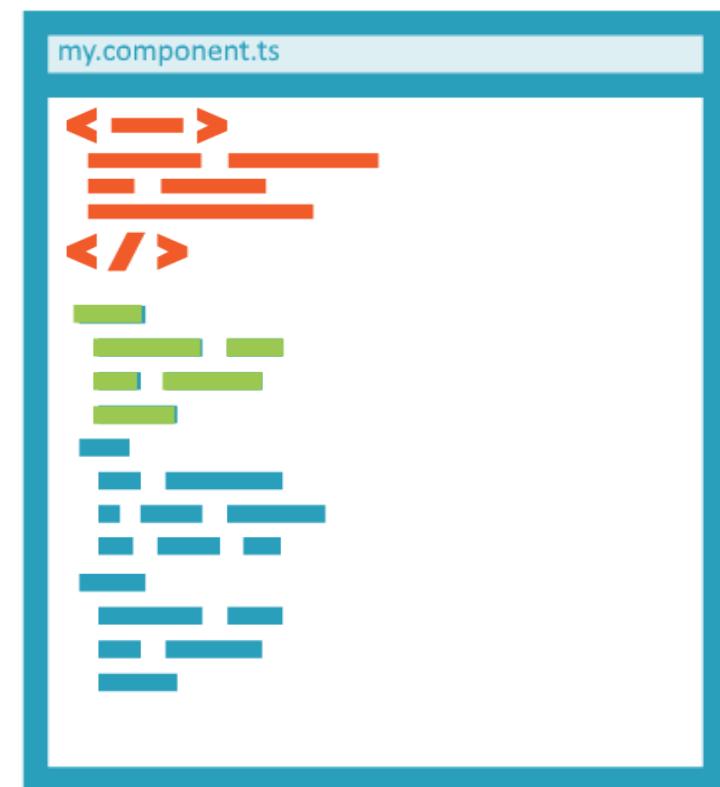


# MVC vs Components

Angular 1



Angular 2

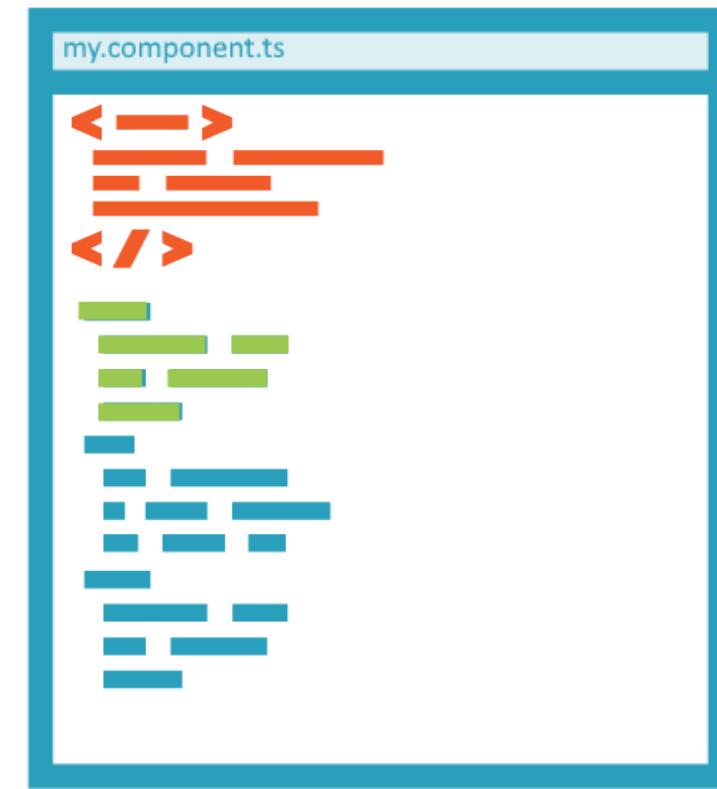


# MVC vs Components

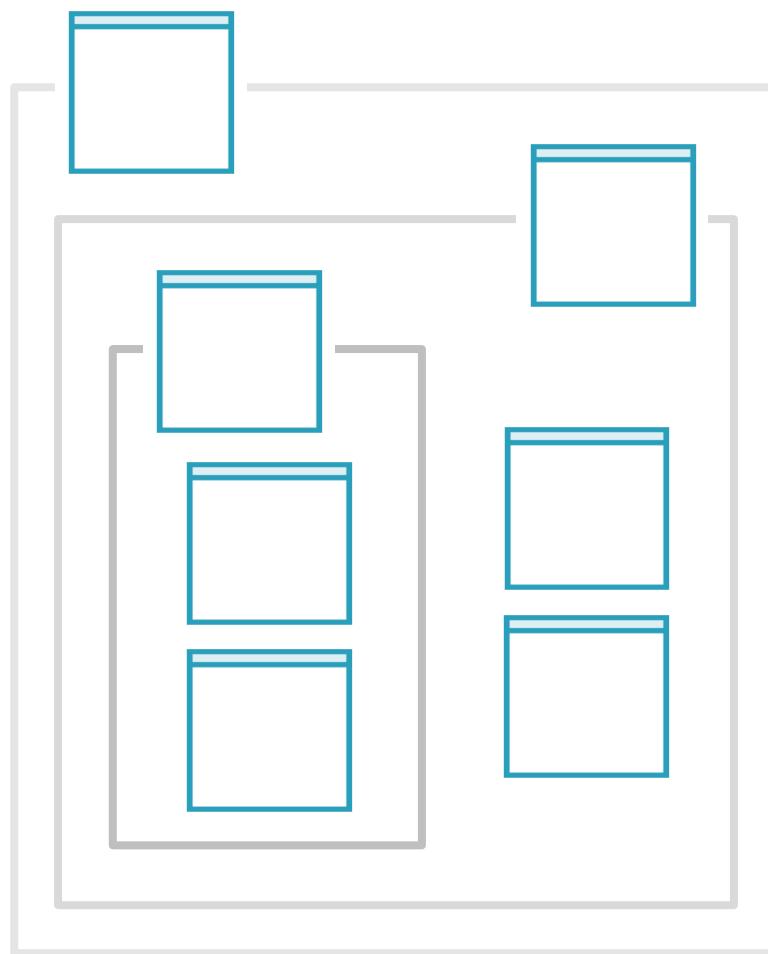
Angular 1



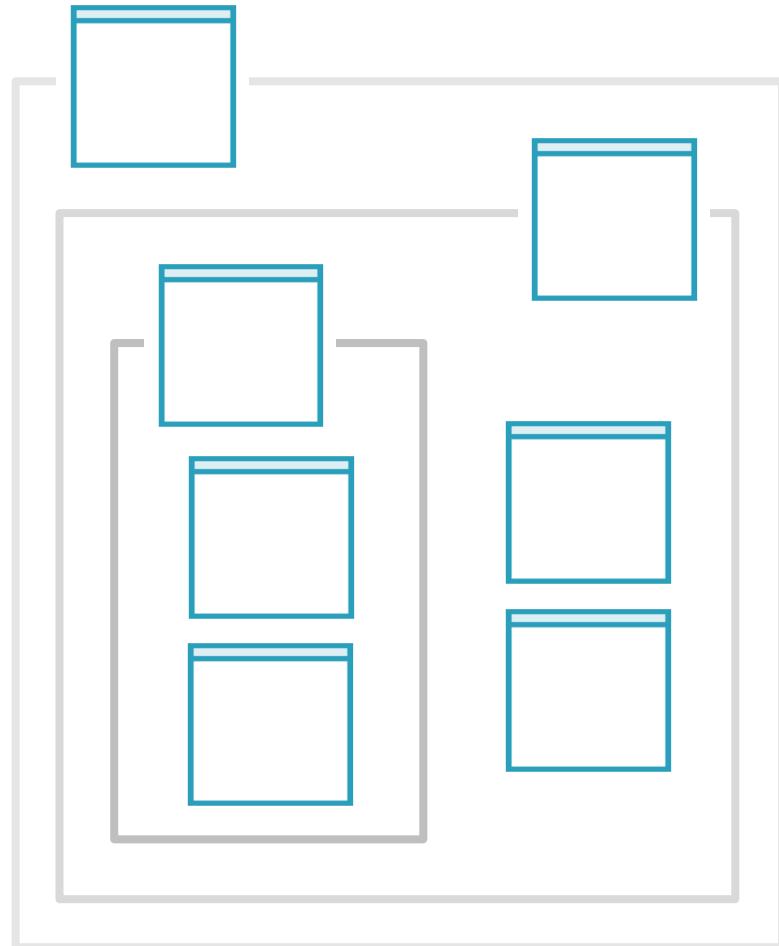
Angular 2



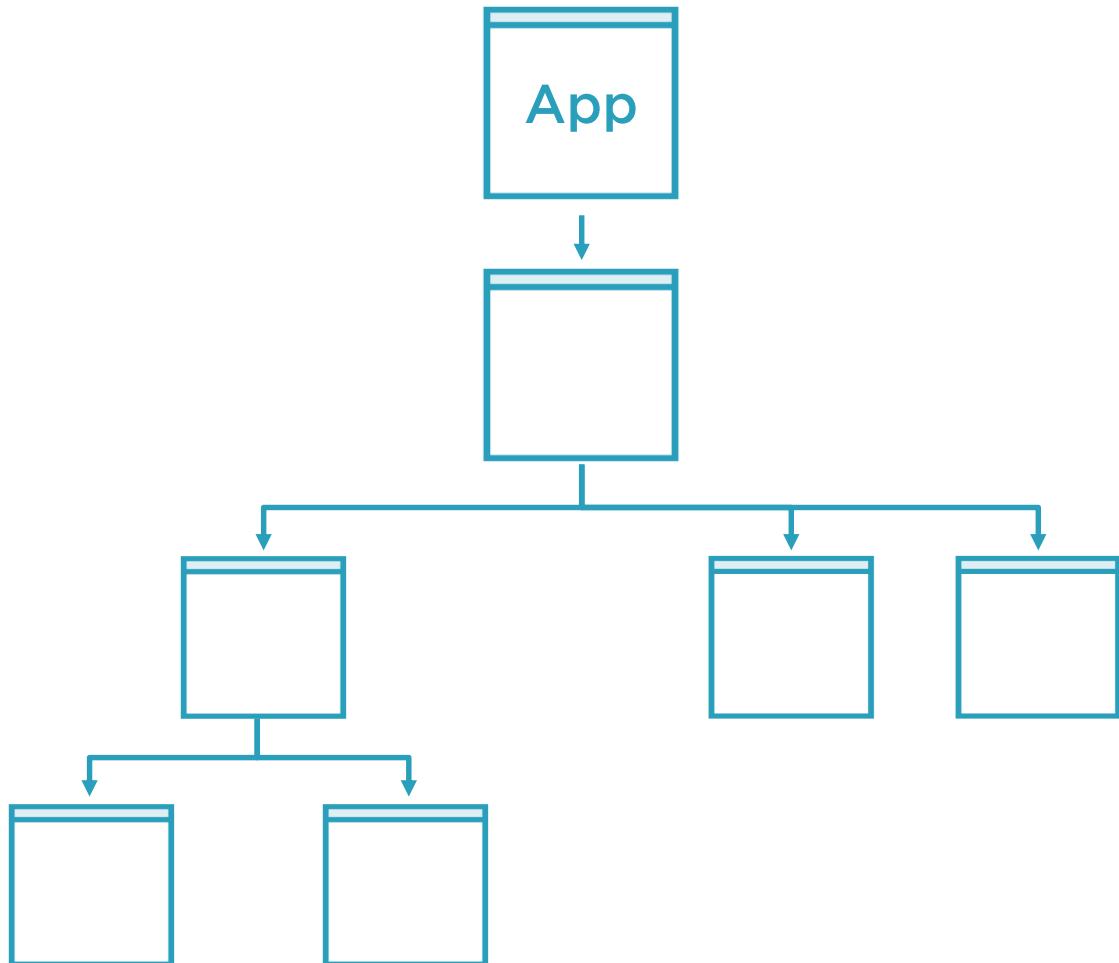
# Angular 2 Component Hierarchy



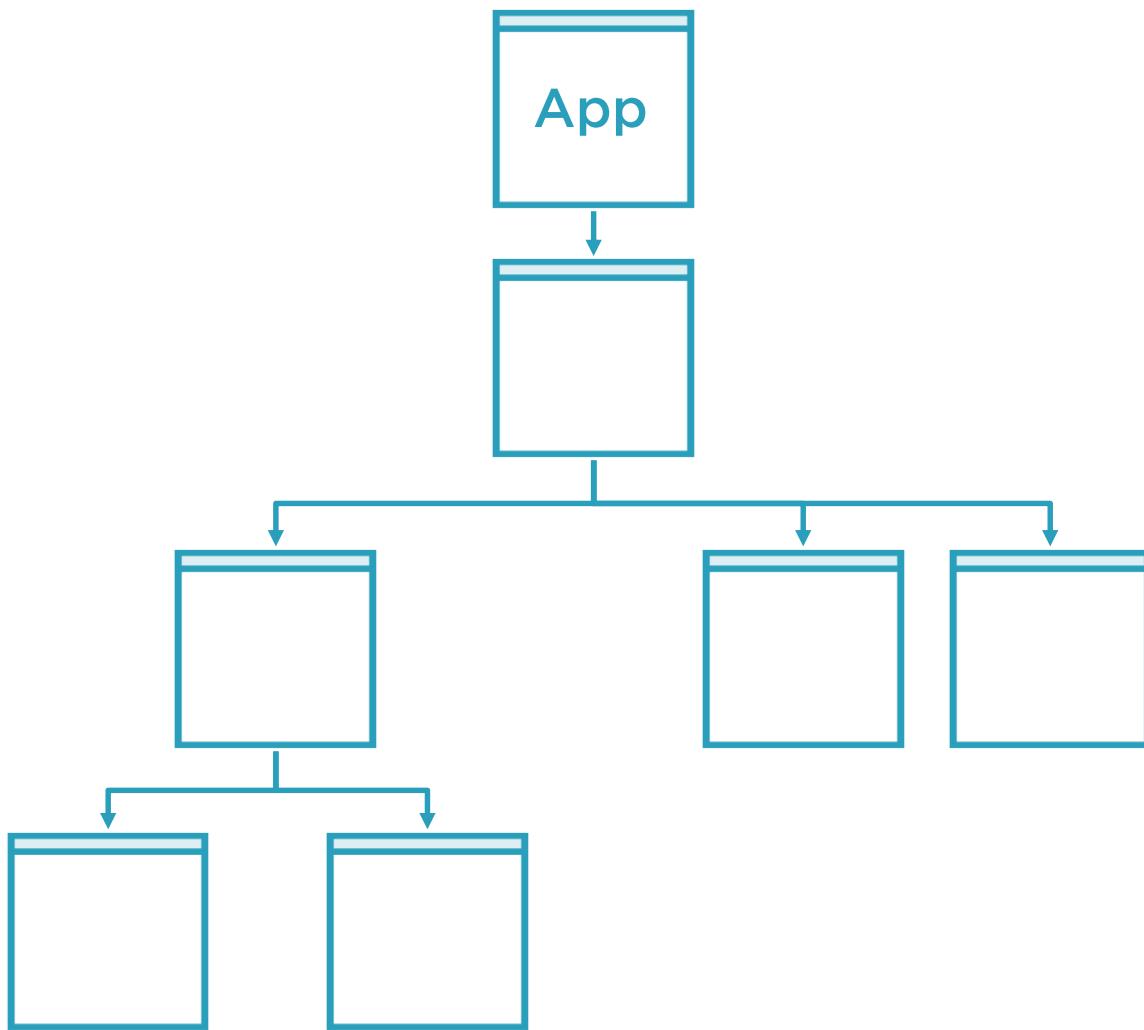
# Angular 2 Component Hierarchy



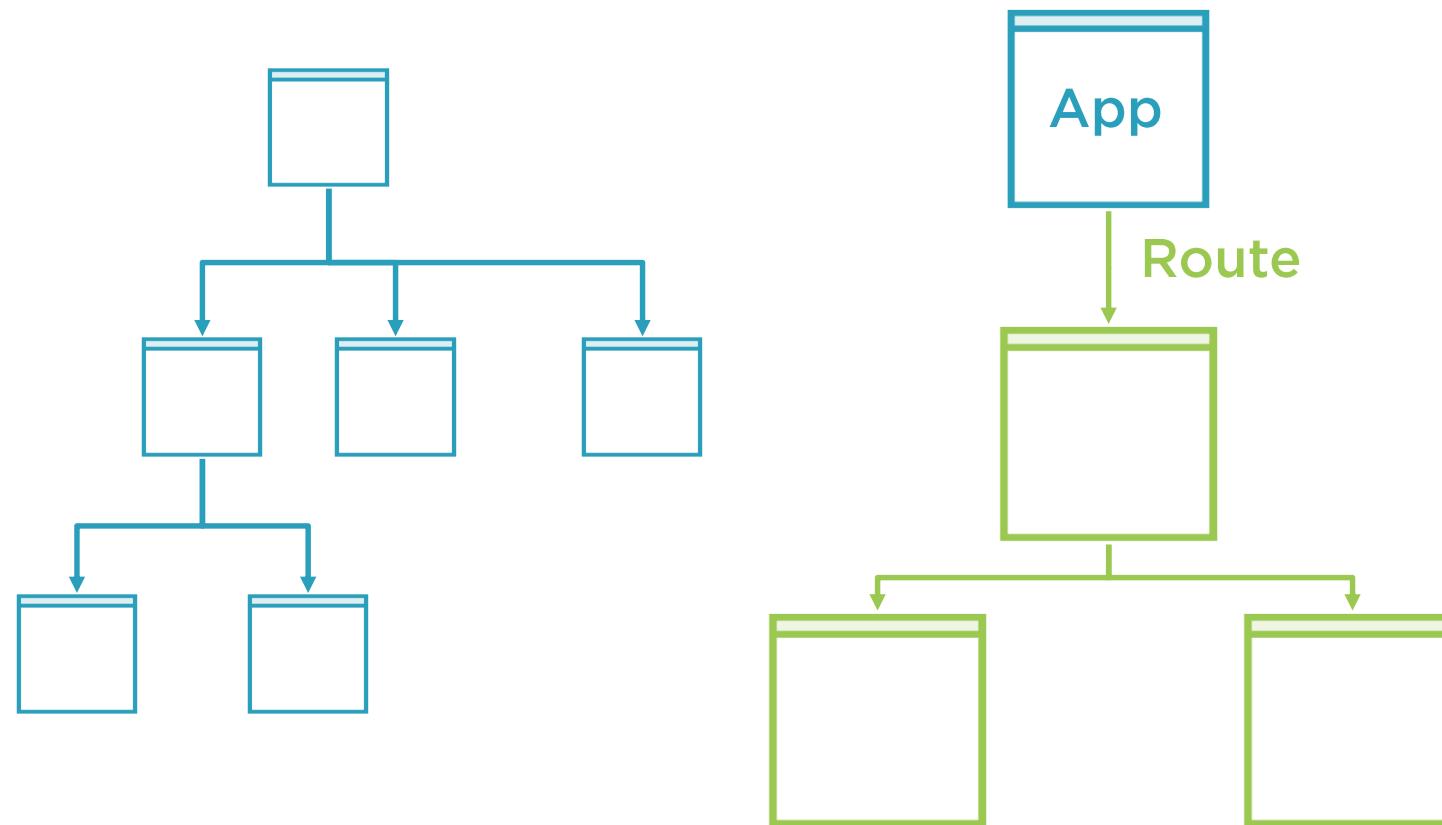
# Angular 2 Component Hierarchy



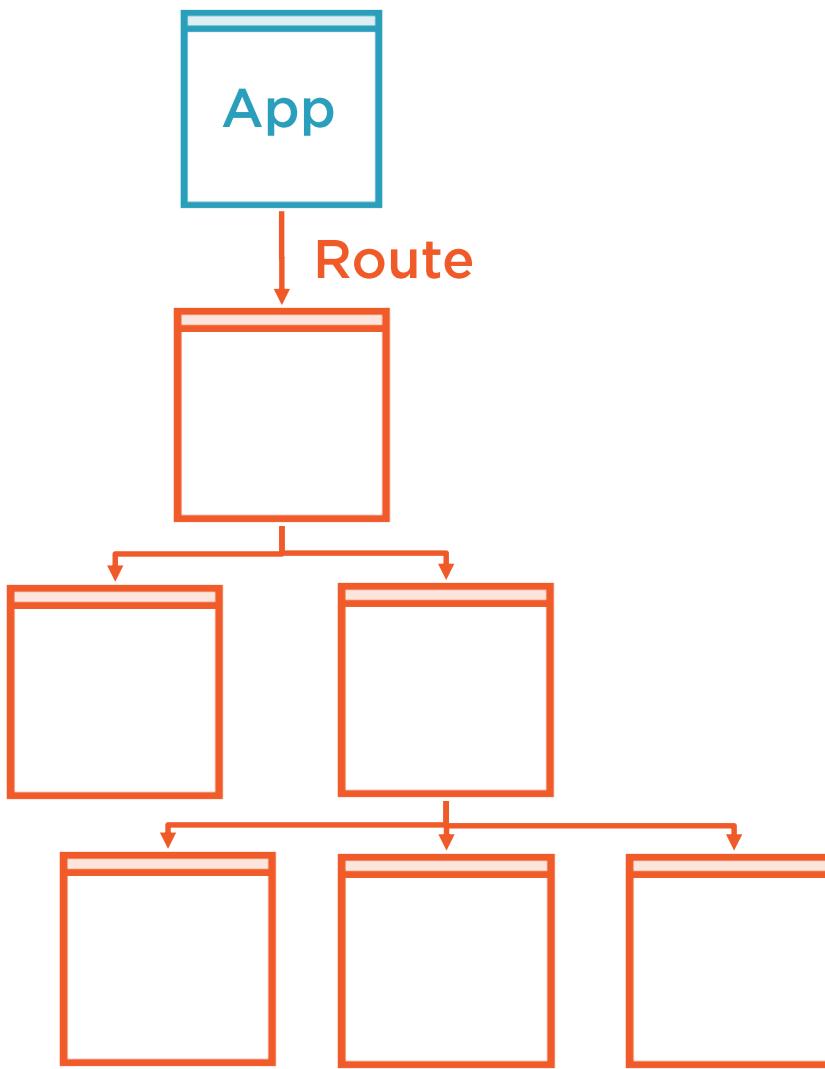
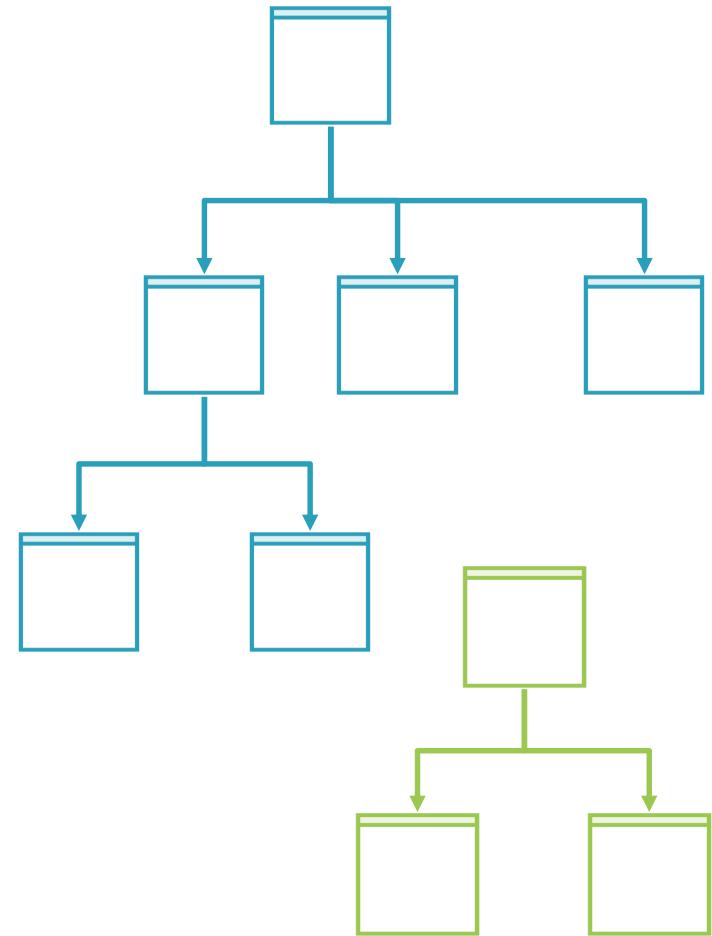
# Angular 2 Component Hierarchy



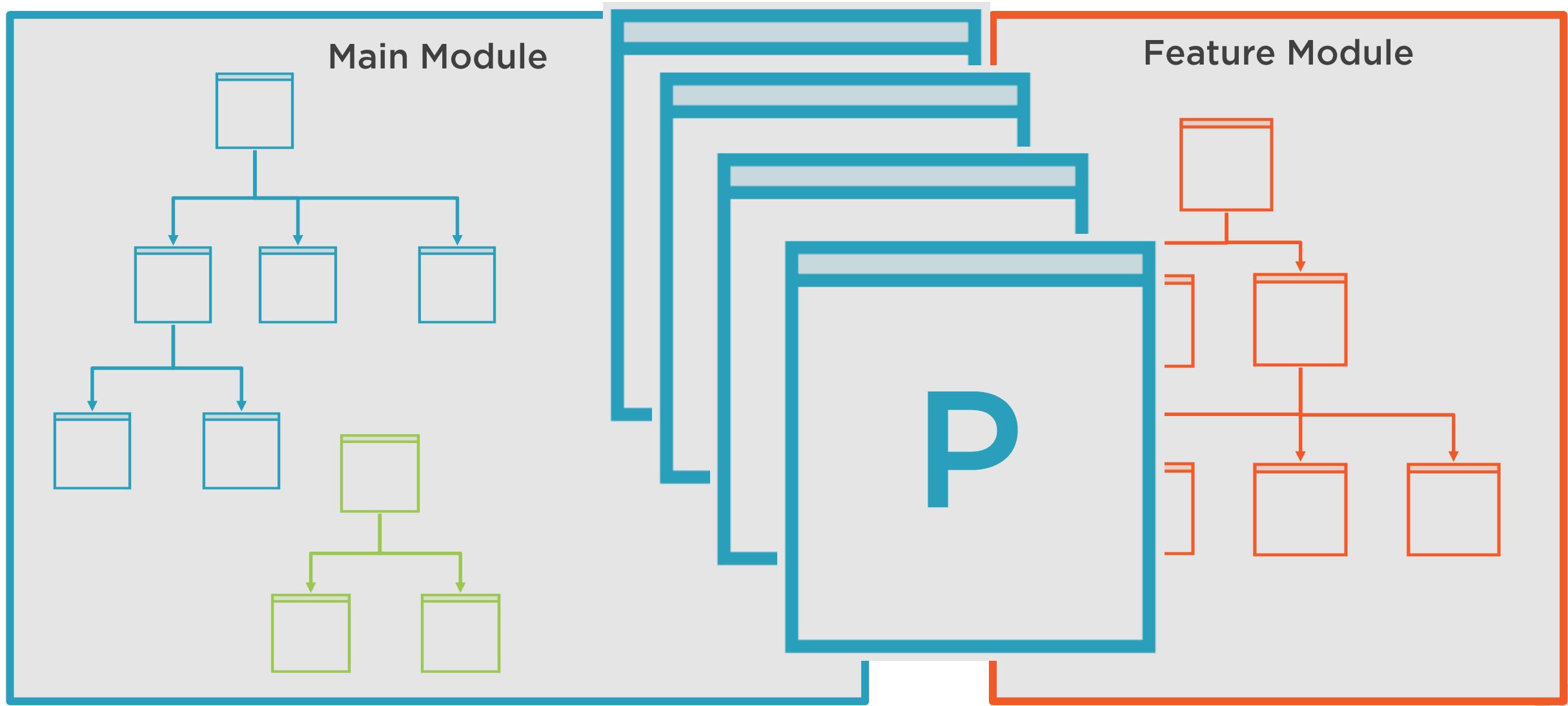
# Angular 2 Component Hierarchy



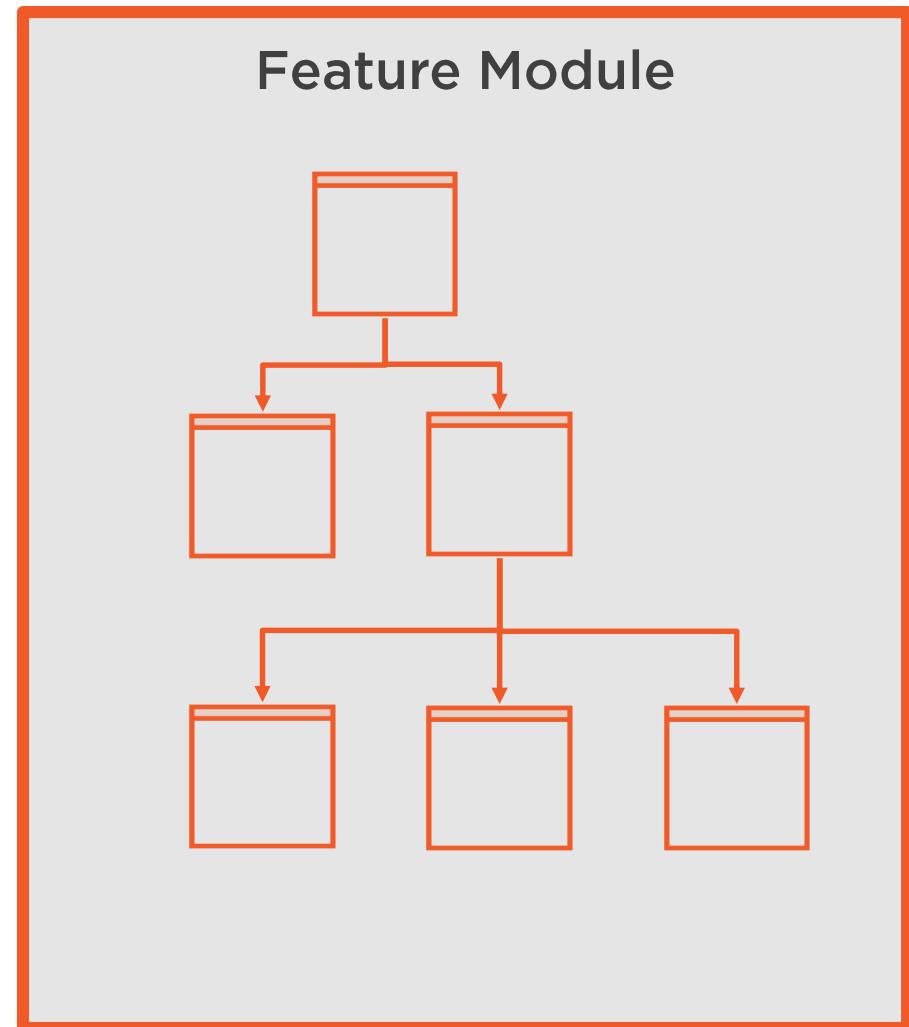
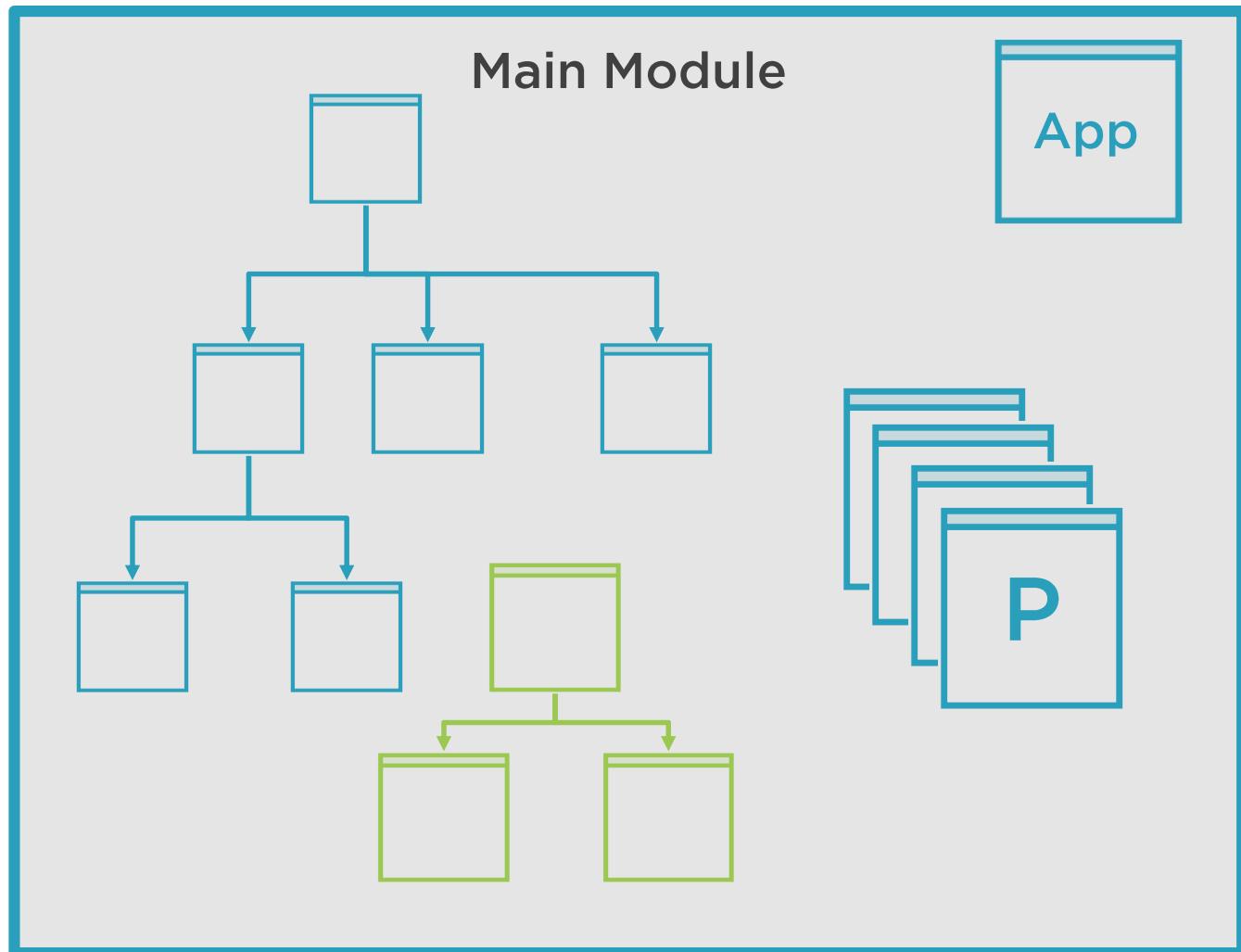
# Angular 2 Component Hierarchy



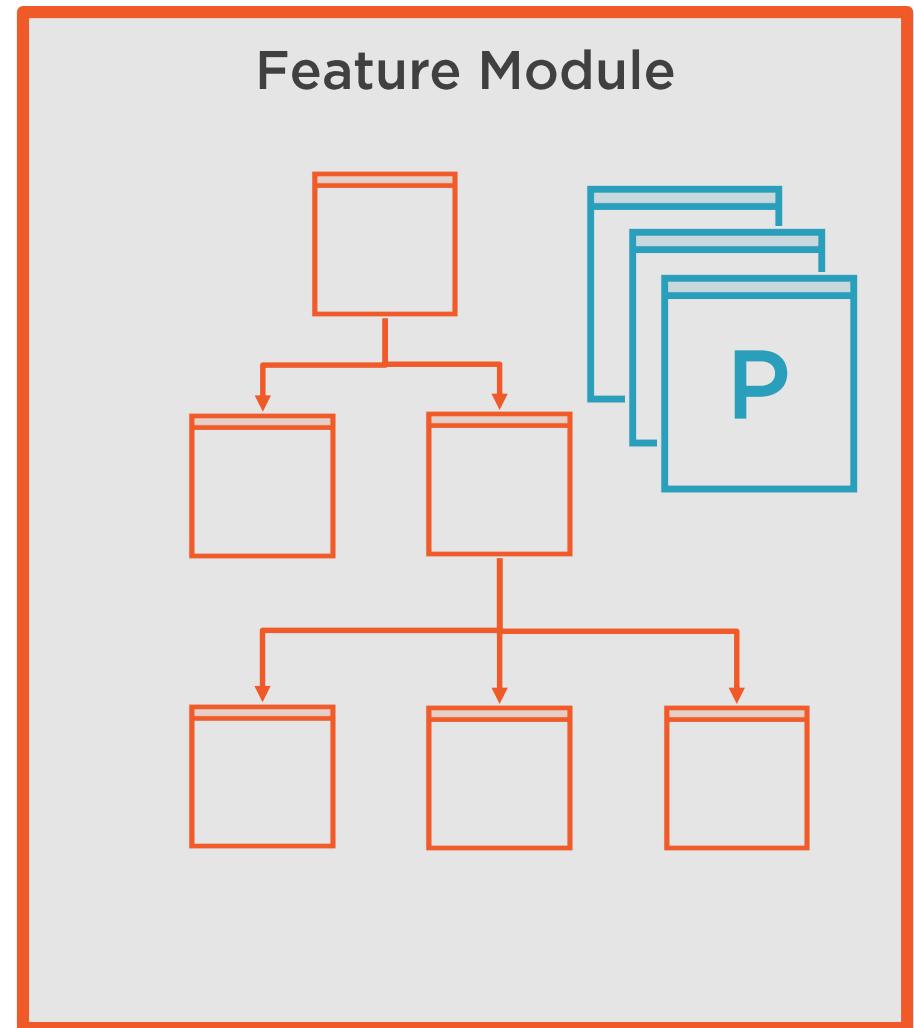
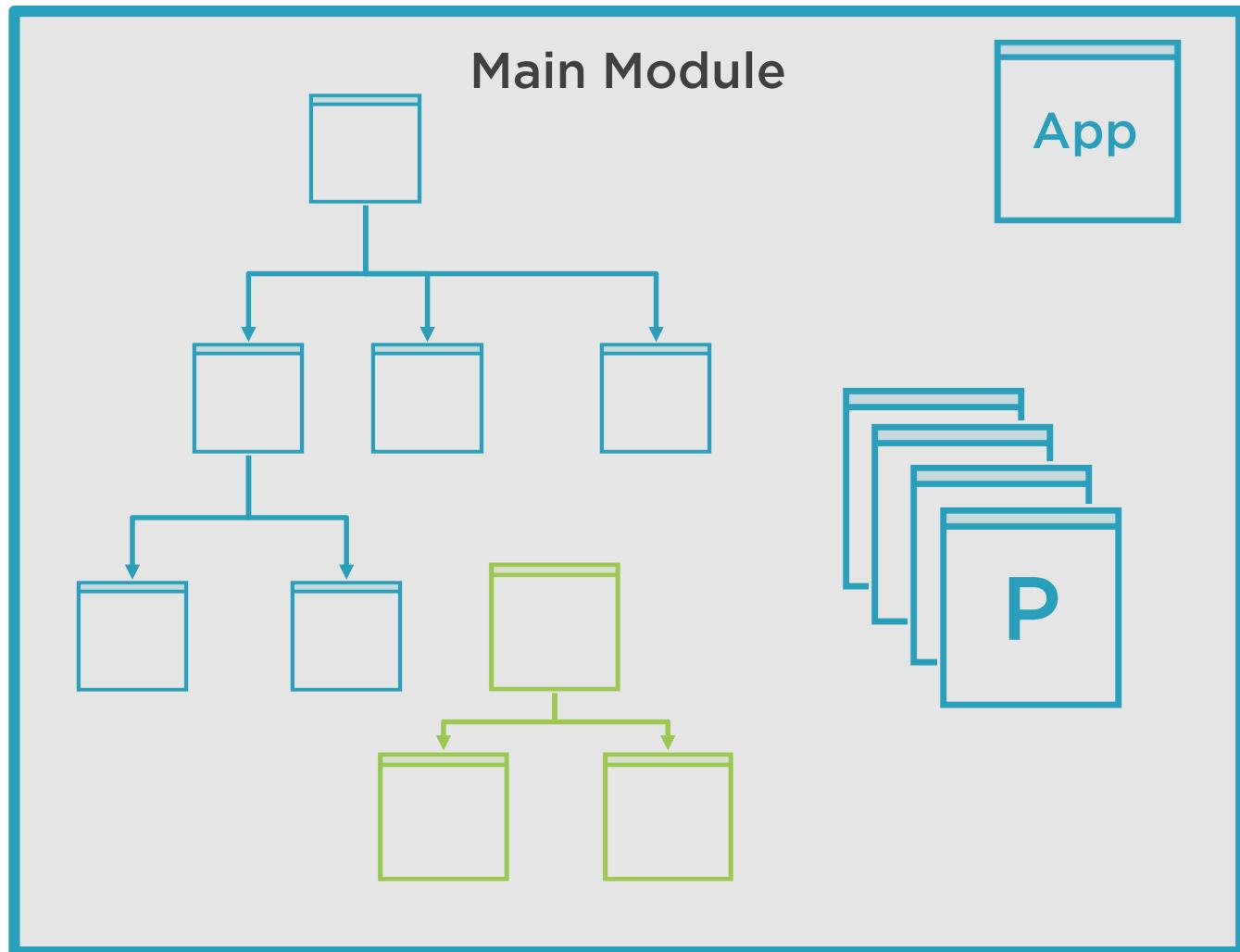
# Angular 2 Component Hierarchy



