DÉCOUVRONS ANGULAR> 2

https://nodejs.org/en/

Première version : 27 mai 2009
Programmé en : C, C++, JavaScript

Installer la version LTS

En installant nodejs, vous obtiendrez automatiquement le gestionnaire de dépendances npm.

Gestionnaire de dépendances
https://yarnpkg.com/fr/

npm install yarn –global

https://blog.zenika.com/2017/03/13/npm-vs-yarn/

ANGULARJS

Première version : 2009
Programmé en : JavaScript
Architecture : MVC
Dernière version : 1.7.2 (12 juin 2018)

ANGULAR

Première version : septembre 2016
Architecture : Component
Programmé en : TypeScript
Dernière version : 7.2.4 (6 février 2019)
Angular est une réécriture complète de AngularJS, cadriciel construit par la même équipe.

VERSION 2.0

Angular 2.0 est annoncé à la conférence ng-europe 2014, qui s'est déroulé les 22 et 23 octobre de cette même année.

Les changements drastiques dans la version 2.0 ont créé beaucoup de controverses parmi les développeurs.

VERSION 7.X

Chaque version est prévue pour être compatible avec la version antérieure. Google a promis de faire des mises à jour deux fois par année. Angular 8 est planifié pour mars/avril 2019. Angular 9 pour septembre/octobre 2019-20

TYPESCRIPT

TypeScript est un langage de programmation libre et open source développé par Microsoft qui a pour but d'améliorer et de sécuriser la production de code JavaScript.

C'est un sur-ensemble de JavaScript (c'est-à-dire que tout code JavaScript correct peut être utilisé avec **TypeScript**).

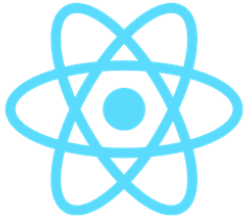
Typage : dynamique, faible, fort optionnel, statique optionnel

Date de première version : 9 février 2012

Extension de fichier : ts

Microsoft

TYPESCRIPT - SAMPLES



https://www.typescriptlang.org/samples/index.html


ionic

PRINCIPES

Le premier principe à retenir est le suivant : **les applications web créées avec Angular sont des SPA** (single page applications).

Concrètement, le seul fichier HTML qui sera exposé au navigateur par le serveur sera le fichier index.html.

Le navigateur ne changera jamais de page.

Toutes les données (templates HTML, données brutes, etc.) seront récupérées en AJAX et injectées dans le DOM par Angular.

DATA BINDING

Angular a défini quatre sortes de Data Binding pour synchroniser le template et le Component.

Il est ainsi possible de propager une donnée du Component vers le DOM, du DOM vers le Component et dans les deux sens.

