# Introduction to Angular

Learn some of the things!

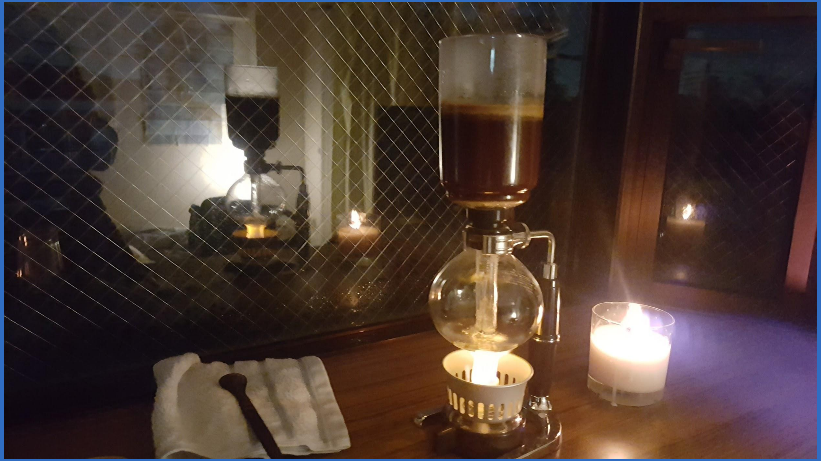
Nate Lapinski
nathan-lapinski
(github)

## About me

Full stack dev. Angular. Javascript

## About me

Full stack dev. Lots of Angular. Lots of Javascript



## Overview

## Overview

- Learn the basics of Angular 5

## Overview

- Learn the basics of Angular 5
- Code along if you want!
  https://github.com/nathan-lapinski/learn-angular-catch-pokemon

- https://stackblitz.com/github/nathan-lapinski/learn-angular-catch-pokemon/tree/master/APP-Start

  https://pokeapi.co/

## Why learn Angular?

## Why learn Angular?

Expressive HTML

## Why learn Angular?

Expressive HTML

```
//display-pokemon
<div *ngIf=”userIsLoggedIn”>
  <div *ngFor=”let poke of pokemon”>
    <span>{{poke.name}}</span>
    <img [src]=”poke.url”>
  </div>
</div>
```

## Why learn Angular?

Expressive HTML

```
//display-pokemon
<div *ngIf=”userIsLoggedIn”>
  <div *ngFor=”let poke of pokemon”>
    <span>{{poke.name}}</span>
    <img [src]=”poke.url”>
  </div>
</div>
```

```
<h1>Welcome!</h1>

<display-pokemon></display-pokemon>

<div>...
...
```

## Why learn Angular?

Expressive HTML

Data Binding

## Why learn Angular?

Expressive HTML

Data Binding

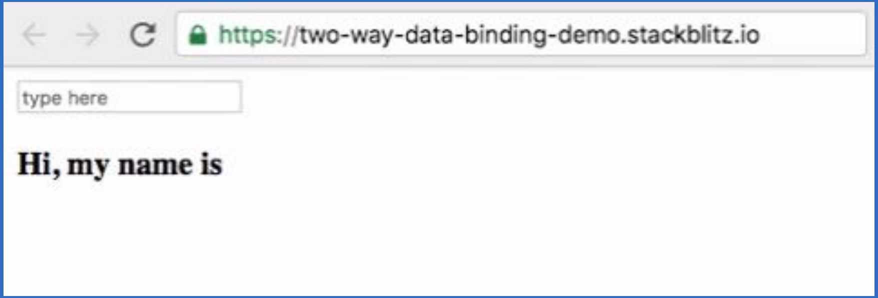
```
<input [(ngModel)]="name" type="text">
<h3>Hi, my name is {{name}}</h3>

export class AppComponent  {
 name = '';
}
```

## Why learn Angular?

Expressive HTML

Data Binding

```
<input [(ngModel)]="name" type="text">
<h3>Hi, my name is {{name}}</h3>

export class AppComponent  {
 name = '';
}
```


