Angular 2

for Ciklum

by Artem Koziar

Plan

1. **Angular 2 vs React** ***(Holywar)***
2. Angular 2 overview
3. CLI tool for Angular2
4. **TypeScript**
5. **Angular 2 Components**
6. Template syntax
7. Observables
8. Forms (Template-Driven & Reactive-Driven)
9. Q

Angular 2 !== Angular 1

Angular 3

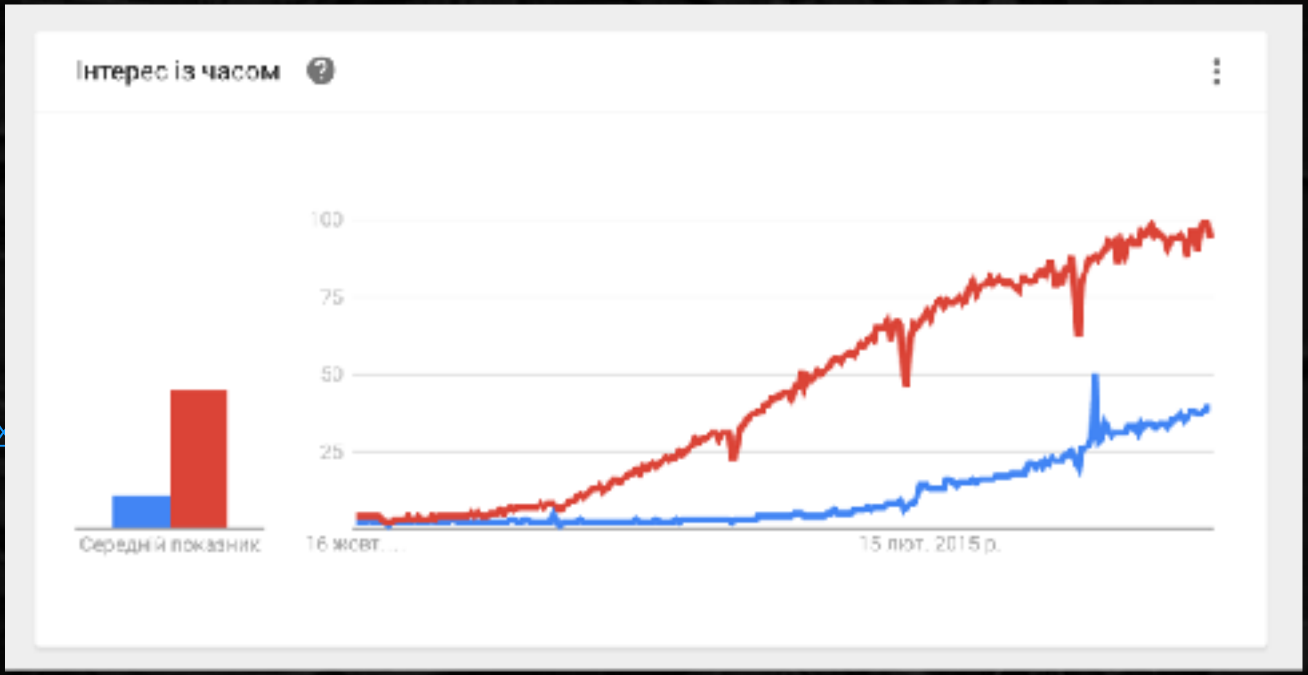
Angular 4

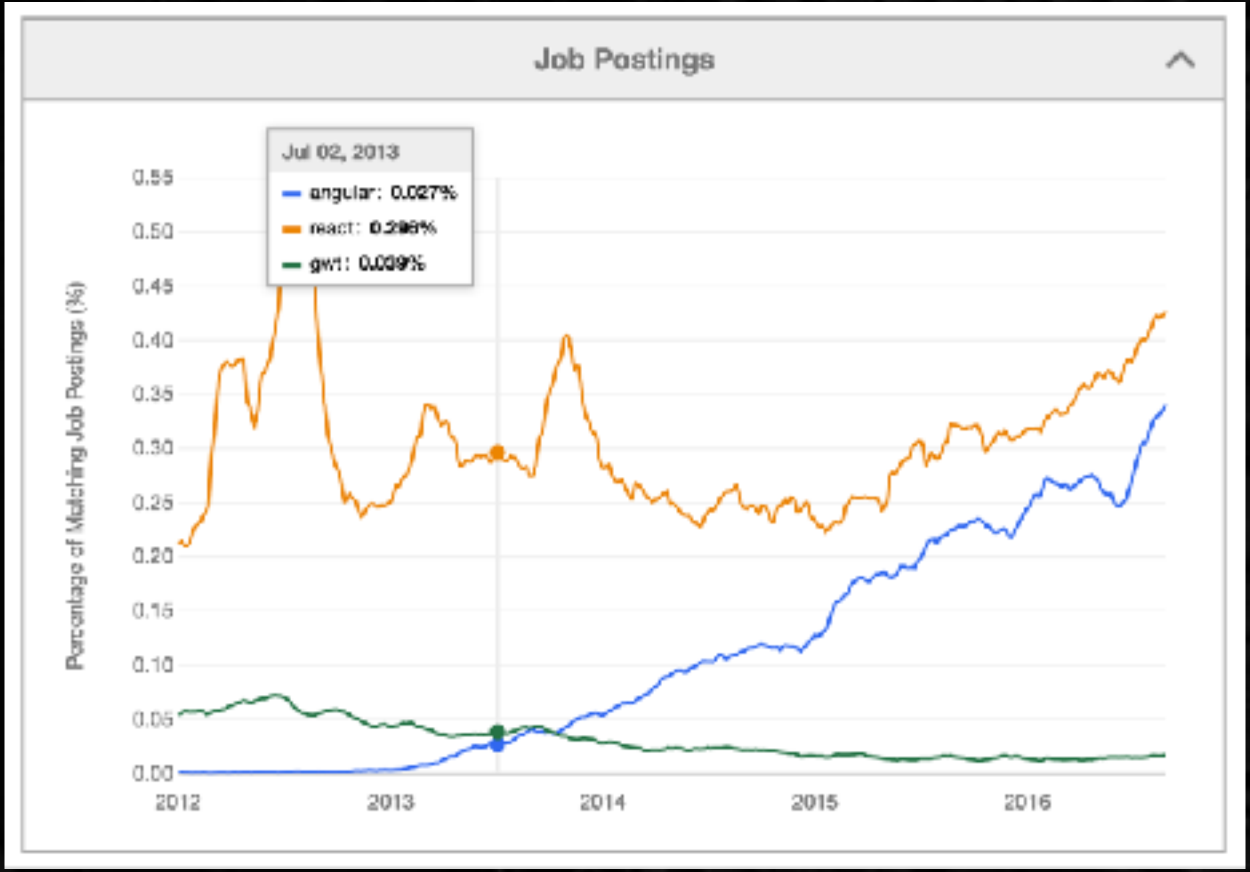
...

