

Front end engineering for the web - Angular

Crelan – December 2023

de

Training objective

After this training you should be able to:

- Set up Angular projects
- Create and maintain Angular apps
- Make sure the next developer doesn't hate your code

Content & schedule

1. Angular what?
2. Angular CLI
3. Architecture
4. Debugging
5. Making it look nice!
6. Reactive Programming
7. Routing
8. QA
9. I18N

How to get most out of this course?



Ask **questions!**



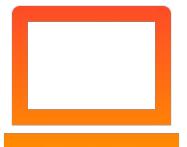
Notify your trainer if the **pace**
is too high



Take **notes**



Listen actively & stay **focused**



If applicable, **code along** with
your trainer



Participate & join the
discussion!

Training resources

<http://tinyurl.com/fee-2023>

IP disclaimer

The contents of this presentation can be downloaded but rights remain with AE; content cannot be duplicated for new courses.

Angu-what?



de

“Angular is an open source platform and framework for building client applications in HTML and TypeScript.”



<https://angular.dev>

NOT TO BE CONFUSED WITH  **ANGULARJS**
by Google



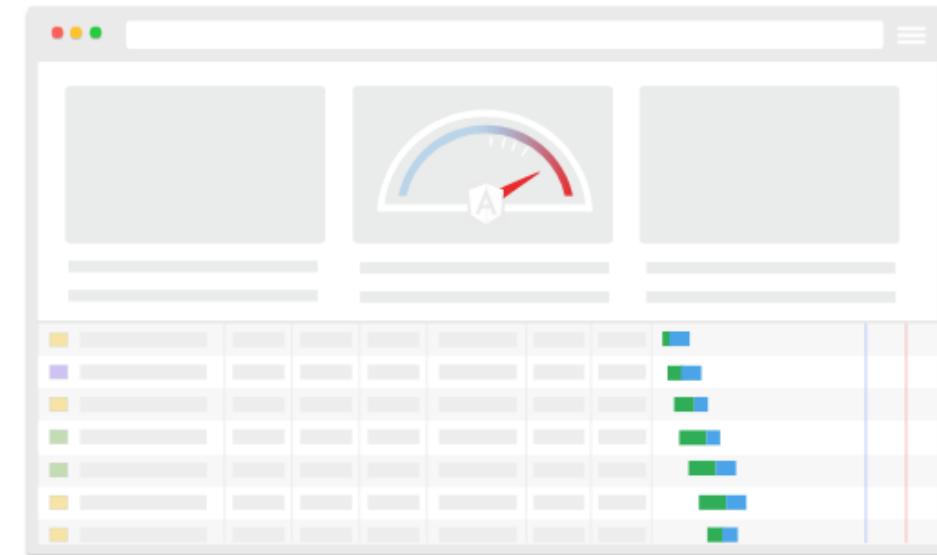
Speed & Performance

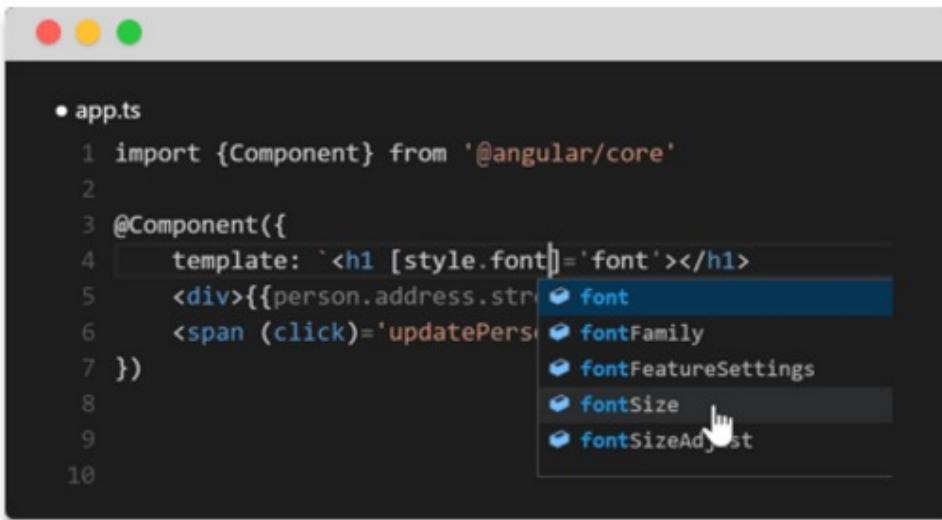
Achieve the maximum speed possible on the Web Platform today, and take it further, via Web Workers and server-side rendering.

Angular puts you in control over scalability. Meet huge data requirements by building data models on RxJS, Immutable.js or another push-model.

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.





A screenshot of an IDE showing an Angular component template. The code is as follows:

```
• app.ts
1 import {Component} from '@angular/core'
2
3 @Component({
4   template: `<h1 [style.font]=`font`></h1>
5   <div>{{person.address.str}
6     <span (click)=`updatePerson`>
7   })`
```

The cursor is over the word 'font' in the template attribute, and a tooltip shows the following options:

- font
- fontFamily
- fontFeatureSettings
- fontSize
- fontSizeAdjust

Incredible Tooling

Build features quickly with simple, declarative templates. Extend the template language with your own components and use a wide array of existing components. Get immediate Angular-specific help and feedback with nearly every IDE and editor. All this comes together so you can focus on building amazing apps rather than trying to make the code work.

Loved by Millions

From prototype through global deployment, Angular delivers the productivity and scalable infrastructure that supports Google's largest applications.



Angular What?

- Ecosystem of @angular libraries
 - Pro: (almost) everything 'out of the box'
 - Con: Tightly coupled with framework, customisation can be tricky
- Mature for business environment
- CLI to enforce style and reduce repetitive coding

Angular CLI



What can it do?

- Bootstrap applications
- Generate code
 - Aka schematics => custom or predefined
- Build tool
- Dev environment

=> Very useful and big advantage

Angular CLI

- Uses Vite behind the scenes
- Abstracts complex Vite config
- Sets up an ALM workflow for you
 - Serve
 - Build
 - Test
 - Lint
 - E2e
- Commands available as npm scripts

How?

- Available as an npm package: `npm i -g @angular/cli`



- Provides **ng** command to CLI (if installed globally)

```
ng ...
```

Angular CLI – styleguide



<https://angular.io/styleguide>

Angular CLI – styleguide

- Use **feature** modules over technical modules
 - All-books, my-books,...
 - Components, services,...
- Create a **shared module for reusable things**
 - UtilityService, ApiService,...
 - Dumb components

Angular CLI – styleguide

- File names are **kebab-cased** and **contain type**
 - all-books.component.ts
 - api.service.ts
- Class names are **PascalCased** and **contain type**
 - class AllBooksComponent
- Properties and functions are **camelCased**
 - books: Array<Book>;
 - getAllBooks(): Array<Book> { }
 - No _ before private functions
 - No UPPERCASED_CONSTANT_NAMES

Generate app

```
ng new my-app
```

- Creates a directory **my-app**
- Creates a new angular application **my-app** in a directory with that name
- Interesting options
 - --style
 - --prefix

Build

- Uses Vite to **build and bundle JS & CSS**
- Result is written to outDir, specified in **angular.json**

```
ng build
```

- Also **minifies** and **uglifyes** bundles
- Adds **hashes** to bundle names

```
ng build --prod
```

Exercise

- MAC: <https://github.com/creationix/nvm#installation>
- Windows: <https://github.com/coreybutler/nvm-windows>
 - Install directly in C:\ to prevent long path errors

```
nvm install 20.10.0
```

```
nvm use 20.10.0
```

```
npm i -g @angular/cli
```

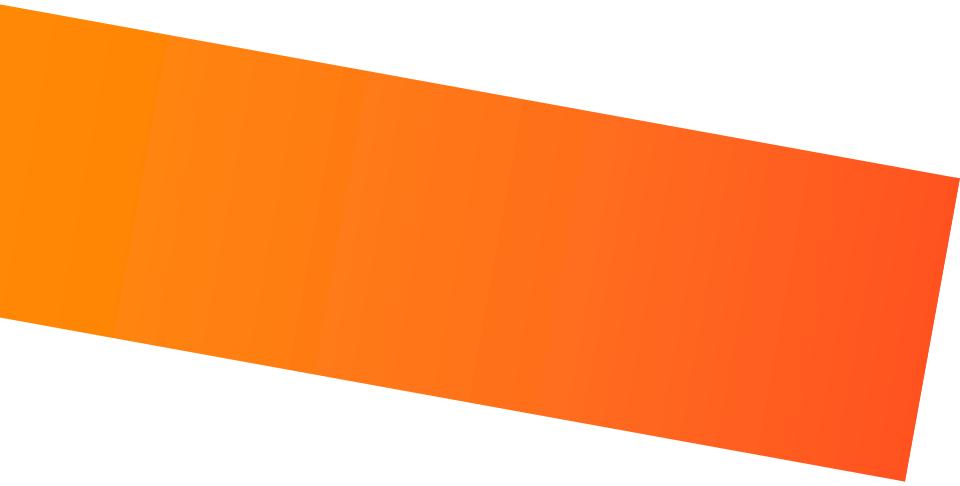
Exercise

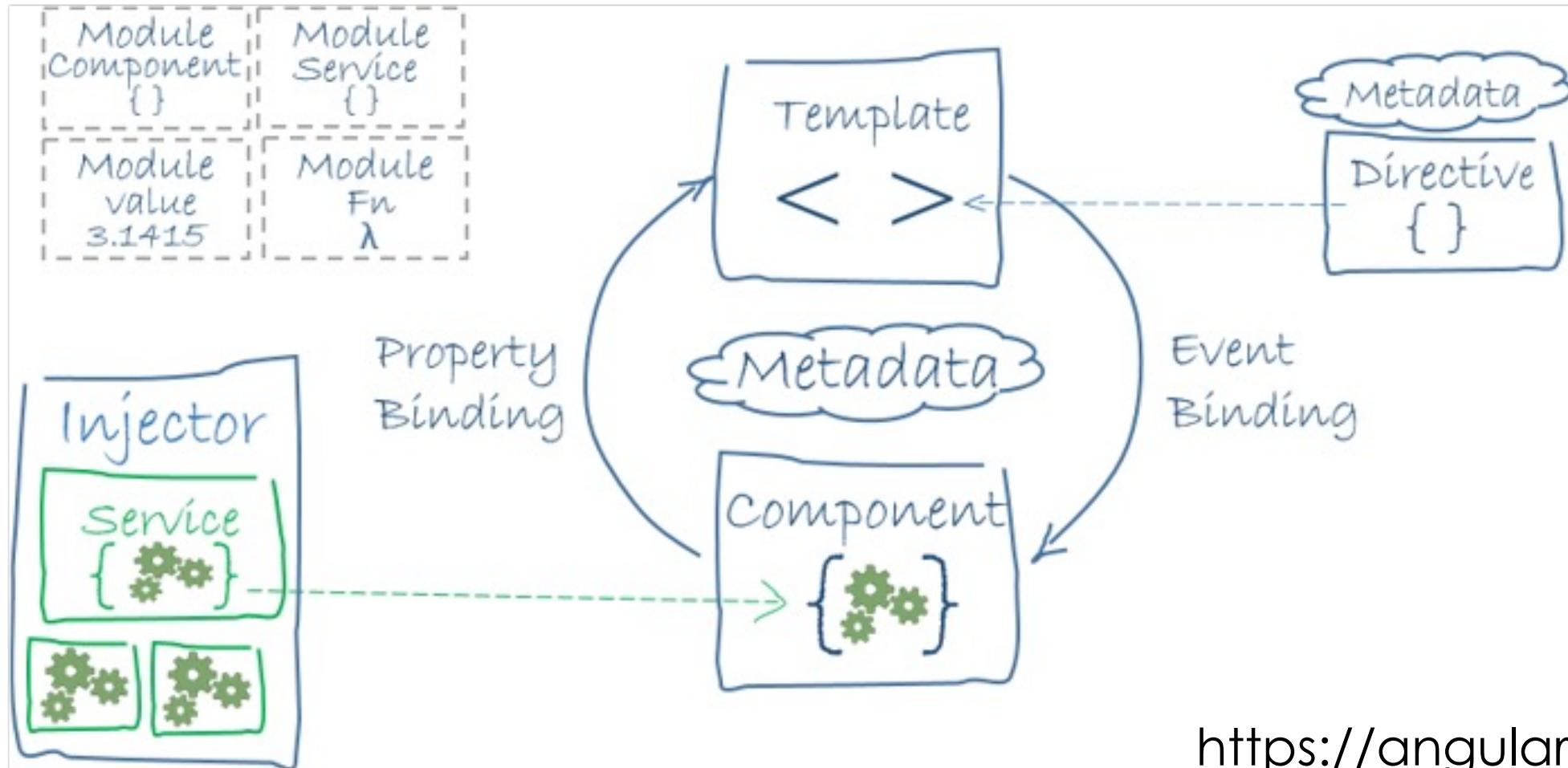
```
git clone https://github.com/AE-nv/fee-for-the-web-angular-2023
```