https://two-way-data-binding-demo.stackblitz.io

type here

**Hi, my name is**

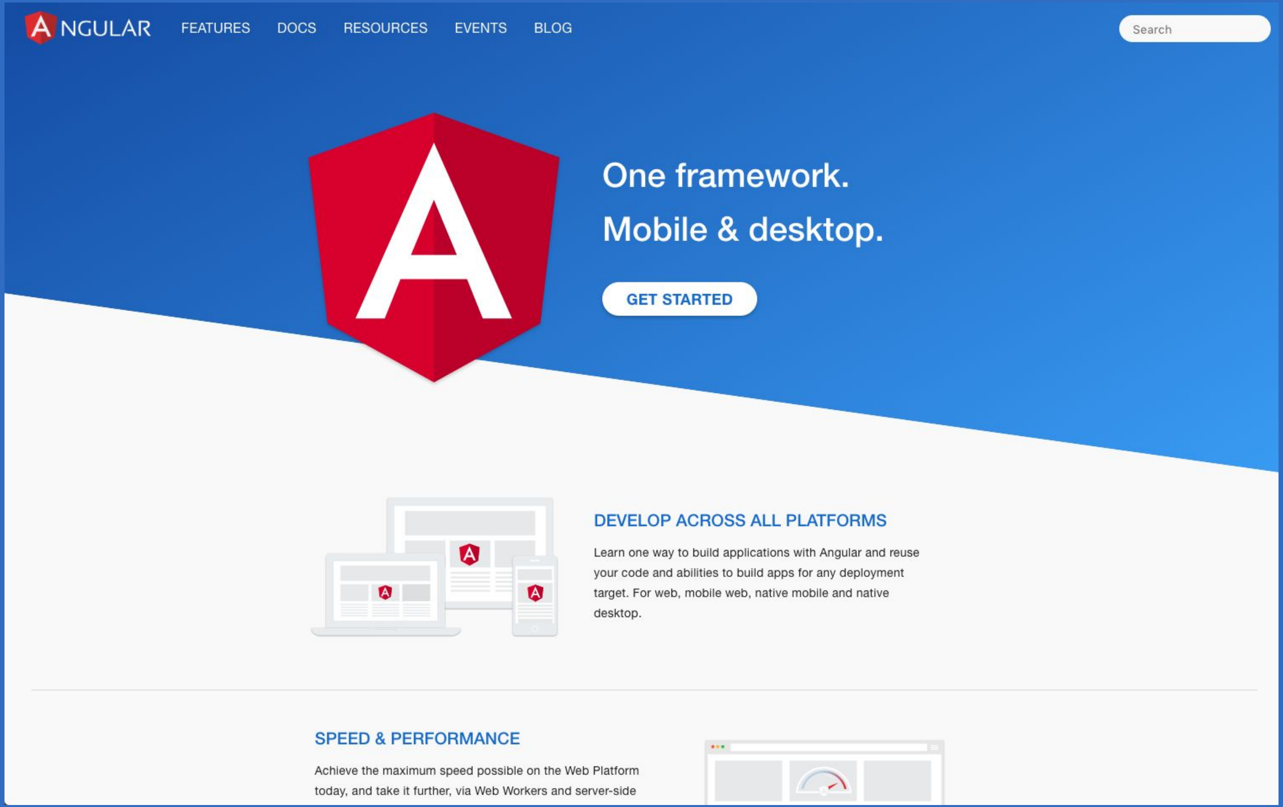
## Why learn Angular?

Expressive HTML

Data Binding

Strong Library Support

# https://angular.io/



One framework.

Mobile & desktop.

DEVELOP ACROSS ALL PLATFORMS

Learn one way to build applications with Angular and reuse your code and abilities to build apps for any deployment target. For web, mobile web, native mobile and native desktop.

SPEED & PERFORMANCE

Achieve the maximum speed possible on the Web Platform today, and take it further, via Web Workers and server-side

## Why learn Angular?

Expressive HTML

Data Binding

Strong Library Support

## Why learn Angular?

Expressive HTML

Data Binding

Strong Library Support

RxJS

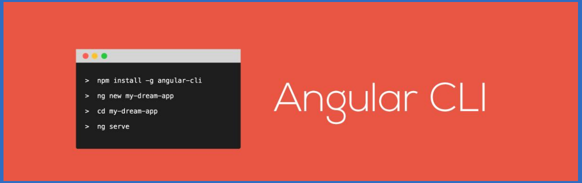
## Why learn Angular?

Expressive HTML

Data Binding

Strong Library Support

RxJS



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

Angular CLI

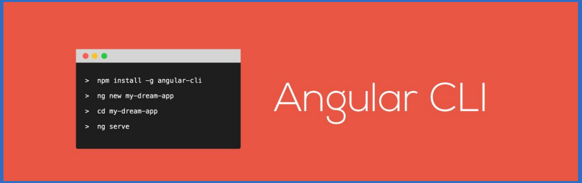
## Why learn Angular?

Expressive HTML

Data Binding

Strong Library Support

RxJS



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

Redux RxJS = NgRX

## A Tale of Two Angulars

## A Tale of Two Angulars

- Angular 1.x is referred to as *AngularJS*

## A Tale of Two Angulars

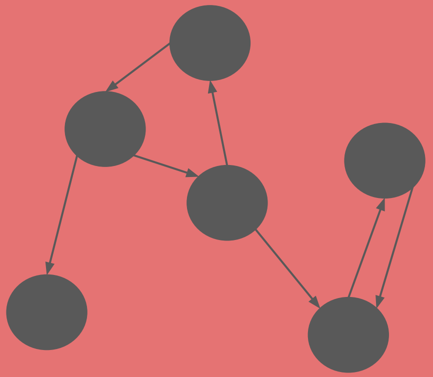
- Angular 1.x is referred to as *AngularJS*
- Angular 2+ is simply referred to as Angular

## A Tale of Two Angulars

- Angular 1.x is referred to as *AngularJS*
- Angular 2+ is simply referred to as Angular
- Why is there a new Angular?

