Front end engineering for the web - Angular

Crelan – January 2025

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

https://tinyurl.com/fee-2025

IP disclaimer

Angu-what?

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

Angular

https://angular.dev

NOT TO BE CONFUSED WITH ANGULARJS by Google

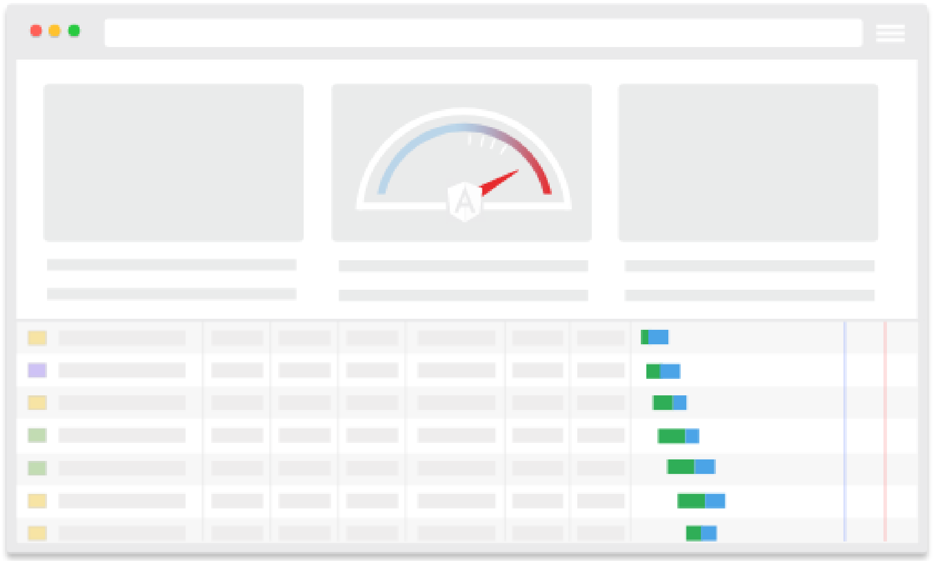
Develop Across All Platforms

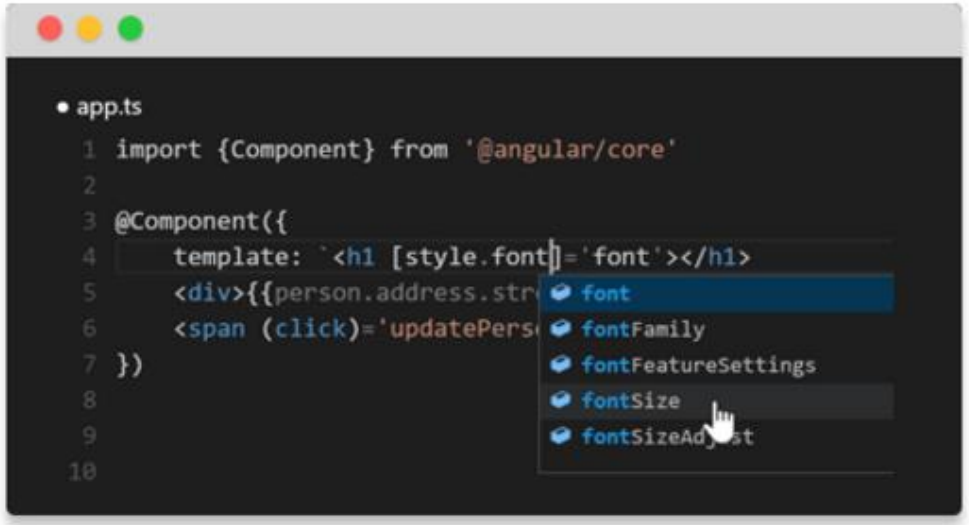
Learn one way to build applications with Angular and reuse your code and abilities to build apps for any deployment target. For web, mobile web, native mobile and native desktop.

Speed & Performance

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

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





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

@Component({
    template: `<h1 [style.font]='font'></h1>
    <div>{{person.address.str
    <span (click)='updatePers
})
```

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?

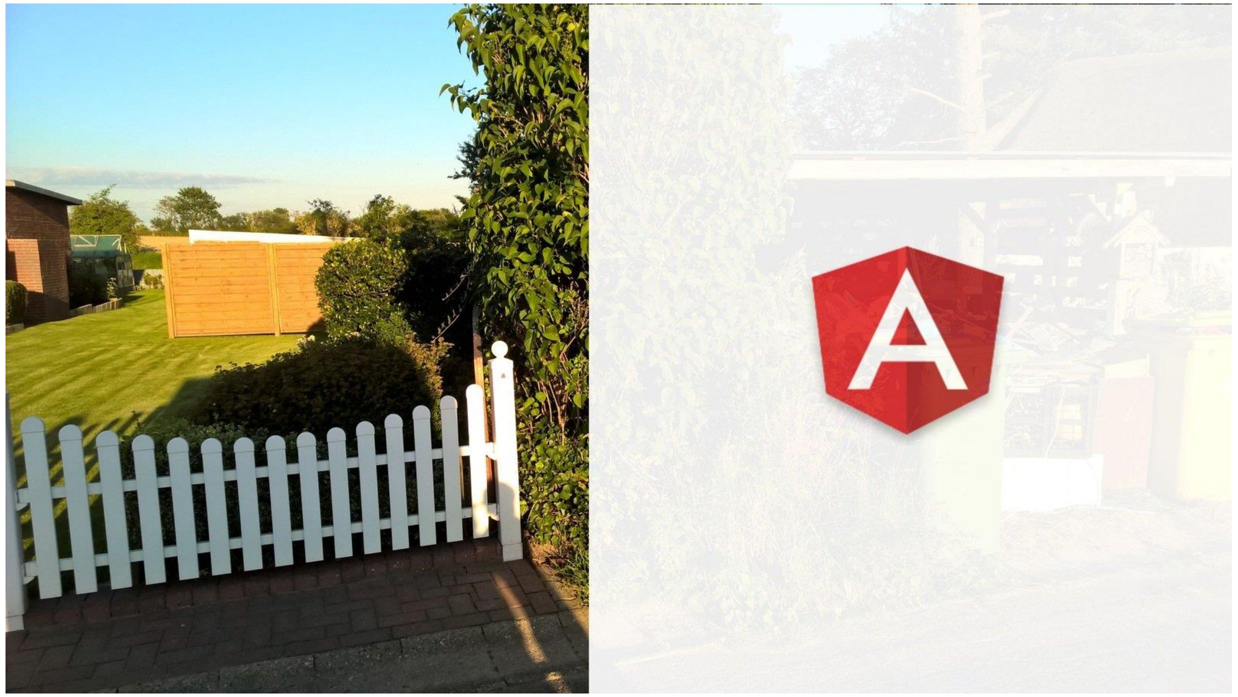
- Available as an npm package:

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

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

```
ng ...
```

Angular CLI – styleguide



https://angular.dev/style-guide

Angular CLI – styleguide

- Use **feature** modules/ folder over technical modules
  - All-books, my-books,…
  - Components, services,…
- Create a **shared module/ folder 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 **uglifies** 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 --lts
```

```
nvm use --lts
```

```
npm i –g @angular/cli
```

Exercise

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

```
cd front-end
```

```
ng serve
```

or

```
npm start
```

Architecture

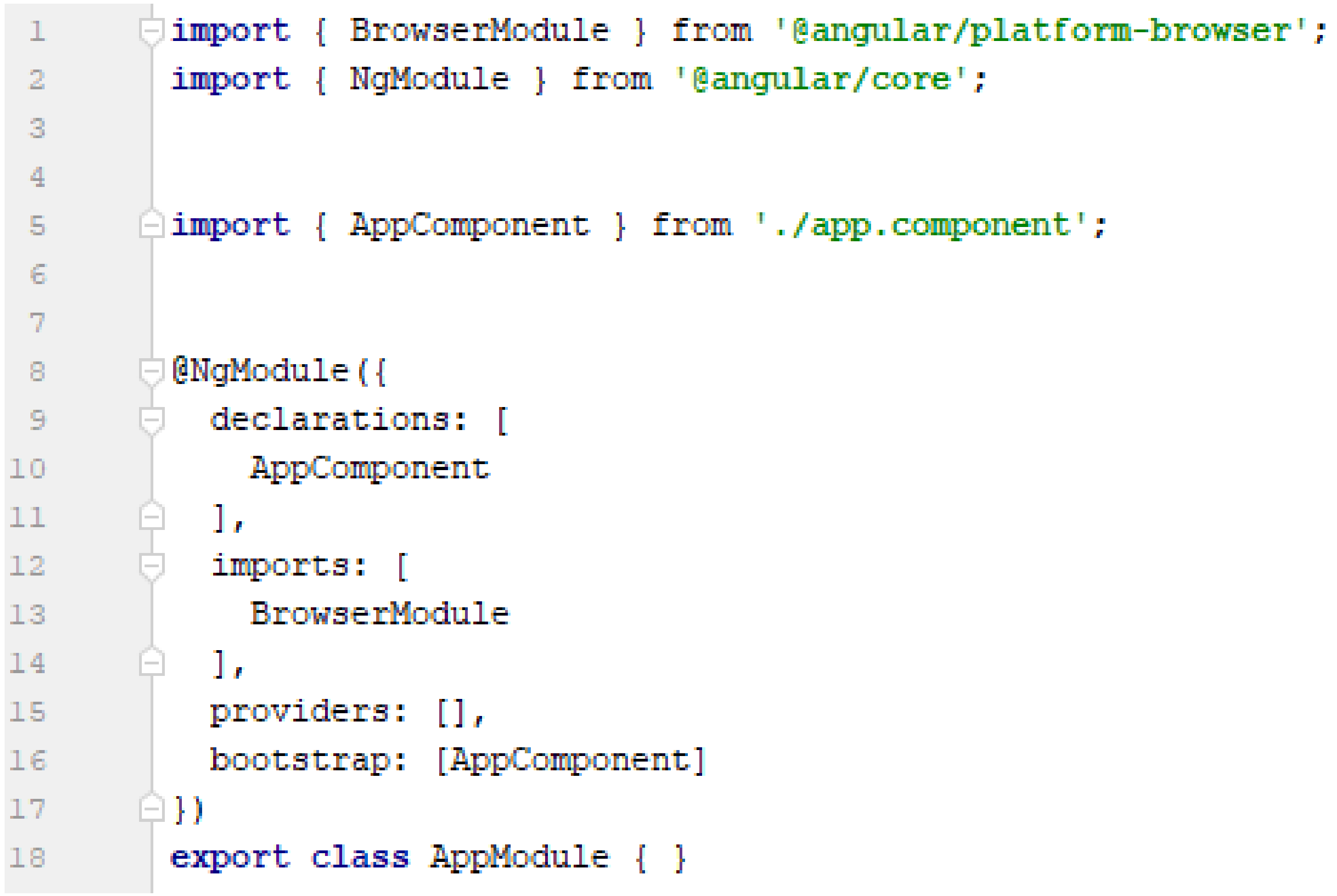
Modules `@NgModule`

- 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



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


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


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

Standalone apps (from v17)