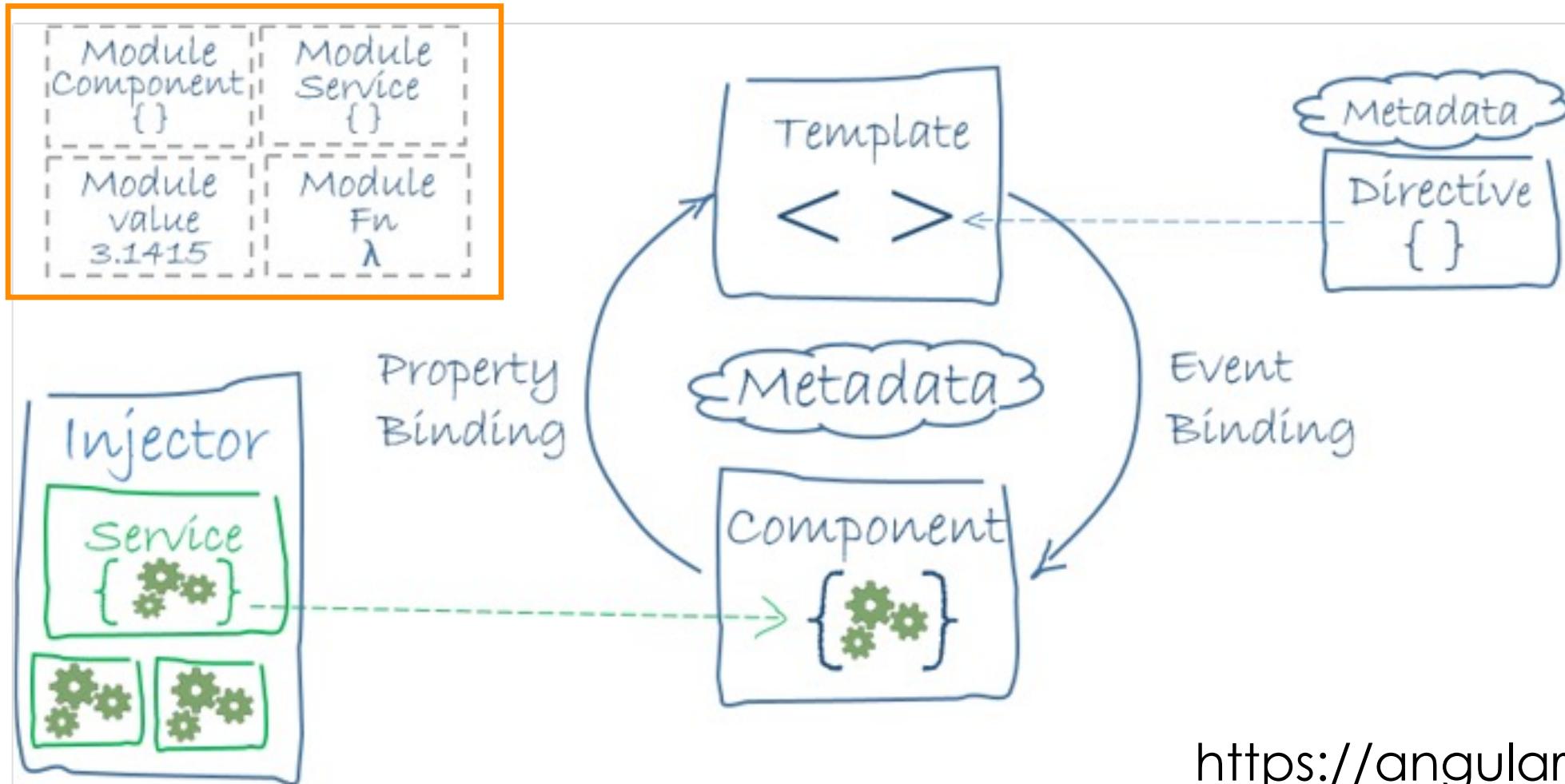
```
cd front-end
```

```
ng serve      or      npm start
```

Architecture







Modules

@NgModule

- Module groups related code: ‘a container for a cohesive block of code dedicated to an application domain, a workflow, or a closely related set of capabilities’
- Everything structured in modules
 - Defines what modules it depends on
 - Defines what it exposes to the outside
 - Provide compilation context
- One root module to bootstrap application

```
ng generate module my-module
```

```
ng g m my-module
```

Modules Example

```
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3
4
5 import { AppComponent } from './app.component';
6
7
8 @NgModule({
9   declarations: [
10     AppComponent
11   ],
12   imports: [
13     BrowserModule
14   ],
15   providers: [],
16   bootstrap: [AppComponent]
17 })
18 export class AppModule { }
```

Standalone apps (from ng v17)



- Simplicity
- Smaller bundle sizes
- Easier refactoring
- Lazy loading support

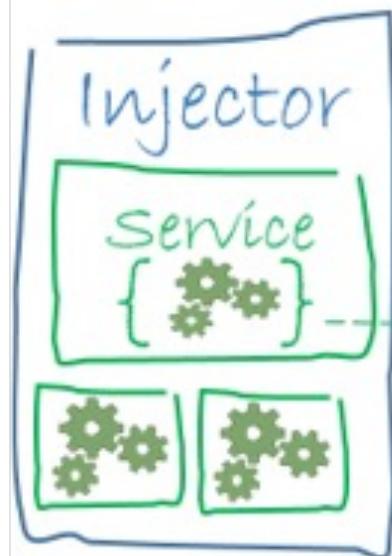
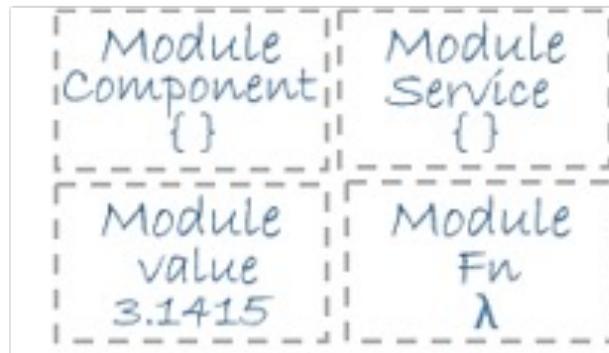
- More complex dependency management

→ No shared dependencies or relations with other components

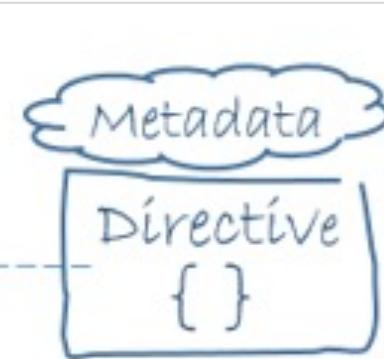
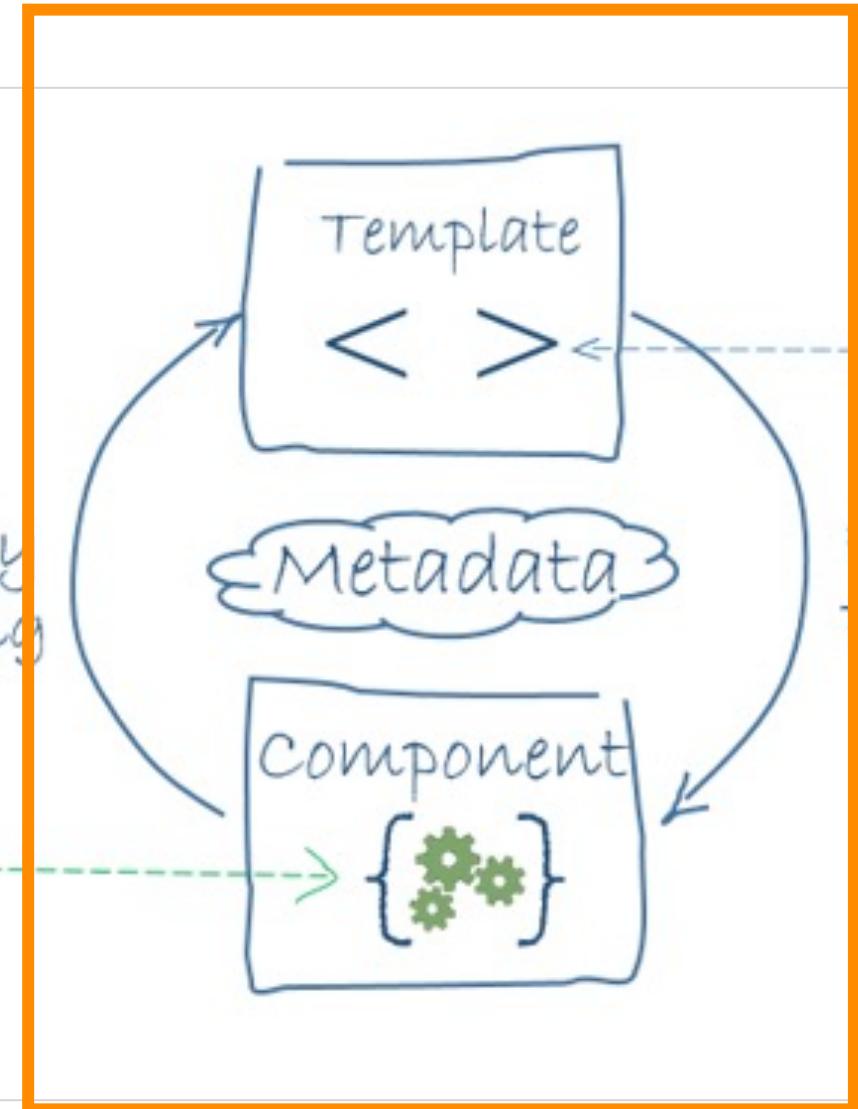
Exercise

```
git checkout project-setup
```

- Create a module named `shared`
- Create a routing module named `all-pokemon`
- Create a routing module named `my-pokemon`



Property
Binding



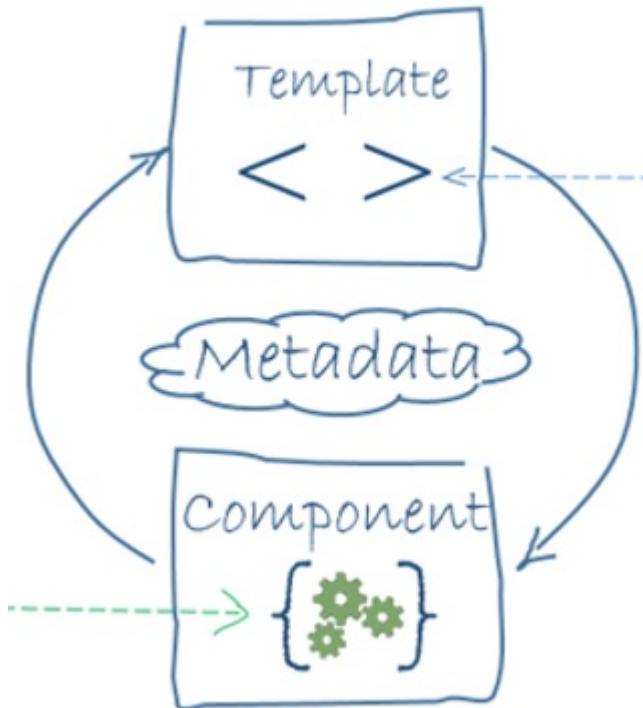
Event
binding

<https://angular.io>

Components

@Component

A component is a building block of the application, it contains:



Template (HTML), how is it STRUCTURED

Styling (CSS or Sass or ...), how does it LOOK

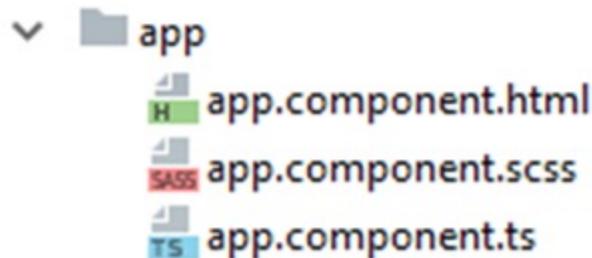
Metadata to describe coupling between these parts

Logic (TypeScript class), what does my component DO

Components

```
ng generate component app
```

```
ng g c app
```

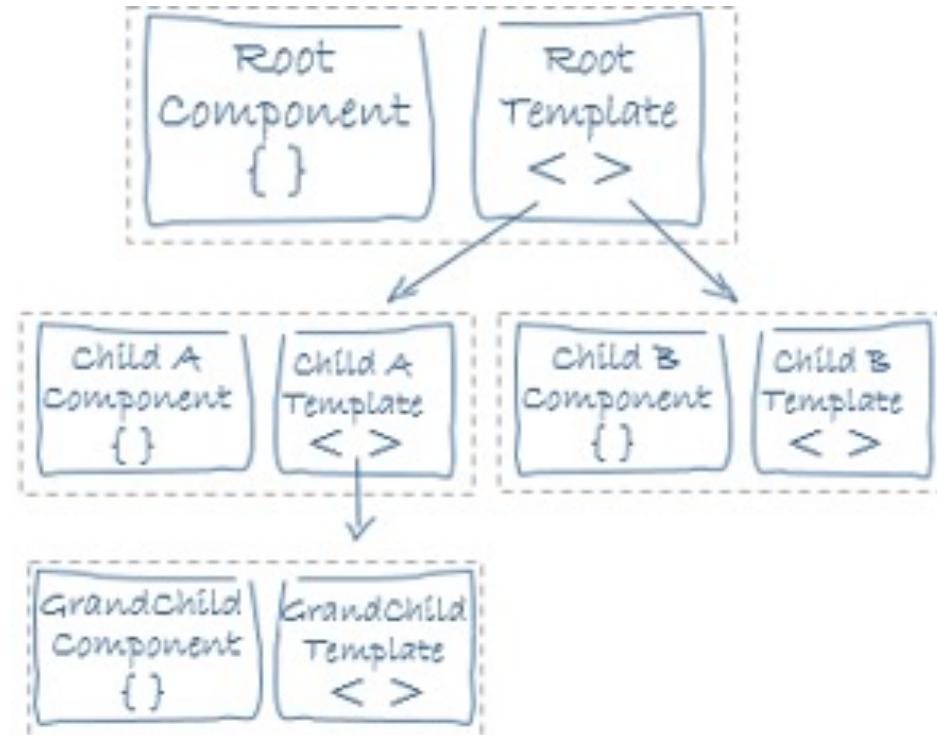


```
<> all-pokemon.component.html  
& all-pokemon.component.scss  
TS all-pokemon.component.spec.ts  
TS all-pokemon.component.ts
```

