Angular Development Workshop

Dan Wahlin

John Papa

John Papa

https://johnpapa.net

@John_Papa

Dan Wahlin

https://codewithdan.com

@DanWahlin

Wifi:

SSID: MGMResorts-Wifi
Password: Room Number and Name
(or visitor)

Get the Content

http://codewithdan.me/ng-ts-1-day

Agenda

- Introduction to Angular
- Angular CLI
- Modules, Components and Templates
- Binding and Directives
- Services
- Http
- Routing

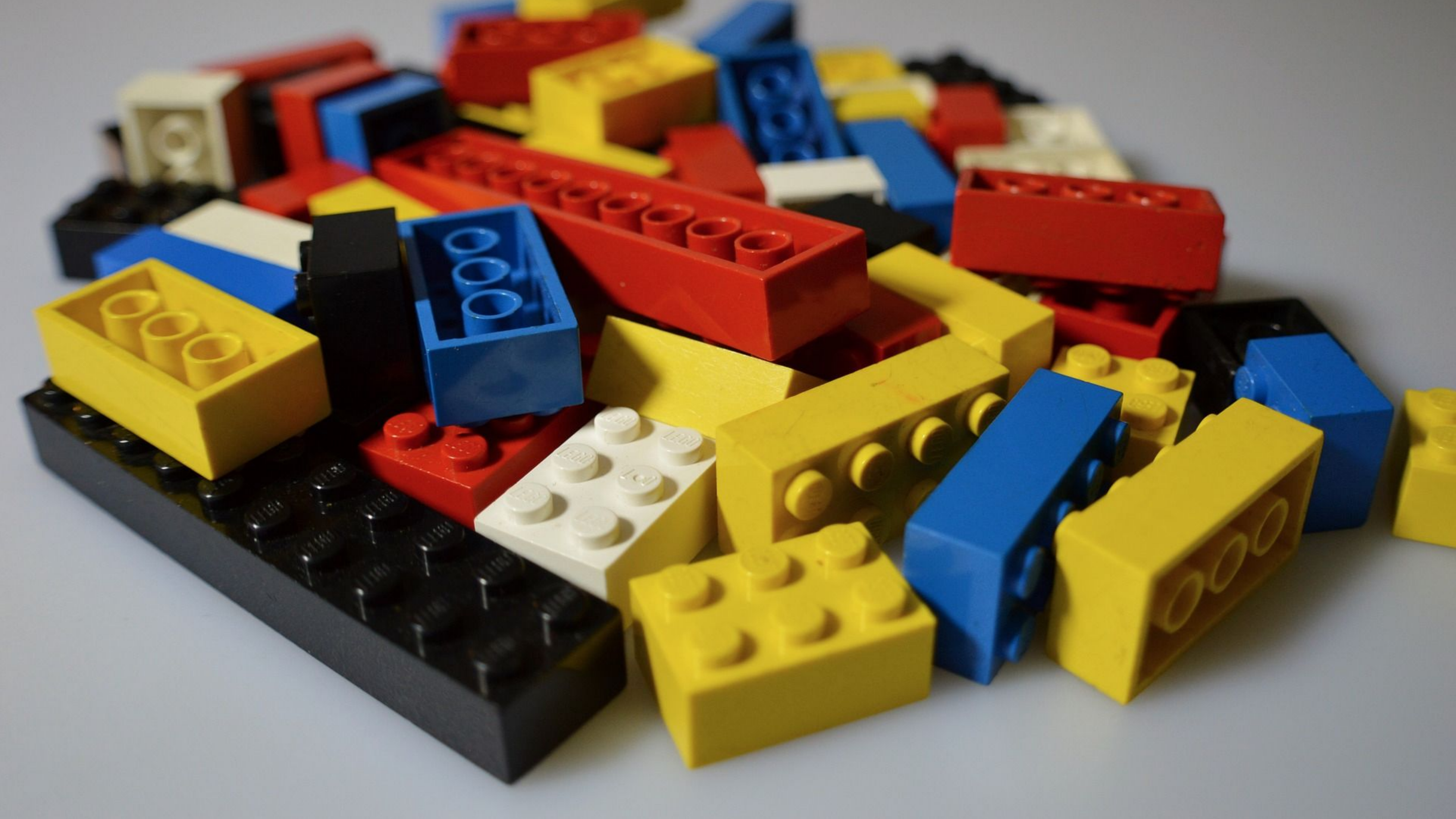
Introduction to Angular

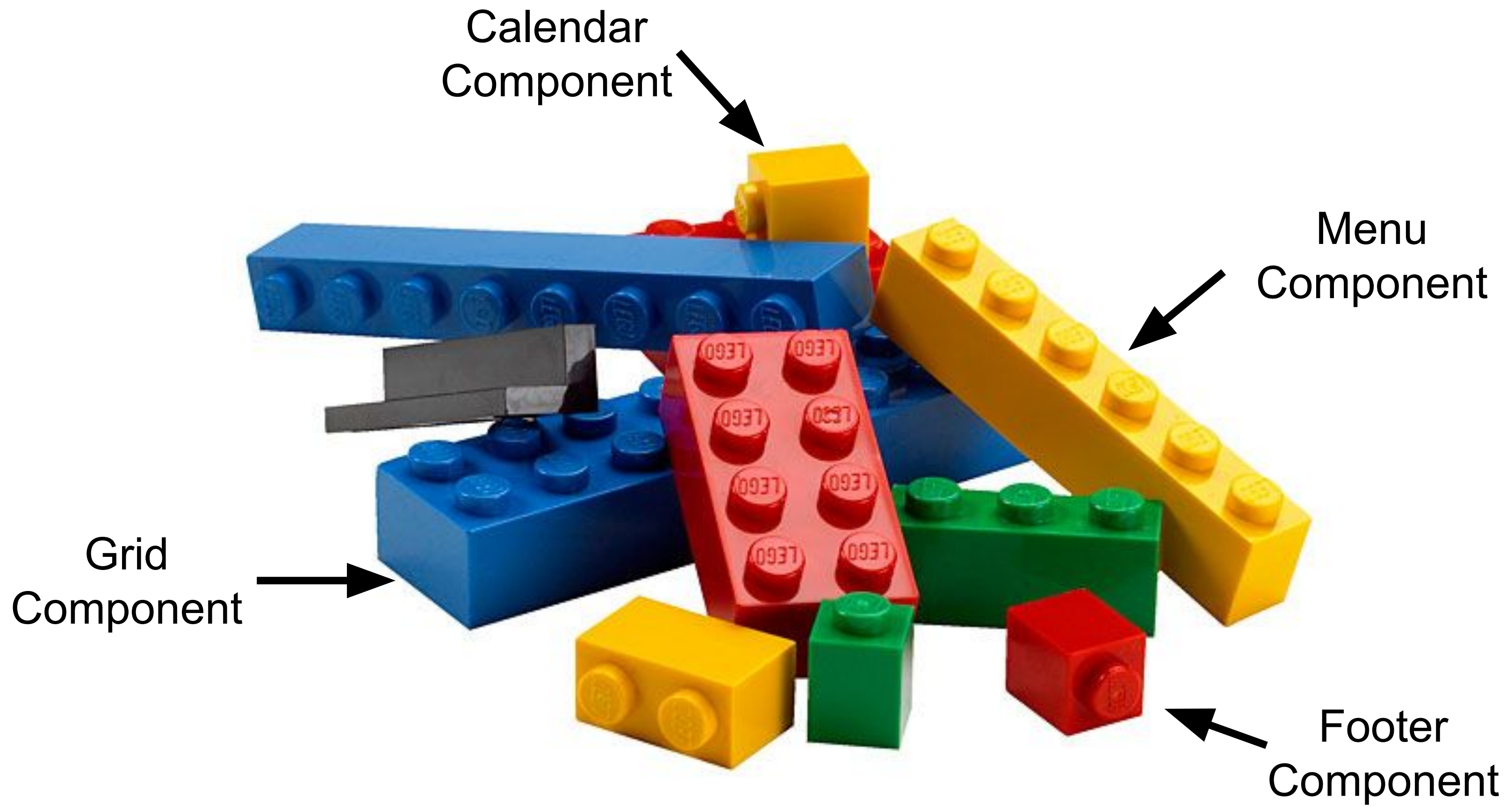
AngularJS

Version 1.x of the framework

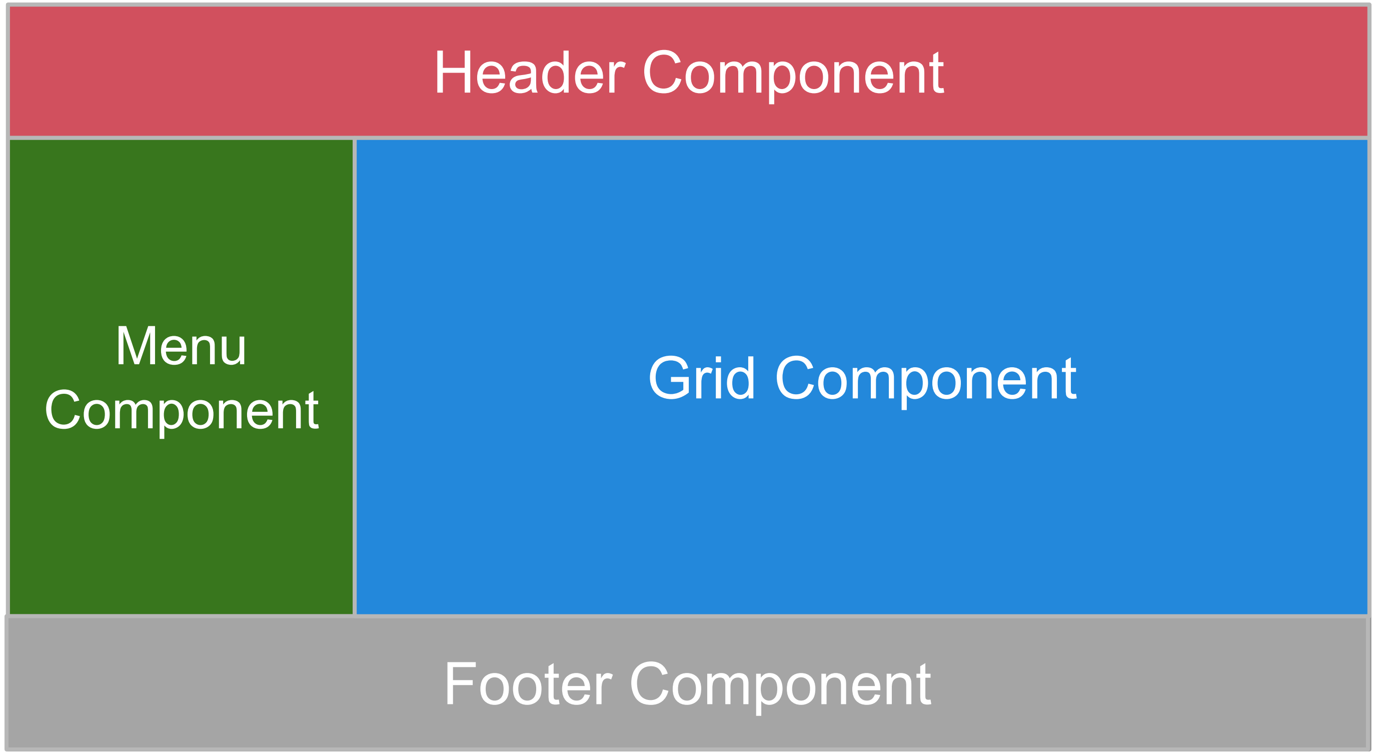
Angular

Version 2 or higher of the framework



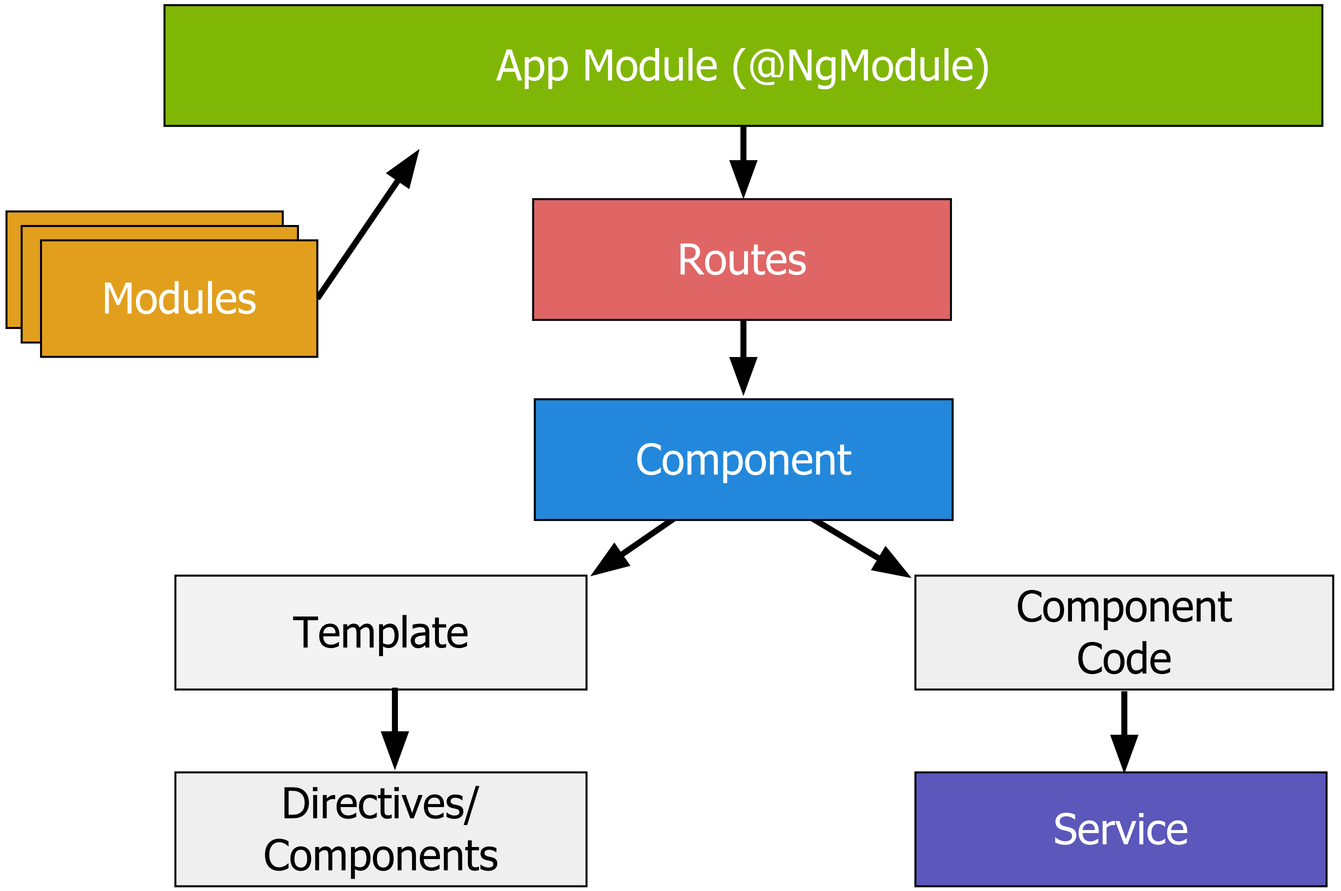

Calendar Component
Menu Component
Grid Component
Footer Component

The Big Picture

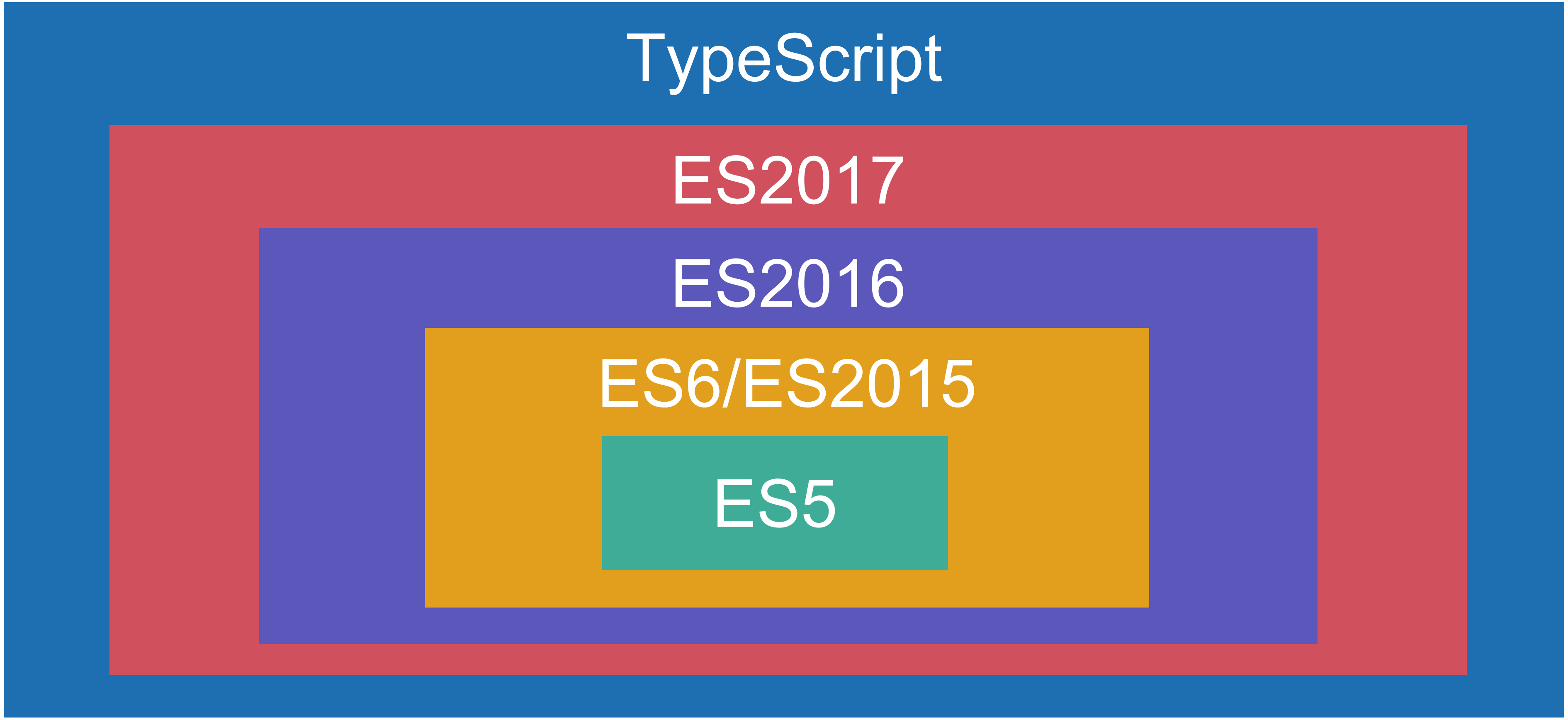


Angular Overview

Modules	Components	Decorators	Languages (TypeScript, ES 20xx, ES5)
Dependency Injection	Services	Data Binding	Performance


App Module (@NgModule)
Modules
Routes
Component
Template
Component Code
Directives/ Components
Service