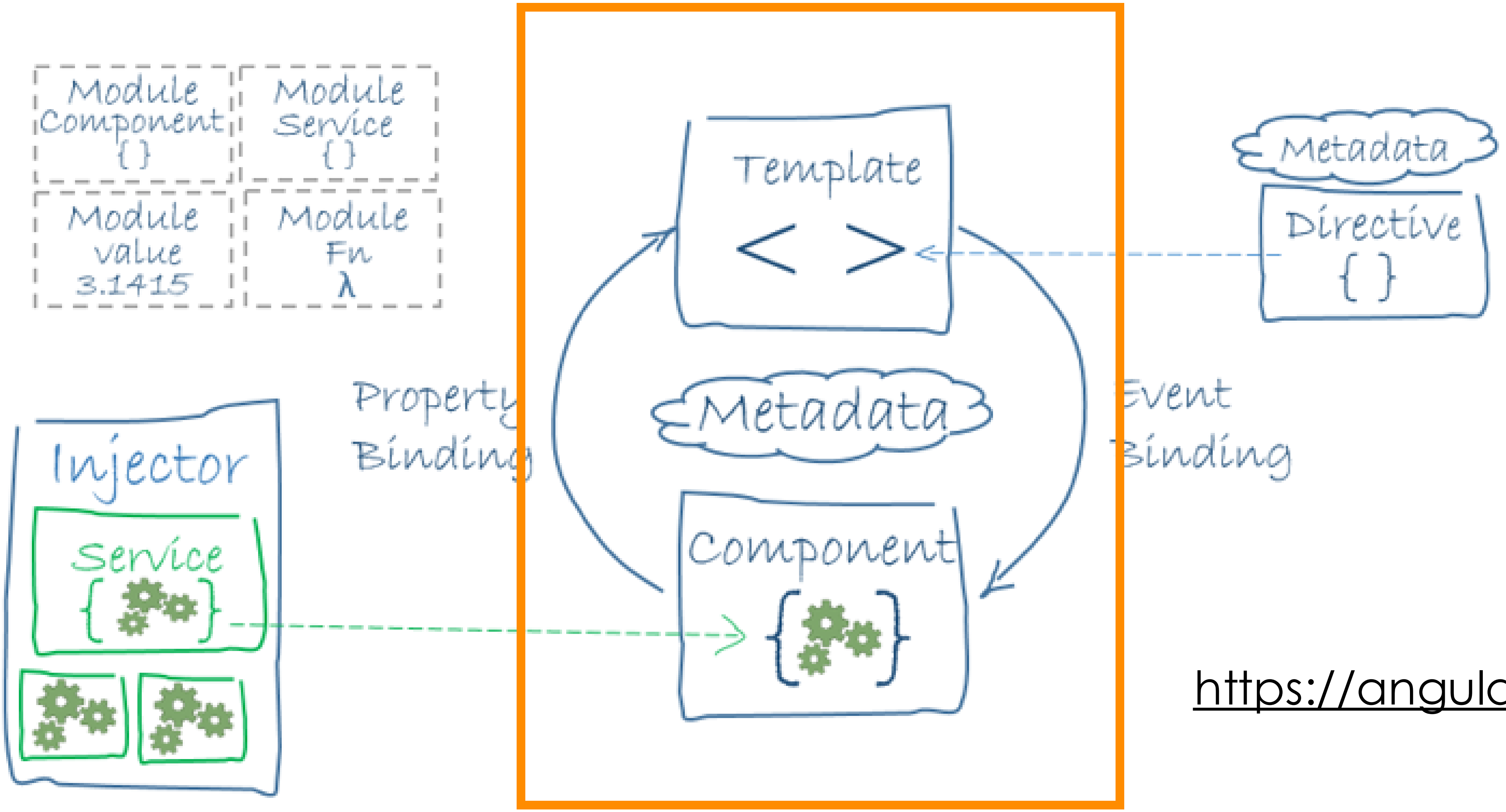
+

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

➔ No shared dependencies or relations with other components

−

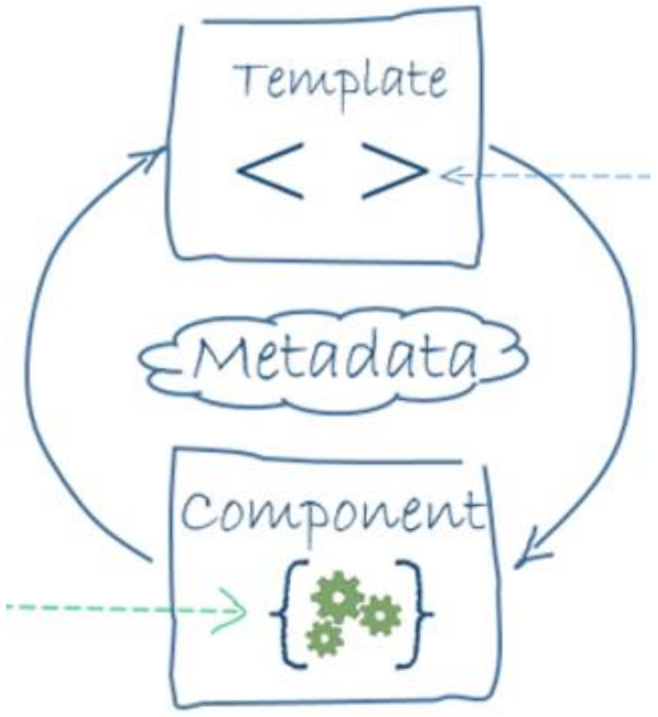
- More complex dependency management



https://angular.dev

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

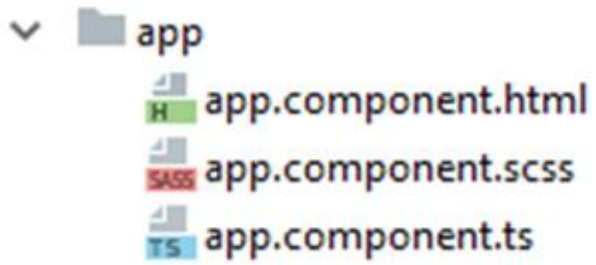
Metadata to describe coupling between these parts

Logic (Typscript class), what does my component DO

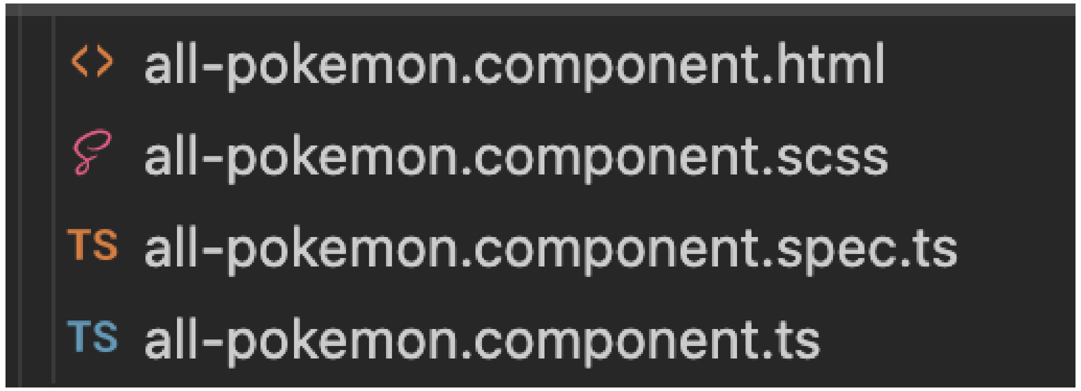
Components

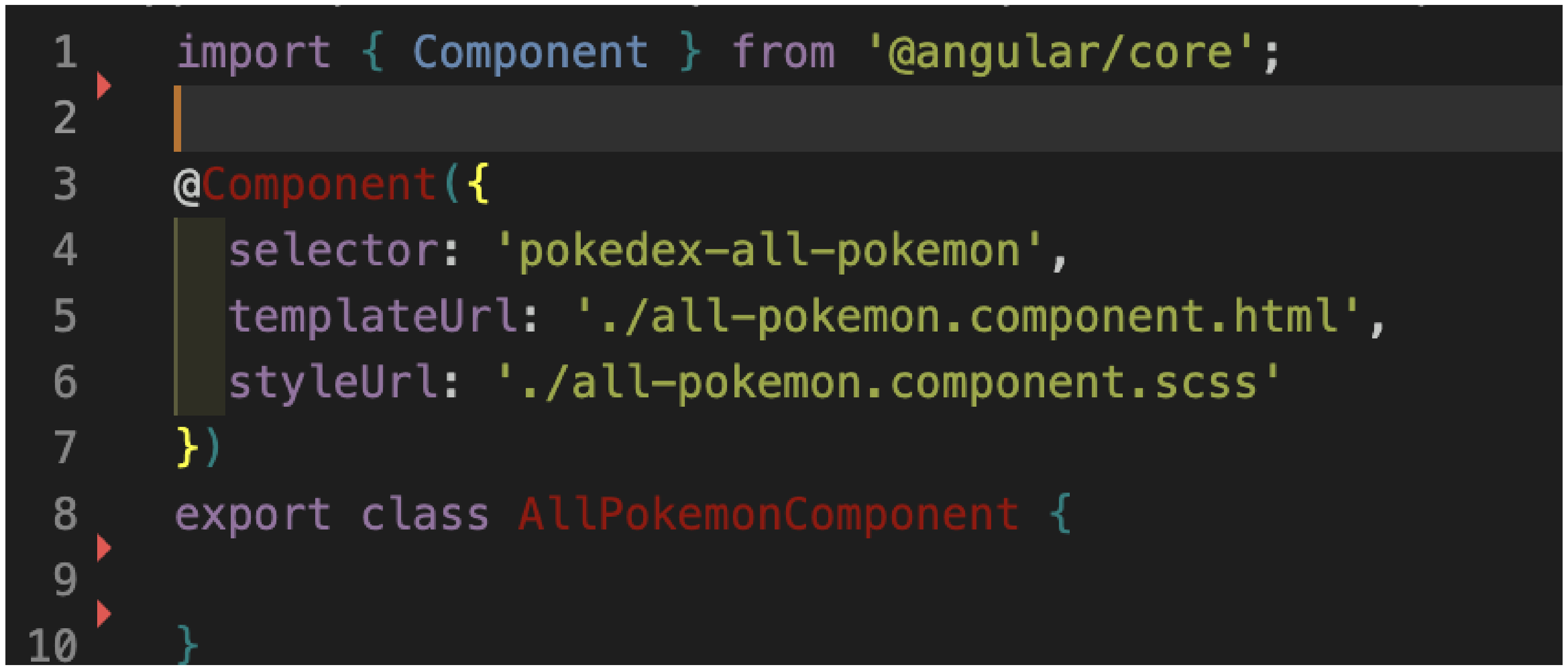
```
ng generate component app
```

```
ng g c app
```



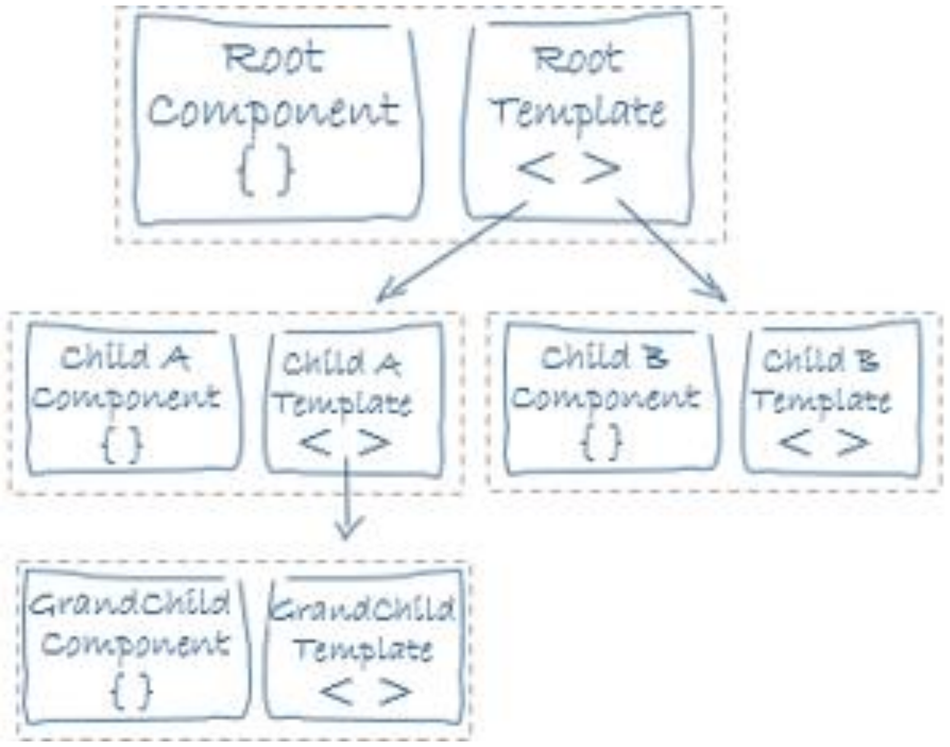
```
app
    app.component.html
    app.component.scss
    app.component.ts
```


all-pokemon.component.html
all-pokemon.component.scss
all-pokemon.component.spec.ts
all-pokemon.component.ts

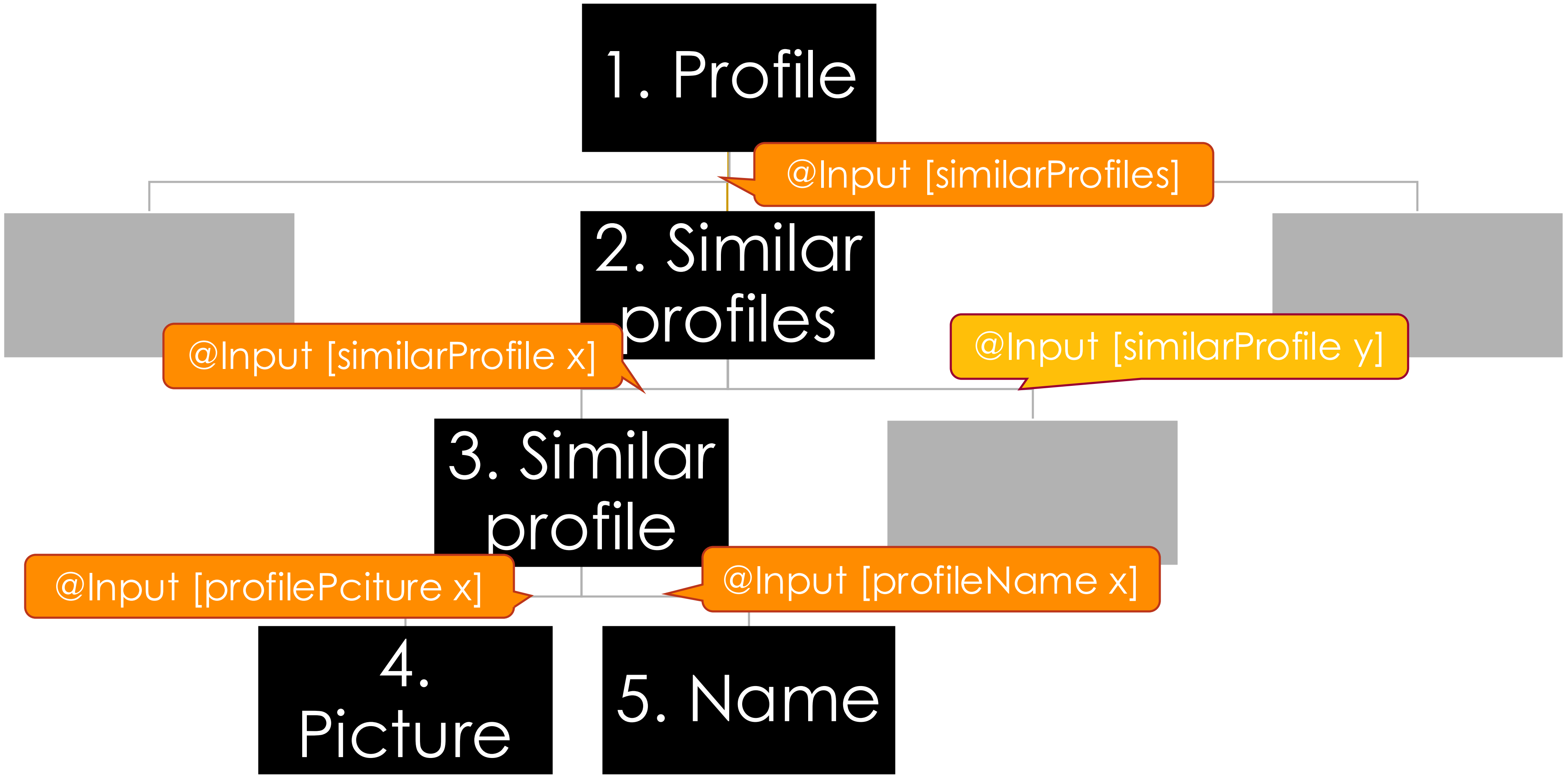

```
import { Component } from '@angular/core';

@Component({
  selector: 'pokedex-all-pokemon',
  templateUrl: './all-pokemon.component.html',
  styleUrl: './all-pokemon.component.scss'
})
export class AllPokemonComponent {

}
```