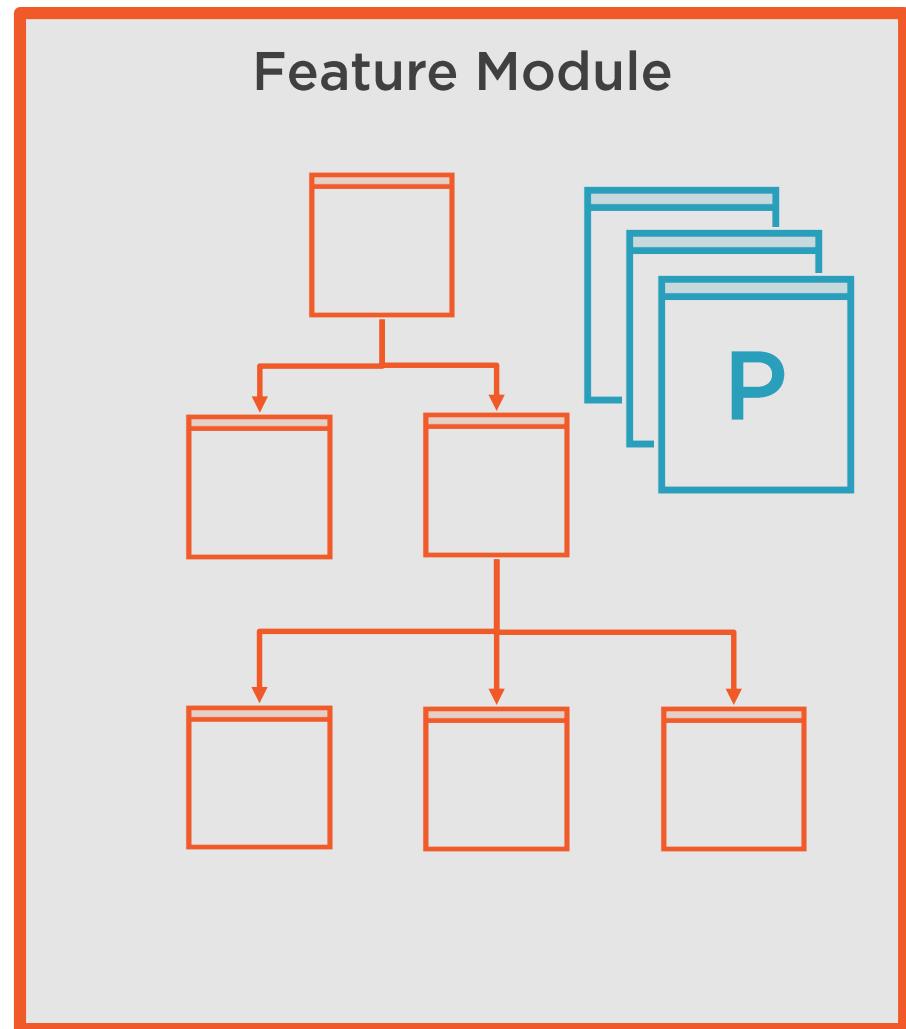
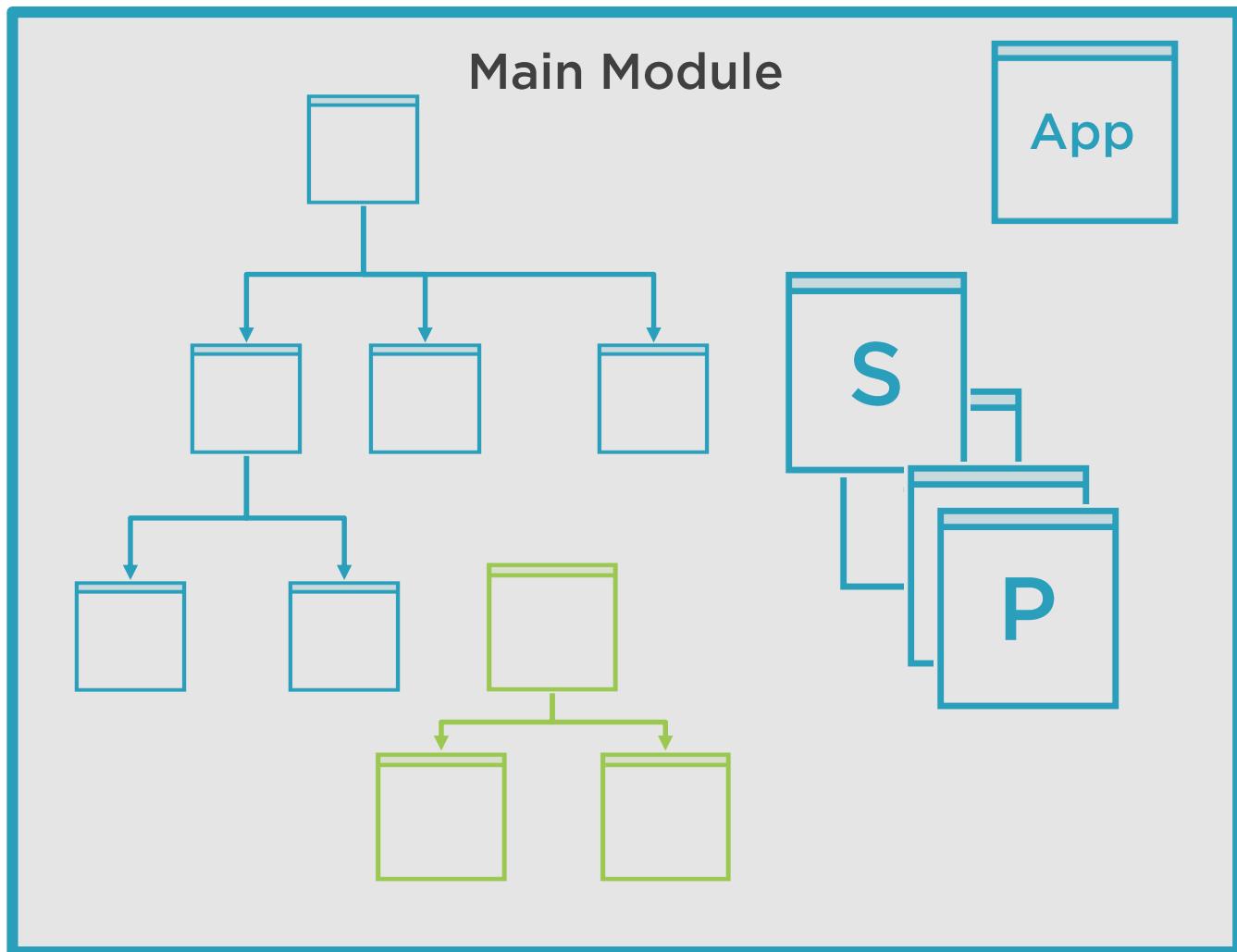
# Angular 2 Component Hierarchy



# Angular 2 Component Hierarchy



# Angular 2 Component Hierarchy



# Creating and Communicating Between Angular Components

---



**Jim Cooper**

SOFTWARE CRAFTSMAN

@jimthecoop



# Agenda

- Inline Templates and Data-Binding
- Using External Templates
- Inter-Component Communication
- Styling Components
- Style Encapsulation



# Exploring the Angular Template Syntax

---



**Jim Cooper**

SOFTWARE CRAFTSMAN

@jimthecoop



# Agenda

- Interpolation and Expressions
- Event Bindings and Statements
- Repeating Data With \*ngFor
- Removing Elements With  
\*ngIf and \*ngSwitch
- Hiding Elements
- Adding Classes and Styles



...

```
@Component({  
  template: `  
    <h2>{{user.name}}</h2>  
    <img [src]="user.imageUrl"/>  
    <button (click)="doSomething()"></button>  
  `  
})  
export class ProfileComponent {  
  user = {name: 'John Doe', imageUrl: 'doe.com/profile.jpg'}  
  
  doSomething() {  
  }  
}
```

# Interpolation, Expressions, Events, and Statements



...

```
@Component({  
  template: `  
    <h2>{{user.name}}</h2>  
    <img [src]="user.imageUrl"/>  
    <button (click)="doSomething()"></button>  
  `})
```

```
export class ProfileComponent {  
  user = {name: 'John Doe', imageUrl: 'doe.com/profile.jpg'}
```

```
  doSomething() {  
  }  
}
```

Interpolation

Property Binding

# Interpolation, Expressions, Events, and Statements



```
...  
@Component({  
  template: `  
    <h2>{{u getIt()}}</h2>  
    <img [src]="user.imageUrl"/>  
    <button (click)="doSomething()"></button>`  
})  
export class ProfileComponent {  
  user = {name: 'John Doe', imageUrl: 'doe.com/profile.jpg'}  
  
  doSomething() {  
  }  
}
```

# Interpolation, Expressions, Events, and Statements



# Expression Restrictions



Assignments (=, +=, ++, etc)

new Keyword

Expression Chaining With ;

Global Namespace



# Expression Recommendations



No Side-Effects

Fast

Simple

Idempotent



```
...  
@Component({  
  template: `  
    <h2>{{user.name}}</h2>  
    <img [src]="user.imageUrl"/>  
    <button (click)="doSomething()"></button>  
  `)  
})  
export class ProfileComponent {  
  user = {name: 'John Doe', imageUrl: 'doe.com/profile.jpg'}  
  
  doSomething() {  
  }  
}
```

# Interpolation, Expressions, Events, and Statements



# Statement Restrictions



Assignments (=, +=, ++, etc)

new Keyword

Expression Chaining With ;

Global Namespace



# Statement Restrictions



Assignments Except = ( $+=$ ,  $++$ , etc)

new Keyword

Expression Chaining With ;

Global Namespace



# Statement Restrictions



Assignments Except = (+=, ++, etc)

new Keyword

~~Expression Chaining With ;~~

Global Namespace



# Statement Restrictions



Assignments Except = (+=, ++, etc)

new Keyword

Global Namespace



# Statement Restrictions



No Side-Effects

Fast

Simple

Idempotent



# Statement Restrictions



~~No Side Effects~~

Fast

Simple

Idempotent



# Statement Restrictions



~~No Side Effects~~

Fast

Simple

~~Idempotent~~



# Statement Restrictions



~~No Side Effects~~

~~Fast~~

Simple

~~Idempotent~~



# Statement Restrictions



Simple



# Creating Reusable Angular Services

---



**Jim Cooper**

SOFTWARE CRAFTSMAN

@jimthecoop



# Agenda

Why Services are Necessary  
Dependency Injection  
Creating Services  
Wrapping 3<sup>rd</sup>-Party Libraries



# Why are Services Necessary?







Jim

ng Events

localhost:8808

ngEvents All Events Create Event Events ▾ Search Sessions Search Welcome John

# Upcoming Angular 2 Events

## Angular Connect

Date: 9/26/2036  
Time: 10:00 am (Late Start)  
Price: \$599.99  
Location: 1057 DT London, England

## ng-nl

Date: 4/15/2037  
Time: 9:00 am (Normal Start)  
Price: \$950  
Online URL: <http://ng-nl.org/>

## ng-conf 2037

Date: 5/4/2037  
Time: 9:00 am (Normal Start)  
Price: \$759  
Location: The Palatial America Hotel Salt Lake City, USA

## UN Angular Summit

Date: 6/10/2037  
Time: 8:00 am (Early Start)  
Price: \$800  
Location: The UN Angular Center New York, USA

events-list.component.ts - ng2-fundamentals - Visual Studio Code

File Edit View Go Help

EXPLORER event-thumbnail.component.ts events-list.component.ts

OPEN EDITORS event-thumbnail.component.ts app... events-list.component.ts app\events

NG2-FUNDAMENTALS .vscode settings.json app assets events event-thumbnail.component.ts events-list.component.ts nav app.module.ts events-app.component.ts main.ts node\_modules typings .gitignore favicon.ico index.html package.json readme.MD styles.css systemjs.config.js tsconfig.json typings.json

```
3 @Component({
4   selector: 'events-list',
5   template: `
6     <div>
7       <h1>Upcoming Angular 2 Events</h1>
8       <hr/>
9       <div class="row">
10         <div *ngFor="let event of events" class="col-md-5">
11           <event-thumbnail [event]="event"></event-thumbnail>
12         </div>
13       </div>
14     </div>
15   `
16 })
17 export class EventsListComponent {
18   events = [
19     {
20       id: 1,
21       name: 'Angular Connect',
22       date: '9/26/2036',
23       time: '10:00 am',
24       price: 599.99
25     }
26   ]
27 }
```



# Dependency Injection



```
@Component({  
...  
})  
class EventsListComponent {  
  events  
  let eventsService = new EventsService()  
  events = eventsService.getEvents()  
}  
}
```

## Dependency Injection



```
@Component({  
...  
})  
class EventsListComponent {  
  events  
  constructor(private eventService:EventService) {  
    events = eventService.getEvents()  
  }  
}  
}
```

## Dependency Injection



# Routing and Navigating Pages

---



**Jim Cooper**  
SOFTWARE CRAFTSMAN  
@jimthecoop



# Agenda

Why Routing is Necessary

Define Routes for Pages

Link to Routes

Navigate From Code

Route Guards

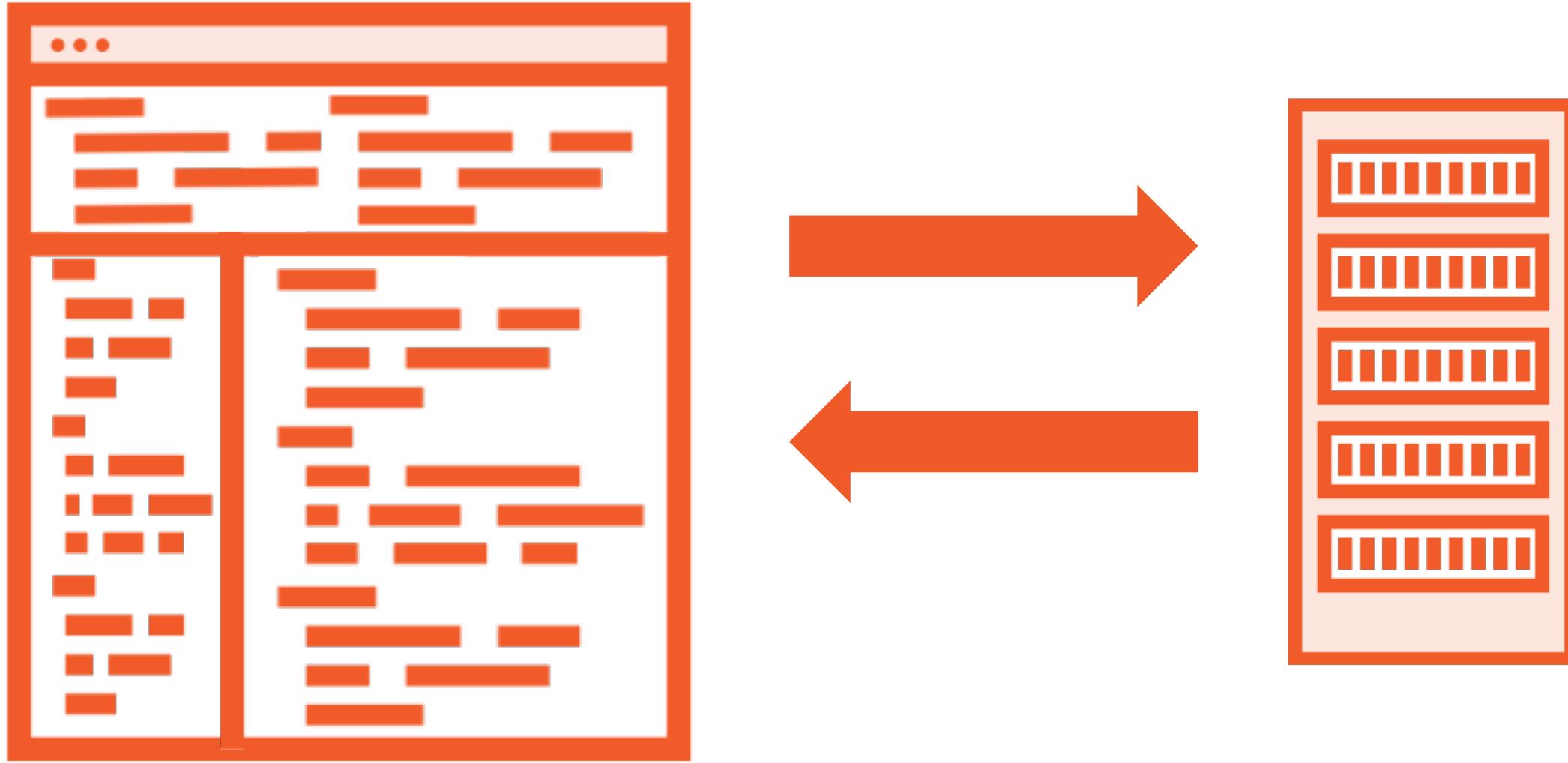
Resolve

Route-Based Link Styling

Lazy Loading



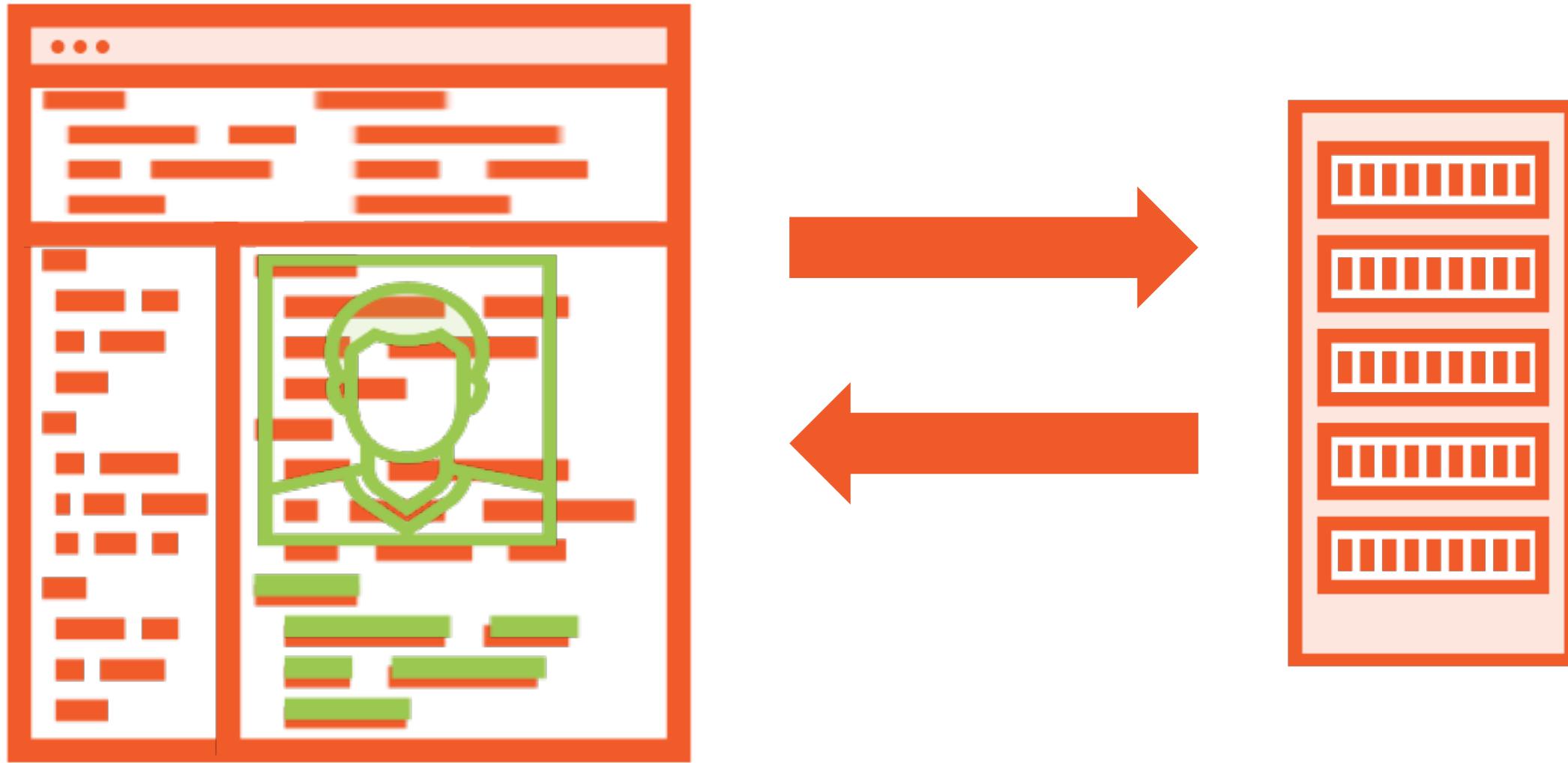
# Why Routing is Necessary



# Why Routing is Necessary



# Why Routing is Necessary



# Summary



**Why Routing is Necessary**

**Define Routes for Pages**

**Link to Routes**

**Navigate From Code**

**Route Guards**

**Resolve**

**Route-Based Link Styling**

**Lazy Loading**



# Collecting Data with Angular Forms and Validation

---



**Jim Cooper**

SOFTWARE CRAFTSMAN

@jimthecoop



# Agenda

- Using Data Models for Type Safety
- Template-based Forms
- Model-driven Forms
- Two-way Data Bindings
- Custom Validators



# Communicating Between Components

---



**Jim Cooper**  
SOFTWARE CRAFTSMAN  
[@jimthecoop](https://twitter.com/jimthecoop)



# Summary



**Inter-Component Communication**

**Communicating with Child Components**

**Communicating with Parent Components**



# Reusing Components with Content Projection

---



**Joe Eames**

WEB DEVELOPER

@josepheames [www.joeeames.me](http://www.joeeames.me)



# Agenda

**Content Projection**

**Multiple Slot Content Projection**



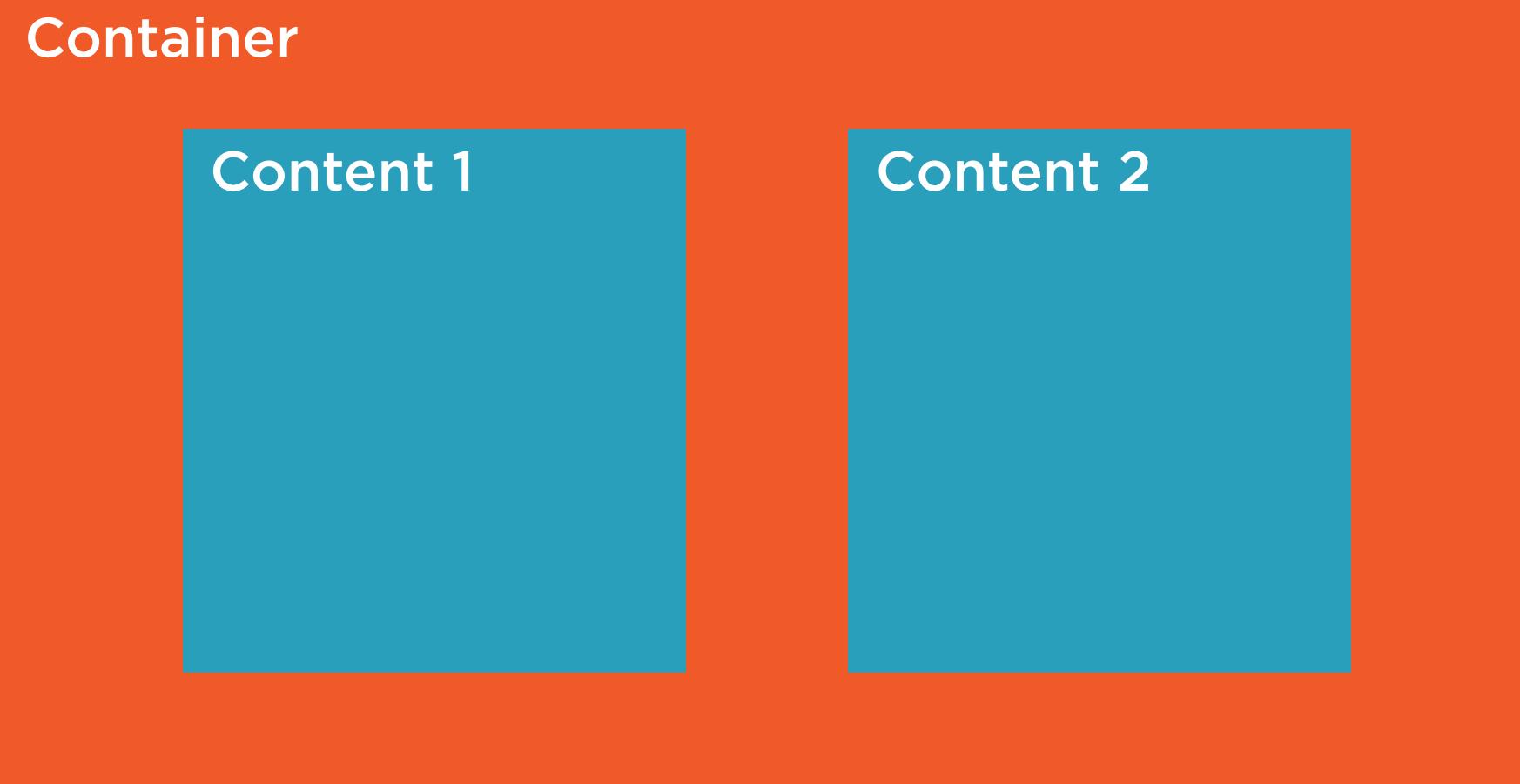
# Content Projection

Container

Content



# Multiple Slot Content Projection



# Summary



**Reusable Components  
Content Projection**



# Displaying Data with Pipes

---



**Joe Eames**

WEB DEVELOPER

@josepheames [www.joeeames.me](http://www.joeeames.me)



# Agenda

**Built-In Pipes**  
**Custom Pipes**  
**Filtering Data**  
**Sorting Data**



# Pipes vs Filters

**A1 Filters**

Formatting

Sorting

Filtering

**Pipes**

Formatting

~~Sorting~~

~~Filtering~~



# Filtering & Sorting Data - Overview

---



# Primitive Identity

```
var a = 3;
```

```
var b = 3;
```

```
a === b TRUE
```

```
var c = a;
```

```
a === c TRUE
```

```
a = 4;
```

```
a === c FALSE
```

a 3

b 3

c 3



# Object Identity

```
var a = {};
```

```
var b = {};
```

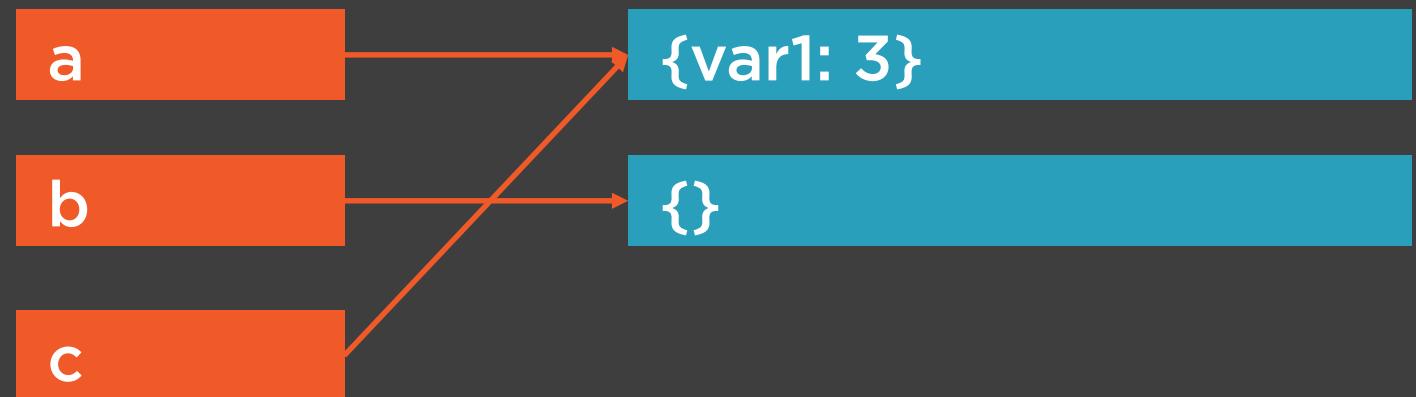
```
a === b FALSE
```

```
var c = a;
```

```
a === c TRUE
```

```
a.var1 = 3;
```

```
a === c TRUE
```



# Object Identity

```
var a = {};
```

```
var b = {};
```

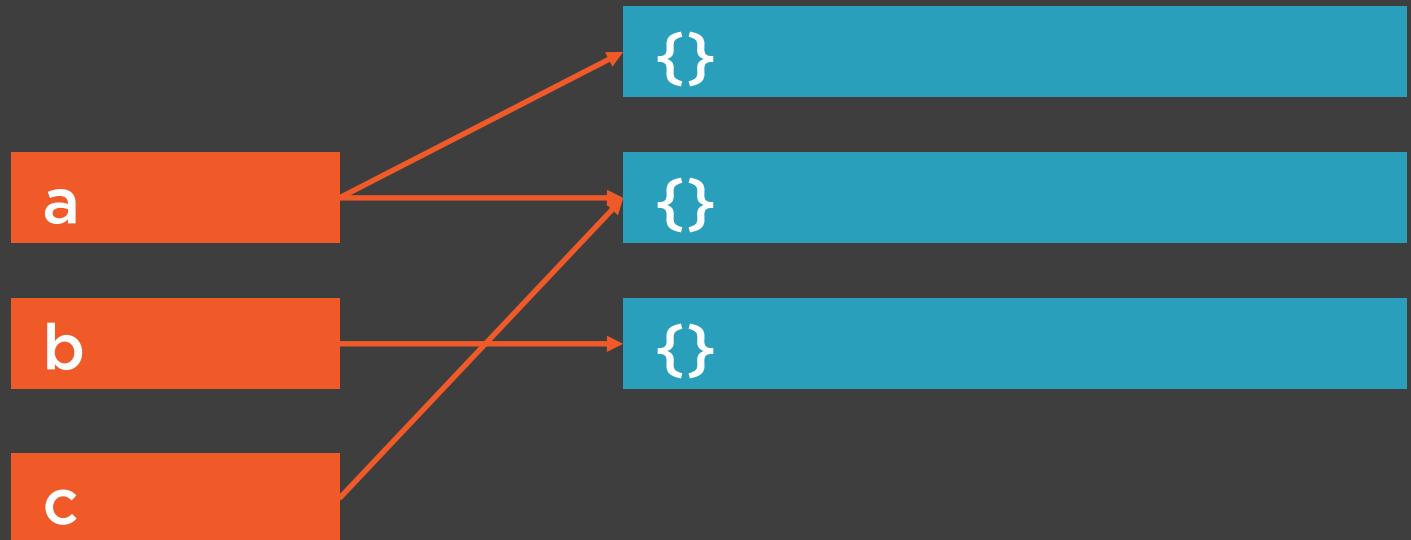
```
a === b FALSE
```

```
var c = a;
```

```
a === c TRUE
```

```
a = {};
```

```
a === c FALSE
```



# Mutability

```
[  
  { name: 'Dave' },  
  { name: 'John' },  
]
```



# Mutability

```
[  
{ name: 'Dave' },  
]  
|
```



# Mutability

```
[  
  { name: 'Dave' },  
  { name: 'Dan' },  
]
```



# Identity, Mutability & Pipes

```
[  
  { name: 'John' },  
  { name: 'Dave' },  
]  
  
[  
  { name: 'Dave' },  
  { name: 'John' },  
]
```

Pipe →



# Identity, Mutability & Pipes

```
[  
  { name: 'John' },  
  { name: 'Ralph' },  
]  
  
[  
  { name: 'Dave' },  
  { name: 'John' },  
]
```

Pipe →



# Another Option

## Impure Pipes

Runs on **EVERY** change  
detection cycle

Same as Angular 1 Filters



# Recommended Filtering & Sorting

**Do It Yourself**

**Only Updated When Source  
Data Changes**

**More efficient**



# Summary



Pipes  
Sorting & Filtering



# Understanding Angular's Dependency Injection

---



**Joe Eames**

WEB DEVELOPER

@josepheames [www.joeeames.me](http://www.joeeames.me)



# Agenda

**Dependency Injection Overview**  
**InjectionToken**  
**@Inject()**  
**Alternative Provider Methods**



# Summary



**InjectionToken & @Inject()**  
**useClass**  
**useValue**  
**useExisting**  
**useFactory**



# Creating Directives and Advanced Components in Angular

---



**Joe Eames**

WEB DEVELOPER

@josepheames [www.joeeames.me](http://www.joeeames.me)



# Agenda

## Modal Component

- Routing to the Same Component
- DOM Manipulation
- Using the `@ViewChild` Decorator

## Directive



# Summary



**Creating Advanced Components**  
**Creating Directives**



# More Components & Custom Validators

---



**Joe Eames**

WEB DEVELOPER

@josepheames [www.joeeames.me](http://www.joeeames.me)



# Agenda

**Voting Component**  
**Custom Validator Directive**



# Summary



## Voting Component Custom Validator



# Communicating with the Server Using HTTP, Observables, and Rx

---



**Joe Eames**

WEB DEVELOPER

@josepheames [www.joeeames.me](http://www.joeeames.me)



# Agenda

**Introduction to RxJS**

**Moving Data Storage to the Server**

- Events Service
- Voter Service

**Authentication**



# HTTP Communication



# Callbacks

```
Server.request(requestData, function(responseData) {  
    // asynchronously handle the data.  
});  
  
// this will execute before the callback  
doMoreThings()
```



# Promises

```
var promise = http.get(url, data);  
promise.then(function(responseData) {  
    // handle response  
})  
  
// this will execute before the then function  
doMoreThings()  
return promise;
```



# Observables

```
var obs = http.get(url, data);  
// manipulate the observable if desired  
obs.subscribe(function(responseData) {  
    // handle response  
});  
doMoreThings()  
return obs;
```



# Promises vs Observables

## Promises

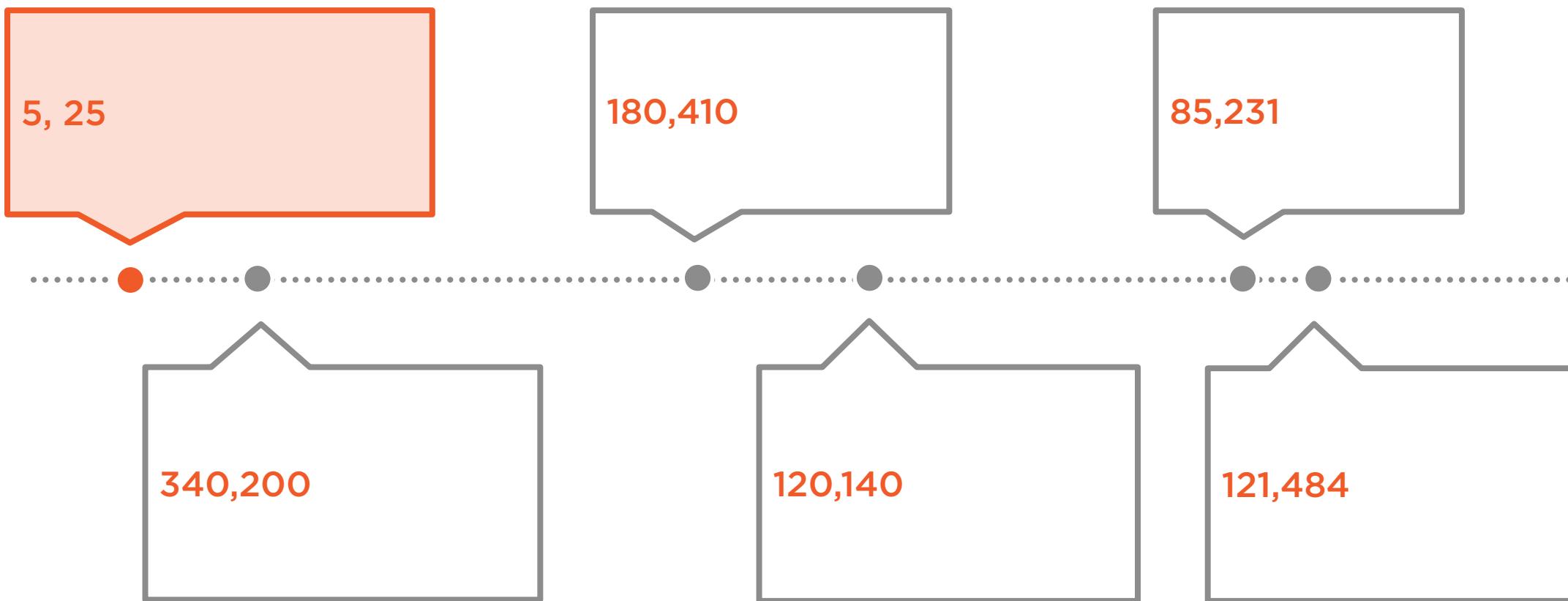
Represent a single value in the future

## Observables

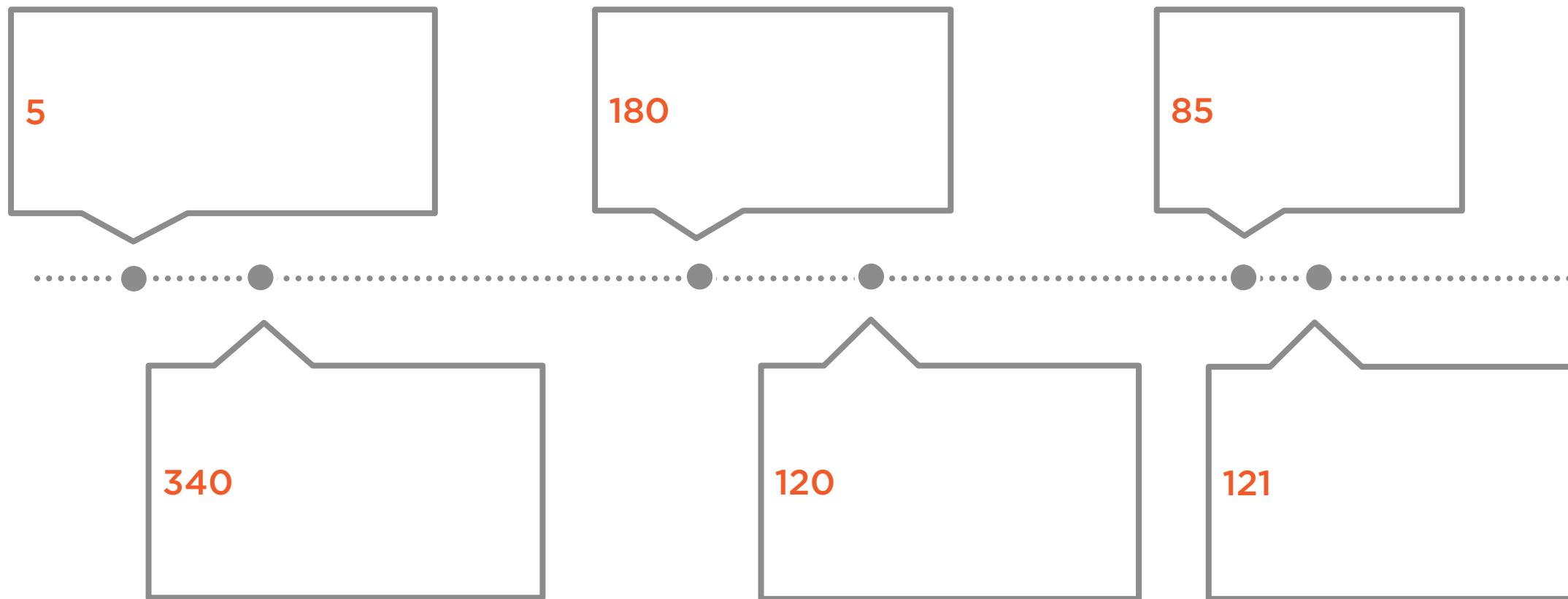
Represent 0 or more values now or in the future



# Timeline of Events



# Timeline of Events



## Other Observable Features

Can Be Synchronous

Improved Error Handling

Can be Closed Independently of Returning a Value

Can Deal with Time

Advanced Operations

- Mathematical Aggregation
- Buffering
- Debounce
- Distinct
- Filtering
- Combining Observables
- Retry



# HTTP Communication



One Key  
Feature

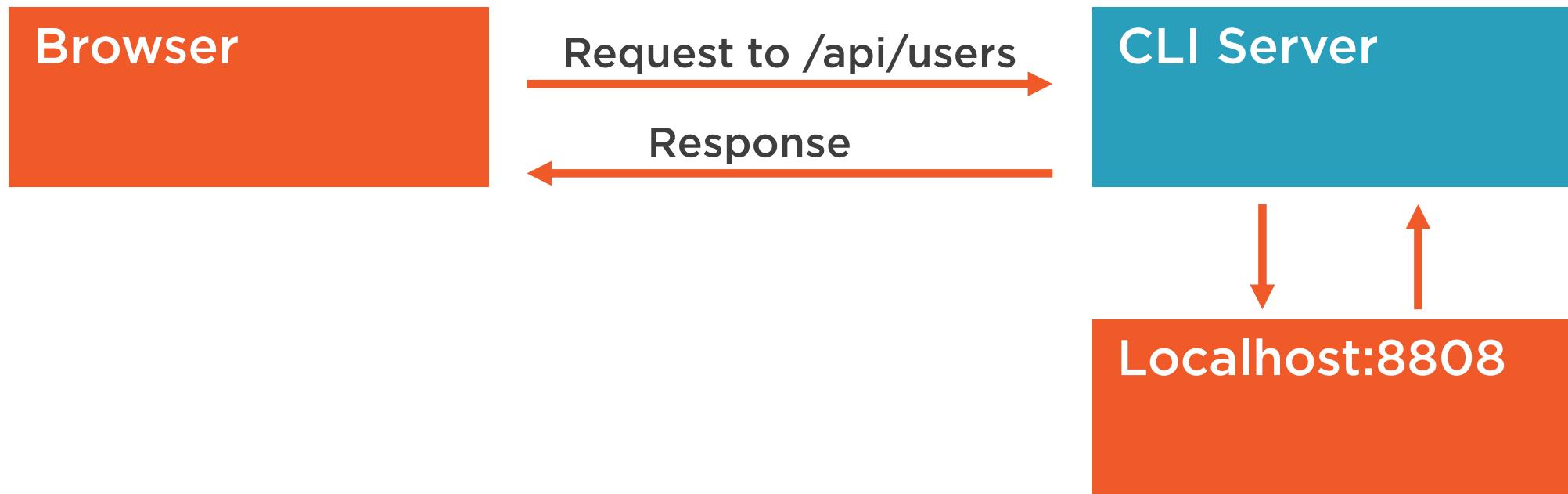
**toPromise()**



Choose What's Best For  
You



# Server Arrangement



# Summary



**Observables  
Data Storage on Server  
Authentication**



# Unit Testing Your Angular Code

---



**Joe Eames**

WEB DEVELOPER

@josepheames [www.joeeames.me](http://www.joeeames.me)



# Agenda

Introduction

Karma

Services

Isolated Component Tests



# Unit Test

A test of a single “unit” of code.