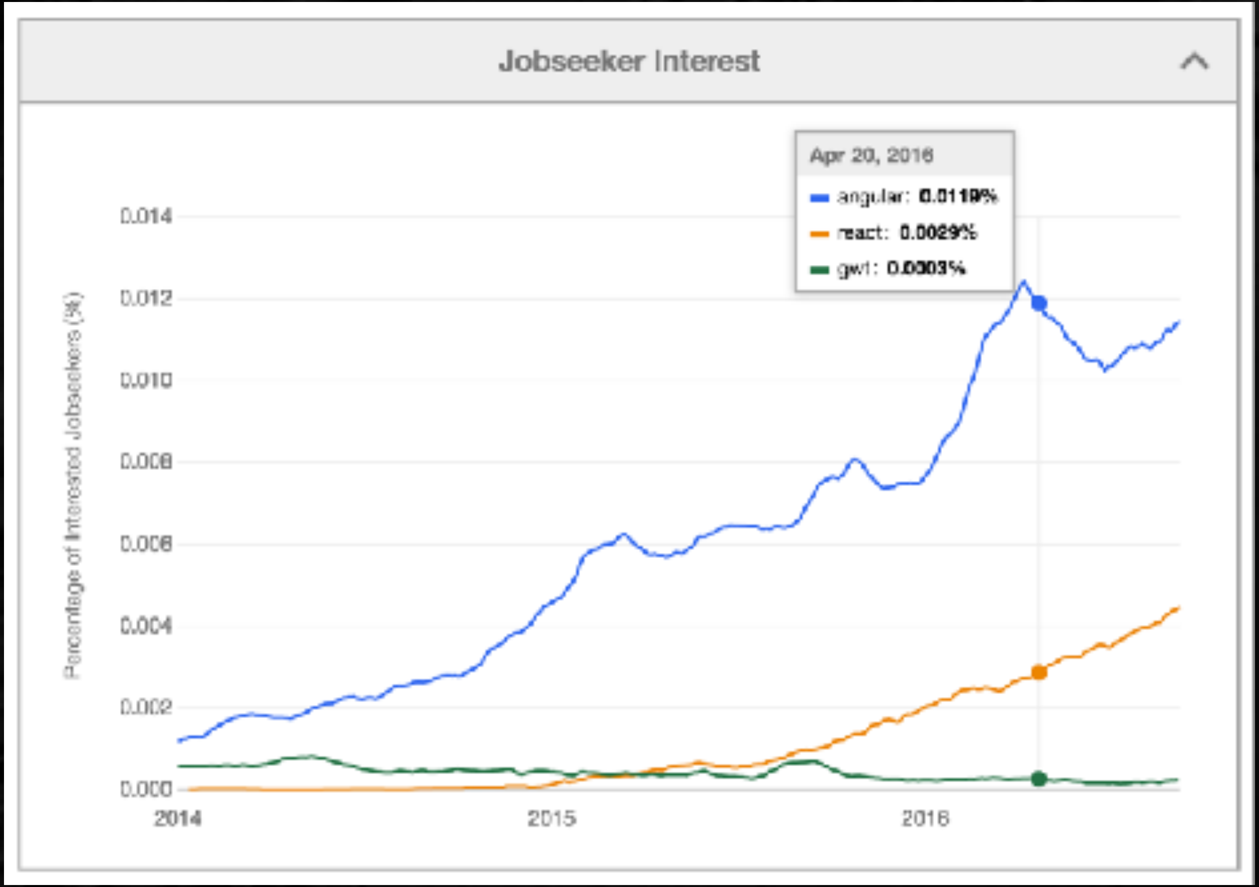
AngularJS vs ReactJS

Angular 1 — 53.5k stars
Angular 2 — 18.3k stars
React — 54.6k stars

https://www.google.com.ua/trends/explore?q=%2Fm%2F012l1vx
http://www.indeed.com/jobtrends/q-angular-q-react-q-gwt.html







Angular 2 vs React

Angular 2
* Simple learning
* TypeScript
* Simple refactoring
* Testing
* Lower price (team)

React
* Customization
* JS->HTML
* Quick development
* Team of professionals

Angular 2

Google

15 Sep 2016 — Final Release

22 Oct 2016 — KRVR Release!

Angular 2: Links

- https://angular.io/
- https://github.com/angular/angular/
- https://angular.io/styleguide
- https://github.com/angular/angular-cli

Angular 2: Overview

1. TypeScript and Decorators
2. Components
3. Observables
4. Dependency injection
5. Routing
6. Change detection strategies
7. (Forms)

CLI tool for Angular2