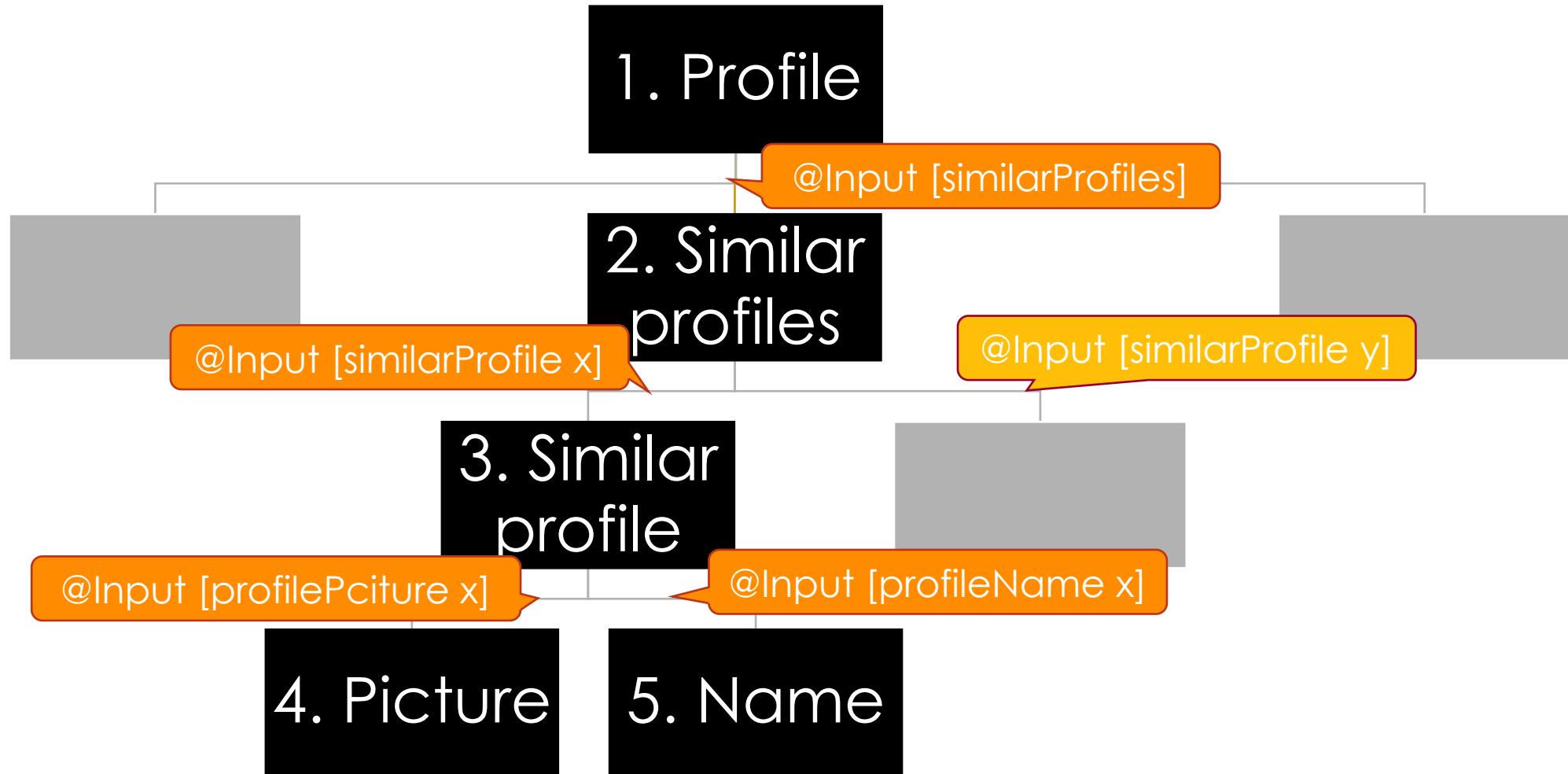
```
1 import { Component } from '@angular/core';  
2  
3 @Component({  
4   selector: 'pokedex-all-pokemon',  
5   templateUrl: './all-pokemon.component.html',  
6   styleUrls: ['./all-pokemon.component.scss']  
7 })  
8 export class AllPokemonComponent {  
9  
10 }
```

Components

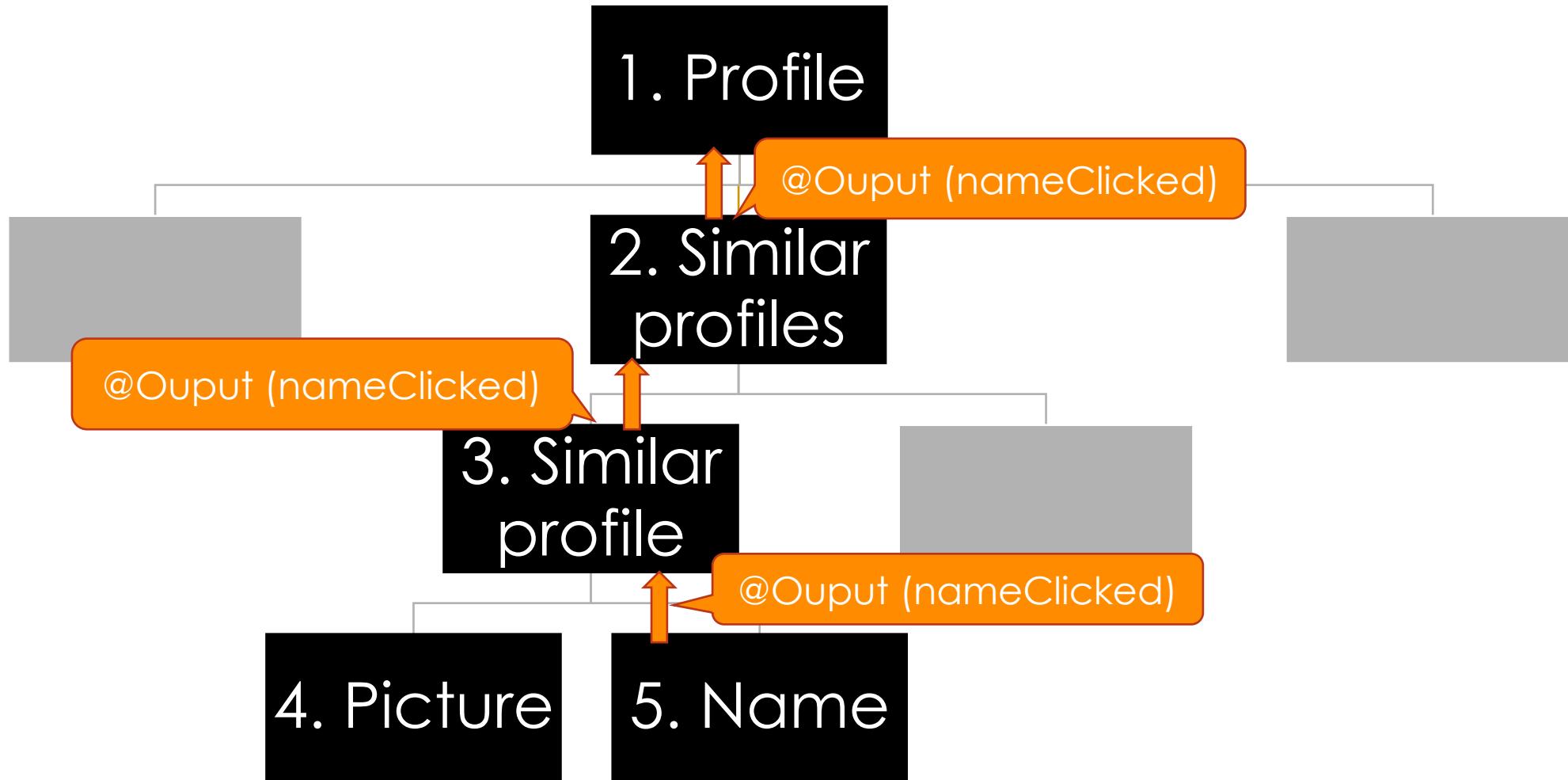
Hierarchically structured

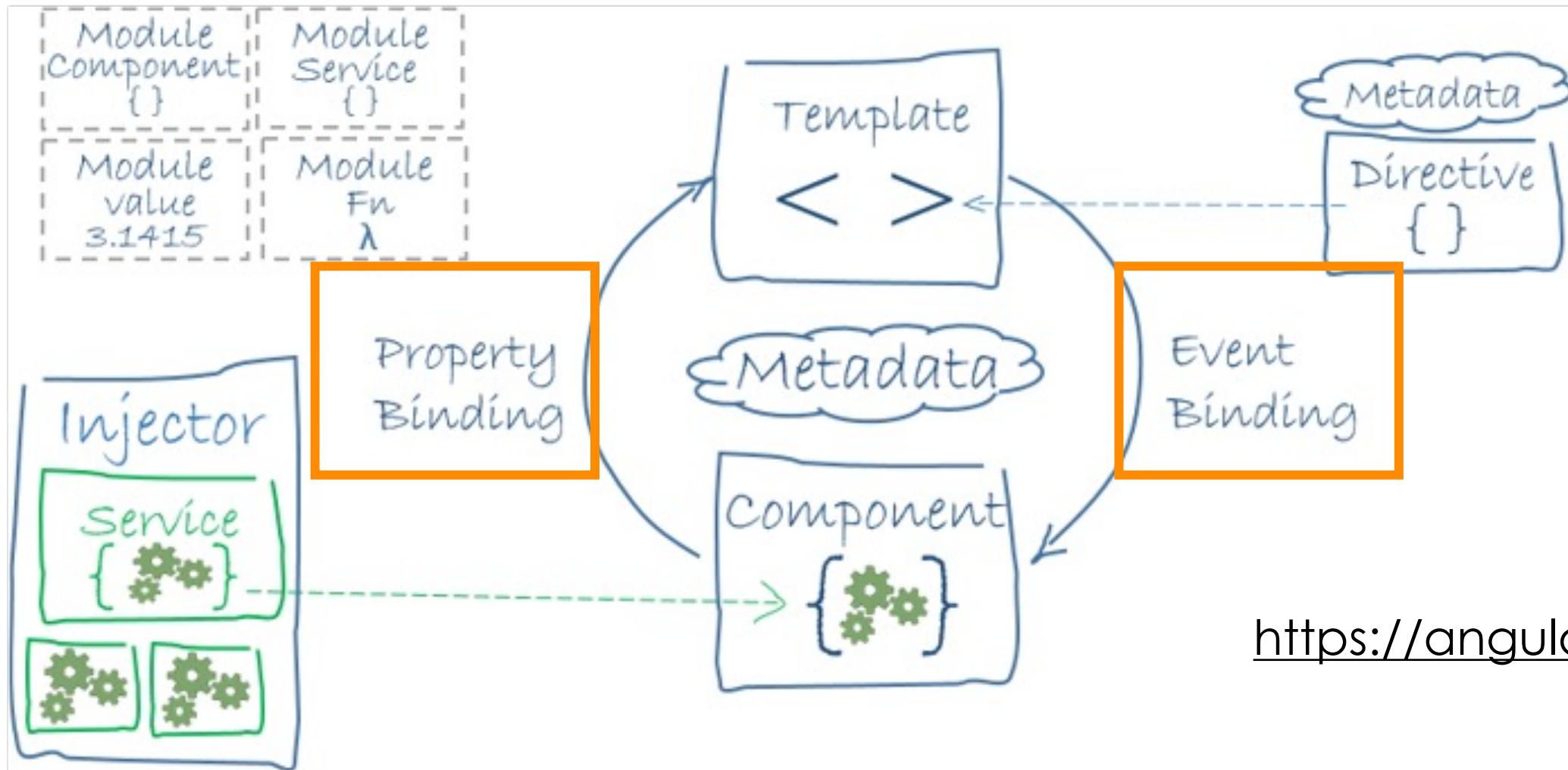


Components example



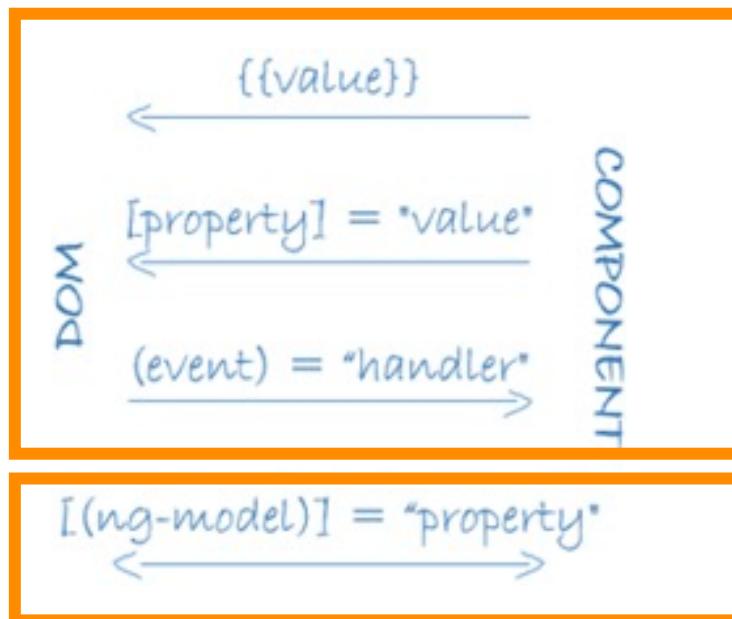
Components example





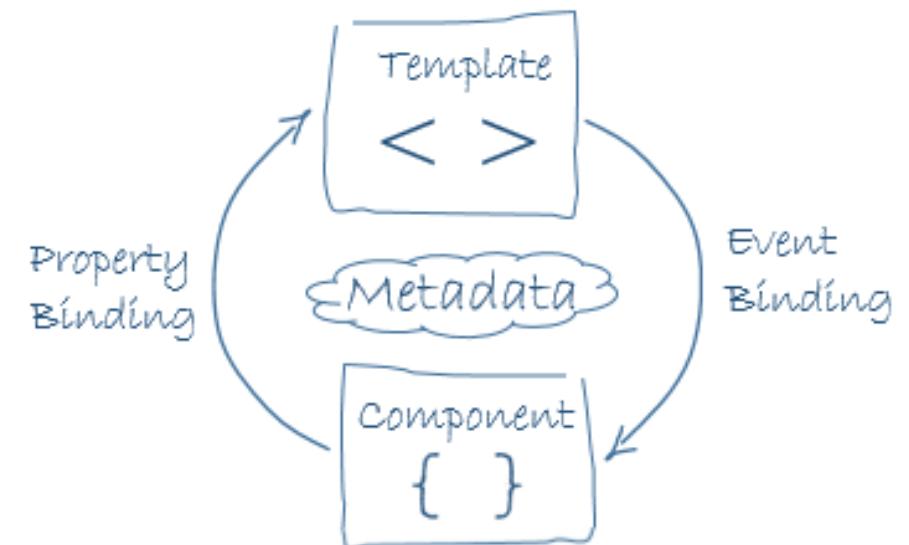
Data Binding

- Angular supports two-way data binding
- Component coordinates parts of the template
- One way to communicate within component hierarchy