# Unit Tests vs End to End Tests

## Unit Test

Fast

Involve Isolated Pieces of  
Code

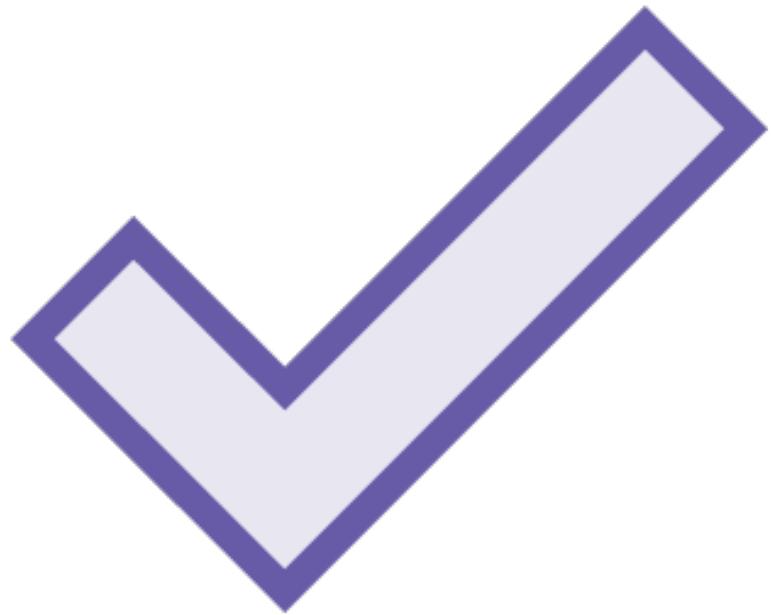
## End to End Test

Slow

Exercises the Entire System



# Good Unit Tests



**Fast**

**Cheap to Write**

**Single State Change**

**Assert 1 Thing**

**Doesn't Cross Process Boundaries**

**Reliable**



# Unit Test Structure

**AAA**

Arrange

Act

Assert

**Damp**

Can Have Some Duplication

Tells the Story



```
var user = new User("Sally");
```

```
user.friends = ["Ralph"]
```

```
user.addFriend("John");
```

```
expect(user.friends.length)
```

```
.toBe(2);
```

◀ Arrange

◀ Act

◀ Assert



```
var user = new User("Sally");
```

```
user.friends = ["Ralph"]
```

```
user.addFriend("Ralph");
```

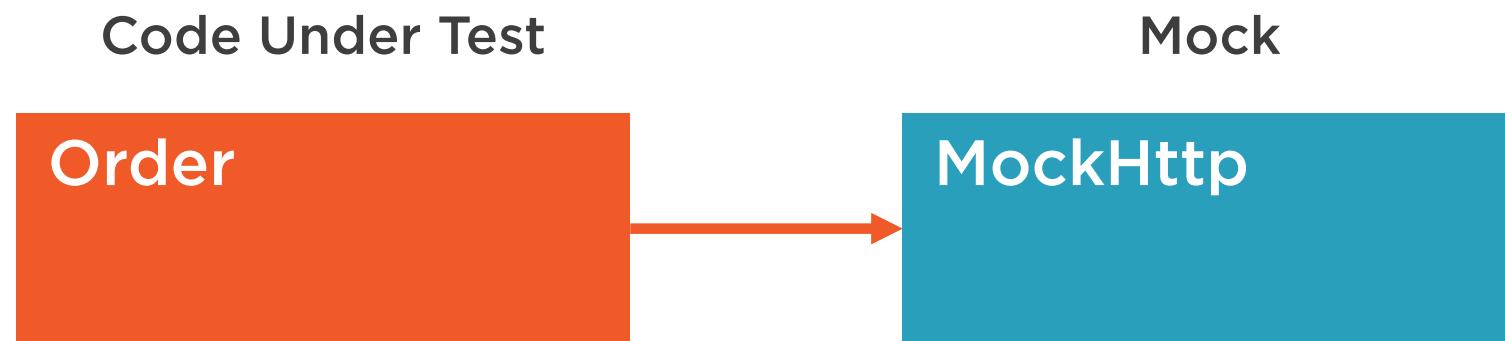
```
expect(user.friends.length)
```

```
.toBe(1);
```

◀ Remove Duplication?



# Mocking



# Jasmine

`describe()`

`beforeEach()`

`it()`

`expect()`

**Matchers**

- `toBe()`
- `toContain()`
- `toBeDefined()`



Karma

**Runs Tests in Browser**  
**Multiple Browsers**  
**Reports Test Run Results**



# Isolated vs Integrated Tests

## Isolated Tests

**Test Class Only – No Template**

**Constructed in Test**

**Simple**

**Best for Services & Pipes**

**Appropriate for Components & Directives**

## Integrated Tests

**Test Class & Template**

**Constructed by Framework**

**Complex**

**Mainly Used for Components & Directives**

**Sometimes Used for Services**

**Deep or Shallow**



# Summary



**Karma & Jasmine  
Services  
Mocking  
Isolated Component Tests**



# Testing Angular Components with Integrated Tests

---



**Joe Eames**

WEB DEVELOPER

@josepheames [www.joeeames.me](http://www.joeeames.me)



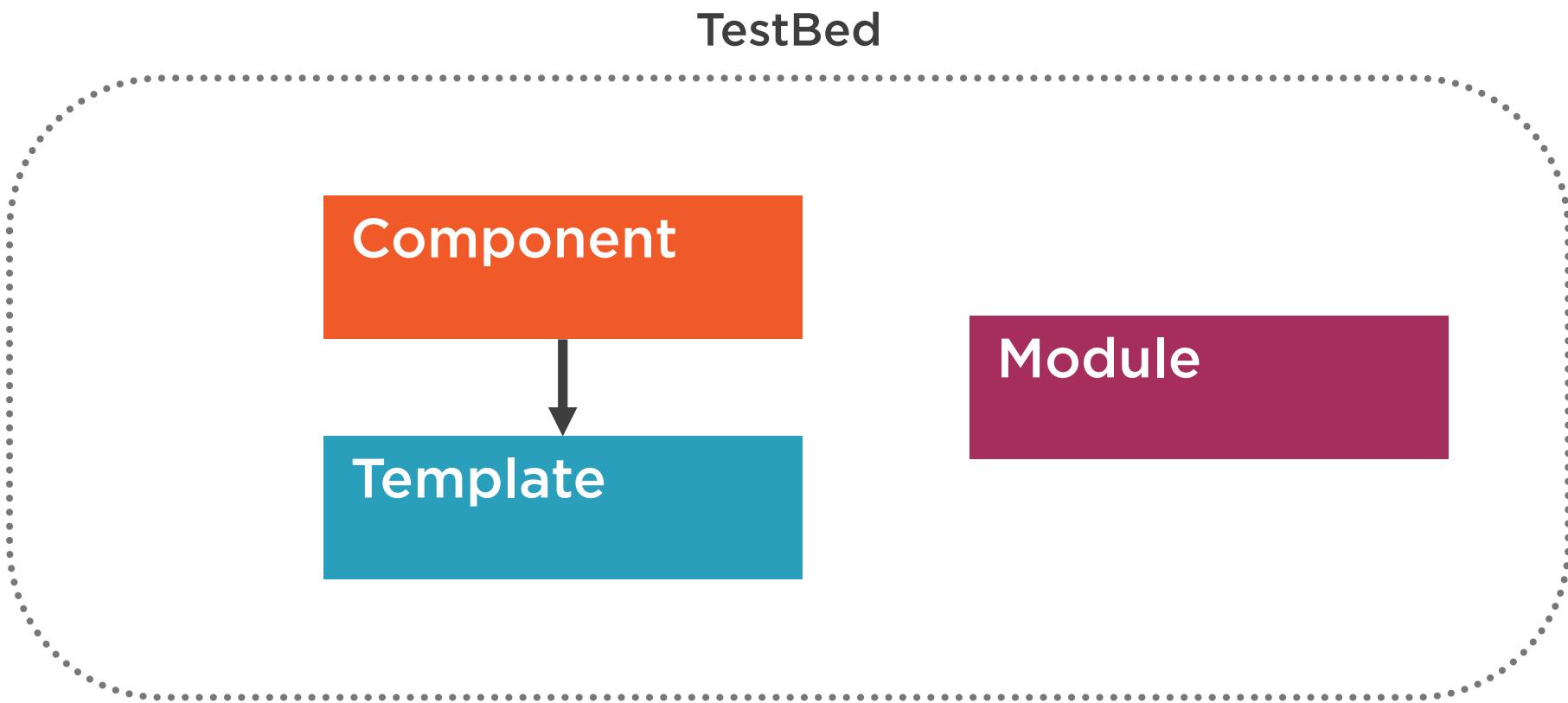
# Agenda



**Setup**  
**Testing Components**  
**Mocking Services**  
**Shallow vs Deep Tests**



# Integration Tests



# Value of Integration Tests

**Complex**

**Fix Functionality in Place**

- More Difficult to Change Code
- More Difficult to Introduce Bugs



# Summary



**Integrated Component Tests**

**Deep**

**Shallow**



# Taking an Angular App to Production

---



**Joe Eames**

WEB DEVELOPER

@josepheames [www.joeeames.me](http://www.joeeames.me)



# Agenda

## Linting

- VS Code
- Command line

## Creating Builds

- Build flags & the AOT

## Deploying

## Optimistic module downloading



# Linting

---



# Linting

Finding the “lint” in your code.



# What Is Linting Really?

**Pointing out and fixing potential problems**

**Mostly coding style changes**

- Missing semicolons
- Double vs. single quotes
- Long lines
- Etc.



# Linting in Action

```
var x = 3;  
console.log("I have " + x + ' problems')
```

Should be a  
const

Should be  
single quote

Missing  
semicolon



# Benefits of Linting

- Consistency across your team**
- Catching potential errors**
- Automatic error fixes**
- CI or commit hooks**



# Other Considerations

**JavaScript vs. TypeScript**  
**Prettier**



# Going to Production - Overview

---



# Building an Angular Application

ng build



## ng build Command

**Produces a deliverable code package**

**Production optimizations**

- Development mode off
- Bundling
- Minification
- Tree shaking
- Dead code elimination
- Asset inlining
- Executes AOT



# Minification

Source

```
export class orderService {  
    addOrder(order) {  
        ...  
    }  
  
    findOrder(id) {  
        ...  
    }  
}
```

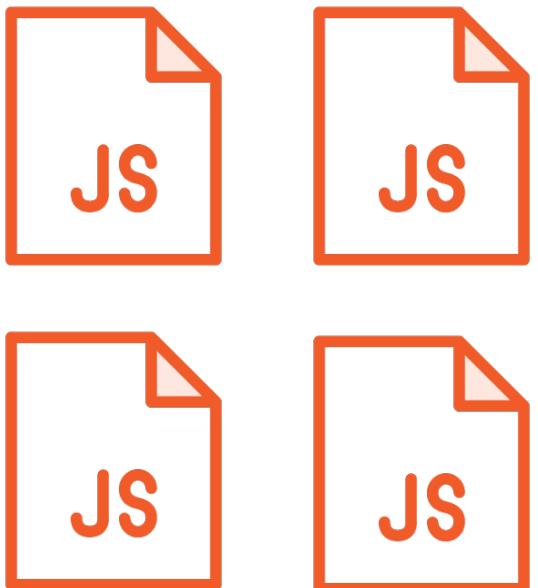
Service

```
export class orderService{add  
Order(order){...}findOrder(id){  
...}}
```

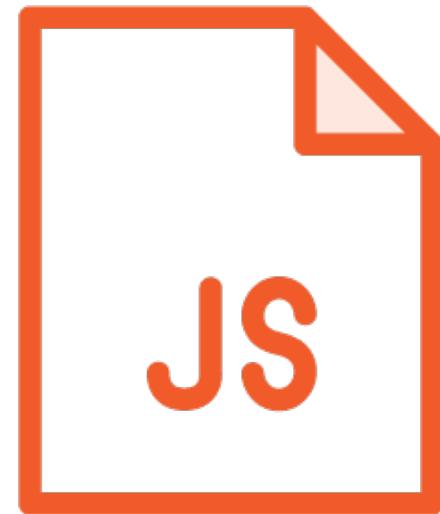


# Bundling

Development



Production



# Tree Shaking

## Calling Code

```
...  
userService.getUsers()  
...
```

## Service

```
export class UserService {  
  getUsers() {  
    ...  
  }  
  
  getUser(id) {  
    ...  
  }  
}
```



# Angular's Compiler

---



# Compiler Benefits

**Faster rendering**

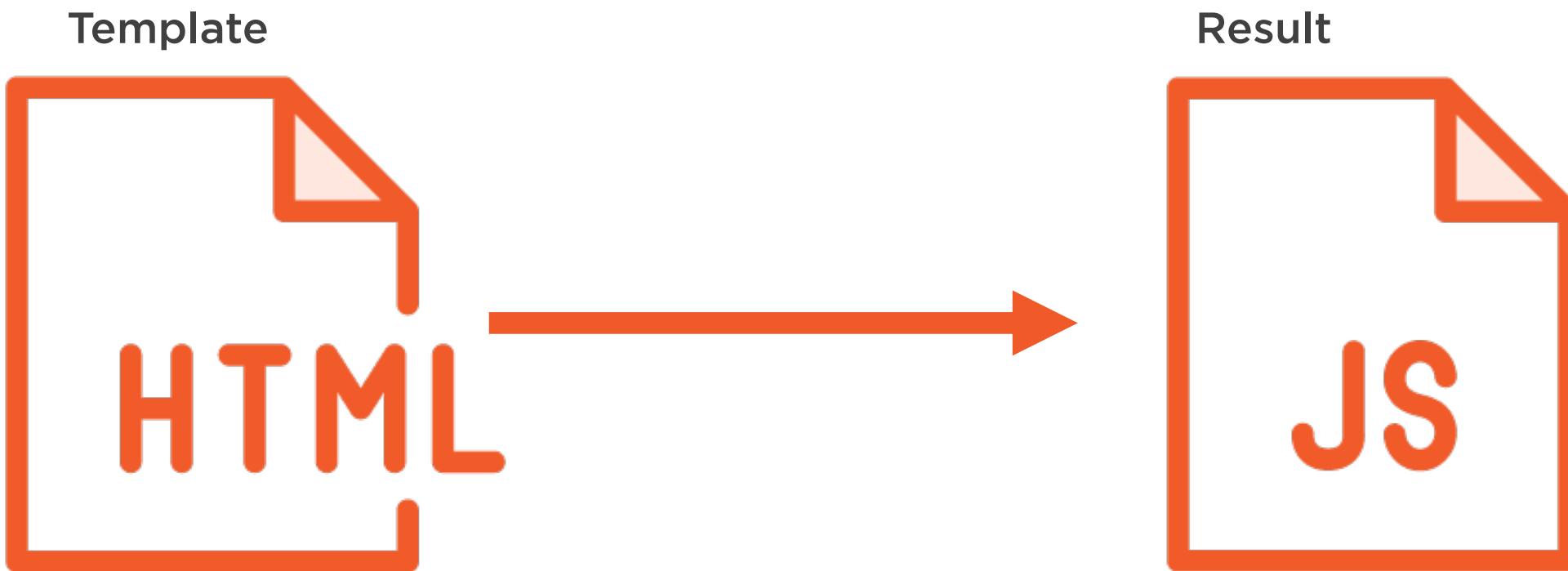
**Smaller Angular framework download**

**Detect template errors**

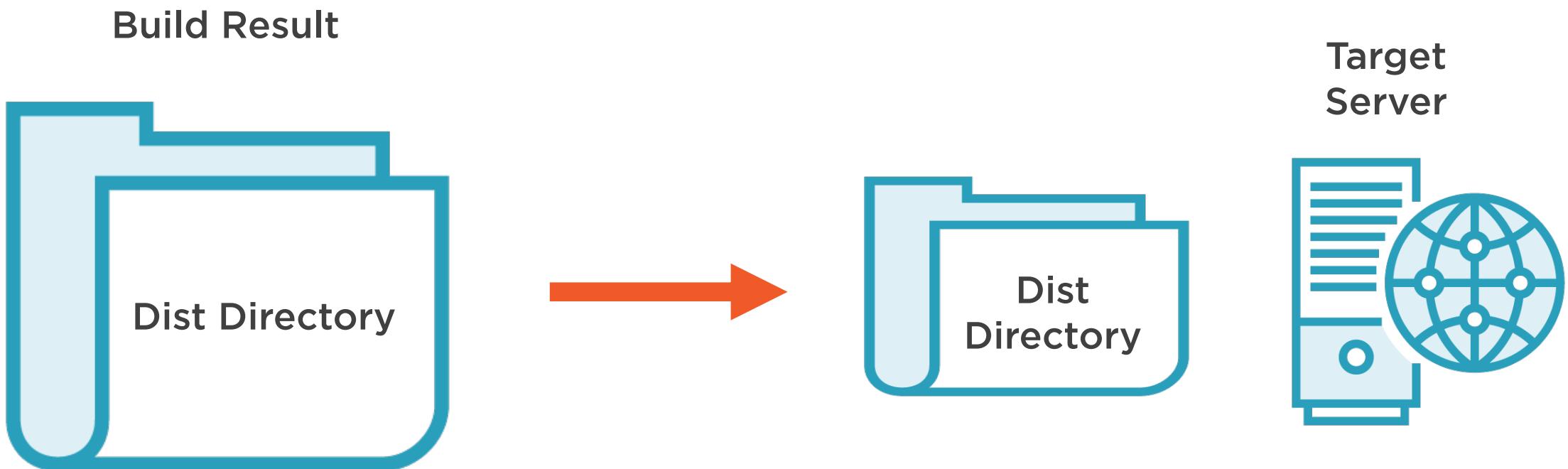
**Better security**



# Compiling Templates



# Simple Copy Deployment



# Summary



**Linting vs Prettier**

**Deployment is simple**

**Optimistic bundle download**



# Thank You



# Angular: First Look

---

ANGULAR 2 IN ACTION



**John Papa**

PRINCIPAL ARCHITECT

@john\_papa

[www.johnpapa.net](http://www.johnpapa.net)



# First Look at Angular 2

The big picture of Angular 2. How you can write apps with it and how your Angular 1 skills translate to it



# Streamlined Learning Experience

**Code here**

```
import { Component } from 'angular2/core';
class Character {
  constructor(public id: number, public name: string, public side: string)
}
class Vehicle {
  constructor(public id: number, public name: string) {}
}
@Component({
  selector: 'story-character-solved',
  templateUrl: './app/solution/character-solved.component.html',
  styleUrls: ['./app/solution/character-solved.component.css']
})
export class CharacterSolvedComponent {
  character: Character;
  color = '';
  isSelected = false;
  selectLabel = 'Select a Character';
  vehicles = [
    new Vehicle(1, 'Slave 1'),
    new Vehicle(2, 'Imperial Star Destroyer'),
    new Vehicle(3, 'Escape Pod')
  ];
  constructor() {
    this.character = new Character(100, 'Boba Fett', 'dark', 'assets/man.base64.png');
  }
  select(name: string) {
    let msg = `You selected ${name}`;
    console.log(msg);
    this.isSelected = !this.isSelected;
  }
}
```

**See results immediately**

Plunker v0.8.21

Save New Stop

FILES

- api/characters.json
- api/vehicles.json
- app/app.component.ts
- app/character.component.css
- app/character.component.html
- app/character.component.ts

app/solution/character-solved.component.html

app/solution/character-solved.component.ts

assets/man.base64.png

assets/material.deep\_orange-pink.min.css

index.html

license.md

typings.json

New file

PLUNK

Description:

Angular 2 Example - Data Binding

Tags:

angular2 example first look

SWITCH TO STARTER

BOBA FETT

BOBA FETT

boba fett

Name

Boba Fett

Color

#EEE

SELECT

Your character Boba Fett has these vehicles:

- Slave 1
- Imperial Star Destroyer

Live Preview

johnpapa

# TypeScript

We can use ES5, ES2016, or TypeScript to write Angular 2. We will write all code samples with TypeScript.



# JavaScript Basics

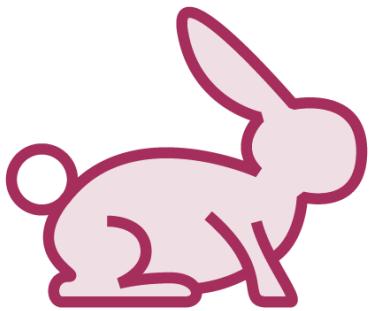


# Why Angular 2?

---



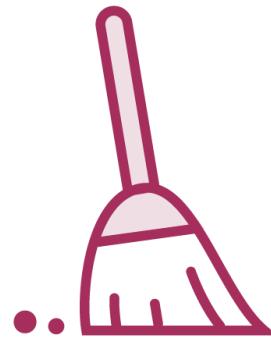
# Angular 2



Fast



Powerful



Clean



Easy





iOS



Internet  
Explorer

9, 10, 11 and Edge



4.1+



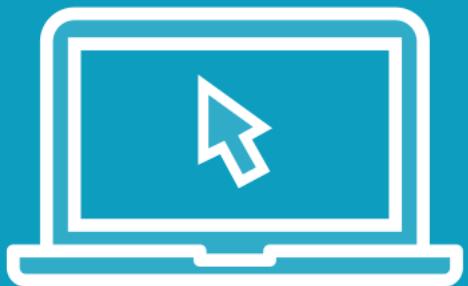
# Google AdWords

---

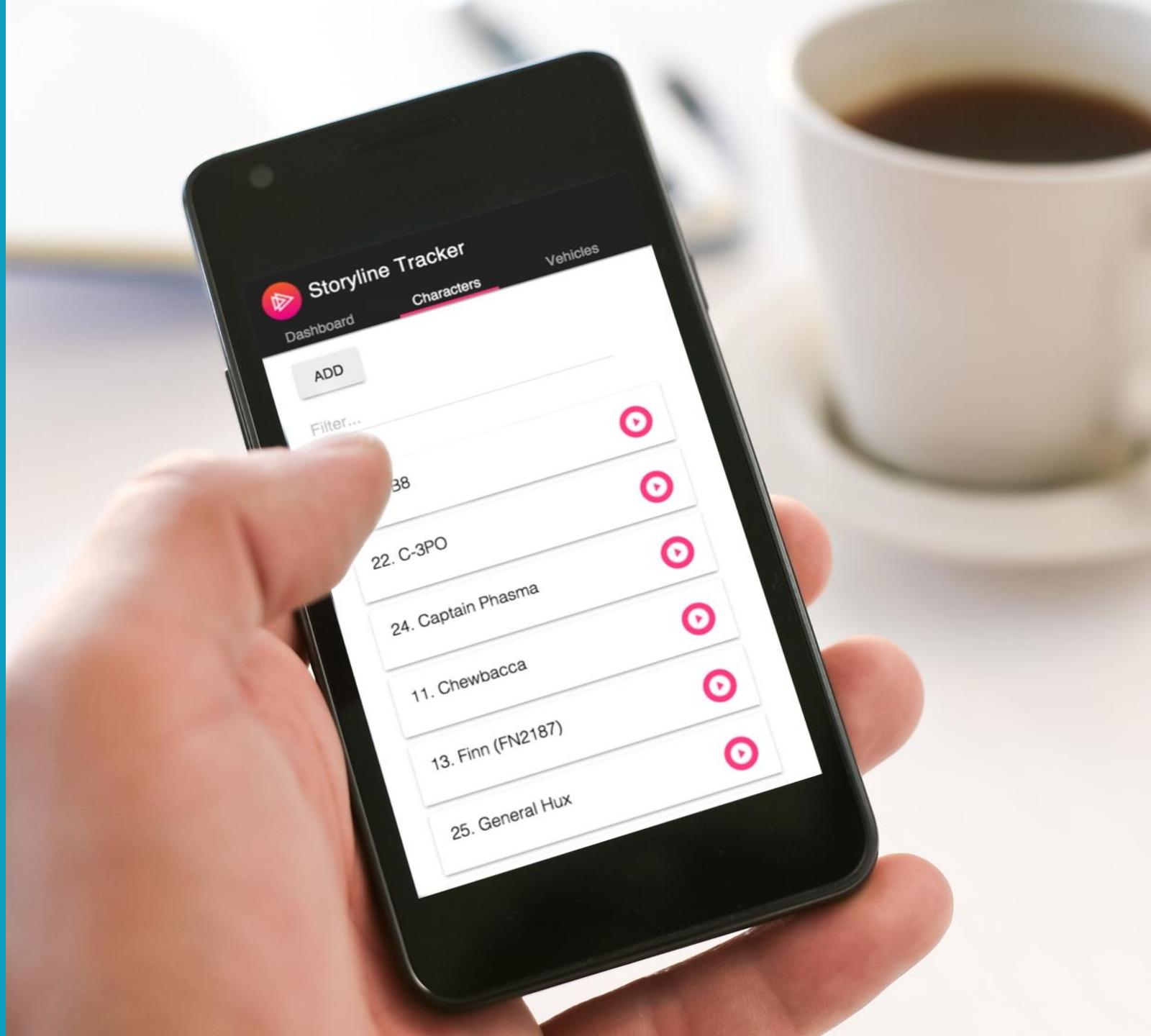
100s of Developers, Millions of lines of code



# Demo



## Storyline Tracker



# Running the Sample Code

---



# Live Samples on the Web

Code along and run all samples live on the web.

No setup required.

<http://jpapa.me/a2firstlook>



# Beyond the First Look

Follow the QuickStart and tutorial on <http://angular.io>

Getting Started and Fundamentals courses on Pluralsight



# Angular 2 Architecture, What's New and What's Different

---



**John Papa**

PRINCIPAL ARCHITECT

@john\_papa

[www.johnpapa.net](http://www.johnpapa.net)



# Angular 2



**Language Choices**

**Angular 1's Impact**

**Comparing Concepts from Angular 1 to 2**

**Resources**



# Language Choices

---



# Coding Angular 2

ES5

ES6/ES2015+

TypeScript

Dart



# Coding Angular 2

ES5

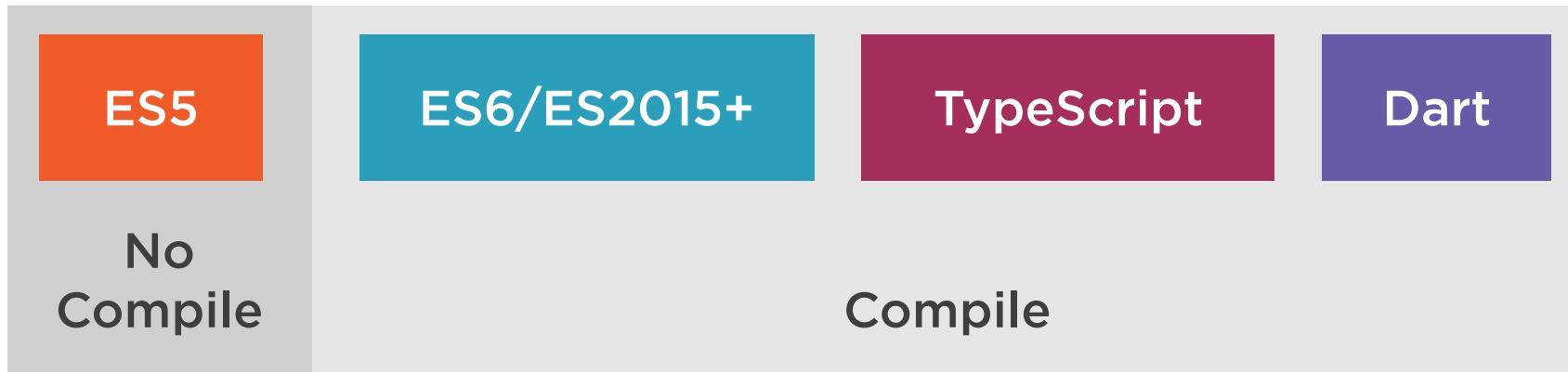
ES6/ES2015+

TypeScript

Dart



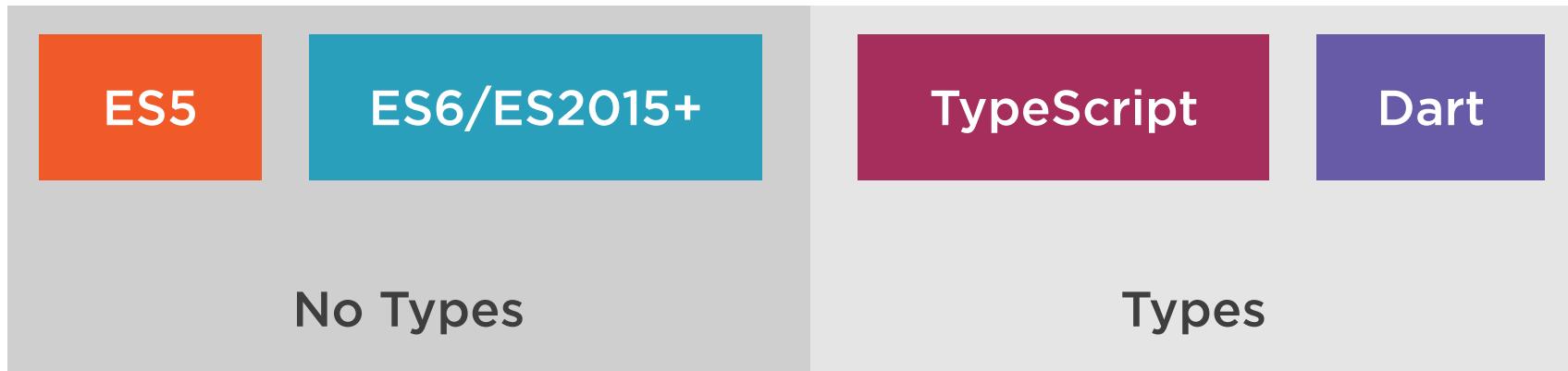
# Coding Angular 2



# Coding Angular 2



# Coding Angular 2



# Coding Angular 2



# Coding Angular 2

ES5

ES6/ES2015+

TypeScript

Dart



# Coding Angular 2

ES5

ES6/ES2015+

TypeScript

Dart



# Angular 1's Impact

---





**Angular 1 is widely used**  
**Huge ecosystem**



# 1.1 Million Developers



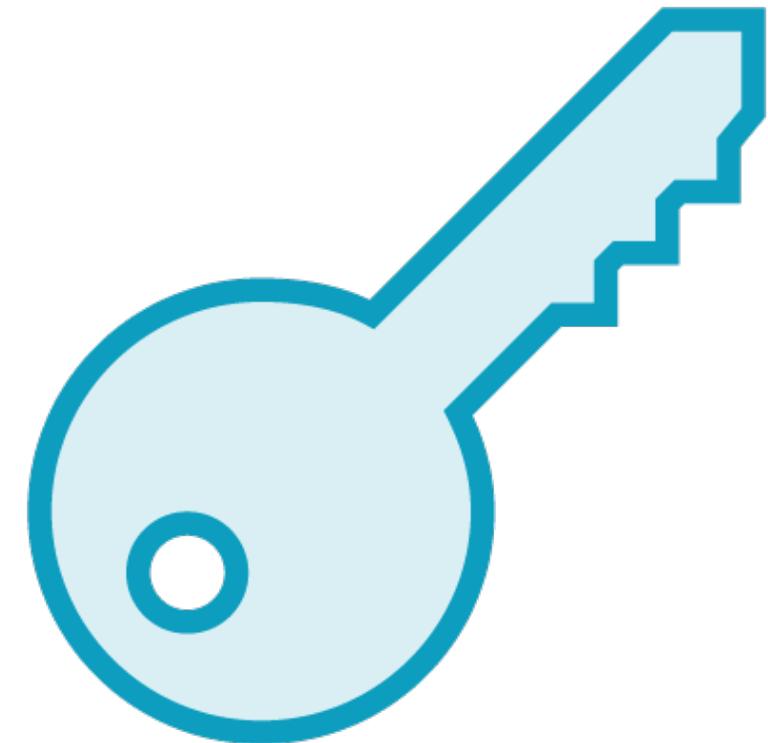
# Leaning on Your Angular 1 Skills

---



# Angular 1 to Angular 2

## 7 Key Comparisons



## 1

# Angular Modules

## Angular 1

```
(function () {  
  angular  
    .module('app', []);  
})();
```

Dependencie  
s

```
<html ng-app="app">
```

Root module

## Angular 2

```
import { NgModule } from '@angular/core';  
import { BrowserModule } from '@angular/platform-browser';  
  
@NgModule({  
  imports: [BrowserModule],  
  declarations: [ ... ],  
  bootstrap: [ ... ]  
})  
export class AppModule { }
```

Dependencie  
s

Root module



## 2

# Controllers to Components



Component

## Angular 1

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

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

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

## 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: 'The Force Awakens' };
}
```



## 3

# Structural Built-In Directives

## Angular 1

```
<ul>
  <li ng-repeat="vehicle in vm.vehicles">
    {{vehicle.name}}
  </li>
</ul>
<div ng-if="vm.vehicles.length">
  <h3>You have {{vm.vehicles.length}} vehicles</h3>
</div>
```

## Angular 2

```
<ul>
  <li *ngFor="let vehicle of vehicles">
    {{vehicle.name}}
  </li>
</ul>
<div *ngIf="vehicles.length">
  <h3>You have {{vehicles.length}} vehicles</h3>
</div>
```



## Structural Directive

```
<ul>
  <li *ngFor="let vehicle of vehicles">
    {{vehicle.name}}
  </li>
</ul>
<div *ngIf="vehicles.length">
  <h3>You have {{vehicles.length}} vehicles</h3>
</div>
```

Local Variable

Structural  
Directive

## Structural Directives

Indicated by the **\*** prefix

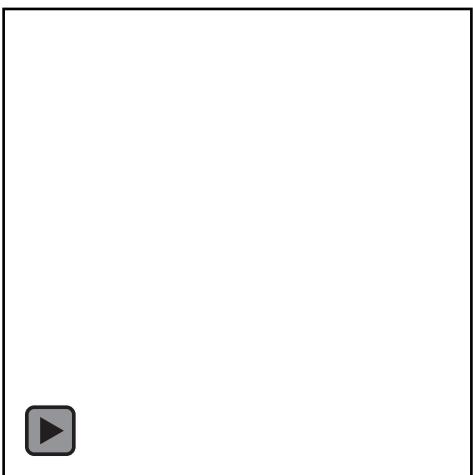
Changes the structure



# 4

## Data Binding

Interpolation

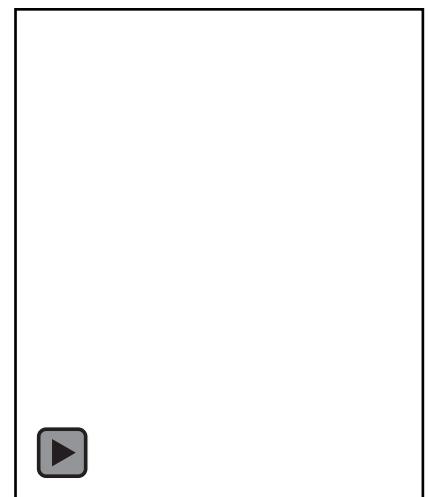


DOM

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

```
<h3 [innerText]="story.name"></h3>
```

Any HTML Element Property

```
<div [style.color]="color">{{story.name}}</div>
```



# 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">
```

Football in a  
Box

[ ( ) ]



## 5

# Removes the Need for Many Directives

## Angular 1

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

  
    {{vm.story}}
  </a>

</div>
```

## Angular 2

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

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

</div>
```



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



# No Longer Need These Directives Either

## Angular 1

```
ng-click="saveVehicle(vehicle)"
```

```
ng-focus="log('focus')"
```

```
ng-blur="log('blur')"
```

```
ng-keyup="checkValue()"
```

## Angular 2

```
(click)="saveVehicle(vehicle)"
```

```
(focus)="log('focus')"
```

```
(blur)="log('blur')"
```

```
(keyup)="checkValue()"
```



# 6

Se

Graphics Team:

Can we replace this with the same video you used in module 5 (services) ?

