



Enhance Web Development

Angular 2



Presentation

- Qui suis-je ?
- Agenda
- Quelques règles !
- Les outils nécessaires

Qui suis-je ?

Benjamin Longearet

- Développeur Enthousiaste !
- CTO & Co-founder @EliumTV (6 mois)
- Directeur Frontend @Teads (4 ans)
- Première web-app en 1998



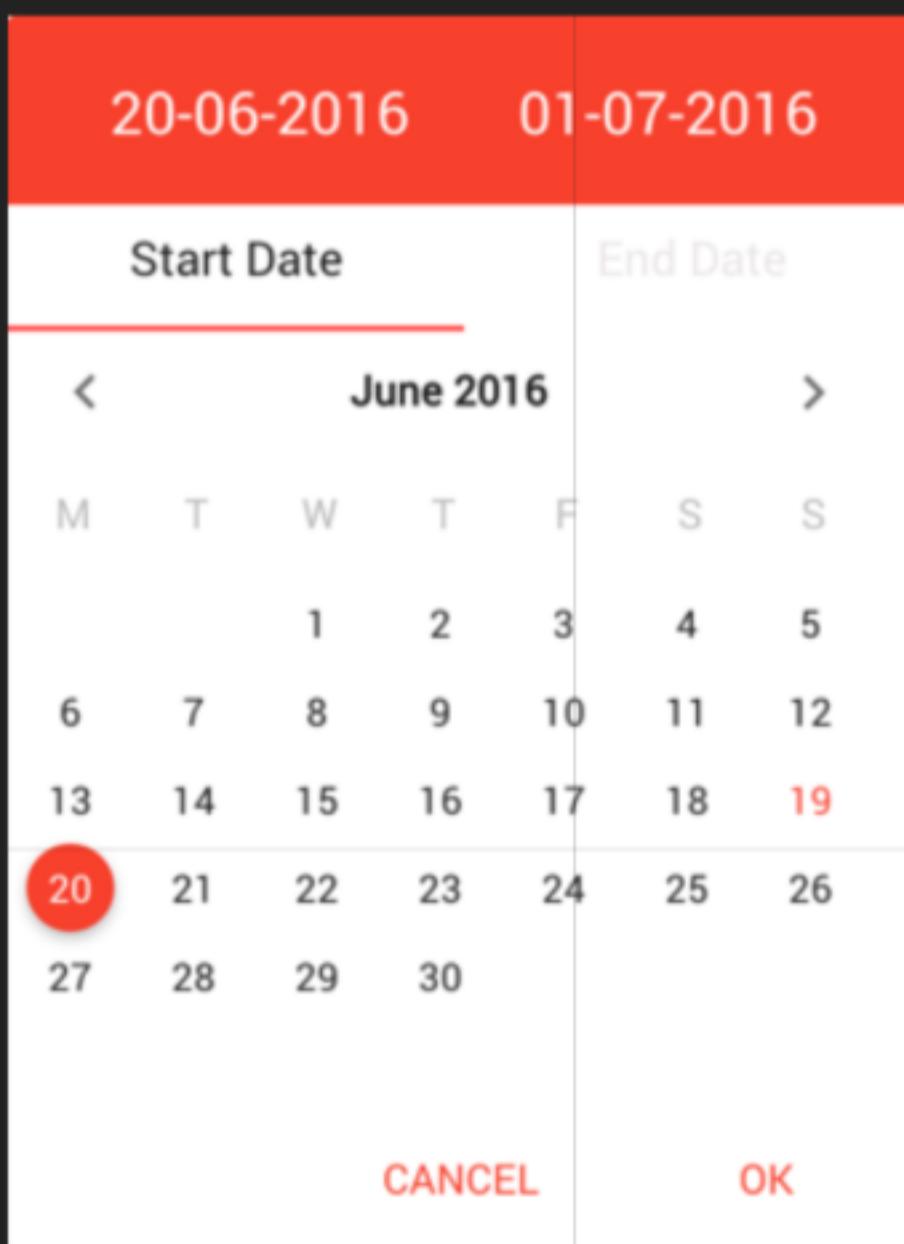
@blongearet



firehist



Agenda



Semaine 1

Jour 1

- Accueil & Agenda
- Pourquoi Angular 2
- Le panorama

Semaine 1

Jour 2

- L'écosystème Web : Webpack, ES5/6, TypeScript, NPM, NodeJS, etc.

Semaine 1

Jour 3

- Les artefacts fondamentaux
- Les templates

Semaine 1

Jour 4

- Les services
- Les formulaires

Semaine 1

Jour 5

- Le routeur
- Comprendre une architecture orientée composant

Semaine 2

Jour 1

- Bien utiliser son navigateur
- Versionner son code source

Semaine 2

Jour 2

- Les directives
- Les pipes

Semaine 2

Jour 3

- Communication avec un serveur
- Firebase : Kezako !

Semaine 2

Jour 4

- Panorama de la programmation Reactive
- Les Observables et RxJS
- Les opérations immutables

Semaine 2

Jour 5

- Les états Reactive et
@ngrx/store

Quelques règles

1. Ne pas hésiter à poser des questions !
2. Ne pas couper la parole (même si on a la réponse :-D)
3. Pause entre 10h30 et 11h00
4. Du travail personnel est nécessaire !