```
> npm install -g angular-cli
> ng --help

> ng new PROJECT_NAME
> cd PROJECT_NAME
> ng serve

> ng g component my-new-component
> ng g service my-new-service
> ng g directive my-new-directive
> ng g interface my-new-interface
> ng g enum my-new-enum
```

https://github.com/angular/angular-cli

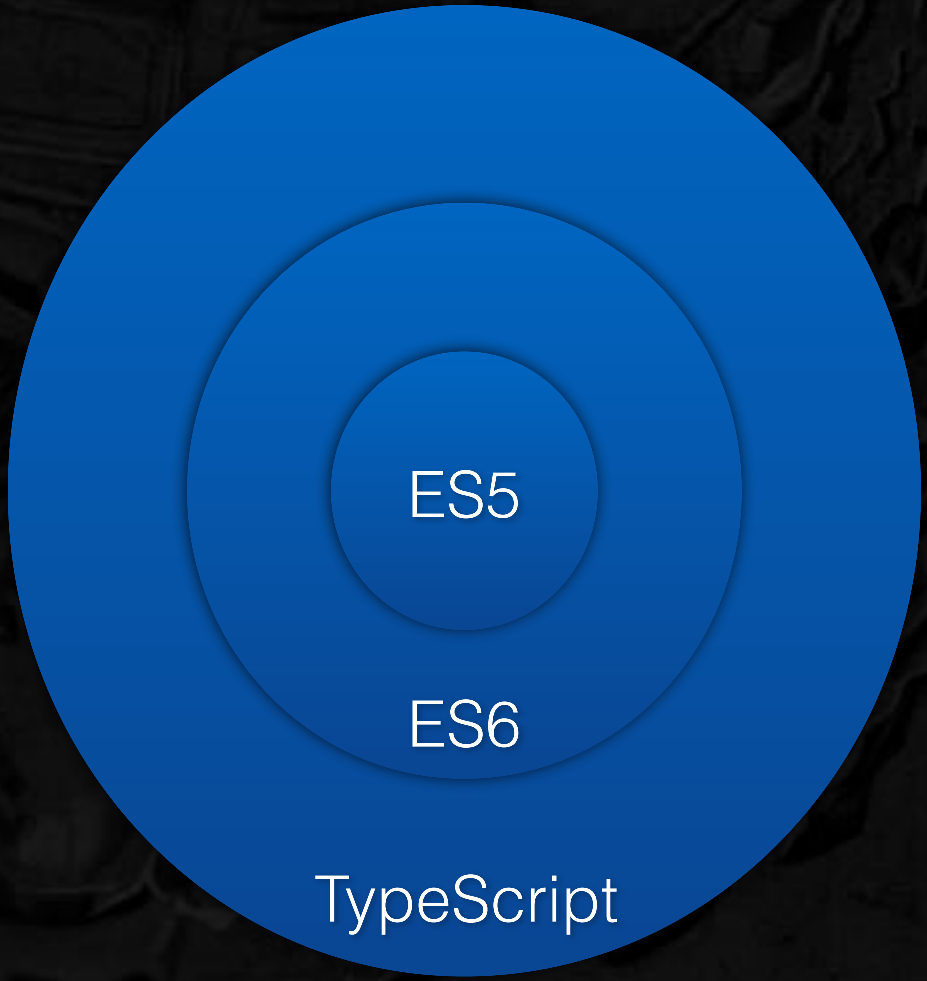
TypeScript

Microsoft

TypeScript

ES6:
— classes
— modules
ES7:
— decorators

TypeScript:
— types
— annotations



```
> npm install -g typescript
```

TypeScript

- JavaScript's types also exist in TypeSctipt
- TypeScript also adds **enum, any** & **void** (like **undefined**)
- **Interface** allows for custom, abstract types
- Function Signatures can be typed Using **Interfaces**
- **Class**es also define types
- *If it walks like a duck, it is a duck, types with the same shapes are compatible*