Angular 1

Angular 2

Factories

Services

Providers

Constants

Values

Class



# Creating Services

## Angular 1

```
angular
  .module('app')
  .service('VehicleService', VehicleService);

function VehicleService() {
  this.getVehicles = function () {
    return [
      { id: 1, name: 'X-Wing Fighter' },
      { id: 2, name: 'Tie Fighter' },
      { id: 3, name: 'Y-Wing Fighter' }
    ];
  }
}
```

## Angular 2

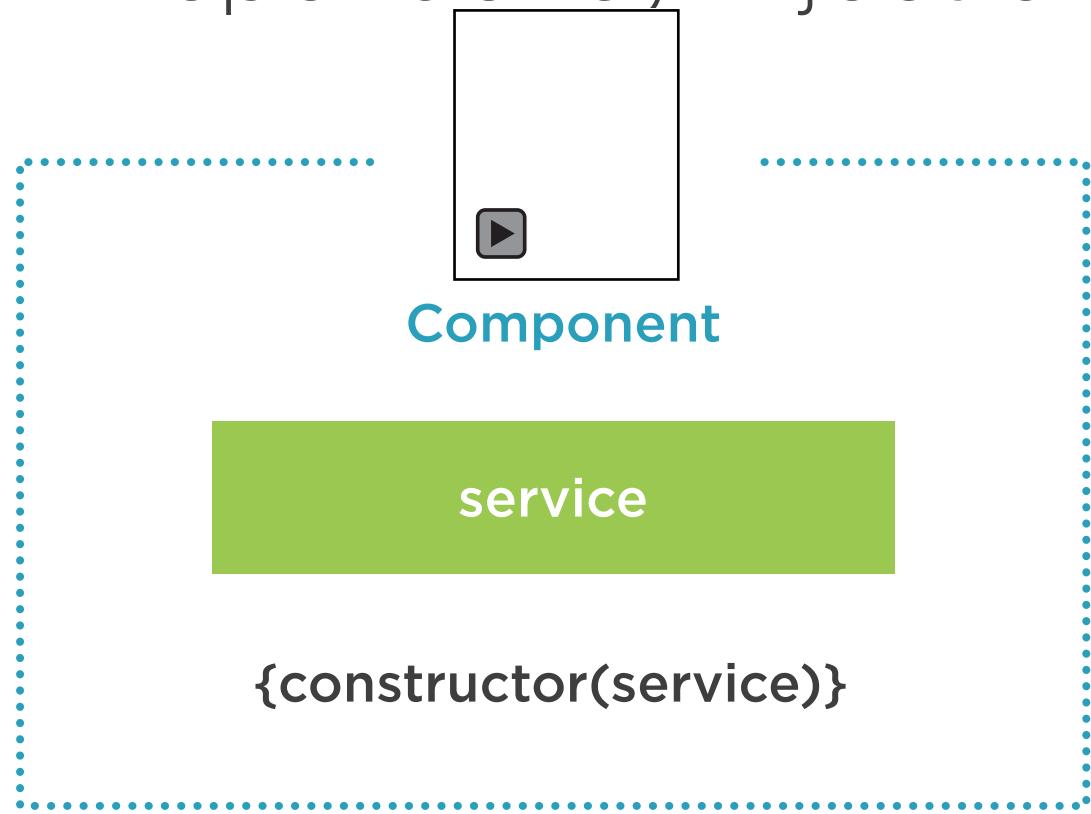
```
import { Injectable } from '@angular/core';
@Injectable()
export class VehicleService {
  getVehicles = () => [
    { id: 1, name: 'X-Wing Fighter' },
    { id: 2, name: 'Tie Fighter' },
    { id: 3, name: 'Y-Wing Fighter' }
  ];
}
```

**Just a Class**



## 7

# Dependency Injection



# Registering Services with the Injector

## Registration

```
angular
  .module('app')
    .service('VehicleService', VehicleService);

function VehicleService() {
  this.getVehicles = function () {
    return [
      { id: 1, name: 'X-Wing Fighter' },
      { id: 2, name: 'Tie Fighter' },
      { id: 3, name: 'Y-Wing Fighter' }
    ];
  }
}
```

## Angular 1

## Angular 2

## Registration

```
@NgModule({
  imports: [BrowserModule, FormsModule],
  declarations: [VehiclesComponent],
  providers: [VehicleService],
  bootstrap: [VehiclesComponent],
})
export class AppModule { }
```



# Dependency Injection

## Angular 1

```
angular
  .module('app')
  .controller('VehiclesController', VehiclesController);

VehiclesController.$inject = ['VehicleService'];
function VehiclesController(VehicleService) {
  var vm = this;
  vm.title = 'Services';
  vm.vehicles = VehicleService.getVehicles();
}
```

## Angular 2

```
@Component({
  moduleId: module.id,
  selector: 'my-vehicles',
  templateUrl: 'vehicles.component.html',
})
export class VehiclesComponent {
  vehicles = this.vehicleService.getVehicles();

  constructor(private vehicleService: VehicleService) { }
```

Injection

Injection



7

Removal of  
Many  
Directives

Angular  
Modules

Structural  
Directives

Components

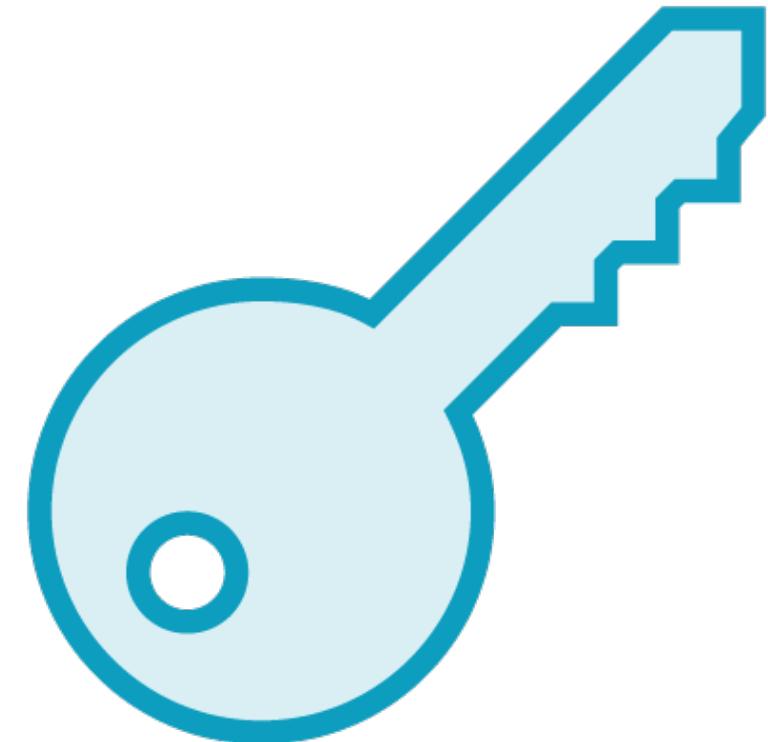
Services

Data Binding

Dependency  
Injection

## Angular 1 to Angular 2

### 7 Key Comparisons



# Many Angular 1 Concepts Translate

**Services**

**Filters / Pipes**

**Routing**

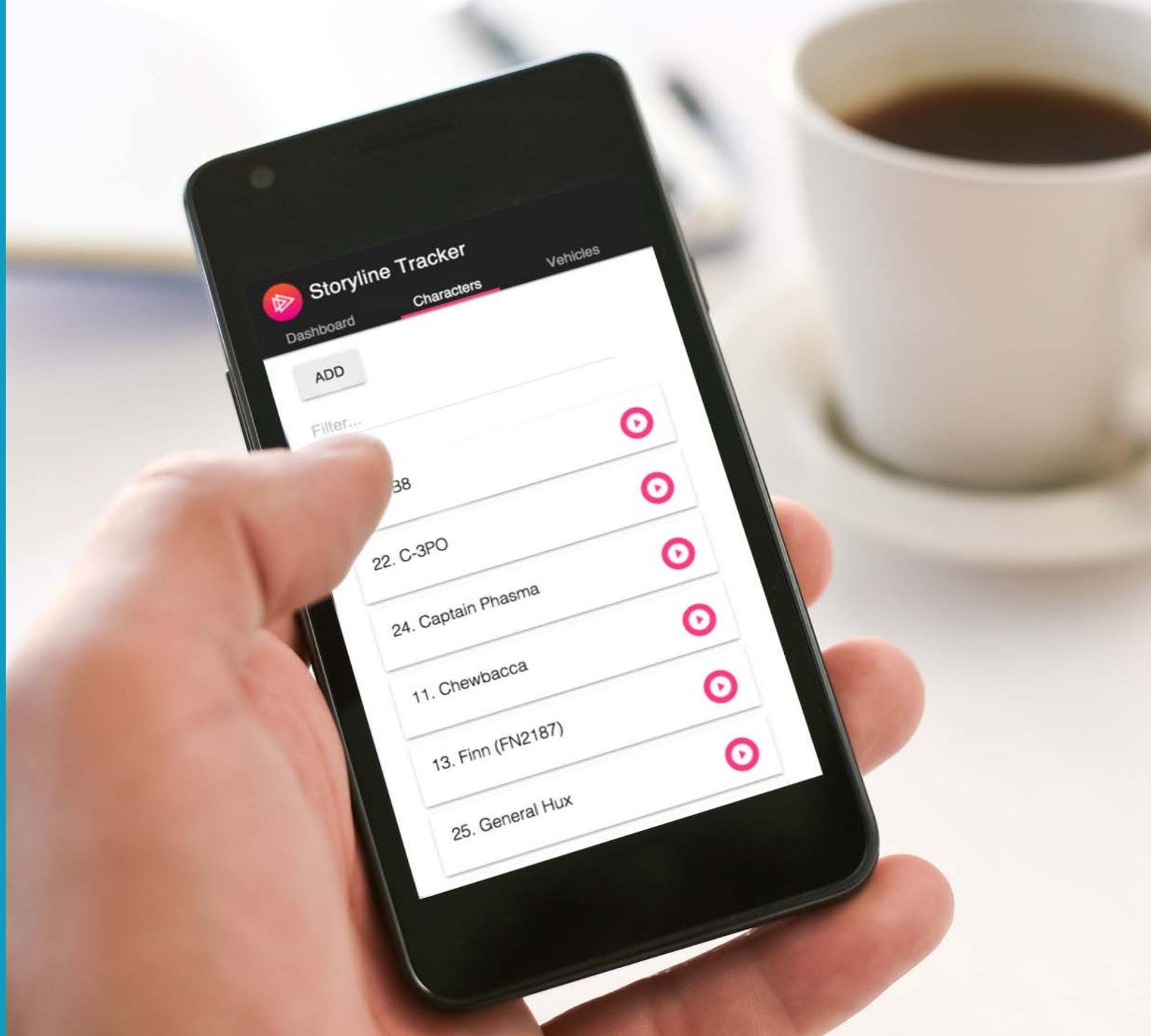
**HTTP**

**Events**

**Promises**



# Demo



# Angular 2 Team Resources

---



# Angular Documentation

<http://angular.io/docs>

The screenshot shows the Angular documentation website with a blue header. The header includes the Angular logo, navigation links for FEATURES, DOCS, EVENTS, and NEWS, and a 'GET STARTED' button. A search bar labeled 'SEARCH DOCS...' is also present. The main content area has a dark blue header with the title 'STYLE GUIDE' and a dropdown menu set to 'Angular 2 for TypeScript'. Below this, a sub-header reads 'Write Angular 2 with style.' The main text explains the purpose of the style guide: 'The purpose of this style guide is to provide guidance on building Angular applications by showing the conventions we use and, more importantly, why we choose them.' It then lists several sections: 'Style Vocabulary', 'Each guideline describes either a good or bad practice, and all have a consistent presentation.', and 'The wording of each guideline indicates how strong the recommendation is.' At the bottom, three definitions are provided in colored boxes: 'Do' (green), 'Consider' (blue), and 'Avoid' (red).

Angular

FEATURES DOCS EVENTS NEWS

GET STARTED

SEARCH DOCS...

QUICKSTART

TUTORIAL

BASICS

1. Overview

2. Architecture

3. Displaying Data

4. User Input

5. Forms

6. Dependency Injection

7. Template Syntax

8. Angular Cheat Sheet

9. Style Guide

10. Glossary

DEVELOPER GUIDE

COOKBOOK

API REFERENCE

## STYLE GUIDE

Angular 2 for TypeScript

Write Angular 2 with style.

The purpose of this style guide is to provide guidance on building Angular applications by showing the conventions we use and, more importantly, why we choose them.

### Style Vocabulary

Each guideline describes either a good or bad practice, and all have a consistent presentation.

The wording of each guideline indicates how strong the recommendation is.

**Do** is one that should always be followed. Always might be a bit too strong of a word. Guidelines that literally should always be followed are extremely rare. On the other hand, we need a really unusual case for breaking a *Do* guideline.

**Consider** guidelines should generally be followed. If you fully understand the meaning behind the guideline and have a good reason to deviate, then do so. Please strive to be consistent.

**Avoid** indicates something we should almost never do. Code examples to *avoid* have an unmistakable red header.



# Tour of Heroes Tutorial

The screenshot shows the Angular documentation website. At the top, there is a blue header bar with the Angular logo on the left and a "SITE MENU ▾" button on the right. Below the header is a search bar labeled "SEARCH DOCS..." and a "DOCS ▾" button. The main content area has a blue background and features the title "TUTORIAL: TOUR OF HEROES" in large white capital letters. Below the title is a dropdown menu with the option "Angular 2 for TypeScript". The main text on the page reads: "The Tour of Heroes tutorial takes us through the steps of creating an Angular application in TypeScript." Underneath this, there is a section titled "Tour of Heroes: the vision" with the text: "Our grand plan is to build an app to help a staffing agency manage its stable of heroes. Even heroes need to find work." At the bottom of the page, there is a footer note: "Of course we'll only make a little progress in this tutorial. What we do build will have many of the features". In the bottom right corner of the page, there is a small circular icon containing a play button symbol.

SITE MENU ▾

SEARCH DOCS...

DOCS ▾

TUTORIAL: TOUR OF HEROES

Angular 2 for TypeScript ▾

The Tour of Heroes tutorial takes us through the steps of creating an Angular application in TypeScript.

## Tour of Heroes: the vision

Our grand plan is to build an app to help a staffing agency manage its stable of heroes. Even heroes need to find work.

Of course we'll only make a little progress in this tutorial. What we do build will have many of the features

# Comparing Angular 1 to Angular 2



TypeScript is a Superset of ES6/ES2015+

40+ Built-In Directives Removed

Bind to any HTML Element Property or Event

Angular 1 Concepts Map to Angular 2



# Angular 2 Essentials: Modules, Components, Templates, and Metadata

---



**John Papa**

PRINCIPAL ARCHITECT

@john\_papa

[www.johnpapa.net](http://www.johnpapa.net)



# Overview



**ES Modules**

**Angular Modules**

**Components**

**Templates**

**Metadata**



# ES Modules

---



ES Modules are often referred to  
simply as “Modules”



# Modules

We assemble our application from modules.

A module exports an asset such as a Service, Component, or a shared value



```
export interface Vehicle {  
  id: number;  
  name: string;  
}  
  
export class VehicleService {  
  //...  
}
```

## Exporting modules

Assets can be exported using the **export** keyword



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

import { Vehicle, VehicleService } from './vehicle.service';
```

## Importing modules

Modules and their contents can be imported using the **import** keyword

We import the Vehicle and VehicleService using destructuring



# Angular Modules

---

aka NgModules

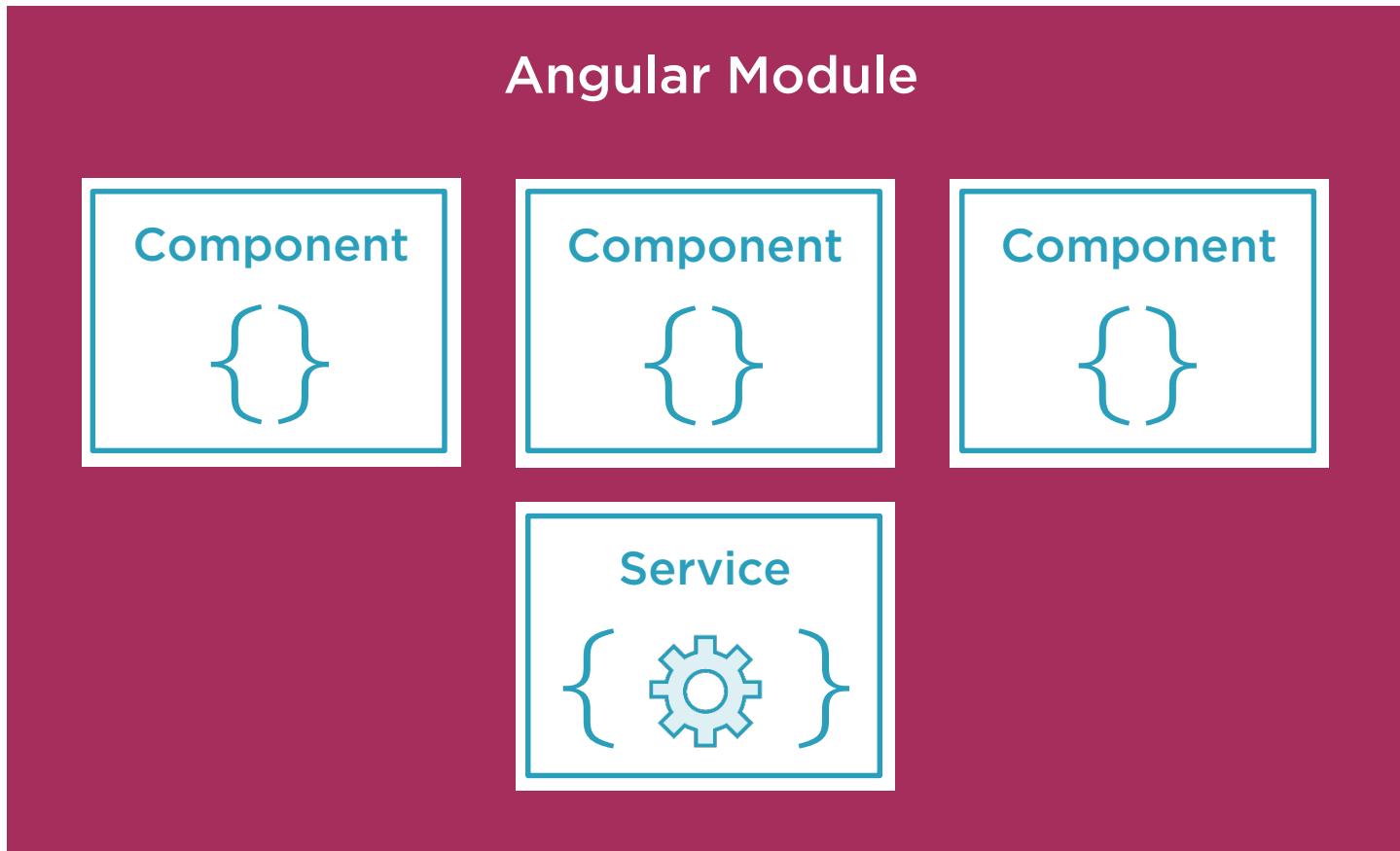


# Angular Modules

We use **NgModule** to organize our application into cohesive blocks of related functionality



# Angular Modules Organize Functionality



# Separate Features into Angular Modules



An Angular Module is a class  
decorated by  
`@NgModule`



# Roles of Angular Modules

Import other  
Angular Modules

Identify  
Components, Pipes,  
and Directives

Export it's  
Features

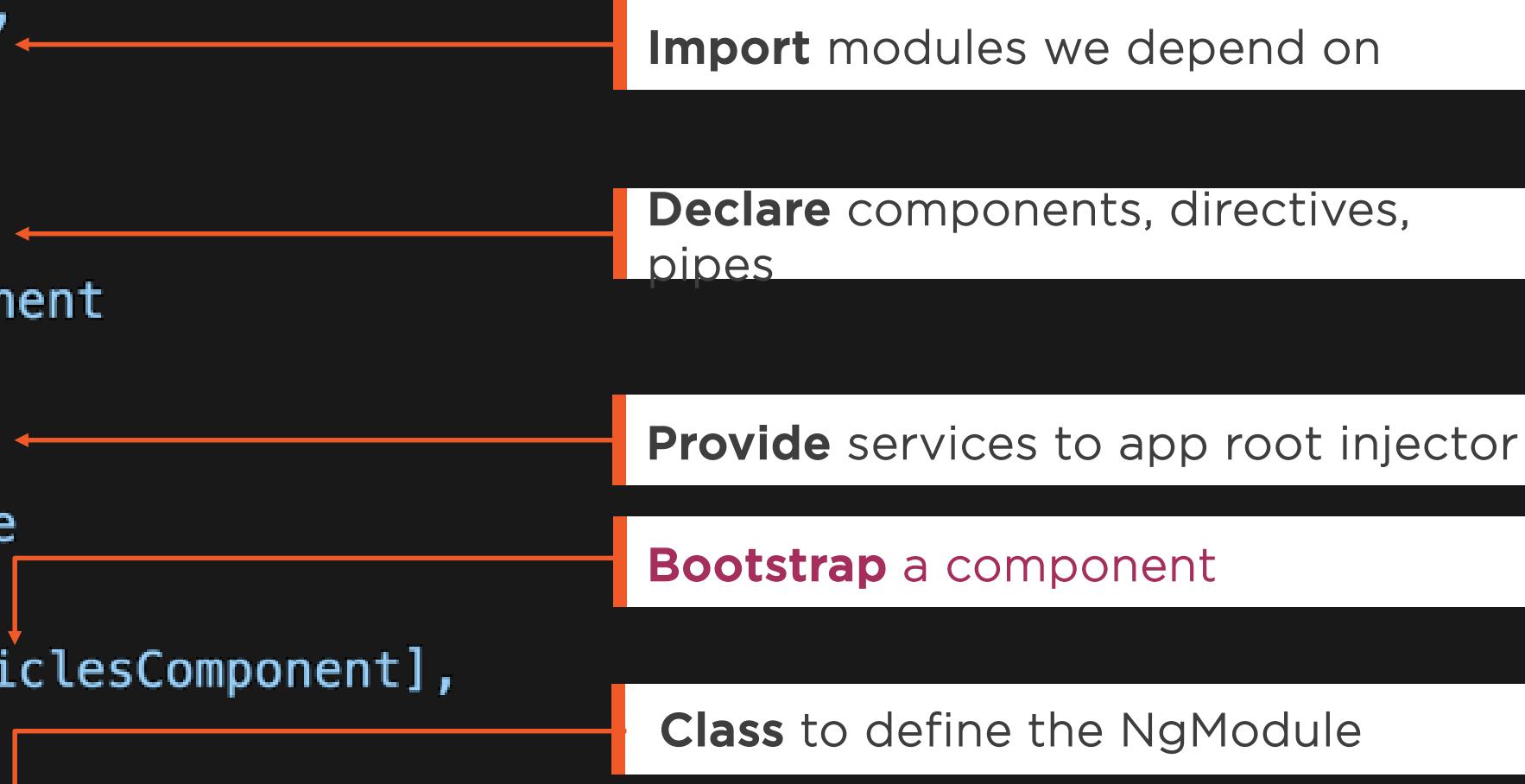
Provide services  
to injectors

Can be eagerly or  
lazily loaded



## The Root Angular Module

```
@NgModule({  
  imports: [  
    BrowserModule,  
    FormsModule  
,  
  declarations: [  
    VehiclesComponent  
,  
  providers: [  
    VehicleService  
,  
  bootstrap: [VehiclesComponent],  
})  
export class AppModule { }
```



**Import** modules we depend on

**Declare** components, directives, pipes

**Provide** services to app root injector

**Bootstrap** a component

**Class** to define the NgModule



Every app begins with one  
Angular Module



# Components

---



# Angular 2 Components

A Component contains application logic that controls a region of the user interface that we call a view.



## Anatomy of a Component

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

```
import { Vehicle } from './vehicle.service';
```

```
@Component({  
  moduleId: module.id,  
  selector: 'story-vehicles',  
  templateUrl: 'vehicles.component.html',  
})
```

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

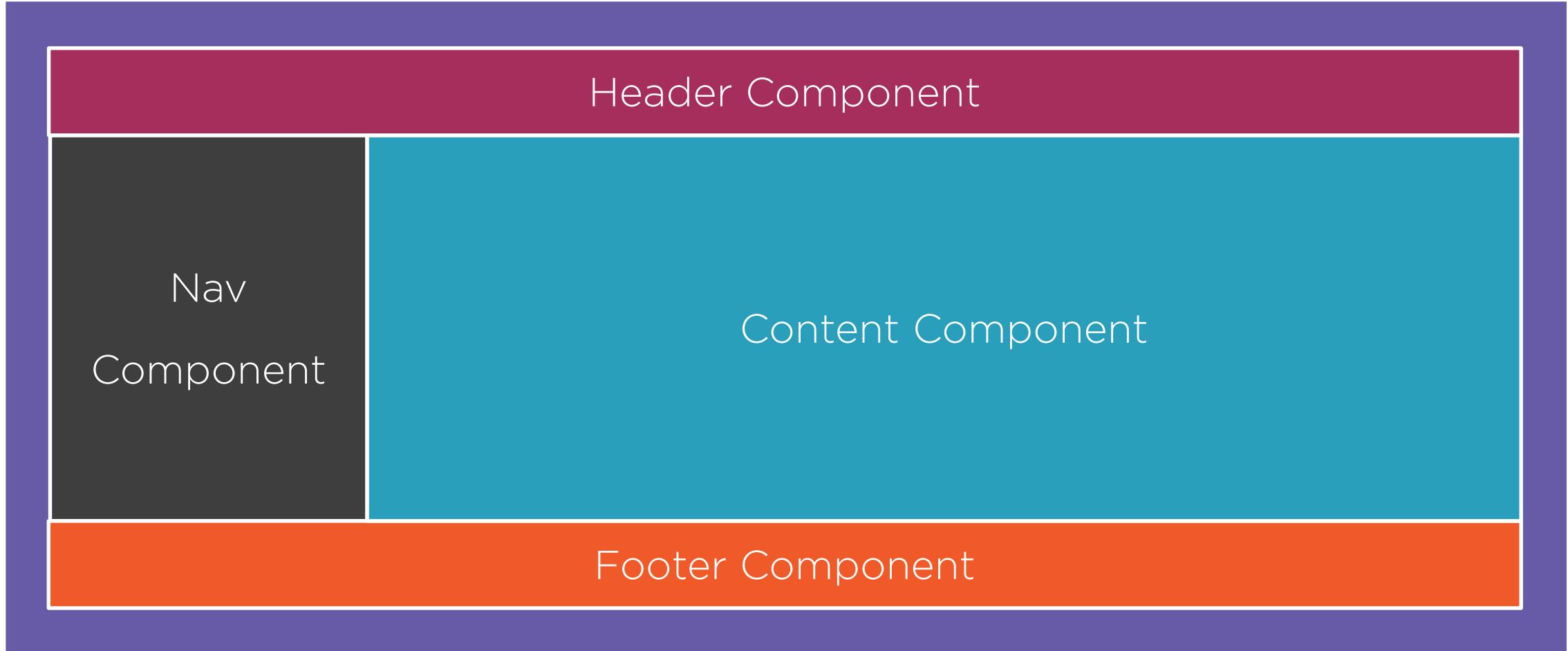
**Imports** (use other modules)

**Metadata** (describe the component)

**Class** (define the component)



# Assembling Our App from Components



```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';

platformBrowserDynamic().bootstrapModule(AppModule);
```

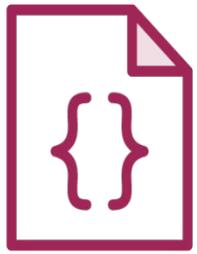
## Bootstrapping a Component

**Entry point for the app**

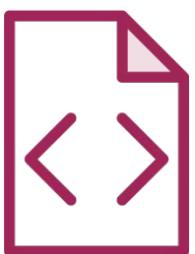
**This is where we start**



# Comparing Angular 1 to Angular 2



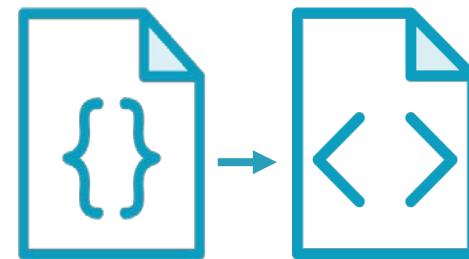
Controller



Template



Angular 1



Component Identifies  
its Template



Angular 2



## character.component.ts

```
@Component({  
  moduleId: module.id,  
  selector: 'story-character',  
  templateUrl: 'character.component.html'  
})  
export class CharacterComponent {  
  name = 'Han Solo';  
}
```

Component has logic

## character.component.html

```
<h3>My name is {{name}}</h3>
```

What is rendered

## index.html

```
<story-character>Loading Demo ...</story-character>
```

Where the component is placed



## character.component.ts

```
@Component({  
  moduleId: module.id,  
  selector: 'story-character',  
  templateUrl: 'character.component.html'  
})  
export class CharacterComponent {  
  name = 'Han Solo';  
}
```

Component has logic

## character.component.html

```
<h3>My name is {{name}}</h3>
```

What is rendered

## index.html

```
<story-character>Loading Demo ...</story-character>
```

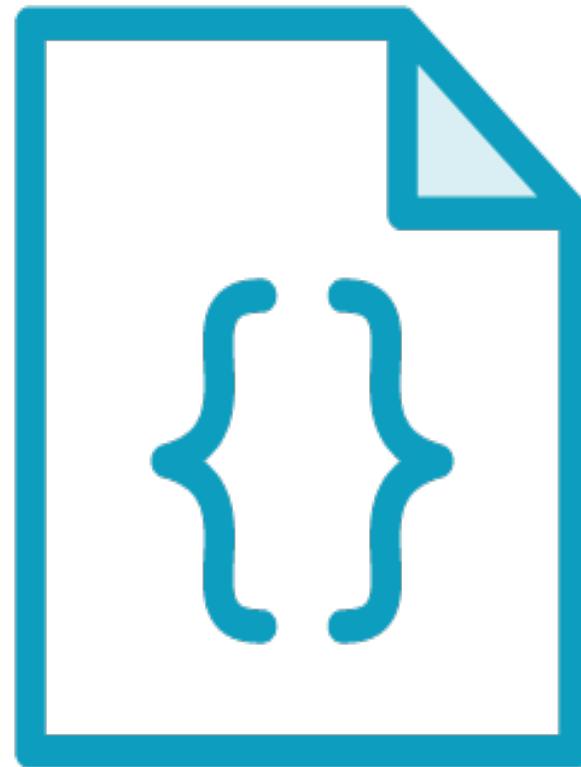
Where the component is placed



# Components Demo



# Components



# Templates

---



# Templates are the View

Templates are mostly HTML, with a little help from Angular. They tell Angular how to render the Component



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

<my-character *ngIf="selectedCharacter"
  [character]="selectedCharacter"></my-character>
```

**Directives** (e.g. \*ngFor)

**Interpolation**

**Nested Component**

## Templates

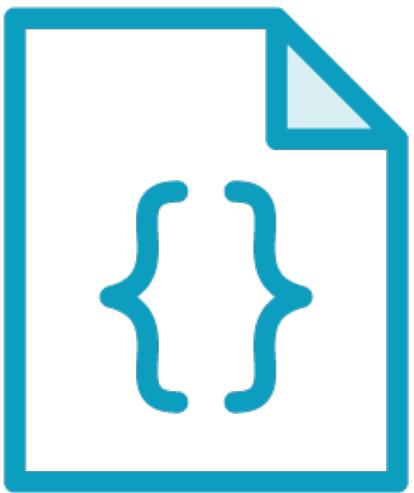
### HTML

Directives, as needed

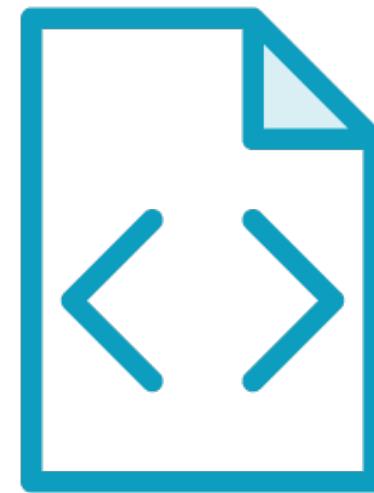
Template Binding Syntax



# Connecting the Component to its Template



Component



Template



```
@Component({  
  selector: 'my-character-list',  
  template: `  
    <ul>  
      <li *ngFor="let character of characters">  
        {{ character.name }}  
      </li>  
    </ul>  
`  
})  
export class CharacterListComponent { }
```

Template String

## Inline Templates

**template** defines an embedded template string

Use back-ticks for multi-line strings



```
@Component({  
  moduleId: module.id,  
  selector: 'story-vehicles',  
  templateUrl: 'vehicles.component.html',  
})  
export class VehiclesComponent { }
```

Template Url

## Linked Templates

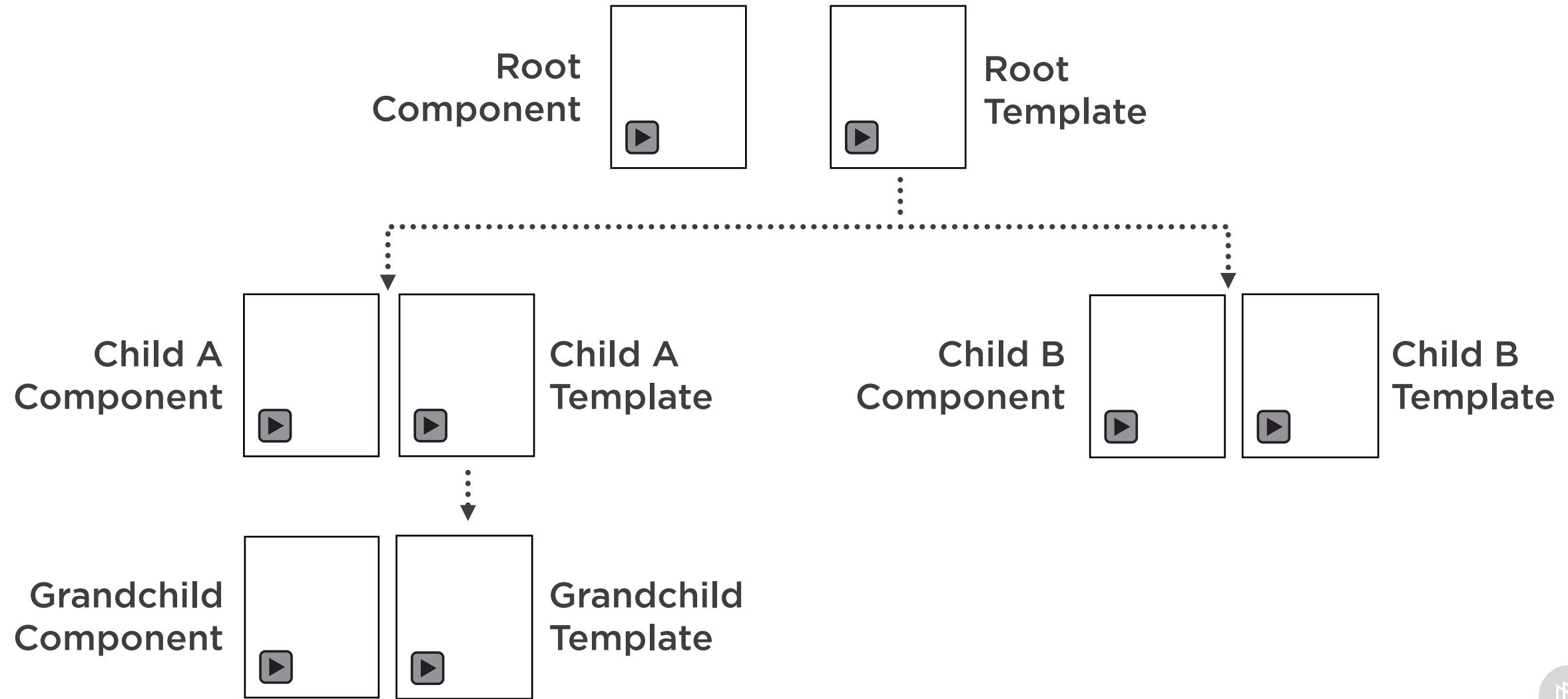
**templateUrl** links the Component to its Template



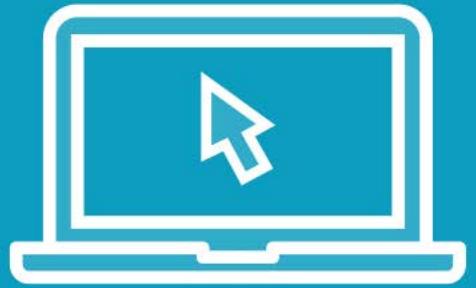
Components have templates,  
which may use other components



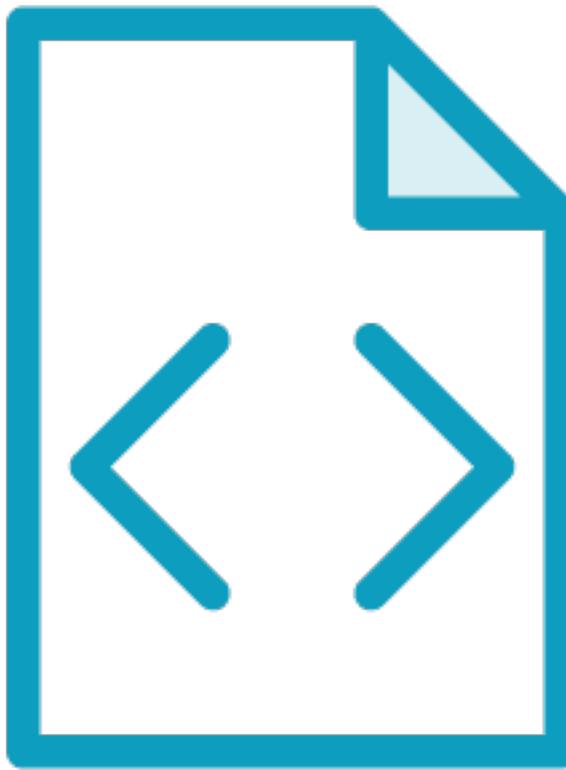
# Templates Contain Other Components



Demo



# Templates



# Metadata

---

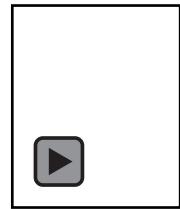


# Metadata

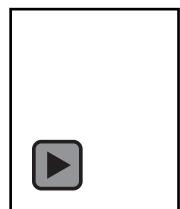
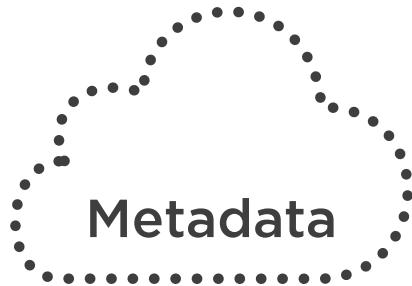
We use Metadata to tell Angular about the objects we build.



# Metadata Links the Template to the Component



Template



Component



We declare our components, directives  
and pipes in an Angular Module



```
@NgModule({  
  imports: [BrowserModule],  
  declarations: [  
    CharacterComponent,  
    CharacterListComponent  
,  
  bootstrap: [CharacterListComponent],  
})  
export class AppModule { }
```



## Declaring Components

**BrowserModule** includes **CommonModule**

Built-in directives like **\*ngFor** and **ngClass** are in **CommonModule**

We tell Angular what **<my-character-list>** and **<my-character>** are



# Examining a Component and its Metadata

---



# Decorators

The @ is a decorator that provides metadata describing the Component

# Component

Component definition class. Controls a patch of screen real estate that we call a View

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

  constructor(private characterService: CharacterService) { }

  ngOnInit() {
    this.characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

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



# Template and Styles

Tells the Component where to find them.

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

  constructor(private characterService: CharacterService) { }

  ngOnInit() {
    this.characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

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



# Providers

These services will be registered with this component's injector. Only do this once.

Generally, prefer registering providers in angular modules to registering in components.

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

  constructor(private characterService: CharacterService) { }

  ngOnInit() {
    this.characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

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



## Injection

Inject a Service into another object.

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

  constructor(private characterService: CharacterService) { }

  ngOnInit() {
    this.characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

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



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

  constructor(private characterService: CharacterService) { }

  ngOnInit() {
    this.characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

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

## Output

Component can communicate to anyone hosting it

## Emit Events

Component emits events via output



## Input

Pass values into  
the Component

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

  constructor(private characterService: CharacterService) { }

  ngOnInit() {
    this.characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

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



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

  constructor(private characterService: CharacterService) { }

  ngOnInit() {
    this.characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

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

## Properties

Component exposes properties that can be bound to its Template



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

  constructor(private characterService: CharacterService) { }

  ngOnInit() {
    this.characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

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

## Actions

Functions can be exposed, bound and called by the Template to handle events



# Input and Output

Components allow input properties to flow in, while output events allow a child Component to communicate with a parent Component.