Components

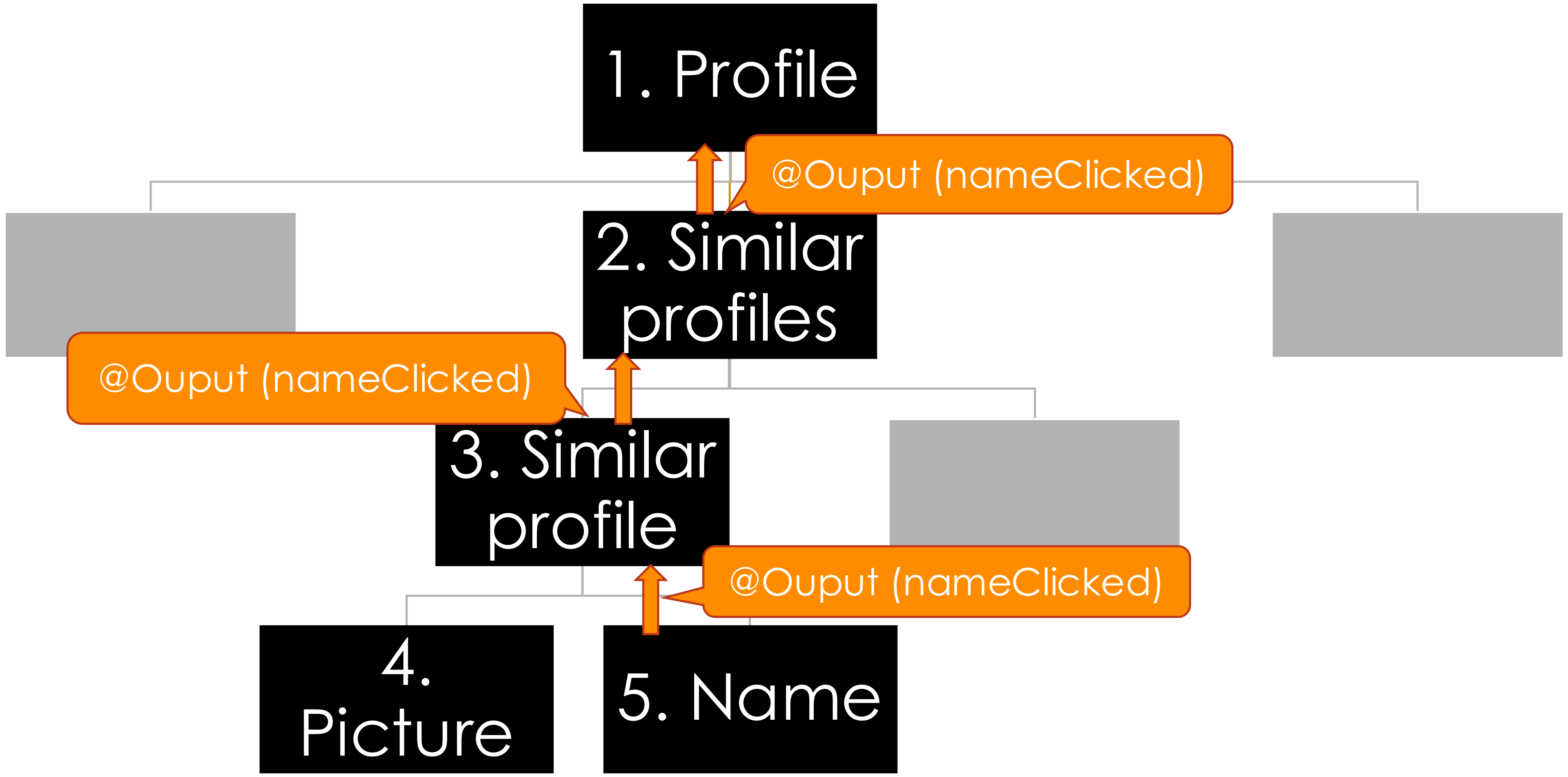
Hierarchically structured

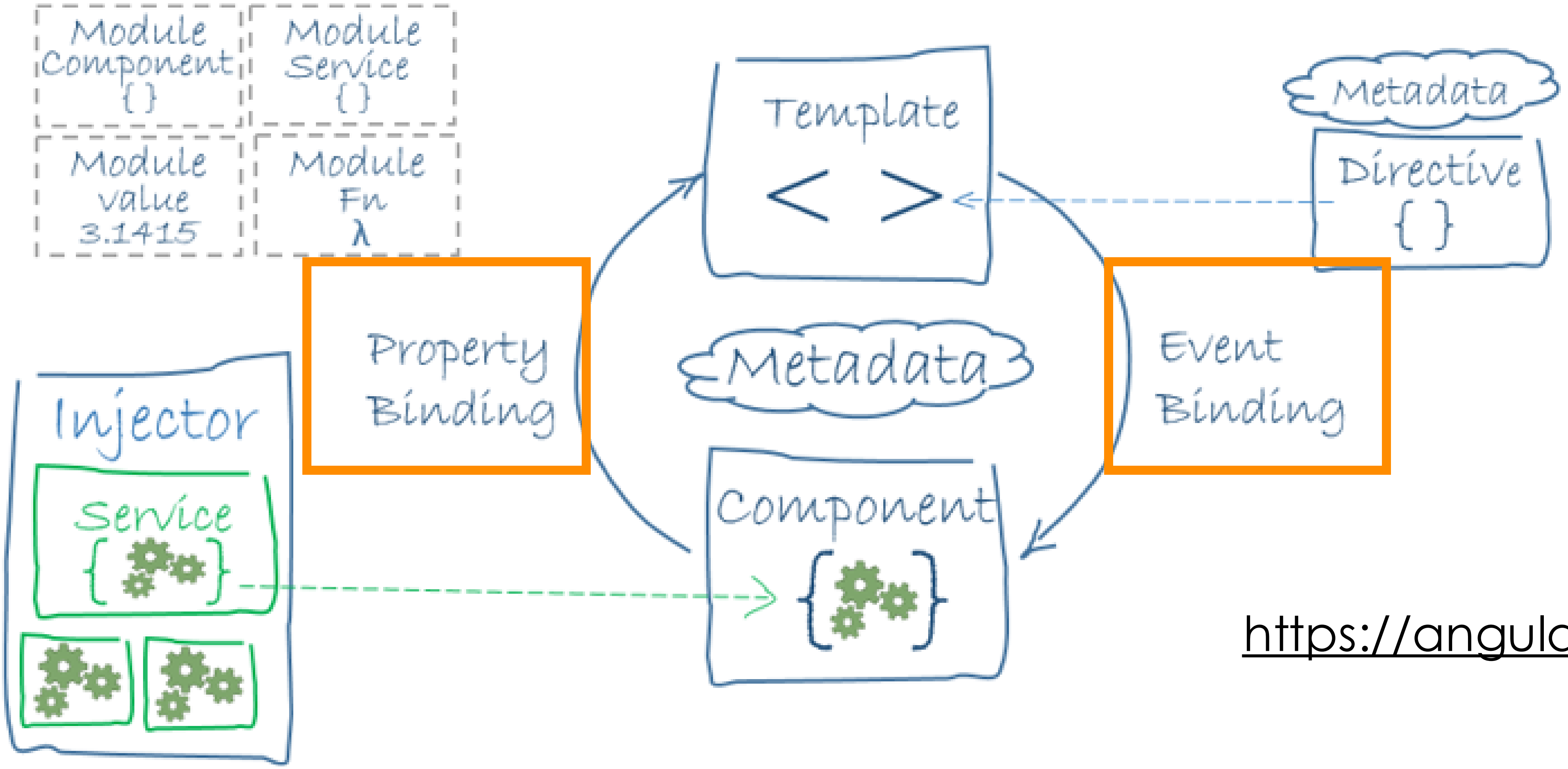


Components example



Components example

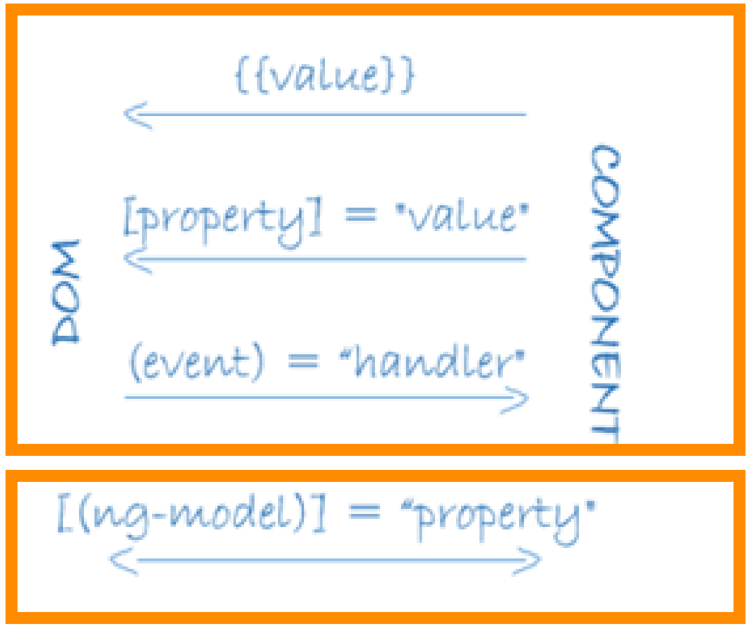




https://angular.io

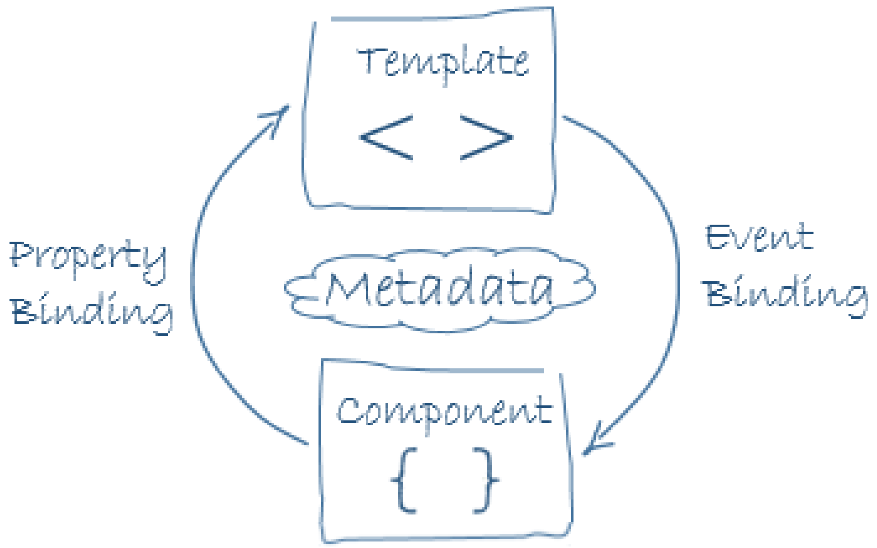
Data Binding

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



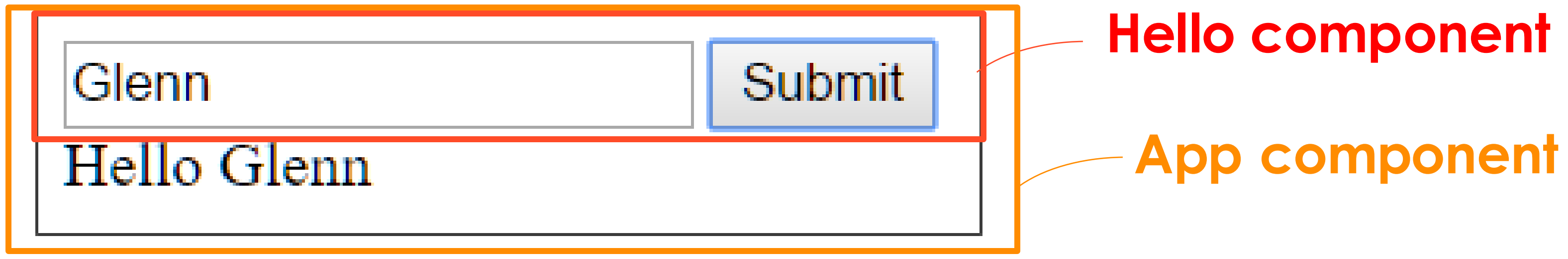
One-way binding

Two-way binding



Example

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

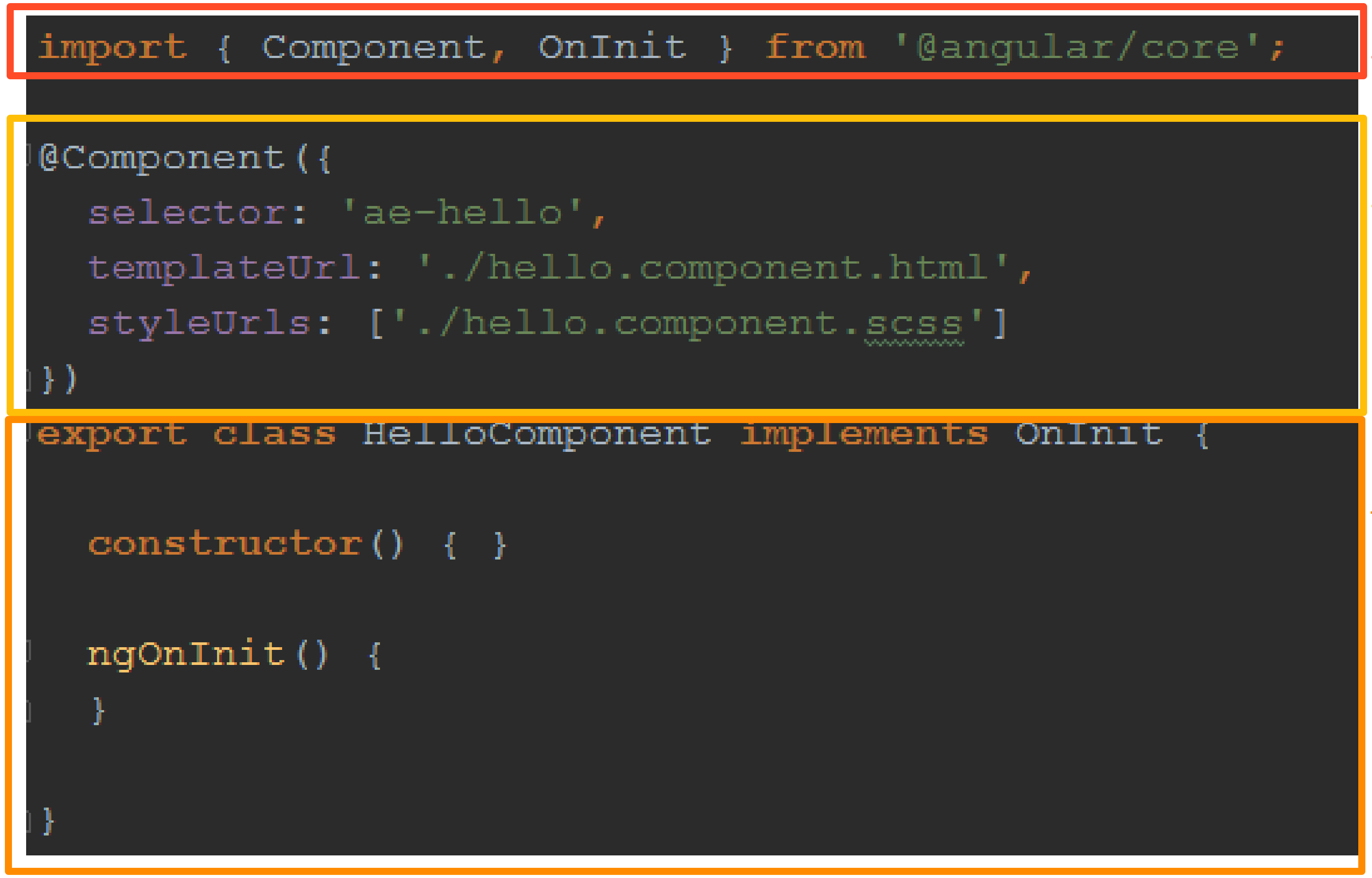

Glenn

Submit

Hello Glenn

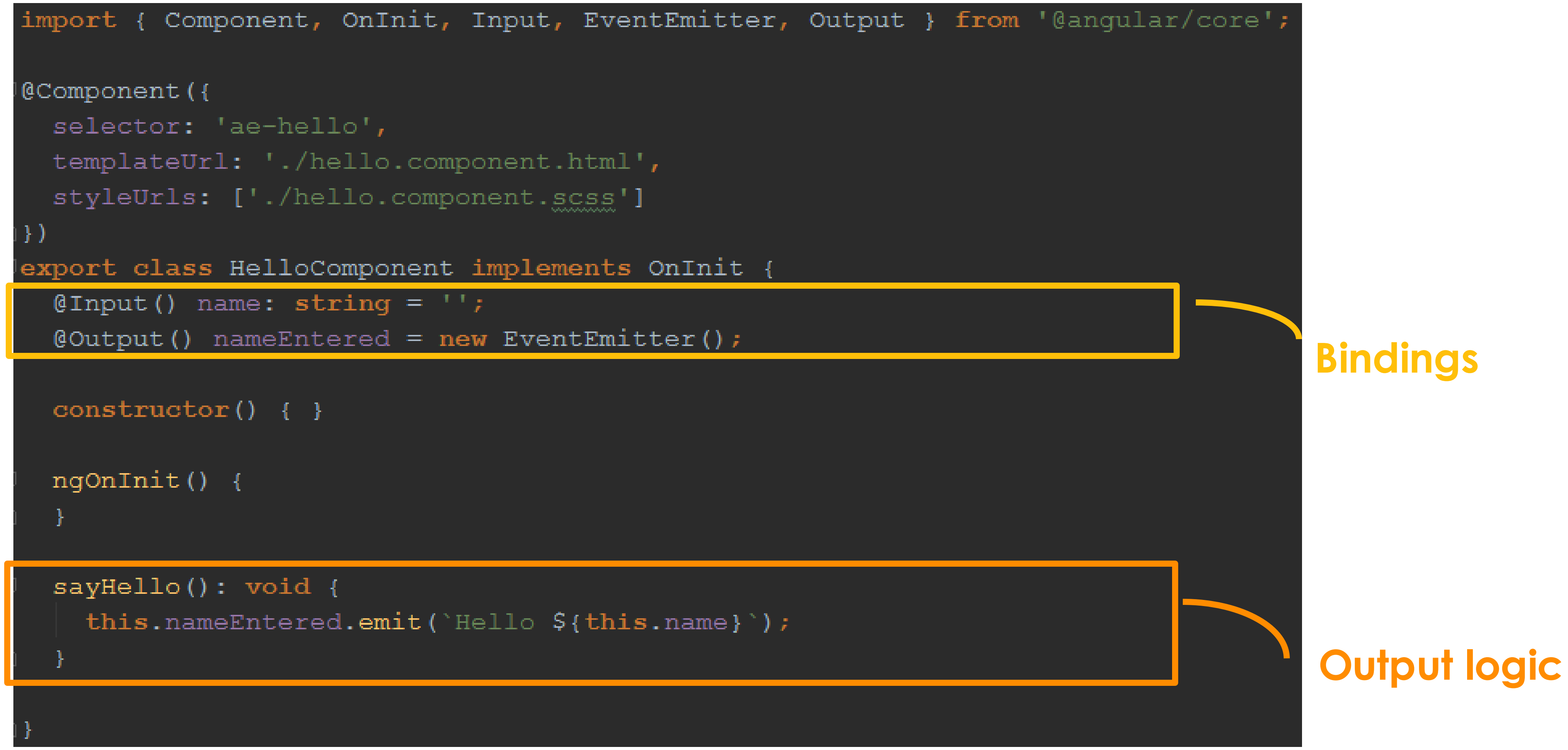
- Responsibility of ‘hello’ component
  - Get name via **INPUT**
  - Emit name after submit via **OUTPOUT**

hello.component.ts



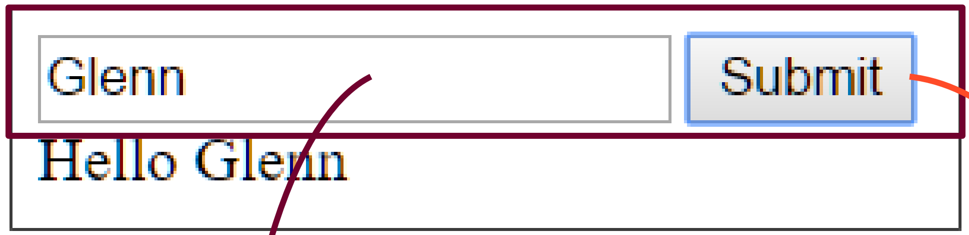
```
import { Component, OnInit } from '@angular/core';

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

  constructor() { }

  ngOnInit() {
  }

}
```

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}`);
  }

}
```

hello.component.html

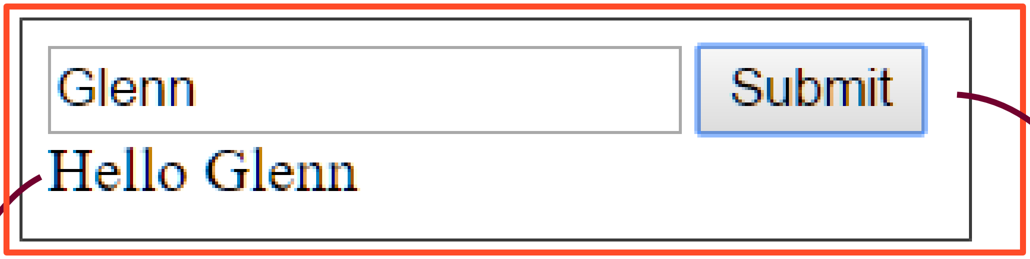


[(Two-way databinding)]

```
<input [(ngModel)]="name">
<button (click)="sayHello()">Submit</button>
```

(Output)

app.component.html



Hello Glenn

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

[Input]

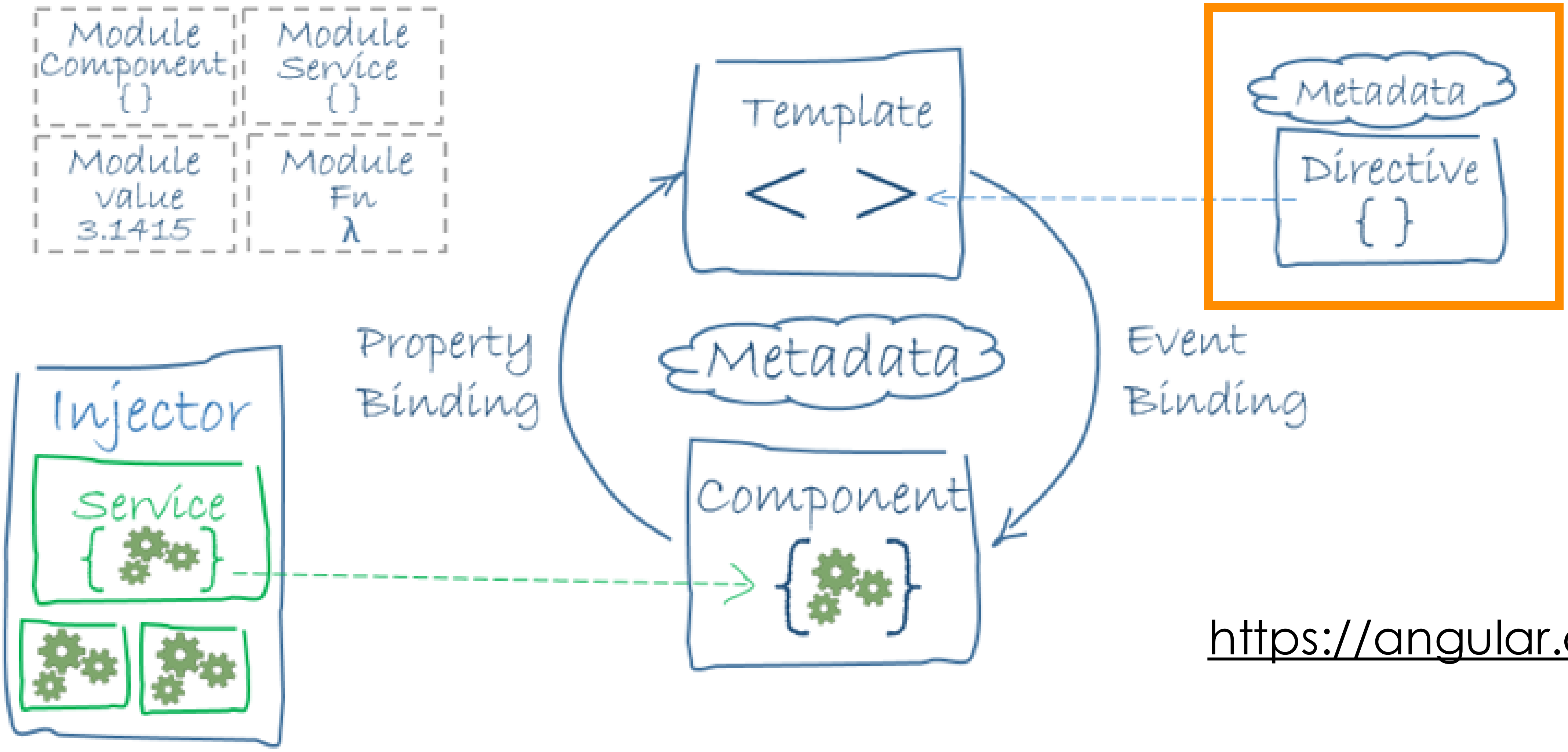
(Output)

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

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