Les outils nécessaire

1. Webstorm, Sublime Text, etc.
2. Terminal, iTerm
3. NodeJS (nvm)
4. NPM (installed with NodeJS)

Pourquoi Angular 2 ?

- Un peu d'histoire
- Standards du web
- Performances
- Environnement "Reactive"
- Open-source & travail d'équipe

Un peu d'histoire



Angular 1

- Première version : 2009
- Créer par Miško Hevery
- 1.3m développeur utilise la version 1.X

Angular 2

- Annoncée en 2014
- RC disponible en mai 2016
- Actuellement : 2.0.0-rc2
- Déjà 360k développeurs
- Utilisation des bonnes pratiques AngularJS 1.X

Ok et ? sinon pourquoi Angular 2 ?

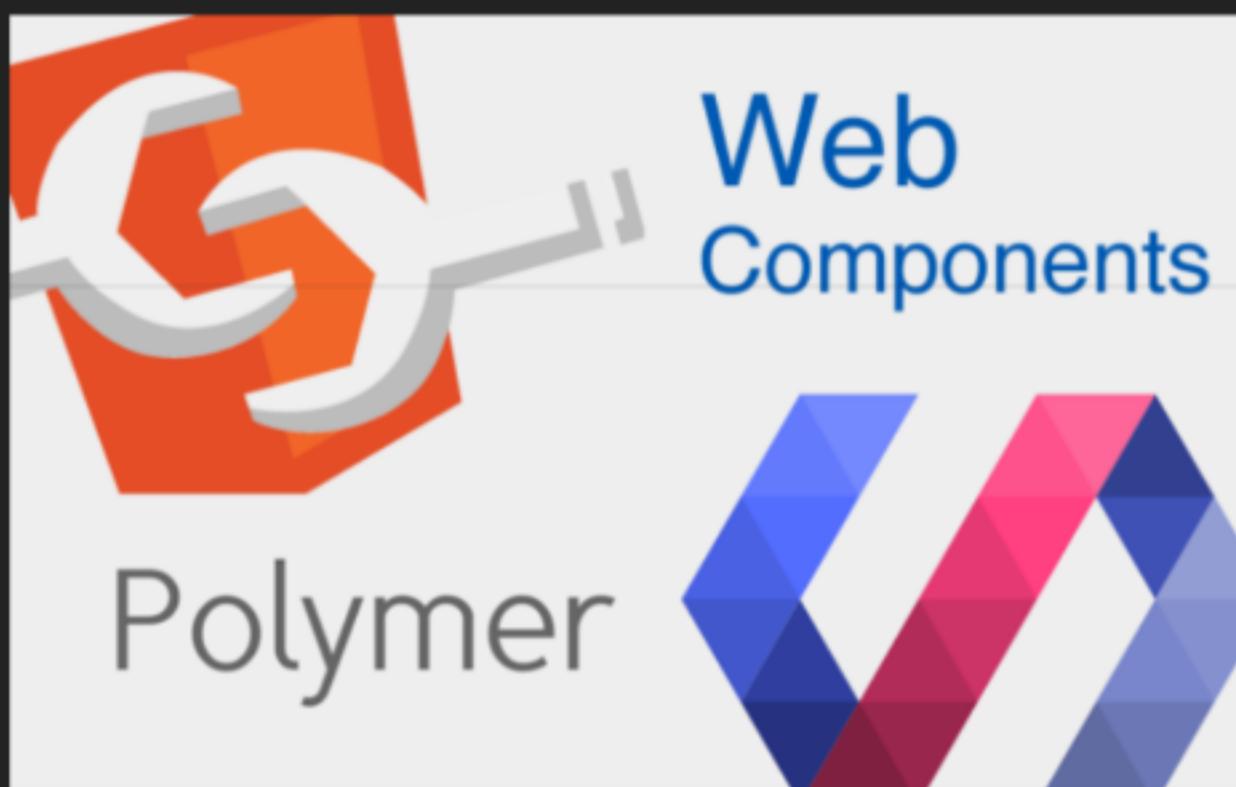
Standard du web



HTML



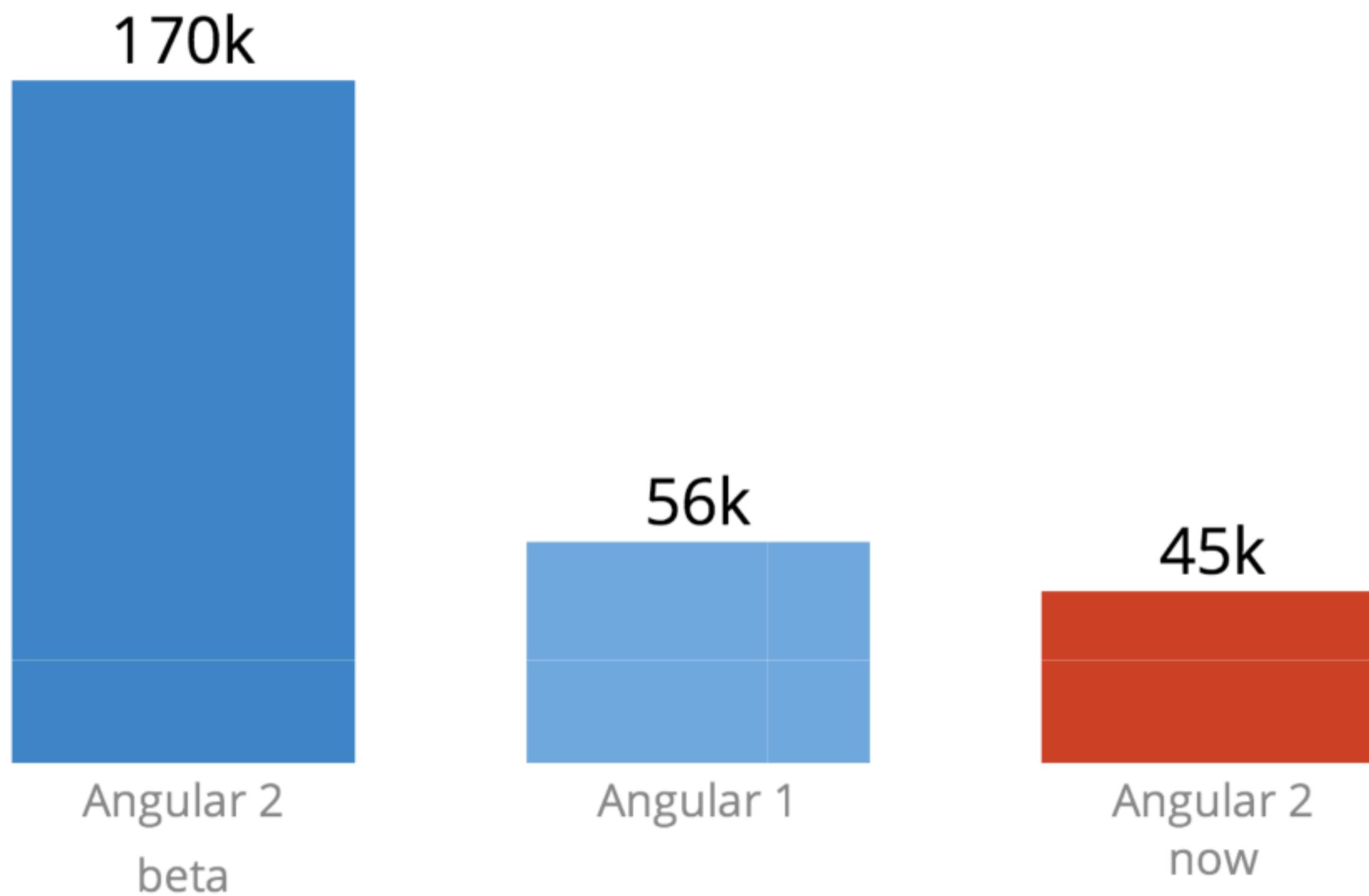
CSS



Performances



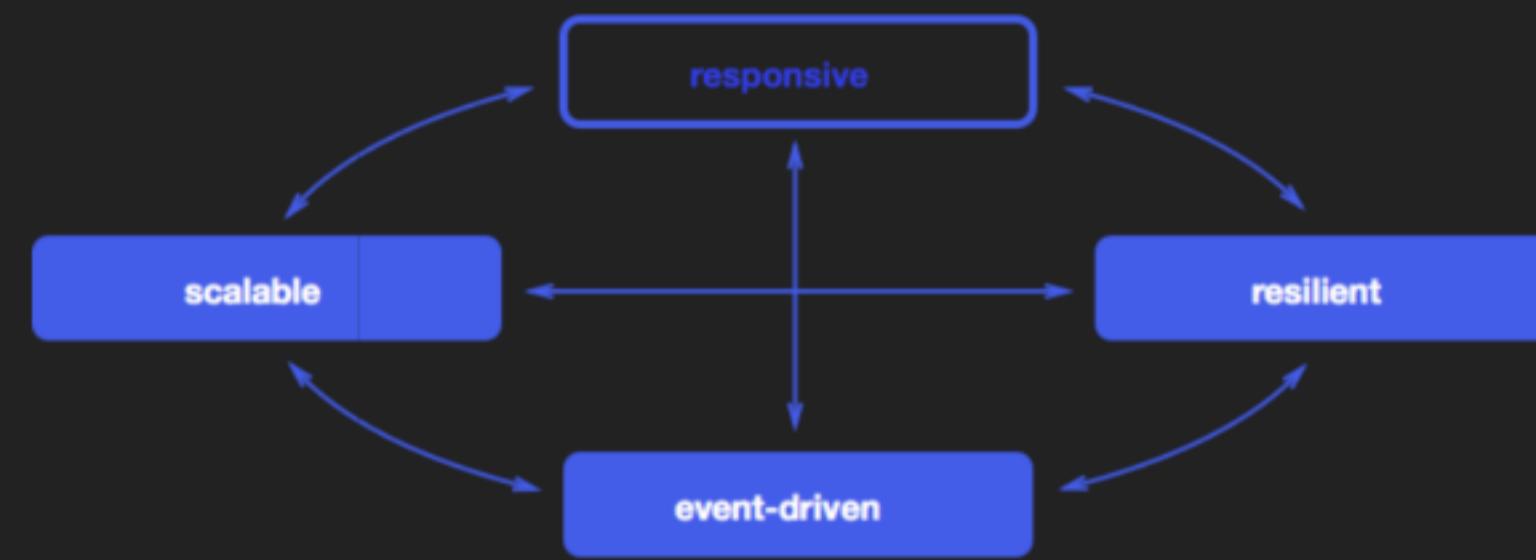
Small is beautiful



Fast is beautiful

- Lazy-loading
- Rendu très rapide (vs Angular 1)
 - 2,5x plus rapide la première fois
 - 4,2x plus rapide lors d'une mise à jour

Environnement "Reactive"