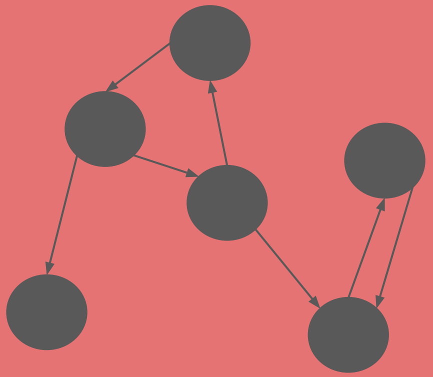
## Performance!

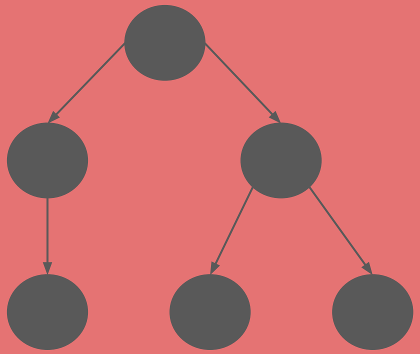
## Change Detection



AngularJS

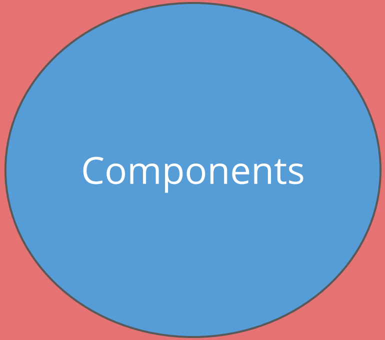
## Change Detection


AngularJS


Angular

## Concepts in Angular

## Concepts in Angular



## Concepts in Angular

Components