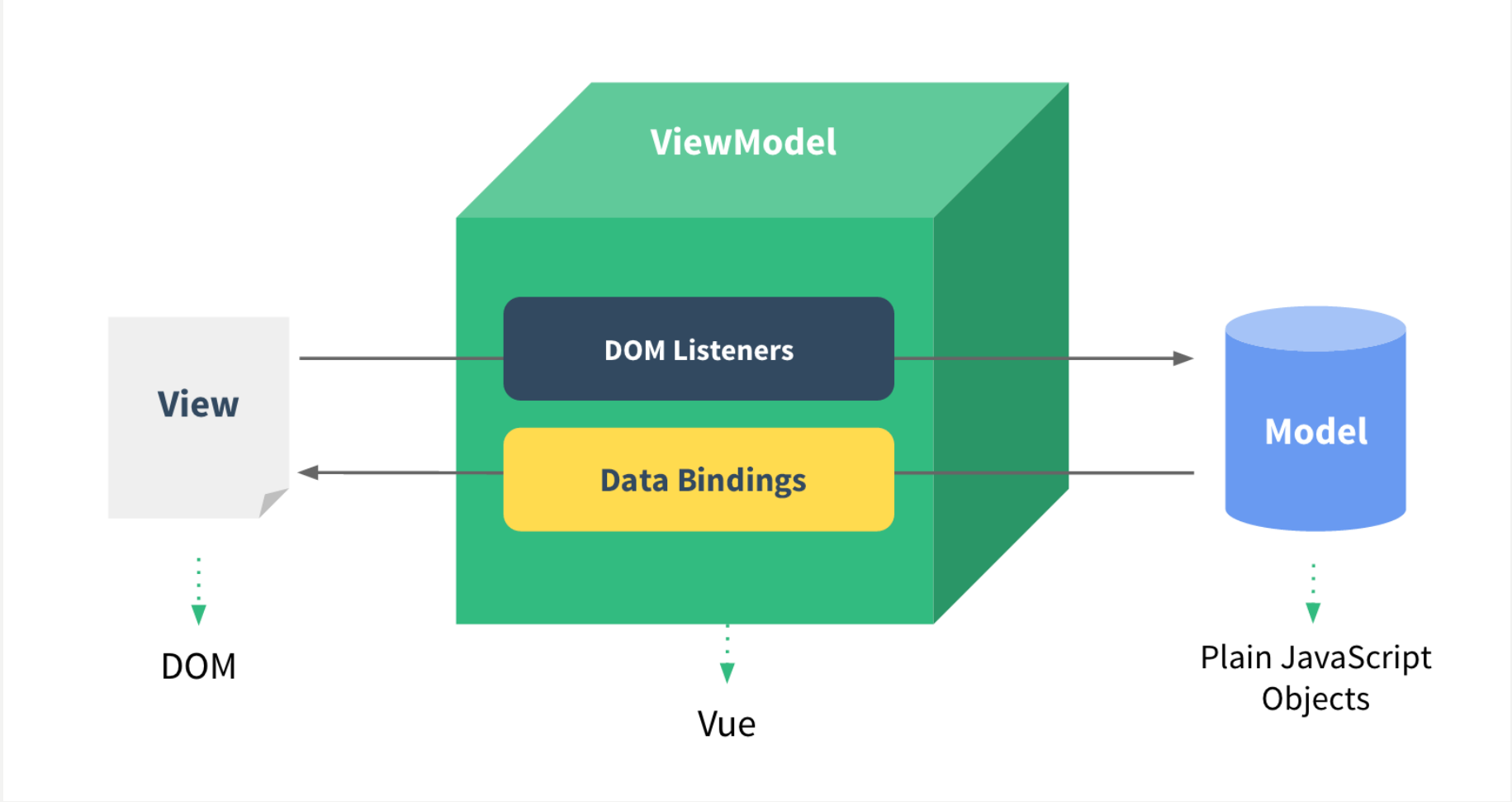
Ces formes de Data Binding sont communément nommées:

- Interpolation : Ce mécanisme permet de modifier le DOM à partir du modèle, si un changement est intervenu sur une valeur de ce dernier.
- Property Binding : Ce mécanisme permet de valoriser une propriété d'un composant ou d'une directive à partir du modèle, si un changement est intervenu sur une valeur de ce dernier.
- Event Binding : Ce mécanisme permet d'exécuter une fonction portée par un Component suite à un évènement émis par un élément du DOM.
- Le "two-way" Data Binding

TWO WAY – DATA BINDING

Le "two-way" Data Binding : C'est une combinaison du Property Binding et du Event Binding sous une unique annotation.

Dans ce cas là, le component se charge d'impacter le DOM en cas de changement
du modèle ET le DOM avertit le Component d'un changement via l'émission d'un évènement.


ViewModel
DOM Listeners
Data Bindings
View
Model
DOM
Vue
Plain JavaScript
Objects

INSTALLER ANGULAR CLI

NPM :
npm install -g @angular/cli@latest

YARN :

yarn global add @angular/cli@latest

GÉNÉRER UN PROJET

Exécutez la ligne de commande suivante :
$ ng new app --skip-install

Would you like to add Angular routing? Yes
Which stylesheet format would you like to use? Sass (.scss) [http://sass-lang.com]

$ cd app
$ yarn

En exécutant la commande yarn, l'ensemble des dépendances du projet vont être installées.