TypeScript: Simple types

- let a = 123; // Number
- let b: number = 123; // Number

the same other types (boolean, string and objects etc)

- let a1: string[] = []; // Array of Strings (define empty array)
- let a2: string[]; // Array of Strings (undefined)
- let a3: Array<string> // Array of Strings (undefined)
- let a4 = [“a”, “b”]; // Array of Strings

TypeScript: Interface

```
interface User {
  name: string;
}

class UserModel {
  constructor(public name: string, private age?: number) {}
}

let u: User = { name: 'foo' };

u = new UserModel(‘bar');


function useUser(user: UserModel) {
  console.log(user.name);
}
```

TypeScript: Parameters

optional parameter

```
interface User {
  name: string;
}

class UserModel {
  constructor(public name: string,
    private age?: number,
    private city? = 'Kyiv') {}
  }

  let u: User = { name: 'foo' };

  u = new UserModel(‘bar');


function useUser(user: UserModel) {
  console.log(user.name);
}
```

optional parameter
with predefined value

TypeScript: Functions

```
interface CallbackForUser {
  (userName: string,
   age: number): number;
}

class UserModel {
  constructor(
        public name: string,
        private age?: number,
private city? = 'Kyiv') {}

doSome(cb:CallbackForUser)
{
      cb(this.name,this.age);
   }
}

let u: UserModel = new
UserModel('bar');
```

```
// 1
u.doSome((name: string,
age: number) => {
   console.log(`User ${name}
is ${age} years old`);
   return age * 100;
});

// 2

let cb: CallbackForUser =
(name: string, age: number)
=> {
   console.log(`User ${name}
is ${age} years old`);
   return age * 100;
};
u.doSome(cb);
```

