Angular Basics – I

EPAM Samara· Autumn 2017

ABOUT ME

Dmitry Popov

EPAM Systems,
Lead Software Engineer

- Experience: 10 years in software development
- Main Focus: web application development using Angular and React JS frameworks
- Wide experience in development of various solutions starting from web content management systems up to data-driven workflow engines.

AGENDA

- Introduction
- Components
- Pipes & Directives
- DI & Services
- Routing

INTRODUCTION

INTRODUCTION

- Components
  - CSS encapsulation
  - ~~$scope~~, ~~controller~~
  - Event based data binding, no more digest cycles!
- ES6/Typescript
- Server-side rendering
- Reactive, ngrx
- Redux compatible architecture
- Enhanced DI
- DynamicComoponentLoader
- … more other features!

Example

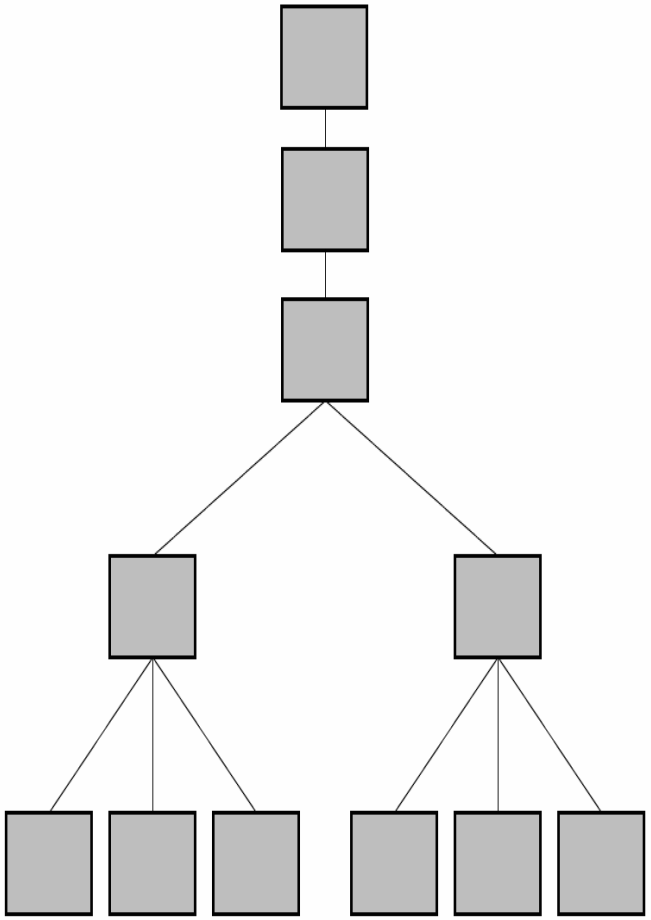
COMPONENTS

COMPONENT

- Declare reusable UI building blocks for an application.
- The **@Component** annotation specifies when a component is instantiated, and which properties and hostListeners it binds to.

COMPONENT'S DEFINITION

- An Angular application is a **tree** of components.
- A component controls a patch of screen real estate that we could call a ***view***.
- We define a component's application logic inside a **class**. The class interacts with the view through an API of properties and methods.



COMPONENT === DIRECTIVE?

- Components

```
<my-component></my-component>
```

- *Structural directives

```
<div *myDirective></div>
```

- [Attribute directives]

```
<div myDirectie></div>
<div [myDirectie]></div>
<div (myDirectie)></div>
<div [(myDirectie)]></div>
```