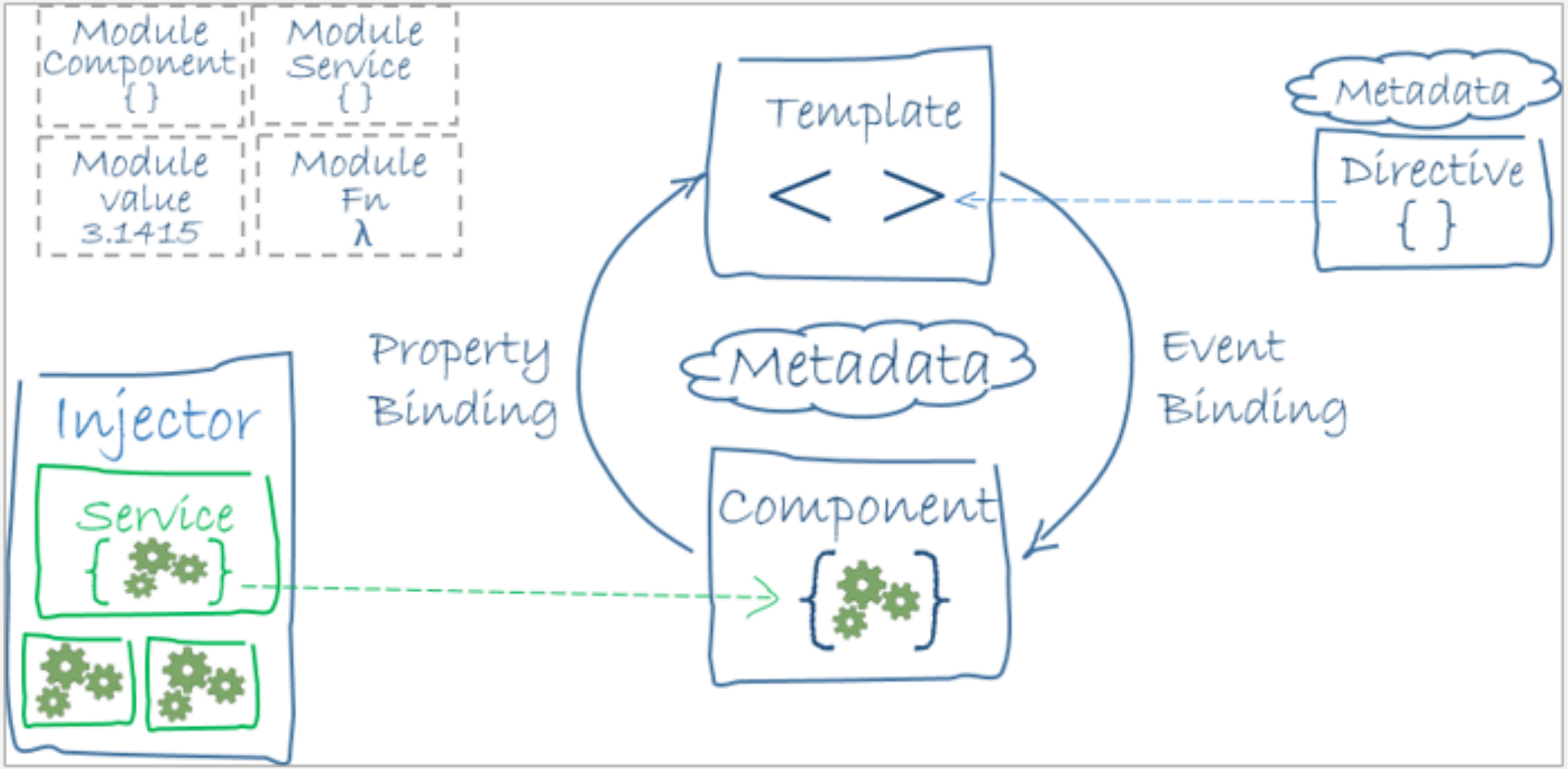
$ ng serve

ARCHITECTURE

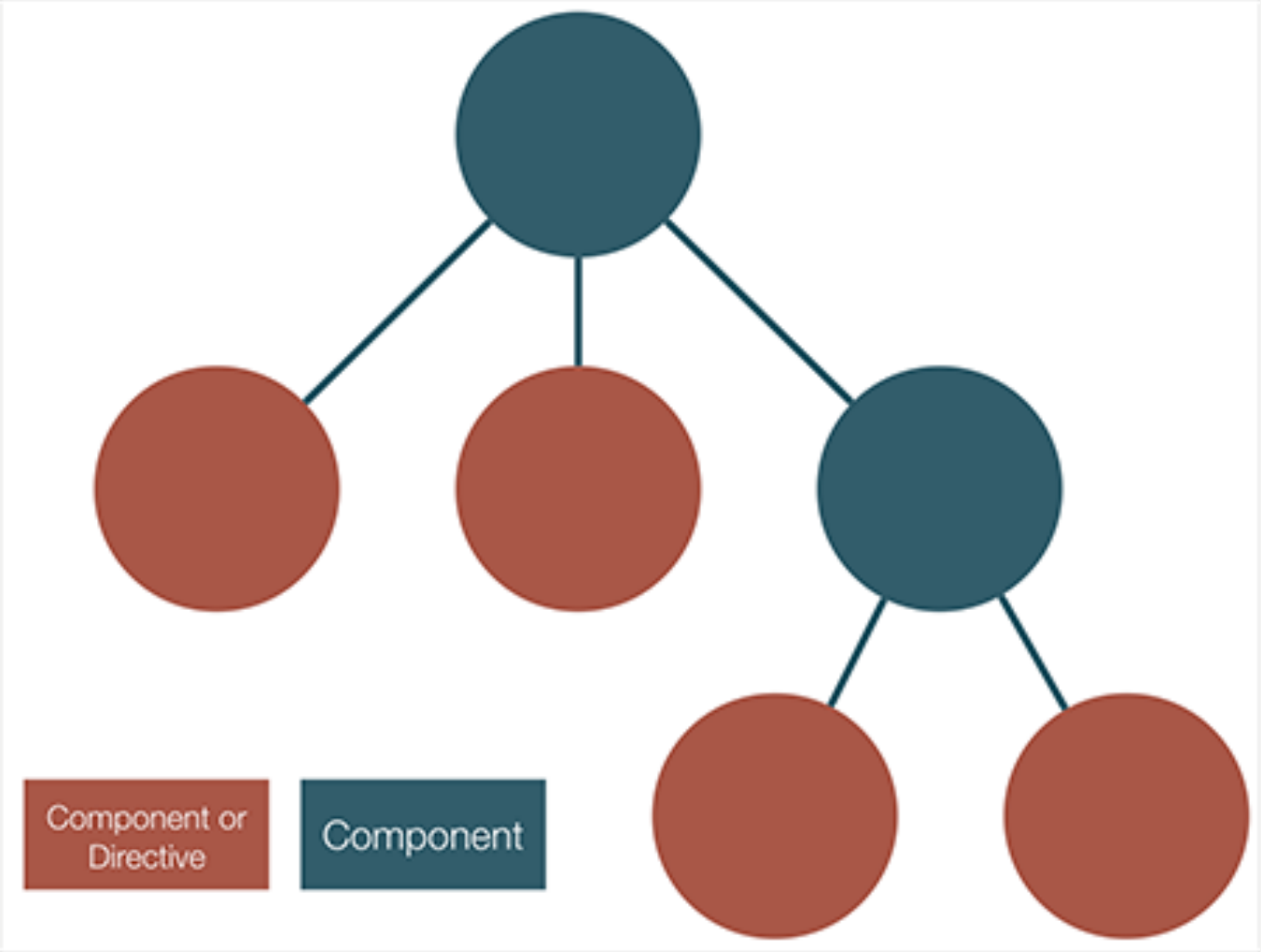
Angular est aujourd'hui basé sur **une architecture de composants complètement indépendants les uns des autres**. Une fois le composant principal chargé, il analyse ensuite la vue html correspondant à celui-ci et détecte si il comporte des composants imbriqués. Si c'est le cas, Angular va trouver toutes les correspondances et exécuter le code lié a celles-ci.