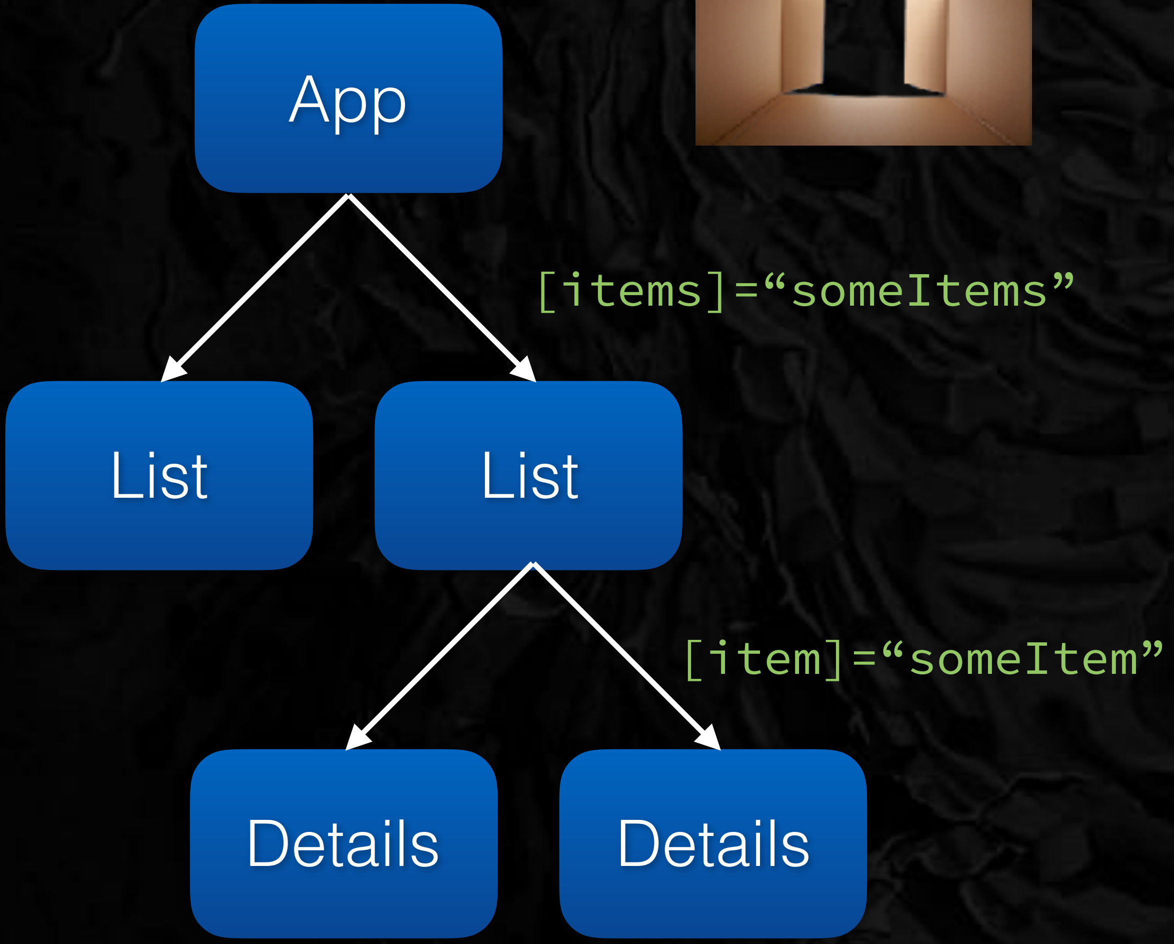
Angular 2 Components

Component

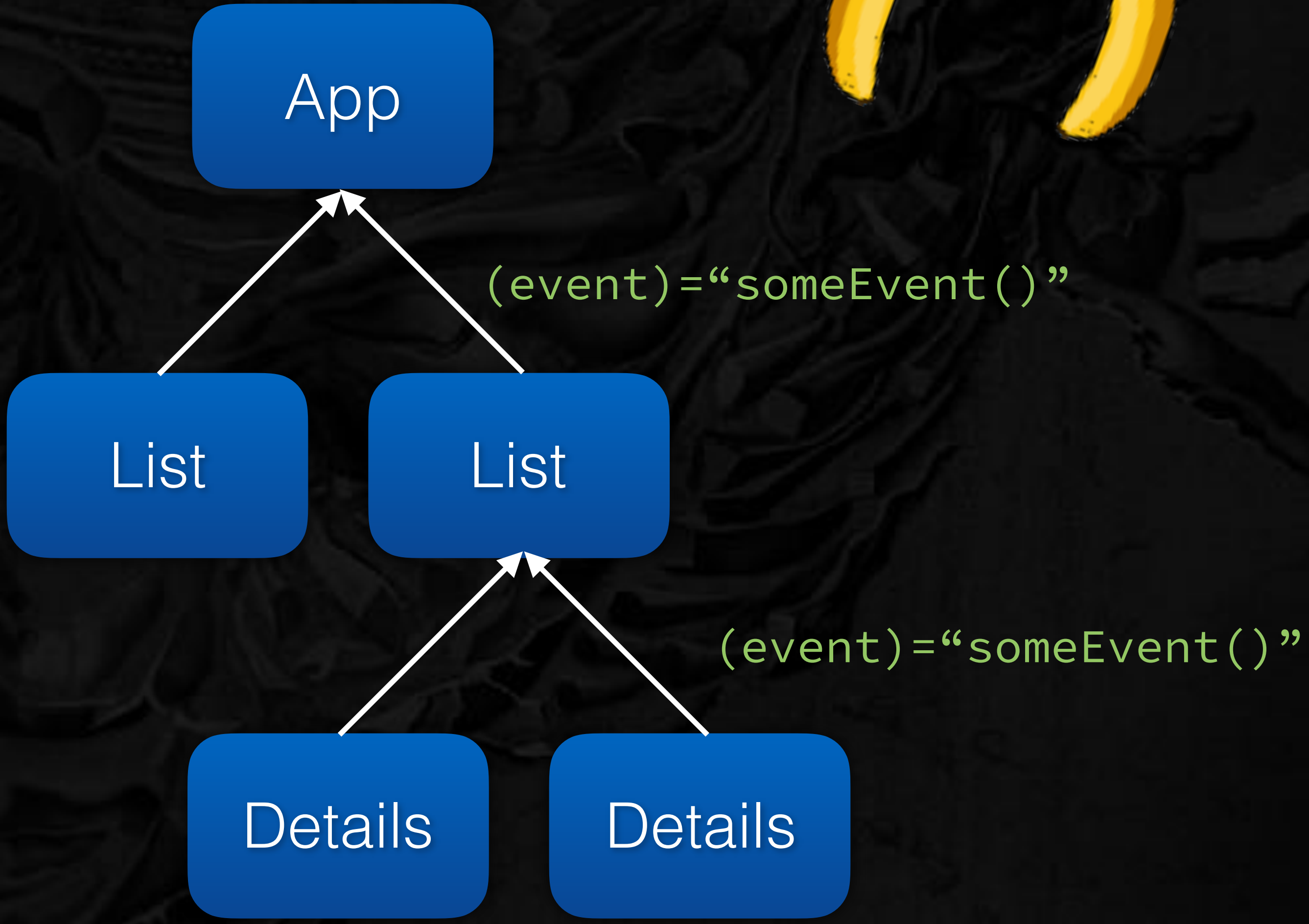
```
<app-hello-world>

  <app-header></app-header>

  <app-user-list>
    <app-user-item></app-user-item>
    <app-user-item></app-user-item>
    <app-user-item></app-user-item>
    <app-user-item></app-user-item>
  </app-user-list>

  <app-add-user-form></app-add-user-form>

  <app-footer></app-footer>

</app-hello-world>
```