## Output

```
export class CharactersComponent implements OnInit {  
  @Output() changed = new EventEmitter<Character>();  
  @Input() storyId: number;  
  characters: Character[];  
  selectedCharacter: Character;  
  
  select(selectedCharacter: Character) {  
    this.selectedCharacter = selectedCharacter;  
    this.changed.emit(selectedCharacter);  
  }  
}
```

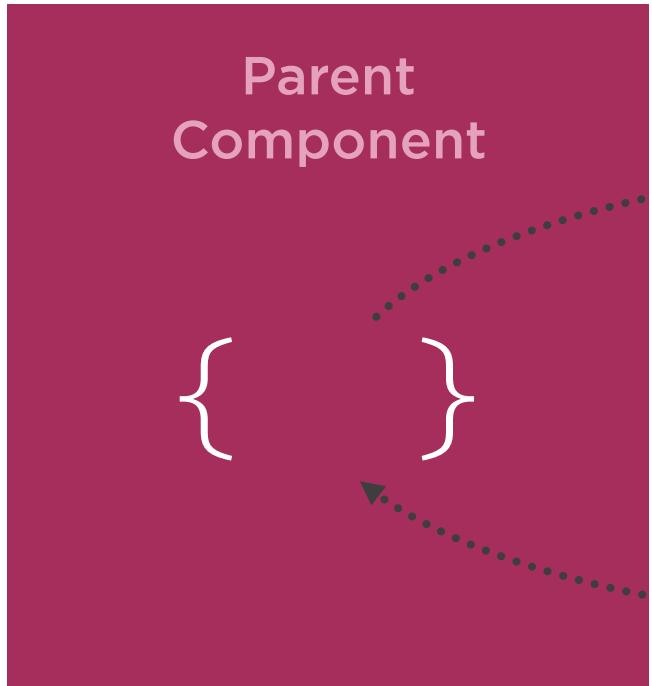
Output property

Emit the output

```
<div>  
  <h1>Storyline Tracker</h1>  
  <h3>Component Demo</h3>  
  <story-characters [storyId]="7"  
    (changed)=changed($event)>  
  </story-characters>  
</div>
```

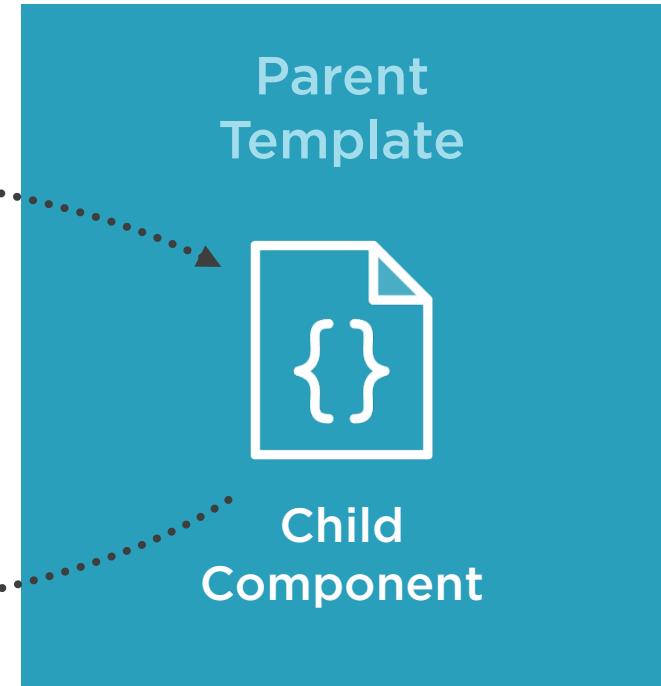
Bind to the event in  
the Parent  
Component





Property  
Binding

Event  
Binding

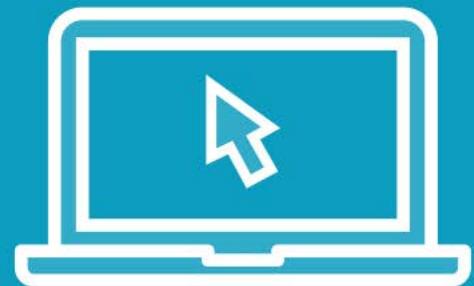


Parent  
Template

Child  
Component



Demo



# Component Input and Output



# ViewChild

Use ViewChild when a parent Component needs to access a member of its child Component





# ViewChild

Components

## Child

```
export class FilterComponent {  
  @Output() changed: EventEmitter<string>;  
  filter: string;  
  
  clear() {  
    this.filter = '';  
  }  
  // ...  
}
```

Child  
Component's  
function

## Parent

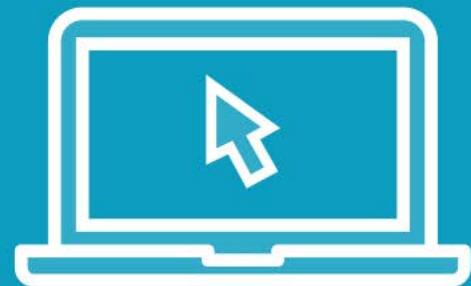
```
export class CharacterListComponent {  
  characters: Character[];  
  @ViewChild(FilterComponent) filter: FilterComponent;  
  
  filtered = this.characters;  
  
  getCharacters() {  
    this.characterService.getCharacters()  
      .subscribe(characters => {  
        this.characters = this.filtered = characters;  
        this.filter.clear();  
      });  
    // ...  
  }  
}
```

Grab the child

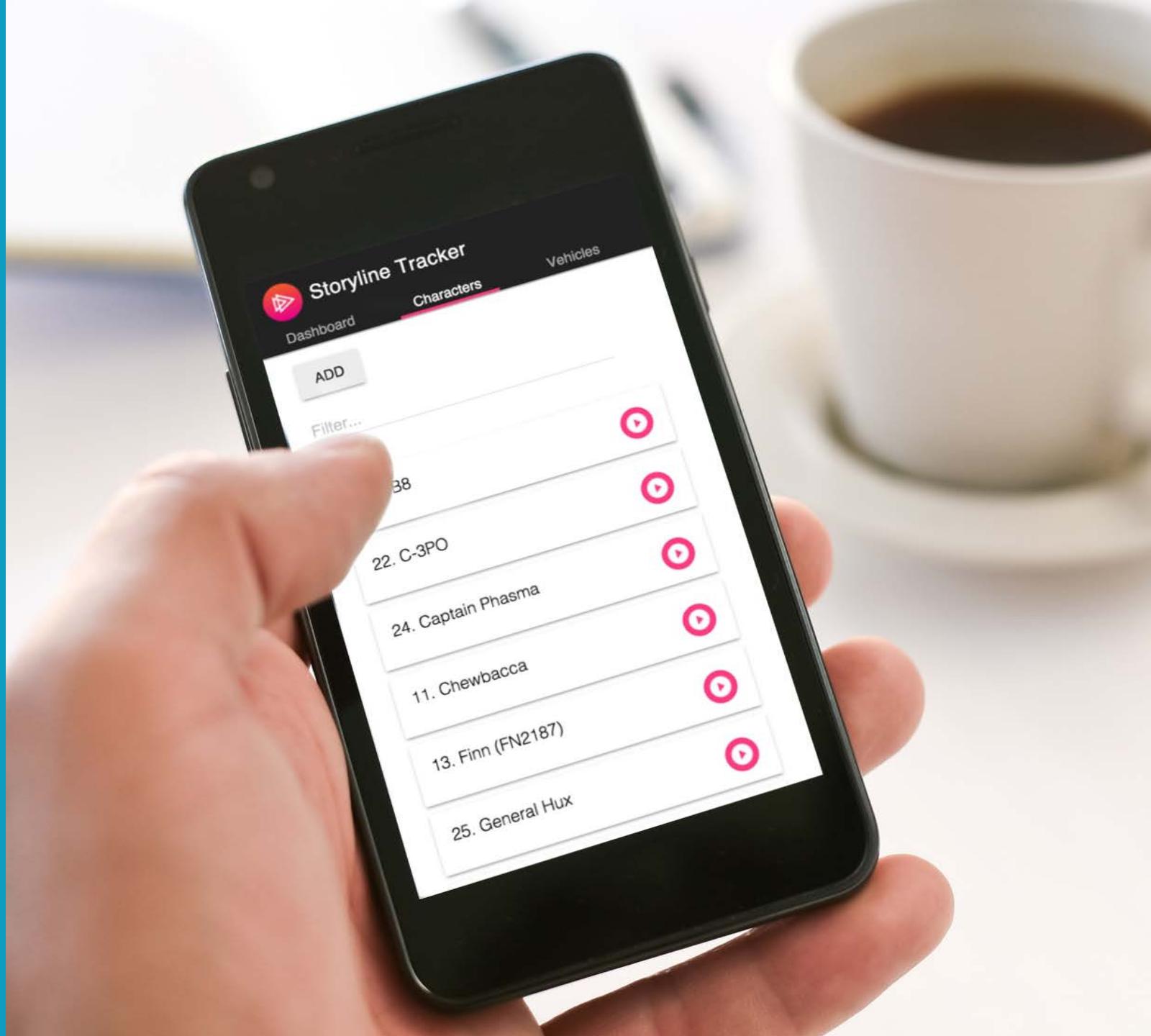
Call its member



# Demo



## Putting It All Together



# Summary



**Angular Modules organize an app**

**Components control a region of the page**

**Templates tell Angular how to render**

**Metadata describes objects**



# Displaying Data: Data Binding, Directives, and Pipes

---



**John Papa**

PRINCIPAL ARCHITECT

@john\_papa

[www.johnpapa.net](http://www.johnpapa.net)



# Overview



**Data Binding**  
**Built-in Directives**  
**Pipes**



# Data Binding

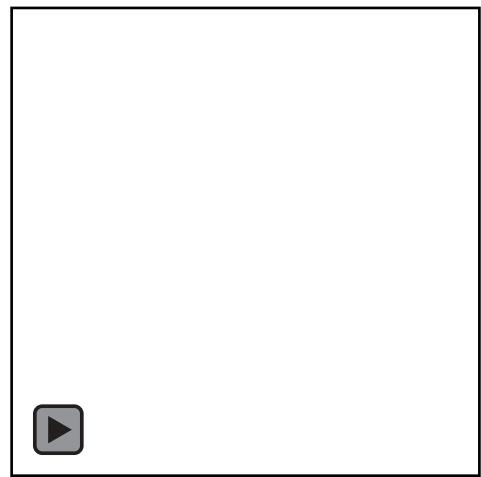
---



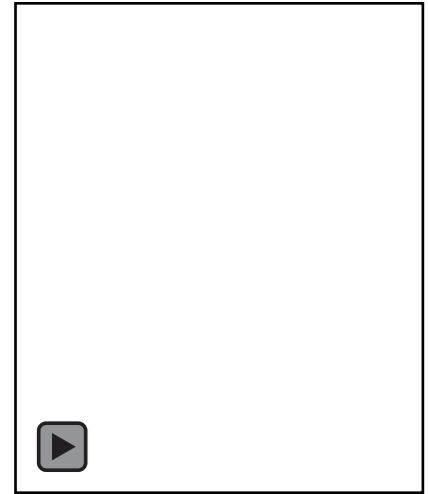
# Data Binding

We use data binding to help coordinate communication between a Component and its Template.





DOM



Component

`{{expression}}`

Interpolation

`[property] = “expression”`

One Way Binding

`(event) = “statement”`

Event Binding

`[(ngModel)] = “property”`

Two Way Binding



Angular 2's change detection is based on unidirectional data flow



# Benefits of Angular 2's Unidirectional Data Flow

Easier widget integration

No more \$apply

No more repeated digest cycles

No more watchers

No more performance issues with digest cycle and watcher limits



# Interpolation

Using the {{ }} to render the bound value to the Component's Template



## One Way In

```
<h3>Vehicle: {{vehicle.name}}</h3>
<div>
  
  <a href="{{vehicle.wikiLink}}>Wiki</a>
</div>
```

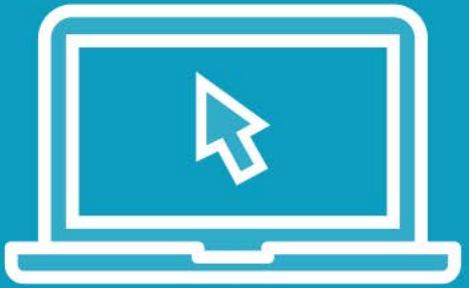
## Interpolation

Evaluate an expression between double curly braces

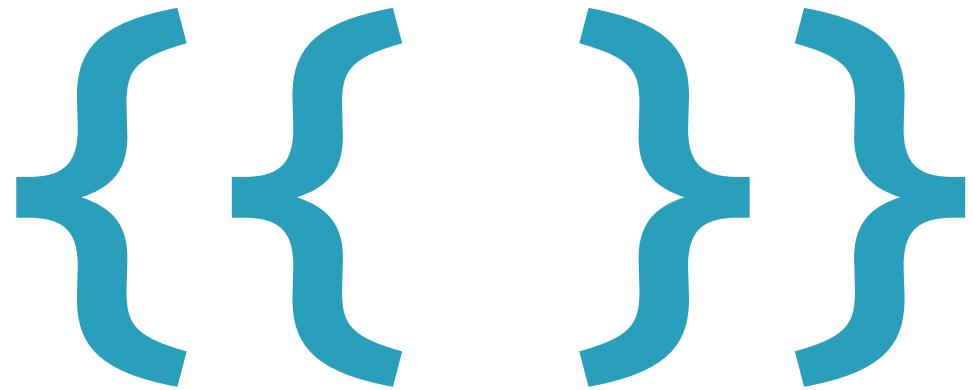
`{{ expression }}`



Demo



Interpolation

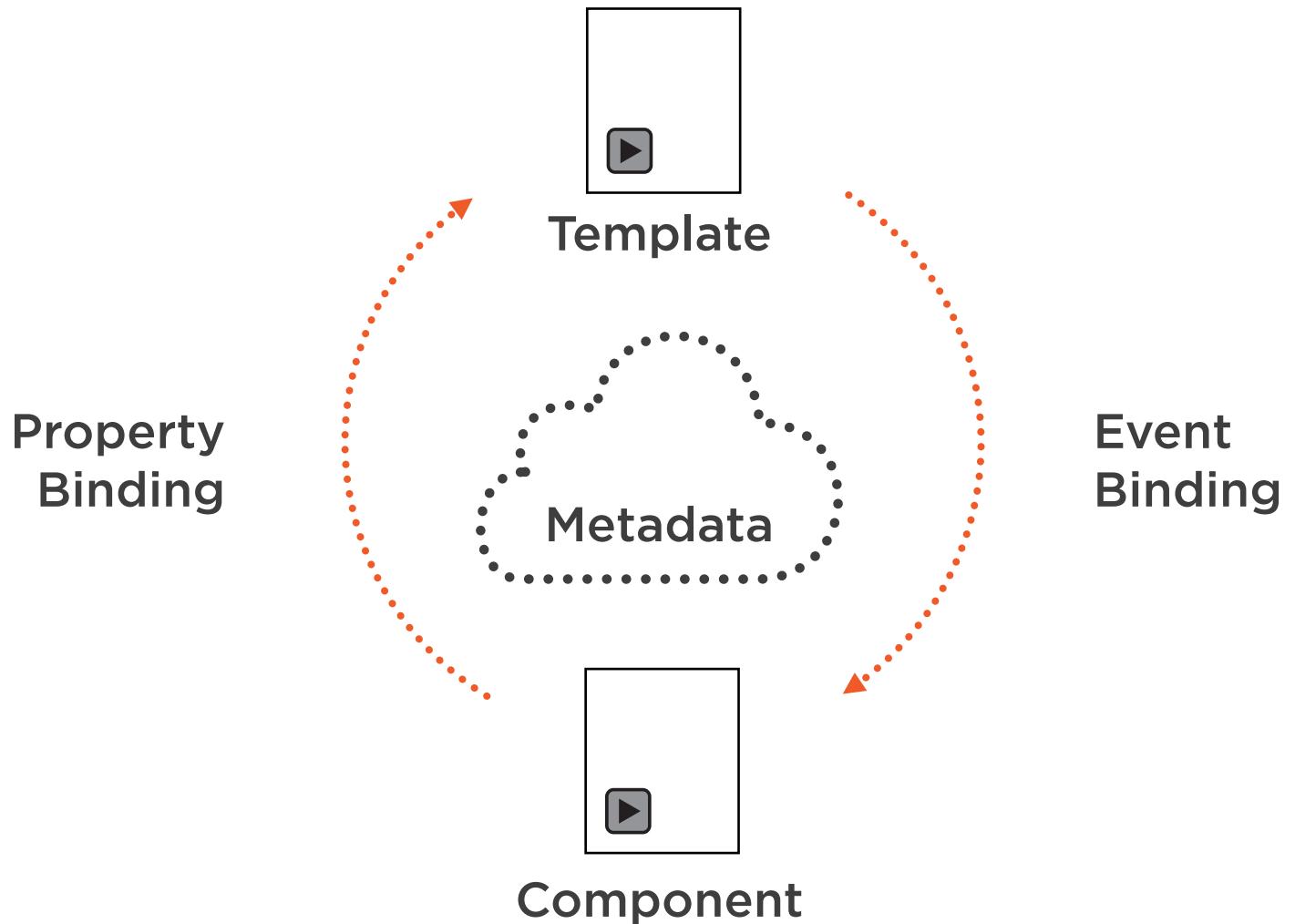


# Property Binding

Using the `[ ]` to send values from the Component to the Template



# Data Binding Communication



We set properties and events of  
DOM elements, not attributes



# One Way

Binding target property



`{{expression}}`  
`[target] = "expression"`  
`bind-target = "expression"`

Data source to view target

## One Way In

```
<img [src]="vehicle.imageUrl">  
<vehicle-detail [vehicle]="currentVehicle"></vehicle-detail>  
<div [ngClass] = "{selected: isSelected}">X-Wing</div>
```

Element property

Component property

Directive property

## Property Binding

[property]=“expression”

Bind to element, Component or a directive property



## One Way In

```
<button [attr.aria-label]="ok">ok</button>
```

Attribute binding

```
<div [class.isStopped]="isStopped">Stopped</div>
```

Class property binding

```
<button [style.color]="isStopped ? 'red' : 'blue'">
```

Style property binding

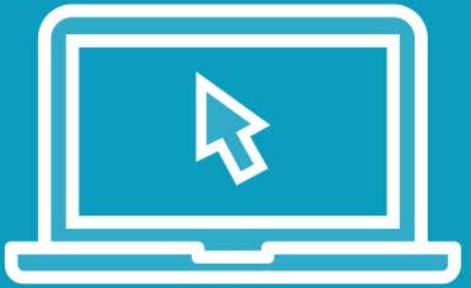
## Property Binding

For attributes use **attr**

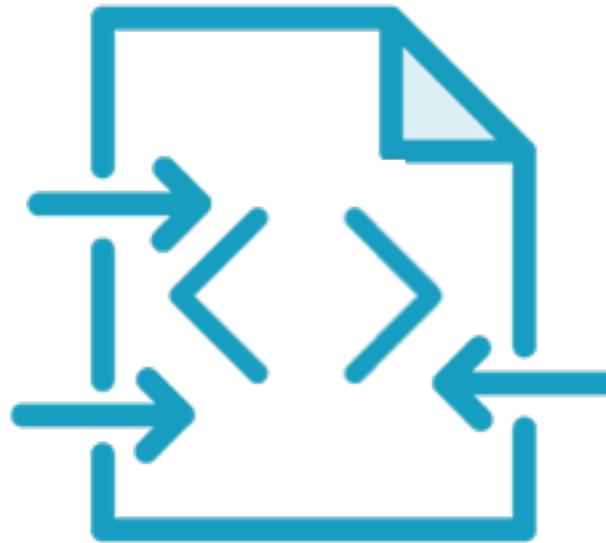
Use dots for nested properties



Demo



# Property Binding



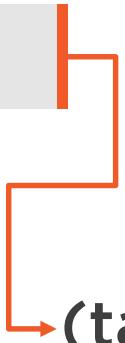
# Event Binding

Using the `( )` to send events from the Template to the Component



# One Way

Binding target event



`(target) = "statement"`  
`on-target = "statement"`

View target to data source

## One Way to the Component

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

Element event

```
<vehicle-detail (changed)="vehicleChanged()"></vehicle-detail>
```

Component event

## Event Binding

Execute an expression when an event occurs

**(event-target)=“statement”**



## One Way to the Component

```
<input [value]="vehicle.name"  
       (input)="vehicle.name=$event.target.value">
```

Event message

Input change event

\$event

Contains a message about the event



```
@Input() vehicle: Vehicle;  
@Output() onChanged = new EventEmitter<Vehicle>();  
changed() { this.onChanged.emit(this.vehicle); }
```

Custom event

```
<vehicle-detail [onChanged]="vehicleChanged($event)"  
[vehicle]="currentVehicle"> </vehicle-detail>
```

Output (event)

## Custom Events

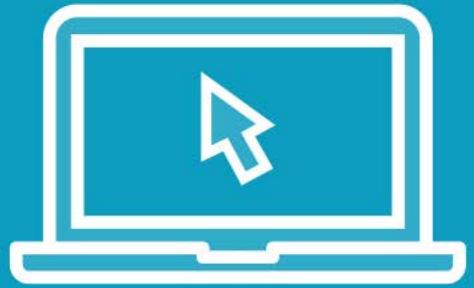
**EventEmitter** defines a new event

Fire its **emit** method to raise event with data

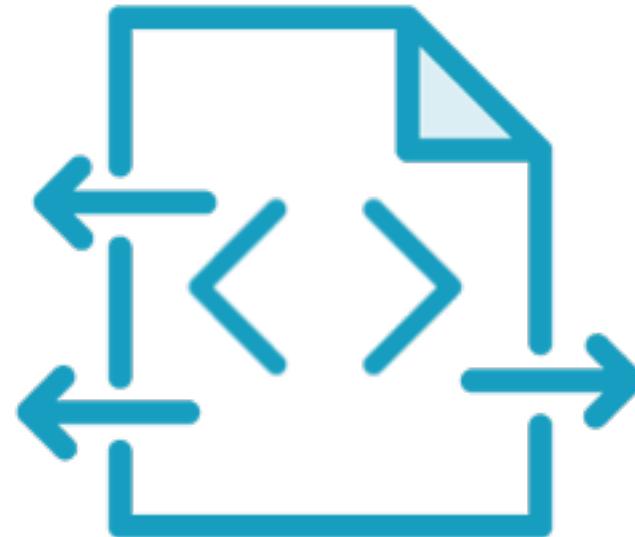
Bind to the event on the Component's Template



Demo



# Event Binding



# Two Way Binding

[`( )`] sends a value from Component to Template, and  
sends value changes in the Template to the Component



# Two Way

```
[(ngModel)] = "expression"  
bindon-ngModel= "expression"
```



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



Built-in directive

## Two Way Binding

**[ ( ) ] = Football in a box**



## Importing the FormsModule

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';

@NgModule({
  imports: [
    BrowserModule,
    FormsModule
  ],
  // ...
})
```

Import FormsModule

Import into the Angular module

## Using ngModel

Import **FormsModule**

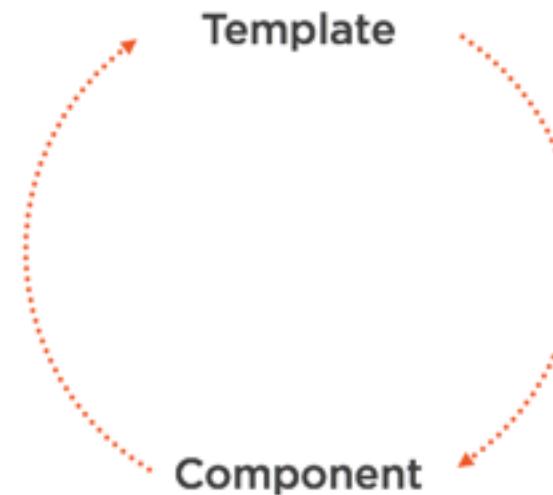
Add it to the Angular module's import list



Demo



# Data Binding



# Built-in Directives

---



# Directives

When Angular renders templates, it transforms the DOM according to instructions from Directives



# Angular Class and Style Directives

## Angular 1

ng-class

```
ng-class="{active: isActive, color: myColor}"
```

ng-style

```
ng-style="{color: colorPreference}"
```



## Angular 2

ngClass

```
[ngClass]="{active: isActive, color: myColor}"
```

ngStyle

```
[ngStyle]="{color: colorPreference}"
```

```
[style.color]="colorPreference"
```



## Style Binding

```
<div [ngStyle]="setStyles()">{{vehicle.name}}</div>
```

Style binding

ngStyle

Alternative to **[style.style-name]**

Setting multiple styles



## Class Binding

```
<div [ngClass]="setClasses()">{{vehicle.name}}</div>
```

Class binding

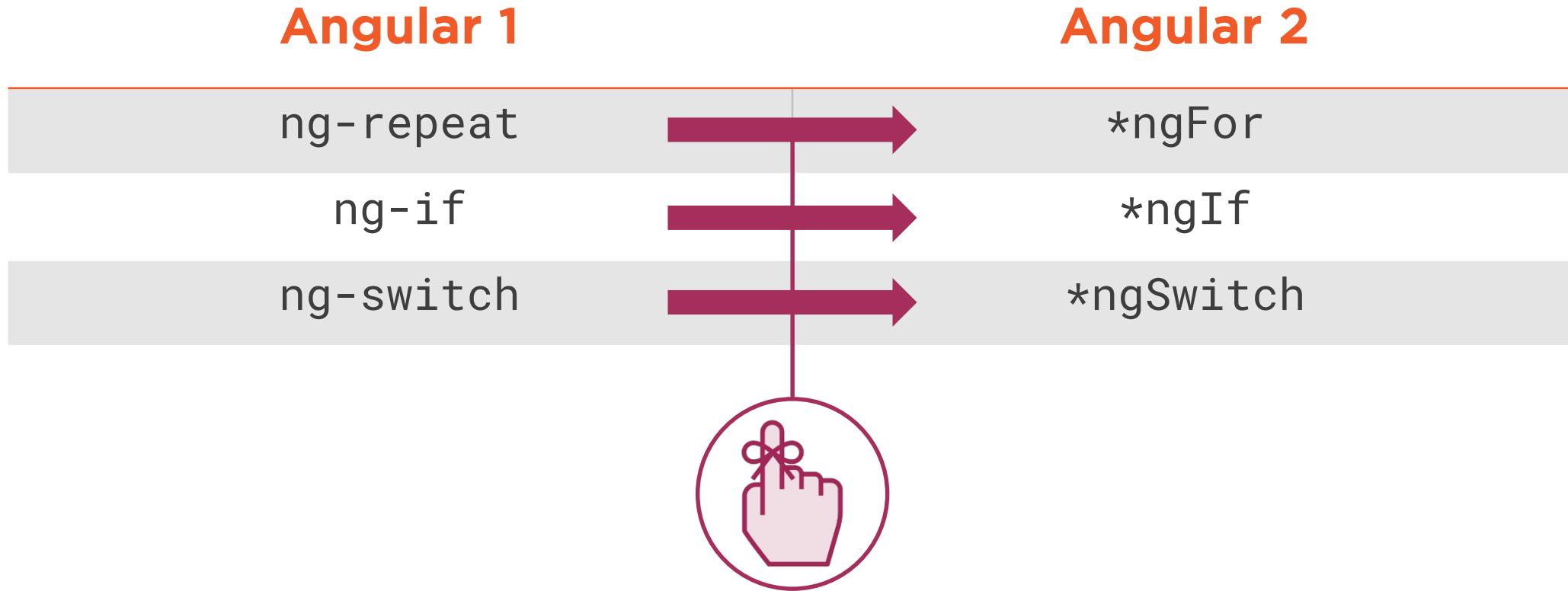
ngClass

Alternative to **[class.class-name]**

Setting multiple classes



# Angular Structural Directives



Familiar concepts translate to Angular 2



## Conditional Template

```
<div *ngIf="currentVehicle">  
  You selected {{currentVehicle.name}}</div>
```

Show template if truthy

\*ngIf

Conditionally removes elements from the DOM

Structural directive

Use `[style.visibility] = "isVisible()"` to hide



## Repeating a Template

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

Iterate over the characters

Local variable

\*ngFor

Structural directive

Show an element n number of times

let declares a local variable



## Creating an index

```
<ul>
  <li *ngFor="let character of characters, let i = index">
    {{i}}. {{ character.name }}
  </li>
</ul>
```



Local variable

## Local Variables

**let** declares a local variable



## Importing the CommonModule

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';

@NgModule({
  imports: [
    BrowserModule, ←
    FormsModule
  ],
  // ...
})
```

Import BrowserModule

Import into the Angular module

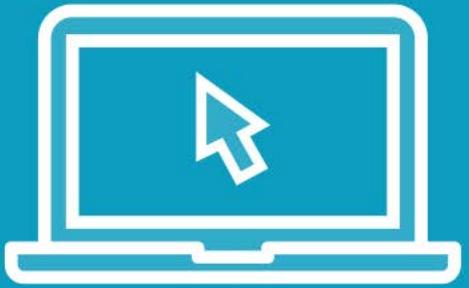
## Using Built-In Directives

BrowserModule imports CommonModule

Import and add it to the Angular module's import list



Demo



# Directives



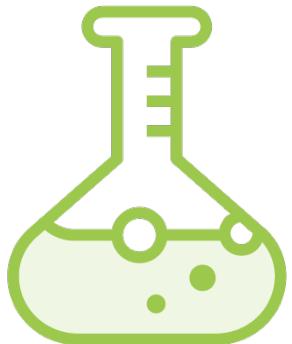
# Pipes

---

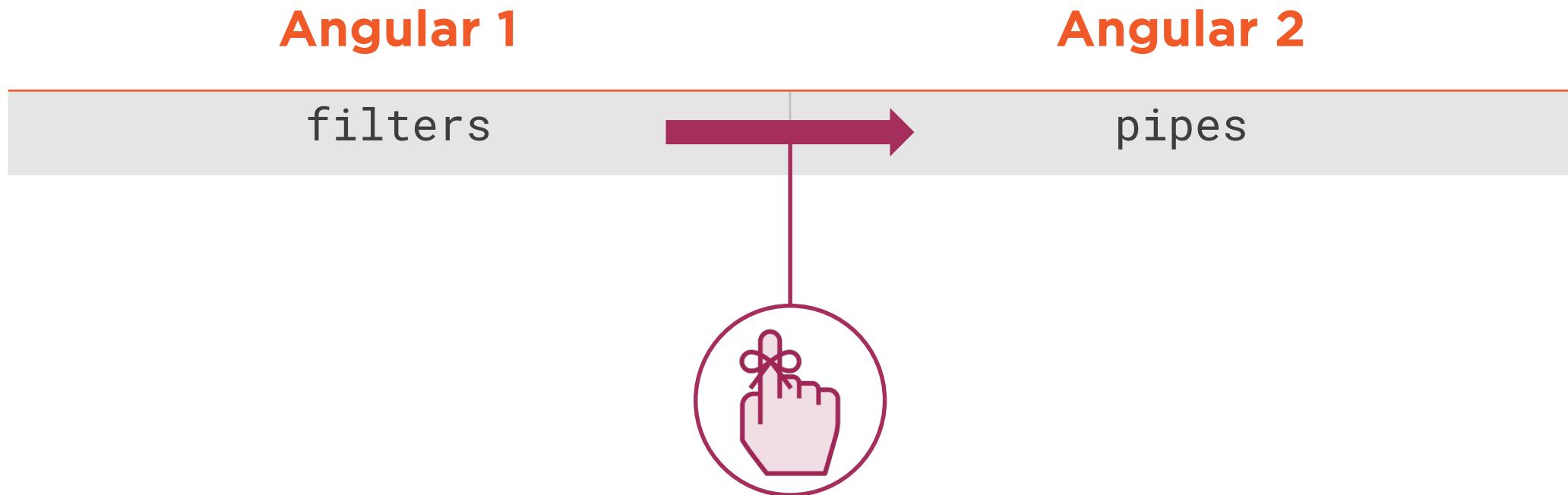


# Pipes

Pipes allow us to transform data for display in a Template.



# Angular Formatters



## String Pipes

```
<p>{{character.name | uppercase}}</p>
<p>{{character.name | lowercase}}</p>
```

Lowercase Pipe

## Built-in Pipes

Format a value in a Template



## Date formatting

```
<p>{{eventDate | date:'medium' }}</p>
<p>{{eventDate | date:'y MMMM d' }}</p>
```

Date Pipe

Date Pipe

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

**Date accepts format**

**expression | date[:format]**



## Numeric formatting

```
<p>{{price | currency}}</p>
<p>{{value | percent:'1.1-1'}}</p>
<p>{{value | number:'1.1-3'}}</p>
```

Number Pipe

## Numeric Pipes

**Number** and **Percent** accept **digitInfo**

**Expression | number[:digitInfo]**

{minIntegerDigits}.{minFractionDigits}-{maxFractionDigits}



# Async Pipe

Subscribes to a Promise or an Observable, returning the latest value emitted



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

@Pipe({ name: 'initCaps' })
export class InitCapsPipe implements PipeTransform {
  transform(value: string, args?: any[]) {
    return value.toLowerCase()
      .replace(/(?:^|\s)[a-z]/g, m => m.toUpperCase());
  }
}
```

Implement the interface

## Custom Pipes

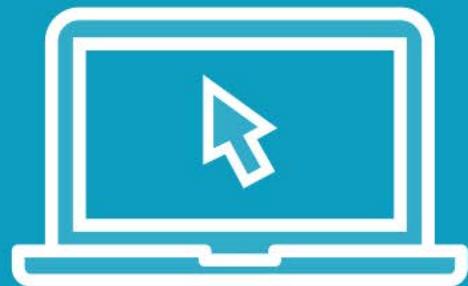
**@Pipe** decorator

**value** to transform

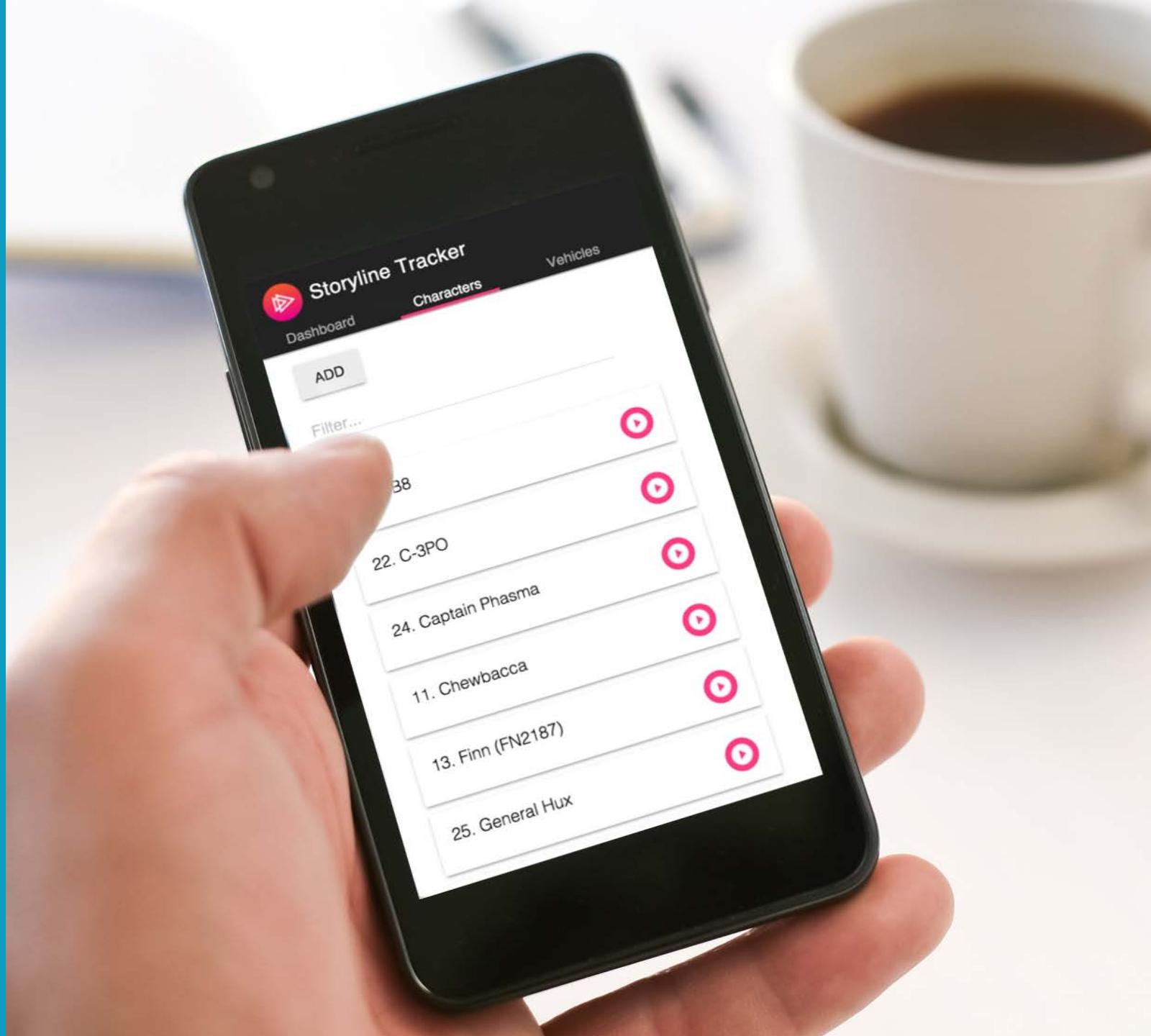
**Optional args**



# Demo



## Putting It All Together



# Template Syntax



**Data Binding**  
**Unidirectional Data Flow**  
**Attribute Directives**  
**Structural Directives**  
**Pipes**



# Services, Dependency Injection, and Component Lifecycle Hooks

---



**John Papa**

PRINCIPAL ARCHITECT

@john\_papa

[www.johnpapa.net](http://www.johnpapa.net)



# Overview



**Services**

**Dependency Injection**

**Component Lifecycle Hooks**



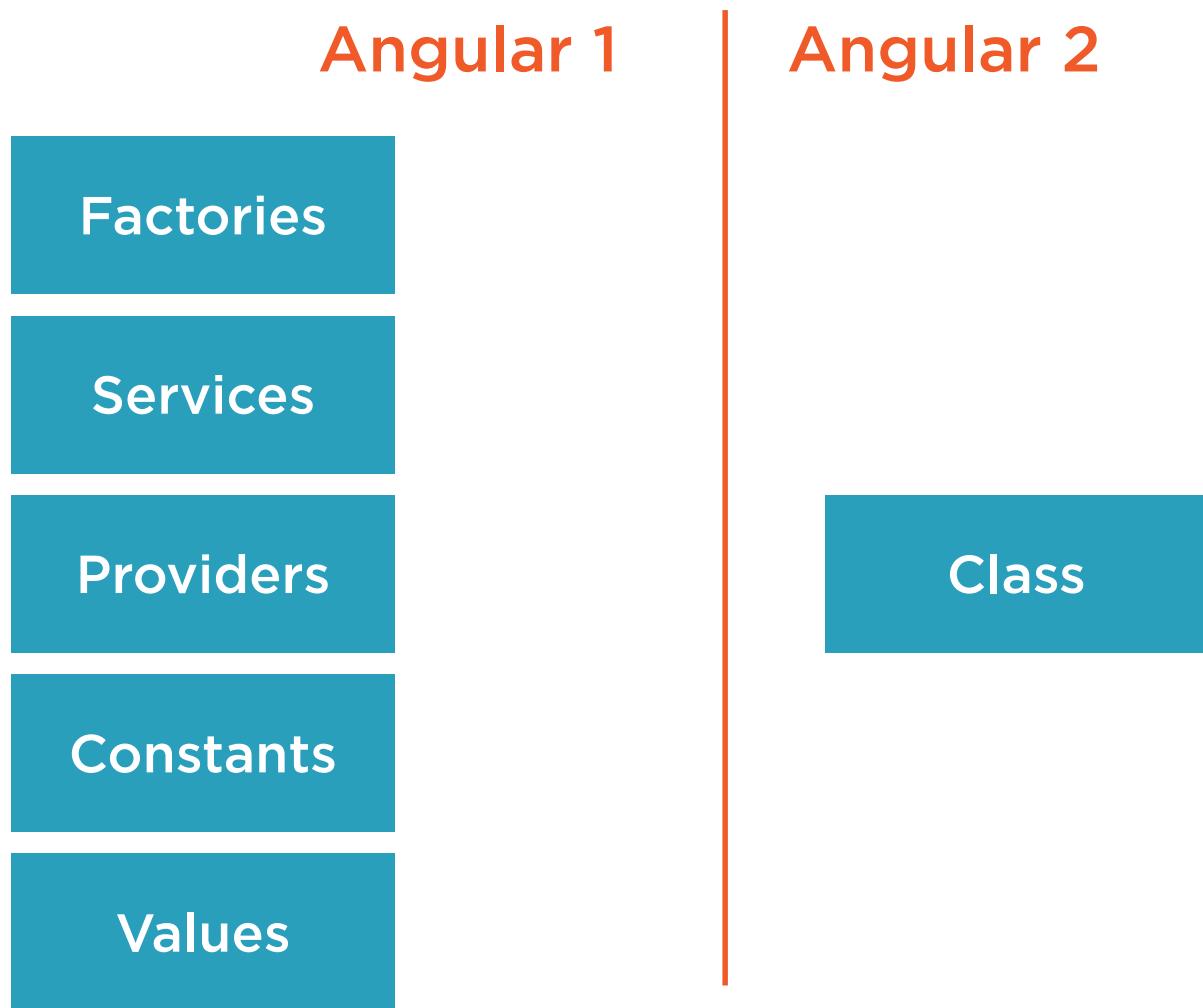


# Services

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



# Services



## 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')  
    ];  
  }  
}
```

Service is simply a class

## Service

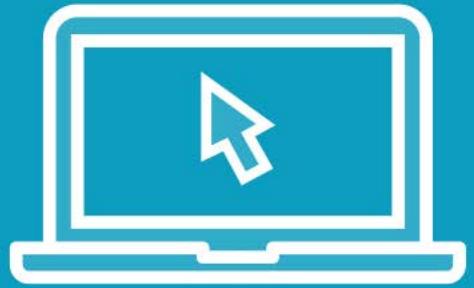
Provides something of value

Shared data or logic

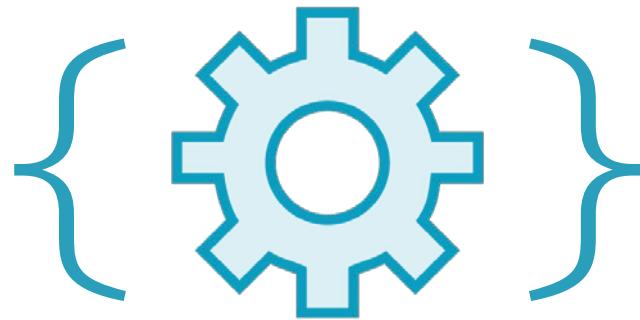
e.g. Data, logger, exception handler, or message service

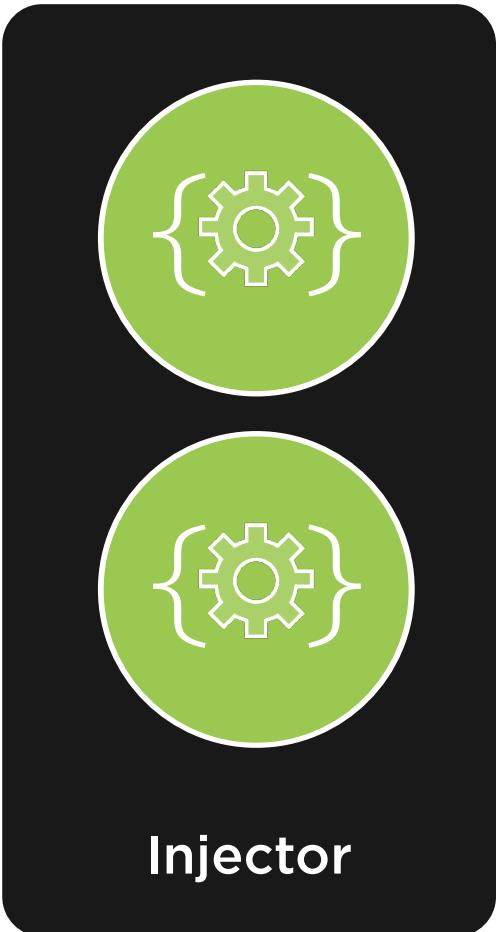


Demo



Services





---

# Dependency Injection

---



# Dependency Injection

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



```
export class VehicleListComponent {  
  vehicles: Vehicle[];  
  
  constructor(private vehicleService: VehicleService) { }  
}
```