JavaScript is Valid TypeScript



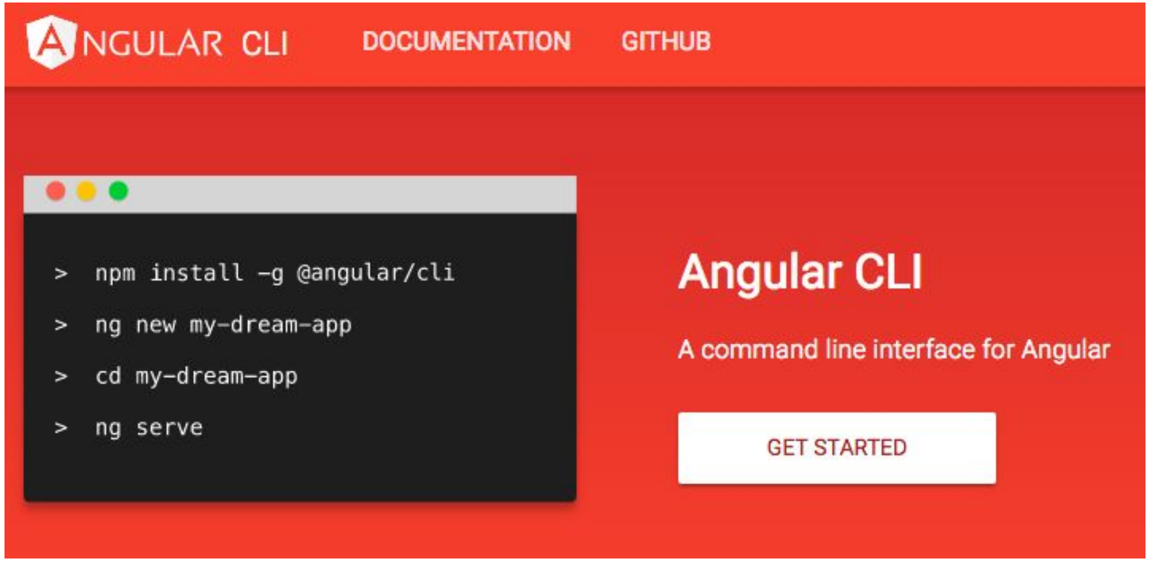
http://www.typescriptlang.org/play

Demo

Angular CLI

The Angular CLI

Angular applications can be generated using the Angular Command-Line Interface (CLI): https://cli.angular.io



```
> npm install -g @angular/cli
> ng new my-dream-app
> cd my-dream-app
> ng serve
```

Angular CLI

A command line interface for Angular

Angular CLI Key Features

- Easily create an Angular application that follows best practices in the Angular style guide:

  https://angular.io/guide/styleguide

- Create new components, directives, pipes, routes and services
- Create a "build" version of the application for deployment
- Run unit tests and end-to-end tests
- Serve up the application in the browser

Key Angular CLI Commands

```
ng --version
ng --help

ng new my-app-name

ng generate

    [component | directive | pipe | service | class | interface | enum | guard]

ng build

ng serve

ng lint

ng test
```

Keeping your Angular App Current

Use this link to learn how to update your app to the latest version https://update.angular.io