Dependency Injection

## Concepts in Angular

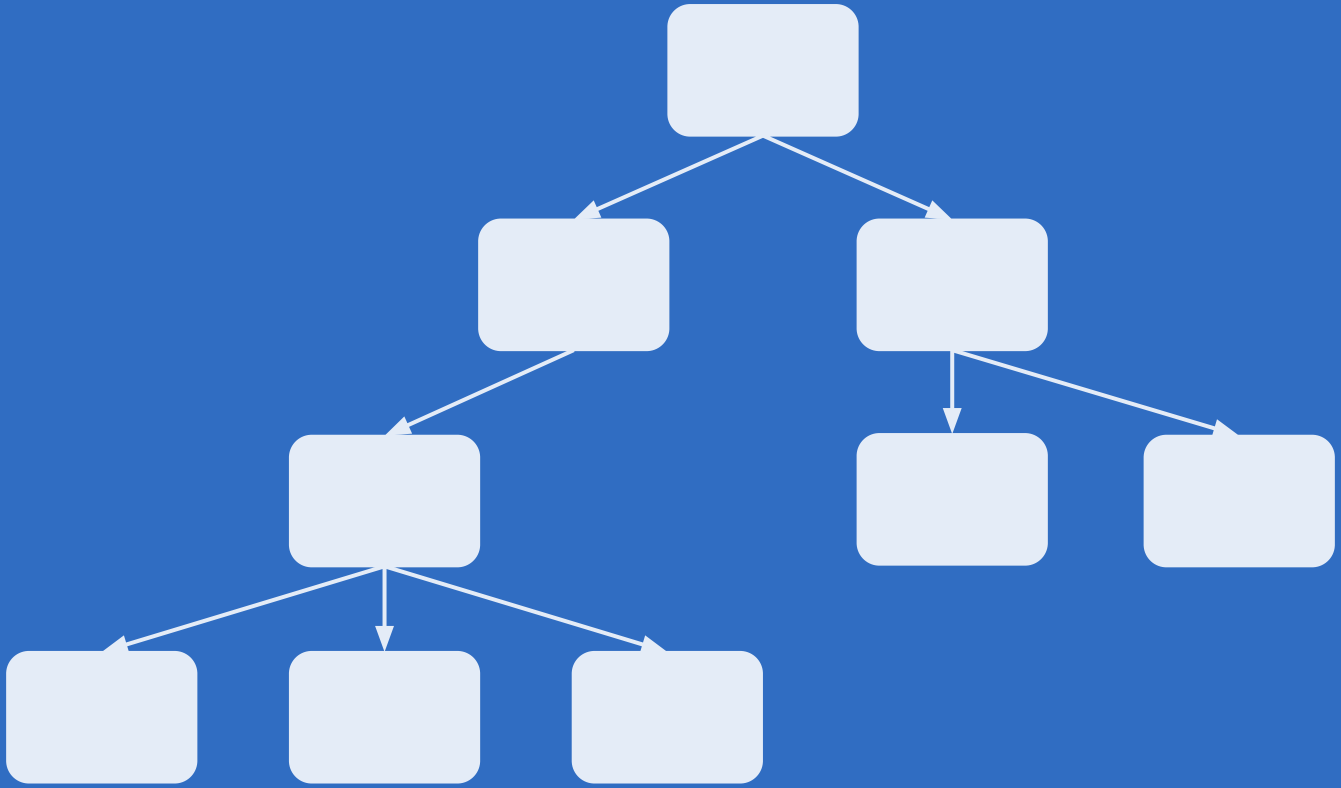
Components

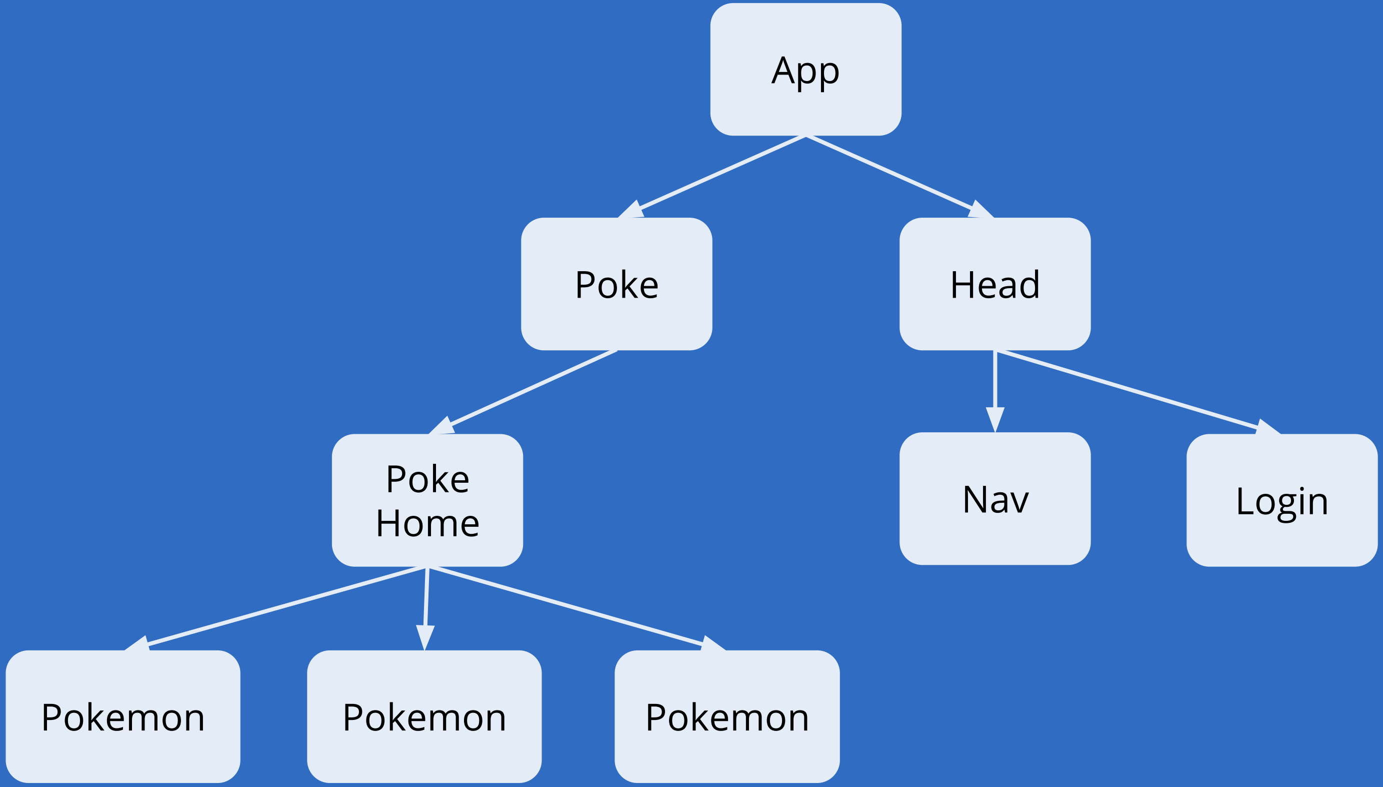
Dependency Injection

Bindings

## Components

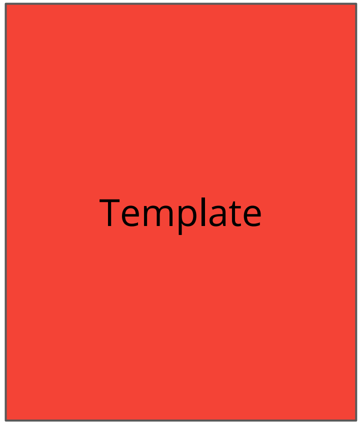
An Angular application is just a tree of components