Lifecycle hooks

constructor	
ngOnChanges	Respond when Angular (re)sets data-bound input properties
ngOnInit	Initialize the directive/component after Angular first displays the data-bound properties and sets the directive/component's input properties
ngDoCheck	Detect and act upon changes that Angular can't or won't detect on its own
ngAfterContentInit	Respond after Angular projects external content into the component's view
ngAfterContentChecked	Respond after Angular checks the content projected into the component
ngAfterViewInit	Respond after Angular initializes the component's views and child views
ngAfterViewChecked	Respond after Angular checks the component's views and child views
ngOnDestroy	Cleanup just before Angular destroys the directive/component



https://angular.dev

Directives `@Directive`

- Used for DOM manipulation
- Structural directives
  - Defined by Angular
  - @for, @if, @switch
  - 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

```
git checkout project-setup
```

- Create a standalone component all-pokemon.
- Add <pokedex-all-pokemon> to app.component.html and run your app

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

- **Add a component pokemon-list to app/shared**
  - **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**

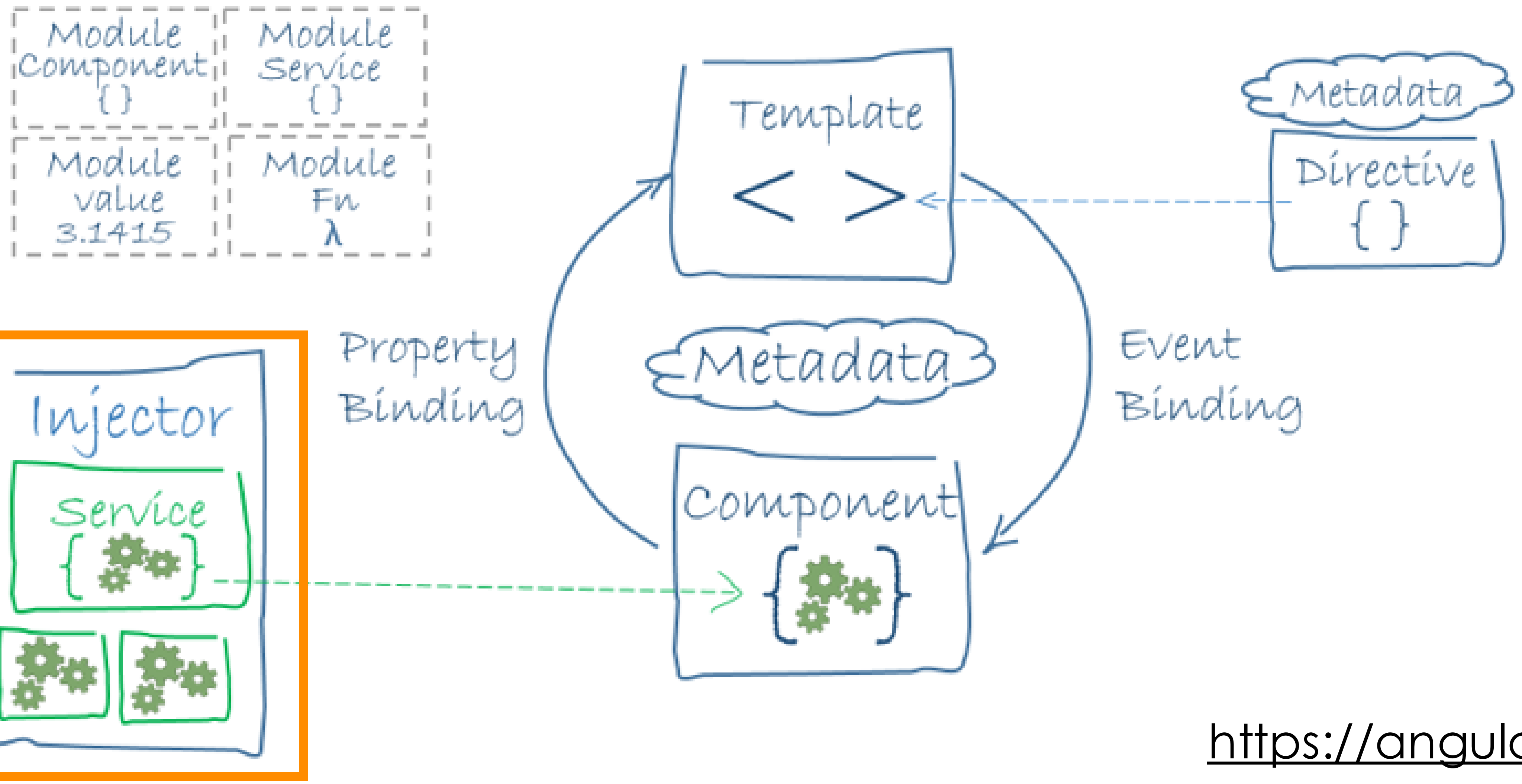
  @for (p of pokemon; track p.name) { … } @empty { … }

- 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**