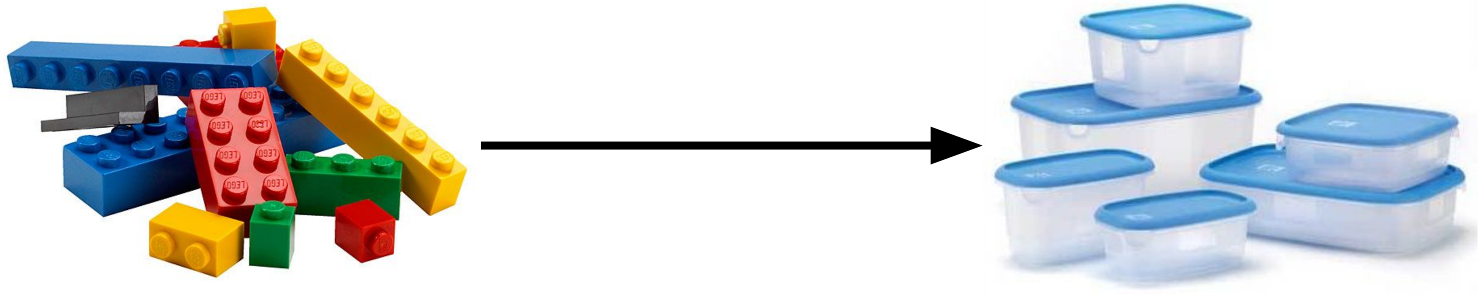
Run this command to ask the Angular CLI what steps you should run to update your Angular app

```
ng update
```

NOTE: You may need to first run npm install -g @angular/cli to get the latest global CLI first

Modules, Components and Templates

Modules and Components



Steps to Build Components

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

Steps to Build Components

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

The Role of ES2015 Modules

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

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

Exporting Modules

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

data.service.ts

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

Importing Modules

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

customers.component.ts

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

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

Steps to Build Components

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

What’s a Component?