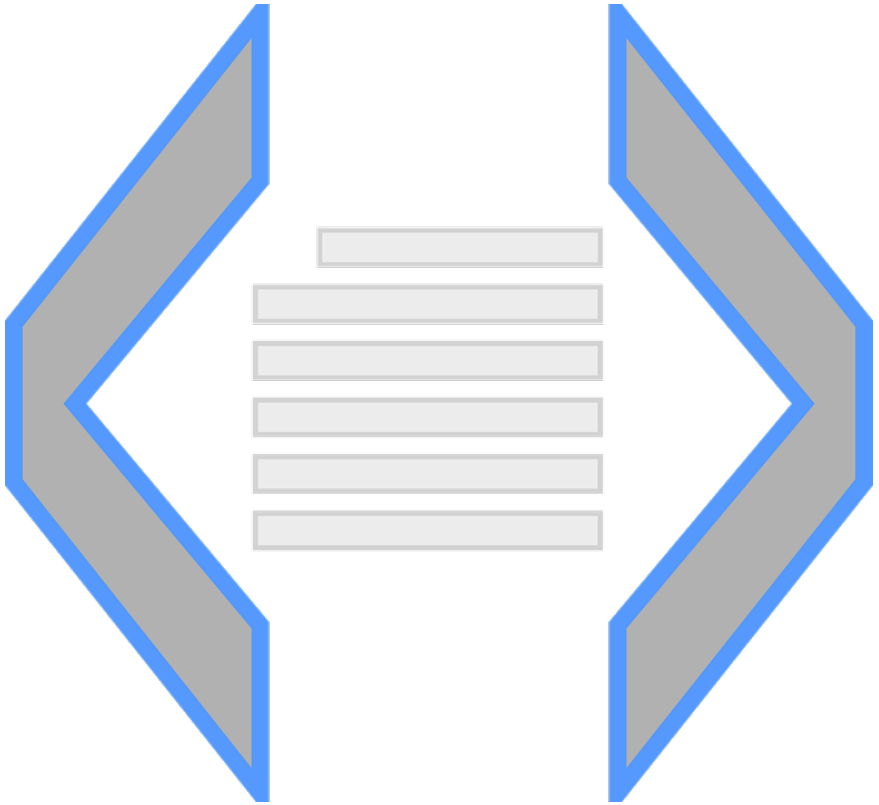
COMPONENT CONFIGURATION

- selector
- templateUrl
- directives
- providers

```
@Component({
  selector: 'my-app',
  template: `
    <app-component></app-component>
  `,
  directives: [ MyAppComponent ],
  providers: [ LoginService ]
})
export class AppComponent {}
```

TEMPLATE

- A template is a form of HTML that tells Angular how to render the component.
  - interpolation {{ }}
  - property binding [ ]
  - event binding ( )
  - two-way data binding [()]



TEMPLATE EXPRESSIONS

JavaScript expressions that have or promote side effects are prohibited, including:

- assignments (=, +=, -=, ...)
- new
- chaining expressions with ; or ,
- increment and decrement operators (++ and --)
- no support for the bitwise operators | and &

PROPERTY BINDING

```
<img [src]="heroImageUrl">

<img bind-src="heroImageUrl">

<button [disabled]="isUnchanged">Cancel</button>

<div [ngClass]="classes">Binding to the classes property</div>

<hero-detail [hero]="currentHero"></hero-detail>
```

ONE-TIME STRING INITIALIZATION

We *should* omit the brackets when all of the following are true:

- The target property accepts a string value.
- The string is a fixed value that we can bake into the template.
- This initial value never changes.

```
<hero-detail prefix="You are my" [hero]="currentHero"></hero-detail>
```

ATTRIBUTE BINDING

- Raise an error:

```
<tr><td colspan="{{1 + 1}}">Three-Four</td></tr>
```

- Correct:

```
<tr><td [attr.colspan]="1 + 1">One-Two</td></tr>
```

COMPONENT STYLES

- We may define not only an HTML template, but also the CSS styles that go with that template:

  - inline in the template HTML
  - by setting **styles** or **styleUrls** metadata
  - with CSS imports

TEMPLATE REFERENCE VARIABLES

- A **template reference variable** is a reference to a DOM element or directive within a template.

- It can be used with native DOM elements but also with Angular 2 components — in fact, it will work with any custom web component.

@INPUT AND @OUTPUT

- Input properties usually receive data values. Output properties expose event producers, such as EventEmitter objects.

```
<hero-detail [hero]="currentHero"
  (deleteRequest)="deleteHero($event)">
</hero-detail>
```

COMPONENT LIFECYCLE

- A Component has a **lifecycle** managed by Angular itself.
- Angular offers **component lifecycle hooks** that give us visibility into component's key moments and the ability to act when they occur.
- Developers can tap into key moments in that lifecycle by implementing Lifecycle Hook **interfaces**.

LIFECYCLE: DIRECTIVES AND COMPONENTS

ngOnInit	Initialize the component after Angular initializes the data-bound input properties.
ngOnChanges	Respond after Angular sets a data-bound input property.
ngDoCheck	Detect and act upon changes that Angular can or won't detect on its own. Called every change detection run.
ngOnDestroy	Cleanup just before Angular destroys the component.