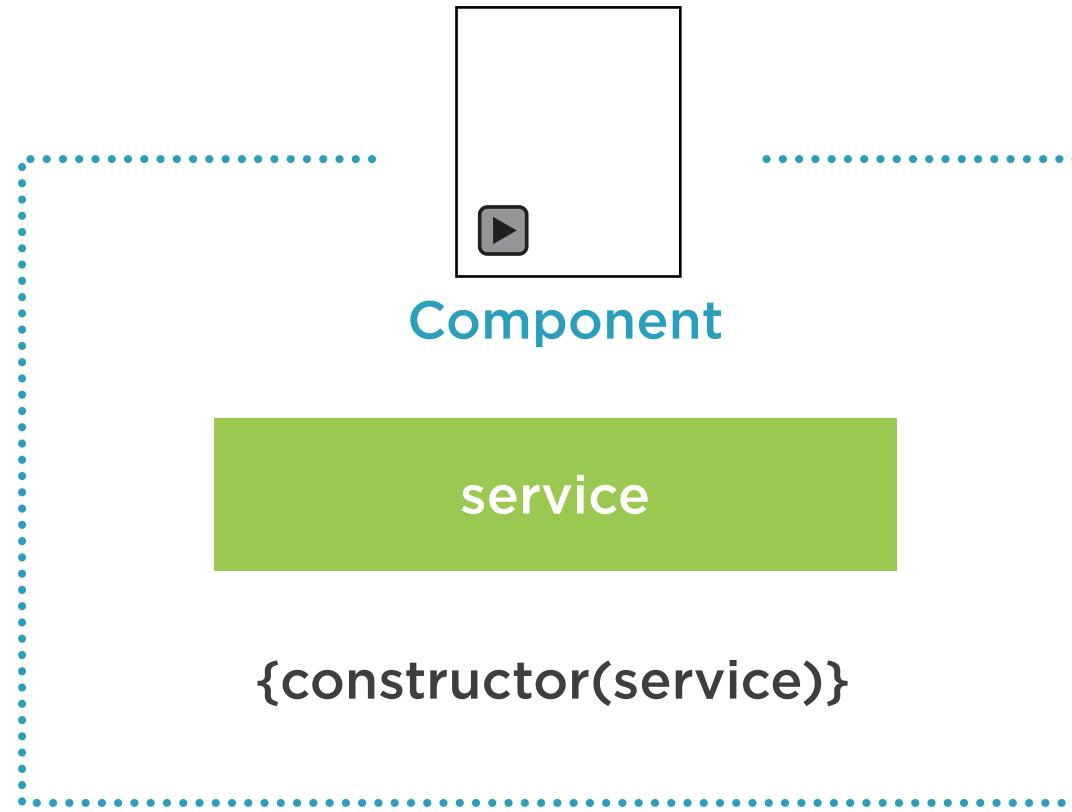
Injecting VehicleService

## Injecting a Service into a Component

Locates the service in an Angular injector

Injects the service into the constructor





Service is injected into the Component's constructor

# Dependency Injection Then and Now

## Angular 1

```
angular
  .module('app')
  .controller('VehiclesController', VehiclesController);

VehiclesController.$inject = ['VehicleService'];
function VehiclesController(VehicleService) {
  var vm = this;
  vm.title = 'Services';
  vm.vehicles = VehicleService.getVehicles();
}
```

## Angular 2

```
@Component({
  moduleId: module.id,
  selector: 'my-vehicles',
  templateUrl: 'vehicles.component.html',
})
export class VehiclesComponent {
  vehicles = this.vehicleService.getVehicles();

  constructor(private vehicleService: VehicleService) { }
}
```



## vehicle.service.ts

```
@Injectable()  
export class VehicleService {  
  constructor(private http: Http) { }  
  
  getVehicles() {  
    return this.http.get(vehiclesUrl)  
      .map((res: Response) => res.json().data);  
  }  
}
```

Provides metadata about the  
Injectables

Injecting http

## Injecting a Service into a Service

Same concept as injecting into a Component

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



We need to provide the service  
to an Angular injector



## Angular 1

```
angular
  .module('app')
  .service('VehicleService', VehicleService);

function VehicleService() {
  this.getVehicles = function () {
    return [
      { id: 1, name: 'X-Wing Fighter' },
      { id: 2, name: 'Tie Fighter' },
      { id: 3, name: 'Y-Wing Fighter' }
    ];
  }
}
```

Providing a service

# Providing Services in Angular 1

**Angular 1 has one global injector**

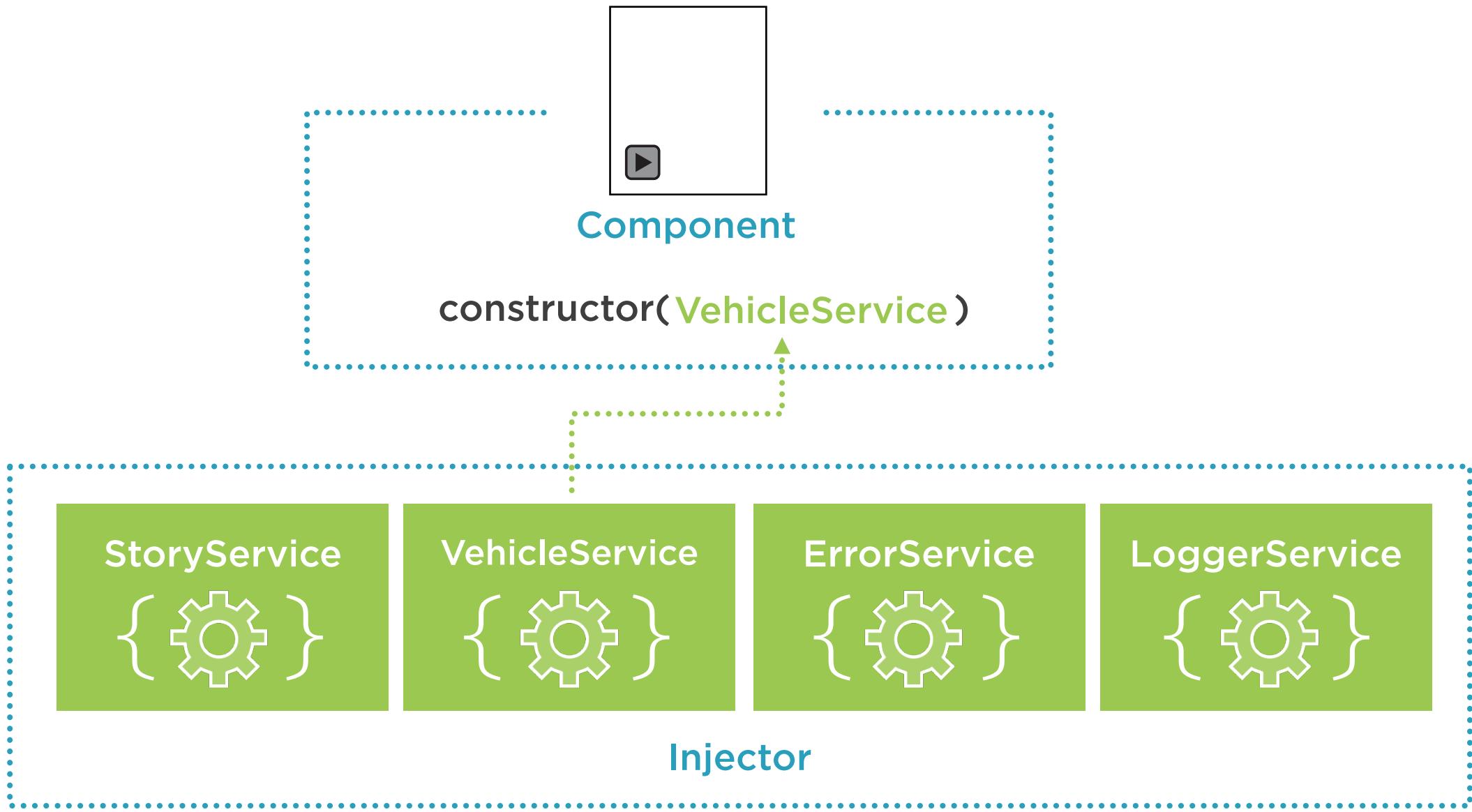
**Angular 2 has hierarchical injectors and an injector at the app root**



# Providing a Service in Angular 2

The Service is now  
available in the root  
application injector

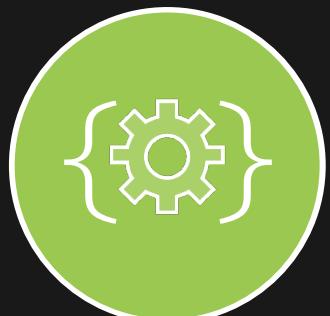
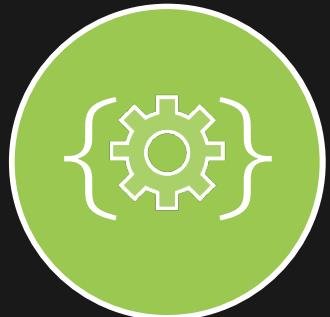
```
@NgModule({  
  imports: [BrowserModule, FormsModule],  
  declarations: [VehiclesComponent],  
  providers: [VehicleService],  
  bootstrap: [VehiclesComponent],  
})  
export class AppModule { }
```



# Injectors

---

Injector



## We provide services to Angular's Injectors

When we inject a service, Angular searches the appropriate injectors for it



One injector for the application root

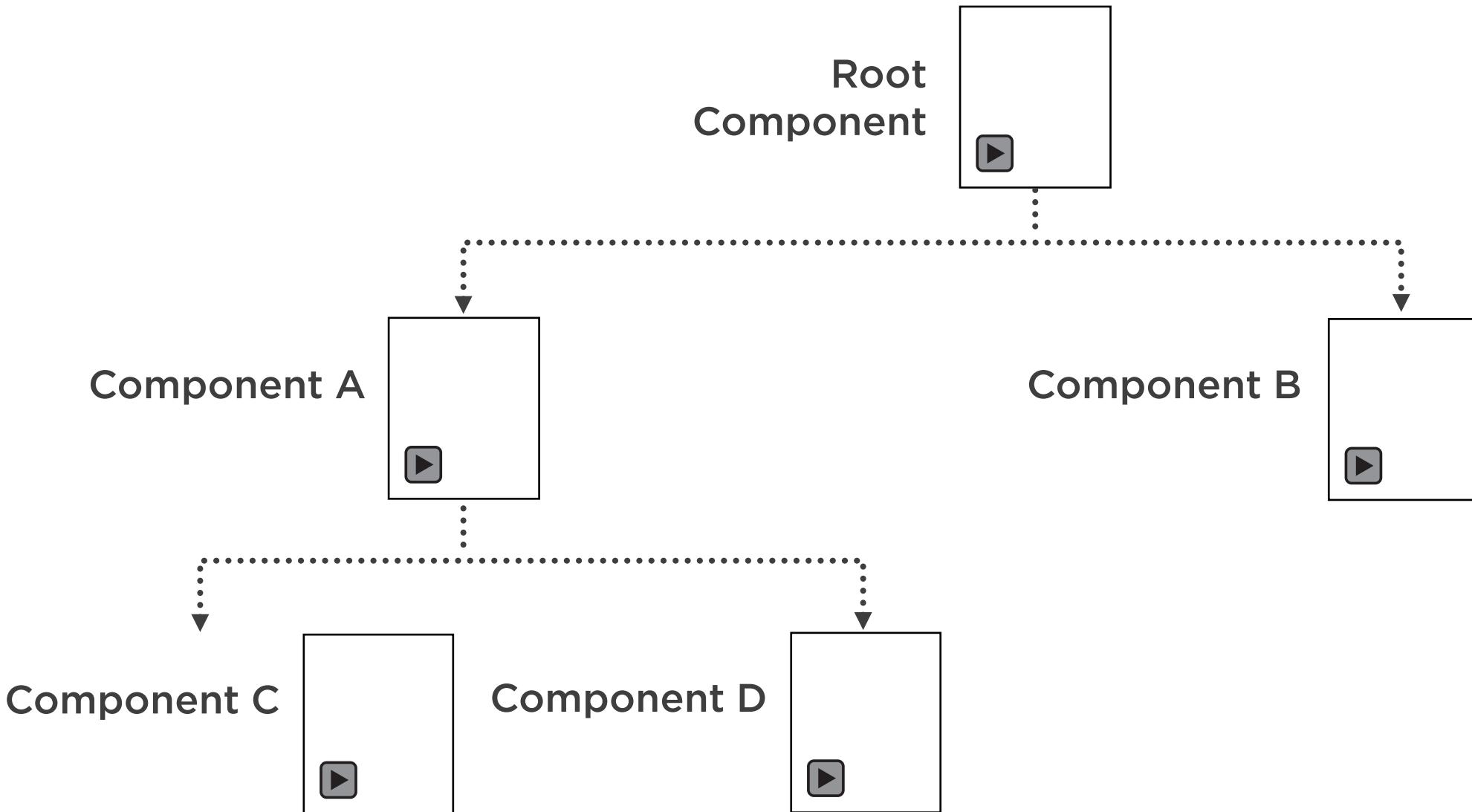


One injector for the application root

And a hierarchical DI system with a tree of injectors that parallel an application's component tree



# Hierarchical Components, Hierarchical Injectors



So where do we set the providers?

In the **Component** or an **Angular Module**?



# Providing in a Component

Available to this Component and any in its tree

```
@Component({
  moduleId: module.id,
  selector: 'story-vehicles',
  templateUrl: 'vehicles.component.html',
  providers: [VehicleService]
})
export class VehiclesComponent {
  // ...
}
```

# Providing in an Angular Module

Eagerly and lazily-loaded modules and their components can inject the root AppModule services

```
@NgModule({
  imports: [BrowserModule, FormsModule],
  declarations: [VehiclesComponent],
  providers: [VehicleService],
  bootstrap: [VehiclesComponent],
})
export class AppModule { }
```

## vehicles.component.ts

```
@Component({  
  moduleId: module.id,  
  selector: 'story-vehicles',  
  templateUrl: 'vehicles.component.html',  
  providers: [VehicleService]  
})  
export class VehiclesComponent {  
  // ...  
}
```

Providing to  
Component

## app.module.ts

```
@NgModule({  
  imports: [BrowserModule, FormsModule],  
  declarations: [VehiclesComponent],  
  providers: [VehicleService],  
  bootstrap: [VehiclesComponent],  
})  
export class AppModule { }
```

Providing to NgModule

Prefer registering providers in Angular Modules

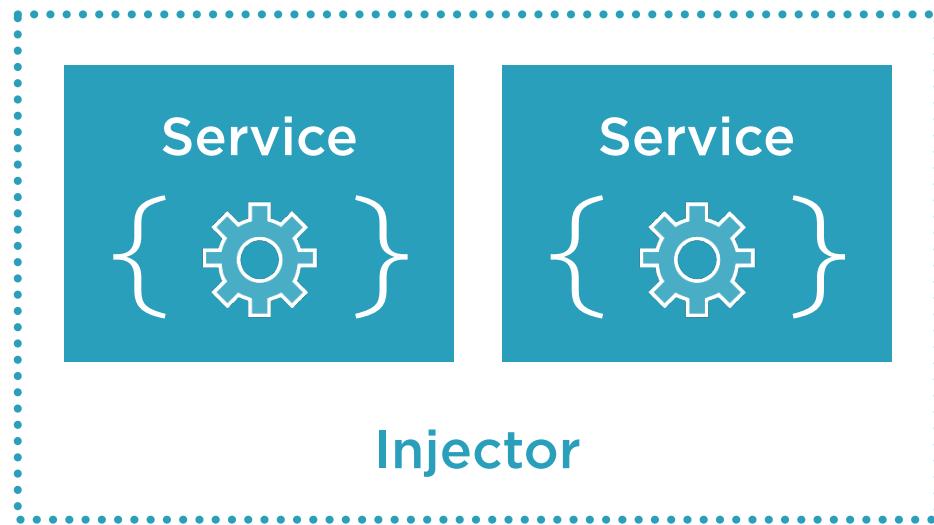
Provide a service once,  
if you want a singleton

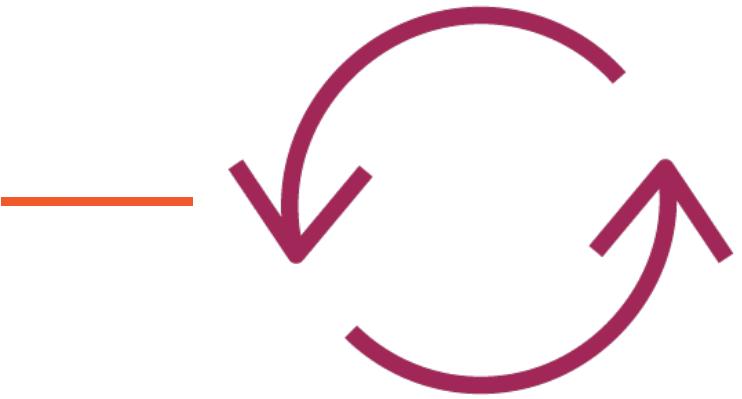


Demo



# Injectors





## Component Lifecycle Hooks

---



# Component Lifecycle Hooks

Lifecycle Hooks allow us to tap into specific moments in the application lifecycle to perform logic.



# Interface

Implement the lifecycle hook's OnInit interface

# Lifecycle Hooks

When the Component initializes, the ngOnInit function is executed

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

  constructor(private characterService: CharacterService) { }

  ngOnInit() {
    this.characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

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



# Component Lifecycle Hooks



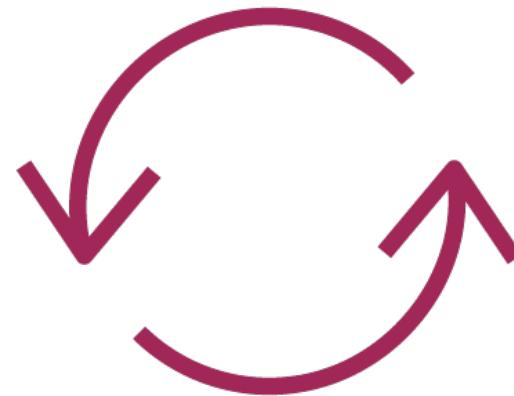
The Lifecycle Interface helps enforce the valid use of a hook



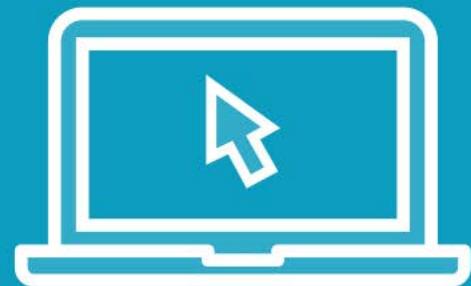
Demo



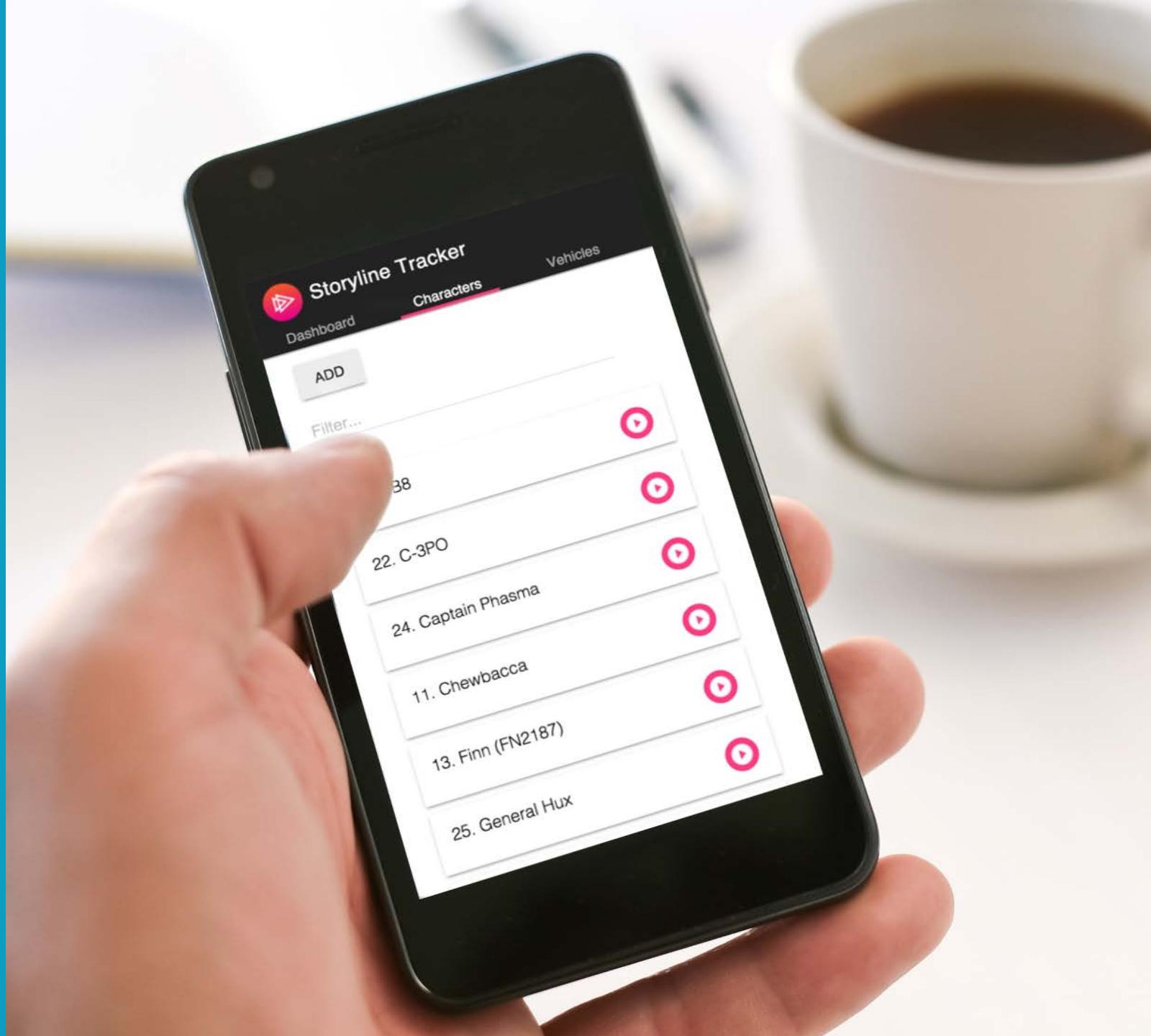
# Component Lifecycle Hooks



# Demo



## Putting It All Together



# Services, DI, and LifeCycle Hooks



**Separation with Services**

**Sharing Instances**

**Registering with the Injector**

**Constructor Injection**

**Tapping into the Component's LifeCycle**



# Data with Http

---



**John Papa**

PRINCIPAL ARCHITECT

@john\_papa

[www.johnpapa.net](http://www.johnpapa.net)



# Overview



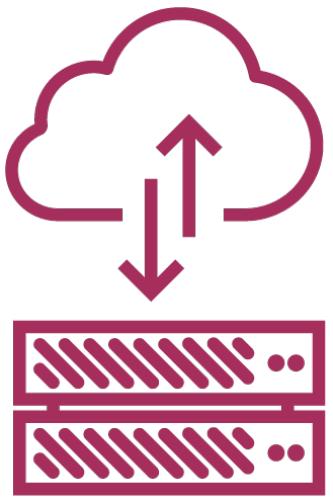
**Http**

**RxJS, Observables, and Subscriptions**

**Async Pipe**

**Promises**





Http

# Http

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



# Http Then and Now

## Angular 1

```
this.getVehicles = function() {
  return $http.get('api/vehicles')
    .then(function(response) {
      return response.data.data;
    })
    .catch(handleError);
}
```

## Angular 2

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



## Providers

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/http';
import './rxjs-extensions';
import { AppComponent } from './app.component';
import { VehicleListComponent } from './vehicle-list.component';
```

Located in module @angular/http

```
@NgModule({
  imports: [BrowserModule, HttpClientModule],
  declarations: [AppComponent, VehicleListComponent],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Import the Http module

## Http Requirements

We need the **HttpClient** to make Http calls



## vehicle.service.ts

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

  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');
  }
}
```

Make and return the async GET call

Map the response

Handle any exception



## vehicle-list.component.ts

```
constructor(private vehicleService: VehicleService) { }
getHeroes() {
  this.vehicleService.getVehicles()
    .subscribe(
      vehicles => this.vehicles = vehicles,
      error => this.errorMessage = <any>error
    );
}
ngOnInit() { this.getHeroes(); }
```

Subscribe to the observable

Success and failure cases

Subscribing to the Observable

Component is handed an **Observable**

We **Subscribe** to it



1

2

3

Http Step by Step



# Http Step by Step

1

2

3

1

Import  
HttpModule



# Http Step by Step

1

2

3

1

Import  
HttpModule

2

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



# Http Step by Step

1

1  
Import  
HttpModule

2

2  
Call Http.get in a  
Service and  
return the  
mapped result

3

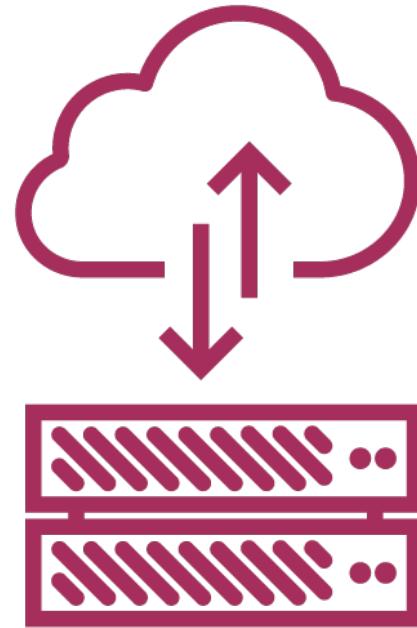
3  
Subscribe to the  
Service's  
function in the  
Component

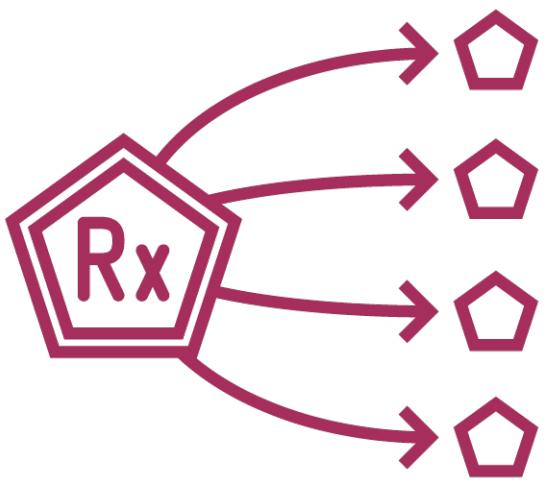


Demo



Http





RxJs



# RxJs

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



```
import 'rxjs/Rx';
```

Imports all of RxJs

## Importing RxJs

RxJs is a large library

For production, only import the modules you require



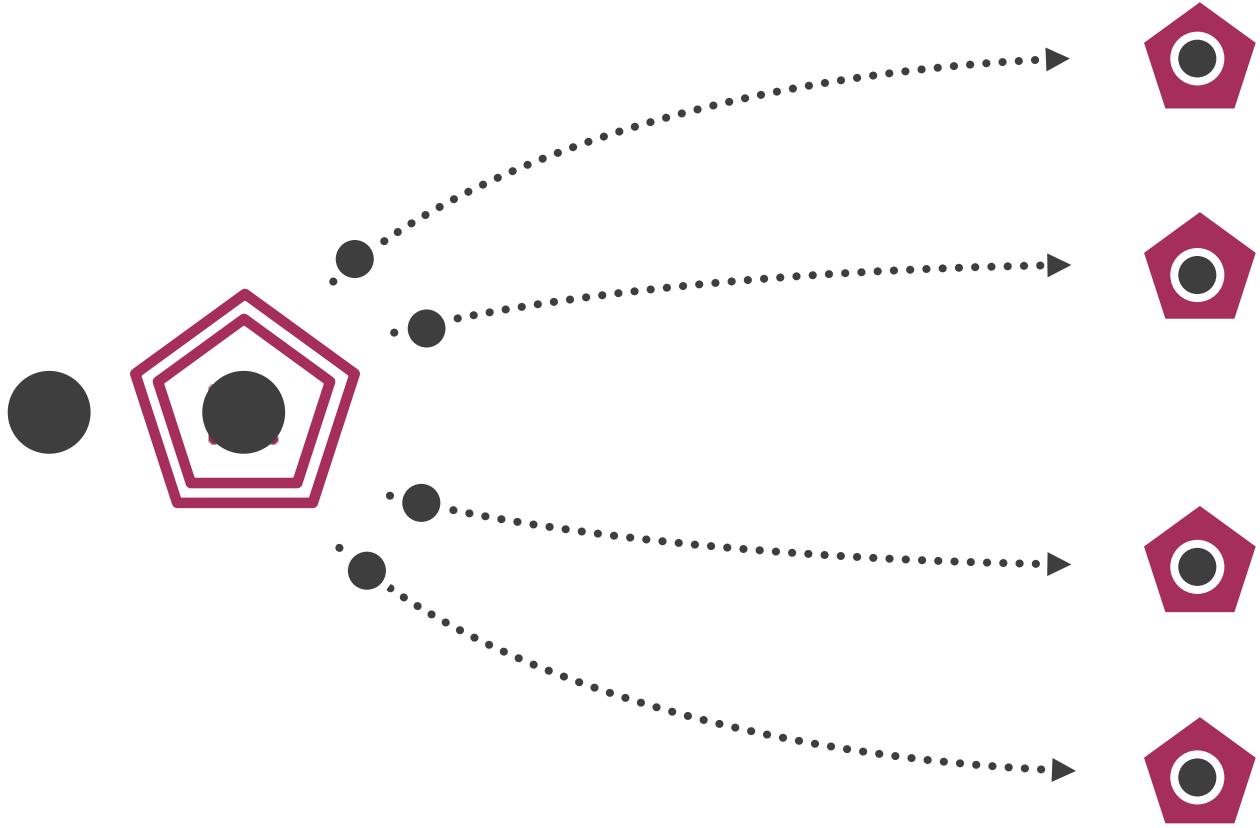
## app.module.ts

```
import './rxjs-extensions';
```

## rxjs-extensions.ts

```
import 'rxjs/add/operator/catch';
import 'rxjs/add/operator/do';
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/toPromise';
```

Prefer importing only what we use from RxJs



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

Retrieve the JSON

**data** is what we defined on the server

## Returning from Http

We don't return the response

Service does the 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);  
  let msg = `Error status code ${error.status} at ${error.url}`;  
  return Observable.throw(msg);  
}
```

Catch

## Exception Handling

We catch errors in the Service

We pass some 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

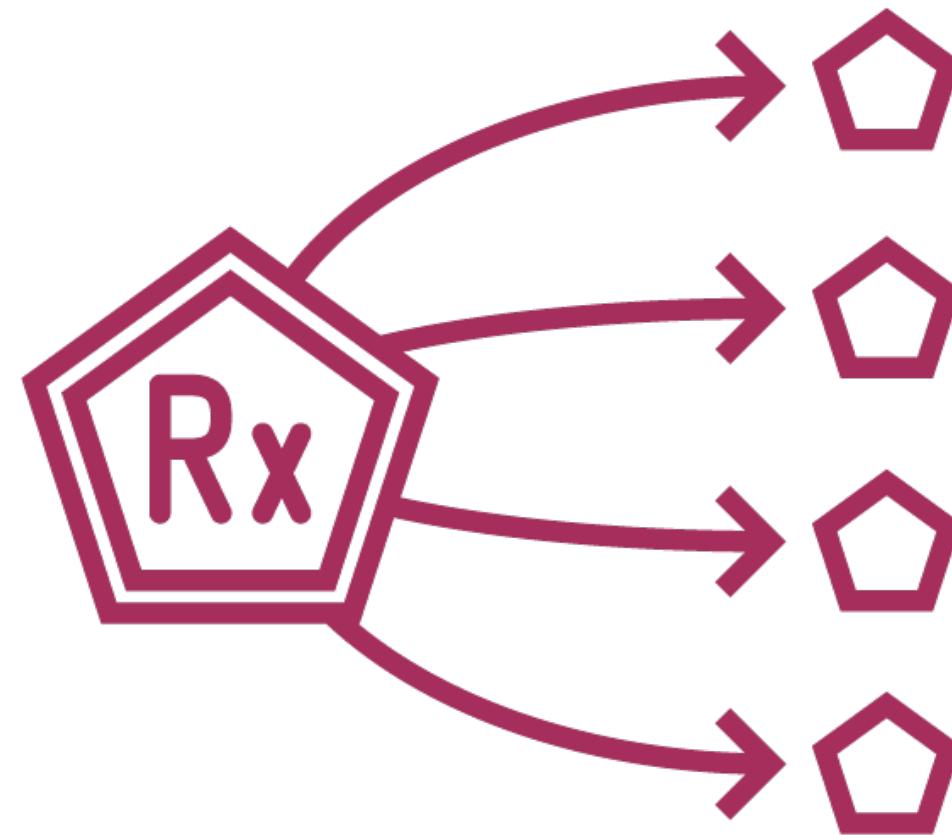
Subscribing to the Observable Component is handed an **Observable**

We **Subscribe** to it

We can handle errors here, for presenting to the user if we wish



# RxJs



1101010  
0101100

---



Async Pipe

---



# 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

```
export class VehicleListComponent {  
  vehicles: Observable<Vehicle[]>;  
  constructor(private vehicleService: VehicleService) { }  
  getVehicles() {  
    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



## vehicle-list.component.html

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

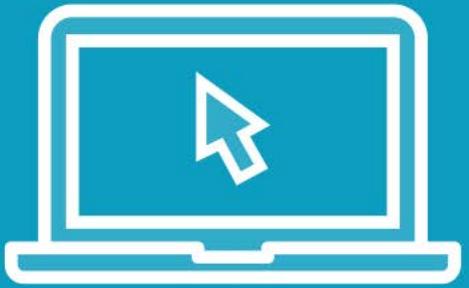
Subscribes to the Vehicles

# Async Pipe in the Template

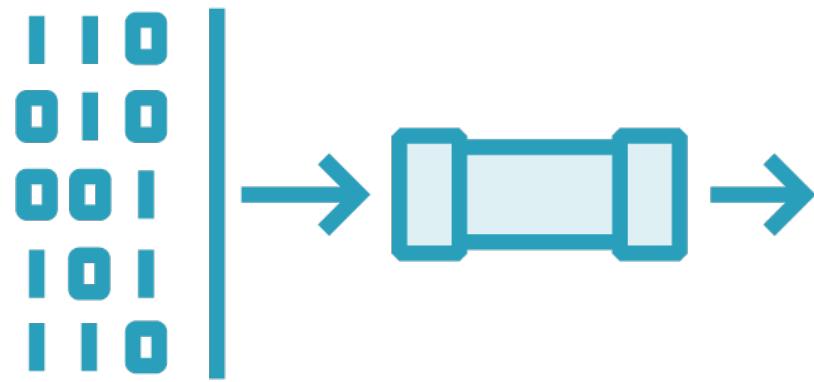
## Apply the **async** Pipe



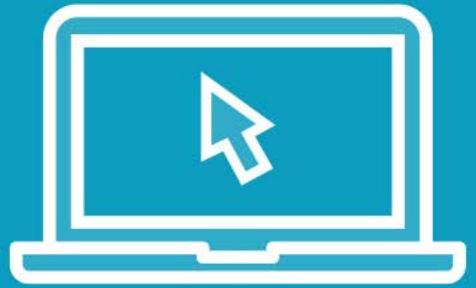
Demo



Async



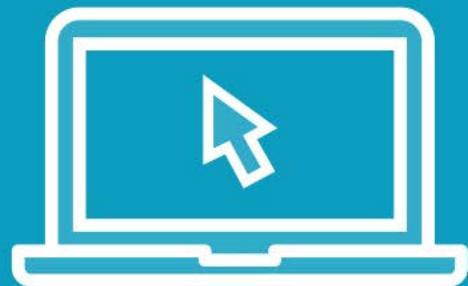
Demo



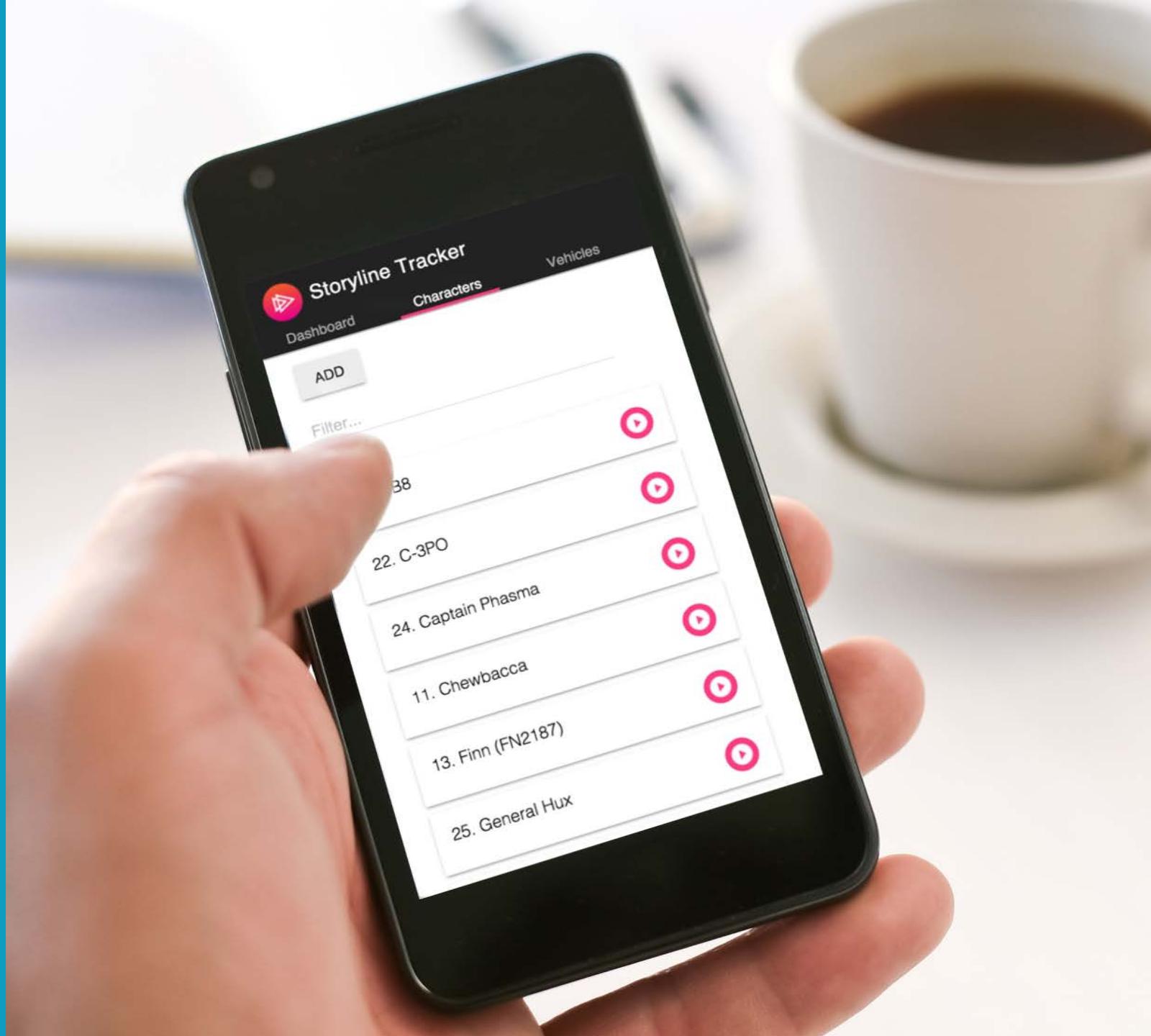
Promises



# Demo



## Putting It All Together



# Http



## Http

**Observables and Subscriptions**

**Async Pipe**

**Promises**



# Routing

---



**John Papa**

PRINCIPAL ARCHITECT

@john\_papa

[www.johnpapa.net](http://www.johnpapa.net)



# Overview



**Defining Routes**

**Routing Modules**

**Route Parameters**

**Guards**



# Routing Basics

Routing allows our application to navigate between different Components, passing parameters where needed



# Routing Then and Now

**Angular 1**

**Angular 2**

```
<base href="/">
```

Define the `<base>` element  
**Required for routing to work properly**



1

2

3

4

5

Routing, Step by Step



1

2

3

4

5

# Routing, Step by Step

1

Import  
RouterModule



1

# Routing, Step by Step

2

1

Import  
RouterModule

2

Import  
@angular/router

3

4

5



1

# Routing, Step by Step

2

1  
Import  
RouterModule

3

2  
Import  
@angular/router

3  
Define the routes

4

5



1

# Routing, Step by Step

2

1

Import  
RouterModule

3

2

Import  
@angular/router

3

Define the routes

4

4

Declare a  
<router-outlet>

5



1

# Routing, Step by Step

2

1

Import  
RouterModule

2

Import  
@angular/router

3

Define the routes

3

4

4

Declare a  
<router-outlet>

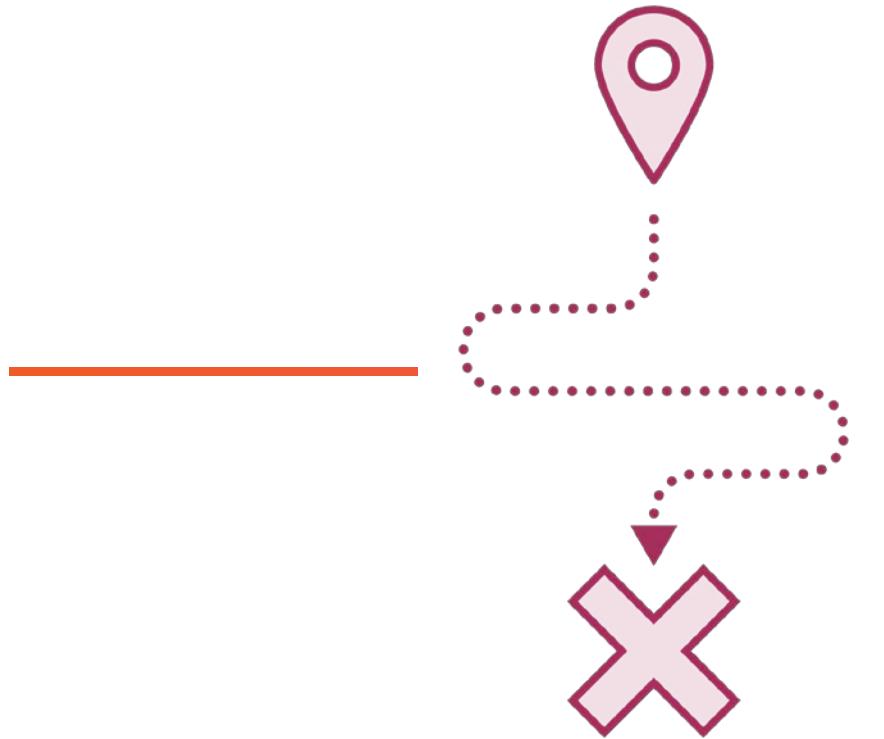
5

5

Add  
[routerLink]  
bindings



# Routing



```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
```

Import routing features

## Import RouterModule

**RouterModule** gives us access to routing features

**Routes** help us declare or route definitions



```
const routes: Routes = [
  { path: '', pathMatch: 'full', redirectTo: 'characters', },
  { path: 'characters', component: CharacterListComponent },
  { path: 'character/:id', component: CharacterComponent },
  { path: '**', pathMatch: 'full', component: PageNotFoundComponent },
];
```

## Defining Routes

Define the route's **path**

Indicate parameters with :

Set the **component** that we'll route to



## app-routing.module.ts

```
@NgModule({  
  imports: [RouterModule.forRoot(routes)],  
  exports: [RouterModule]  
})  
export class AppRoutingModule { }
```

Only use `forRoot()` for the root module's routes

## Define a Module

Create a routing module using our routes, and import it

Export our new `AppRoutingModule`



## app-routing.module.ts

```
const routes: Routes = [  
  { path: '', pathMatch: 'full', redirectTo: 'characters', },  
  { path: 'characters', component: CharacterListComponent },  
  { path: 'character/:id', component: CharacterComponent },  
  { path: '**', pathMatch: 'full', component: PageNotFoundComponent }  
];
```

Configure routes

```
@NgModule({  
  imports: [RouterModule.forRoot(routes)],  
  exports: [RouterModule]  
})  
export class AppRoutingModule {}  
  
export const routableComponents = [  
  CharacterListComponent,  
  CharacterComponent,  
  PageNotFoundComponent  
];
```

Use route to make a NgModule

Create and export an explicitly named NgModule

Export components

Export the Module and the Components

We define a Routing  
Module, and import it into  
the App Root or Feature  
Module



## app.module.ts

```
import { AppComponent } from './app.component';
import { AppRoutingModule, routableComponents } from './app-routing.module';

@NgModule({
  imports: [BrowserModule, AppRoutingModule],
  declarations: [AppComponent, routableComponents]
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Import the module

Declare the components

## Using Our Routing Module

**Import our routing module**

**Declare the components**



EXPLORER

app.component.html x

...