App
Poke
Head
Poke Home
Nav
Login
Pokemon
Pokemon
Pokemon

## What is a component

## What is a component

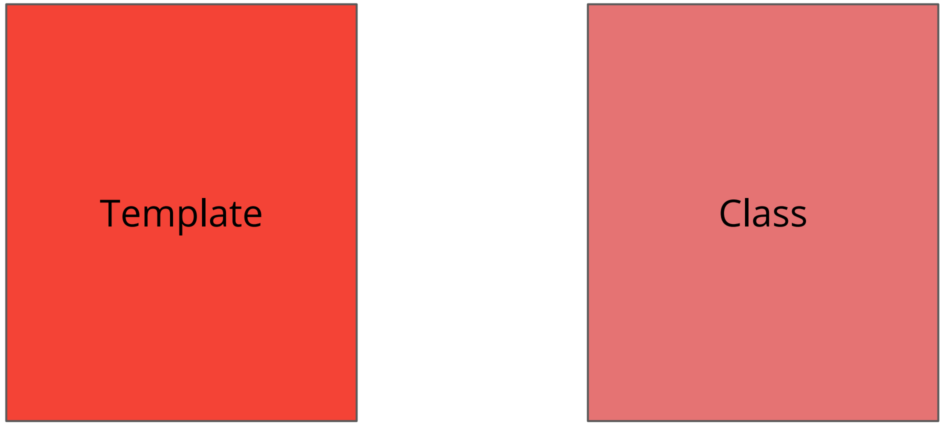


## What is a component

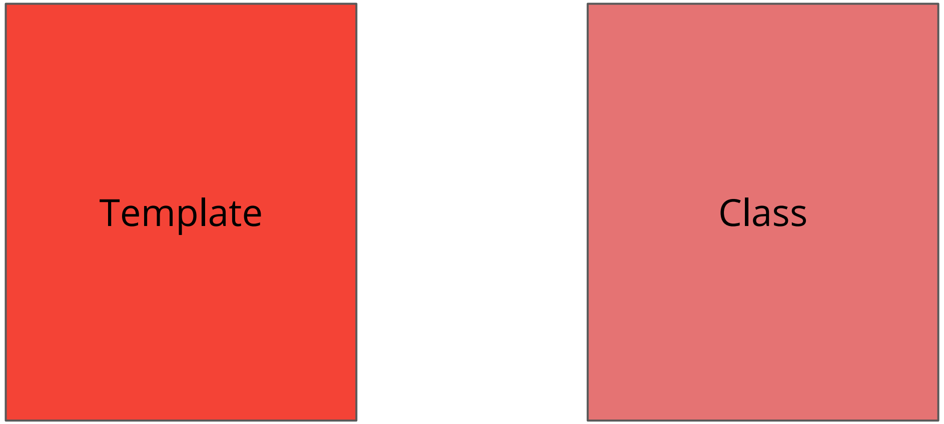
Template

```
//display-pokemon
<div *ngIf=”userIsLoggedIn”>
  <div *ngFor=”let poke of pokemon”>
    <span>{{poke.name}}</span>
    <img [src]=”poke.url”>
  </div>
</div>
```

## What is a component

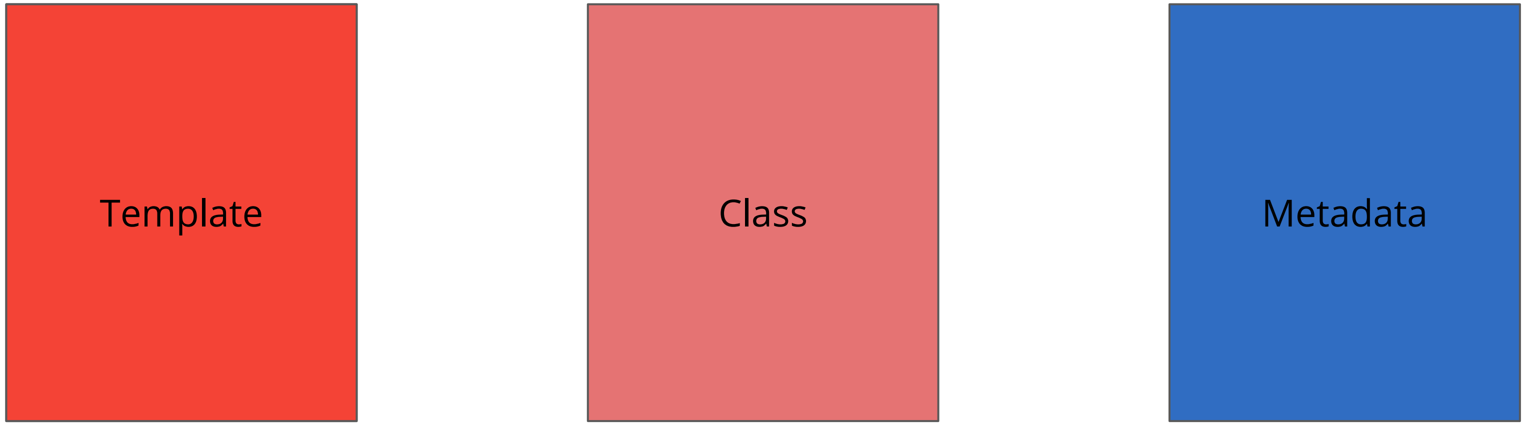


## What is a component



```
export class DisplayPokemonComponent {...}
```

## What is a component



## Example

```
export class PokemonHomeComponent {
  name: string = ‘Pokemon, go!’;
  constructor(){}
}
```

## Example

```
import { Component } from ‘@angular/core’;

@Component({
  selector: ‘pokemon-home’,
  template:
})
export class PokemonHomeComponent {
  name: string = ‘Pokemon, go!’;
  constructor(){}
}
```

## Example

```
import { Component } from ‘@angular/core’;

@Component({
  selector: ‘pokemon-home’,
  template:  `
    <div><h1>{{name}}</h1></div>
  `
})
export class PokemonHomeComponent {
  name: string = ‘Pokemon, go!’;
  constructor(){}
}
```

## Demo

Let’s build a component!