NGONINIT

- To perform complex initializations shortly after construction
- To set up the component after Angular sets the input properties

ONDESTROY

- This is the place to free resources that won't be garbage collected automatically
  - Unsubscribe from observables and DOM events
  - Stop interval timers
  - Unregister all callbacks that this directive registered with global or application services

ONCHANGES

- Detects changes to ***input properties*** of the component (or directive)

- Angular only calls the hook when the value of the input property changes

COMPONENT INTERACTION: FROM PARENT TO CHILD

- Pass data from parent to child with input binding

COMPONENT INTERACTION: FROM PARENT TO CHILD

- Pass data from parent to child with input binding
- Intercept input property changes with a setter

```
@Input()
 set title(title: string) {
   this._title = title ? title.toUpperCase() : ''
```

COMPONENT INTERACTION: FROM PARENT TO CHILD

- Pass data from parent to child with input binding
- Intercept input property changes with a setter

```
@Input()
 set title(title: string) {
   this._title = title ? title.toUpperCase() : ''
```

- Intercept input property changes with *ngOnChanges*

COMPONENT INTERACTION: FROM CHILD TO PARENT

- Parent listens for child event

COMPONENT INTERACTION: FROM CHILD TO PARENT

- Parent listens for child event
- Parent calls a *ViewChild*

COMPONENT INTERACTION: FROM CHILD TO PARENT

- Parent listens for child event
- Parent calls a *ViewChild*
- Parent interacts with child via *local variable*

```
<button (click)="timer.start()">Start</button>
<div class="seconds">{{timer.seconds}}</div>

<countdown-timer #timer></countdown-timer>
```

COMPONENT INTERACTION: FROM CHILD TO PARENT

- Parent listens for child event
- Parent calls a *ViewChild*
- Parent interacts with child via *local variable*

```
<button (click)="timer.start()">Start</button>
<div class="seconds">{{timer.seconds}}</div>

<countdown-timer #timer></countdown-timer>
```

- Parent and children communicate via a service

PIPES & DIRECTIVES (BUILT-IN)

WHAT ARE PIPES ON ANGULAR 2?

- Pipes allow us to change data inside of a template

JavaScript

```
var now = new Date();
function formatDate(date) {
    var month = date.getMonth() + 1;
    return date.getDate() + '.'
    + (month < 10 ? '0' : '') + month + '.'
    + date.getFullYear();
}
$("#datepicker").val(formatDate(now));
```

Angular2

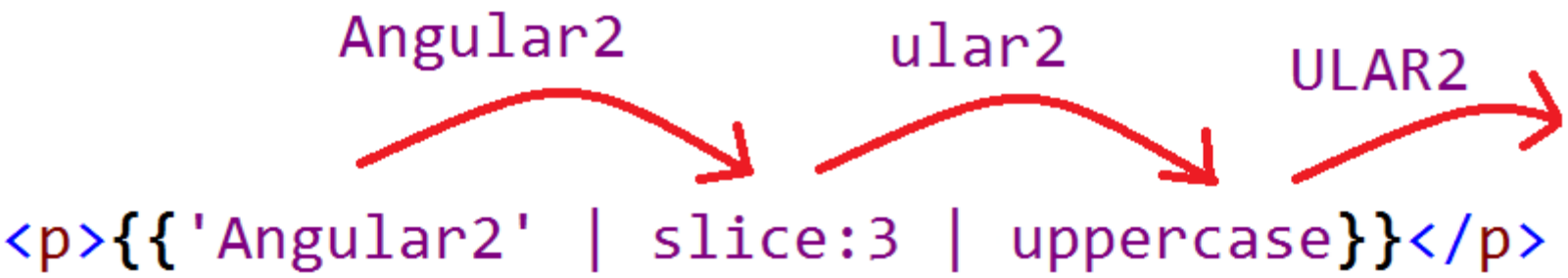
```
{{now | date: 'dd.MM.yyyy'}}
```

USING PIPES

- General syntax(template)

```
<p>{{ expression | pipeName:parameter1:parameter2 }}</p>
```

- Chaining Pipes



BUILT-IN PIPES

- DatePipe

```
curDate: Date = new Date();
```

```
{{expression | date[:format]}}
```

```
<p>output is: {{curDate | date}}</p>
<p>output is: {{curDate | date:'medium'}}</p>
<p>output is: {{curDate | date:'short'}}</p>
<p>output is: {{curDate | date:'mediumTime'}}</p>
<p>output is: {{curDate | date:'shortTime'}}</p>
<p>output is: {{curDate | date:'dd/MM/yyyy'}}</p>
```

→ Jul 26, 2016
→ Jul 26, 2016, 4:35:42 PM
→ 7/26/2016, 4:35 PM
→ 4:35:42 PM
→ 4:35 PM
→ 26/07/2016

BUILT-IN PIPES

- slice
- uppercase
- Lowercase
- json

```
{{object | json}}
```

With JSON pipe:

```
{
  "name": "Ivan",
  "surname": "Ivanov",
  "details": {
    "number": 3,
    "city": "Saratov"
  }
}
```

