

Jour 1 1. Angular ?? 2. Les Fondamentaux 3. Données et Événements	Jour 2 1. Composant réutilisable 2. Directives 3. Template-driven Forms	Jour 3 1. Reactive Forms 2. Consommer des Services HTTP 3. Routing et Navigation	Jour 4 1. Authentification et Autorisation 2. Consommer des Services HTTP	Jour 5 1. Déploiement	TP 1. exos 2. fil rouge
--	--	---	---	--	---

Angular - Présentation

1 </Présentation

Angular est :

- un framework pour construire des applications client
- en HTML, CSS, Javascript / Typescript
- Angular a été codé en Typescript, nous allons donc privilégier le Typescript

Quelle est la valeur ajoutée d'Angular par rapport à une application client utilisant jQuery ?

- Angular est plus structurant = plus facile à maintenir = plus facile à faire évoluer
- Angular permet d'utiliser des concepts de Javascript complexe comme les class / l'héritage ... plus facilement
- Angular met à disposition de nombreuses fonctions utilitaires qui l'on pourra utiliser et réutiliser
- **Utiliser Angular facilite la vie du développeur**



2 </Angular ou AngularJS ???

Angular 1



2010

Angular 2 et suivant...



2016

- AngularJS correspond à la version 1 du framework lancé en 2010 par Google
- En 2016, le framework est intégralement réécrit : AngularJS version 2
- Toutes les versions ultérieures à la version 2 sont appelées **Angular** directement
- Nous allons étudier **Angular**
- On précisera la version de Angular par exemple 4 ou 8 pour donner des informations sur la compatibilité / modules disponibles
- Le site officiel du framework : <https://angular.io/>
- Page wikipedia du framework : [wikipedia.org](https://en.wikipedia.org/wiki/Angular_(JavaScript_framework))

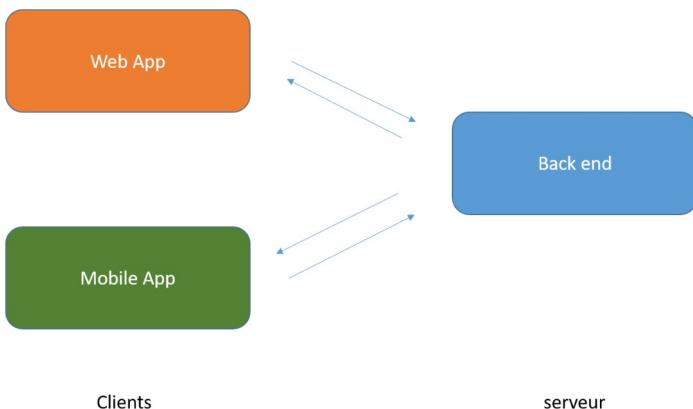
3 </Architecture

Angular est utilisé pour créer :

- des front End aussi appelé Client aussi appelé UI : User Interface

Qu'est ce qu'un client / un serveur ??

- Le Client est ce que voit l'internaute et avec lequel il va interagir
- Le Back End est responsable du stockage des données et réaliser des traitements demandés par le Client
- Donc **aucune donnée n'est stockée** dans un client (=Angular)
- Le client va interroger le serveur via des Services HTTP aussi appellés API (Application Program Interface pour réaliser des traitements : Ajouter, Lire, Mettre à jour ou Supprimer des données de la Base de données par exemple)
- Le Client ne va se charger **que de la présentation**



Cas Pratiques

Article sur le sujet

[The Obvious UI is Often the Best UI](#)

4 </Installer Node sur votre ordinateur



- Aller sur <https://nodejs.org/fr/>
- Deux versions sont disponibles :
 - TLS : Lastest Stable Version
 - Current : version en cours avec de nouvelles fonctionnalités
- Télécharger et installer la version TLS

Node.js® est un environnement d'exécution JavaScript construit sur le moteur
JavaScript V8 de Chrome.

Téléchargements pour Windows (x64)

10.16.2 LTS
Recommandé pour la plupart des utilisateurs
Autres téléchargements Journal des modifications API Docs
12.8.0 Actuelle
Dernières fonctionnalités
Autres téléchargements Journal des modifications API Docs

- Lorsque c'est fini, lancer un terminal : touche Windows + R
- saisir `cmd` dans l'invite de commande
- saisir `node -v`
- le numéro de version doit apparaître = Node est installé sur votre machine !!!

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [version 10.0.17134.885]
(c) 2018 Microsoft Corporation. Tous droits réservés.

C:\Users\HP>node -v
v10.14.2
```

5 </Installer Angular CLI

`npm i -g @angular/cli`

Client en ligne de commande permettant de créer un projet Angular

Le package est disponible sur [npmjs.com](https://www.npmjs.com/package/@angular/cli)

```
C:\WINDOWS\system32>npm i -g @angular/cli@latest
C:\Users\HP\AppData\Roaming\npm> -> C:\Users\HP\AppData\Roaming\npm\node_modules\@angular\cli\bin\ng
> @angular/cli@8.3.3 postinstall C:\Users\HP\AppData\Roaming\npm\node_modules\@angular\cli
> node ./bin/postinstall/script.js
+ @angular/cli@8.3.3
added 243 packages from 185 contributors in 41.877s
```

```
{lamb} ng --version
Angular CLI: 8.0.2
Node: 10.14.2
OS: win32 x64
Angular: undefined
...
Package          Version
@angular-devkit/architect    0.800.2
@angular-devkit/core         8.0.2
@angular-devkit/schematics   8.0.2
@angular/cli                 8.0.2
@schematics/angular          8.0.2
@schematics/update           0.800.2
@rxjs                        6.4.0
```

Lorsque Angular est installé, lancer la commande suivante :

- `ng --version`

6 </Notre première application Angular

Nous allons créer notre première application Angular qui va s'appeler **jour1**. Saisir la commande suivante :

1. se positionner dans le Bureau Windows dans un invite de commande
2. `ng new jour1` // cette commande va installer Angular
3. Would you like to add Angular routing? `N`
4. Which stylesheet format would you like to use? `CSS`
5. Angular va créer la structure d'une projet Angular et télécharger tous les modules nécessaire à son fonctionnement sur npmjs.com (cette opération prendre de 2 à 5 min ...)