Open-source & travail d'équipe



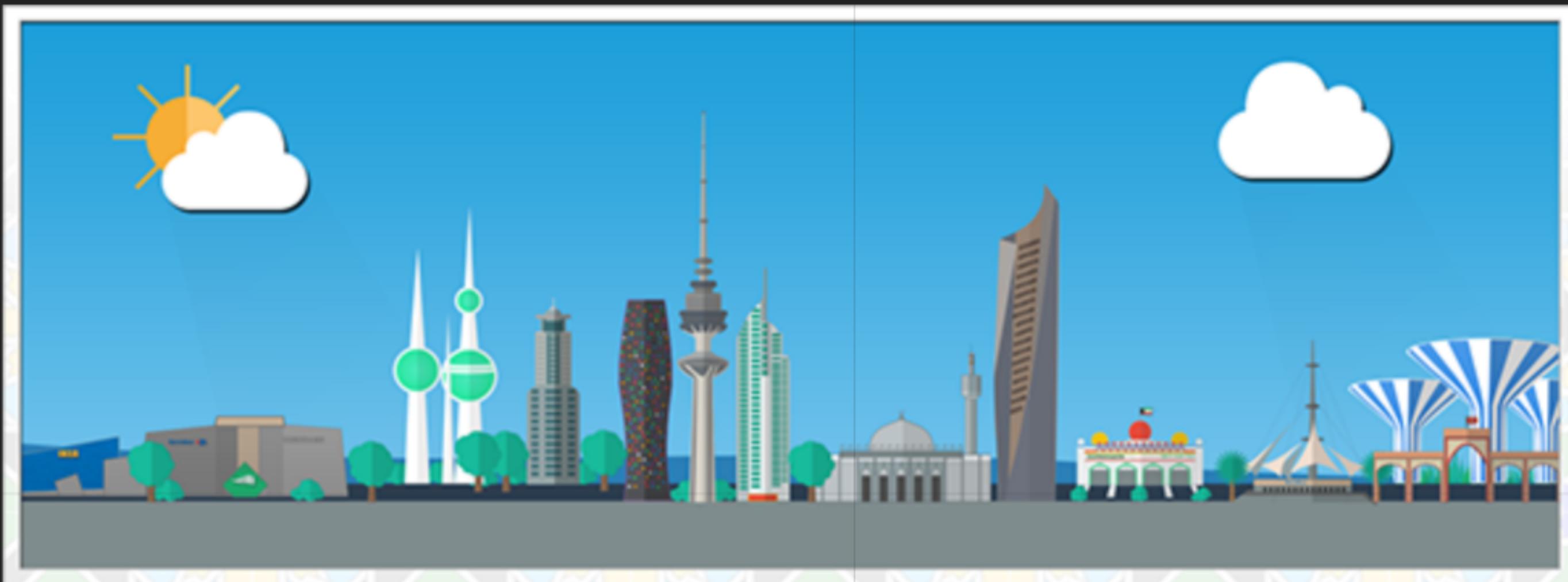
ionic

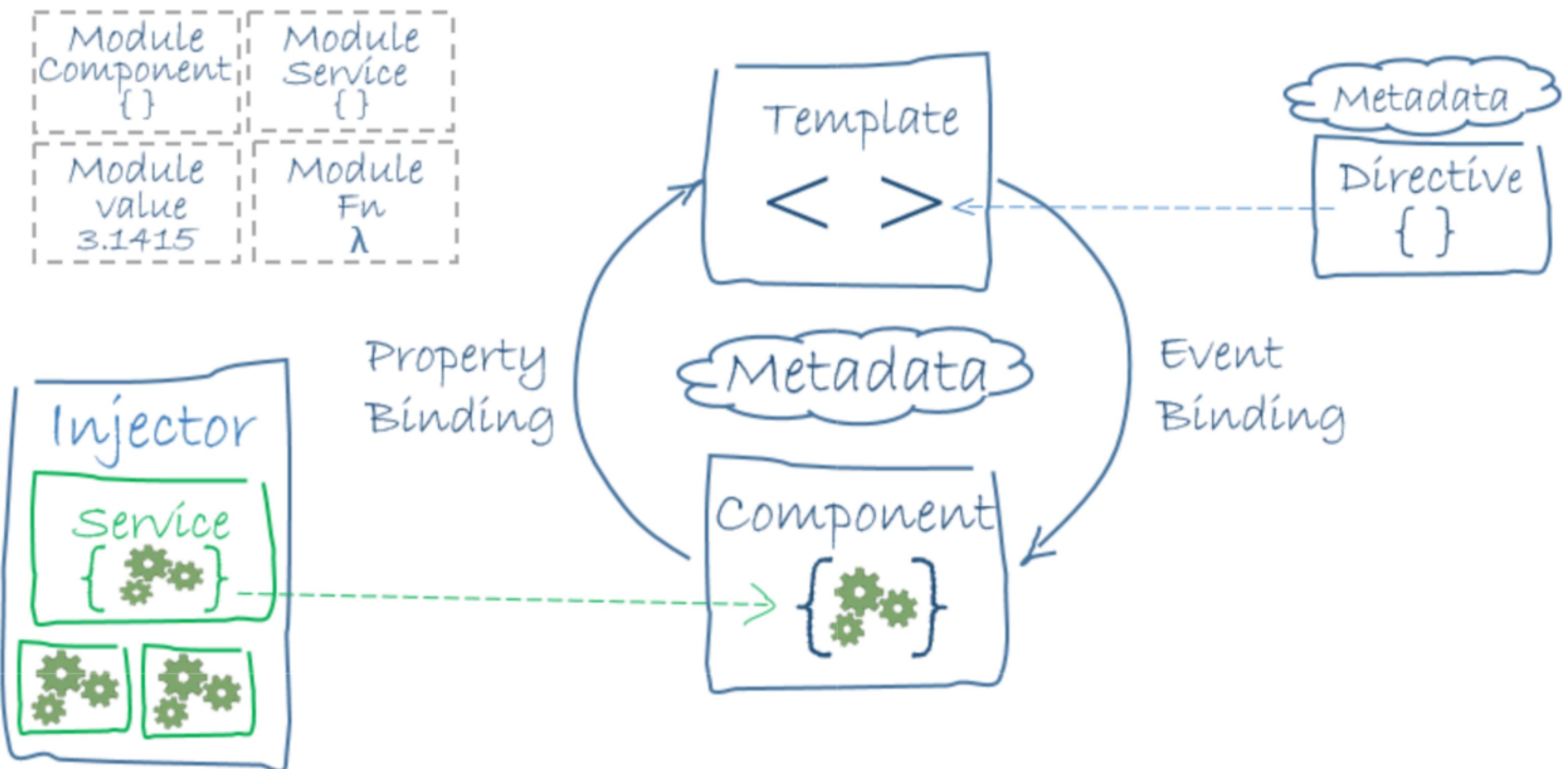


Panorama

- Les principaux artefacts
- Comment démarrer une application
- Et plus en détail !?

Les principaux artefacts





- Module
- Composant
- Metadata
- Template
- Data Binding
- Service
- Directives
- Dependency Injec

Comment démarrer une application

- Importer le module **bootstrap**
- Importer votre composant
- Importer vos dépendances
- Appeler la méthode **bootstrap** en lui donnant le composant et les dépendances

```
import {bootstrap} from 'angular2/platform/browser';
import {ROUTER_PROVIDERS} from 'angular2/router';
import {AppComponent} from './app.component';

bootstrap(AppComponent, [
    ROUTER_PROVIDERS
]);
```





Et plus en détail

Module

- Utilise la syntaxe ES6
- Angular 2 utilise les modules pour découper leur code source
- Les modules exportent des bouts de code que d'autres modules peuvent importer
- **TIPS : Garder les modules léger et spécialisés**

```
// In home.component.ts
export class HomeComponent { }

// In app.component.ts
import {HomeComponent} from './home/home.component';
```

Module

Composant

- Les composants sont des classes ES6
- Les providers sont injectés dans le constructeur
- Déclaration explicite des providers & directives
- Possibilité d'intercepter tous les évènements
- Les propriétés et méthodes du composant sont disponibles dans la vue

```
export class HomeComponent implements OnInit{
  title: string = 'Home Page';
  body: string = 'This is the about home body';
  message: string;

  constructor(private _stateService: StateService) { }

  ngOnInit() {
    this.message = this._stateService.getMessage();
  }

  updateMessage(m: string): void {
    this._stateService.setMessage(m);
  }
}
```