[STRUCTURAL DIRECTIVES] *NGIF BASICS

1.
```
<div>
  <strong>Never displayed!</strong>
  <span *ngIf="false">Show me! *ngIf="false"</span>
</div>
<div>
  <strong>Always displayed!</strong>
  <span *ngIf="true">
Show me! *ngIf="true"</span>
</div>
```

2. **Never displayed!**

   Always displayed! Show me! *ngIf="true"

```
<div> == $0
  <strong>Never displayed!</strong>
  <!--template bindings={
    "ng-reflect-ng-if": "false"
  }-->
</div>
<div>
  <strong>Always displayed!</strong>
  <!--template bindings={
    "ng-reflect-ng-if": "true"
  }-->
  <span>Show me! *ngIf="true"</span>
```

[STRUCTURAL DIRECTIVES] *NGIF USE FUNCTIONS

1.
```
<div>
  <strong>Displayed if myFunc returns a true value</strong>
  <span *ngIf="getTrueOrFalse(true)">*ngIf="getTrueOrFalse(true)"</
span>
</div>
<div>
  <strong>Displayed if myFunc returns a true value</strong>
  <span *ngIf="getTrueOrFalse(false)">*ngIf="getTrueOrFalse(false)"</
span>
</div>
```

2. **Displayed if getTrueOrFalse returns a true value** *ngIf="getTrueOrFalse(true)"

   Displayed if getTrueOrFalse returns a true value

[STRUCTURAL DIRECTIVES] NGSWITCH

1.
```
<div class="container">
    <div *ngIf="name=='Alex'">Hi, Alexey!</div>
    <div *ngIf="name=='Bob'">Hi, Robert!</div>
    <div *ngIf="name=='Chris'">Hi, Christian!</div>
    <div *ngIf="name!='Alex' && name!='Bob' && name!='Chris'">Hi,
unknown friend!</div>
</div>
```

2.
```
<div class="container" [ngSwitch]="name">
    <div *ngSwitchWhen="'Alex'">Hi, Alexey!</div>
    <div *ngSwitchWhen="'Bob'">Hi, Robert!</div>
    <div *ngSwitchWhen="'Chris'">Hi, Christian!</div>
    <div *ngSwitchDefault>Hi, unknown friend!</div>
</div>
```

[STRUCTURAL DIRECTIVES] *NGFOR GETTING AN INDEX

```
1. <ul>
     <li *ngFor="let user of users; let i = index">
       <strong>{{i + 1}} Name: </strong><span>{{ user.name }}</span>
       <strong>Age: </strong><span>{{user.age}}</span>
     </li>
   </ul>
```

- **1 Name:** Maggie **Age:** 2
- **2 Name:** Lisa **Age:** 6
- **3 Name:** Bart **Age:** 8
- **4 Name:** Homer **Age:** 38
- **5 Name:** Marge **Age:** 36

Variable	Meaning
index	Index of current element (number, starts 0)
first	Is element first (boolean)
last	Is element last (boolean)
even	Is element even (boolean)
odd	Is element add (boolean)

[ATTRIBUTE DIRECTIVES] [NGSTYLE]

```
1. <div [style.background-color]="'yellow'">
       Uses fixed yellow background
   </div>

2. <div [ngStyle]="{color: 'white', 'background-color': 'blue'}">
       Uses fixed white text on blue background
   </div>
```

Uses fixed yellow background

Uses fixed white text on blue background

```
<div style="background-color: yellow;">Highlight demo</div>
<div style="color: white; background-color: blue;" ng-reflect-raw-style="[object Object]">Simple highlight</div>
```

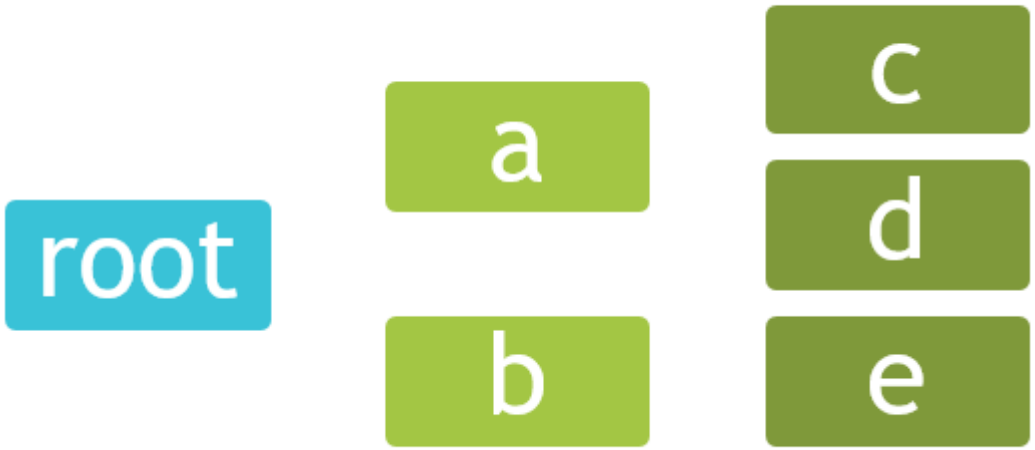
[ATTRIBUTE DIRECTIVES] [NGCLASS]