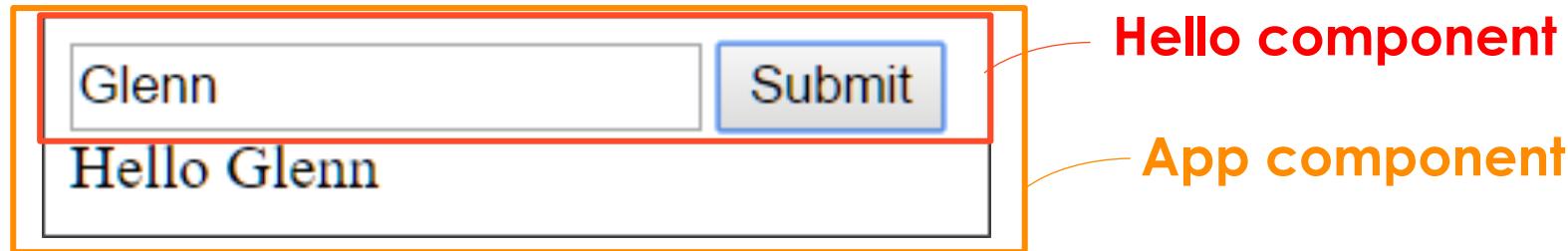
One-way binding

Two-way binding



Example

- Fill in name in input field
- After pressing 'submit', welcome message is displayed below



- Responsibility of 'hello' component
 - Get name via **INPUT**
 - Emit name after submit via **OUTPUT**

hello.component.ts

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

@Component({
  selector: 'ae-hello',
  templateUrl: './hello.component.html',
  styleUrls: ['./hello.component.scss']
})
export class HelloComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }
}
```

Imports

Component descriptor

Component logic

hello.component.ts

```
import { Component, OnInit, Input, EventEmitter, Output } from '@angular/core';

@Component({
  selector: 'ae-hello',
  templateUrl: './hello.component.html',
  styleUrls: ['./hello.component.scss']
})
export class HelloComponent implements OnInit {
  @Input() name: string = '';
  @Output() nameEntered = new EventEmitter();

  constructor() { }

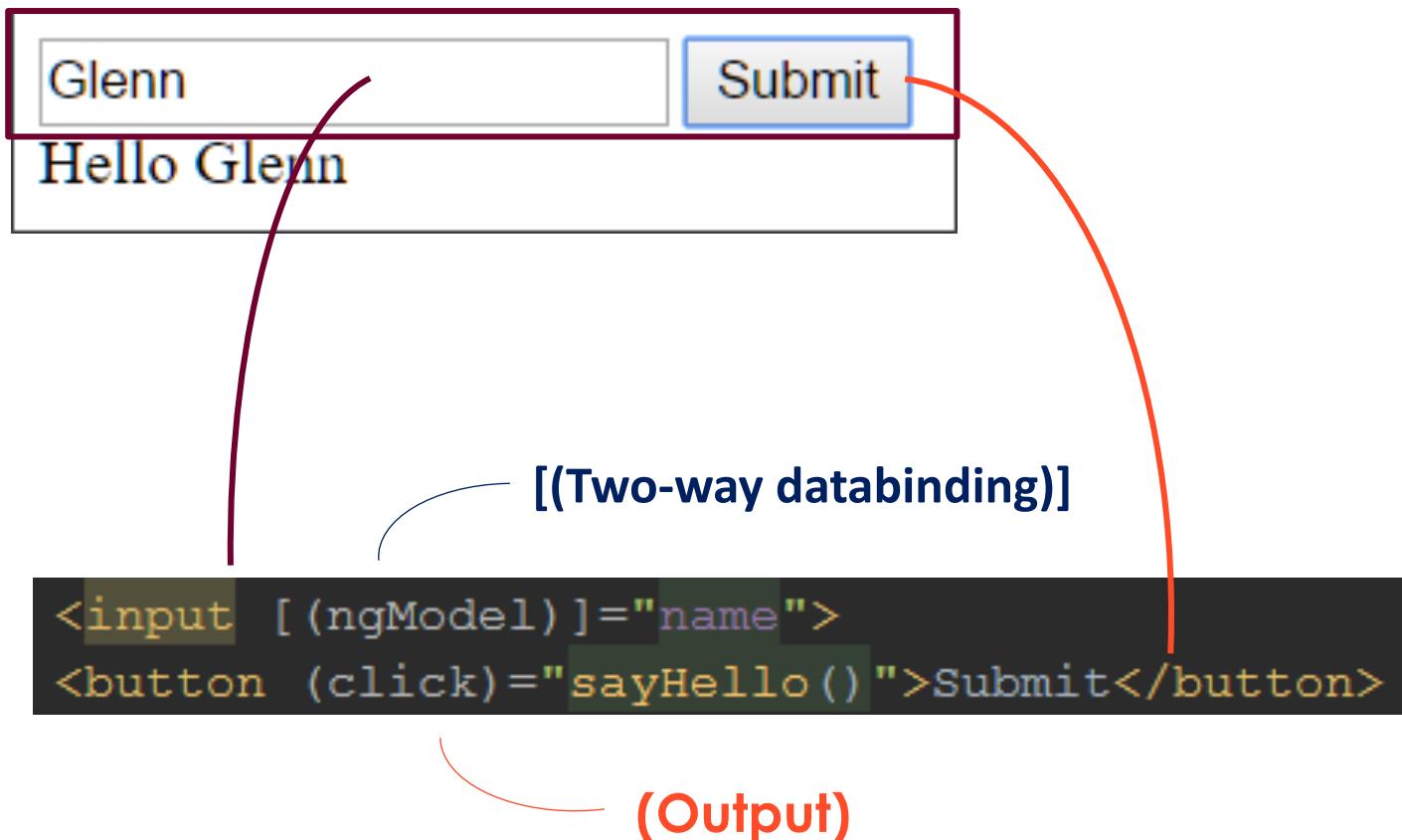
  ngOnInit() {
  }

  sayHello(): void {
    this.nameEntered.emit(`Hello ${this.name}`);
  }
}
```

Bindings

Output logic

hello.component.html



app.component.html

The diagram illustrates the data flow in the component. A red box encloses a form with a text input containing 'Glenn' and a 'Submit' button. An arrow points from the text input to the code block below, labeled '[Input]'. Another arrow points from the 'Submit' button to the same code block, labeled '(Output)'. A third arrow originates from the code block and points to the output message 'Hello Glenn' displayed below the form.

Glenn

Submit

Hello Glenn

```
<ae-hello [name]="name" (nameEntered)="showHelloMessage($event)"></ae-hello>
```

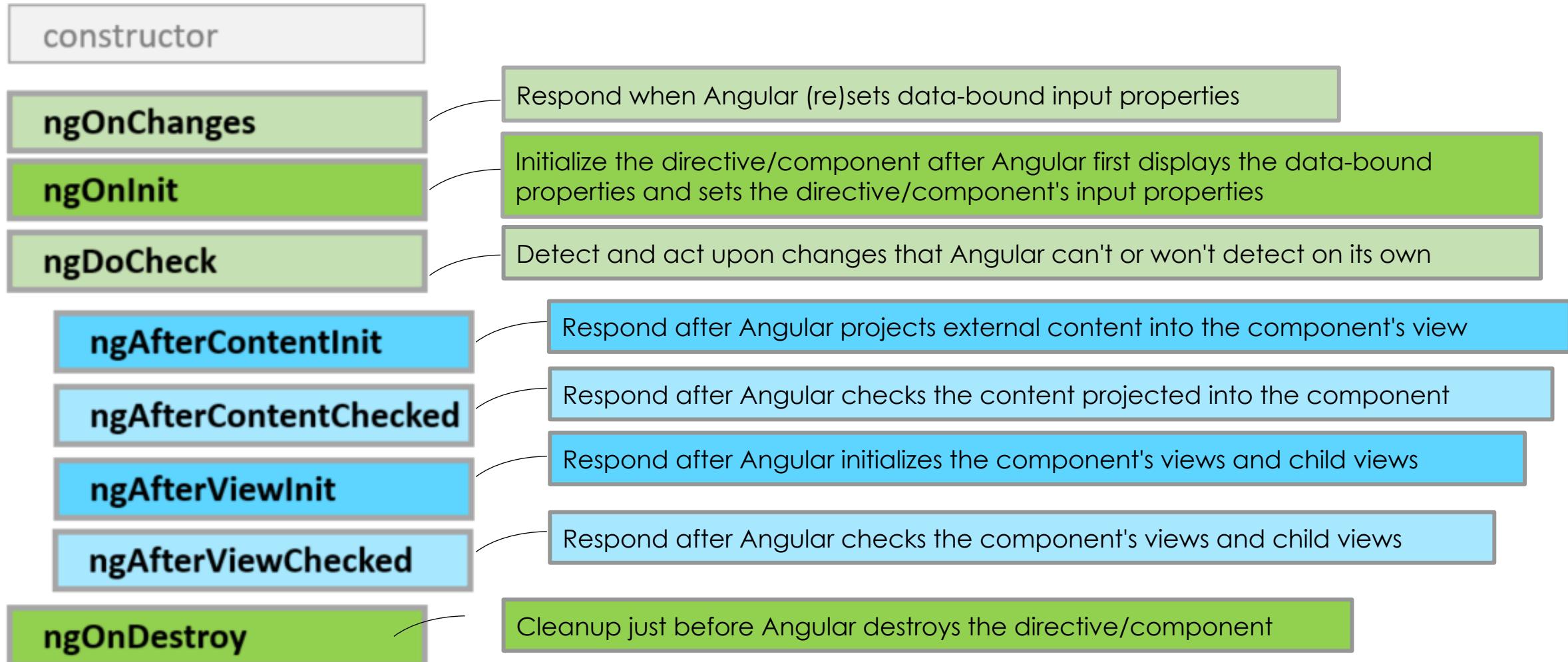
[Input]

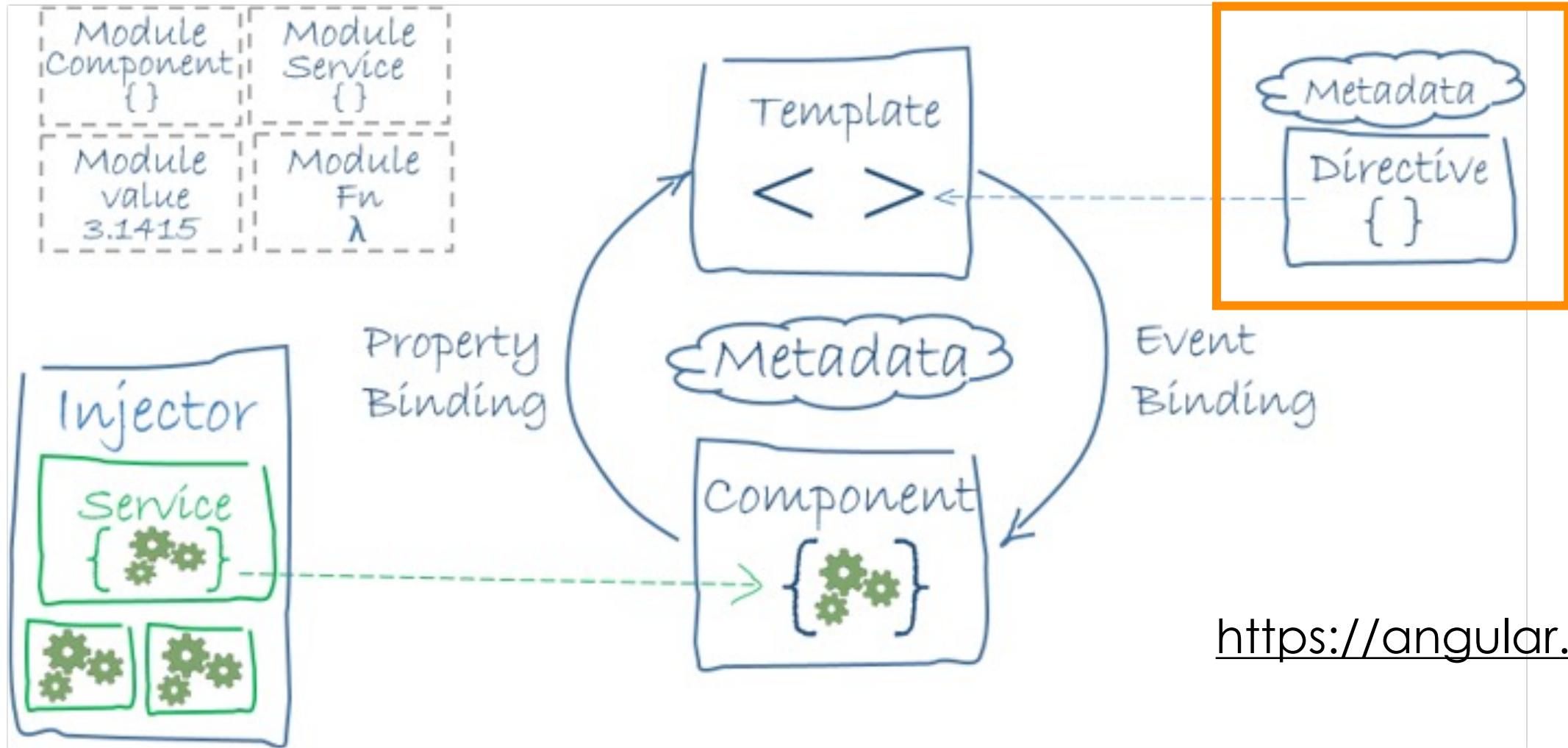
(Output)

Gets value from event
and assigns it to property
'helloMessage'