## Data Binding

Enables communication between component class and template

## Data Binding - Interpolation

{{}} syntax. One way binding from class to template.

{{pokemon.name}}

## Data Binding - Property Binding

```
<img [src]='pokemon.sprites.front_default'>
```

## Data Binding - Property Binding

```
<img [src]=’pokemon.sprites.front_default’>

<img src={{pokemon.sprites.front_default}}>
```

## Data Binding - Demo

## Event Binding

```
<button (click)='feedPokemon()'>
```

## Nested Components

An Angular application is just a tree of components

## Nested Components

An Angular application is just a tree of components

Components are frequently nested inside of other components, creating a parent/child relationship

## Nested Components

An Angular application is just a tree of components

Components are frequently nested inside of other components, creating a parent/child relationship

```
<parent-component>
  <child-component></child-component>
<parent-component>
```

## View Layer Architecture

## View Layer Architecture

Container Components (smart): Contain data and some business logic

# View Layer Architecture

Container Components (smart): Contain data and some business logic

Presentational Components (dumb): Display data

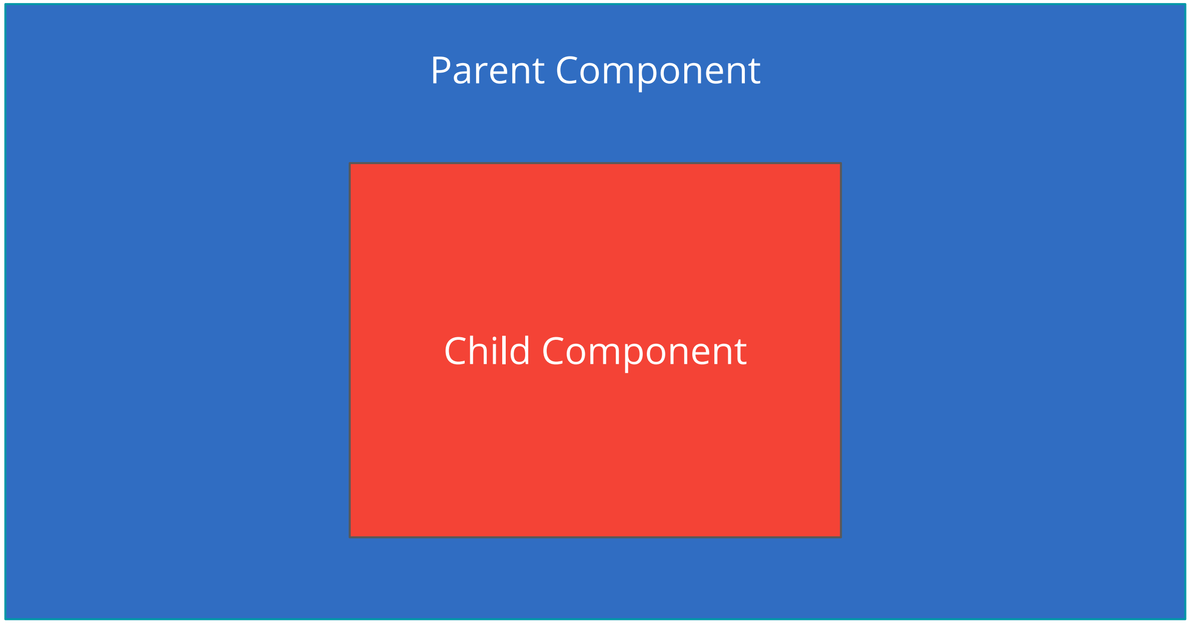
## View Layer Architecture

Container Components (smart): Contain data and some business logic

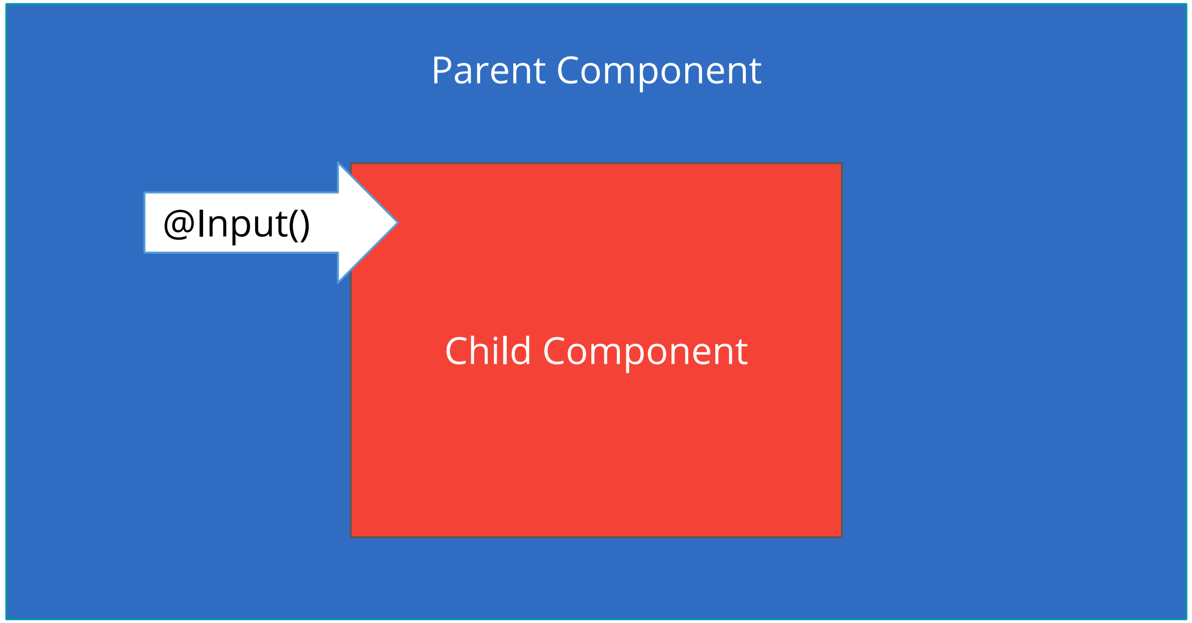
Presentational Components (dumb): Display data