```
C:\Users\HP\Desktop (git)\hg
lamb:~ ng new jour1
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? CSS
CREATE jour1/angular.json (3585 bytes)
CREATE jour1/package.json (1279 bytes)
CREATE jour1/README.md (1022 bytes)
CREATE jour1/tsconfig.json (543 bytes)
CREATE jour1/tsconfig.app.json (543 bytes)
CREATE jour1/tsconfig.spec.json (543 bytes)
CREATE jour1/.editorconfig (246 bytes)
CREATE jour1/.gitignore (631 bytes)
CREATE jour1/.browserslist (429 bytes)
CREATE jour1/karma.conf.js (1017 bytes)
CREATE jour1/tsconfig.app.json (270 bytes)
CREATE jour1/tsconfig.spec.json (270 bytes)
CREATE jour1/src/favicon.ico (948 bytes)
CREATE jour1/src/main.ts (373 bytes)
CREATE jour1/src/polyfills.ts (2838 bytes)
CREATE jour1/src/styles.css (86 bytes)
CREATE jour1/src/test.ts (642 bytes)
CREATE jour1/src/assets/.gitkeep (0 bytes)
```

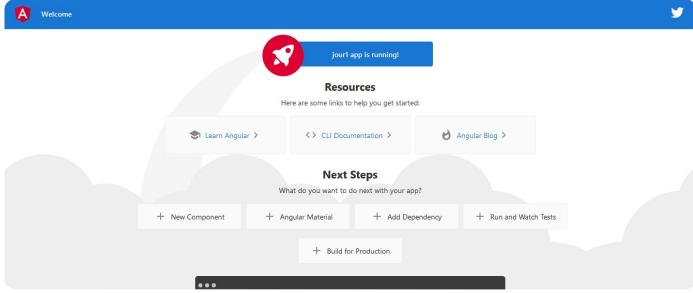
Maintenant que nous avons installé Angular, il faut le lancer :

6. `cd jour1` // se positionner dans le dossier jour1 qui contient le code source de Angular
7. `ng serve -o` ou `ng serve --open` ** // opération peut prendre 30s à 2 min

** possibilité de changer le port par défaut : `ng serve -o --port=4201`

```
C:\Users\HP\Desktop\git\{hg}
{lamb} cd jour1
{lamb} ng serve -o
10% building 3/3 modules 0 active [wds]: Project is running at http://localhost:4200/webpack-dev-server/
[ wds]: webpack output is served from /
[ wds]: 404s will fallback to /index.html

chunk {main} main.js, main.js.map (main) 47.8 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 264 kB [initial] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 7.8 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 3.81 MB [initial] [rendered]
Date: 2019-09-09T10:52:53.263Z - Hash: d2feef2887cf1c09af0ee - Time: 11762ms
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/
**
[ wds]: Compiled successfully.
```



7 </Structure d'un projet Angular

```
jour1
+-- e2e : e2e : end to end - ici que l'on écrit des test de bout en bout pour notre application : test qui simule un internaute
+-- node_modules : ici que sont stockés toutes les librairies qui font fonctionner Angular. Elles sont disponibles lors de la phase de développement.
+-- src : Lorsque l'on mettra en ligne nos projets, elles seront stockées sous forme de bundle
    Le dossier src ou source : ici qu'est stockée les sources de notre application Angular

    +-- app : app contient notre application :
        |   app.component.css
        |   app.component.html
        |   app.component.spec.ts
        |   app.component.ts
        |   app.module.ts
        C'est dans ce dossier que nous allons passer le plus de temps

    +-- assets : ici qu'il faut mettre les assets statiques de notre application : images / vidéo / document pdf ...
    +-- environments : ici que l'on va stocker les variables d'environnement (rappel : set PORT=3001)
        |   environment.prod.ts
        |   environment.ts

        • index.html : la page qui est utilisée pour gérer l'intégralité de notre application
            <!DOCTYPE html>
            <html lang="en">
            <head>
                <meta charset="utf-8">
                <title>jour1</title>
                <base href="/">
                <meta name="viewport" content="width=device-width, initial-scale=1">
                <link rel="icon" type="image/x-icon" href="favicon.ico">
            </head>
            <body>
                <app-root></app-root>
            </body>
            </html>

        • main.ts : le point de départ de notre application (appelé par angular.json)
            import { enableProdMode } from '@angular/core';
            import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

            import { AppModule } from './app/app.module';
            import { environment } from './environments/environment';

            if (environment.production) {
                enableProdMode();
            }

            platformBrowserDynamic().bootstrapModule(AppModule)
            .catch(err => console.error(err));

        • polyfills.ts : fichier qui importe des librairies pour que Angular puisse fonctionner avec n'importe quelle navigateur
        • style.css : styles globaux (s'applique à tous les composants / modules) de notre application
        • test.ts : définir notre environnement de test

    Des fichiers de configuration pour les outils en ligne de commande :
        • .editorconfig : fichier pour normaliser les IDE entre les développeurs
        • .gitignore : fichier de configuration pour l'outil de versionning git
        • angular.json : fichier de configuration en angular : c'est ce fichier est appelé en 1er
        • browserslist
        • karma.conf.js : fichier de configuration de karma, un outil en ligne de commande pour réaliser des tests
        • package-lock.json - package.json : fichier de configuration de npm
        • README.md : fichier d'aide
        • tsconfig.json : fichier de configuration de Typescript
        • tslint.json : fichier de configuration de tslint, un outil en ligne de commande pour réaliser des tests des fichiers Typescript
```

8 </webpack

[webpack](#) est un des outils présents dans Angular. Il va réaliser un certain nombre d'actions pour nous :

- Dès que vous réalisez une modification dans un fichier source, webpack va automatiquement remettre à jour le bundle (`main.js`) : Hot module replacement
 - l'ensemble des css que nous allons écrire dans notre application sont eux aussi transformé en bundle grâce à webpack : (`styles.js`)
 - si nous ajoutons une nouvelle librairie dans notre application, webpack va faire le nécessaire dans (`vendor.js`)
 - injecte l'ensemble des bundles dans notre page (`index.html`)

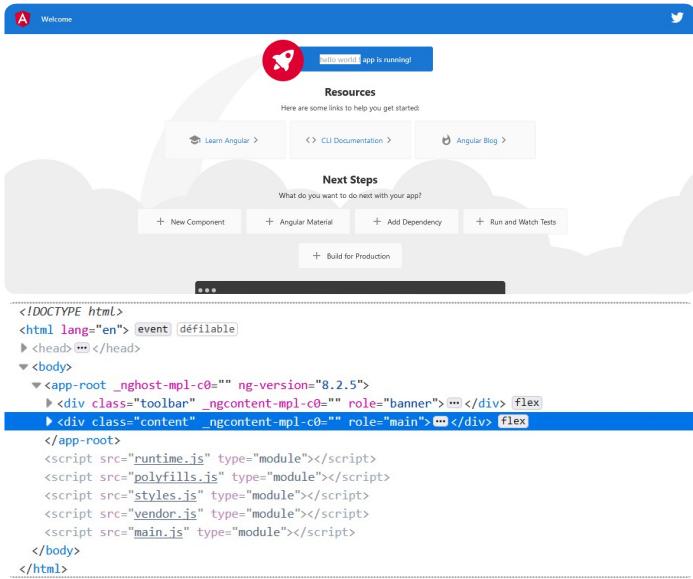
 Cas Pratiques

modifier le fichier app.component.ts

```
  /src/app/app.component.ts
  import { Component } from '@angular/core';