```
<div>{{helloMessage}}</div>
```

Lifecycle hooks





Directives

@Directive

- Used for DOM manipulation
- Structural directives
 - Defined by Angular
 - Start with *ng (e.g. *ngFor, *ngIf,...)
 - Change the structure of the DOM
- Attribute directives
 - Look like regular HTML directives
 - Alter appearance or behaviour from existing directive

```
ng generate directive my-directive
```

```
ng g d my-directive
```

Pipes

@Pipe

- Used to define display value transformations
- Executed when a value is displayed

```
<!-- Default format: output 'Jun 15, 2015'-->
<p>Today is {{today | date}}</p>

<!-- fullDate format: output 'Monday, June 15, 2015'-->
<p>The date is {{today | date:'fullDate'}}</p>

<!-- shortTime format: output '9:43 AM'-->
<p>The time is {{today | date:'shortTime'}}</p>
```

```
ng generate pipe my-pipe
```

```
ng g p my-pipe
```

Exercise

In case you did not finish the previous one in time:

```
git checkout module-setup
```

- Add a component named `all-pokemon` to the `all-pokemon` module
- Add `<pokedex-all-pokemon>` to `app.component.html` and run your app
- Don't forget to include the correct module in the `AppModule` and to export your new component in the correct way!

EXTRA TIP: Use `--dry-run` to see preview effects

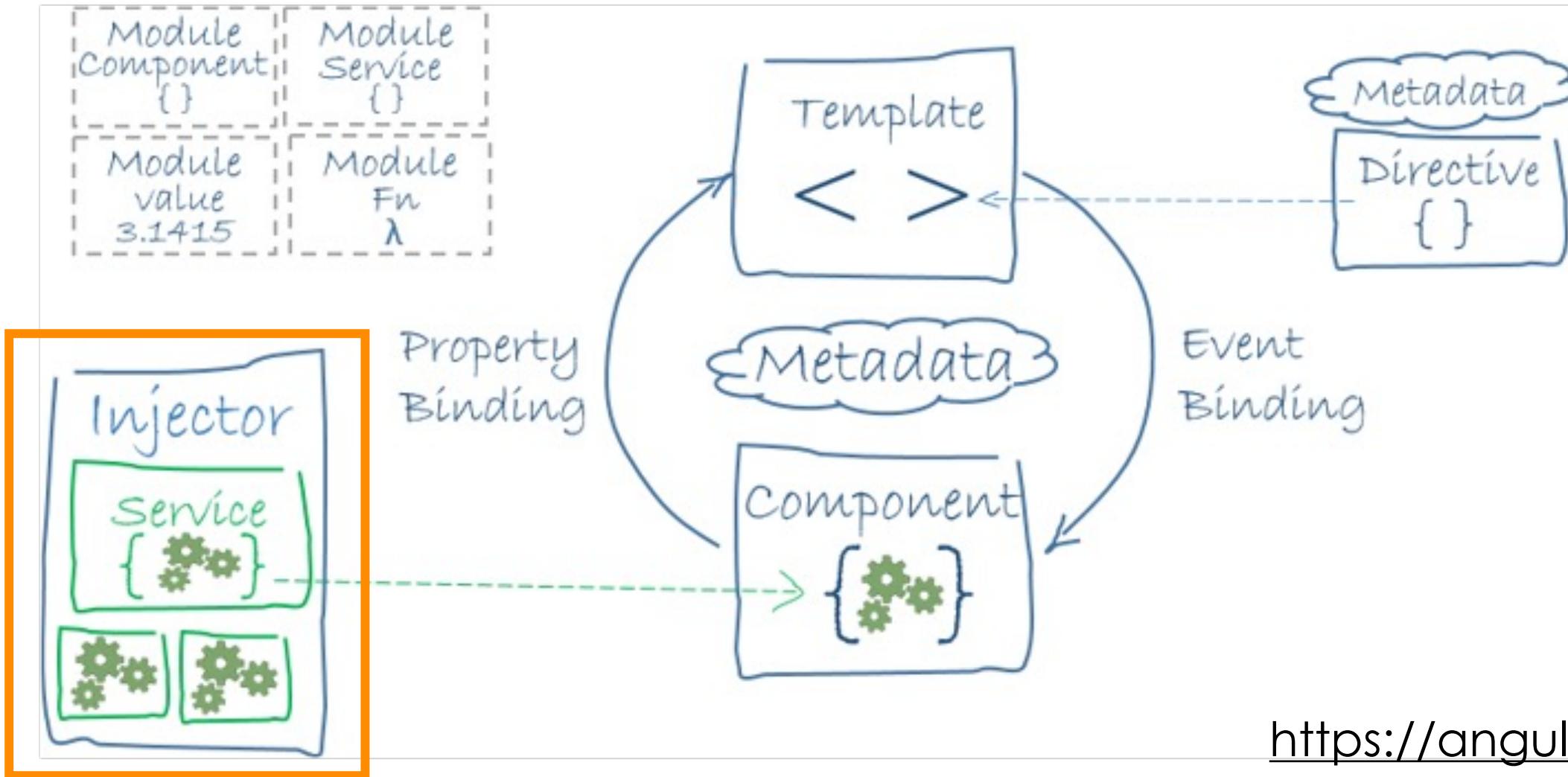
- Add a component `pokemon-list` to the `shared` module
 - Template contains a table with 2 columns
 - 1st column header: `Name`
 - 2nd column header: `Type`
- Use the component `pokemon-list` in the `all-pokemon` component

EXTRA TIP: Check out Emmet Coding (<https://docs.emmet.io/>)

- Create an interface **Pokemon** containing the model (in the shared module)
- Add input binding to **pokemon-list** component
 - `@Input() pokemon`
- Pass array of dummy data to the **pokemon-list** component
 - Declare it in the **all-pokemon** component
 - Pokémon's plural form is also Pokémon 😎
- Display **name** and **type** of every **pokemon** in table of **pokemon-list** component
 - `*ngFor="let pokemon of pokemons"`

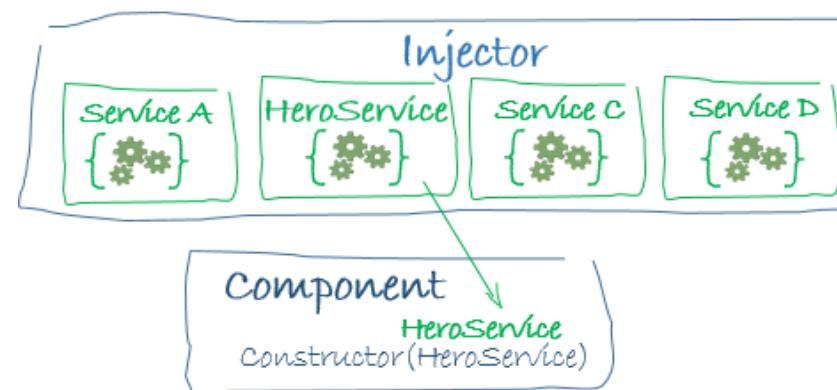
- Add output binding to pokemon-list component
 - `@Output() clicked`
- Add a 3rd column to table in `pokemon-list` component and render a "Catch!" button in it
 - Add some simple table styling
- Call a function `pokemonCaught()` in all-pokemon component if button clicked
 - Eg: print out the pokemon was caught

```
table { border-collapse: collapse; }
table, th, td { border: 1px solid black; }
```



Services @Injectable

- Classes with well defined, shared purpose (e.g. calling an API)
- Logic that is not specific for one view/component
- Dependency injection to provide it to components



Services as Singletons

- Two ways to make sure it is a singleton
 1. Include the service in the AppModule or in a module that is only imported by the AppModule
 2. Declare that the service should only be provided in the app root

```
@Injectable({  
    providedIn: 'root'  
})
```

- Our shared module will not only be imported in the AppModule
=> We declare that service should be provided in the app root

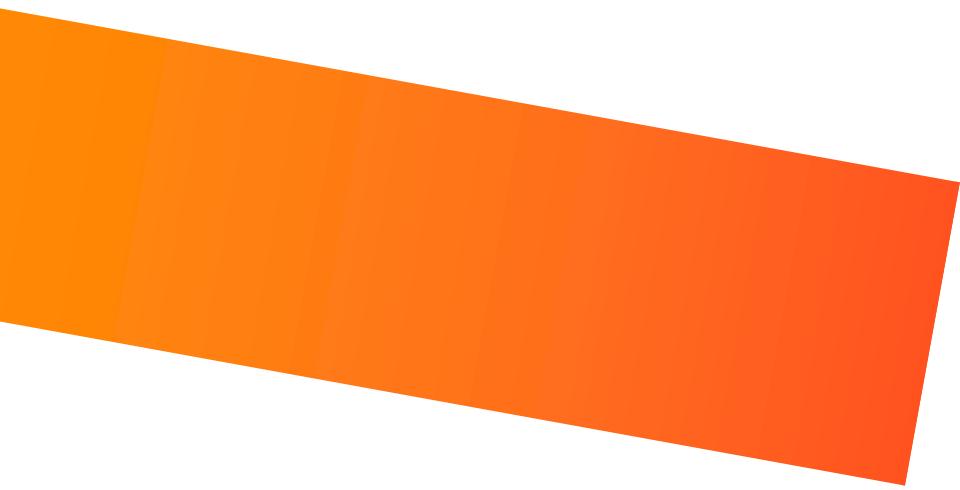
Exercise

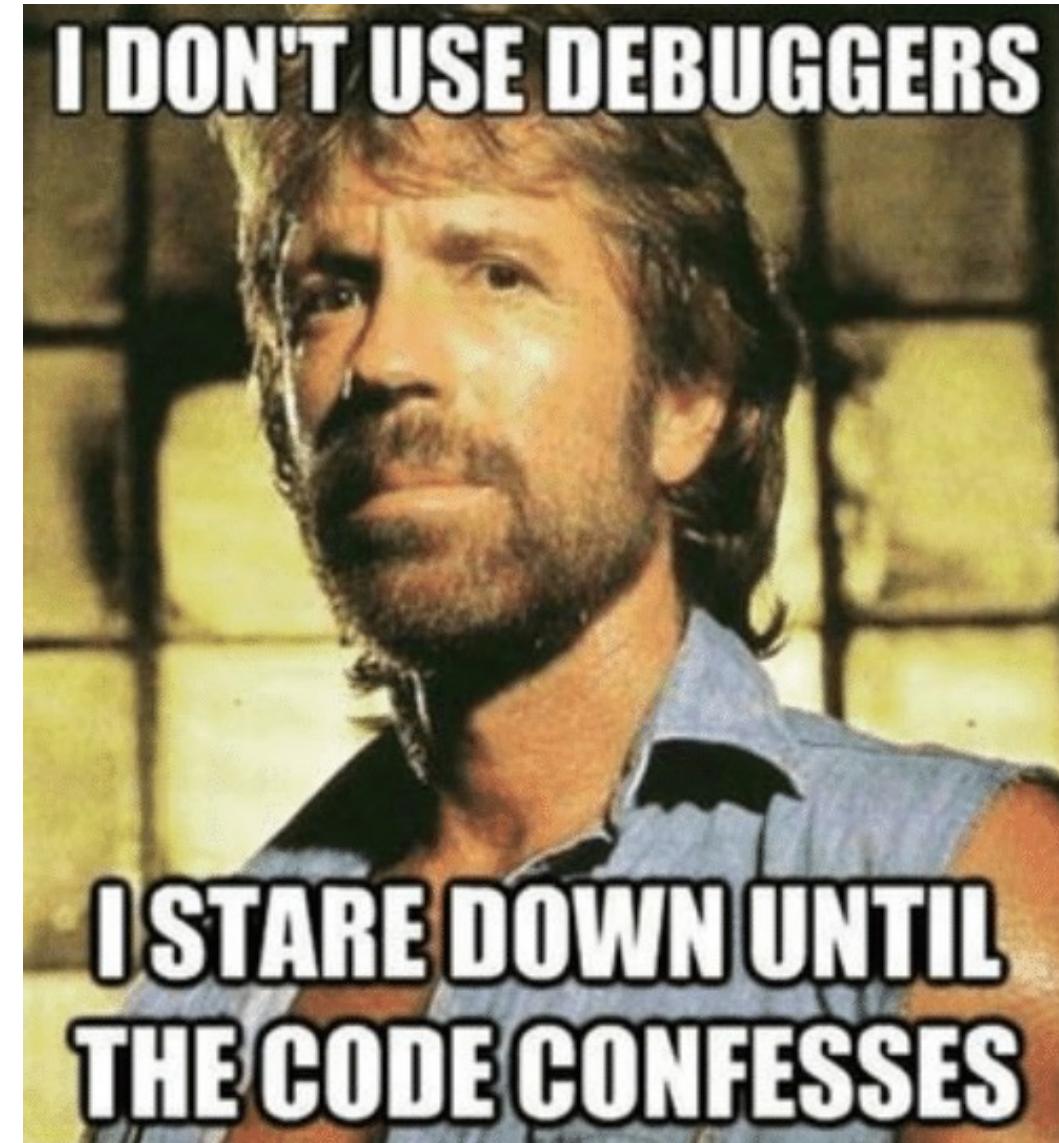
In case you did not finish the previous one in time:

```
git checkout first-components
```

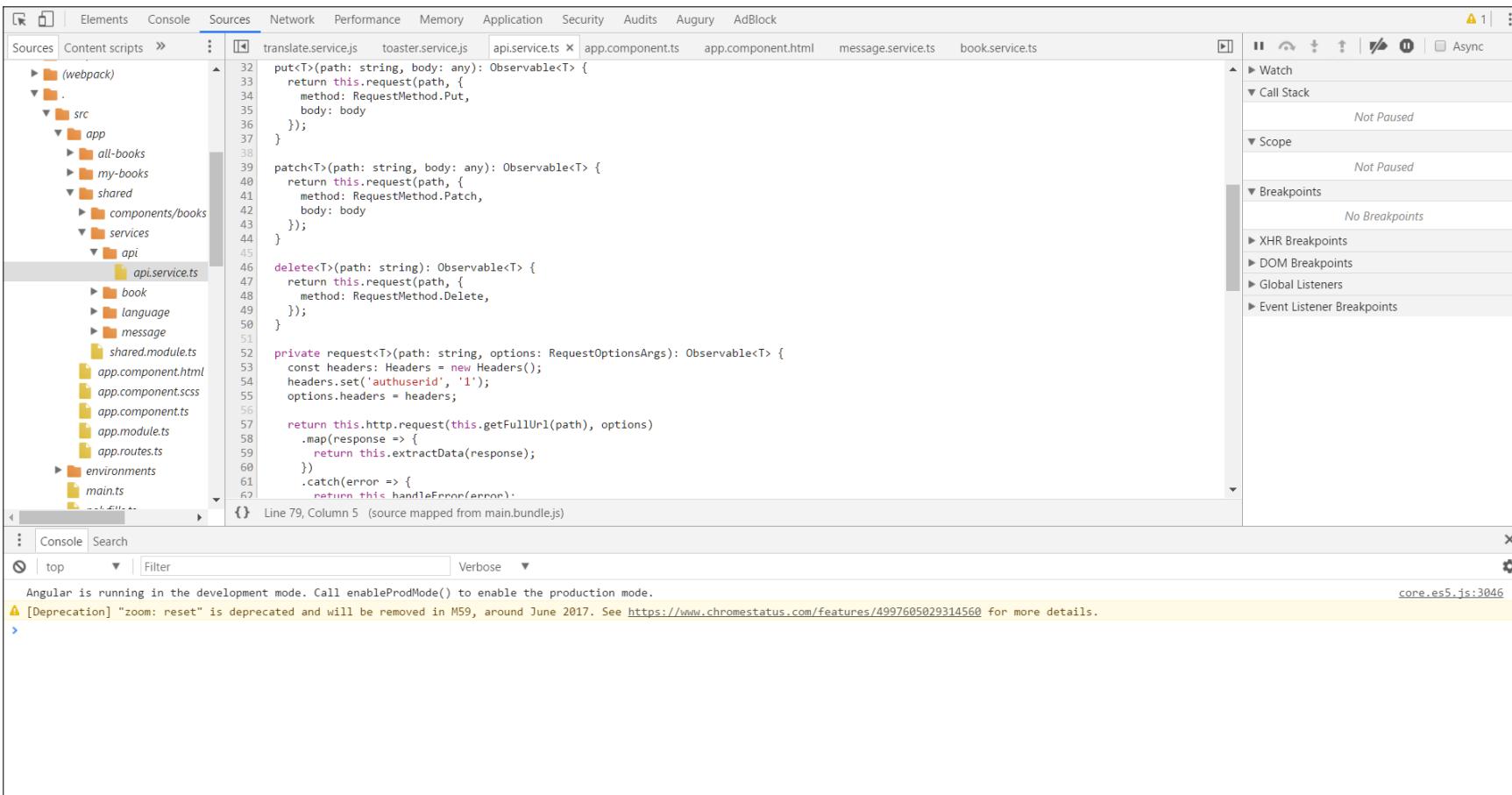
- Add a **pokemon-service** to the shared module
- Instead of implementing the list of dummy pokemon in the all-pokemon component, make it so that this service can be used.

Debugging





Chrome Dev Tools



<https://developer.chrome.com/devtools>

Angular Dev Tools

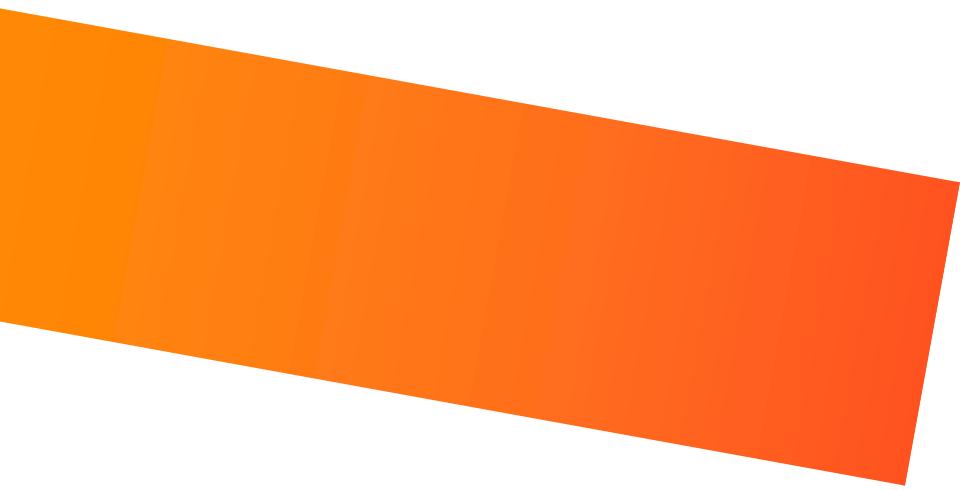
The screenshot shows the Angular DevTools interface with the 'Components' tab selected. In the left sidebar, there is a search bar labeled 'Search components' and a tree view of component hierarchies. One node, 'mat-toolbar == \$ng0', is highlighted with a blue selection bar. The right panel displays detailed information about this component. At the top, it says 'Angular version: 12.0.0-next.9 | DevTools SHA: ce5b7cb'. Below this, the component name 'mat-toolbar' is shown, followed by a copy/paste icon. The properties section is expanded, showing the following properties:

- _color: primary
- _document: HTMLDocument
- _elementRef : {...}
- _platform : {...}
- _toolbarRows : {...}
- defaultColor: undefined

If all else fails...



User Interfaces aka making it look nice





No fear, help is here!

- CSS preprocessors (see yesterday)
- Nice new CSS keywords
 - Flexbox (see yesterday)
 - CSS grid (see yesterday)
- Design frameworks
 - Most popular: Bootstrap
- Component libraries
- Web fonts

CSS Preprocessors

- Built-In in CLI: support for all major preprocessors
 - sass/scss (<http://sass-lang.com/>)
 - less (<http://lesscss.org/>)
 - stylus (<http://stylus-lang.com/>)
- Configure which one...
 - ...when app is generated
 - ...in angular.json

Component libraries

- Material Design (<https://material.angular.io>)
- NGX Bootstrap (<https://github.com/valor-software/ngx-bootstrap>)
- Prime NG (<https://www.primefaces.org/primeng/>)
- NG Bootstrap (<https://ng-bootstrap.github.io/#/home>)
- Onsen UI (<https://onsen.io/angular2/>)
- Vaadin Components (<https://vaadin.com/docs/v7/elements/angular2-polymer/overview.html>)
- NG-ZORRO (<https://github.com/NG-ZORRO/ng-zorro-antd>)
- NG Lightning (<https://github.com/ng-lightning/ng-lightning>)
- NG Semantic-UI (<https://github.com/vladotesanovic/ngSemantic>)
- Clarity (<https://github.com/vmware/clarity>)
- NG2 Charts (<https://valor-software.com/ng2-charts>)
-

Icon libraries



Used to be the silver bullet but there are many vector libraries out there. Choose wisely.

Exercise

In case you did not finish the previous one in time:

```
git checkout first-service
```

- Install Bootstrap 5

```
npm i bootstrap
```

Exercise

- Import files from node_modules in your `styles.scss`

```
@import "../node_modules/bootstrap/scss/bootstrap";  
  
@import "../node_modules/bootstrap/scss/functions";  
@import "../node_modules/bootstrap/scss/variables";  
@import "../node_modules/bootstrap/scss/mixins";
```

- Also update `angular.json`

```
"styles": [  
    "node_modules/bootstrap/dist/css/bootstrap.min.css", // <--add this line  
    "src/styles.scss"  
,  
  "scripts": [  
    "node_modules/bootstrap/dist/js/bootstrap.bundle.min.js" // <--add this line  
]
```

- Use the bootstrap **navbar** component
- Style the buttons and table by adding bootstrap specific classes

BONUS!

- Check out the [Angular Material UI component library!](#)
- Check out <https://fonts.google.com> and change your font
- Create your own pokédex theme using Bootstrap SASS. Check out [Sass](#) .
[Bootstrap v5.0 \(getbootstrap.com\)](#)

Reactive programming





We're still using good old imperative-style programming to deal with problems that are essentially asynchronous



- Sergi Mansilla

Callbacks

book.service.ts

```
getAllBooks(successCallback: Function, errorCallback: Function): void {
  try {
    let books = this.apiService.get(`/books`);
    successCallback(books);
  } catch(error) {
    errorCallback(error);
  }
}
```

books.component.ts

```
this.bookService.getAllBooks((books) => {
  this.books = books;
}, (error) => {
  this.logService.error(error);
});
```

Promises

book.service.ts

```
getAllBooks(): Q.Promise<Array<IBook>> {
  return this.apiService.get(`/books`);
}
```

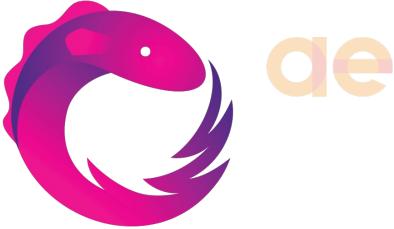
books.component.ts

```
this.bookService.getAllBooks.then( (books: Array<IBook>) => {
  this.books = books;
}.catch( (error) => {
  this.logService.error(error);
}) ;
```

Reactive programming

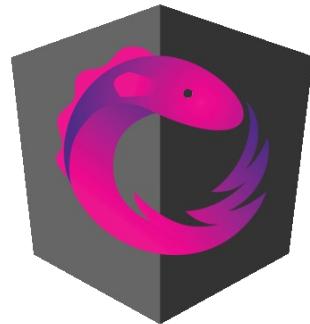
- A programming paradigm that encompasses many concepts and techniques
- With these techniques you can create, transform and react to streams of data

ReactiveX



- An API for async programming with observable streams
- A combination of the best ideas from the **Observer** pattern, the **Iterator** pattern, and **functional programming**
- **RxJS**: A JavaScript implementation of Reactive Extensions
- **NGRX**: Reactive extensions for Angular
- Good dev support available
 - rxjs.dev
 - learnrxjs.io
 - rxmarbles.com

Angular Embraces RXJS in libs



de

- **Databinding**
- **HTTP**
 - HTTP calls return observables. Can be used instead of promises
- **Async pipe**
 - Subscribe to streams in the DOM by using the `async` pipe

Observables

book.service.ts

```
getAllBooks(): Observable<Array<IBook>> {
  return this.apiService.get(`/books`);
}
```

books.component.ts

```
this.books = this.bookService.getAllBooks()
  .catch((error, observable) => {
    this.logService.error(error);
    return observable;
  });

```

books.component.html

```
<app-books [books]="books | async"></app-books>
```

Observables

books.component.ts

```
this.bookService.getAllBooks()
  .subscribe(books => {
    this.books = books;
  })
  .catch((error, observable) => {
    this.logService.error(error);
    return observable;
});
```

books.component.html

```
<app-books [books]="books"></app-books>
```

Intermezzo

Component lifecycle hooks

de

Tapping into the hooks

```
export class AllPokemonComponent implements OnInit {  
  allPokemon$: Observable<Pokemon[]>;  
  
  constructor(private pokemonService: PokemonService) { }  
  
  ngOnInit(): void {  
    this.allPokemon$ = this.pokemonService.getAllPokemon();  
  }  
  
  catchPokemon(pokemon: Pokemon) {  
    this.pokemonService.catchPokemon(pokemon.id).subscribe();  
  }  
}
```

Operators