https://blog.angular-university.io/angular-2-smart-components-vs-presentation-components-whats-the-difference-when-to-use-each-and-why/

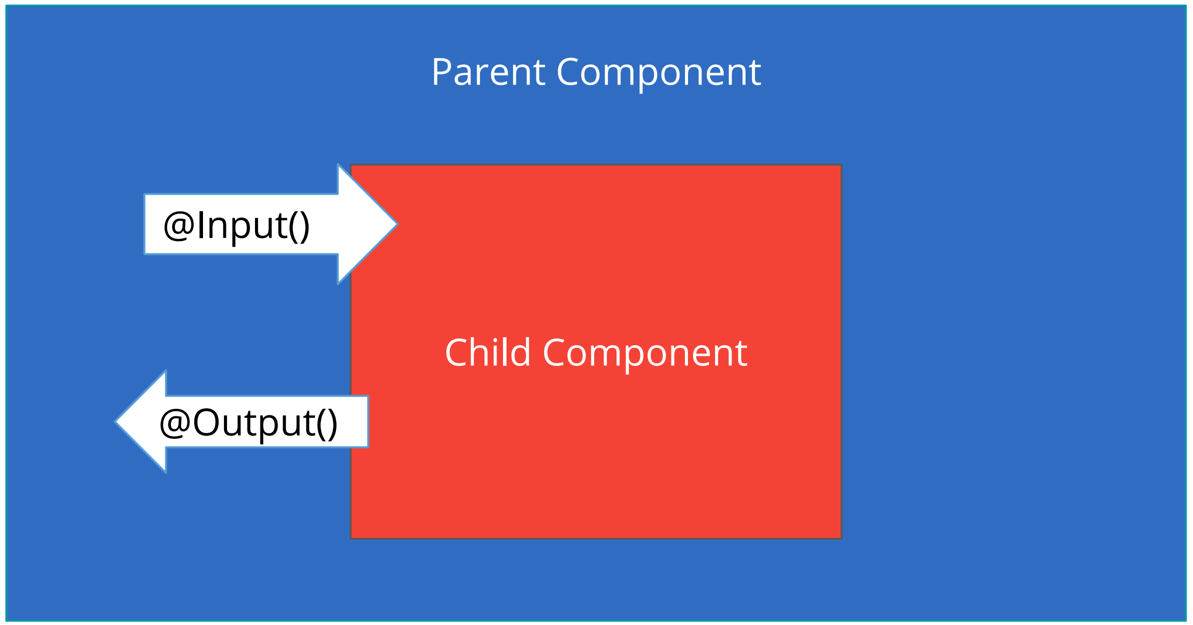
# Nested Component Communication



# Nested Component Communication



# Nested Component Communication



@victorsavkin

*“Data flows into a component via input properties. Data flows out of a component via output properties”*

*- Victor Savkin, creator of Angular’s dependency injection and change detection mechanisms*

*https://vsavkin.com/the-core-concepts-of-angular-2-c3d6cbe04d04*

## Lifecycle Hooks

Every component has a lifecycle managed by Angular

## Lifecycle Hooks

Every component has a lifecycle managed by Angular

Angular creates it, renders it, creates and renders its children, checks for changes to its data, and destroys it before removing it from the DOM.

## Lifecycle Hooks

Every component has a lifecycle managed by Angular

Angular creates it, renders it, creates and renders its children, checks for changes to its data, and destroys it before removing it from the DOM.

Angular offers **lifecycle hooks**, which provide visibility into this cycle.

- constructor
- ngOnChanges
- ngOnInit
- ngDoCheck
    - ngAfterContentInit
    - ngAfterContentChecked
    - ngAfterViewInit
    - ngAfterViewChecked
- ngOnDestroy

## Service

A class with a focused purpose.

## Service

A class with a focused purpose.

- Independent from any one component

## Service

A class with a focused purpose.

- Independent from any one component
- Provide shared data or logic across components

## Service

A class with a focused purpose.