  @Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
  })
  export class AppComponent {
    title = 'Hello Word !';
  }

chunk {main} main.js, main.js.map (main) 47.8 kB [initial] [rendered]
chunk [polyfills] polyfills.js, polyfills.js.map (polyfills) 264 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.15 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 9.7 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 3.81 MB [initial] [rendered]
Date: 2019-09-09T10:52:53.263Z - Hash: df3ee2887fc10caafee - Time: 11762ms
Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/
**
[wdm]: Compiled successfully.
[wdm]: Compiling...
Date: 2019-09-09T12:04:15.002Z - Hash: c82223e0229869058390
4 unchanged chunks
chunk {main} main.js, main.js.map (main) 47.8 kB [initial] [rendered]
time: 557ms
[wdm]: Compiled successfully.
```



malik.h@webdevpro.net

Jour 1 1. Angular ?? 2. Les Fondamentaux 3. Données et Événements	Jour 2 1. Composant réutilisable 2. Directives 3. Template-driven Forms	Jour 3 1. Reactive Forms 2. Consommer des Services HTTP 3. Routing et Navigation	Jour 4 1. Authentification et Autorisation	Jour 5 1. Déploiement	TP 1. exos 2. fil rouge
--	--	---	---	--	---

Les fondamentaux

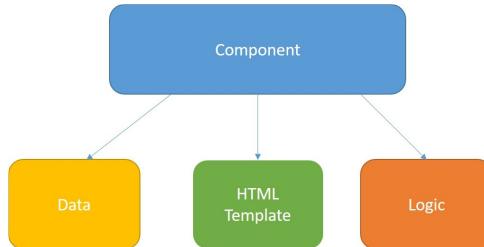
1 </> Qu'est qu'un composant ??

Un composant, ou Component, est **LA BRIQUE DE BASE** d'une application Angular.
Un composant associe 3 concepts :

- 1. Des Données
- 2. Des Balises HTML (Template HTML)
- 3. Une Logique

d'une vue (= une zone de l'écran visible par un internaute)

Documentation officielle : [Introduction to components](#)

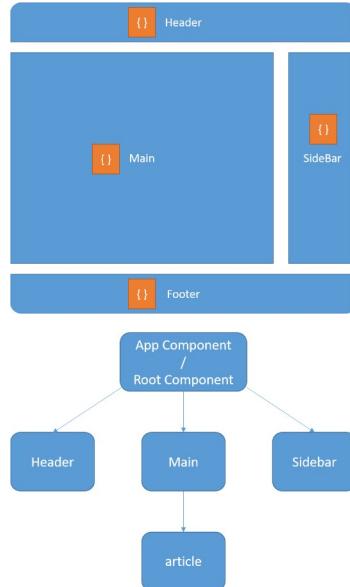


Voici un exemple de vue découpée en composant ======>

avantages de découper une vue en composants :

- Plus facile à maintenir et à mettre à jour
- Potentiellement réutilisable

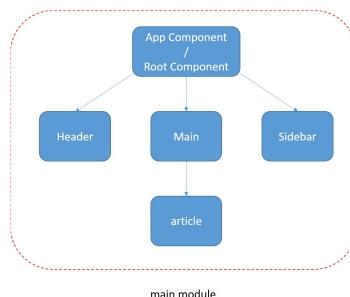
une application Angular est **un arbre de composant** qui commence par le composant root ou app



2 </> Qu'est ce qu'un module Angular ?

Module : conteneur de plusieurs composants

- Tout application Angular dispose au minimum d'un module qui s'appelle `app module` ou aussi `module root`
- les modules permettent de rassembler des fonctionnalités / zone de template d'une application
- Documentation officielle : [Introduction to modules](#)
- Documentation officielle : [Feature modules vs. root modules](#)



3 </> Créez un composant manuellement

3 étapes pour afficher un composant à l'écran :

1. Créer le composant

2. Enregistrer le composant dans le module root
3. Ajouter une balise html personnalisée dans la vue du module root

1 Créer un composant

1. Créer le fichier `/src/app/premier.component.ts`

2. Contenant :

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

@Component({
  selector: "premier", // 
  template : "<h1>premier composant</h1>" 
})
export class PremierComponent { }
```

respecter les règles de nommage du nom de fichier

- nom du composant,
 - si le nom simple tout en minuscule
 - si mot composé utilisé des tirets entre les mots par exemple premier-composant
 - Ne pas mettre de majuscule
 - ne pas mettre de chiffre en 1ère lettre
 - ne pas utiliser le mot test
- .component
- .ts : un composant est un fichier typescript
- C'est dans ce fichier que la logique composant va être stockée

Quelques commentaires :

1. composant est une class typescript
2. Attention au nommage de la class
 - 1ère lettre en majuscule
 - si nom composé utilisé le camelCase : MenuTop
 - suffixe `Component`
3. Utiliser `export` car elle va être utilisée dans d'autre fichiers .ts
4. Cette class est annotée d'une fonction Décoratrice `@Component()` :
5. **Le fait d'utiliser ce décorateur sur la class transforme la class en un composant pour Angular**
6. [Documentation officielle sur le sujet](#)
7. Pour pouvoir utiliser `@Component`, il faut l'importer la librairie depuis le dossier `node_modules` dans le dossier `@angular` dans le dossier `core`
8. enfin cette fonction décoratrice dispose d'un argument un objet avec les propriétés suivantes :
 - selector
 - template
9. `selector` définit une balise html personnalisée où le composant va être affiché
10. Angular permet d'ajouter de nouvelles balises html avec le nom de notre choix, attention ces nouvelles balises ne sont comprises que par Angular
11. `template` vaut les balises html du composant qui vont être affichées dans la vue

2 Enregistrer le composant dans le module root

Dans le fichier `/src/app/app.module.ts`, réaliser les modifications suivantes :

```
// /src/app/app.module.ts
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

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

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

Quelques commentaires

1. en haut des imports
2. en bas la class `AppModule`
3. au milieu la fonction décoratrice `@NgModule()` qui transforme la class `AppModule` en module Angular
4. pour déclarer notre composant il faut l'ajouter dans la propriété `declarations` de la fonction décoratrice
5. Enfin pour pouvoir l'utiliser il faut l'importer :
 - le fichier `src/app/premier.component.ts` est dans le même dossier que le fichier en cours, il faut ajouter ./
 - puis le nom du fichier `premier.component`
 - pas besoin de mettre l'extension .ts
6. le plugin pour [Visual Studio Auto-import](#) plugin ajoute la ligne d'import

3 Ajouter une balise html personnalisée dans la vue du module root

Dans le fichier `/src/app/app.component.html`, vider le fichier et saisir :

```
<premier></premier>
```

- la balise html personnalisée correspond à l'attribut `selector` du décorateur `@Component()` de la class `PremierComponent`

premier composant

4 Lancer http://localhost:4200

- regarder la page
- regarder l'inspecteur : la balise (`<premier></premier>`) contient de notre html écrit dans l'attribut (`template`) du décorateur (`@Component()`) de la classe (`PremierComponent`)

4 </> Créer un composant via Angular CLI

Dans un deuxième terminal, saisir la commande

- (`ng generate component [nom_composant]` ou (`ng g c [nom_composant]`))
- regarder les fichiers générés et mis à jour :
 - créé : (`src/app/deuxieme/deuxieme.component.ts`)
 - mis à jour : (`src/app/app.module.ts`)
- puis modifier le fichier (`src/app/app.component.html`)