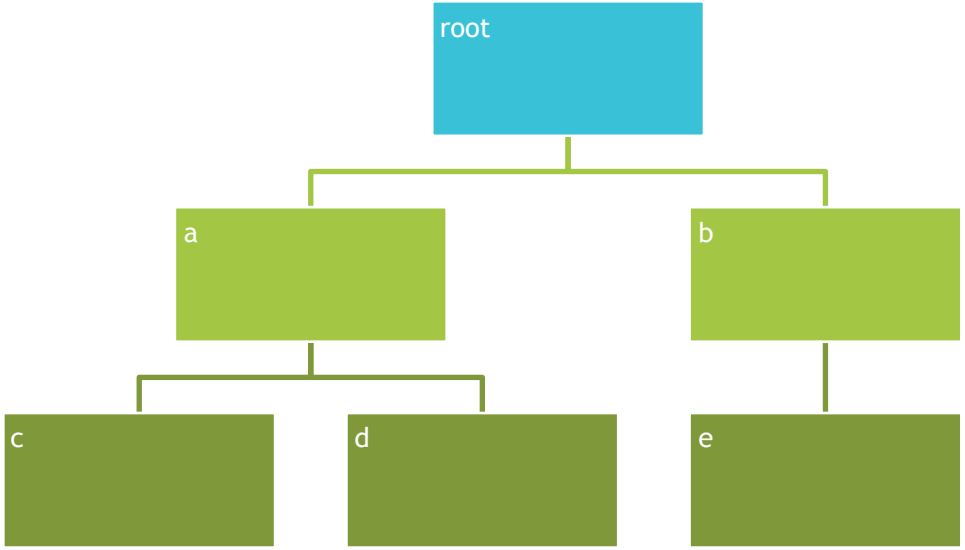
```
1. <div [ngClass]="{bordered: false}">This is never bordered</div>

2. <div [ngClass]="{bordered: true}">This is always bordered</div>

3. <div [ngClass]="{bordered: isBordered}">
       Using object literal.
   </div>

4. <div [ngClass]="classesObj">
       Using object var.
   </div>
```

```
<h3 class="bordered" ng-reflect-raw-class="[object Object]">Highlight demo</h3>
```

DI & SERVICES

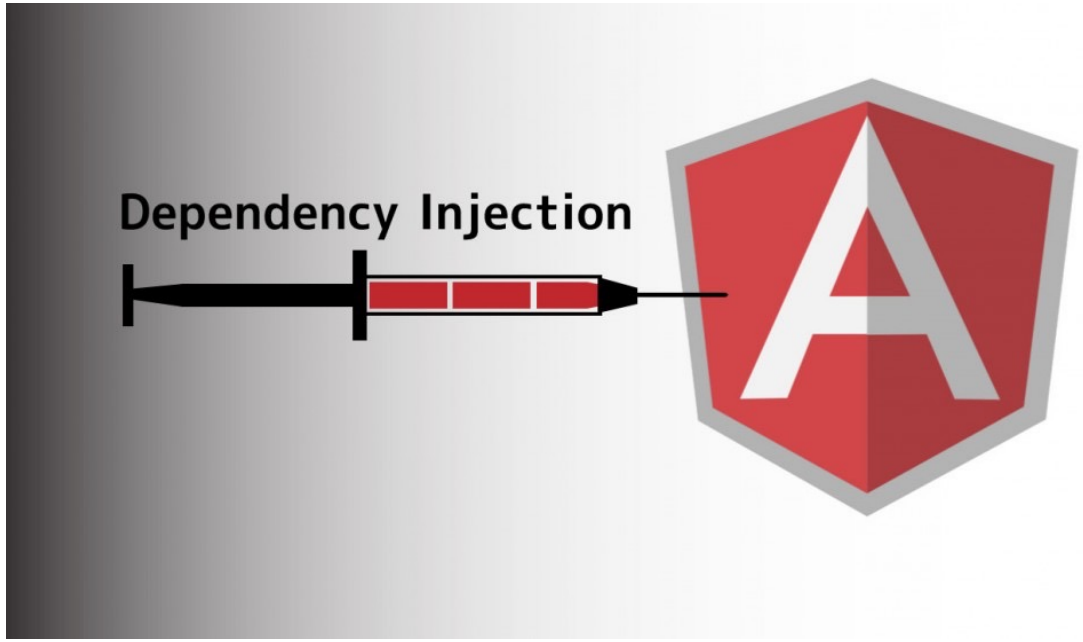
WHAT IS DI?