Composant

Metadata

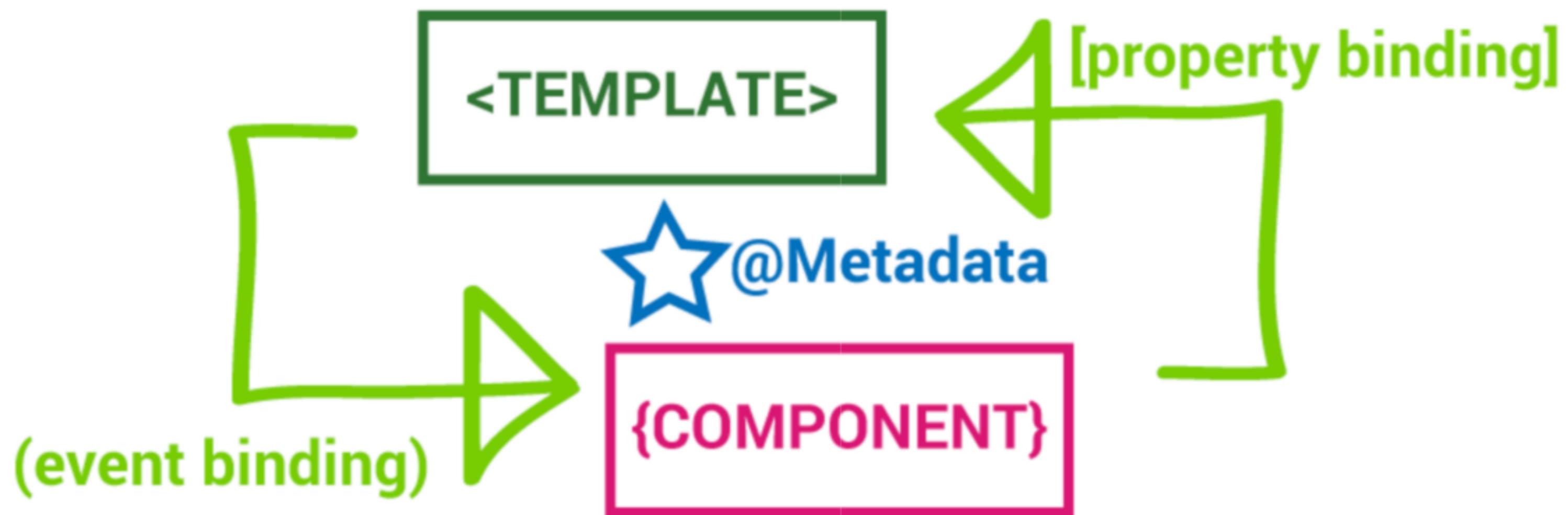
- Décore une classe ES6
- On peut attacher des metadatas avec les annotations/décorateurs de TypeScript (ES7?)
- Les décorateurs sont des fonctions
- Le plus utilisé : `@Component()`
 - prend un objet de config avec le sélecteur, le template, les providers, les directives, les pipes et les styles

```
@Component({
  selector: 'home',
  templateUrl: 'app/home/home.component.html'
})
export class HomeComponent{ }
```

Metadata @Component()

Template

- Un template est écrit en HTML et définit le rendu d'un composant
- Les templates comprennent des "data-binding", d'autres composants et directives
- Angular 2 utilise les évènements DOM natifs
- Angular 2 utilise le shadow DOM



Template

```
@Component({
  selector: 'experiment',
  templateUrl: './experiment.detail.component.html',
  styles: [
    .experiment {
      cursor: pointer;
      outline: 1px lightgray solid; padding: 5px;
      margin: 5px;
    }
  ]
})
```



Template

```
@Component({
  selector: 'experiment',
  template: `
    <div class="experiment" (click)="doExperiment()">
      <h3>{{ experiment.name }}</h3>
      <p>{{ experiment.description }}</p>
      <p><strong>{{experiment.completed}}</strong></p>
    </div>
  `,
  styles: [
    .experiment {
      cursor: pointer;
      outline: 1px lightgray solid; padding: 5px;
      margin: 5px;
    }
  ]
})
```