# Routing in a Template

OPEN EDITORS

A2-FIRST-LOOK

router-guard

router-lazy

api

app

characters

vehicles

app.component.html

app.component.ts

app.module.ts

app.routing.ts

config.ts

page-not-found.component.ts

rxjs-extensions.ts

node\_modules

typings

.editorconfig

{} example-config.json

index.html

karma-test-shim.js

karma.conf.js

main.ts

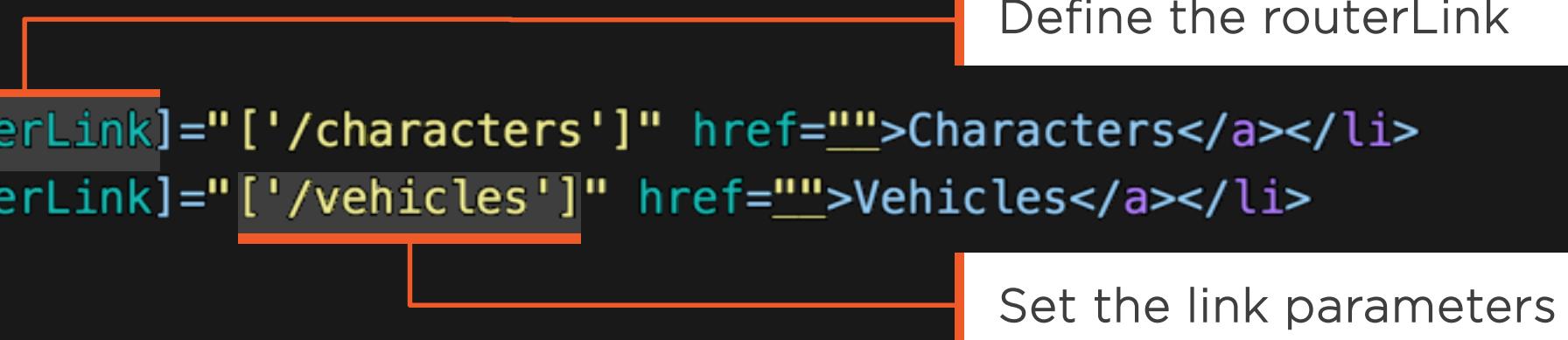
```
1  <div>
2    <header>
3      <h1>Storyline Tracker</h1>
4      <h3>Lazily Loaded Router Demo</h3>
5      <nav>
6        <ul>
7          <li><a [routerLink]=["/characters"] href="">Characters</a></li>
8          <li><a [routerLink]=["/vehicles"] href="">Vehicles</a></li>
9        </ul>
10       </nav>
11     </header>
12     <main>
13       <section>
14         <router-outlet></router-outlet>
15       </section>
16     </main>
17   </div>
18
```

# RouterLink

The **RouterLink** directive navigates to a route path

We define the link parameters array

```
<nav>
  <ul>
    <li><a [routerLink]=["/characters"] href="">Characters</a></li>
    <li><a [routerLink]=["/vehicles"] href="">Vehicles</a></li>
  </ul>
</nav>
```



Define the routerLink

Set the link parameters

```
<router-outlet></router-outlet>
```

## Using the RouterOutlet

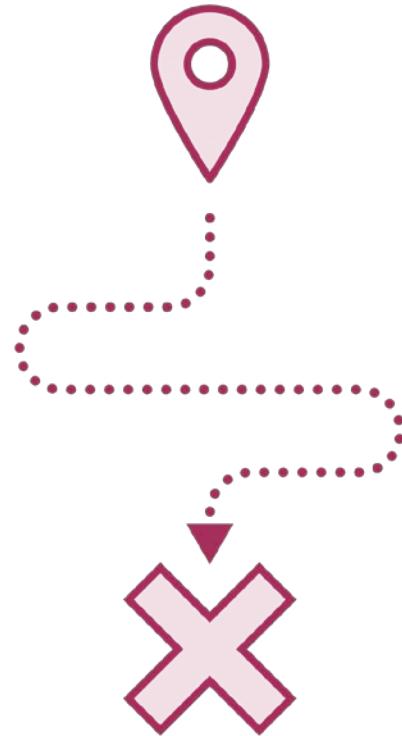
**Defines where to put the components when we route**

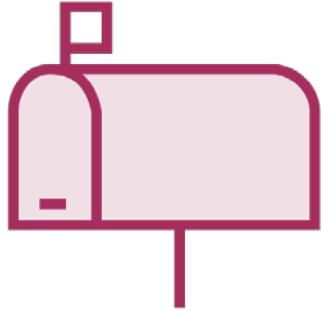


Demo



# Routing





## Routing Parameters



```
const routes: Routes = [
  { path: '', pathMatch: 'full', redirectTo: 'characters', },
  { path: 'characters', component: CharacterListComponent },
  { path: 'character/:id', component: CharacterComponent },
  { path: '**', pathMatch: 'full', component: PageNotFoundComponent },
];
```

Define the id parameter

## Defining Parameters

Indicate parameters with :



# Passing Data Through Routing

## Snapshot

Easiest, as long  
as parameter  
values do not  
change

## Observable

Gets new  
parameter values  
when component  
is re-used

## Resolvers

Gets data before  
a component  
loads



## session.component.ts

```
export class SessionComponent implements OnInit {  
  private id: any;  
  
  constructor(private route: ActivatedRoute) { }  
  
  ngOnInit() {  
    this.id = parseInt(this.route.snapshot.params['id']);  
    this.getSession();  
  }  
}
```

Inject the ActivatedRoute

Grab the parameter

## Snapshots

Grab the ActivatedRoute

Access the snapshot parameter by name

Easiest, if component is not reused



# ActivatedRoute and Observables

**route.params** is an Observable

Map the response, perform side effects, and subscribe

```
export class SessionComponent implements OnInit {
  private id: any;

  constructor(private route: ActivatedRoute) { }

  ngOnInit() {
    this.route.params.map(params => params['id'])
      .do(id => this.id = parseInt(id))
      .subscribe(id => this.getSession());
  }
}
```

Map each response

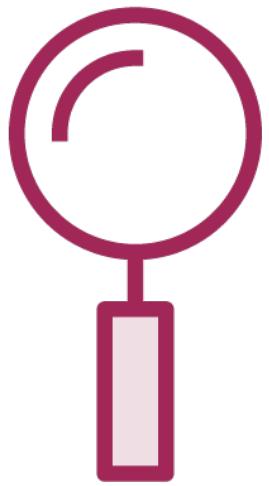
Subscribe to the stream

Demo



# Routing Parameters





## Routing Resolvers



# Resolvers

Get data prior to traversing the route



## vehicle-resolver.service.ts

```
@Injectable()
export class VehicleResolver implements Resolve<Vehicle> {
  constructor(
    private vehicleService: VehicleService,
    private router: Router) { }

  resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
    let id = +route.params['id'];
    return this.vehicleService.getVehicle(id)
      .map(vehicle => vehicle ? vehicle : new Vehicle())
      .catch((error: any) => {
        console.log(`$${error}. Heading back to vehicle list`);
        this.router.navigate(['/vehicles']);
        return Observable.of(null);
      });
  }
}
```

Grab the route and its parameter

Return the vehicle

Service that operates in the midst of a routing action

```
{  
  path: 'vehicles:id',  
  component: VehicleComponent,  
  resolve: {  
    vehicle: VehicleResolver  
  }  
},
```

Define one or more resolvers

## Resolvers

Defined within the route configuration

Remember to provide the resolver service



```
this.route.data.subscribe(data: { vehicle: Vehicle }) => this.vehicle = data.vehicle);
```

Subscribe to the data

Grab the resolved vehicle

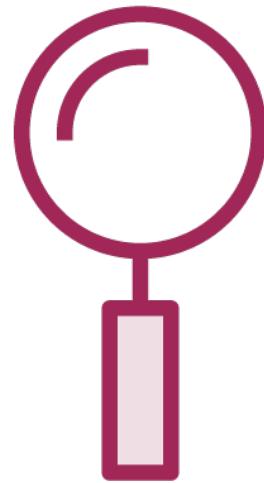
## Getting Resolver Data

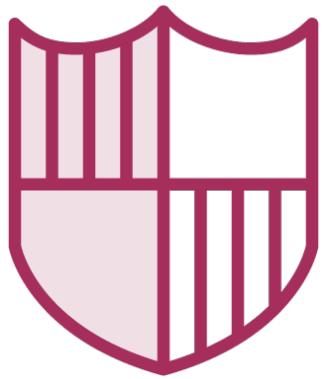
**Subscribe to the `data` property on the `ActivatedRoute`**



# Routing Resolvers

Demo





## Routing Guards



# Guards

Guards allow us to make a decision at key points in the routing lifecycle and either continue, abort or take a new direction





## Types of Router Guards





# Types of Router Guards



Resolve





# Types of Router Guards



Resolve



CanActivate





# Types of Router Guards



Resolve



CanActivate



CanActivateChild





# Types of Router Guards



Resolve



CanActivate



CanActivateChild



CanDeactivate





# Types of Router Guards



Resolve



CanActivate



CanActivateChild



CanDeactivate



CanLoad



## auth-guard.service.ts

```
@Injectable()
export class AuthGuard implements CanActivate {
  constructor(private userProfileService: UserProfileService, private router: Router) { }

  canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
    if (this.userProfileService.isLoggedIn) {
      return true;
    }
    this.router.navigate(['/login'], { queryParams: { redirectTo: state.url } });
    return false;
  }
}
```

Implement the interface

Return true or false

## CanActivate Guard

Implement the **CanActivate** interface

Make a determination if the route should be activated

Can re-navigate elsewhere



## app-routing.module.ts

```
{  
  path: 'dashboard',  
  component: DashboardComponent,  
  canActivate: [AuthGuard]  
},
```

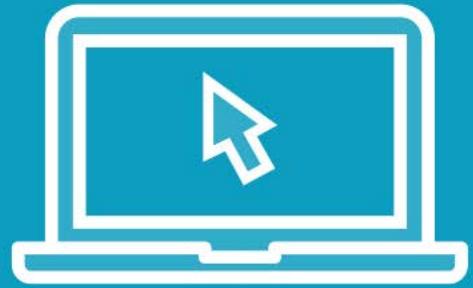
Apply the guard(s)

## Applying a Guard

### Apply to the route



Demo



# Routing Guards



# Child Routes

A Component may define routes for other Components.  
This creates a series of hierarchical child routes.



## app-routing.module.ts

```
const routes: Routes = [
  { path: '', pathMatch: 'full', redirectTo: 'characters', },
  { path: 'login', component: LoginComponent },
  {
    path: 'characters',
    component: CharactersComponent,
    canActivate: [CanActivateAuthGuard],
    children: [
      { path: '', component: CharacterListComponent },
      { path: ':id', component: CharacterComponent },
    ]
  },
]
```

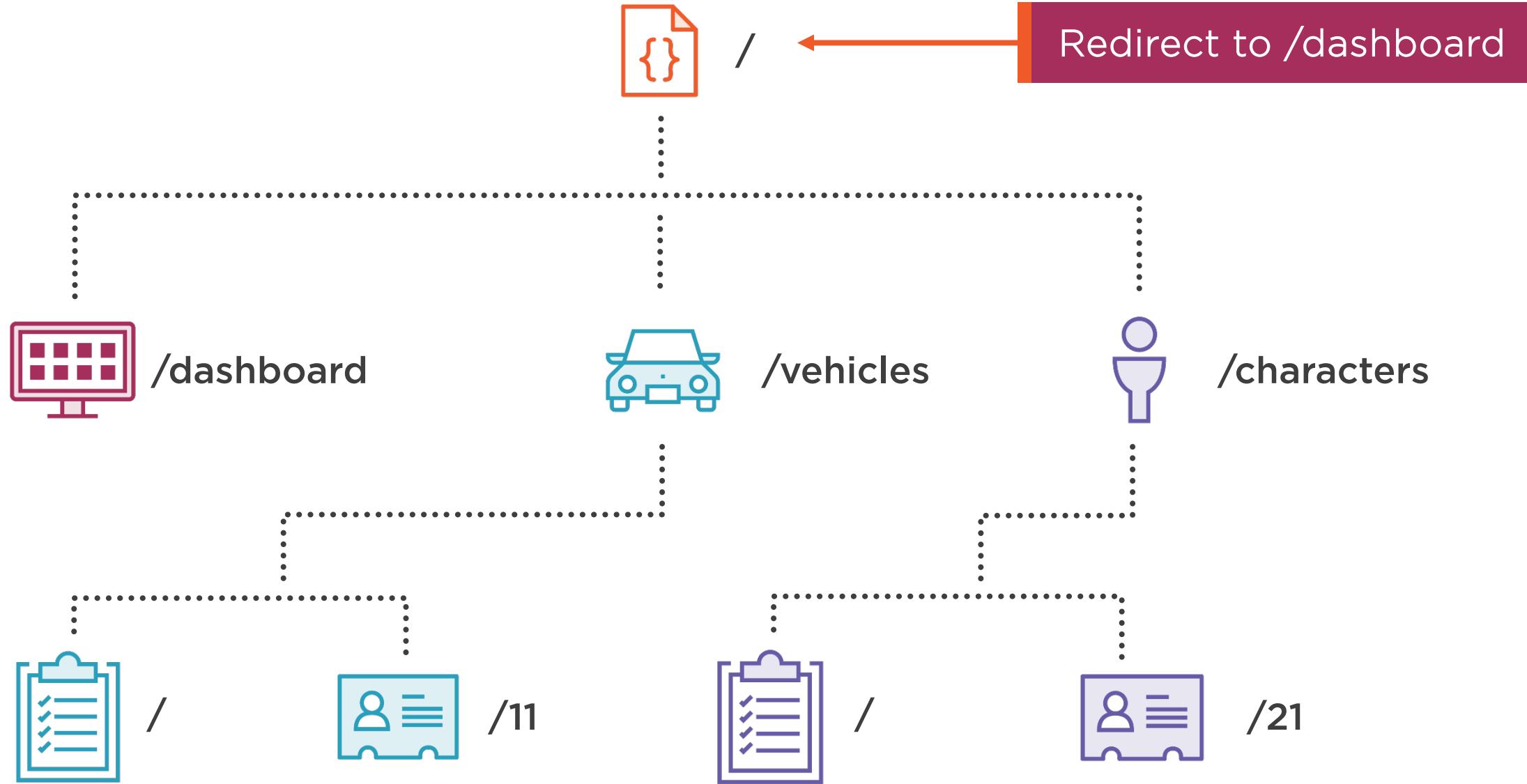
Child routes

## Child Routes

URLs for parent child

e.g. **characters/** and **characters/72**





Notice where we provided?



# Module Providers

Provided to the root injector

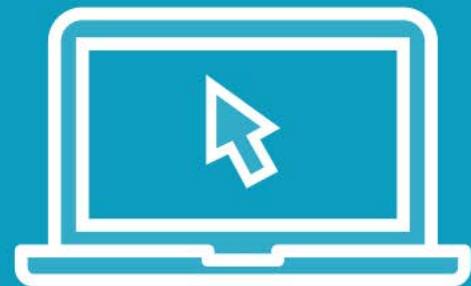
Available everywhere

```
@NgModule({
  imports: [...],
  declarations: [AppComponent, routableComponents],
  providers: [
    CanActivateAuthGuard, CharacterService, UserProfileService, VehicleService
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

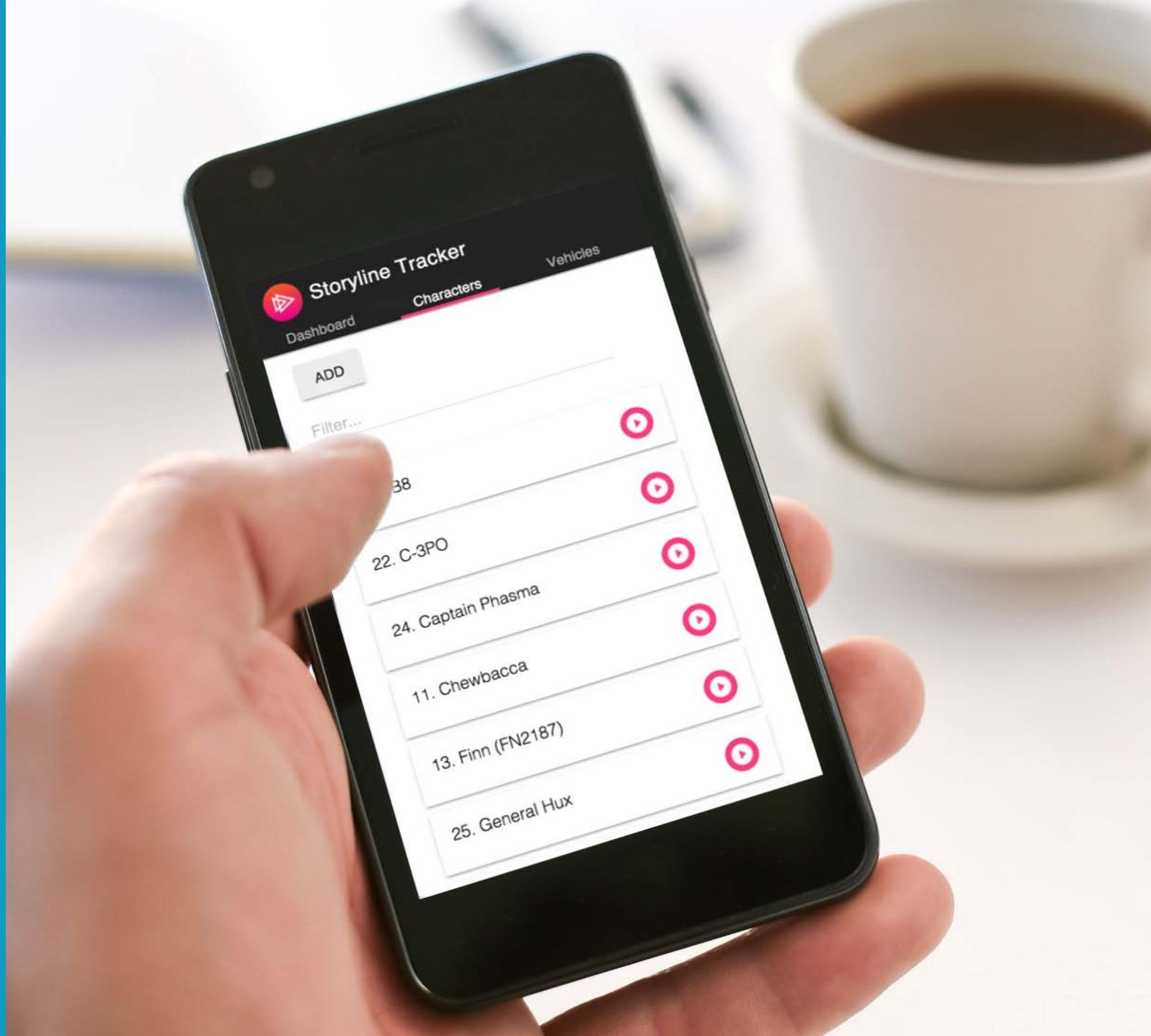
Provide to the root injector

app.module.ts

# Demo



## Putting It All Together



# Routing



**Defining Routes**  
**Routing Modules**  
**Route Parameters**  
**Guards**  
**Child Routes**



# Angular Modules

---



**John Papa**

PRINCIPAL ARCHITECT

@john\_papa

[www.johnpapa.net](http://www.johnpapa.net)



# Overview



## Routed Modules

- Eager Loading
- Lazy Loading

## Preload Strategies

## Feature Modules

## Providers



# Angular Modules

Help organize our applications into cohesive blocks of functionality



# Basic Types of Feature Modules



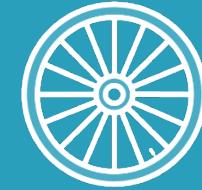
Domain



Routed

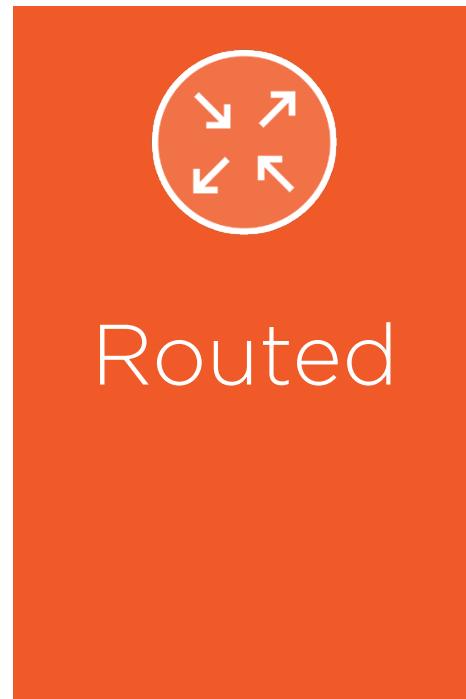


Service

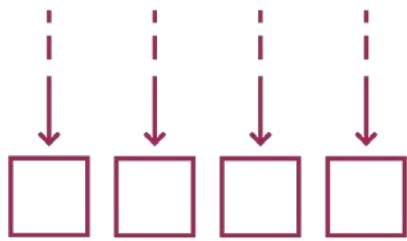


Widget

# Routed Modules Are Eagerly or Lazily Loaded



## Eager Loading Modules



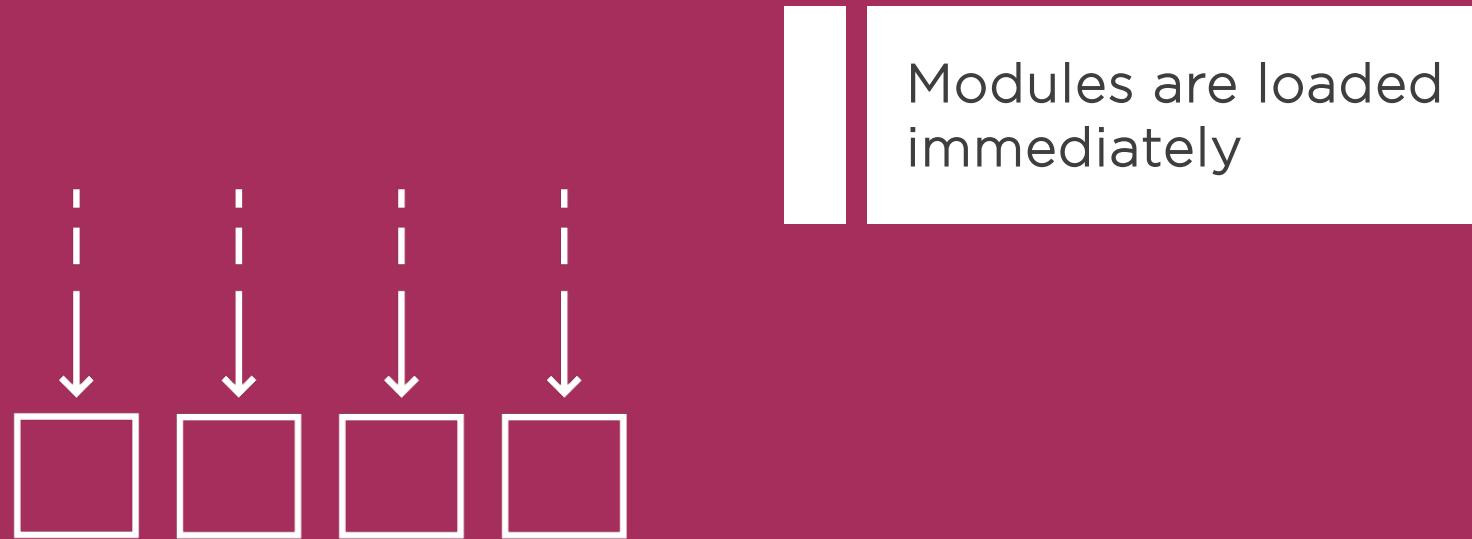
# Eager Loading

Loading a feature module at startup, with the root module.

Begin by creating a **feature module** with routing.



# Eagerly Loading the Modules



## characters.module.ts

```
import {  
  CharactersRouterModule,  
  routedComponents  
} from './characters-routing.module';  
  
@NgModule({  
  imports: [  
    CommonModule,  
    FormsModule,  
    CharactersRouterModule  
  ],  
  declarations: [routedComponents],  
  providers: [CharacterService],  
})  
export class CharactersModule { }
```

Import routing module

## characters-routing.module.ts

```
const routes: Routes = [  
  {  
    path: 'characters',  
    component: CharactersComponent,  
    children: [  
      { path: '', component: CharacterListComponent },  
      { path: ':id', component: CharacterComponent },  
    ]  
  }  
];  
  
@NgModule({  
  imports: [RouterModule.forChild(routes)],  
  exports: [RouterModule]  
})  
export class CharactersRoutingModule { }
```

Configure routes

forChild for feature module routes

Create routing module, and import into feature module

## app.module.ts

```
@NgModule({  
  imports: [  
    BrowserModule, FormsModule, HttpClientModule,  
    CharactersModule, AppRoutingModule  
,  
  declarations: [AppComponent],  
  bootstrap: [AppComponent]  
})  
export class AppModule { }
```

Import Characters module and its routes

## Importing Modules

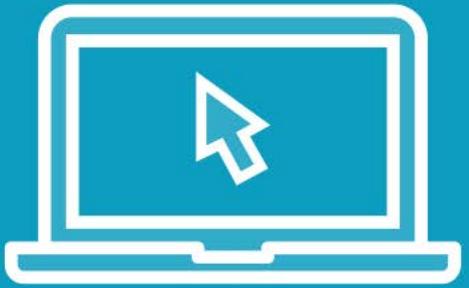
### Import the CharactersModule

We get its routes

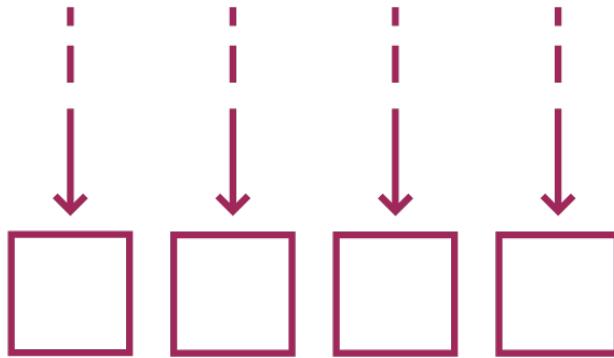
Order matters

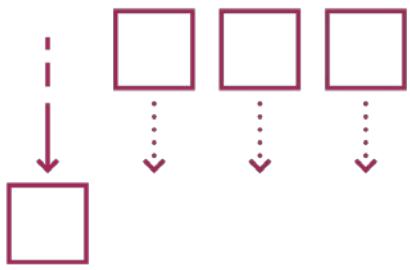


Demo



# Eager Loading Modules





## Lazy Loading Modules

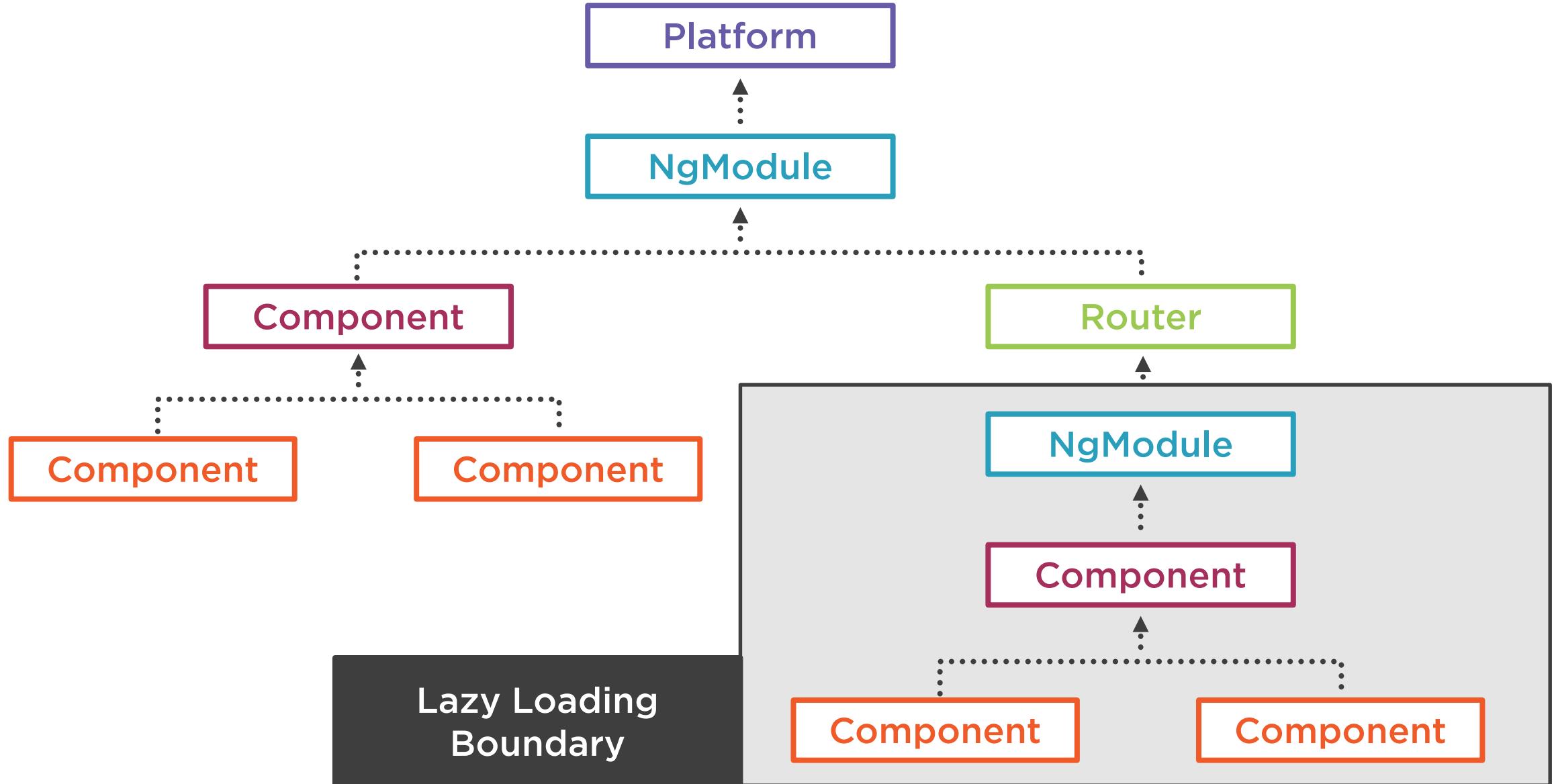


# Lazy Loading

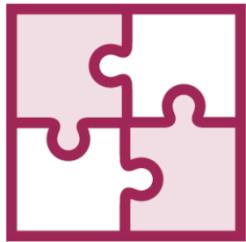
Loading modules on demand, as needed, just in time

Lowers the initial payload, improving the app startup experience





# Routing to Modules



## Eager Loading

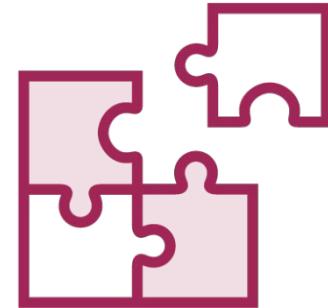
Loads at startup

Ideal for modules users need immediately

## Lazy Loading

Loads on demand

Ideal for modules not immediately in the workflow



## app-routing.module.ts

```
const routes: Routes = [
  { path: '', pathMatch: 'full', redirectTo: 'characters' },
  { path: 'characters', loadChildren: 'app/characters/characters.module#CharactersModule' },
  { path: 'vehicles', loadChildren: 'app/vehicles/vehicles.module#VehiclesModule' },
  { path: '**', pathMatch: 'full', component: PageNotFoundComponent },
];
```

Lazy load

## Lazy Loading

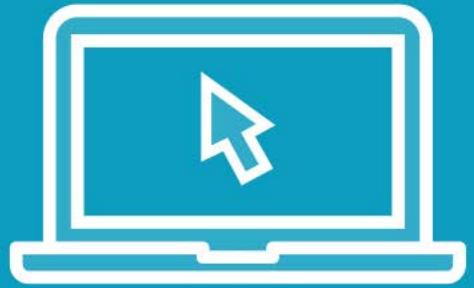
Use **loadChildren**

Load the module by **path # name**

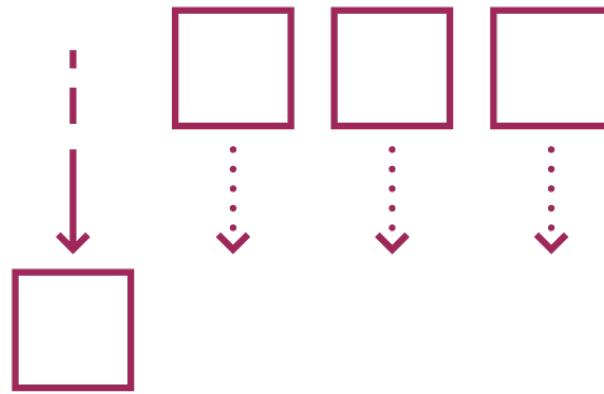
Do not import nor reference the module directly

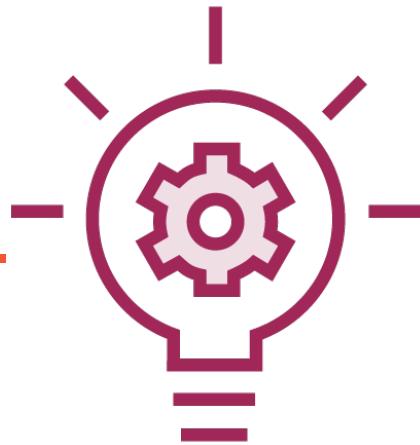


Demo



# Lazy Loading





## Preload Strategies



# Preload Strategies

When a user navigates to a lazily loadable module, the content has to be fetched from the server, which may cause a delay.

The router can preload lazily loadable modules in the background



## app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { PreloadAllModules, NoPreloading, Routes, RouterModule } from '@angular/router';

@NgModule({
  imports: [RouterModule.forRoot(routes, { preloadingStrategy: PreloadAllModules })],
  exports: [RouterModule],
})
export class AppRoutingModule { }
```

Preload Strategy

## Preload Strategies

We can preload all or none, out of the box

Pass the strategy to **forRoot**



# Custom Preload Strategies



## preload-strategy.ts

```
export class PreloadSelectedModulesList implements PreloadingStrategy {  
  preload(route: Route, load: Function): Observable<any> {  
    return route.data && route.data['preload'] ? load() : of(null);  
  }  
}
```

Implement the interface

## app-routing.module.ts

```
{  
  path: 'speakers', loadChildren: 'app/speakers/speakers.module#SpeakersModule',  
  data: { preload: true }  
},
```

Add the preload Observable

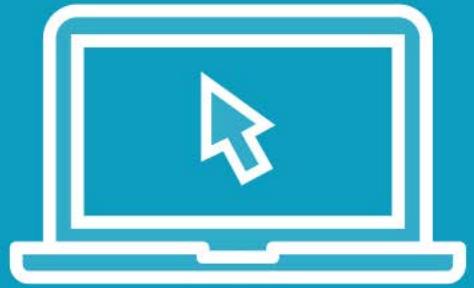
## app-routing.module.ts

```
@NgModule({  
  imports: [RouterModule.forRoot(routes, { preloadingStrategy: PreloadSelectedModulesList })],  
  exports: [RouterModule],  
  providers: [PreloadSelectedModulesList]  
})  
export class AppRoutingModule { }
```