On peut imbriquer autant de composants que l'on souhaite.

Un composant dans Angular sert à **générer une partie de code html et fournir des fonctionnalités** à celle-ci. C'est pour cela qu'un composant est constitué d'une classe dans laquelle on pourra définir la logique d'application pour ce composant avec des propriétés, des méthodes..etc.



https://angular.io/guide/architecture


Component or Directive
Component

COMPONENT

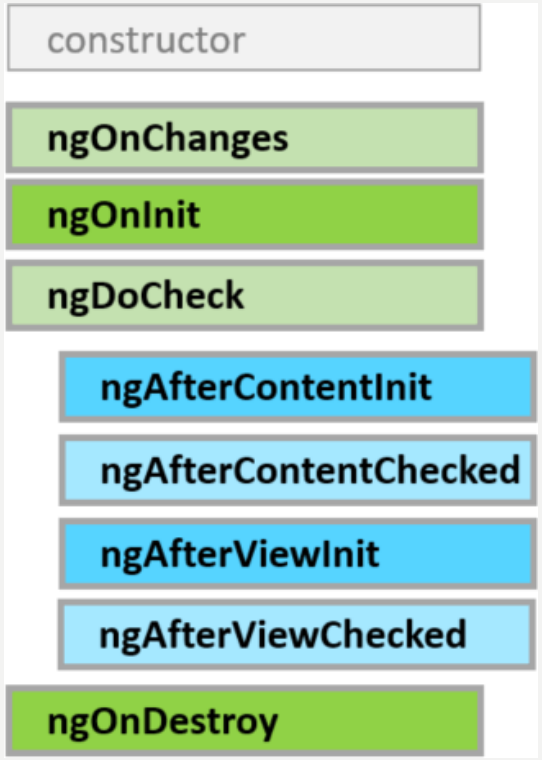
ng generate component « name »

https://angular.io/tutorial/toh-pt1

The CLI generated three metadata properties:

- selector— the component's CSS element selector
- templateUrl— the location of the component's template file.
- styleUrls— the location of the component's private CSS styles.

LIFECYCLE HOOKS



https://angular.io/guide/lifecycle-hooks

DISPLAYING VARIABLE

Définir les variables au niveau de la classe
Au niveau du template, on affiche la variable en utilisant le double accolades.

```
template: `
  <h1>{{title}}</h1>
  <h2>My favorite hero is: {{myHero}}</h2>
  `
```

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

@Component({
  selector: 'app-root',
  template: `
    <h1>{{title}}</h1>
    <h2>My favorite hero is: {{myHero}}</h2>
    `
})
export class AppComponent {
  title = 'Tour of Heroes';
  myHero = 'Windstorm';
}
```

TYPAGE / INTERFACE

```
function greeter(person: string) {
    return "Hello, " + person;
}
```

```
interface Person {
    firstName: string;
    lastName: string;
}

function greeter(person: Person) {
    return "Hello, " + person.firstName + " " + person.lastName;
}

let user = { firstName: "Jane", lastName: "User" };
```

CLASS, CONSTRUCTOR

```
class Student {
    fullName: string;
    constructor(public firstName: string, public middleInitial: string, public lastNam
e: string) {
        this.fullName = firstName + " " + middleInitial + " " + lastName;
    }
}

interface Person {
    firstName: string;
    lastName: string;
}

function greeter(person : Person) {
    return "Hello, " + person.firstName + " " + person.lastName;
}

let user = new Student("Jane", "M.", "User");
```

NGFOR

On doit définir dans un premier temps un tableau au sein du composant.