```
<app-deuxieme></app-deuxieme>
```

Quelques Remarques :

- Par défaut, le générateur va prefixer le selector par (`app-`), vous êtes libres de :
 - le supprimer si vous avez une petite application à créer
 - le modifier éviter un doublon pour une grande application
- Le fichier (`.spec.ts`) est un fichier qui sera utile lorsque l'on veut réaliser des tests unitaires
- le fichier (`.ts`) implementer l'interface `OnInit` : aucun problème pour les enlever
- la class dans le fichier `.ts` contient deux méthodes (`constructor()`) et (`onInit()`) : aucun problème pour les enlever pour l'instant
- la directive (`templateUrl`) est un équivalent à template
- (`styleUrls`) permet de styliser notre composant via le fichier css créé
- vérifier que le fichier (`src/app/app.module.ts`) a été mis à jour
- Documentation complète sur la commande `ng generate`

```
(lamb) ng g c deuxieme
CREATE src/app/deuxieme/deuxieme.component.html (23 bytes)
CREATE src/app/deuxieme/deuxieme.component.spec.ts (642 bytes)
CREATE src/app/deuxieme/deuxieme.component.ts (277 bytes)
CREATE src/app/deuxieme/deuxieme.component.css (0 bytes)
UPDATE src/app/app.module.ts (481 bytes)

import { PremierComponent } from './premier.component';
import { DeuxiemeComponent } from './deuxieme/deuxieme.component';

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

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

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

premier composant
deuxieme works!
```

5 </> Interpolation

ou insérer dynamiquement des données dans notre vue

Modifier le fichier [/src/app/deuxieme/deuxieme.component.ts](#)

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-deuxieme',
  template: '<p>{{ texte }} </p>
<p>( "Description : " + texte ) </p>
<p>{{ getTexte() }} </p>',
  styleUrls: ['./deuxieme.component.css']
})
export class DeuxiemeComponent {
  texte :string = "lorem ipsum";
  getTexte () {
    return this.texte;
  }
}
```

Les données vont être insérées directement dans la vue via la syntaxe `({{ }})` et interprétées : **interpolation** - (insérer une variable dans un texte)

6 </> Faire une boucle dans notre vue via une Directive

Modifier le fichier [/src/app/deuxieme/deuxieme.component.ts](#)

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-deuxieme',
  template: '<p>{{ texte }} </p>
<ul>
  <li *ngFor="let j of jours ">{{ j }}</li>
</ul>',
  styleUrls: ['./deuxieme.component.css']
})
export class DeuxiemeComponent {
  texte :string = "lorem ipsum";
  jours :string[] = ["lundi", "mardi", "mercredi"];
}
```

Ajouter un nouvel attribut dans la balise html li : `*ngFor`, il s'agit d'une Directive qui permet de parcourir un tableau `jours` et modifier le DOM pour ajouter autant de li que d'éléments dans le tableau

7 </> Service

Il n'est pas bon de stocker des données directement dans le composant, il faut les sortir du composant et les stocker dans un service pour permettre de les mutualiser entre les différents composants de notre application Angular

1. créer le fichier [src/app/jours.service.ts](#) (noter la convention de nommage)
2. Ce fichier contient le code suivant :

```
export class JoursService {
  getJours()
  {
    return ["lundi", "mardi", "mercredi"];
  }
}
```

- Comme pour un Composant, un service est une classe
- Cette classe va être spécialisée dans le stockage / appel de données stockées sur notre serveur / réaliser des opérations spécifiques
- Documentation officielle [Introduction to services and dependency injection \(DI\)](#)

☰ Cas Pratiques

Excellent vidéo sur le sujet



8 </> Injection de dépendance

Nos données sont désormais dans un service, nous allons les rendre disponible à nos composants

Première manière : créer une nouvelle instance dans le constructeur

Modifier le fichier [/src/app/deuxieme/deuxieme.component.ts](#)

```
import { Component } from '@angular/core';
import { JoursService } from './jours.service';

@Component({
  selector: 'app-deuxieme',
  template: '<p>{{ texte }} </p>
    <ul>
      <li *ngFor="let j of jours ">{{ j }}</li>
    </ul>
  ',
  styleUrls: ['./deuxieme.component.css']
})
export class DeuxiemeComponent {
  texte :string = "lorem ipsum";
  jours :string[] ;
  constructor()
  {
    const service = new JoursService();
    this.jours = service.getJours() ;
  }
}
```

Deuxième manière : injection de dépendance

Modifier le fichier [/src/app/deuxieme/deuxieme.component.ts](#)

Documentation officielle sur [injection de dépendance](#)

```
import { Component } from '@angular/core';
import { JoursService } from './jours.service';

@Component({
  selector: 'app-deuxieme',
  template: '<p>{{ texte }} </p>
    <ul>
      <li *ngFor="let j of jours ">{{ j }}</li>
    </ul>
  ',
  styleUrls: ['./deuxieme.component.css']
})
export class DeuxiemeComponent {
  texte :string = "lorem ipsum";
  jours :string[] ;
  constructor(service : JoursService)
  {
    this.jours = service.getJours();
  }
}
```

Modifier le fichier [src/app/app.module.ts](#) (register le service)

```
import { PremierComponent } from './premier.component';
import { DeuxiemeComponent } from './deuxieme/deuxieme.component';
import { JoursService } from './jours.service';

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

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

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

III Cas Pratiques

Excellent vidéo sur le sujet : Injection de dépendance

Angular 8 Tutorial - 18 - Dependency Injection



Excellent vidéo sur le sujet : Utiliser un service

Angular 8 Tutorial - 19 - Using a Service



9 </> Créer un service via Angular CLI

Comme pour les composants, il existe une commande qui va permettre de créer plus facilement / rapidement un service :

- `ng generate service [nom service]` ou `ng g s [nom service]`

Deux fichiers créés : le fichier de service et un fichier de spec pour réaliser des tests unitaires

```
import { Injectable } from '@angular/core';
@Injectable({
  providedIn: 'root'
})
export class DataService {
  constructor() { }
}
```

```
(lamb) ng g s data
CREATE src/app/data.service.spec.ts (323 bytes)
CREATE src/app/data.service.ts (133 bytes)
```

c'est quoi le décorateur `@Injectable()` ??

- Permet à un service d'utiliser un autre service (et d'utiliser l'injection de dépendance vue précédemment)
- Mieux de décorer la classe avec sinon il s'agit juste d'une classe TypeScript

III Cas Pratiques

Cas n°1 : Réaliser la vue suivante

Formation IFOCOP

Liste des matières :

- NodeJS
- TypeScript
- Angular
- React

Cas n°2 : Réaliser la vue suivante

Page d'accueil !