- Components are reusable objects
- A component consists of: HTML Template Code
- Has a “selector”: <app-customers></app-customers>

What’s in a Component?

imports

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

decorators

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

class

```
export class CustomersComponent {

}
```

Steps to Build Components

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

The @Component Decorator

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

Property	**Description**
selector	Defines the selector that triggers instantiation of the component (ex: 'customers' = <customers></customers>
template & templateUrl	Defines the template used by the component
styles & styleUrls	Defines any CSS styles used by the component

Using @Component Properties

Defining metadata for providers and directives using @Component decorator

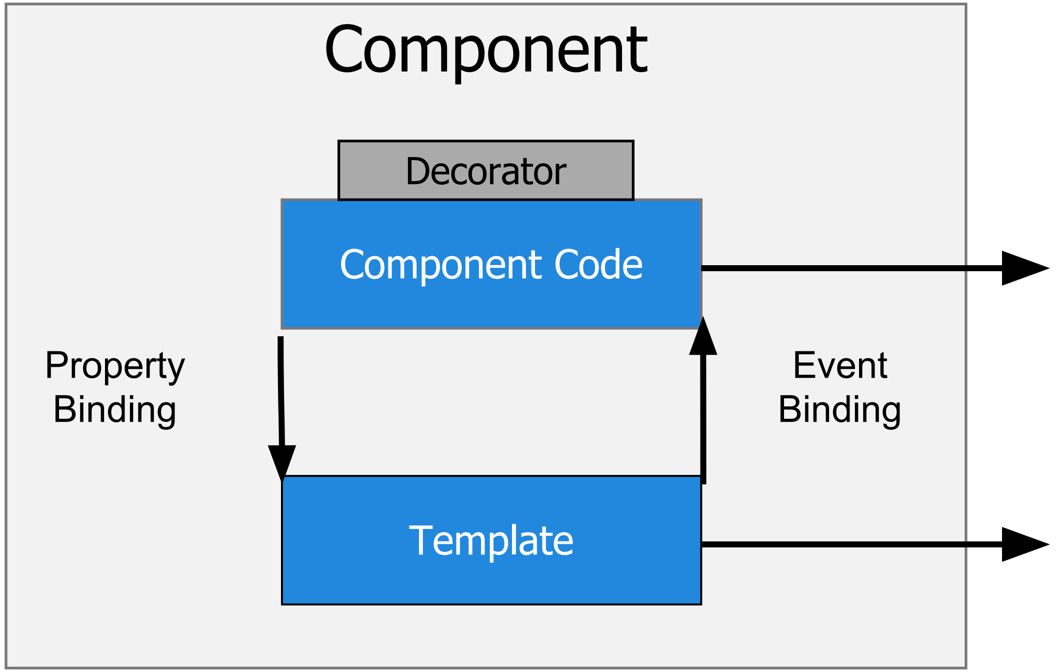
```
import { Component } from '@angular/core';

@Component({
  selector: 'app-customers',
  templateUrl: './customers.component.html'
})
export class CustomersComponent {
  constructor() { }
}
```

<app-customers><app-customers>

Steps to Build Components

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

Component Code and Templates



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

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

Demo: optional

Bootstrapping Angular

Creating a Template

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

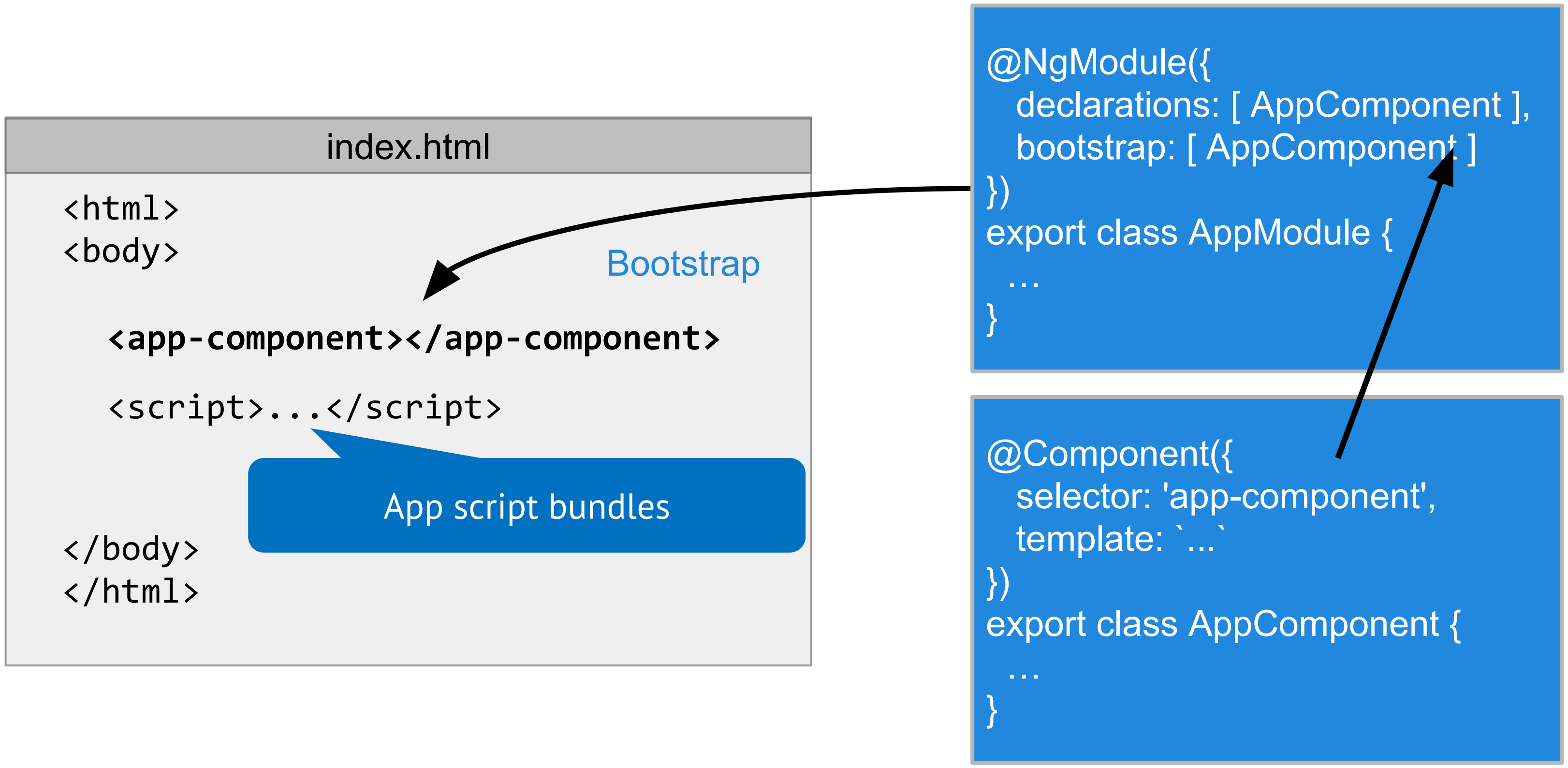
customer.component.ts

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

customer.component.html

```

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

Components, Modules and Bootstrapping



index.html

```
<html>
<body>

   <app-component></app-component>

   <script>...</script>


</body>
</html>
```

```
@NgModule({
   declarations: [ AppComponent ],
   bootstrap: [ AppComponent ]
})
export class AppModule {
   …
}
```