```
filter(x => x > 10)
```



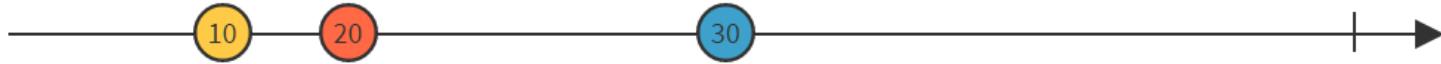
```
take(2)
```



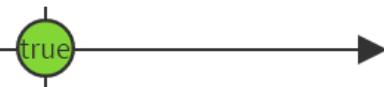
Operators



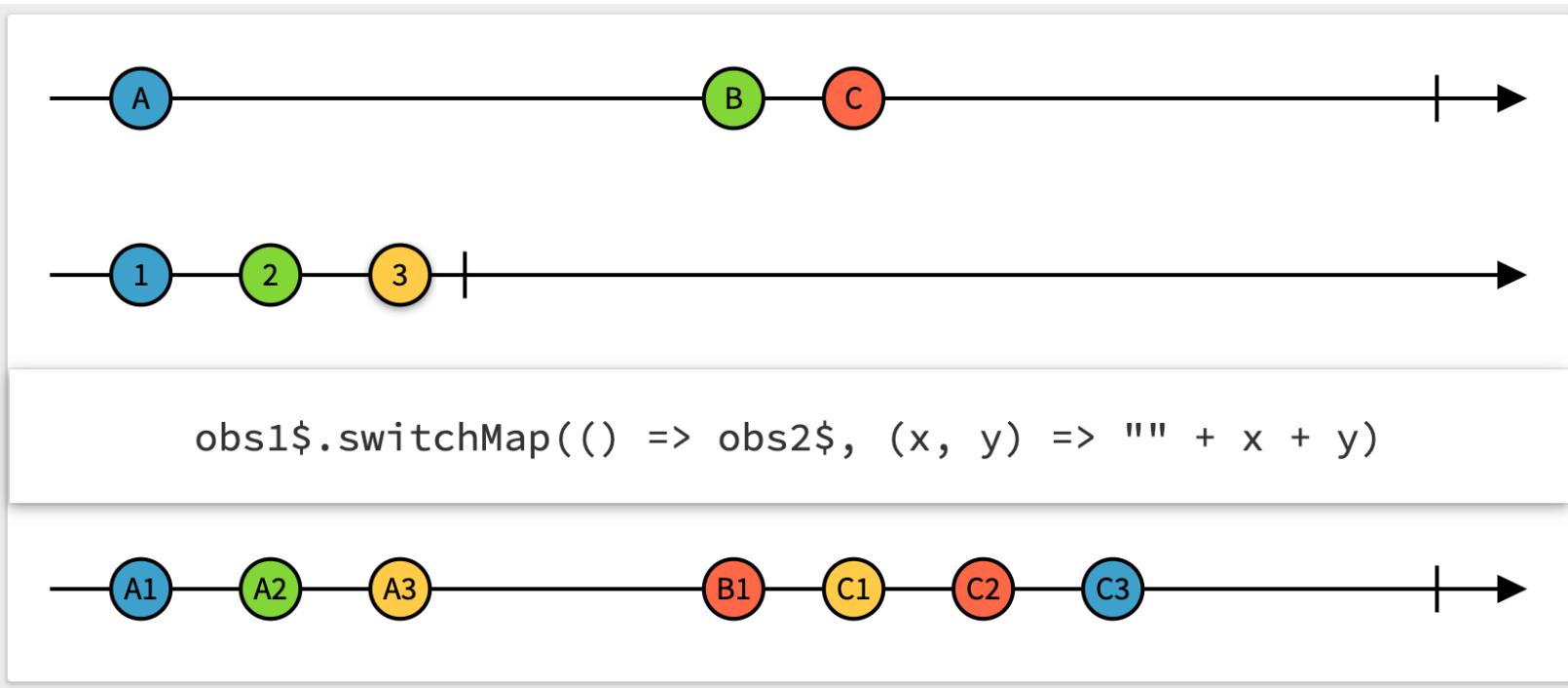
```
map(x => 10 * x)
```



```
every(x => x < 10)
```



Operators



RxMarbles

Interactive diagrams of Rx Observables

Fork me on GitHub

TRANSFORMING OPERATORS

- [delay](#)
- [delayWithSelector](#)
- [findIndex](#)
- [map](#)
- [scan](#)
- [debounce](#)
- [debounceWithSelector](#)

COMBINING OPERATORS

- [combineLatest](#)
- [concat](#)
- [merge](#)
- [sample](#)
- [startWith](#)
- [withLatestFrom](#)
- [zip](#)

FILTERING OPERATORS

- [distinct](#)
- [distinctUntilChanged](#)
- [elementAt](#)
- [filter](#)
- [find](#)
- [first](#)
- [last](#)

merge

v1.4.1 built on RxJS v2.5.3 by [@andrestaltz](#)

<http://rxmarbles.com/>

Signals

- Wrapper around a value
- Notify interested consumers when value changes
- Read with getter => tracking where it is used
- Writable or read-only

Signals

```
const count = signal(0);

// Signals are getter functions – calling them reads their value.
console.log('The count is: ' + count());

// Change a signal value
count.set(3);

// When the new value depends on the current value use update
count.update(value => value + 1)

// When handling arrays or complex objects
const currencies = signal([
  {currency: 'USD', rate: 1},
  {currency: 'EUR', rate: 0.88}
])

currencies.mutate(value => value[0].rate = 1.1)
```

Exercise

In case you did not finish the previous one in time:

```
git checkout bootstrap
```

In this exercise we will start using a Pokémon API

You can find this API in the **back-end** folder

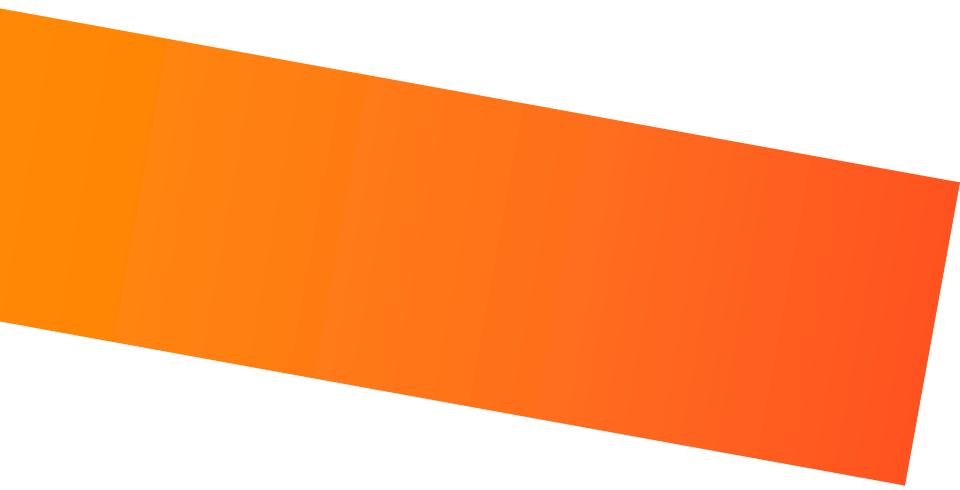
Install the packages and run it to **serve locally**

- Make the front end consume the API to show all Pokémon
- Update the pokemon.model.ts file to contain the complete model based on the server responses (hint: checkout [JSON to TypeScript](#))
- Replace the previously created method in pokemon.service.ts with an HTTP call.
 - Import an Angular module in the AppModule

EXTRA EXTRA!

- Extract the URL from the `pokemon.service.ts` file to the environment variables. (`ng generate environments`)
- The assets folder contains `sprites` for each pokemon. Add them to your table based on their ID.
- Create a `MessageService`. The use of this service is that when you click 'catch' a message is displayed if the action was successful.
 - Look at `ngx-toastr` to display these messages
- Add `error handling` to your calls to the `PokemonService` to display an error message when something goes wrong. (You can test your error message by shutting down the server!)

Angular Router



Angular router

- Enables **navigation from one view to another** as users perform application tasks
- During navigation, URL changes
- Navigation can be triggered by changing URL
- Packaged in module **@angular/router**

Trigger route changes

- Do not use href
- Use either
 - routerLink in templates (HTML)
 - routerLinkActive to add CSS classes if routerLink on element is active

```
<a class="nav-link" routerLinkActive="active" routerLink="/all-pokemon">  
  My Pokémon  
</a>
```

- Router in components (TypeScript)

```
constructor(private router: Router) { }  
  
this.router.navigate(['/team'], {  
  queryParams: { userId: this.userId, userName: this.userName }  
});
```

Angular router - setup

- **AppRoutingModule** is imported into root AppModule

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

app-routing.module.ts

```
const routes: Routes = [
{ path: '', redirectTo: '/dashboard', pathMatch: 'full' },
{ path: 'dashboard', component: DashboardComponent },
{ path: 'detail/:id', component: HeroDetailComponent },
{ path: 'heroes', component: HeroesComponent }
];

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

Angular router

- Better approach: configure specific **routing in feature module** itself
 - Feature module becomes **lazy-loaded**
 - Feature module is fully **self-contained**
 - Components
 - Services
 - Routing

all-pokemon.module.ts

- **FeatureRoutingModule** is imported into featureModule

```
@NgModule({  
  declarations: [  
    AllPokemonComponent  
,  
  imports: [  
    CommonModule,  
    AllPokemonRoutingModule  
,  
  exports: [  
    AllPokemonComponent  
,  
  ]  
})  
export class AllPokemonModule { }
```

all-pokemon-routing.module.ts

```
const routes: Routes = [
  { path: '', component: AllPokemonComponent }
];

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

Lazy loading feauture modules

```
const routes: Routes = [
  {
    path: 'all-pokemon',
    loadChildren: () => import('./all-pokemon/all-pokemon.module')
      .then(m => m.AllPokemonModule)
  }
];
```

Extra Remarks

- Configuration of routes on first-match principle
- The ‘**’ – path is a wildcard for all pages, useful for 404 pages
- The output of the configured path is rendered as follows:

```
<router-outlet></router-outlet>
<!-- Routed views go here -->
```

Exercise

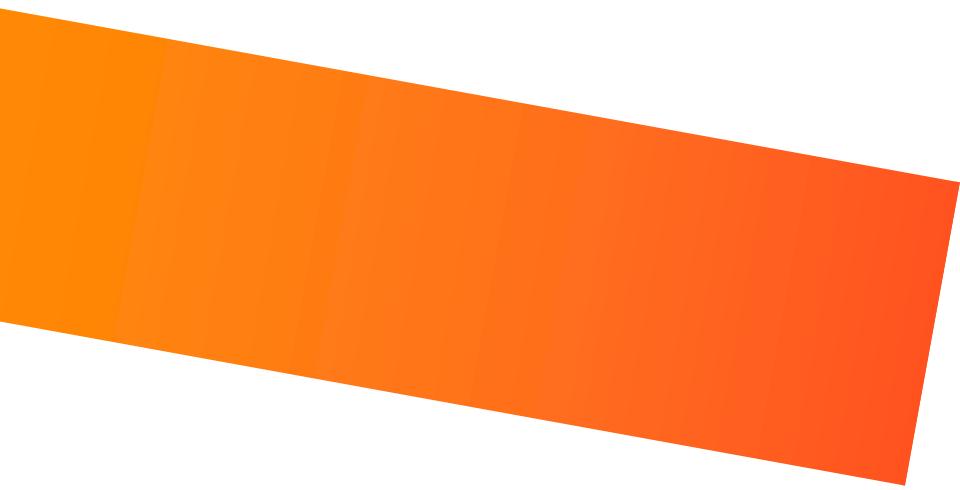
In case you did not finish the previous one in time:

```
git checkout calling-apis
```

- Add a component named `my-pokemon`
- Use the component `pokemon-list` in the `my-pokemon` component
- Implement the `PokemonService.getMyPokemon` method to fill the component
- Add an `@Input()` `actionLabel` to `pokemon-list` component
 - Pass a value to it in `all-pokemon` and `my-pokemon`
 - Update button rendering in `pokemon-list` using this label
- Use the API to catch & release Pokémons

- Set up routing
 - Add a route `all-pokemon`
 - Add a route `my-pokemon`
 - Set `all-pokemon` as default route
- Remove `<pokedex-all-pokemon>` from `app` component
- Update `items` in `navbar` to navigate to other route

QA





QA default in CLI



Anatomy of a Jasmine spec

```
describe("A spec (with setup and tear-down)", () => {
  let foo;

  beforeEach(()=> {
    foo = 0;
    foo += 1;
  });

  afterEach(()=> {
    foo = 0;
  });

  it("can have more than one expectation", ()=> {
    expect(foo).toEqual(1);
    expect(true).toEqual(true);
  });
});
```

A test suite

Setup

Teardown

Test logic (aka spec)

Expectation

Anatomy of an Angular component spec

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';

import { MyComponent } from './my.component';

describe('MyComponent', () => {
  let component: MyComponent;
  let fixture: ComponentFixture<MyComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ MyComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(MyComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should be created', () => {
    expect(component).toBeTruthy();
  });
});
```

Creates an Angular testing module and declares the component to test

Compile template and css async, because they're external

Creates a handle on the test env surrounding the created component

Trigger data binding and propagation

Test that component has been created

Exercise

In case you did not finish the previous one in time:

```
git checkout routing
```

```
npm i
```

```
npm run lint    or    ng lint
```

- Fix linting errors
 - Let your IDE assist you
 - VS Code: Quick fix (or Fix autofixable problems)
 - Webstorm: settings -> tslint, enable and select tslint.json config file

```
npm test
```

or

```
ng test
```

- Test the `showApiError` function in `MessageService`

I18N

How to handle translations

i18n in Angular

Compile-time translation

- Embedded in application bundle
- Improved performance
- Seamless integration
- Must reload for language switch
- More complex for devs