```
table {
		border-collapse: collapse;
}

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

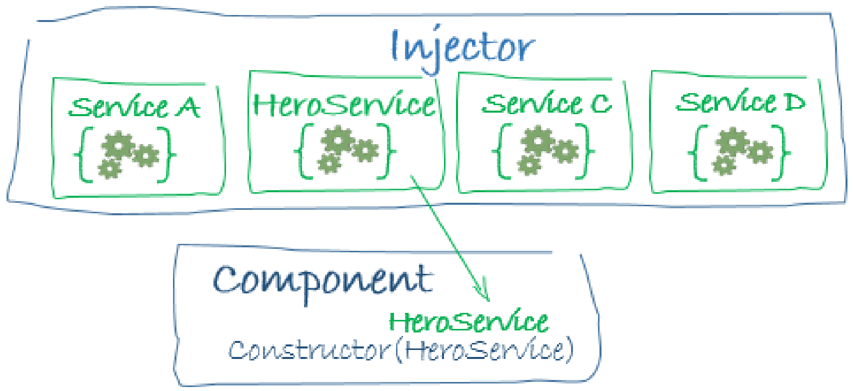
- **Call a function pokemonCaught() in all-pokemon component if button clicked**
  - ***Eg: print out the pokemon was caught***



https://angular.io

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'
})
```

- Since v17, moduled components are used less and less

  => Option 2 is generally the better option

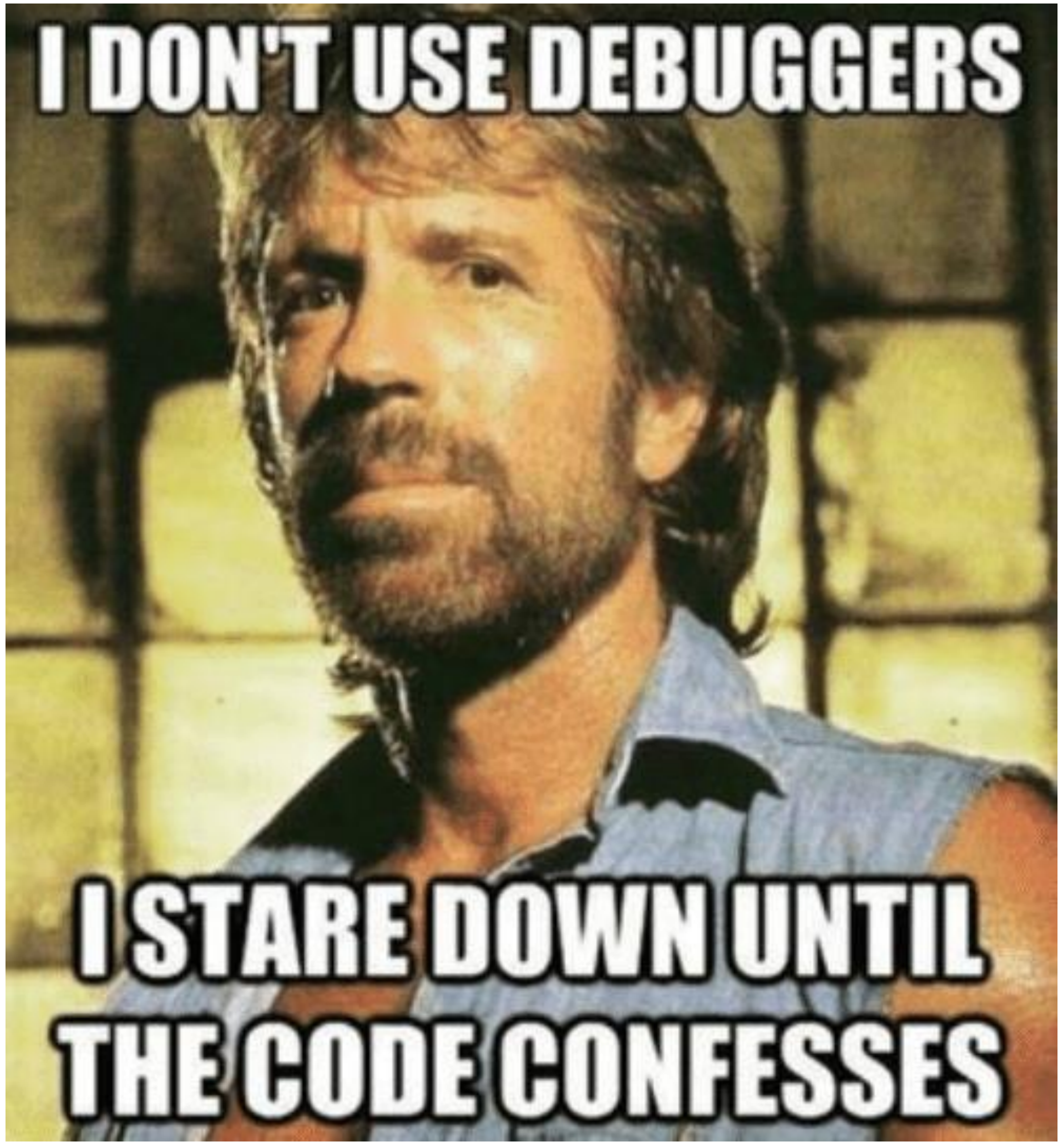
Exercise

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

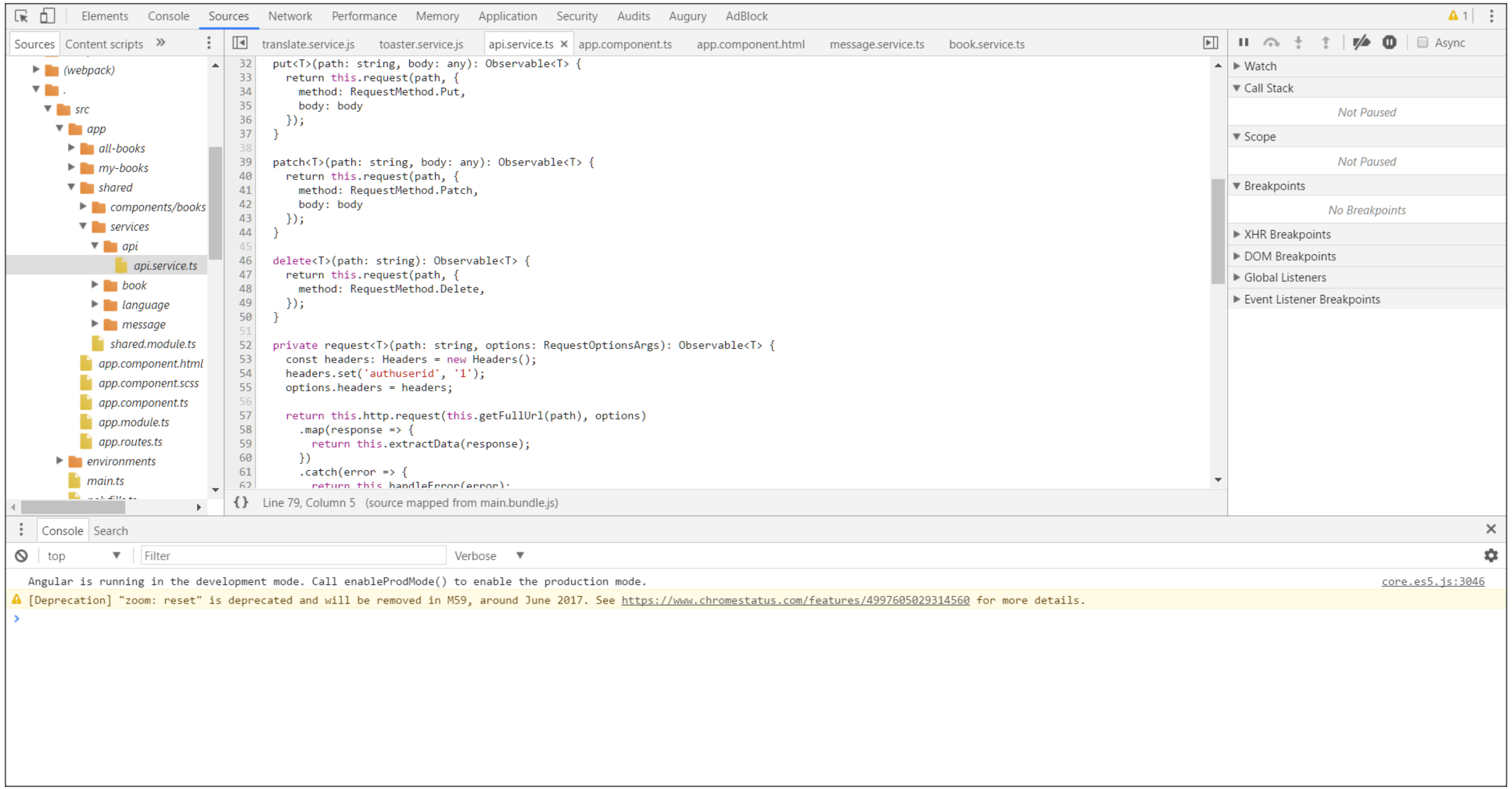
```
git checkout first-components
```

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

Debugging


I DON'T USE DEBUGGERS
I STARE DOWN UNTIL THE CODE CONFESSES

Chrome Dev Tools

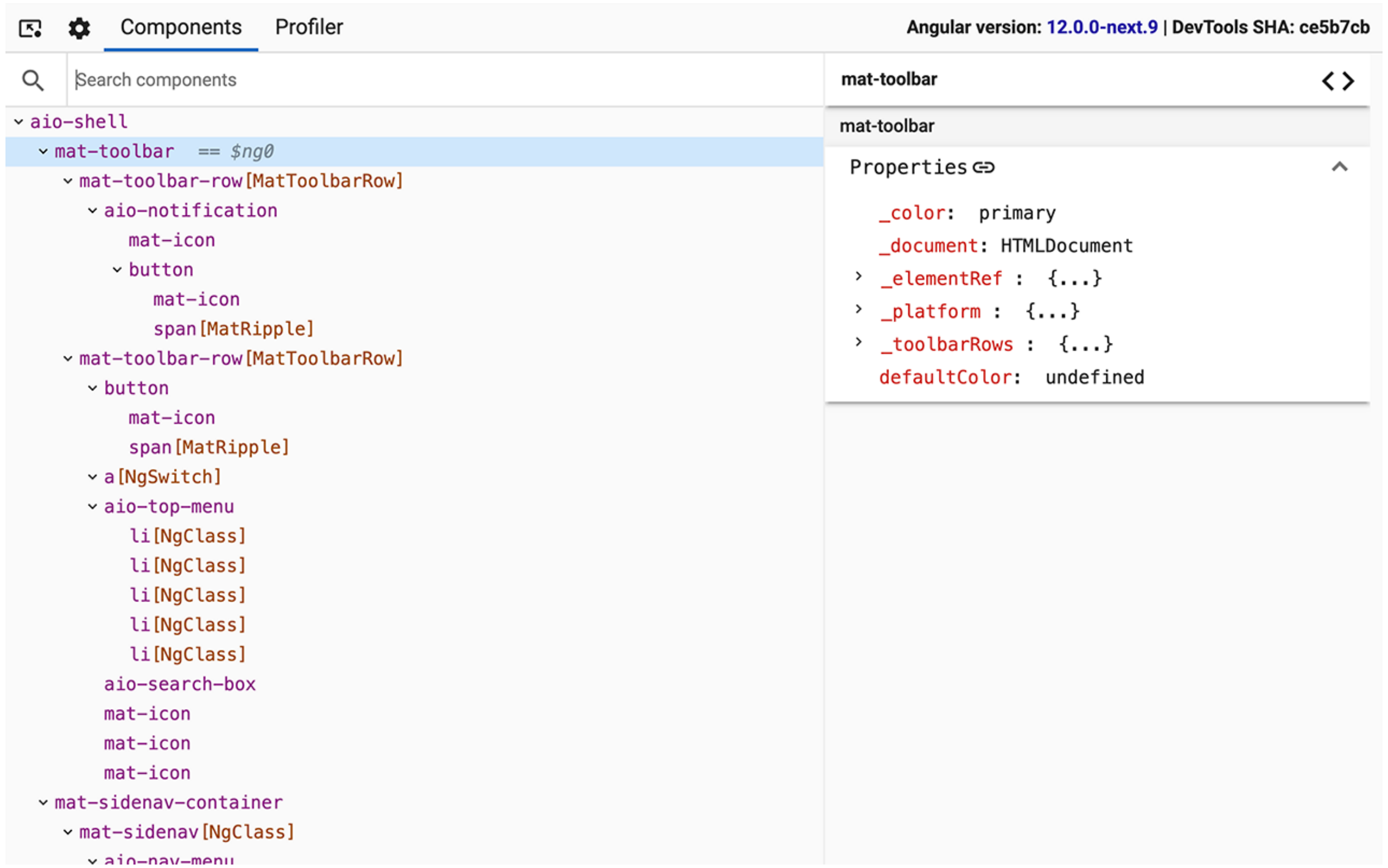


```
put<T>(path: string, body: any): Observable<T> {
  return this.request(path, {
    method: RequestMethod.Put,
    body: body
  });
}

patch<T>(path: string, body: any): Observable<T> {
  return this.request(path, {
    method: RequestMethod.Patch,
    body: body
  });
}

delete<T>(path: string): Observable<T> {
  return this.request(path, {
    method: RequestMethod.Delete,
  });
}

private request<T>(path: string, options: RequestOptionsArgs): Observable<T> {
  const headers: Headers = new Headers();
  headers.set('authuserid', '1');
  options.headers = headers;

  return this.http.request(this.getFullUrl(path), options)
    .map(response => {
      return this.extractData(response);
    })
    .catch(error => {
      return this.handleError(error);
```