DEPENDENCY INJECTION OVERVIEW

- Token
- Provider
- Injector
- Dependency



EASY TO DEVELOP

```
1   import {Http} from '@angular/http';
2   import {Injectable} from '@angular/core';
3
4   @Injectable()
5   export class ExampleService {
6       constructor(private _http: Http) {}
7       getData() {
8           return {data:'data example'};
9       }
10  }
```

PROVIDERS

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

PROVIDERS

```
1   import { Component } from '@angular/core';
2   import {ExampleService} from './services/ExampleService';
3
4   @Component({
5       selector: 'my-app',
6       template: '<h1>My First Angular 2 App</h1>'
7   })
8
9   export class AppComponent {
10      constructor(private exampleService: ExampleService) {
11          alert(exampleService.getData().data);
12      }
13  }
```

DI CONFIGURATION

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

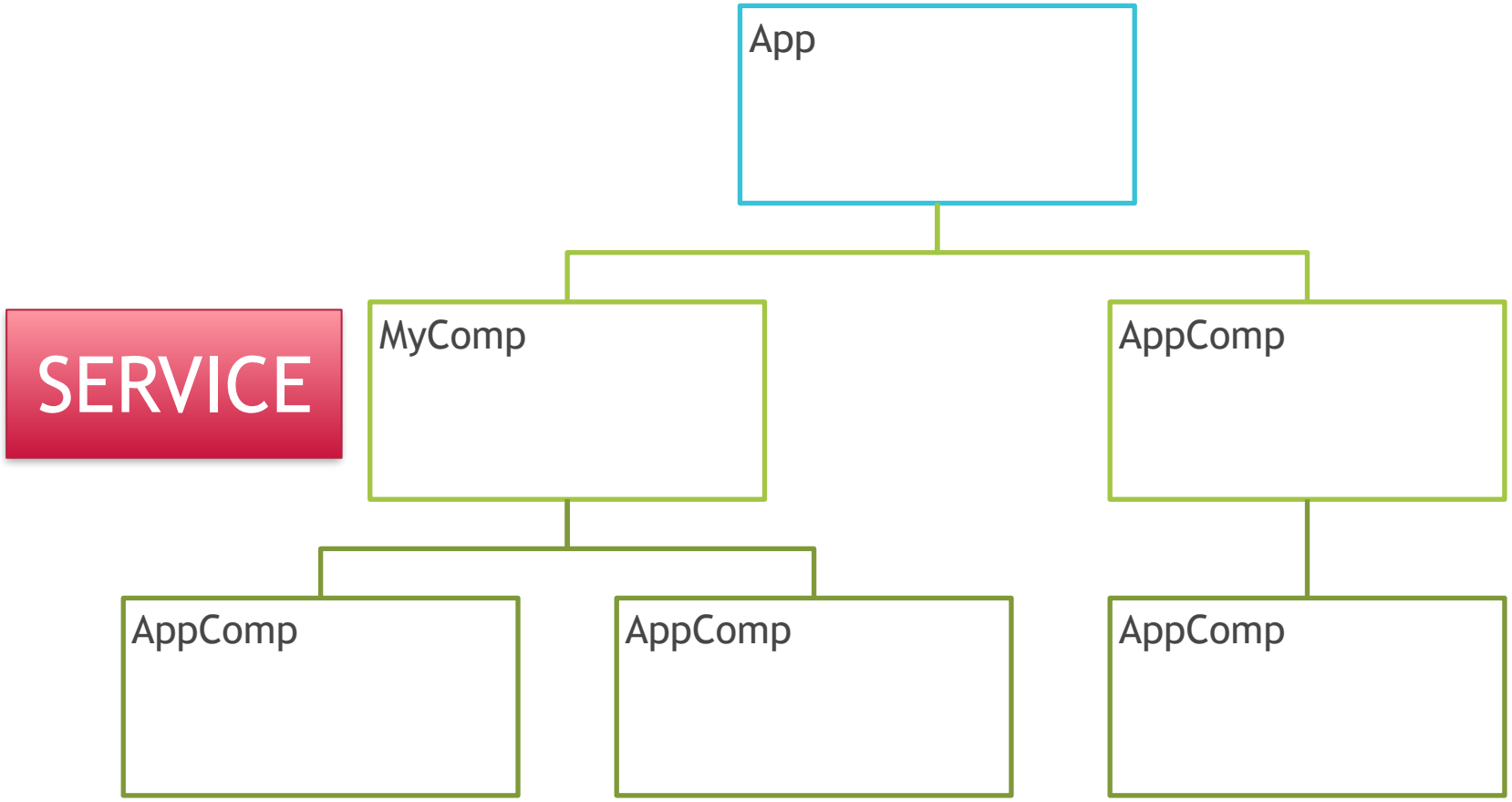
SERVICES

- Provide streams (data models)
- Provide operations with data
- Helpers

NAMING CONVENTION

data.service.ts
some-data.service.ts

SERVICES



ROUTING

USAGE

1. Import it to the application module:

```
import { RouterModule } from '@angular/router'

imports: [ // import Angular's modules
    BrowserModule,
    CommonModule,
    FormsModule,
    ReactiveFormsModule,
    HttpModule,
    RouterModule.forRoot(ROUTES, { useHash: true })
```

2. Define routes:

```
export const ROUTES: Routes = [
    { path: '', component: Home },
    { path: 'home', component: Home }];
```

3. Inject Router to component and use!

```