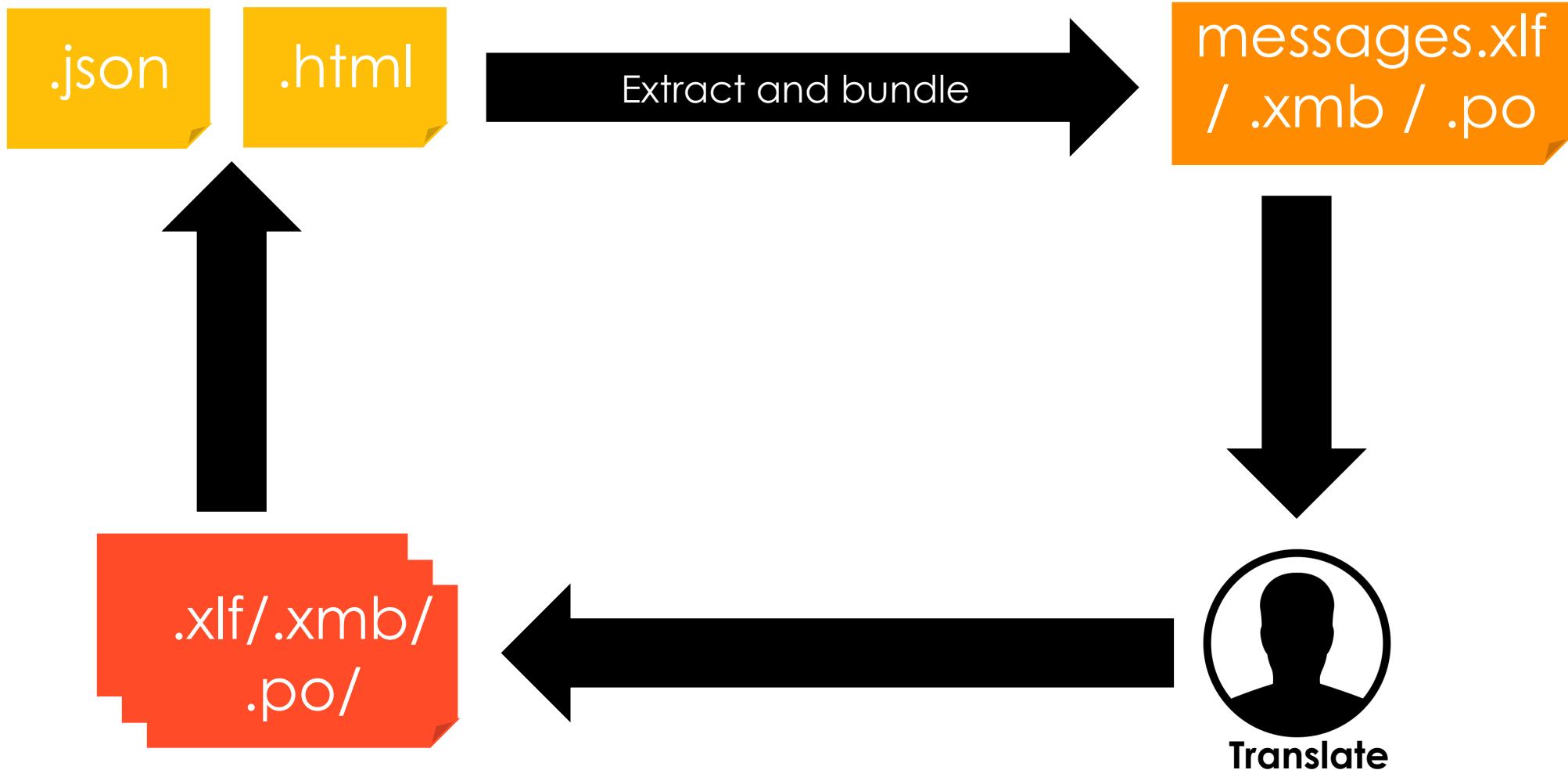
→ Angular i18n

Runtime translation

- Stored in external files (eg JSON)
- Slightly lower performance
- External libraries
- Switch language without page reload
- Easier for devs

○ → ngx-translate

Compile-time process



Angular i18n

- Tooling
 - Message extraction, file transformation, template generation
- Pluralization and gender select
- HTML annotations
 - Context, descriptions and meanings

Angular i18n: mark for translation

```
<h1 i18n="User welcome|An introduction header for this  
sample">Hello i18n!</h1>
```

```
<!--i18n: optional meaning|optional description -->  
I don't output any element either  
<!--/i18n-->
```

```
<img [src]="logo" i18n-title title="Angular logo" />
```

Angular i18n: pluralization & gender select

```
<span i18n>{wolves, plural, =0 {no wolves} =1 {one wolf} =2 {two  
wolves} other {a wolf pack}}</span>
```

```
<span i18n>The hero is {gender, select, m {male} f {female}}</span>
```

Angular i18n: translate

- Copy messages.xlf for every target language
 - Messages.nl.xlf
 - Messages.fr.xlf

```
<trans-unit id="af2ccf4b5dba59616e92cf1531505af02da8f6d2"  
datatype="html">  
  <source>Hello i18n!</source>  
  <target>Hallo i18n!</target>  
  <note priority="1" from="description">An introduction header  
for this sample</note>  
  <note priority="1" from="meaning">User welcome</note>  
</trans-unit>
```

Angular i18n: translate

```
<trans-unit id="6e22e74e8cbd3095560cfe08993c4fdfa3c50eb0" datatype="html">
  <source/>
  <target>{wolves, plural, =0 {geen wolveren} =1 {een wolf} =2 {twee wolveren}}
other {een roedel wolveren}</target>
</trans-unit>
```

```
<trans-unit id="61cafedb85466ab789b3ae817bba1a545468ee1c" datatype="html">
  <source>The hero is <x id="ICU"/></source>
  <target>De held is <x id="ICU"/></target>
</trans-unit>
```

```
<trans-unit id="14c7055d67771a3b7b6888d282ac092896be06b6" datatype="html">
  <source/>
  <target>{gender, select, m {man} f {vrouw}}</target>
</trans-unit>
```

Angular i18n: merge translations into app

- Compile app providing
 - Translation file, translation file format and locale ID (nl or nl-BE)
- JIT: compile in browser while application loads
 - During application bootstrap
 - Reload app after selecting new language
- AOT: compilation is part of build process
 - Separate application package per language is pre-built

Angular i18n

Translation file maintenance and *id* changes

As the application evolves, you will change the *i18n* markup and re-run the `ng-xi18n` extraction tool many times. The new markup that you add is not a problem; but most changes to existing markup trigger generation of new `ids` for the affected translation units.

After an `id` changes, the translation files are no longer in-sync. **All translated versions of the application will fail** during re-compilation. The error messages identify the old `ids` that are no longer valid but they don't tell you what the new `ids` should be.

Commit all translation message files to source control, especially the English source `messages.xlf`. The difference between the old and the new `messages.xlf` file help you find and update `id` changes across your translation files.

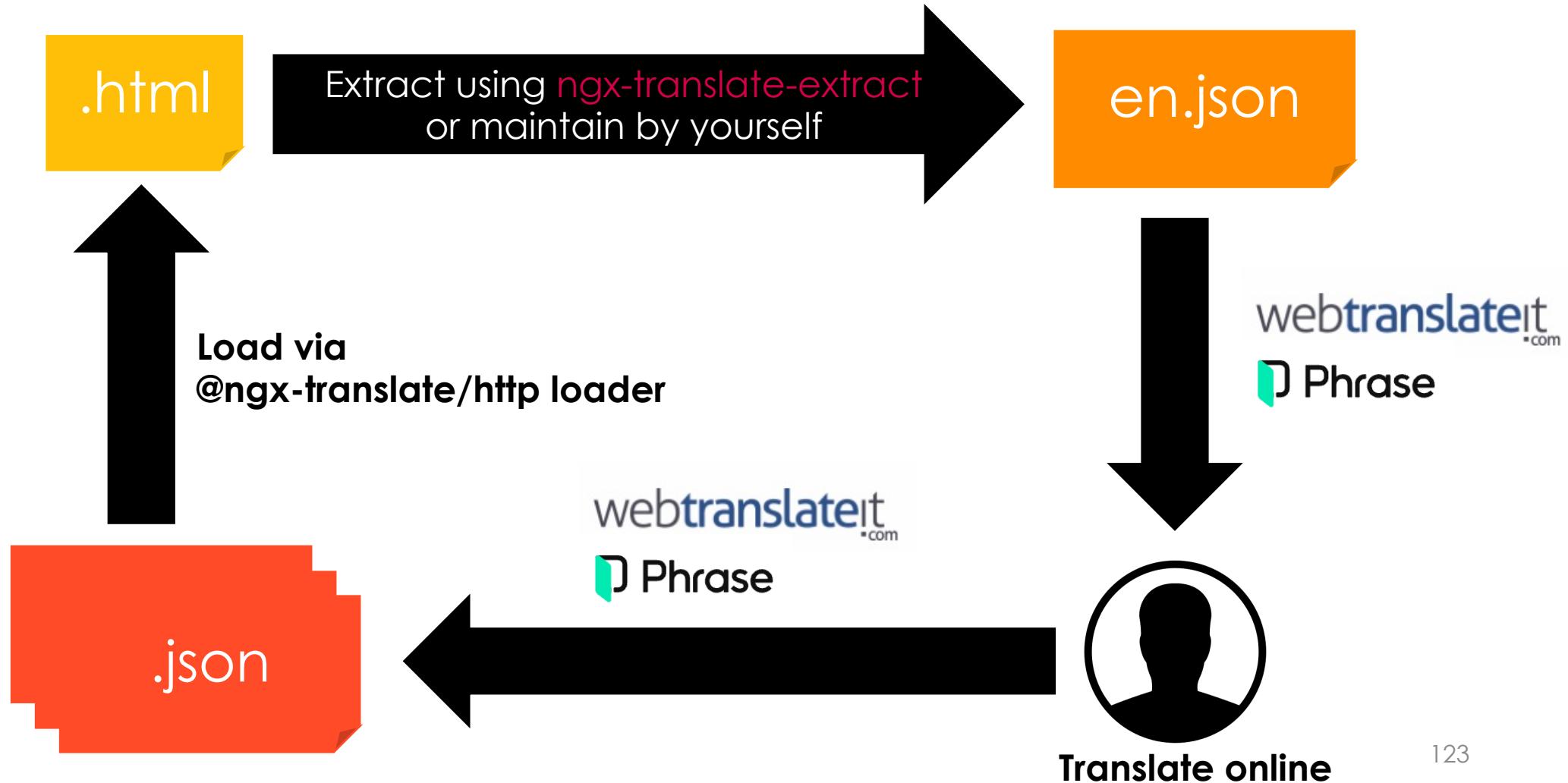


ngx-translate

- Modular i18n library providing service, directive and pipe
- Switching between languages doesn't reload whole app
- By default no process defined
 - But one can set up easily by using
 - Plugins
 - Webtranslateit
 - Or the road in between: ngx-translate-extract



Runtime process



Setup – Import module (root)

```
import {BrowserModule} from '@angular/platform-browser';
import {NgModule} from '@angular/core';
import {TranslateModule} from '@ngx-translate/core';

@NgModule({
  imports: [
    BrowserModule,
    TranslateModule.forRoot()
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Root module config

```
TranslateModule.forRoot({
  loader: {
    provide: TranslateLoader,
    useFactory: HttpLoaderFactory,
    deps: [HttpClient]
  }
})
```

Setup – Import module (shared)

```
@NgModule({  
  exports: [  
    CommonModule,  
    TranslateModule  
  ]  
})  
export class SharedModule { }
```

No need to import in other modules
using SharedModule

Setup – Set default language

```
@NgModule({
  imports: [
    BrowserModule,
    TranslateModule.forRoot({
      defaultLanguage: 'en'
    })
  ],
  providers: [
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Define translations

```
{  
    "HOME": {  
        "HELLO": "hello {{value}}"  
    }  
}
```

Usage with pipes

```
<div>{{ 'HELLO' | translate:param }}</div>
```

Exercise

In case you did not finish the previous one in time:

```
git checkout qa
```

```
npm i @ngx-translate/core @ngx-translate/http-loader
```

```
npm i @vendure/ngx-translate-extract --save-dev
```

- Set up ngx-translate
 - Supported languages: `en` & `nl`
 - Use `HttpLoader`
 - Set `en` as default
- Create a `LanguageService` in shared module
 - It can `get` and `change` language

- Internationalize labels used in UI (use translate pipe and functions)
 - Also @Input actionLabel
- Extract your labels to a json file using ngx-translate-extract
 - First add npm scripts

```
"scripts": {  
  "i18n:init": "ngx-translate-extract --input ./src --output ./src/assets/i18n/template.json --key-as-default-value --replace --format json",  
  "i18n:extract": "ngx-translate-extract --input ./src --output ./src/assets/i18n/{en,da,de,fi,nb,nl,sv}.json --clean --format json"  
}
```

- Copy the en.json file to nl.json and translate it

- Add language button to nav
 - NL: changes language to NL
 - EN: changes language to EN
 - Item is only present if `!== currentLanguage`

Summary

- Angular CLI is a great tool to setup and maintain structure in angular applications
- Architecture is as important in the front-end as it is in the back-end
- Use the devtools to debug your application
- There are a lot of tools out there to easily make beautiful applications
- Reactive programming can simplify complex applications
- Use the built in angular router tools and don't link with href
- There are different I18N tools that make it easy to translate applications

Final questions?

**THANK
YOU**