```
Angular is running in the development mode. Call enableProdMode() to enable the production mode.        core.es5.js:3046
⚠ [Deprecation] "zoom: reset" is deprecated and will be removed in M59, around June 2017. See https://www.chromestatus.com/features/4997605029314560 for more details.
>
```

https://developer.chrome.com/devtools

Angular Dev Tools



```
aio-shell
  mat-toolbar  == $ng0
    mat-toolbar-row[MatToolbarRow]
      aio-notification
        mat-icon
        button
          mat-icon
          span[MatRipple]
    mat-toolbar-row[MatToolbarRow]
      button
        mat-icon
        span[MatRipple]
      a[NgSwitch]
      aio-top-menu
        li[NgClass]
        li[NgClass]
        li[NgClass]
        li[NgClass]
        li[NgClass]
      aio-search-box
      mat-icon
      mat-icon
      mat-icon
  mat-sidenav-container
    mat-sidenav[NgClass]
```

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

[Angular DevTools]

If all else fails…

One does ~~not~~
...simply
restart the server to fix it..

Exercise

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

```
git checkout first-service
```

- Install Bootstrap 5

```
npm i bootstrap
```

Exercise

- 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 Color · Bootstrap v5.3 (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

- **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>
```

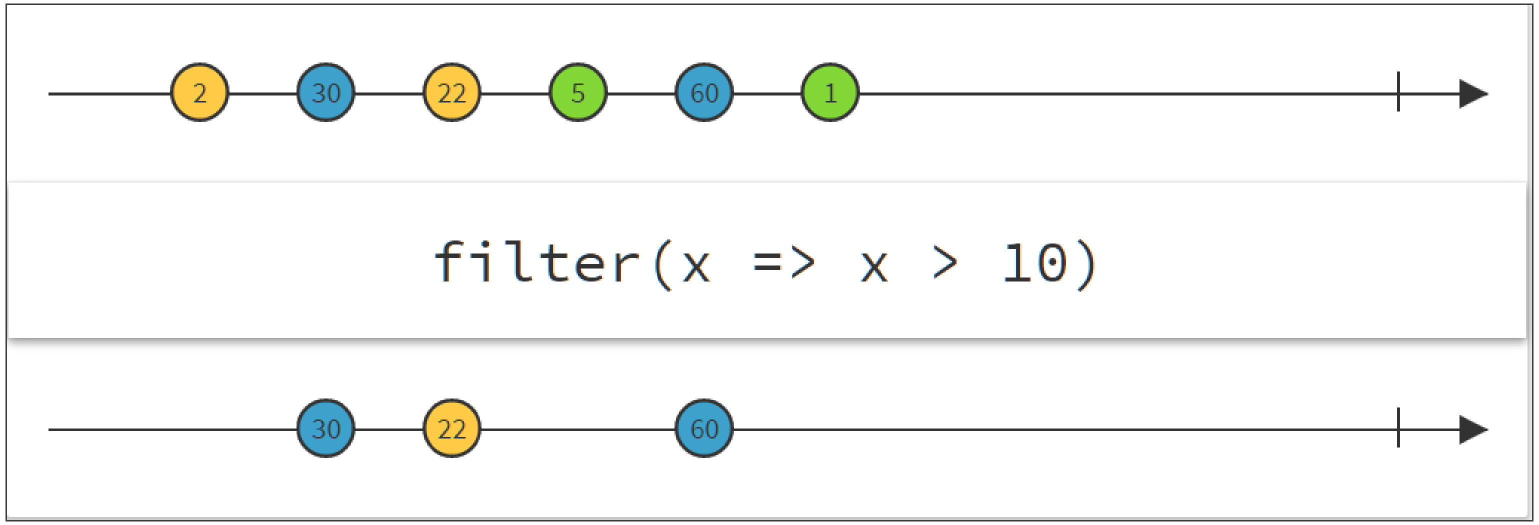
Intermezzo

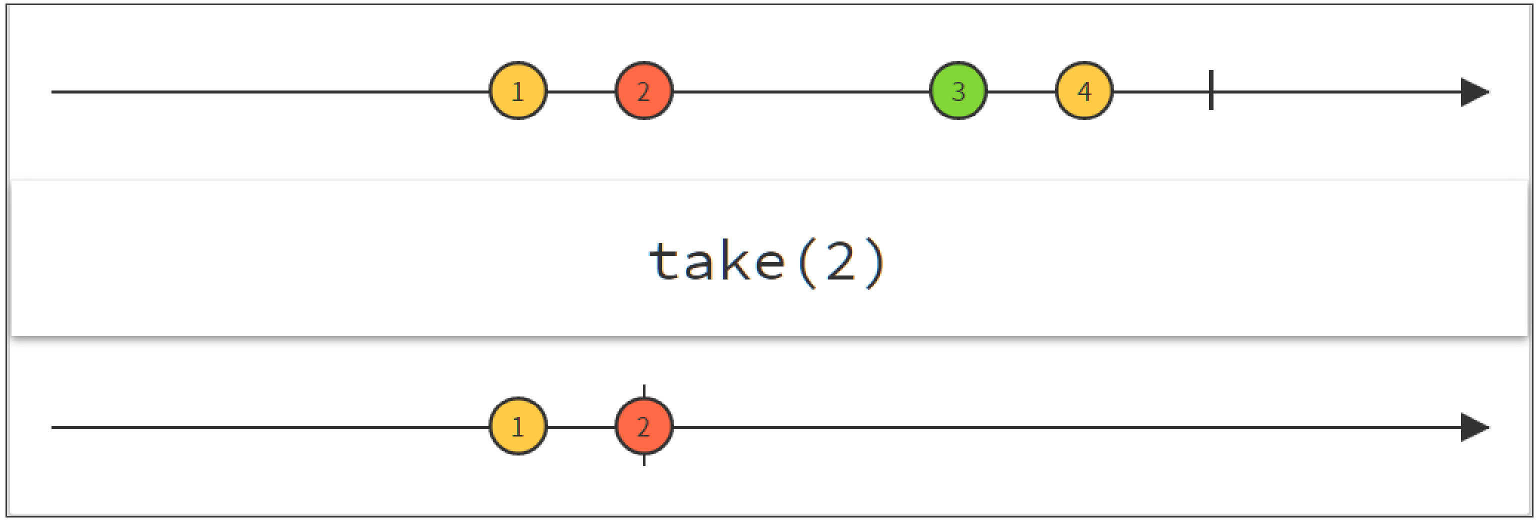
Component lifecycle hooks

Tapping into the hooks

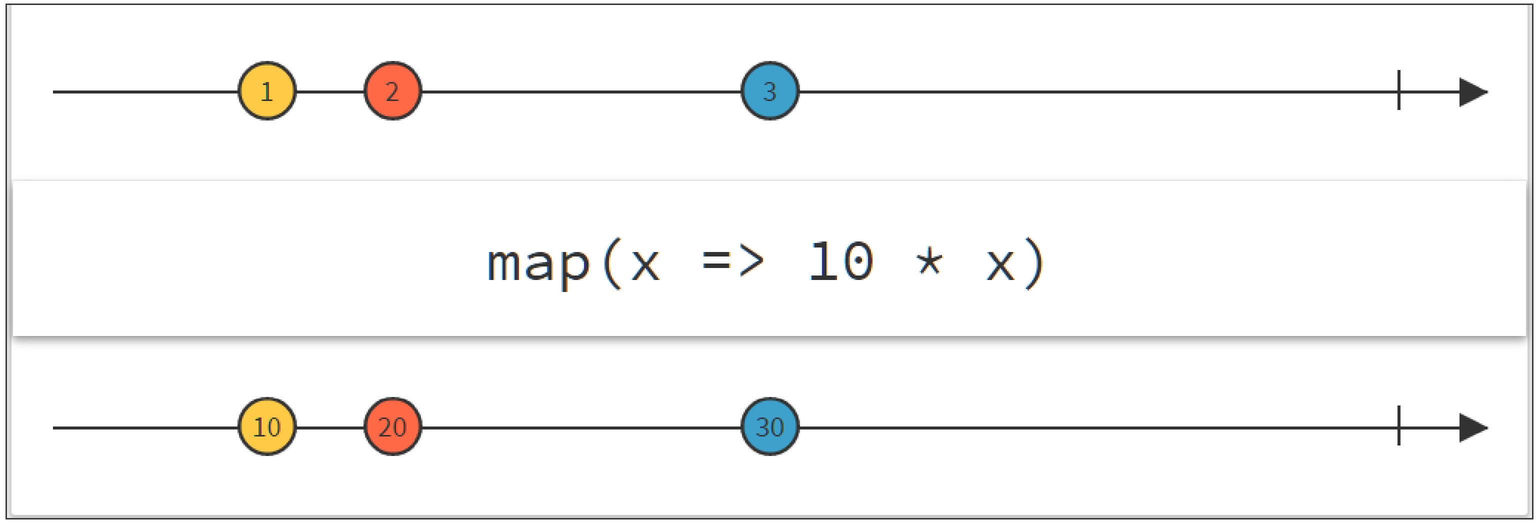
```
export class AllPokemonComponent implements OnInit {
  allPokemon$?: Observable<Pokemon[]>;

  constructor(private pokemonService: PokemonService) { }

      ngOnInit(): void {
            this.allPokemon$ = this.pokemonService.getAllPokemon();
      }

      catchPokemon(pokemon: Pokemon) {
            this.pokemonService.catchPokemon(pokemon.id).subscribe();
      }
}
```