Bienvenue sur mon Blog !!
Article1
 contenu article 1

Article2
 contenu article 2

Article3
 contenu article 3

consignes :

- créer un composant nommé formation
- créer un service nommé cours
- utiliser l'injection de dépendance pour fournir des données à la vue
- Documentation officielle sur le décorateur [@Injection](#)

consignes :

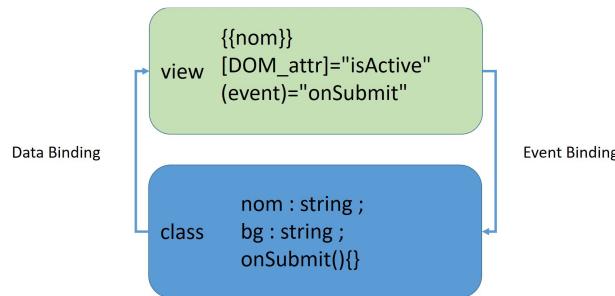
- créer un composant nommé blog
- créer un service nommé DataBlog stockant les données suivantes :

```
[ {titre: "Article1", contenu:"contenu article 1"}, {titre: "Article2", contenu:"contenu article 2"}, {titre: "Article3", contenu:"contenu article 3"} ]
```

- utiliser l'injection de dépendance pour fournir des données à la vue
- les données seront stockées

Jour 1	Jour 2	Jour 3	Jour 4	Jour 5	TP
1. Angular ?? 2. Les Fondamentaux 3. Données et Événements	1. Composant réutilisable 2. Directives 3. Template-driven Forms	1. Reactive Forms 2. Consommer des Services HTTP 3. Routing et Navigation	1. Authentification et Autorisation	1. Déploiement	1. exos 2. fil rouge

Données et gestion des événements



1 </> Property Binding

```

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

@Component({
  selector: 'deuxieme',
  template: '<h2>{{ texte }} </h2>
    <h2 [innerText]="'texte'"> </h2>

    
    <img [src]="'imgUrl'" alt="" />
')
export class DeuxiemeComponent {
  texte :string =
  imgUrl :string =
    "lorem ipsum" ;
    "https://via.placeholder.com/200x100.png" ;
}
  
```

Rendu :



Remarques :

- Nouvelle syntaxe appelée **property binding** `[attribut]="variable class ts"`
- deuxième syntaxe pour faire de l'interpolation (plus longue donc moins pratique pour ajouter du texte)
- A utiliser pour modifier dynamiquement **les valeurs des attributs** des balises html
- utilisation du site <https://placeholder.com/> pour générer une image
- Attention, les informations sont **mono directionnel** : du composant vers le DOM

2 </> Attribute Binding

```

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

@Component({
  selector: 'deuxieme',
  template: '<h2>{{ texte }} </h2>
    <table>
      <tr>
        <td [colSpan]="'nbSpan"’>{{ texteLong }}</td>
      </tr>
      <tr>
        <td>{{ texte }}</td>
        <td>{{ texte }}</td>
      </tr>
    </table>
)
export class DeuxiemeComponent {
  texte :string =
  texteLong: string =
  nbSpan: number = 2 ;
}
  
```

- Attention la valeur entre les crochets est attribut du DOM :

`[attribut_dom] = "variable class ts"` qui n'est pas nécessairement égale à un attribut du HTML

- Liste des attributs DOM par Balise : [w3school](https://www.w3schools.com/html/html_attributes.asp)

HTML	DOM
<td colspan="2"></td>	querySelector("td").colSpan = 2

querySelector("td").colSpan = 2

lorem ipsum
texte long lorem ipsum
lorem ipsum lorem ipsum

3 </> Ajouter Bootstrap & Fontawesome

Ajouter Twitter Bootstrap

Intéressé par la version 3 de la librairie [Bootstrap version 3](#)

- [npm install bootstrap3](#) plus de détails
 - npm install jquery
 - Vérifier que le fichier **package.json** à de nouvelles dépendances
 - vérifier que les dossiers bootstrap3 et jquery sont désormais disponibles dans le dossier [nodes modules](#)
-

Utiliser la librairie dans votre projet

Ne pas mettre la librairie en direct dans le code source de **src/index.html** comme pour un projet en [html classique](#)

Deux possibilités :

dans le projet

- utilisation @import() fonction css dans src/app/styles.css
- utilisation de import { jquery } from ... module

dans [angular.json](#)

1. Arrêter le serveur ng
2. Modifier son fichier de configuration

```
...
"styles": [
  "./node_modules/bootstrap3/dist/css/bootstrap.min.css",
  "src/styles.css"
],
"scripts": [
  "./node_modules/jquery/dist/jquery.min.js",
  "./node_modules/bootstrap3/dist/js/bootstrap.js"
]
...
```

3. Redémarrer le serveur [ng serve -o](#)
 4. Remarquez que **styles.js** et **vendor.js** ont pris du poids, ils intègrent désormais ces deux nouvelles librairies
-

III Cas Pratiques

Ajouter Font Awesome

Intéressé par la version 4 de la librairie [Bootstrap version 4](#)

1. [npm install --save font-awesome angular-font-awesome](#)
plus d'informations

Fichier de configuration [angular.json](#)

```
...
"styles": [
  "./node_modules/font-awesome/css/font-awesome.css",
  "./node_modules/bootstrap3/dist/css/bootstrap.min.css",
  "src/styles.css"
],
...
```

4 </> Class Binding

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

@Component({
  selector: 'deuxieme',
  template: `
    <button class="btn btn-primary" [class.btn-xs]="isActif">
      Valider
    </button>
    <button class="btn btn-warning" [class.active]="isNotActif">
      Reset
    </button>
  `
})
export class DeuxiemeComponent {
  isActif : boolean = true;
  isNotActif : boolean = false;
}
```

Rendu

[Valider](#) [Reset](#)

Remarques :

- Documentation sur le bouton de [Twitter Bootstrap version 3](#)
- possibilité d'ajouter une class de manière dynamique à une balise html
- il faut respecter la syntaxe suivante :

[\[class.class_a_ajouter\]="valeur_boolean"](#)

5 </> Style Binding

```

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

@Component({
  selector: 'deuxieme',
  template: `
    <p [style.textAlign]="center"
       [style.color]="#ffcc51">lorem ipsum</p>
  `
})
export class DeuxiemeComponent {
  center:string = "center";
  pastel:string = "#ffcc51";
}

```

Rendu

lorem ipsum

Remarques :

- Liste de toutes les propriétés de style : [w3school](#)

6 </> Event Binding

```

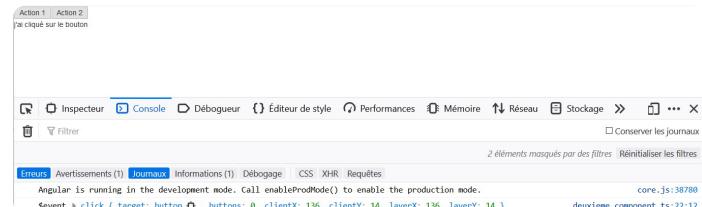
import { Component } from '@angular/core';

@Component({
  selector: 'deuxieme',
  template: `
    <button (click)="onSave()">Action 1</button>
    <button (click)="onEnregistrer($event)">Action 2 </button>
    <p>{ texte }</p>
  `
})
export class DeuxiemeComponent {
  texte:string;

  onSave() {
    this.texte = "j'ai cliqué sur le bouton";
  }

  onEnregistrer($event)
  {
    $event.stopPropagation();
    console.log("$event" ,$event);
  }
}

```

Rendu**Remarque :**

- Nouvelle syntaxe pour lier des événements `(event)=="method class()"`
- liste des événements du DOM : [w3school](#)
- `$event` : permet de récupérer l'événement du DOM déclenché

III Cas Pratiques

Utiliser \$event pour filtrer les touches saisies