```
@Component({
   selector: 'app-component',
   template: `...`
})
export class AppComponent {
   …
}
```

NgModule

@NgModule helps organize an application

NgModules set us up for success with lazy loading, too

app.module.ts

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

import { AppComponent }  from './app.component';

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

Bootstrapping Angular

Applications must bootstrap a root app module

Import the platformBrowserDynamic() function and pass the root app module

main.ts

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule }              from './app.module';

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

Angular Bootstrap Flow

1. Index.html
2. Application bundles/scripts load
3. File at app/main.js is loaded
4. main.js bootstraps "root" module (AppModule)
5. AppModule contains the "root" component

Lab: Components

http://codewithdan.me/ng-workshop-labs

Binding and Directives

Template Syntax

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

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

One-Way Interpolation

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

{{ expression }}

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

<br />

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

One-Way Property Bindings

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

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

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

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

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

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

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

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

Event Bindings

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

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

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

Two-Way Binding

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

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

"Banana/Football in a Box" syntax

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

Structural Directives

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

Angular directive that generates a template

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

Add <div> to DOM if customer property has a value

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

Pipes

Angular apps can use Pipes to filter and format data

Several built-in pipes

uppercase, lowercase, slice, date, currency, json

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

Format as currency

Lab: Data Binding and Directives

http://codewithdan.me/ng-workshop-labs

Services and DI

Services

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

Service

Provides something of value

Shared data or logic

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

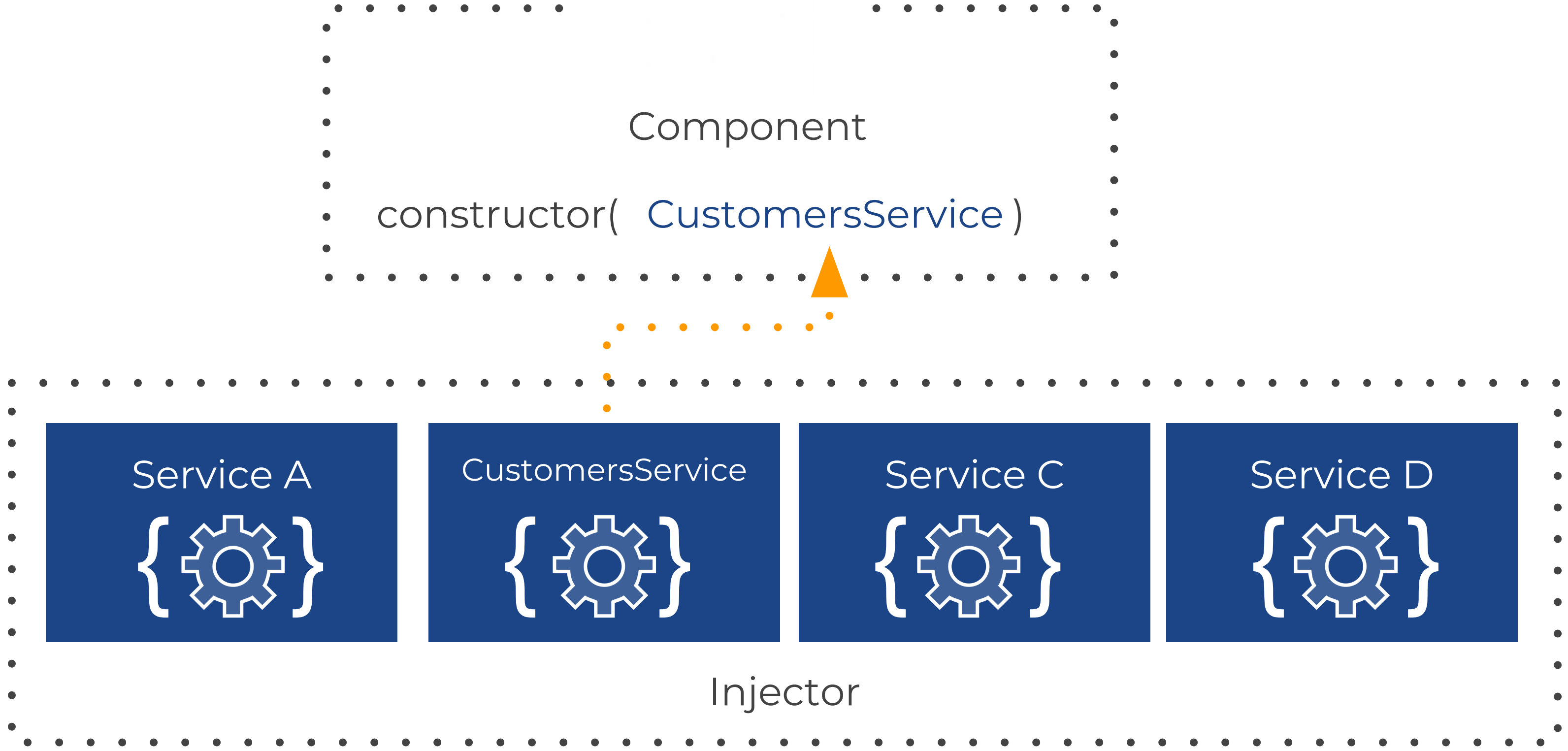
customers.service.ts

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

@Injectable()
export class CustomersService {

    getCustomers() {
        return ...;
    }

}
```

Service is just a class

Dependency Injection

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


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

Registering a Service Provider: Option 1

Register the provider for a service using **@Injectable()**

Works with **Angular v6 or higher**

customers.service.ts

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

@Injectable({ providedIn: 'root' })
export class CustomersService {

    getCustomers() {
        return ...;
    }

}
```

Will be registered/provided in the root module

Registering a Service Provider: Option 2

A service provider can be registered in a module

app.module.ts

```
import { NgModule } from '@angular/core';
import { CustomersService } from '../core/services/customers.service';

@NgModule({
  ...
  providers: [ CustomersService ]
})
export class AppModule { }
```

Create a provider for service

Injecting a Service into a Component

Locates the service in the Angular injector

Injects into the constructor

customers.component.ts

```
export class CustomersComponent implements OnInit {
    customers: ICustomer[];

    constructor(private customersService: CustomersService) { }

    ngOnInit() {
       this.customers = this.customersService.getCustomers();
    }
}
```

Inject service

Http

Http

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

Http Step by Step

Import the HttpClientModule

Inject HttpClient in a service

Call get() function

Subscribe to the Service's function in the Component

Http Requirements

HttpClientModule contains the providers for HttpClient (Angular 4.3+)

app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule  } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';

import { AppComponent } from './app.component';
import { CustomersComponent } from './customers.component';


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

Import HttpModule

Http and Observables

Angular's HttpClient provides standard GET, PUT, POST, DELETE functions

Relies on Observables

customers.service.ts

```
    constructor(private http: HttpClient) {}

    getCustomers() : Observable<ICustomer[]> {
      return this.http
        .get<ICustomer[]>('api/customers.json')
        .pipe(
            tap(data => console.log(data)),
            catchError(this.handleError)
        )
    }
```

Subscribing to the Observable

Component is handed an Observable

We subscribe to it

customers.component.ts

```
getCustomers() {
  this.customers = [];

  this.customersService.getCustomers()
    .subscribe(
      customers => this.customers = customers,
      error => this.errorMessage = error
    );
}
```

Subscribe to the Observable

Handle error conditions

Lab: Services and Http

http://codewithdan.me/ng-workshop-labs

Routing

What is Routing?

- Multiple pages (views)
- Menu systems
- Very distinct feature areas in the app
- Desire to lazy load sections of code, only when needed

Angular Routing

- Components can be changed/swapped by using routing
- Import and use RouterModule from **@angular/router**
- Can define parent and child routes

Steps to Use Routing

1. Add a <base> Element
2. Define routes
3. Pass routes to the root module
4. Add a <router-outlet>
5. Use the routerLink Directive

Add a <base> Element

Router supports history.pushState
Allows paths like http://yourdomain.com/customers to be used
The <base> element needs to be set for it to work properly

index.html

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


</body>
</html>
```

Steps to Use Routing

1. Add a <base> Element
2. Define routes
3. Pass routes to the root module
4. Add a <router-outlet>
5. Use the routerLink Directive

Import Routes and RouterModule

RouterModule gives us access to routing features

Routes help us declare or route definitions

app-routing.module.ts

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

Import routing features

Defining Routes

Define the route's path

Indicate parameters with :

Set the component that we'll route to

app-routing.module.ts

When I see this path, go to this component

```
const routes: Routes = [
  { path: '', pathMatch: 'full', redirectTo: 'customers' },
  { path: 'customers', component: CustomersComponent },
  { path: 'customers/:id', component: CustomerComponent },
  { path: '**', component: PageNotFoundComponent },
];
```

Define a Module

Create a routing module using our routes, and import it

Export our new AppRoutingModule

app-routing.module.ts

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

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

Routing, All Together

app-routing.module.ts

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

const routes: Routes = [
  { path: '', pathMatch: 'full', redirectTo: 'customers', },
  { path: 'customers', component: CustomersComponent },
  { path: 'customers/:id', component: CustomerComponent },
  { path: '**', component: PageNotFoundComponent },
];

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

Steps to Use Routing

1. Add a <base> Element
2. Define routes
3. Pass routes to the root module
4. Add a <router-outlet>
5. Use the routerLink Directive

Importing Routing in the AppModule

Pass application routes to the root module
Import our AppRoutingModule

app.module.ts

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent }  from './app.component';
import { CustomersComponent } from './customers/customers.component';
import { AppRoutingModule } from './app-routing.module';
import { DataService } from './core/services/data.service';

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

Import routes into our root module

Steps to Use Routing

1. Add a <base> Element
2. Define routes
3. Pass routes to the root module
4. Add a <router-outlet>
5. Use the routerLink Directive

```
@Component({
  selector: 'app-container',
  template: `<router-outlet></router-outlet>`
})
export class AppComponent { }
```

Define where components get loaded in the application

app.component.html

RouterOutlet

Angular puts components in a "component container"

<router-outlet> defines location where components are loaded

Steps to Use Routing

1. Add a <base> Element
2. Define routes
3. Pass routes to the root module
4. Add a <router-outlet>
5. Use the routerLink Directive