Template

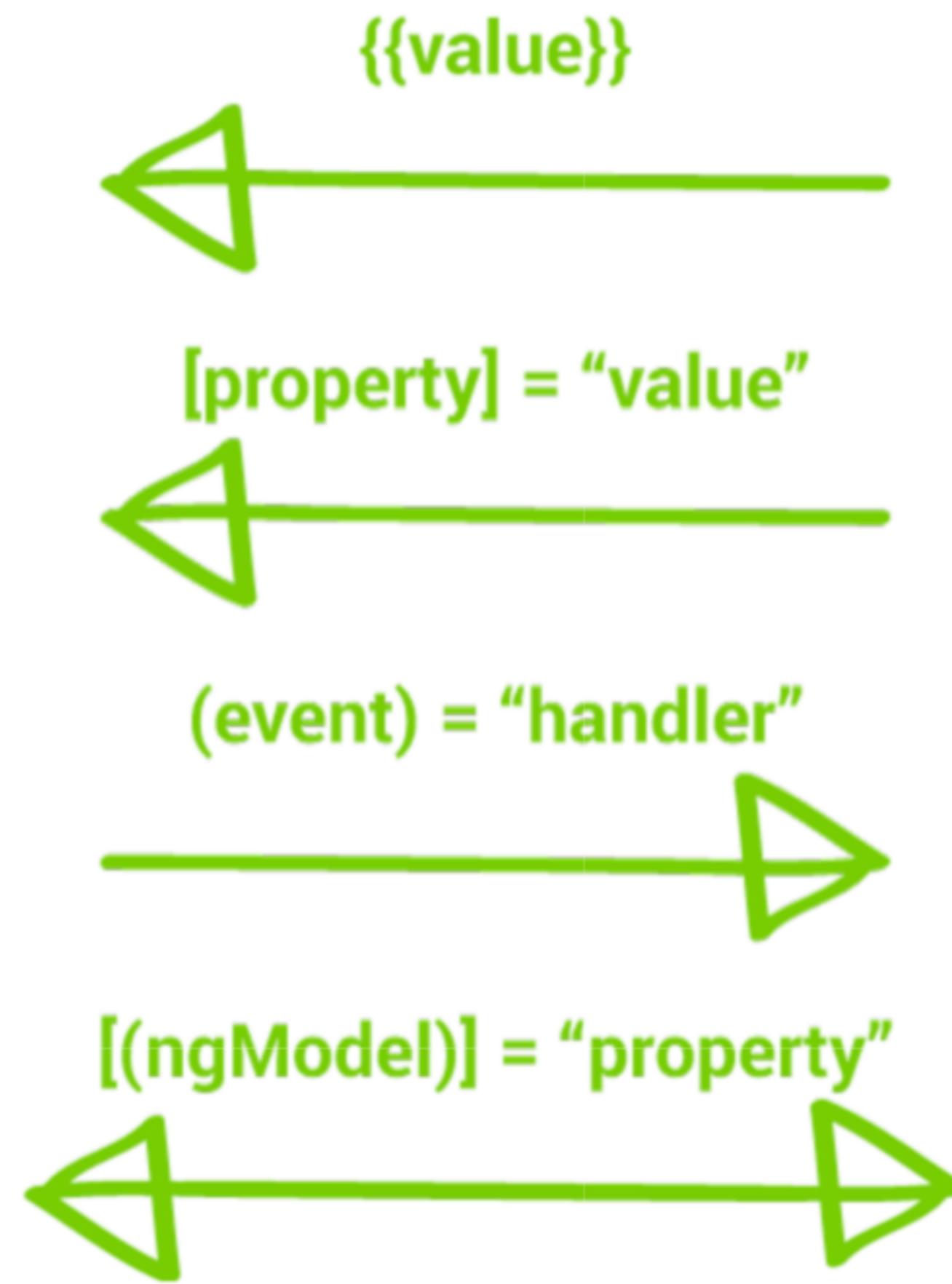
Data-binding

- Permet de créer un échange de données entre le composant et le template
- Inclut : interpolation, le binding de propriété, d'évènement et le two-way binding (propriété + évènement)

Data-binding

<TEMPLATE>

{COMPONENT}



```
<h1>{{title}}</h1>
<p>{{body}}</p>
<hr/>
<experiment *ngFor="#experiment of experiments"
            [experiment]="experiment"></experiment>
<hr/>
<div>
    <h2 class="text-error">Experiments: {{message}}</h2>
    <form class="form-inline">
        <input type="text"
              [(ngModel)]="message" placeholder="Message">
        <button type="submit" class="btn" (click)="updateMessage(message)">
            Update Message
        </button>
    </form>
</div>
```



Data-binding

Service

- Un service est une classe ES6
- Un service s'occupe d'une chose uniquement
- Embarque toute la logique business du composant
- Classe ES6 décorée avec `@Injectable` si on veut l'injecteur dans no service

```
import {Injectable} from 'angular2/core';
import {Experiment} from './experiment.model';

@Injectable()
export class ExperimentsService {
    private experiments: Experiment[] = [];

    getExperiments(): Experiment[] {
        return this.experiments;
    }
}
```



Service

Directive

- Une directive est une classe ES6
- Décorée avec `@Directive()`
- Un composant est une directive avec un template !
- Directives disponibles nativement dans Angular 2

```
import { Directive, ElementRef } from 'angular2/core';

@Directive({
  selector: '[femBlinker]'
})

export class FemBlinker {
  constructor(element: ElementRef) {
    let interval = setInterval(() => {
      let color = element.nativeElement.style.color;
      element.nativeElement.style.color =
        (color === '' || color === 'black') ? 'red' : 'black';
    }, 300);

    setTimeout(() => {
      clearInterval(interval);
    }, 10000);
  }
}
```



Directive

Dependency Injection

- Permet de récupérer l'instance d'une class par son nom (Singleton)
- Permet d'initialiser qu'une seule fois les services
- Pour utiliser DI pour un service, il faut l'enregistrer en tant que provider : au bootstrap de l'application ou dans les metadata d'un composant

```
// experiments.service.ts
import {Injectable} from 'angular2/core';
@Injectable()
export class ExperimentsService { }

// experiments.component.ts
import {ExperimentsService} from '../common/experiments.service';
import {StateService} from '../common/state.service';
export class ExperimentsComponent {
    constructor(
        private _stateService: StateService,
        private _experimentsService: ExperimentsService
    ) {}
}
```



Dependency Injection

Détection de changement

- Regarde les changement dans le modèle de donnée pour recompiler le DOM
- Les changements sont : les évènements, les requêtes XHR et les timers
- Chaque composant possède son détecteur de changement
- On peut personnaliser le `ChangeDetectionStrategy.OnPush` pour utiliser des objets immutable ou observables
- On peut dire à Angular d'écouter un composant particulier en injectant `ChangeDetectorRef` et en appelant `markForCheck()` dans le composant

```
export interface Item {  
    id: number;  
    name: string;  
    description: string;  
};  
export interface AppStore {  
    items: Item[];  
    selectedItem: Item;  
};
```