import {Router} from '@angular/router';

export class Home {

    constructor(private router: Router) {

    }

    ngOnInit() {
        this.router.
    }

}
```

```
config                    (property) Router.config: Route[]
createUrlTree
dispose
errorHandler
events
initialNavigation
isActive
navigate
navigateByUrl
```

OUTLET

```
<nav>
    <h1>Angular Routing</h1>
</nav>

<main>
    <router-outlet></router-outlet>
</main>

<footer></footer>
```

BASIC NAVIGATION

From HTML:

```
<a routerLink="./pageOne" [routerLinkActive]="'active'">Page one</a>
<a [routerLink]="['./pageWithParams', 10, 'video']" [routerLinkActive]="'active'">Page with params</a>
```

From controller:

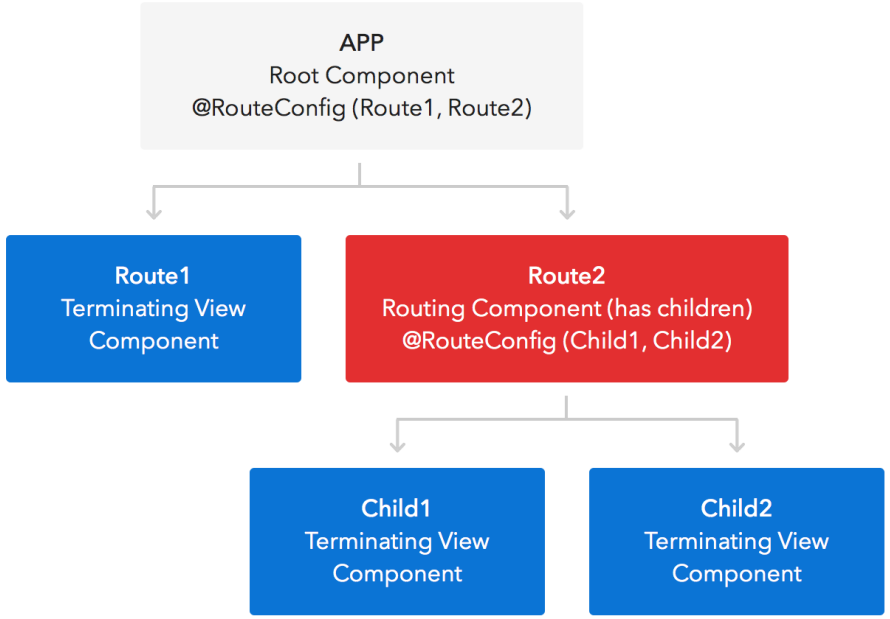
```
export class Home {

    constructor(private router: Router) {

    }

    ngOnInit() {
        this.router.navigate(['./url', ...params]);
    }

}
```

Base href:

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

CHILD ROUTES

```
{
    path: 'pageWithChilds',
    component: PageWithChilds,
    children: [
        {
            path: 'firstChild',
            component: FirstChild
        },
        {
            path: 'secondChild',
            component: SecondChild
        }
    ]
},
```




```
<div class="header">
    <h1>With Childs</h1>
    <div class="router-links">
        <a [routerLink]="['./firstChild']" [routerLinkActive]="'active'">First Child</a>
        <a [routerLink]="['./secondChild']" [routerLinkActive]="'active'">Second Child</a>
    </div>
    <router-outlet></router-outlet>
</div>
```

TIME TO PRACTICE!