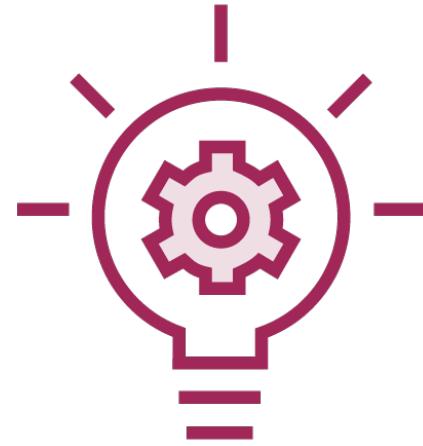
Provide the Strategy



Demo

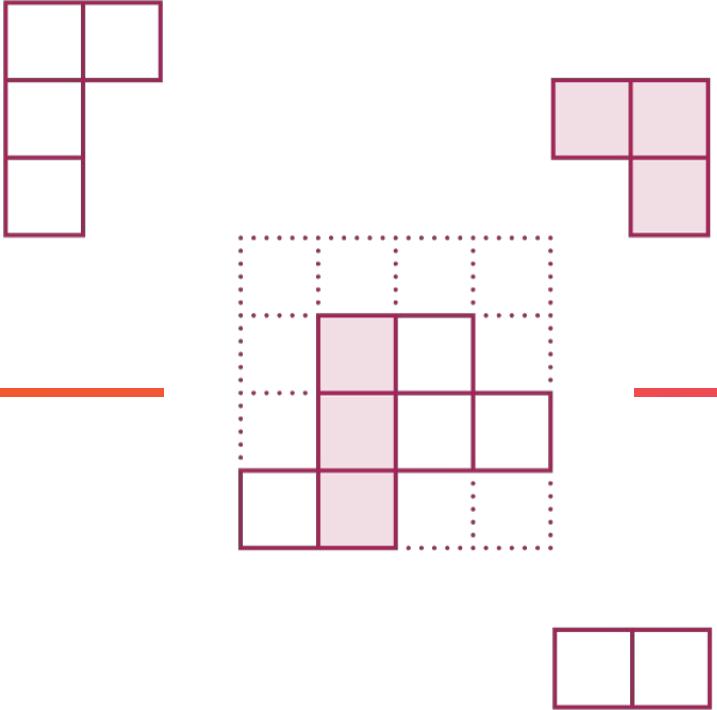


# Preload Strategies



# Feature Modules

---



# Basic Types of Feature Modules



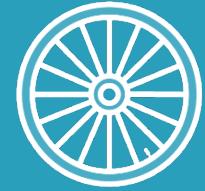
Domain



Routed



Service



Widget

# Basic Types of Feature Modules



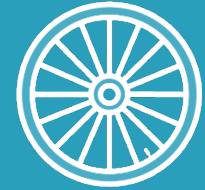
Domain



Routed



Service



Widget



e.g. **MenuModule**

**Deliver a user experience dedicated to a particular application domain.**

**Typically imported once by the root AppModule, without routing.**





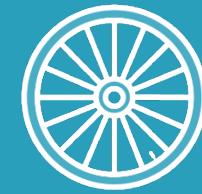
Domain



Routed



Service



Widget



## e.g. CharactersModule



Routed

**Routed feature modules are Domain  
feature modules who are the targets of  
navigation routes.**





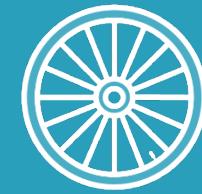
Domain



Routed



Service



Widget



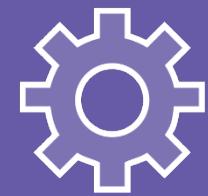
e.g. CoreModule

**Service feature modules contain singleton services used by anyone across the app.**

**Generally all providers, no components.**

**We import them once in App Root Module.**

**Do not import these in other modules**



**Service**

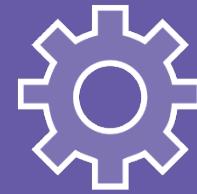




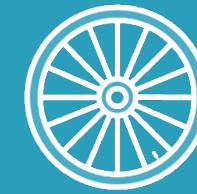
Domain



Routed



Service



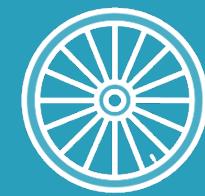
Widget



## e.g. SharedModule

**Shared feature modules generally contain components, directives and pipes.**

**We import these in every module that uses them.**



**Widget**



# AppModule



MenuModule



CoreModule



CharactersModule

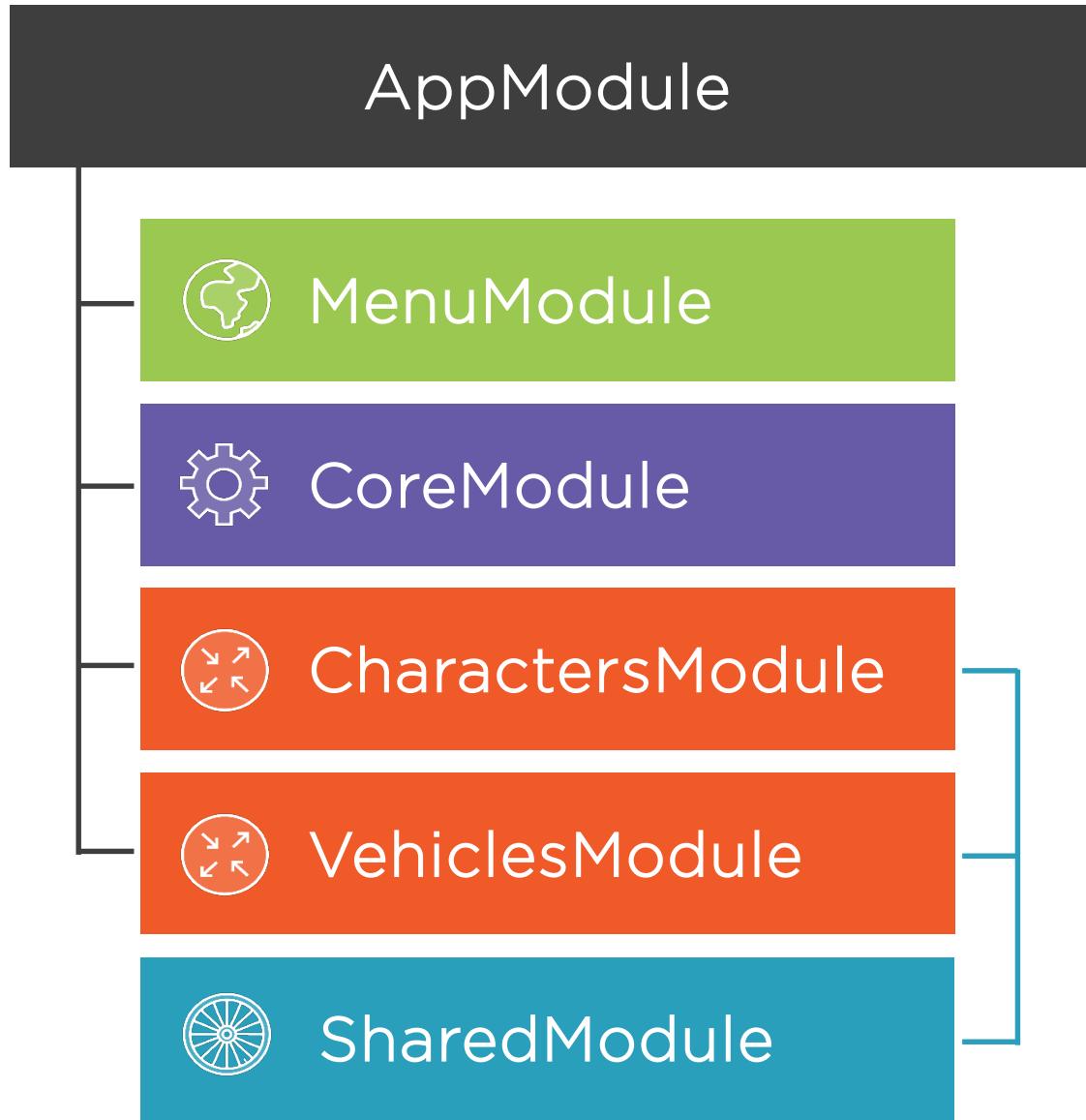


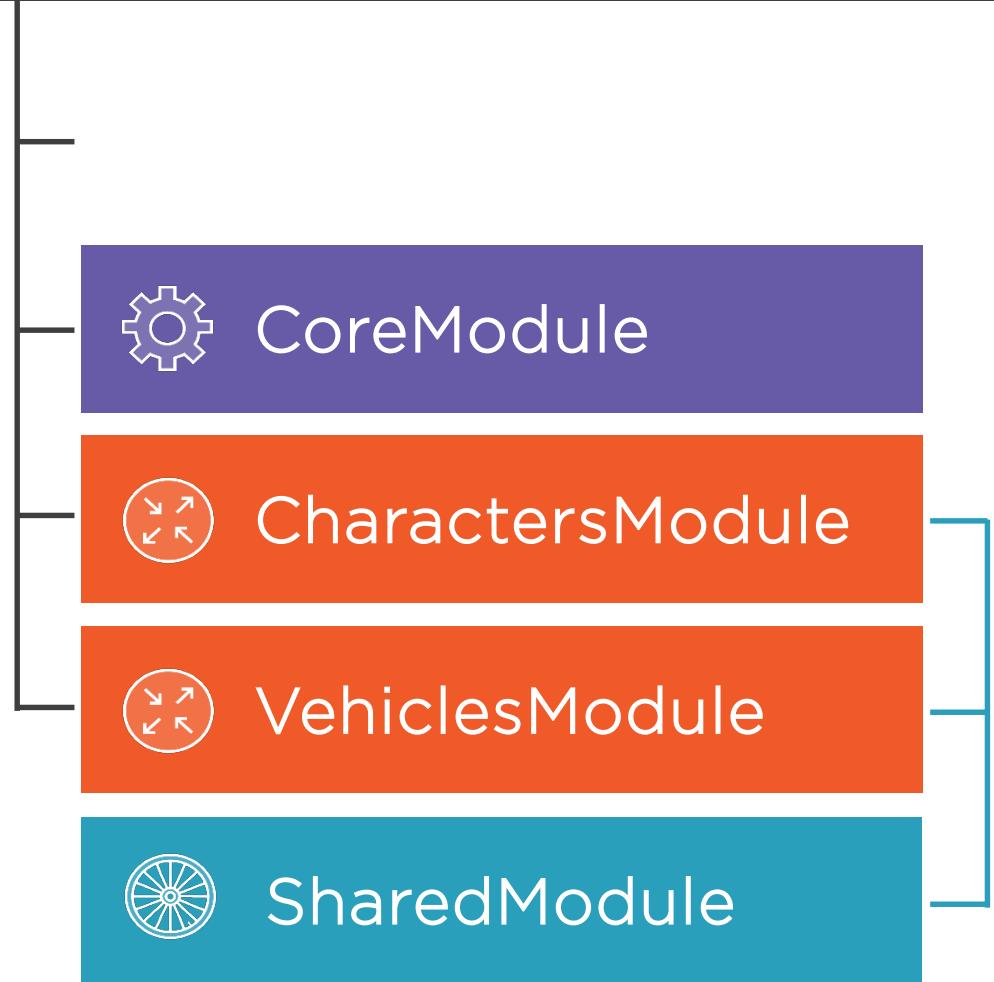
VehiclesModule

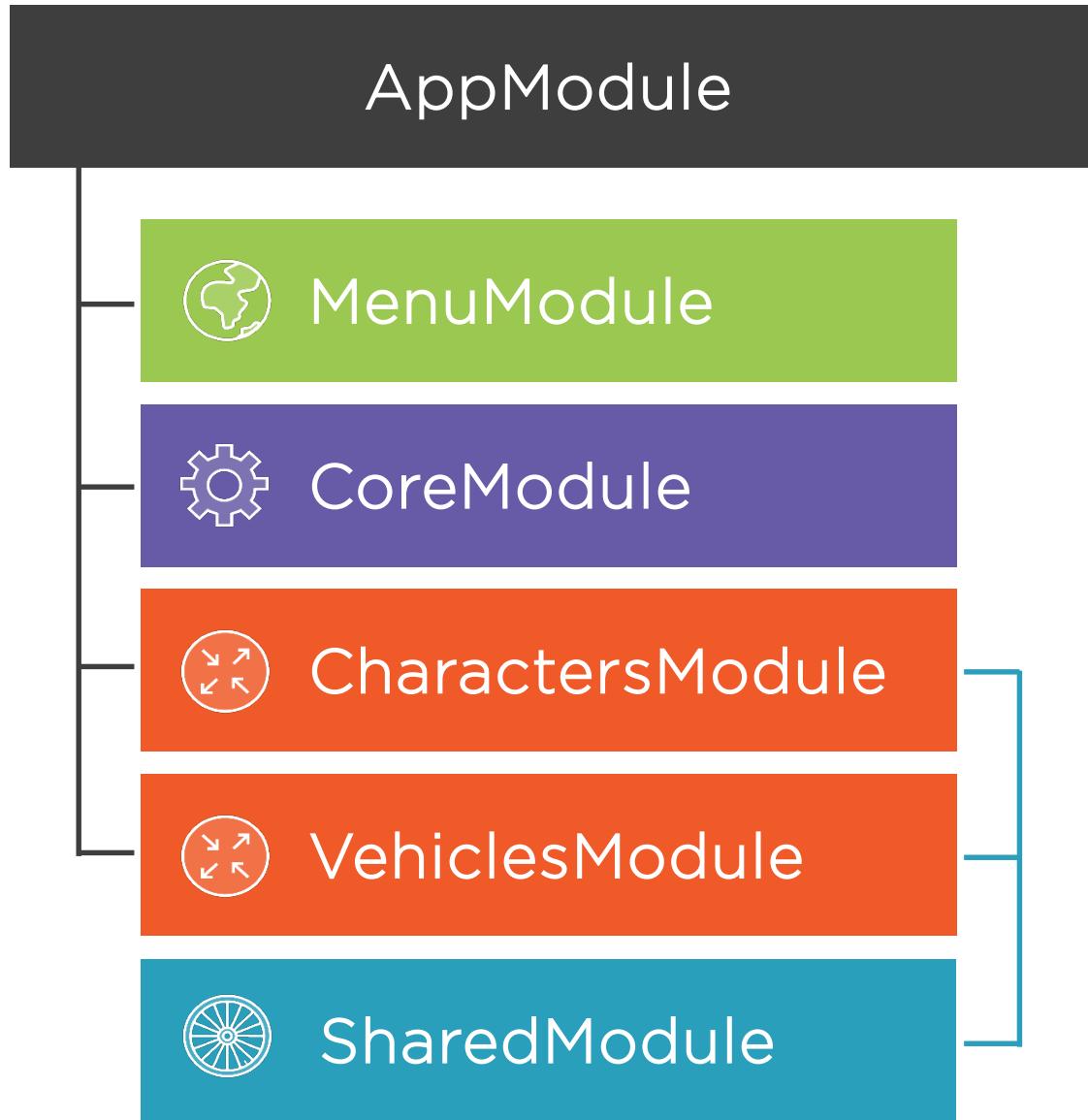


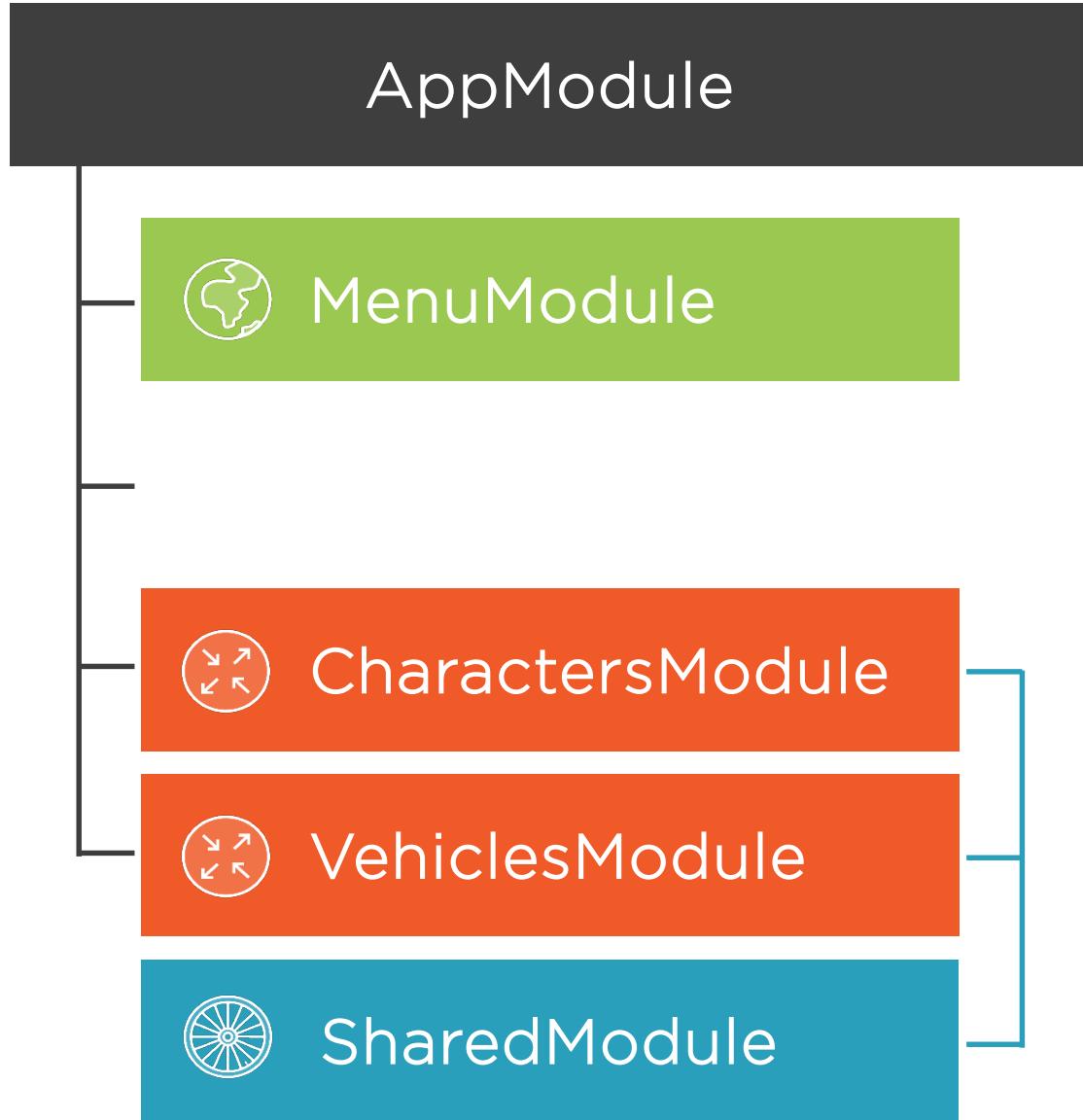
SharedModule

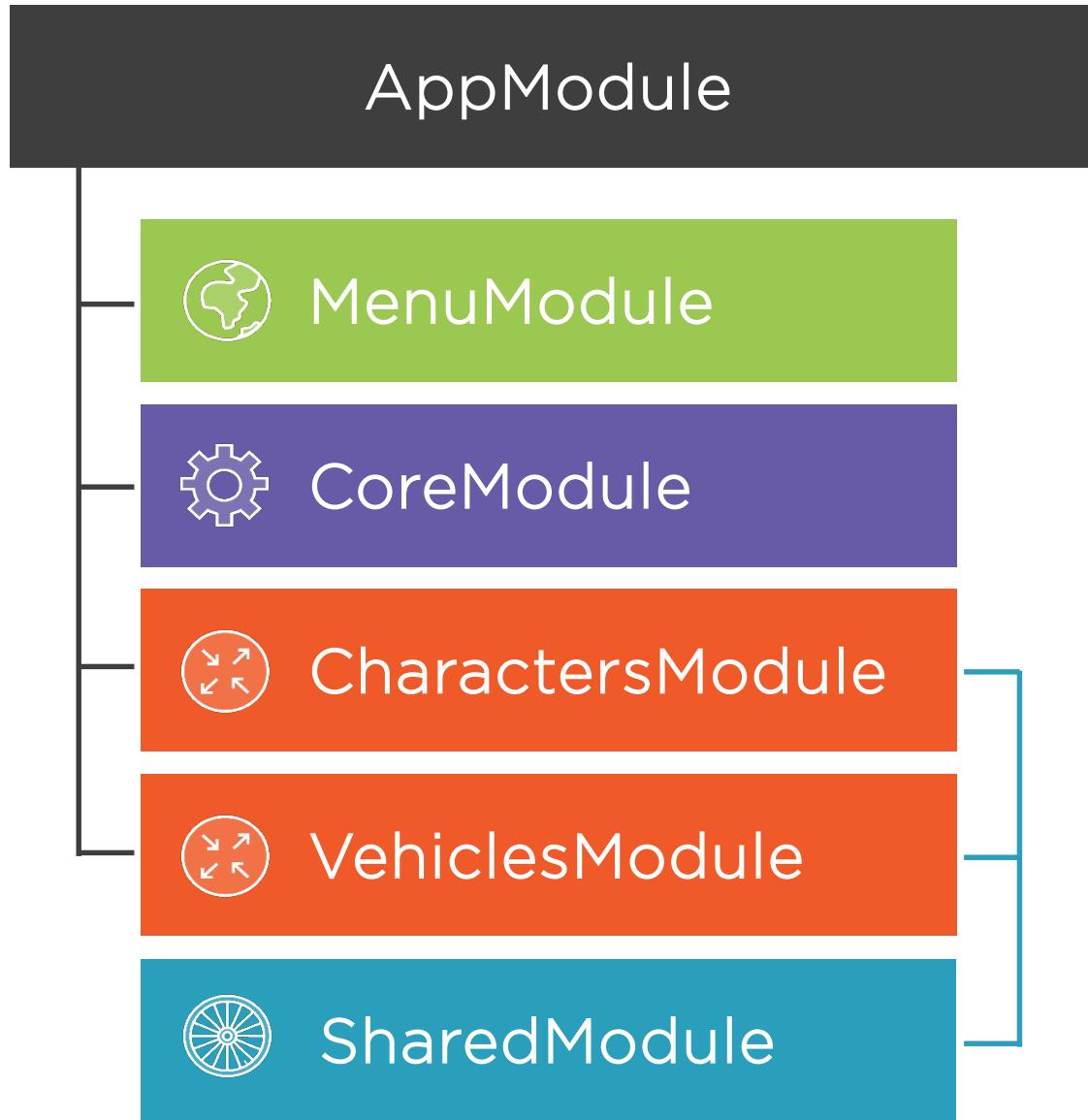




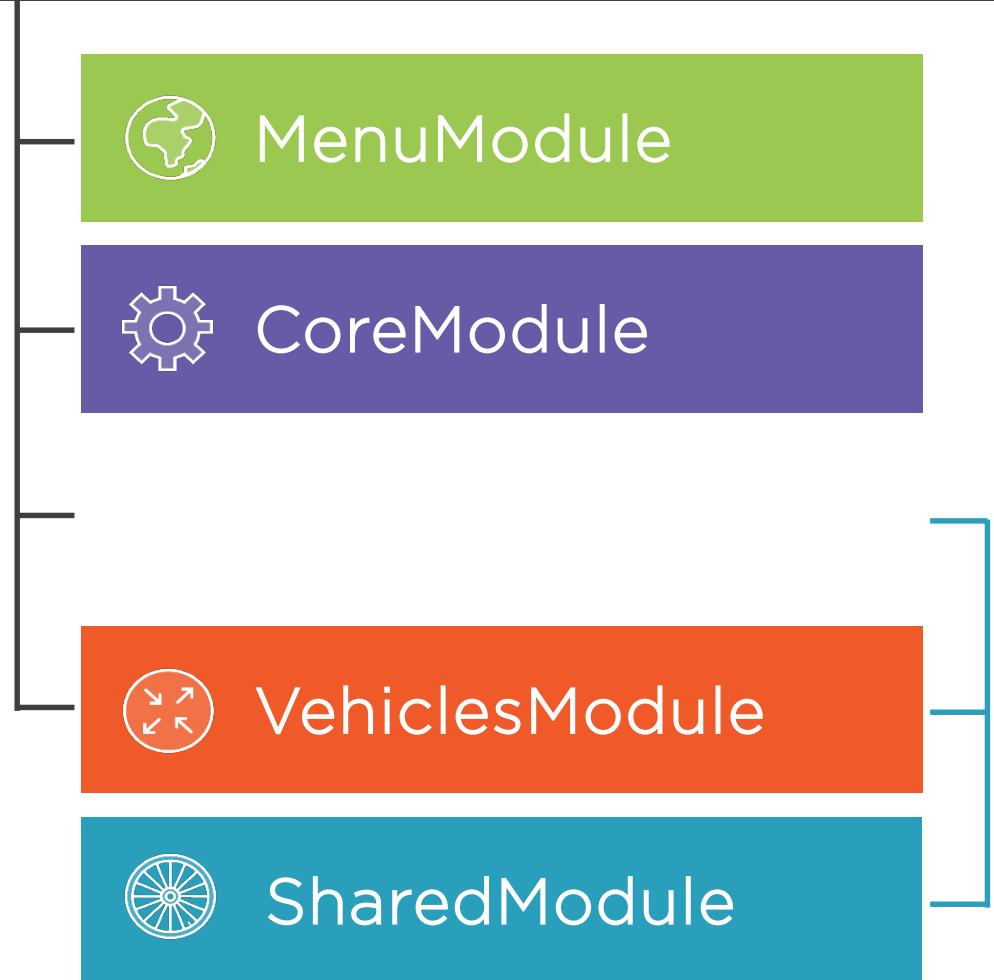


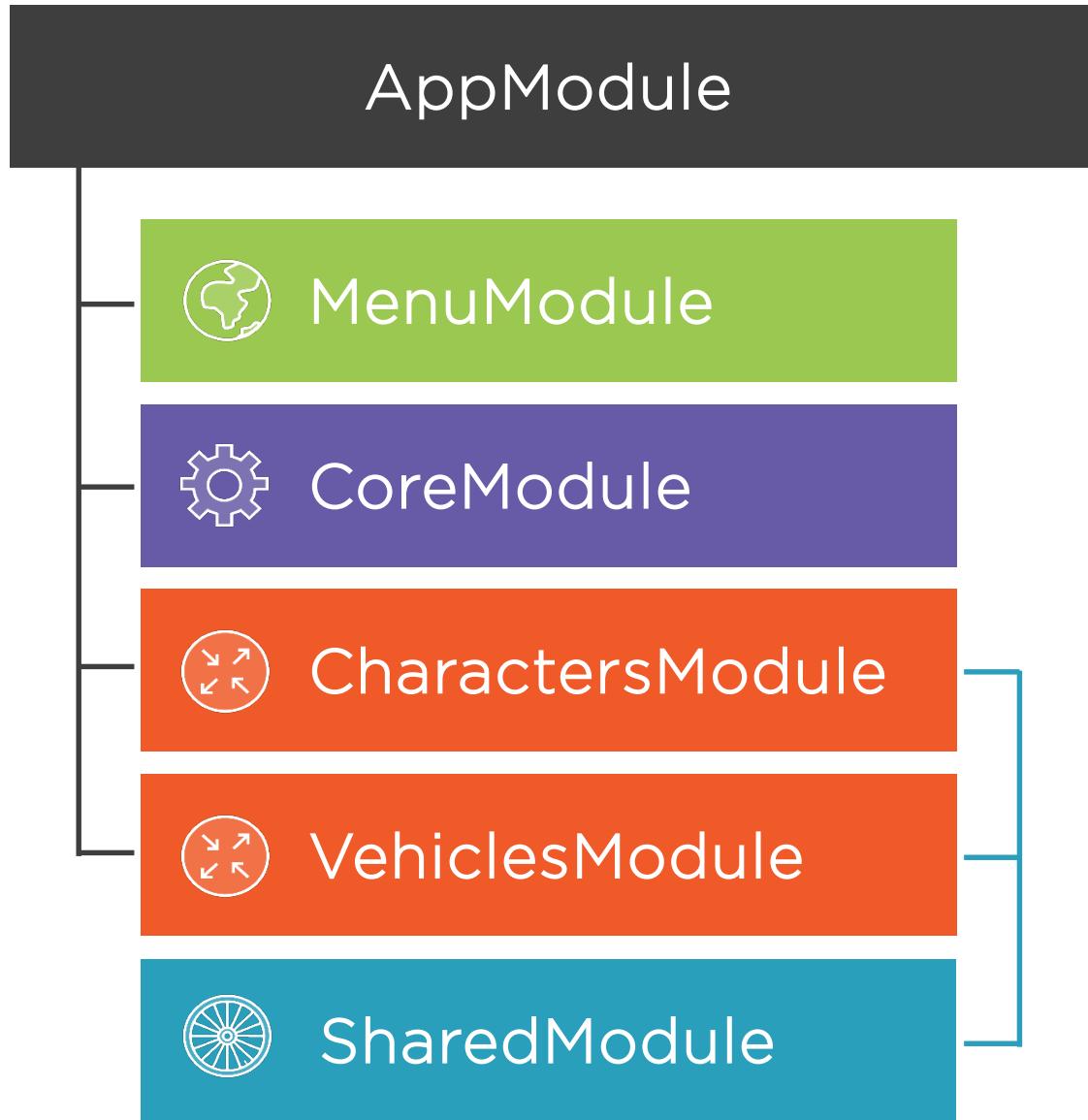




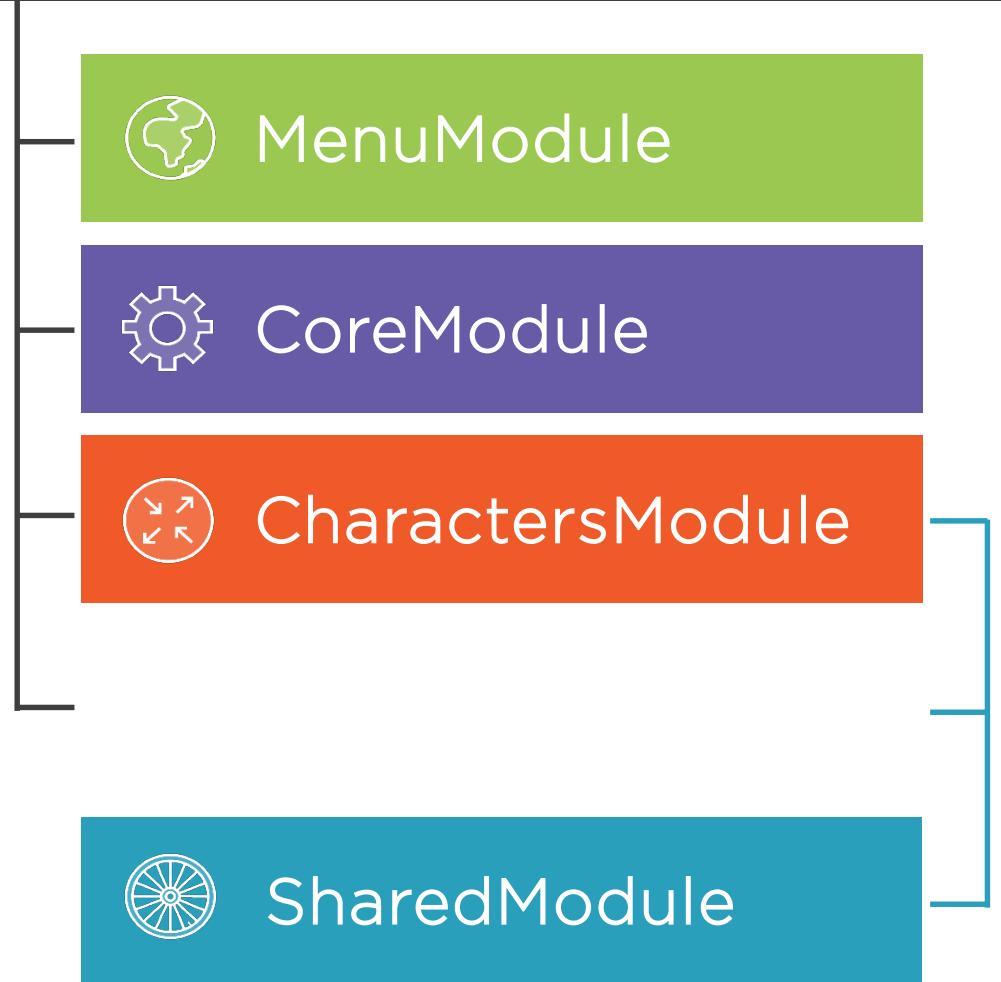


# AppModule



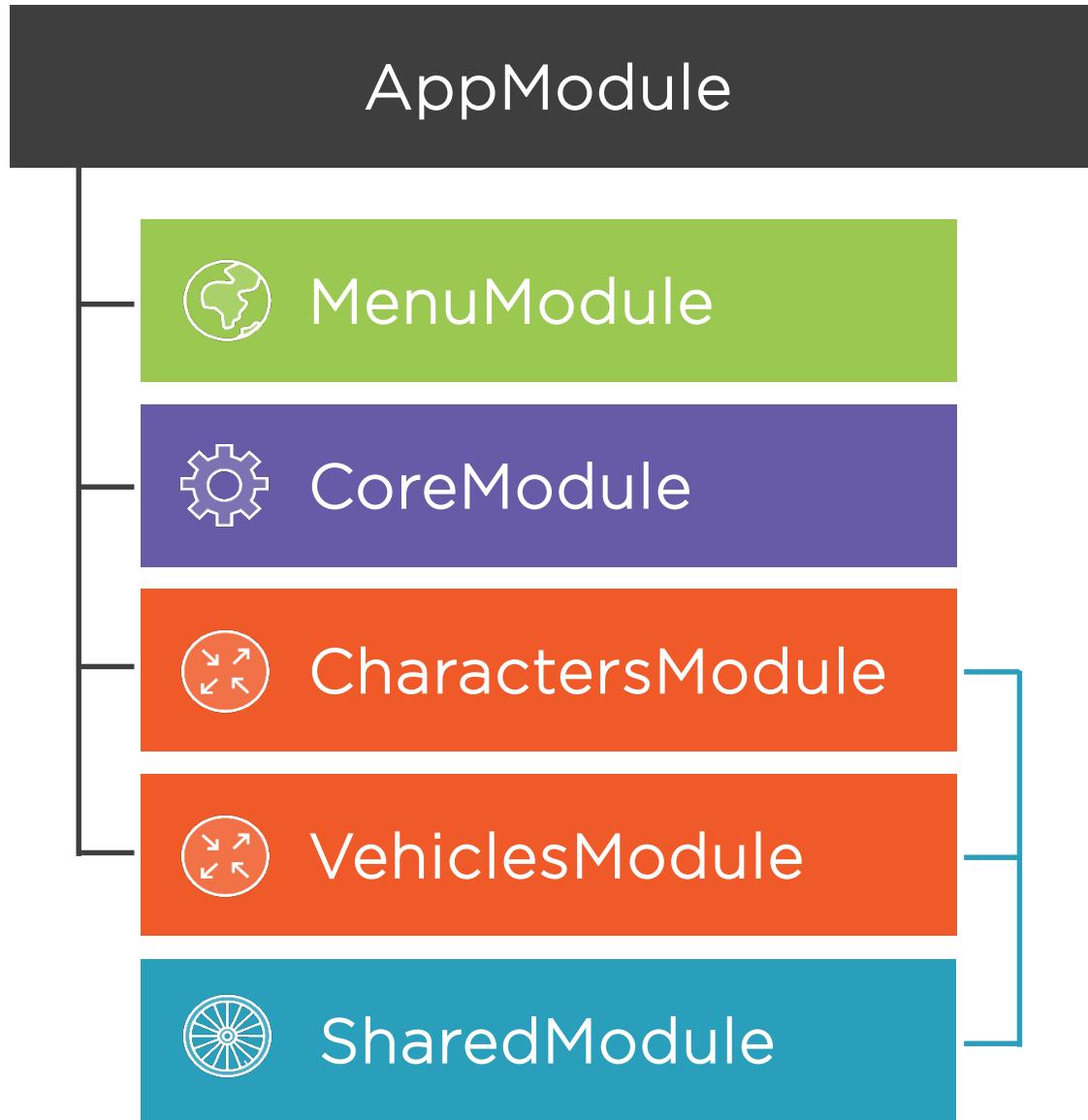


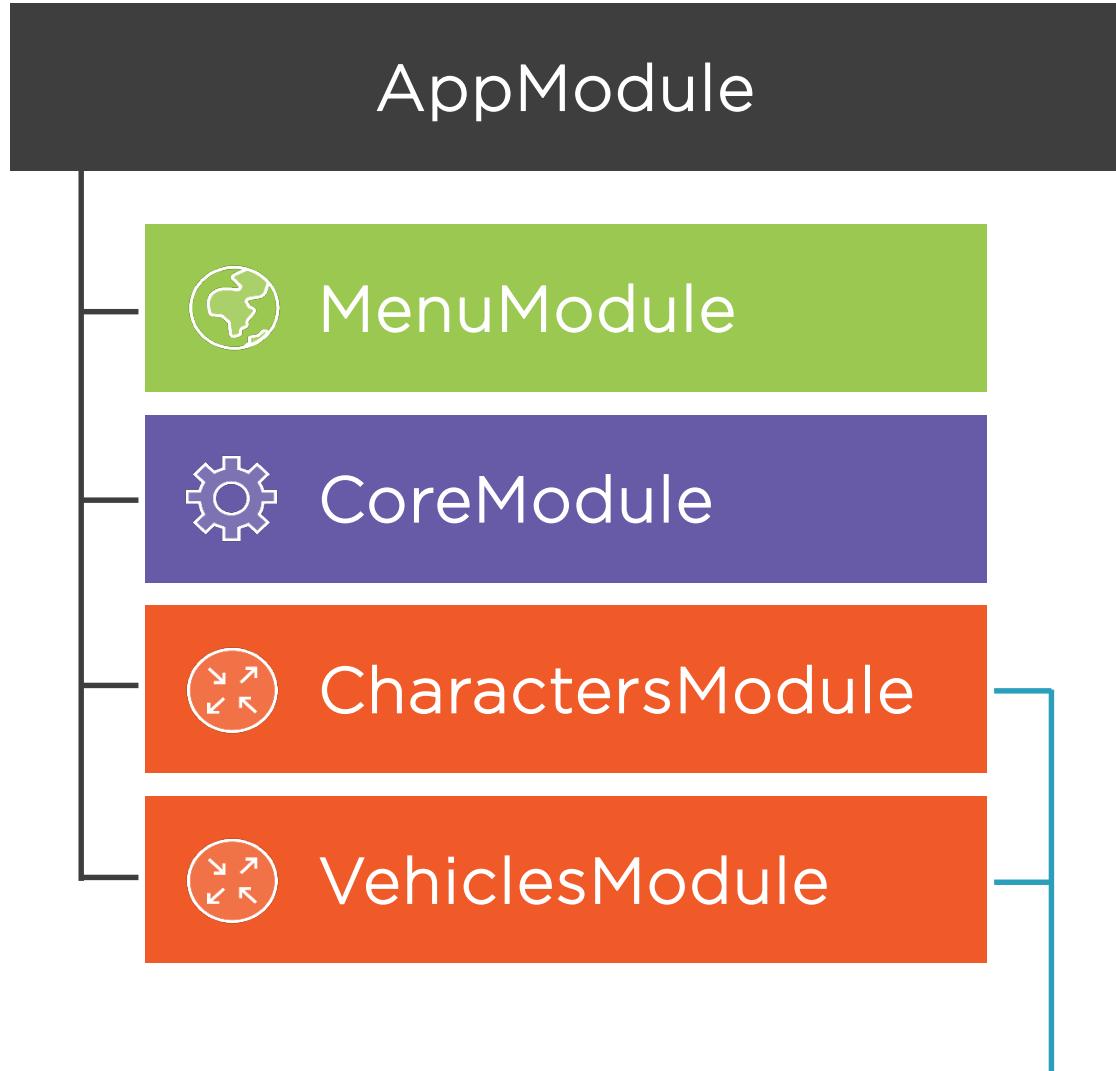
# AppModule



## VehiclesModule





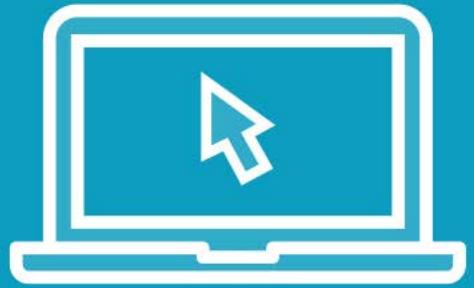


# Basic Types of Feature Modules

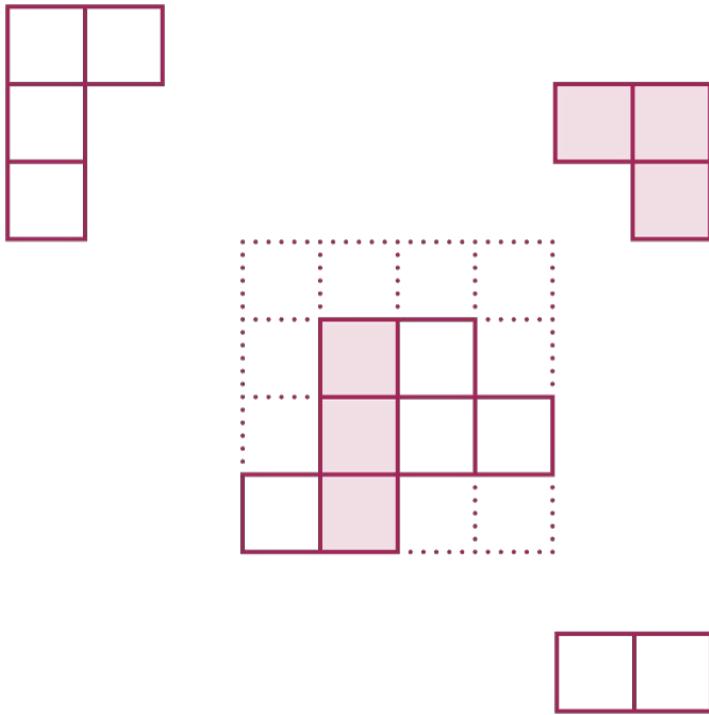
	Declarations	Providers	Exports	Imported By	Examples
	Domain		Rare	Some	Feature MenuModule
	Routed		Rare		Nobody if Lazy CharactersModule
	Service				AppModule CoreModule, HttpModule
	Widget		Rare		Feature CommonModule, SharedModule

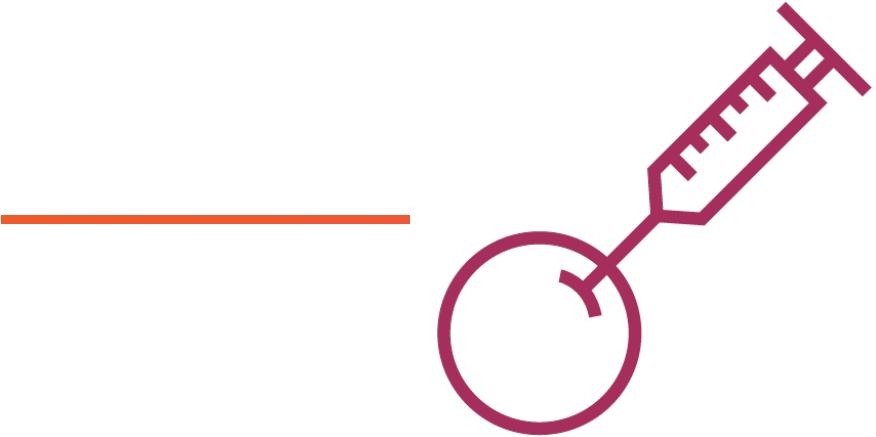


Demo



# Feature Modules





Providers



It is important to consider  
how we want our providers to  
behave in the module system



# Providing Services and Their Behavior

A service provided  
in a component, is  
accessible to that  
component and to  
its children



# Providing Services and Their Behavior

When we import a module with providers, we'll get a new instance of that service upon injection



# Providing Services and Their Behavior

A service provided in a component, is accessible to that component and to its children

Be careful putting providers in a module that can be imported more than once

When we import a module with providers, we'll get a new instance of that service upon injection



## login.component.ts

```
@Component({  
  moduleId: module.id,  
  templateUrl: 'login.component.html',  
  providers: [LoginService]  
})  
export class LoginComponent implements OnDestroy {
```

This component and its children can inject it

## Component Providers

**LoginService** is only used by **LoginComponent**

Available to this component and its children



# Module Providers

Provided to the root injector

Available everywhere

```
@NgModule({
  imports: [...],
  declarations: [AppComponent, routableComponents],
  providers: [
    CanActivateAuthGuard, CharacterService, UserProfileService, VehicleService
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Provide to the root injector

app.module.ts

# Summary



**Eager Loading**

**Lazy Loading**

**Preload Strategies**

**Identifying Feature Modules**

**Providers**