- Independent from any one component
- Provide shared data or logic across components
- Encapsulate external logic (http, etc)

## Service

A class with a focused purpose.

- Independent from any one component
- Provide shared data or logic across components
- Encapsulate external logic (http, etc)
- Makes components easier to test, debug, and reuse

## Dependency Injection

## Dependency Injection

A design pattern in which a class receives its dependencies from an external source rather than creating them itself.

## Services

Services are registered as dependencies in providers array of a module in an application

## Services

Services are registered as dependencies in providers array of a module in an application

By registering a service in the module, we can let Angular inject it for us in any component or service that uses it as a dependency.

## How to build a service

- Create a class

## How to build a service

- Create a class
- Use the @Injectable decorator

## How to build a service

- Create a class
- Use the @Injectable decorator
- Import any needed files

```
export class PokemonDataService {
  public getPokemon(): any[] {

  }
}
```

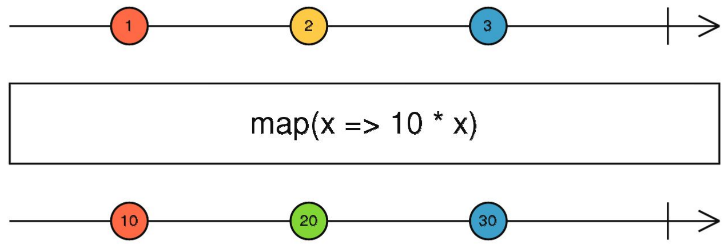
```
@Injectable()
export class PokemonDataService {
  public getPokemon(): any[] {

  }
}
```

```
import { Injectable } from ‘@angular/core’

@Injectable()
export class PokemonDataService {
  public getPokemon(): any[] {

  }
}
```

RxJS

## Observables and Reactive Extensions

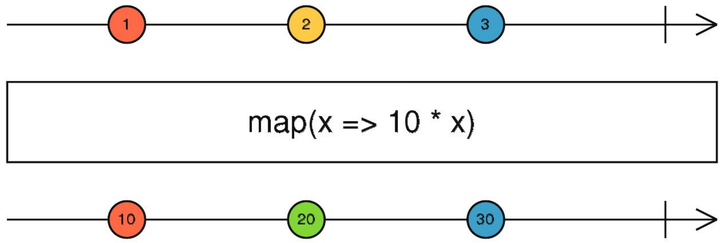
Treat events as a collection



# Observables and Reactive Extensions

Treat events as a collection

Think of them as asynchronous arrays

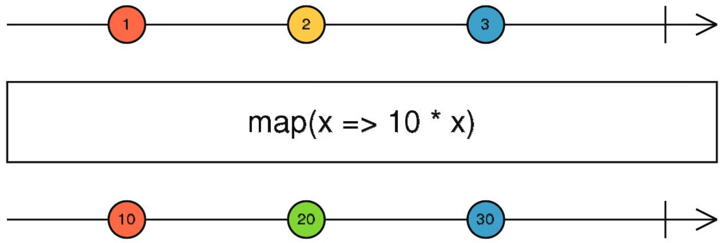


## Observables and Reactive Extensions

Treat events as a collection

Think of them as asynchronous arrays

May eventually make it into JS (stage 1). Use RxJS for now



# Promises vs Observables

| Promise | Observable |
|---|---|

## Promises vs Observables

| Promise | Observable |
|---|---|
| Provides a single future value | Emits multiple values over time |

## Promises vs Observables

| Promise | Observable |
|---|---|
| Provides a single future value | Emits multiple values over time |
| Immediate execution | Lazy |

# Promises vs Observables

| Promise | Observable |
|---|---|
| Provides a single future value | Emits multiple values over time |
| Immediate execution | Lazy |
| Not cancellable | Cancellable via unsubscribe() |

## Promises vs Observables

| Promise | Observable |
|---|---|
| Provides a single future value | Emits multiple values over time |
| Immediate execution | Lazy |
| Not cancellable | Cancellable via unsubscribe() |
| No operator support | Supports many operators, such as map, filter expand, etc |

## Wrap Up

Resources and exercises are in these slides.

## Resources

Angular docs: https://angular.io/

Angular testing guide: https://angular.io/guide/testing

Angular routing: https://angular.io/guide/router

Change Detection:
https://blog.angular-university.io/how-does-angular-2-change-detection-really-work/

Unidirectional Data Flow:
https://blog.angular-university.io/angular-2-what-is-unidirectional-data-flow-development-mode/

## Resources

My favorite blog on advanced Angular topics: https://blog.angularindepth.com/

Intro to Observables and RxJS: https://gist.github.com/staltz/868e7e9bc2a7b8c1f754

Diagrams for understanding RxJS Operators:
https://blog.angularindepth.com/learn-to-combine-rxjs-sequences-with-super-intuitive-interactive-diagrams-20fce8e6511

## Additional Exercises

Three tasks to extend the app!

## Exercises

Descriptions in APP-Finished.

add_delete_button_description.md

add_nickname_option_description.md

add_parallel_requests_description.md

Solutions are in APP-Solutions

THANK YOU!