```

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

@Component({
  selector: 'deuxieme',
  template: `
    <input (keyup)="onSaisie($event)" />
  `
})
export class DeuxiemeComponent {

  onSaisie($event)
  {
    //console.log($event);
    if($event.keyCode == 13){
      console.log("Enregistrer %s dans la BDD", $event.target.value);
    }
  }
}

```

7 </> Template variable

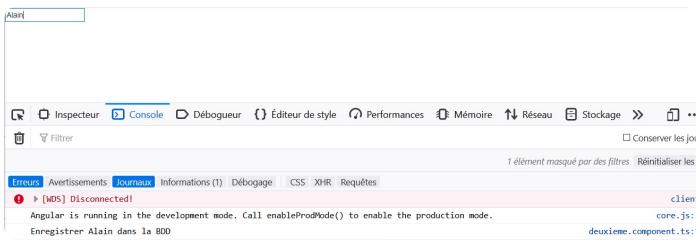
```

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

@Component({
  selector: 'deuxieme',
  template: `
    <input #nom (keyup)="onSaisie($event, nom.value)" />
  `
})
export class DeuxiemeComponent {

  onSaisie($event, val :string) {
    //console.log(val);
    if($event.keyCode == 13){
      console.log("Enregistrer "+val+" dans la BDD", val);
    }
  }
}

```

Rendu :**Remarque**

- Nouvelle syntaxe : `#[variable_template_name]` permet de sélectionner l'élément comme un `getById()`
- utilisable directement comme argument de fonction

8 </> Two Way Binding

```

//src/app/app.module.ts
import { PremierComponent } from './premier.component';
import { DeuxiemeComponent } from './deuxieme/deuxieme.component';
import { JoursService } from './jours.service';
import { DataService } from './data.service';
import { FormsModule } from "@angular/forms";

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

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

@NgModule({
  declarations: [
    AppComponent,
    PremierComponent,
    DeuxiemeComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [
    JoursService,
    DataService
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

```

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

@Component({
  selector: 'deuxieme',
  template: `
    <input [(ngModel)]="email" (keyup)="onSaisie($event)" />
  `
})
export class DeuxiemeComponent {

  email: string = "test@yahoo.fr";
  onSaisie($event) {
    //console.log(val);
    if($event.keyCode == 13){
      console.log(this.email);
    }
  }
}

```

Rendu :**Remarques**

Dans toutes les techniques précédentes, nous avons eu une modification monodirectionnel : de la class vers le DOM

Two way binding permet de modifier la class à partir d'une action dans le DOM

- il faut par défaut l'activer en modifiant le module root du composant sinon

Error: Template parse errors:
 Can't bind to 'ngModel' since it isn't a known property of 'input'. ("<input [ERROR ->][(ngModel)]="email" (keyup)="onSaisie(\$event)" />"):
 ng:///AppModule/DeuxiemeComponent.html@1:11
- Nouvelle syntaxe : `[(ngModel)]="variable_class"`, `[(())]` une banane dans une boîte
- Two way Binding = Data Binding (class => vue) + Event Binding (vue => class)

III Cas Pratiques

Vidéo excellente sur le sujet**Angular 8 Tutorial - 11 - Two Way Binding****9 </> Pipe****Rendu**

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

@Component({
  selector: 'deuxieme',
  template: `
    {{ article.titre | uppercase }} >br>
    {{ article.titre | lowercase }} >br>
    {{ article.titre | titlecase }} >br>
    {{ article.prix | number }} >br>
    {{ article.prix | currency:"EUR" }} >br>
    {{ article.prix | currency:"EUR":'symbol':'5.0-4' }} >br>
    {{ article.dt_publication | date:"dd/MM/yyyy" }} >br>
  `
})
export class DeuxiemeComponent {
  article = {
    titre : "Article n°1",
    prix : 2012.3,
    dt_publication: new Date(2001,1,2)
  };
}
```

ARTICLE N°1
article n°1
Article N°1
2,012.3
€2,012.30
€02,012.3
02/02/2001

Remarques

Pipe : Modifier les données avant d'être affichées dans la vue :

- documentation sur les Build In Pipe <https://angular.io/guide/pipes>
- [LowerCasePipe](#) | [UpperCasePipe](#) | [TitleCasePipe](#)
- [CurrencyPipe](#)
- [DatePipe](#)
- [DecimalPipe](#)

III Cas Pratiques**Article sur le sujet****Angular 8 Tutorial - 16 - Pipes****10 </> Pipe Personnalisé**

créer le fichier (`src/app/more.pipe.ts`) contenant le code suivant :

```
import { Pipe , PipeTransform} from "@angular/core";
@Pipe({
  name:"more"
})
export class MorePipe implements PipeTransform
{
  transform(text:string, limit?: number)
  {
    if (text.length == 0) return null;
    const vraiLimit = (limit) ? limit : 50;
    return text.substr(0,vraiLimit);
  }
}

@Pipe({
  name:"moreInit"
})
export class MoreInitPipe implements PipeTransform
{
  transform(text:any, args?: any)
  {
    //console.log(text);
    return text.substr(0,args);
  }
}
```

Register ces nouveaux pipes dans le root module

```
import { PremierComponent } from './premier.component';
import { DeuxiemeComponent } from './deuxieme/deuxieme.component';
import { JoursService } from './jours.service';
import { DataService } from './data.service';
import { FormsModule } from '@angular/forms';
import { MorePipe , MoreInitPipe } from './more.pipe';

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

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

@NgModule({
  declarations: [
    AppComponent,
    PremierComponent,
    DeuxiemeComponent,
    MorePipe,
    MoreInitPipe
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [
    JoursService,
    DataService
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

utiliser ces pipes personnalisés dans une vue

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

@Component({
  selector: 'deuxieme',
  template: `
    {{ article.titre | moreInit }} >br>
    {{ article.titre | more:20 }} >br>
  `
})
export class DeuxiemeComponent {