Data Binding

Data
[Parent->Child]



Event
(Child->Parent)



Two-Way Data Binding

Combines the input and output binding into single notation using the **ngModel** directive.

```
<input [(ngModel)]=“user.name">
```

[()] = BANANA IN A BOX

https://angular.io/docs/ts/latest/guide/template-syntax.html

= “someValue"

Template-Driven Forms: ngModelChange

```
<input [(ngModel)]="user.name">
<input [ngModel]="user.name"` (ngModelChange)="user.name = $event">
```

+

Decorators

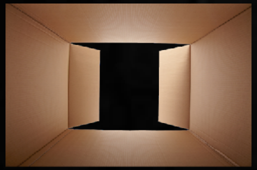
- Decorators — functions that operate on a “target”
- “Target” are classes, methods, properties and parameters
- Decorators invoked with leading **@** like **@Component()**
- Angular 2 decorators always use training brackets, like **@Inject()**
- Decorators do not get follower by **;**

Component

```
// > ng g component hello-world

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

@Component({
  selector: 'app-hello-world',
  template: ‘<p>Hello, {{ title }}</p>’
})
export class HelloWorldComponent {

  title: string;

  constructor() {
    this.title = 'World';
  }

}

// <app-hello-world></app-hello-world>
```

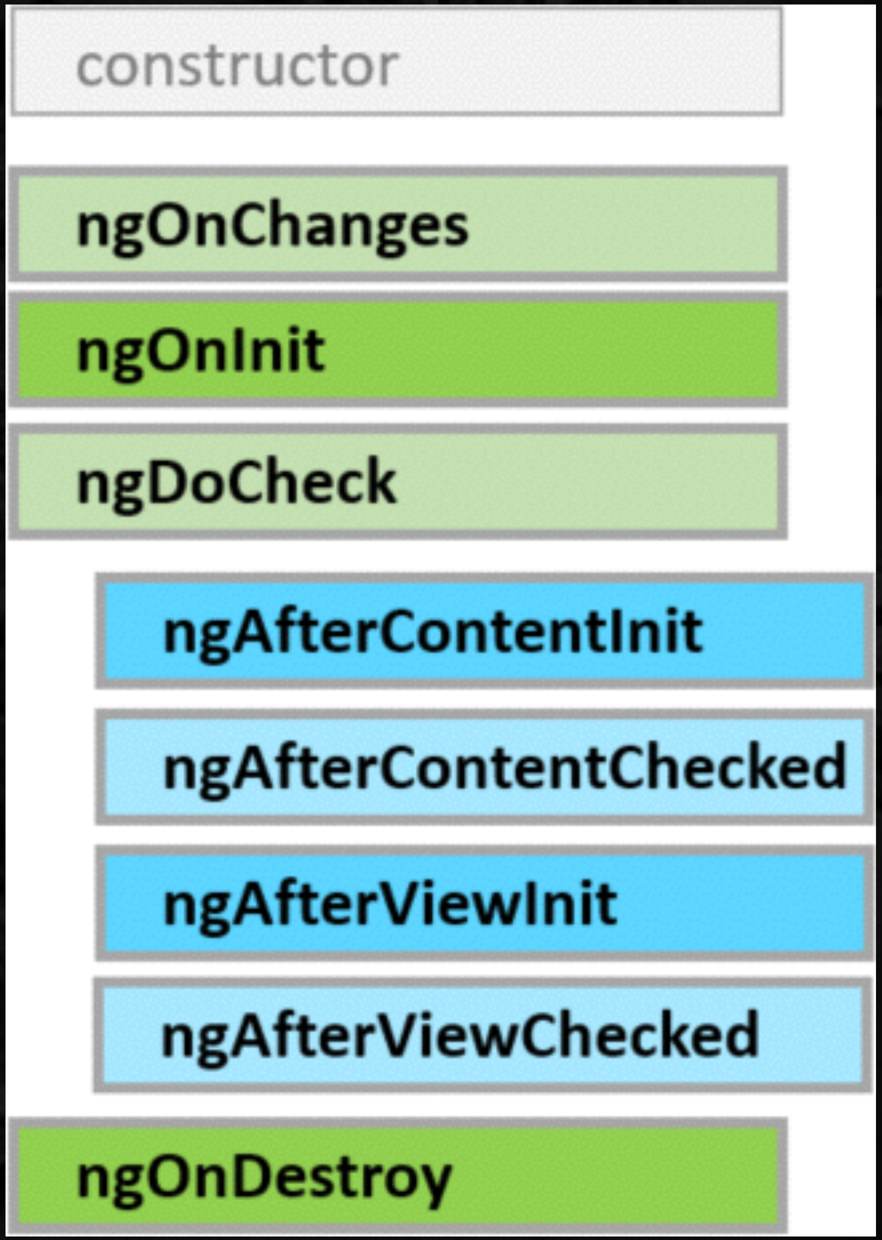
Component: Input/Output

```
// <app-counter [title]="someTitle" (result)="onResult()"></app-counter>

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

@Component({
  selector: 'app-counter',
  template: `
    <div>
      <h2>{{ title }}</h2>
      <span>{{ counter }}</span>
      <p>
        <button (click)="inc()" class="inc">inc</button>
      </p>
    </div>
`
})
export class CounterComponent {

  @Input()
  title: string = '';

  counter: number = 0;

  @Output()
  result: EventEmitter<number> = new EventEmitter();

  inc() {
    this.counter++;
    this.result.emit(this.counter);
  }

}
```