Détection de changement

Les tests

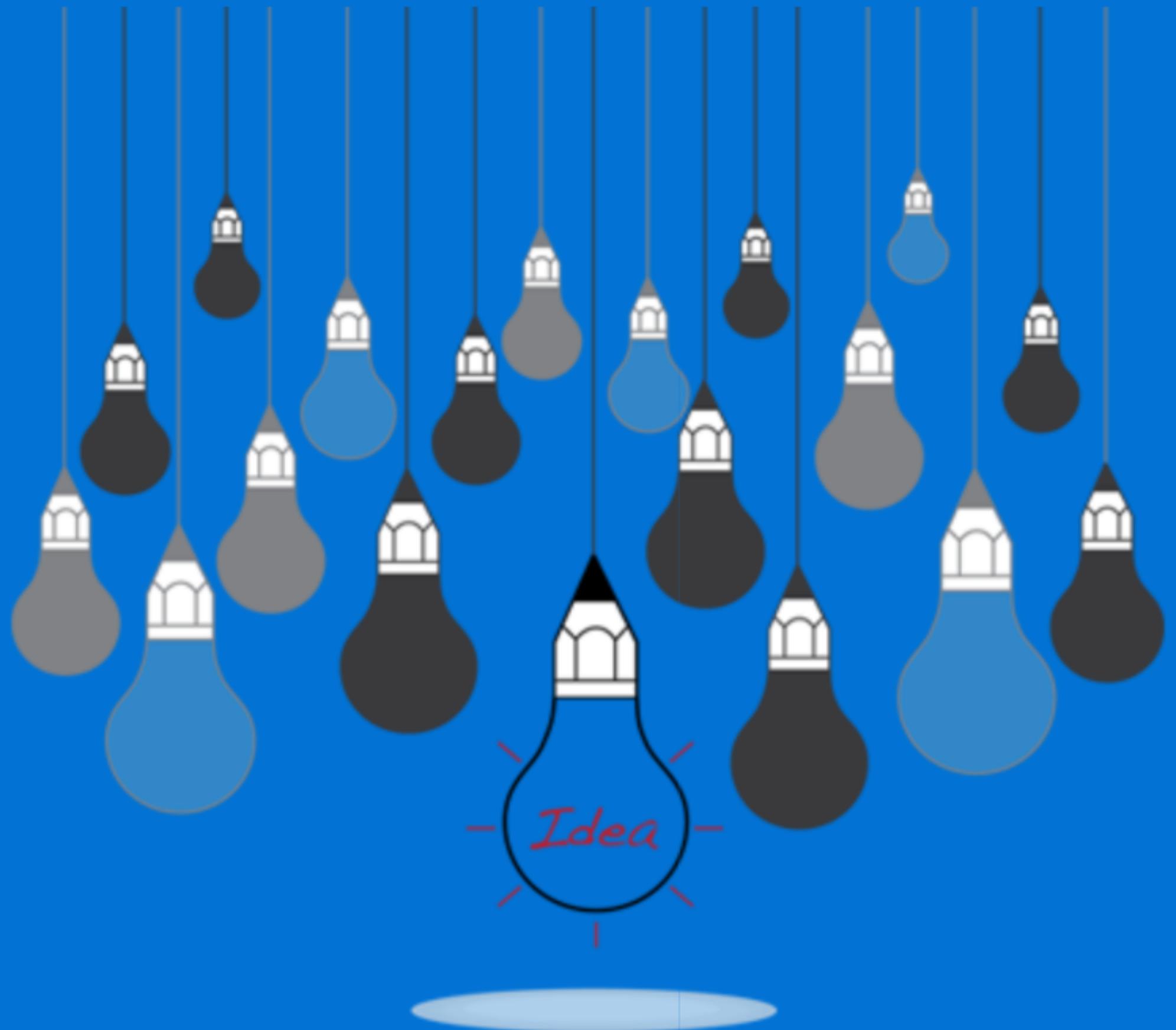
- Angular utilise Jasmine (framework de test)
- On doit :
 - Importer les dépendances depuis
@angular/testing
 - Importer les classes à tester
 - Inclure les providers en important
beforeEachProviders
 - Injecter les providers en appelant
inject([arrayOfProviders], (providerAliases) => {})

```
import { describe, it, expect } from 'angular2/testing';
import { AppComponent } from './app.component';
describe('AppComponent', () => {
  it('should be a function', () => {
    expect(typeof AppComponent).toBe('function');
  });
});
```

Les tests

Les conseils d'architecture

- Inclure tous les fichiers liés à un composant dans le même dossier
- **CIA** pour **Class Import Anotate**
- Utiliser des templates écrit dans le fichier JS
- Garde le template simple et court
- Déporter la logique métier du composant vers un provider
- Découper vos composants complexe
- Toujours réfléchir à comment les données vont être changer (impact sur le Change Detection)



Défi

- Récupérer et faire tourner l'application
- Identifier les artefacts Angular
- Ajouter une propriété dans un composant et l'afficher dans la vue
- Ajouter une propriété dans le service **stateService** et l'utiliser dans un composant
- **BONUS** : Créer une interface pour typer votre propriété

Les outils de développement

- NodeJS
- Webpack
- ES6/ES5
- TypeScript
- Typings