  article = {
    titre : `Lorem ipsum dolor sit amet,
    consectetur adipiscing elit. Duis
    feugiat ligula at mattis consequat.
  `;
}
```

Rendu

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis feugiat ligula at mattis consequat.
Lorem ipsum dolor si

Remarque

- Documentation officielle sur l'interface [PipeTransform](#)
- comme pour les services / les composants, les Pipe doivent être enregistrés dans le Module Root pour pouvoir être utilisé
- la syntaxe de création rappelle celle des services / des composants : nommage fichier / nommage class / ajout d'un décorateur
- transform() dispose de deux arguments :
 1. `value` : la donnée qui va être transformée
 2. `args` : des paramètres complémentaires sur le pipe personnalisé

III Cas Pratiques

Créer un composant favori

Créer un composant permettant de mettre un article en favori :

- par défaut, il s'agit d'une étoile vide : <https://fontawesome.com/v4.7.0/icon/star-o>



- lorsque l'internaute clique une fois dessus, l'étoile se remplit
<https://fontawesome.com/v4.7.0/icon/star>



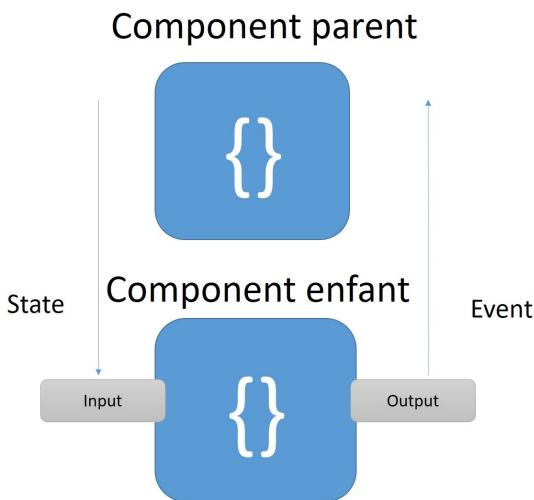
▲ ▾

malik.h@webdevpro.net

Jour 1	Jour 2	Jour 3	Jour 4	Jour 5	TP
1. Angular ?? 2. Les Fondamentaux 3. Données et Événements	1. Composant réutilisable 2. Directives 3. Template-driven Forms	1. Reactive Forms 2. Consommer des Services HTTP 3. Routing et Navigation	1. Authentification et Autorisation 2. Consommer des Services HTTP 3. Routing et Navigation	1. Déploiement	1. exos 2. fil rouge

Composant avancé

Documentation complète sur les Components Angular sur le site Officiel : angular.io



1 </> @Input

```
//src/app/app.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  article = {
    titre : "Article 1",
    contenu : "Contenu pour l'article 1"
  }
}
```

```
//src/app/app.component.html
<deuxieme [texte]="article.contenu"></deuxieme>
```

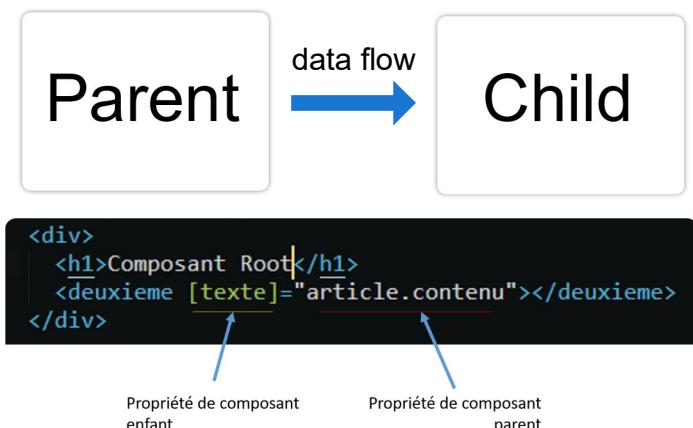
```
//src/app/deuxieme/deuxieme.component.ts
import { Component, Input } from '@angular/core';

@Component({
  selector: 'deuxieme',
  templateUrl: './deuxieme.component.html'
})
export class DeuxiemeComponent {
  @Input() texte : string ;
}

//src/app/deuxieme/deuxieme.component.html
<p>{{texte}}</p>
```

Le décorateur `@Input` permet d'**envoyer une donnée depuis le composant parent vers le composant enfant**

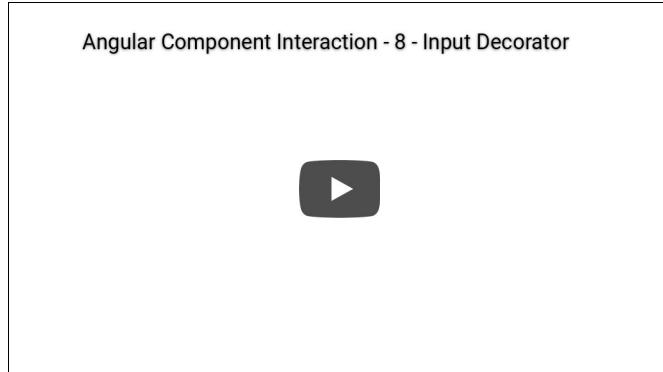
@Input



- Documentation officielle <https://angular.io/api/core/Input>
- Documentation officielle [Input et Output](#)

Cas Pratiques

Super vidéo sur le sujet



2 </> @Output

```
//src/app/app.component.ts
import { Component } from '@angular/core';

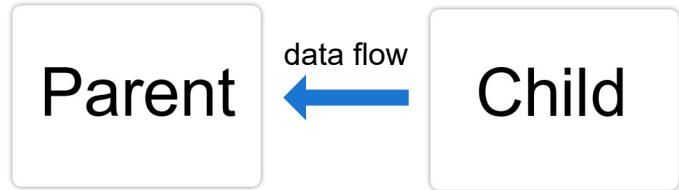
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {

  onDeuxiemeChange($event, titre)
  {
    titre.innerHTML = $event.prenom;
    //console.log($event)
    //console.log("Deuxieme a été modifié");
  }
}

//src/app/app.component.html
<h1 #titre>Composant Root</h1>
<deuxieme (change)="onDeuxiemeChange($event, titre)"></deuxieme>
```

Le décorateur `@Output` permet d'envoyer une donnée depuis le composant enfant vers le composant parent

@Output



Dans le emit de l'enfant

```
<div>
  <h1 #titre>Composant Root</h1>
  <deuxieme (change)="onDeuxiemeChange($event, titre)" ></deuxieme>
</div>
```

Propriété de composant enfant

méthode du composant parent

III Cas Pratiques

Output Décorateur**Angular Component Interaction - 13 - Output Decorator****3 </> Template externalisé / imbriqué**

Il existe deux manières d'associer un composant à un template

- Template externalisé
- Template imbriqué - embed

Il n'est pas possible d'utiliser les deux simultanément

Template externalisé

```
// src/app/deuxieme/deuxieme.component.ts
import { Component } from "@angular/core";

@Component({
  selector: 'app-deuxieme',
  templateUrl: './deuxieme.component.html',
  styleUrls: ['./deuxieme.component.css']
})
export class DeuxiemeComponent { }
```

Template imbriqué - embed

```
// src/app/deuxieme/deuxieme.component.ts
import { Component } from "@angular/core";

@Component({
  selector: 'app-deuxieme',
  template: `<h1>titre</h1>
    <p>un peu de texte</p>
    <p><a href="http://google.fr">Google</a></p>
  `,
  styleUrls: ['./deuxieme.component.css']
})
export class DeuxiemeComponent { }
```

Quelques remarques :

- Idéal pour les composants ayant plus de 5 lignes de balise html
- Appel le fichier `deuxieme.component.html` situé dans le même dossier que le fichier typescript

4 </> Style CSS externalisé / en ligne et priorités

Il existe deux manières d'associer un style à un composant

- Style externalisé
- Style en ligne - inline

Attention, ici il est possible d'utiliser les deux simultanément

Style externalisé