Component Lifecycle hooks

```
import { Component, Input, OnInit } from '@angular/core';
import { TodoService } from '../shared/todo.service';
@Component({
  selector: 'app-hello-world',
  template: `<p>Hello, {{ title }}!</p>`
})
export class HelloWorldComponent implements OnInit {

  @Input()
  title: string;

  constructor(private todoService: TodoService) {
    // if (!this.title) {
    //   this.title = 'World';
    // }
    console.log('constructor', this.title); // undefined
  }

  ngOnInit() {
    if (!this.title) {
      this.title = 'World';
    }
    console.log('ngOnInit', this.title); // World
  }

}
```



Template Syntax

```
<ul>
  <li *ngFor="let menu of menus" [ngClass]="{'active': menu === activeMenu}”>
    <a routerLink="{{menu.link}}" (click)=“onClick(menu)">{{menu.title}}</a>
  </li>
</ul>

<div *ngIf="currentHero">Hello, {{currentHero.firstName}}</div>

<div [class.hidden]="isSpecial">Hide with class</div>
<div [style.display]="isSpecial ? 'block' : 'none'">Show with style</div>
<div [ngClass]="{'first': true, 'second': true, 'third': false}">...</div>

<select [(ngModel)]="employee.manager" (ngModelChange)="change($event)">
  <option *ngFor="let manager of managers" [ngValue]="manager">{{ manager.name }}</
option>
</select>

<span [ngSwitch]="toeChoice">
  <span *ngSwitchCase="'Eenie'">Eenie</span>
  <span *ngSwitchCase="'Meanie'">Meanie</span>
  <span *ngSwitchCase="'Miney'">Miney</span>
  <span *ngSwitchCase="'Moe'">Moe</span>
  <span *ngSwitchDefault>other</span>
</span>
```

https://angular.io/docs/ts/latest/guide/template-syntax.html

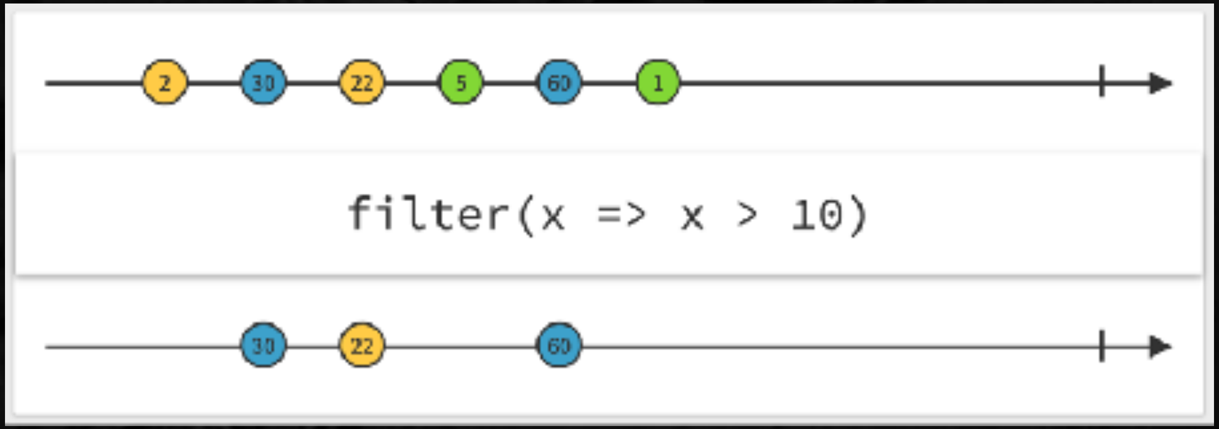
Observables (ES7)

- Observables open up a continuous channel of communication in which multiple values of data can be emitted over time.
- From this we get a pattern of dealing with data by using array-like operations to parse, modify and maintain data.
- Angular 2 uses observables extensively - you'll see them in the HTTP service and the event system.