RouterLink Directive

The routerLink directive can be used to add links to routes

Defines the route path and any route parameter data

customer.component.ts

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

customer.component.html

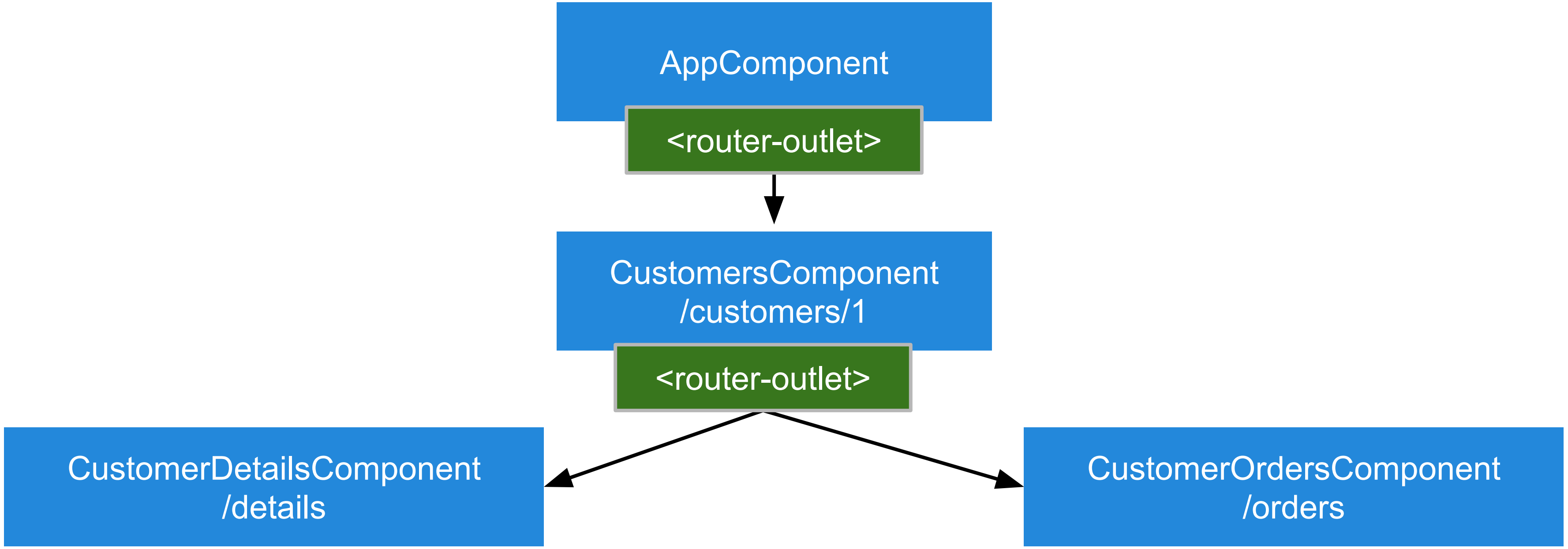
```
<a routerLink="/customers">
  Customers
</a>

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

Child Routes

We may have the need for multiple routers or nested routes

Child Routes

Angular supports child routes



Child Routes

app-routing.module.ts

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

const routes: Routes = [
  {
    path: 'customers/:id', component: CustomerComponent,
    children: [
       { path:'details', component: CustomerDetailsComponent },
       { path:'orders',  component: CustomerOrdersComponent },
       { path:'edit',    component: CustomerEditComponent }
    ]
  }
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Parent Route

Child Routes

Use forRoot()

Route Parameters

Snapshot	Observable
Easiest, as long as parameter values do not change	Gets new values as parameters change when component is re-used

Snapshot Parameters

customer.component.ts

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

  constructor(private route: ActivatedRoute) { }

  ngOnInit() {
      this.id = +this.route.snapshot.paramMap.get('id');
      this.getCustomer();
    }
  }

  // ...
}
```

Access route information

Grab the snapshot of the current route parameters

Observables and Parameters

customer.component.ts

```
export class CustomerComponent implements OnInit {
  id: number;

  constructor(private route: ActivatedRoute) { }

  ngOnInit() {
      this.route.paramMap
        .subscribe(params => {
          const id = params.get('id');
          this.getCustomer(id);
        });
    }
  }
  // ...
}
```

Access route information

Get route parameters, as they change. Ideal when routing to the same component.

Eager and Lazy Loading

Routing allows us to load NgModules eagerly or lazily

Loading Routes

Eager	Lazy
At startup	On demand

Lazy Loading: Defining Lazy Routes

Use loadChildren

Load the module by path and name

Do not import nor reference the module directly

app-routing.module.ts

Lazy

```
const routes: Routes = [
  { path: '', pathMatch: 'full', redirectTo: 'customers' },
  { path: 'customers', loadChildren: 'app/customers/customers.module#CustomersModule'},
  { path: 'orders', loadChildren: 'app/orders/orders-routing.module#OrdersRoutingModule'},
  { path: '**', component: PageNotFoundComponent },
];
```

Eager

A Lazy Loaded NgModule

orders-routing.module.ts

```
const routes: Routes = [
  {
    path: '', component: OrdersComponent,
    children: [
      { path: '', component: OrderListComponent },
      { path: ':id', component: OrderComponent}]
  },
];

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

/orders

/orders

/orders/37

Must use forChild

Lab: Routing

http://codewithdan.me/ng-workshop-labs

Angular Apps

https://github.com/johnpapa/angular-event-view-cli
https://github.com/johnpapa/heroes-angular

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

Thanks for Coming!

@John_Papa
@DanWahlin

http://codewithdan.me/ng-ts-1-day

Github Repo for Labs

https://github.com/DanWahlin/AngularWorkshopLabs

(copy the "labs" folder out of the project)

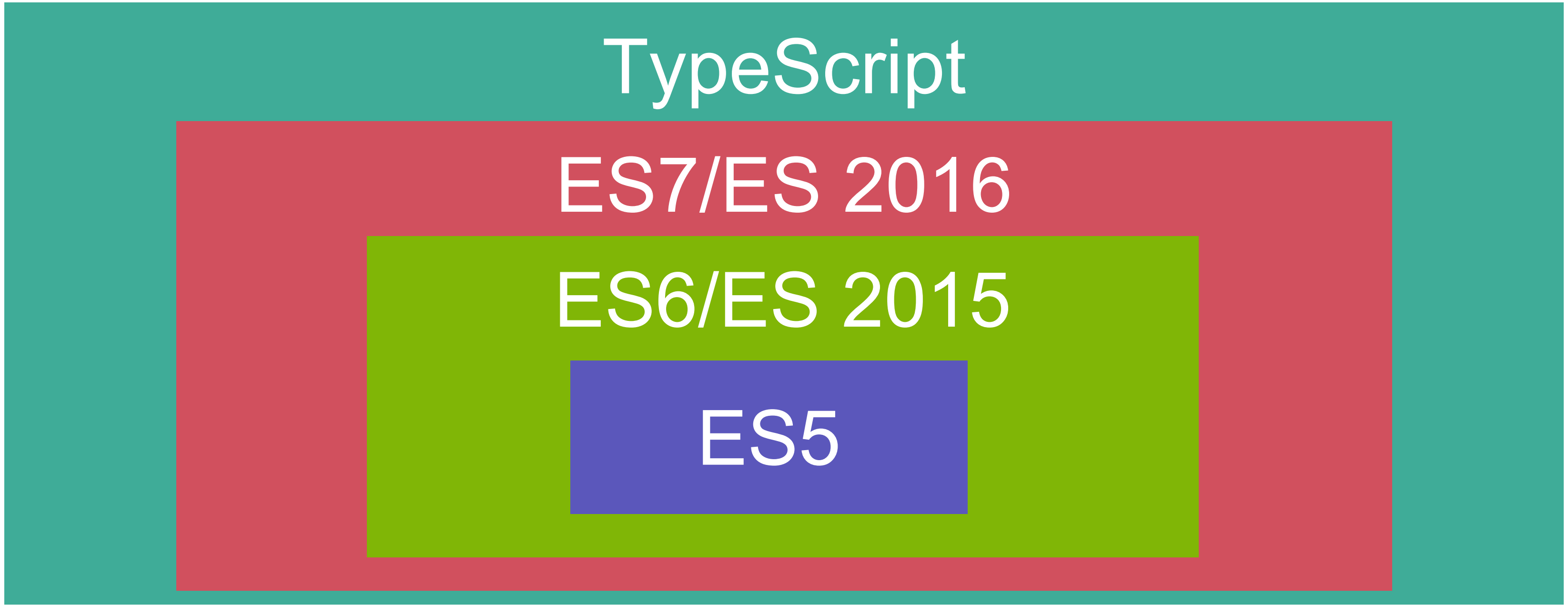
TypeScript (Bonus)

Using TypeScript

- The future of JavaScript is ES 2015
    - Major update to the JavaScript language
    - Modules, classes and more
    - Will help you align with Angular

- TypeScript builds on top of ES 2015

JavaScript is Valid TypeScript



Key ES 2015 Features

Classes

Block Scope

Destructuring

Arrow Functions

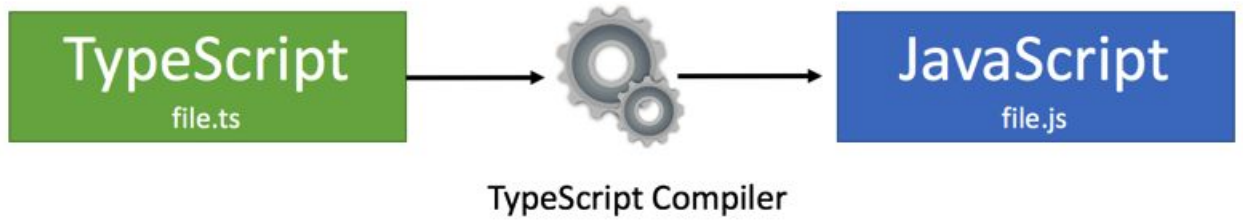
Modules

Default/Rest Parameters

More...

https://github.com/DanWahlin/ES6Samples

TypeScript Compilation



tsconfig.json

TypeScript configuration for your project

tsconfig.json

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "moduleResolution": "node",
    "sourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "removeComments": false,
    "noImplicitAny": true,
    "allowJs": true
  }
}
```