QUICK START

https://angular.io/guide/quickstart

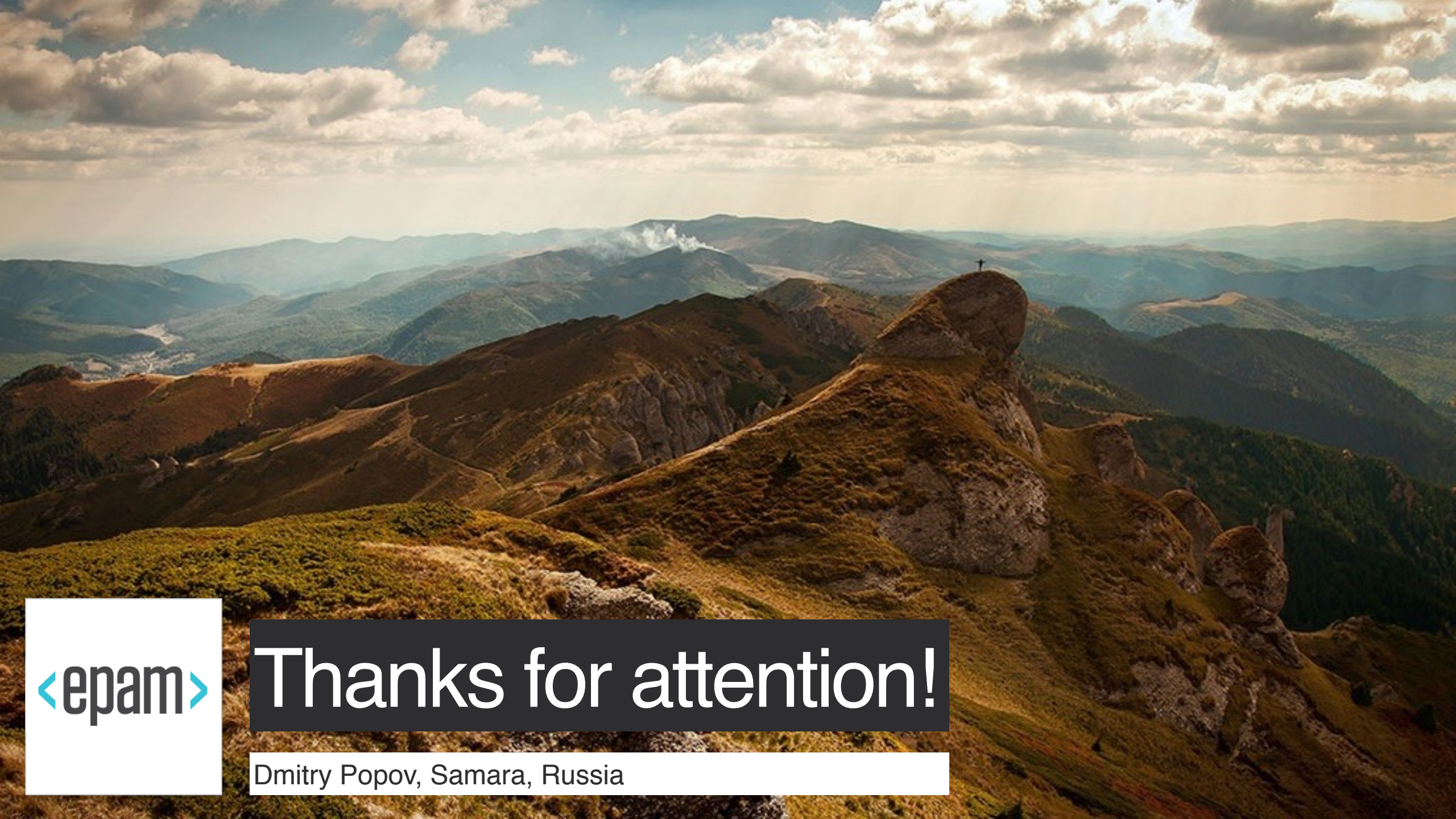
- npm install -g @angular/cli
- ng new netflix --routing true --style scss
  --prefix netflix
- cd netflix
  ng serve --open

LINKS

- https://angular.io/docs/ts/latest/
- https://angular.io/docs/ts/latest/guide/pipes.html
- https://medium.com/front-end-hacking/angular-2-component-lifecycle-hooks-fa5a84b4b64d
- http://blog.angular-university.io/how-does-angular-2-change-detection-really-work/
- https://angular.io/docs/ts/latest/guide/structural-directives.html
- https://angular.io/docs/ts/latest/guide/attribute-directives.html
- https://angular.io/docs/ts/latest/api/core/index/Directive-decorator.html

LINKS

- https://angular.io/docs/ts/latest/guide/dependency-injection.html
- https://angular.io/docs/ts/latest/tutorial/toh-pt4.html



Thanks for attention!

Dmitry Popov, Samara, Russia