http://rxmarbles.com/#filter

Observables: Example

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

import { Http } from '@angular/http';
import './rxjs-operators';
import { Observable } from 'rxjs/Observable';

@Component({
  selector: 'app-test'
})
export class TestSomponent {
  constructor(private http: Http) {}

  getHeroes (): Observable<Hero[]> {
    return this.http.get(this.heroesUrl)
      .map(res => res.json())
      .filter(data => data.age > 18)
      .subscribe((data) => {
        this.data = data;
      });
  }
}
```



Angular 2 Form

Angular 2 Form

1. Template-Driven Forms (HTML->JS)
2. Reactive Forms (JS->HTML)

Angular 2 Form: @NgModule

```
// signup.interface.ts
export interface User {
  name: string;
  account: {
    email: string;
    confirm: string;
  }
}
```

```
// Init

import {
  FormsModule,
  ReactiveFormsModule
} from '@angular/forms';

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

Template-Driven Forms: Input

```
<form novalidate #f="ngForm">
  ...
  <input
    type="text"
    placeholder="Your full name"
    name="name"
    ngModel>
  ...
</form>

{{ f.value | json }} // { name: '' }
```

Template-Driven Forms: ngModel

```
JS Controller: this.user.name = ‘Artem Koziar’;

<form novalidate #f="ngForm">
  ...
  <input
    type="text"
    placeholder="Your full name"
    name="name"
    [(ngModel)]="user.name">
  ...
</form>

{{ user | json }} // { name: ‘Artem Koziar' }
{{ f.value | json }} // { name: 'Artem Koziar' }
```

Template-Driven Forms: ngModelGroup

```
<div ngModelGroup="account">
  <label>
    <span>Email address</span>
    <input
      type="email"
      placeholder="Your email address"
      name="email"
      ngModel>
  </label>
  <label>
    <span>Confirm address</span>
    <input
      type="email"
      placeholder="Confirm your email address"
      name="confirm"
      ngModel>
  </label>
</div>
```

Template-Driven Forms: Submit

```
<form novalidate (ngSubmit)="onSubmit(f)" #f="ngForm">
  ...
<button type="submit" [disabled]=“f.invalid">
  Sign up
</button>

</form>
```

```
onSubmit({ value, valid }: { value: User, valid: boolean }) {
  console.log(value, valid);
}
```

Template-Driven Forms: Error Validation

```
<input
     ...
     #userName="ngModel"
     required>
</label>
<div *ngIf="userName.errors?.required && userName.touched" class="error">
  Name is required
</div>
```

Reactive-Driven Forms

```
this.someControl = new FormControl('');
```

Reactive-Driven Forms

```
user: FormGroup;

ngOnInit() {
  this.user = new FormGroup({
    name: new FormControl(''),
    account: new FormGroup({
      email: new FormControl(''),
      confirm: new FormControl('')
    })
  });
}
```

Reactive-Driven Forms

```
<form novalidate [formGroup]="myGroup">
  Name: <input type="text" formControlName="name">
  Location: <input type="text" formControlName="location">
</form>
```

```
<form novalidate (ngSubmit)="onSubmit(user)" [formGroup]="user">
  ...
</form>
```

Reactive-Driven Forms

```
onSubmit({ value, valid }: { value: User, valid: boolean }) {
  console.log(value, valid);
}
```

```
onSubmit() {
  console.log(this.user.value, this.user.valid);
}
```

Reactive error validation

```
ngOnInit() {
  this.user = new FormGroup({
    name: new FormControl('', [Validators.required, Validators.minLength(2)]),
    account: new FormGroup({
      email: new FormControl('', Validators.required),
      confirm: new FormControl('', Validators.required)
    })
  });
}
```

Reactive-Driven Forms

```
<div
  class="error"
  *ngIf="user.get('name').hasError('required') && user.get('name').touched">
  Name is required
</div>

<div
  class="error"
  *ngIf="user.get('name').hasError('minlength') && user.get('name').touched">
  Minimum of 2 characters
</div>



<button type="submit" [disabled]="user.invalid">Sign up</button>
```

Simplifying with FormBuilder

```
import { FormControl, FormBuilder, FormGroup, Validators } from '@angular/forms';

export class SignupFormComponent implements OnInit {
  user: FormGroup;
  constructor(private fb: FormBuilder) {}
  ngOnInit() {
    this.user = this.fb.group({
      name: ['', [Validators.required, Validators.minLength(2)]],
      account: this.fb.group({
        email: ['', Validators.required],
        confirm: ['', Validators.required]
      })
    });
  }
  onSubmit({ value, valid }: { value: User, valid: boolean }) {
    console.log(value, valid);
  }
}
```

TODO

1. Routing & Navigation
2. Testing
3. Practices, Practices, Practices

Artem Koziar

http://temich.in.ua/

a@temich.in.ua

artko@ciklum.com

Skype: Isharkichl

Self: temich.in.ua/-i/angular2-4ciklum.pdf

TNX