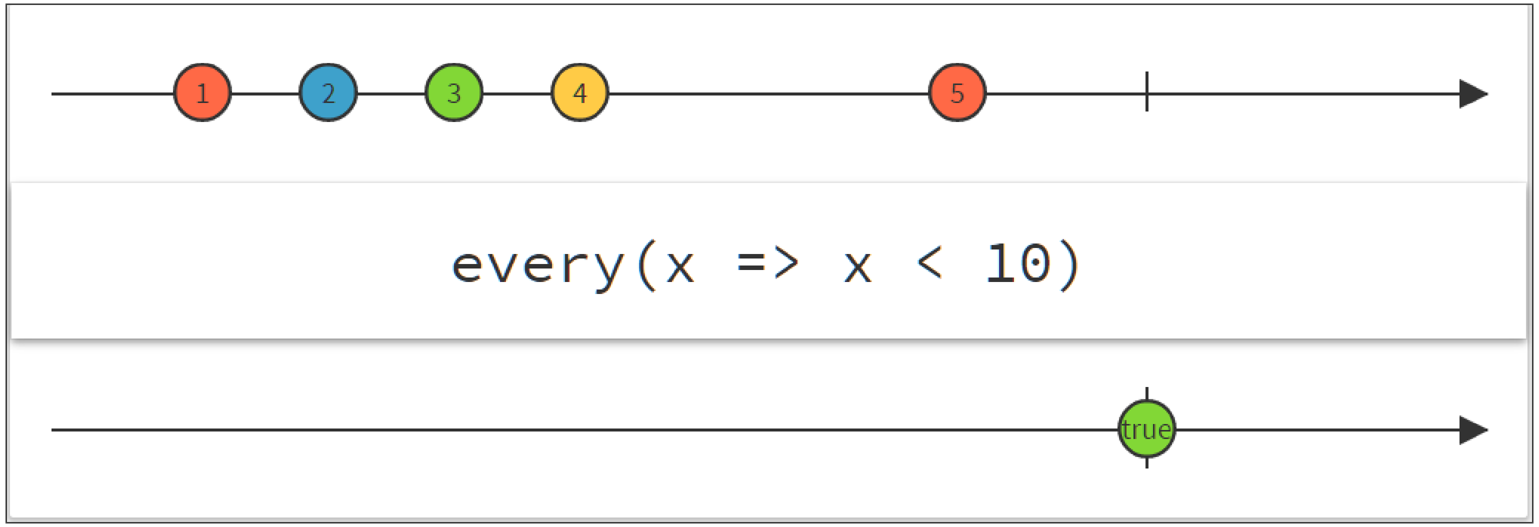
Operators



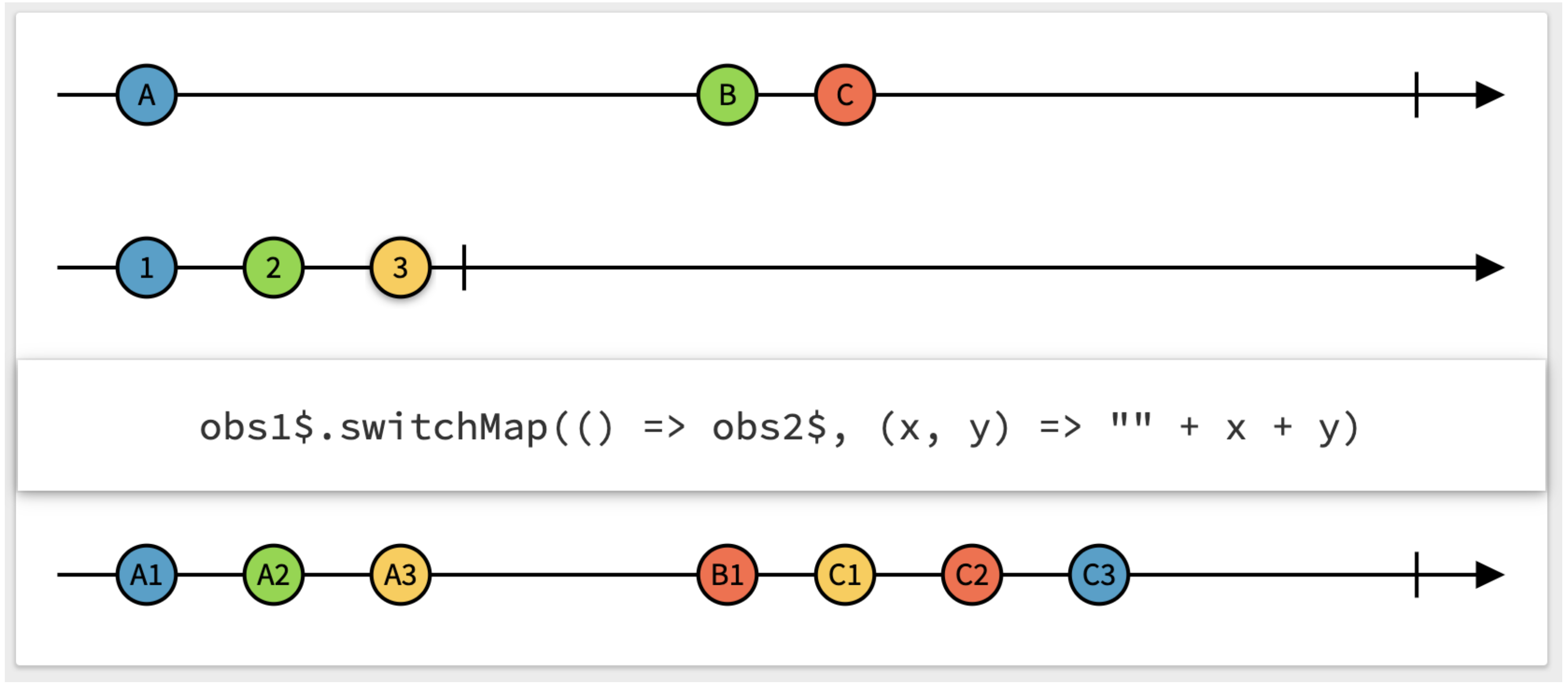


Operators

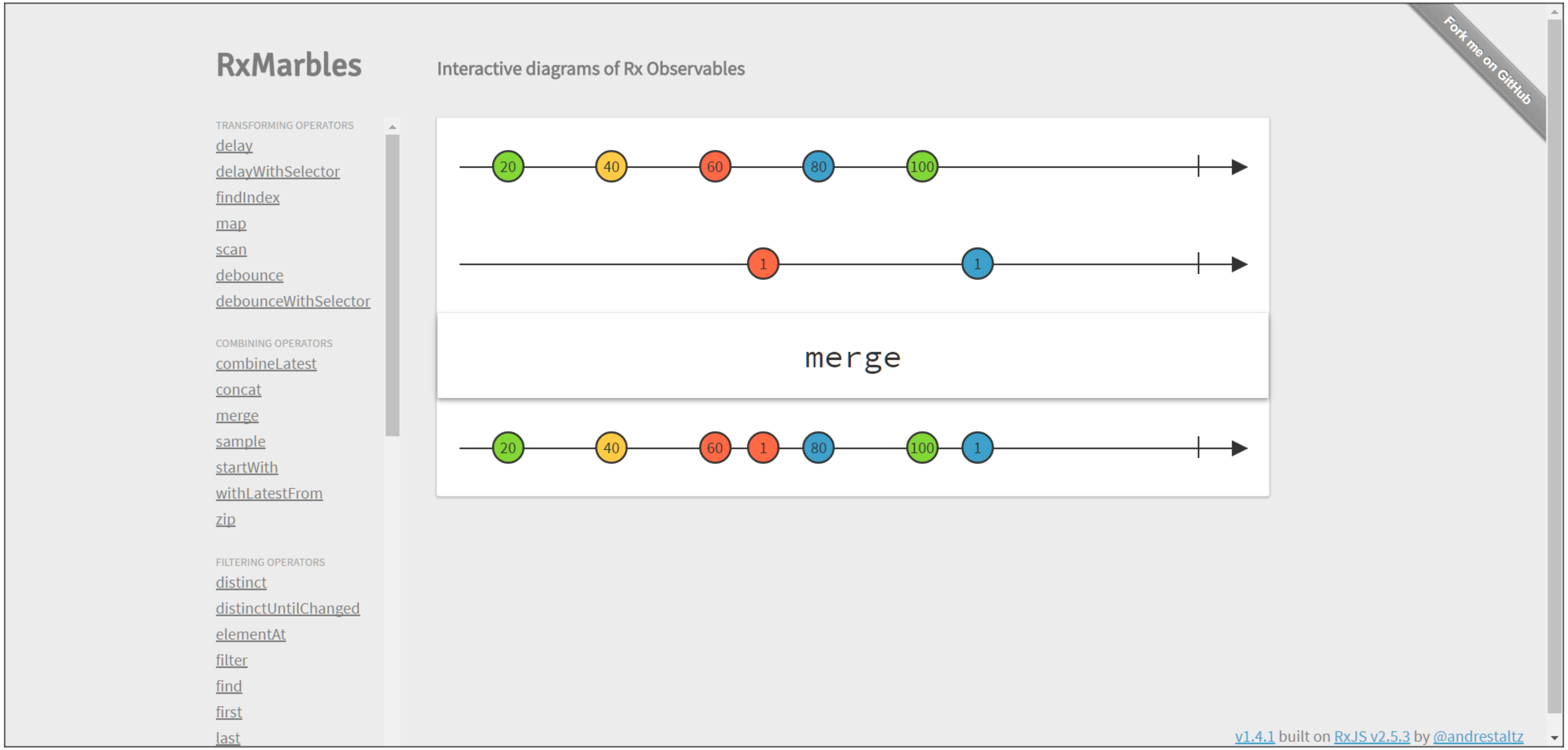




Operators



```
obs1$.switchMap(() => obs2$, (x, y) => "" + x + y)
```



RxMarbles

Interactive diagrams of Rx Observables

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

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.

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 succesful.
  - 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

- **RouterLink & RouterLinkActive** are imported into AppComponent

```
@Component({
  selector: 'app-root',
  imports: [RouterOutlet, RouterLink, RouterLinkActive],
  templateUrl: './app.component.html',
  styleUrl: './app.component.scss'
})
export class AppComponent { }
```

app.routes.ts

```
const routes: Routes = [
  { path: 'first-component', component: FirstComponent },
  { path: 'second-component', component: SecondComponent },
];
```

Lazy loading components

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

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 in the router outlet:

```
<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 Pokémon

- 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

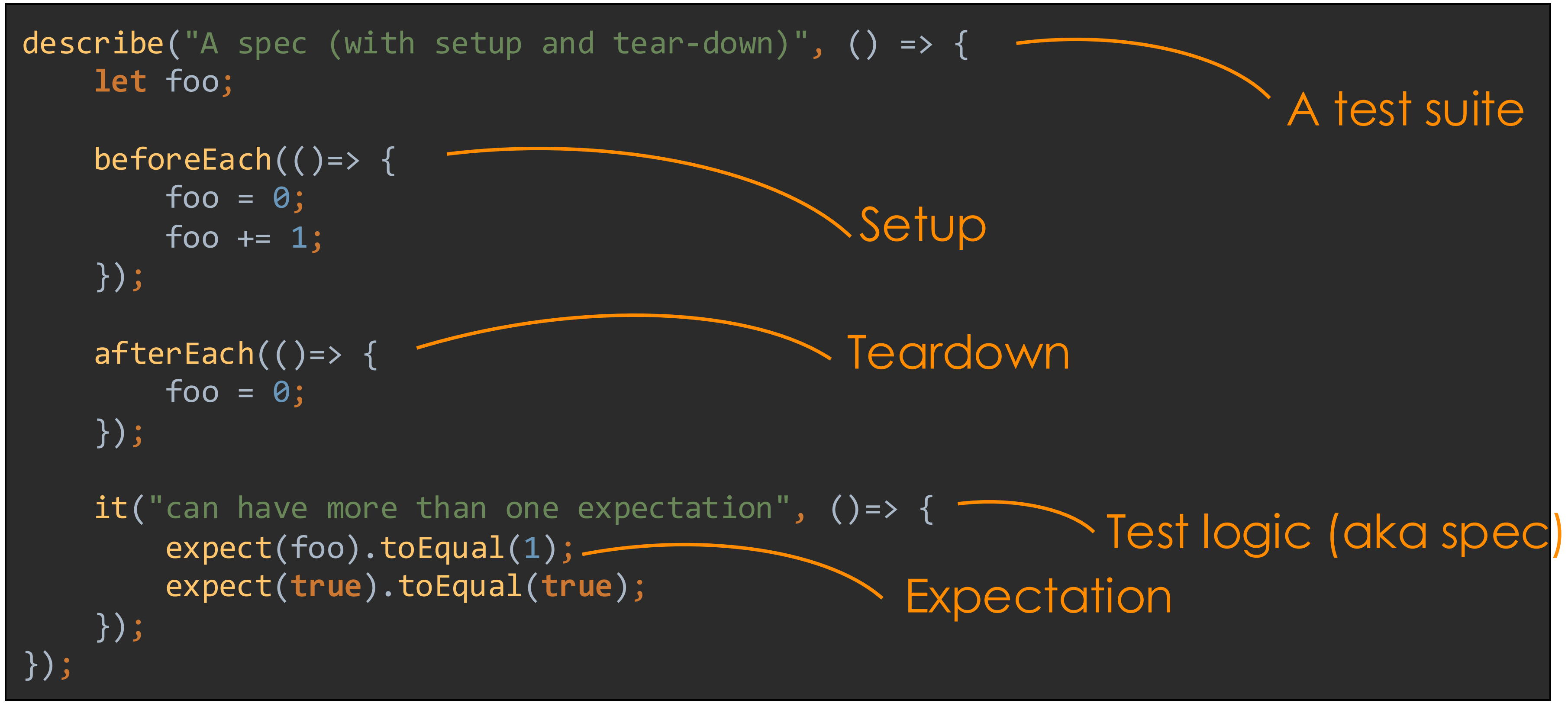
ALRIGHT YOU JOLLY JOKERS...
ITS TESTING TIME
imgflip.com

QA default in CLI

Jasmine

{TS Lint}

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);
    });
});
```