Target to Transpile to

ES2016 Features

BYOE: Bring Your Own Editor

ATOM

Visual Studio

Classes

- TypeScript/ES 2015 supports classes:
  - Define constructors
  - Define properties
  - Support for "inheritance" (uses prototyping)
  - Shortcut function names
  - Encapsulation of code!

TypeScript Class Example

auto.ts

```
class Auto {
  constructor(engine) {
        this._engine = engine;
    }
  get engine() {
        return this._engine;
    }
  set engine(val) {
        this._engine = val;
  }
  start() {
        console.log(this.engine);
  }
}
```

constructor

get/set property blocks

function

Modules

- TypeScript/ES 2015 supports importing "modules"
- Will appeal to CommonJS (Node.js) and AMD (require.js) developers
- Compact syntax that relies on **import** and **export** keywords

Using export and import

Modules rely on export and import keywords

```
export var name = 'James';
export var city = 'Chandler';



import { name, city } from 'customer';
console.log(name); // 'James'

import * as customer from 'customer';
console.log(customer.name); // 'James'
console.log(customer.city); // 'Chandler'
```

Arrow Functions

- Allow anonymous functions to be defined using a compact "arrow" syntax

- Nest functions more easily
  - Callback functions
  - Promise success/failure functions
  - Event functions

- Simplifies working with "this"

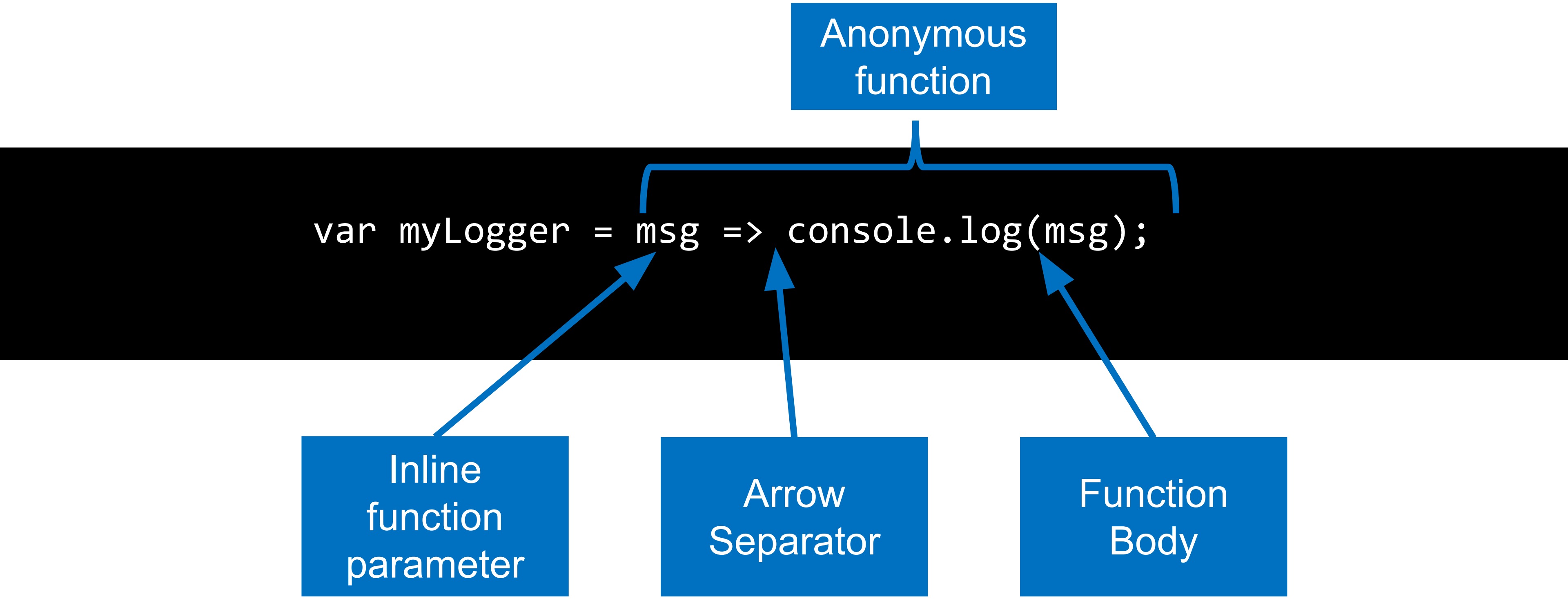
Anonymous Function and Arrow Functions

```
var myLogger = function(msg) {
    console.log(msg);
};
```

$\downarrow$

```
var myLogger = msg => console.log(msg);
```

Arrow Function Syntax



```
var myLogger = msg => console.log(msg);
```

```
//Create an anonymous function
var myLogger = msg => console.log(msg);
myLogger('Testing out arrow functions!');


map.forEach((val, key) => console.log(key + ': ' + val));


document.querySelector('#submit')
        .addEventListener('click', e => alert(e.target));
```

Using Arrow Functions

Arrow Functions and this

ES5

```
function Car() {
     var self = this;
     this._seats = 4;

     this.timeout = function() {
       setTimeout(function() {
         console.log(self._seats++);
       }, 1000);
     }
}
```

Capture this

Use captured this

ES 2015

```
class CarWithArrow {
     constructor() {
          this._seats = 6;
     }

     timeout() {
       setTimeout(() => {
         console.log(this._seats++);
       }, 1000);
     }
}
```

this captured automatically

Template Strings

- Supports multi-line strings
- Allow variables and expressions to be embedded in string literals using **${expression}**
- Use the **`** character to start and end a string
- Clean up situations where multiple strings are concatenated

Template Strings In Action

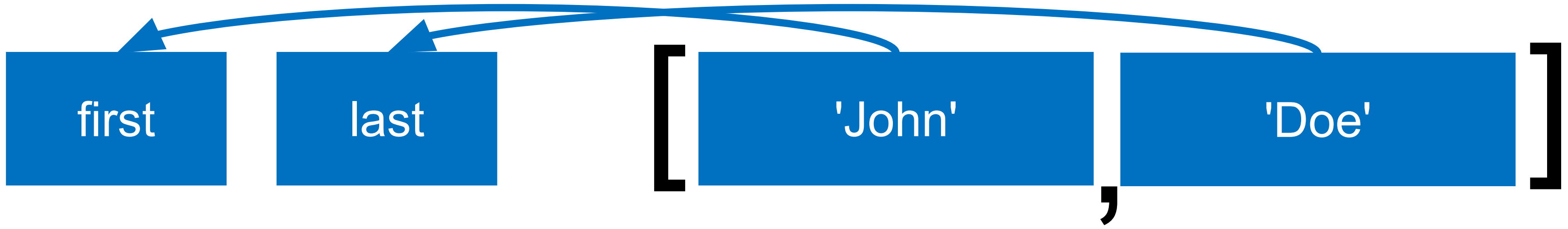
The back-tick ` indicates a template string

```
class Car {
    constructor(make, model, engine) {
        this._make = make;
        this._model = model;
        this._engine = engine;
    }
    start() {
        return `${this._make} ${this._model} with a
                ${this._engine} engine is started!`;
    }
}
```

Template String

Destructuring

Arrays can be "destructured" into variables



Using Destructuring

```
var [first, last] = ['John','Doe'];
```

Destructuring

Destructuring arrays and object literals into variables

```
var {total2, tax2} = {total:9.99, tax:.50};
var [total, tax] = [9.99, .50];


var [red, yellow, green] = ['red', 'yellow', 'green'];
console.log(`Destructuring colors: ${red} ${yellow} ${green}`);
```