```
tasks = [{
    id: 1,
    name: 'Faire le ménage'
},{
    id: 2,
    name: 'Laver la voiture'
},{
    id: 3,
    name: 'Ranger mon ordinateur'
}];
```

```
<div>
    <h1>Liste des tâches</h1>
    <table class="table table-striped">
        <thead>
            <tr>
                <th>Id</th>
                <th>Nom</th>
            </tr>
        </thead>
        <tbody>
            <tr *ngFor="let task of tasks">
                <td>{{ task.id }}</td>
                <td>{{ task.name }}</td>
            </tr>
        </tbody>
    </table>
</div>
```

NGMODEL

```
<input type="text" class="form-control" placeholder
="Saisissez le nom de la tâche" [(ngModel)]="name"
>
```

Le ngModel est une référence que l'on doit ajouter sur un champs afin que l'on puisse récupérer sa donnée.

Vous devrez déclarer une variable au sein de votre TS, cette variable sera associée à la valeur du champs.

La notion de data binding vue préalablement rentre en jeu.

```
name: string = '';
```

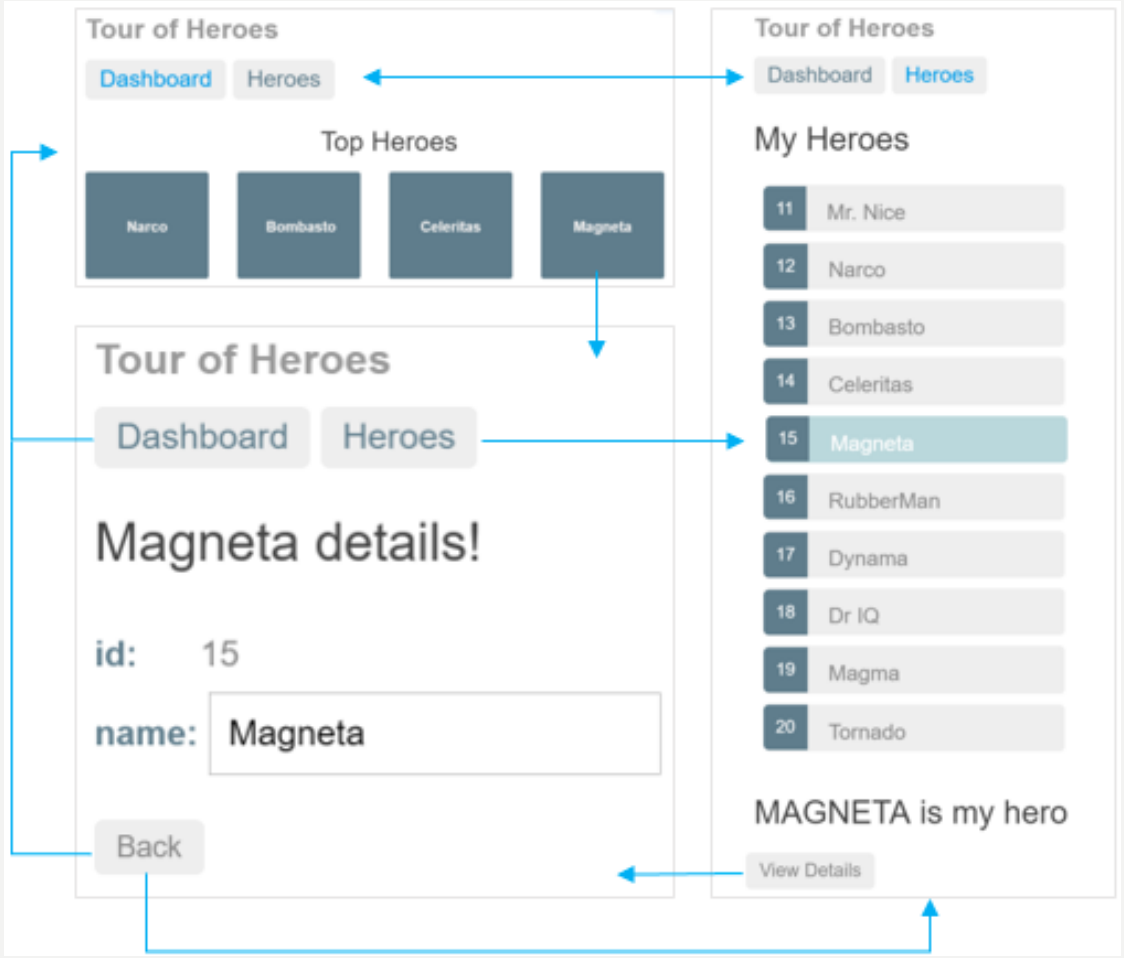
CLICK EVENT

https://angular.io/guide/user-input#binding-to-user-input-events