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
```

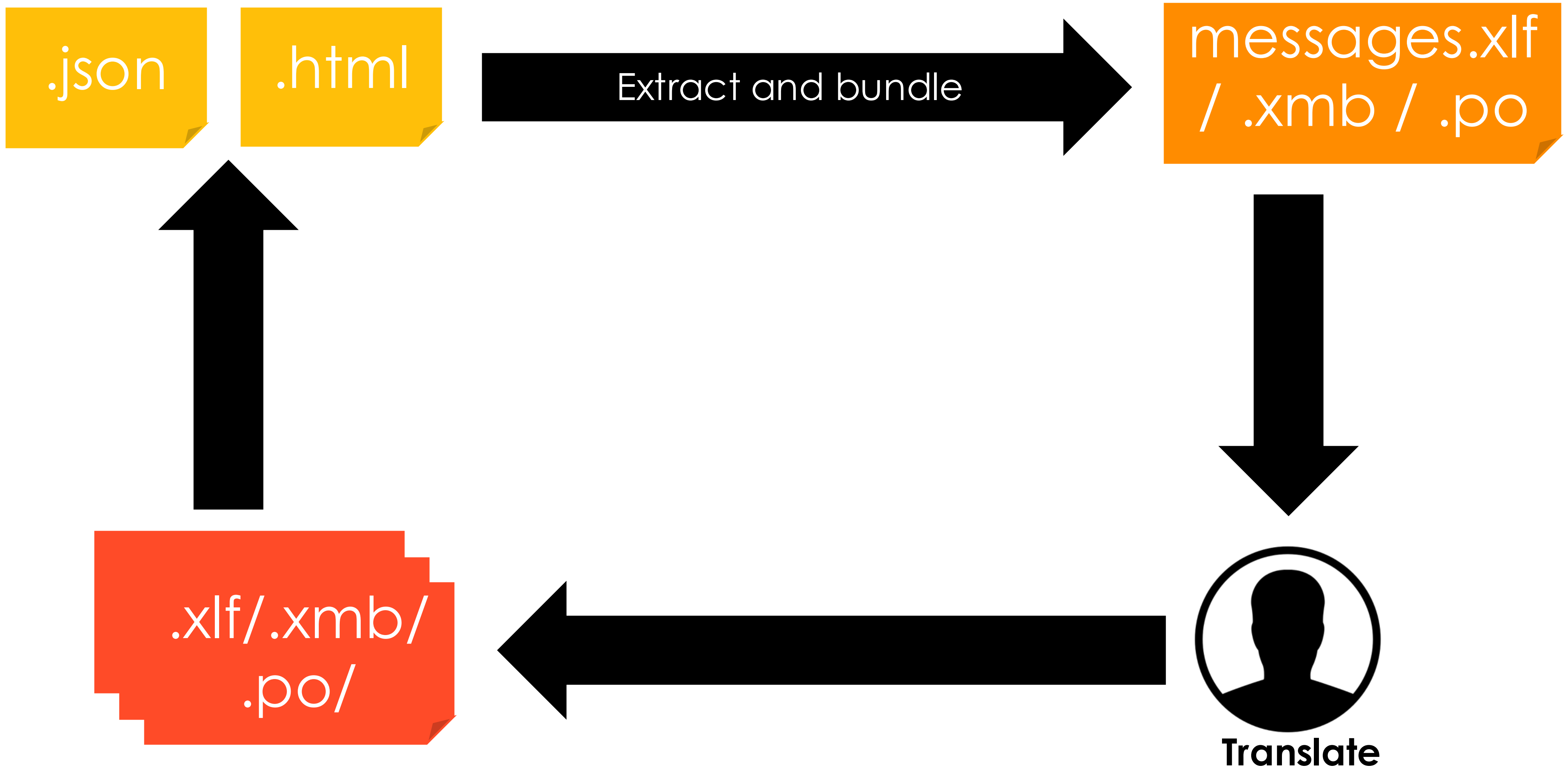
- Test the showApiError function in MessageService

https://angular.io/docs/ts/latest/guide/testing.html

I18N

How to handle translations

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 wolven} =1 {een wolf} =2 {twee wolven} other {een roedel wolven}}</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.

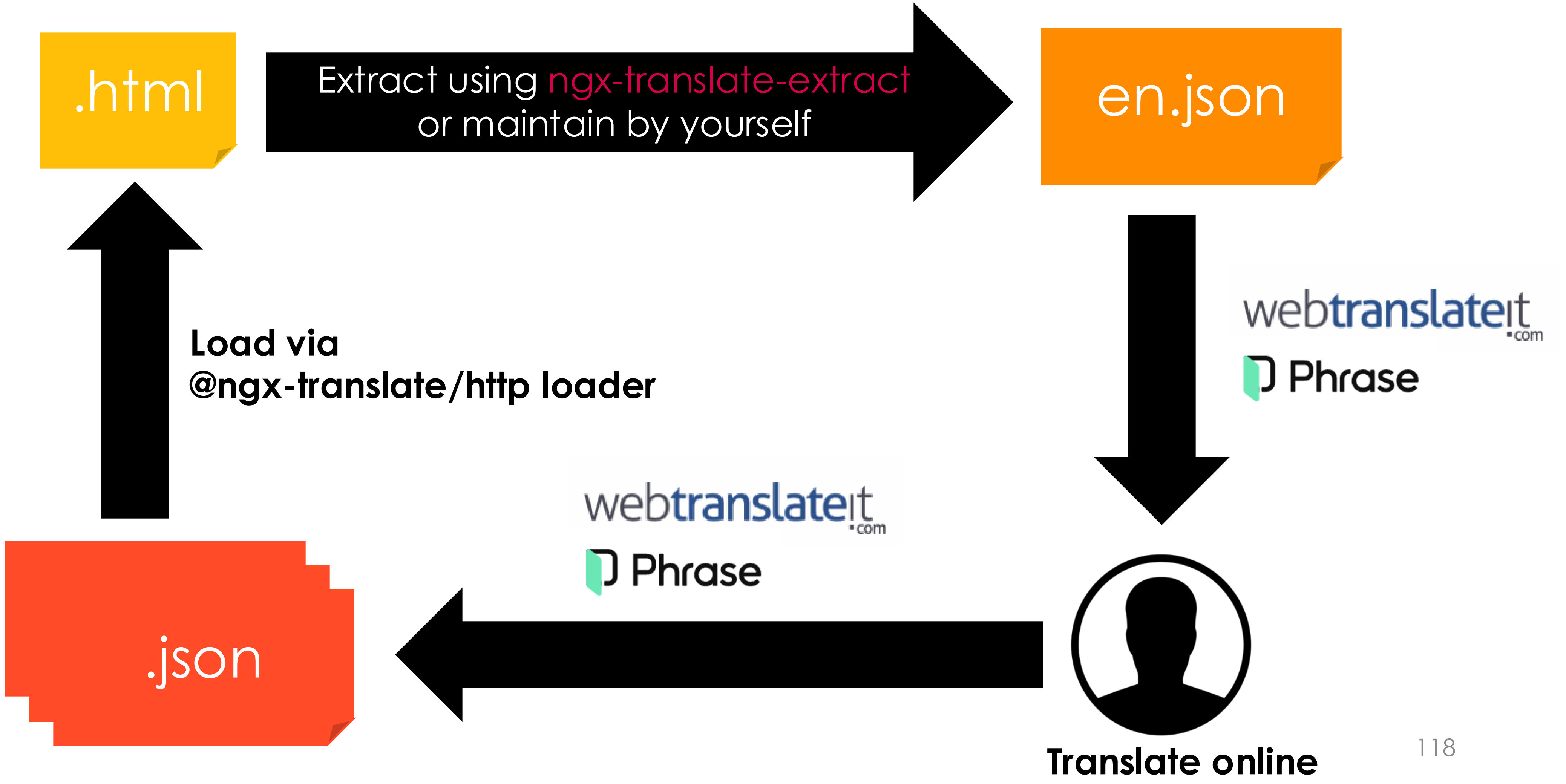
BOOOOOOO

ngx-translate

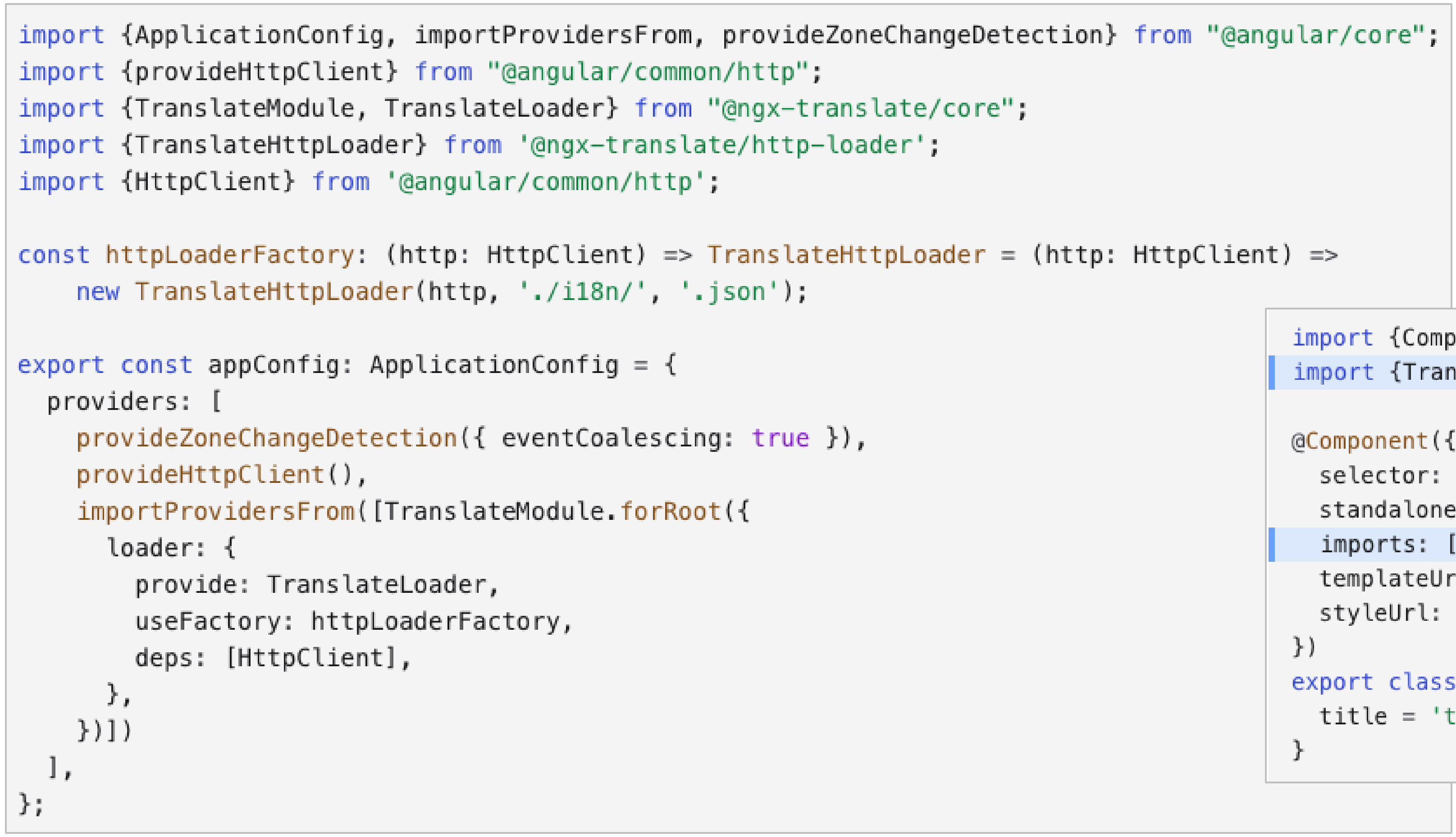
- Modular i18n library providing service, directive and pipe
- Switching between languages doesn’t reload whole app
- By default no process defined
  - But one can set up easily by using
    - Plugins
    - Webtranslateit
  - Or the road in between: ngx-translate-extract

ngx-translate
The internationalization (i18n) library for Angular 2+

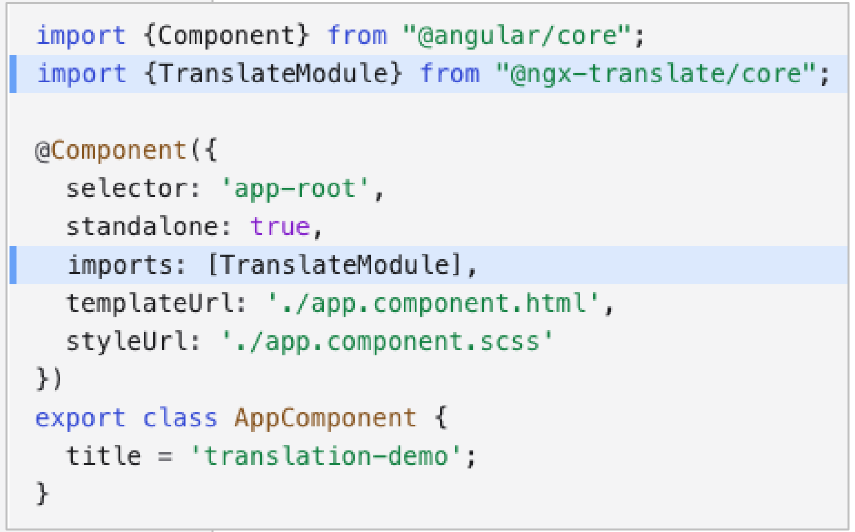
Runtime process



Setup – Import module (root)



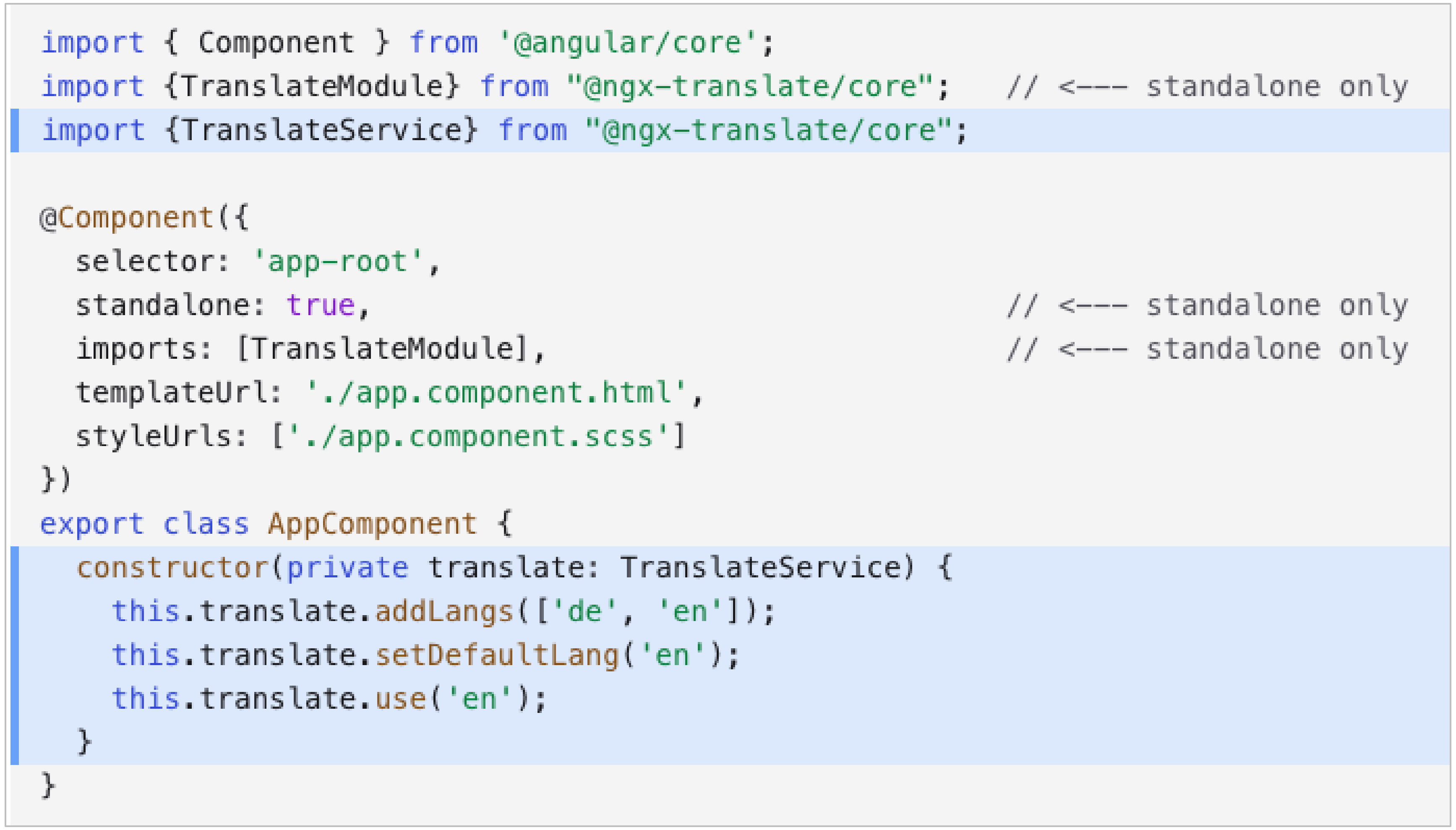
```typescript
import {ApplicationConfig, importProvidersFrom, provideZoneChangeDetection} from "@angular/core";
import {provideHttpClient} from "@angular/common/http";
import {TranslateModule, TranslateLoader} from "@ngx-translate/core";
import {TranslateHttpLoader} from '@ngx-translate/http-loader';
import {HttpClient} from '@angular/common/http';

const httpLoaderFactory: (http: HttpClient) => TranslateHttpLoader = (http: HttpClient) =>
    new TranslateHttpLoader(http, './i18n/', '.json');

export const appConfig: ApplicationConfig = {
  providers: [
    provideZoneChangeDetection({ eventCoalescing: true }),
    provideHttpClient(),
    importProvidersFrom([TranslateModule.forRoot({
      loader: {
        provide: TranslateLoader,
        useFactory: httpLoaderFactory,
        deps: [HttpClient],
      },
    })])
  ],
};
```



```typescript
import {Component} from "@angular/core";
import {TranslateModule} from "@ngx-translate/core";

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [TranslateModule],
  templateUrl: './app.component.html',
  styleUrl: './app.component.scss'
})
export class AppComponent {
  title = 'translation-demo';
}
```

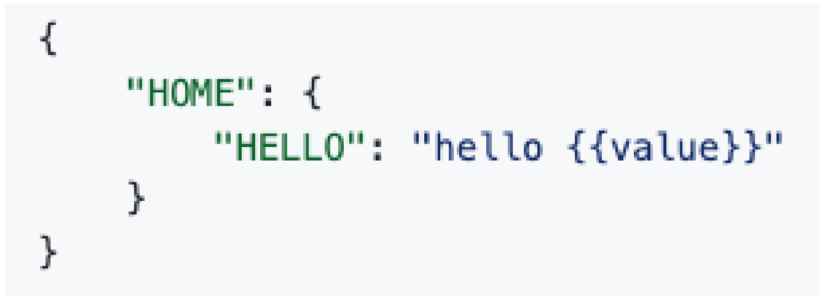
ngx-translate installation

Setup – Set default language



```
import { Component } from '@angular/core';
import {TranslateModule} from "@ngx-translate/core";   // <--- standalone only
import {TranslateService} from "@ngx-translate/core";

@Component({
  selector: 'app-root',
  standalone: true,                                     // <--- standalone only
  imports: [TranslateModule],                           // <--- standalone only
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  constructor(private translate: TranslateService) {
    this.translate.addLangs(['de', 'en']);
    this.translate.setDefaultLang('en');
    this.translate.use('en');
  }
}
```

Define translations



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

Usage with pipes

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

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**

Exercise

Also if you did finish last exercise:

```
git checkout qa
```

```
npm i @ngx-translate/core (--legacy-peer-deps)
```

```
Npm i @ngx-translate/http-loader
```

- **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**
- 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 (including signals) 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