Ignore specific members

```
var [red2, , green2] = ['red', 'yellow', 'green'];
console.log(`Destructuring with an ignore: ${red2} ${green2}`);
```

Destructuring Examples

Allows arrays or objects to be mapped to variables using a compact syntax

Eliminates “manual” mapping

```
var colors = ['red', 'yellow', 'green'],
    redOld = colors[0],
    yellowOld = colors[1],
    greenOld = colors[2];
```

Manual Mapping

```
var [red, yellow, green] = ['red', 'yellow', 'green'];
```

Destructuring

Rest and Default Parameters

```
class Car {

    currentYear() {
        return new Date().getFullYear();
    }


    setDetails(make = 'None', model = 'None', year = this.currentYear()) {


        console.log(make + ' ' + model + ' ' + year);
    }
}
```

make, model, and year are default parameters

Rest Parameters

Allow a variable number of parameters to be passed to a function
Identified by the triple dots ...

car.ts

Accessories is a rest parameter

```
class Car {
    setDetails(make = 'No Make', ...accessories) {
        console.log(make);


        if (accessories) {
            for (var i = 0; i < accessories.length; i++) {
                console.log('\n' + accessories[i]);
            }
        }
    }
}
```

Key ES 2015 Features

- Classes
- Block Scope
- Destructuring
- Arrow Functions
- Modules
- Default/Rest Parameters
- More…

https://github.com/DanWahlin/ES6Samples

Route Guards
(Bonus)

Route 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 Guards

Protect this route

CanActivate

Protect child routes

CanActivateChild

Before we load this code ...

CanLoad

CanDeactivate

Ideal for asking the user if they want to cancel changes before leaving

Ideal for getting data before going to the route destination

Resolve

Creating a CanActivate Guard

Implement the CanActivate interface
Make a determination if the route should be activated
Can re-navigate elsewhere

can-activate-auth.service.ts

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

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

Return true or false

Applying Guards

app-routing.module.ts

```
const routes: Routes = [
  { path: '', pathMatch: 'full', redirectTo: 'customers', },
  { path: 'login', component: LoginComponent },
  {
    path: 'customers',
    component: CustomersComponent,
    canActivate: [CanActivateAuthGuard],
    canActivateChild: [CanActivateAuthGuard],
    children: [
      { path: '', component: CustomersListComponent },
      { path: ':id', component: CustomerComponent },
    ]
  },
  { path: '**', pathMatch: 'full', component: PageNotFoundComponent },
];
```

Guard the route

Guard the children

Async Guards - Example

can-deactivate.guard.ts

```
export class CanDeactivateGuard implements CanDeactivate<CanComponentDeactivate> {

 canDeactivate(component: CanComponentDeactivate): Observable<boolean> | boolean {
   return component.canDeactivate ?
     this.toObservable(component.canDeactivate()) : true;
 }

 private toObservable(deactivate: Promise<boolean> | boolean ): Observable<boolean>
| boolean {
   const p = Promise.resolve(deactivate);
   const o = Observable.fromPromise(p);
   return o;
 }
```

AngularJS to Angular

Controllers to Components

AngularJS

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

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

  function CustomersController() {
    var vm = this;
    vm.customer = { id:100, name:'Jed' };
  }
})();
```

Angular

```
<my-customer></my-customer>



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

@Component({
  selector: 'my-customer',
  template: '<h3>{{customer.name}}</h3>'
})
export class CustomerComponent {
  customer = {id: 100, name: 'Jed' };
}
```

Structural Built-In Directives

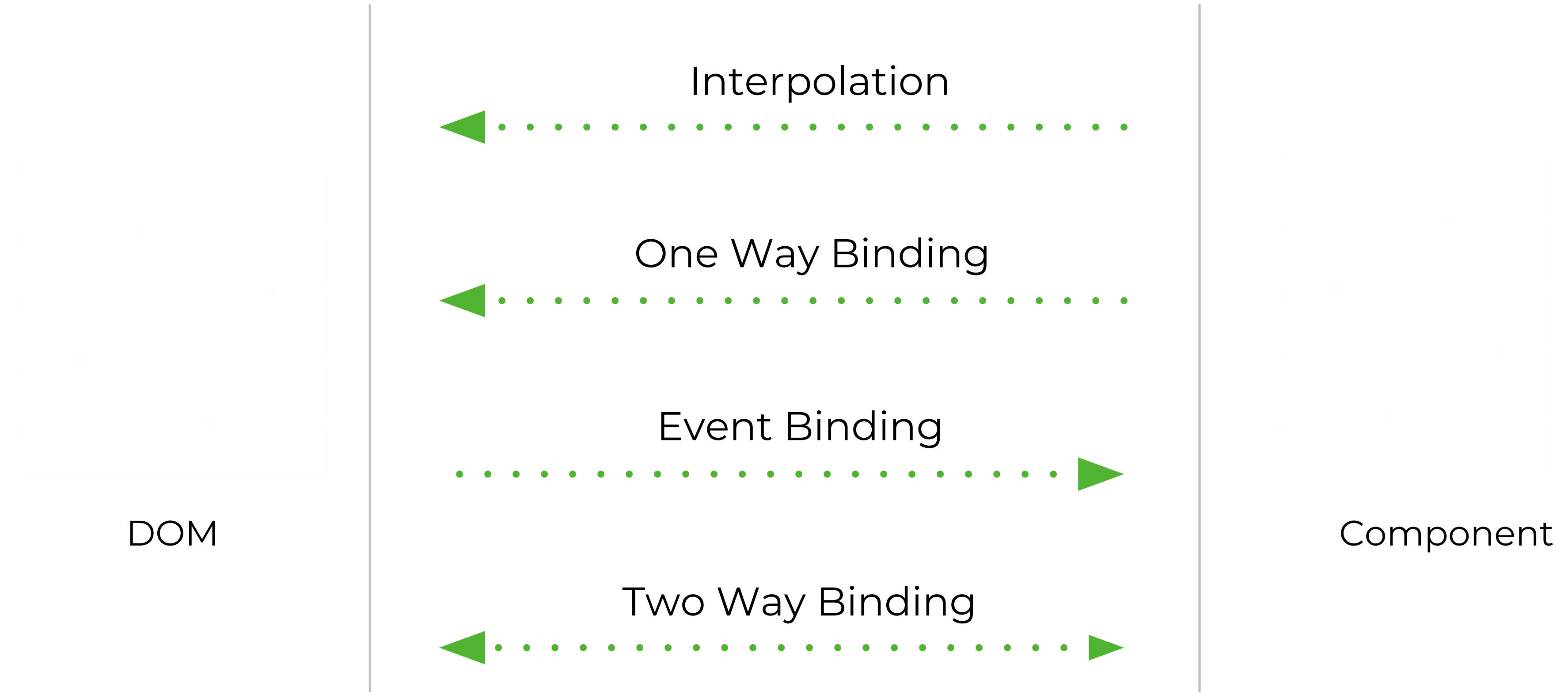
AngularJS

```
<div ng-if="vm.customers.length">
  <ul>
    <li ng-repeat="cust in vm.customers">
      {{ cust.name }}
    </li>
  </ul>
</div>
```

Angular

```
<div *ngIf="customers.length">
  <ul>
    <li *ngFor="let cust of customers">
      {{ cust.name }}
    </li>
  </ul>
</div>
```

Data Binding



Interpolation

AngularJS

```
<div>{{ vm.customer.name }}</div>
```

Angular

```
<div>{{ customer.name }}</div>
```

One-Way Binding

AngularJS

```
<div ng-bind="vm.customer.name"></div>
```

Angular

```
<div [innerText]="customer.name"></div>
```

Event Binding

AngularJS

```
<button
  ng-click="vm.submitCustomer()"></button>
```

Angular

```
<button (click)="submitCustomer()"></button>
```

Two-Way Binding

AngularJS

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

Angular

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

Angular Removes Many Directives

AngularJS

```
ng-click="saveCustomer(customer)"

ng-focus="handleFocus()"

ng-blur="handleBlur()"

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

Angular

```
(click)="saveCustomer(customer)"

(focus)="handleFocus()"

(blur)="handleBlur()"

(keyup)="handleKeyUp()"
```

Angular Template Concepts Remove 40+ AngularJS Built-In Directives

Services

AngularJS	Angular
Factories	
Services	
Providers	Class
Constants	
Values	