```
@Component({
  selector: 'app-click-me',
  template: `
    <button (click)="onClickMe()">Click me!</button>
    {{clickMessage}}`
})
export class ClickMeComponent {
  clickMessage = '';

  onClickMe() {
    this.clickMessage = 'You are my hero!';
  }
}
```

ROUTING



https://angular.io/tutorial/toh-pt5

https://angular.io/tutorial/toh-pt5#add-routes

FORM

Importer ReactFormsModule dans notre App.

https://angular.io/guide/reactive-forms#getting-started

```
import { ReactiveFormsModule } from '@angular/forms';

@NgModule({
  imports: [
    // other imports ...
    ReactiveFormsModule
  ],
})
export class AppModule { }
```

FORM CONTROL

Définir sur chaque champs un FormControl.

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

@Component({
  selector: 'app-name-editor',
  templateUrl: './name-editor.component.html',
  styleUrls: ['./name-editor.component.css']
})
export class NameEditorComponent {
  name = new FormControl('');
}
```

FORM GROUP

Définition d'un formulaire de champs.

FormGroup permetttra de vérifier si l'ensemble des champs du formulaire.

```
import { Component } from '@angular/core';
import { FormGroup, FormControl } from '@angular/forms';

@Component({
  selector: 'app-profile-editor',
  templateUrl: './profile-editor.component.html',
  styleUrls: ['./profile-editor.component.css']
})
export class ProfileEditorComponent {
  profileForm = new FormGroup({
    firstName: new FormControl(''),
    lastName: new FormControl(''),
  });
}
```

FORM TEMPLATE

Au niveau de notre template, on ajoute ce formGroup sur notre form.

Et les formControlName sur les différents champs que l'on a créé.

```
<form [formGroup]="profileForm">

  <label>
    First Name:
    <input type="text" formControlName="firstName">
  </label>

  <label>
    Last Name:
    <input type="text" formControlName="lastName">
  </label>

</form>
```

FORM - BUILDER

Le FormBuilder permet de créer un formulaire plus facilement.

FormBuilder doit être importé au niveau du constructeur.

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

@Component({
  selector: 'app-profile-editor',
  templateUrl: './profile-editor.component.html',
  styleUrls: ['./profile-editor.component.css']
})
export class ProfileEditorComponent {
  profileForm = this.fb.group({
    firstName: [''],
    lastName: [''],
    address: this.fb.group({
      street: [''],
      city: [''],
      state: [''],
      zip: ['']
    }),
  });

  constructor(private fb: FormBuilder) { }
}
```

FORM VALIDATION

```
import { Validators } from '@angular/forms';
```

```
profileForm = this.fb.group({
  firstName: ['', Validators.required],
  lastName: [''],
  address: this.fb.group({
    street: [''],
    city: [''],
    state: [''],
    zip: ['']
  }),
});
```

On doit dans un premier temps importer Validators de @angular/forms.

Puis, sur les champs créé précédemment, on doit saisir nos règles en 2ème paramètre.

Si on souhaite que notre champs soit requis, on peut mettre Validators.required.

HTTP

https://angular.io/guide/http

SOURCES

http://www.learn-angular.fr/le-data-binding-angular/