```
// src/app/deuxieme/deuxieme.component.ts
import { Component } from "@angular/core";

@Component({
  selector: 'app-deuxieme',
  templateUrl: './deuxieme.component.html',
  styleUrls: ['./deuxieme.component.css']
})
export class DeuxiemeComponent { }
```

Quelques remarques :

- Idéal pour les composants qui ont besoin de beaucoup de mise en forme
- Appel le fichier `deuxieme.component.css` situé dans le même dossier que le fichier typescript

Style imbriqué

```
// src/app/deuxieme/deuxieme.component.ts
import { Component } from "@angular/core";

@Component({
  selector: 'app-deuxieme',
  templateUrl: './deuxieme.component.html',
  styles: [
    `h1{ font-size : 10px }
     p{ color: blue }
   `
])
export class DeuxiemeComponent { }
```

Quelques remarques :

- Idéal pour les composants qui ont besoin de peu de mise en forme
- Utiliser les quotes grave pour utiliser les sauts de ligne
- Ne pas oublier les crochets

Les règles de priorités

Sachez sur plusieurs feuilles de styles vont affecter l'affichage à l'écran

1. Le fichier [/src/styles.css](#) : idéal pour ajouter des règles css globales qui vont impacter l'ensemble des composants
2. puis le style externalisé / imbriqué **en fonction de l'ordre de déclaration** dans le fichier typescript

Cas n°1

```
// src/app/deuxieme/deuxieme.component.ts
import { Component } from "@angular/core";

@Component({
  selector: 'app-deuxieme',
  templateUrl: './deuxieme.component.html',
  styleUrls: ['./deuxieme.component.css'],
  styles: [
    `h1{ font-size : 10px }
     p{ color: blue }
   `
])
export class DeuxiemeComponent { }
```

Quelques remarques

- ici on utilise les deux manières d'incorporer des styles simultanément
- ce sont les règles de [styles](#) qui seront prioritaires

Cas n°2

```
// src/app/deuxieme/deuxieme.component.ts
import { Component } from "@angular/core";

@Component({
  selector: 'app-deuxieme',
  templateUrl: './deuxieme.component.html',
  styles: [
    `h1{ font-size : 10px }
     p{ color: blue }
   `,
    styleUrls: ['./deuxieme.component.css']
})
export class DeuxiemeComponent { }
```

Quelques remarques

- ici on utilise les deux manières d'incorporer des styles simultanément
- ce sont les règles de [styleUrls](#) qui seront prioritaires

La portée des styles des composants

Dernière point important sur les styles associés aux composants : **ils ne s'appliquent que au composant concerné.**

Par défaut ils ne vont pas déborder sur les autres composants même si le sélecteur css devrait impacter un autre élément : concept de **Shadow DOM**

Plus de détail sur la méthode [createShadowRoot](#)

Exemple avec débordement d'une règle css

```
const el = document.querySelector("div");
el.innerHTML = `
<style>p{ font-size:10px; }</style>
<p>Bonjour !!</p>
`;
```

Exemple la méthode ShadowDOM

```
const el = document.querySelector("div");
const root = el.createShadowRoot();
root.innerHTML = `
<style>p{ font-size:10px; }</style>
<p>Bonjour !!</p>
`;
```

Angular va gérer ce concept nativement via une nouvelle métadonnée [encapsulation](#) de la fonction décoratrice [@Component\(\)](#)

Plus d'information sur [ViewEncapsulation](#)

Quelques remarques :

```
// src/app/deuxieme/deuxieme.component.ts
import { Component, ViewEncapsulation } from '@angular/core';

@Component({
  selector: 'app-deuxieme',
  templateUrl: './deuxieme.component.html',
  styles: [
    `h1,h2{ color : red }`
  ],
  encapsulation: ViewEncapsulation.Emulated
})
export class DeuxiemeComponent { }
```

- ViewEncapsulation dispose de trois attributs :
 1. **Emulated** : par défaut = utilisé si non appelé
 2. **Native** : utiliser le Shadow DOM du navigateur, attention les navigateurs anciens ne connaissent pas cette méthode
 3. **None** : ne pas utiliser le concept portée du style pour le composant, donc les styles définis dans le composant vont déborder sur les autres composants
- regarder l'inspecteur dans les différents cas

5 </> ngContent**Rendu**

```
//src/app/app.component.html
<deuxieme>
  <div class="head">
    <h1>Entête panneau</h1>
  </div>
  <div class="body">
    <h2>titre</h2>
    <p>Présentation : lorem ipsum</p>
  </div>
</deuxieme>
```

**Remarques :**

```
//src/app/deuxieme/deuxieme.component.html
<div class="panel panel-default">
  <div class="panel-heading">
    <ng-content select=".head"></ng-content>
  </div>
  <div class="panel-body">
    <ng-content select=".body"></ng-content>
  </div>
</div>
```

```
//src/app/deuxieme/deuxieme.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'deuxieme',
  templateUrl: './deuxieme.component.html'
})
export class DeuxiemeComponent { }
```

6 </> ngContainer**Rendu**

```
//src/app/app.component.html
<deuxieme>
  <ng-container class="head">
    <h1>Entête panneau</h1>
  </ng-container>
  <ng-container class="body">
    <h2>titre</h2>
    <p>Présentation : lorem ipsum</p>
  </ng-container>
</deuxieme>
```

**Remarques :**

```
//src/app/deuxieme/deuxieme.component.html
<div class="panel panel-default">
  <div class="panel-heading">
    <ng-content select=".head"></ng-content>
  </div>
  <div class="panel-body">
    <ng-content select=".body"></ng-content>
  </div>
</div>
```

```
//src/app/deuxieme/deuxieme.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'deuxieme',
  templateUrl: './deuxieme.component.html'
})
export class DeuxiemeComponent { }
```

- utilisation du composant Panel de Twitter Bootstrap : [Tuto w3c](#) - [Tuto quackit.com](#)
- utilisation de la balise Angular `<ng-content></ng-content>` dans le template du composant
- Dans la balise `<deuxieme> </deuxieme>` du template du composant root, imbriquer les éléments html à incorporer dans le template du composant
- autre manière de faire du databinding ...

malik.h@webdevpro.net

Jour 1	Jour 2	Jour 3	Jour 4	Jour 5	TP
1. Angular ?? 2. Les Fondamentaux 3. Données et Événements	1. Composant réutilisable 2. Directives 3. Template-driven Forms	1. Reactive Forms 2. Consommer des Services HTTP 3. Routing et Navigation	1. Authentification et Autorisation 2. Consommer des Services HTTP 3. Routing et Navigation	1. Déploiement	1. exos 2. fil rouge

Directives

1 </> ngIf

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

@Component({
  selector: 'deuxieme',
  template: `
    <div *ngIf="matieres.length > 0 ; else noMatiere">
      Liste de matières
    </div>
    <ng-template #noMatiere>
      Aucune matière disponible pour l'instant
    </ng-template>
  `
})
export class DeuxiemeComponent {
  matieres:Array<string | number> = ["js", "jQuery", "Node", 2];
}
```

La Directive `ngIf` modifie le DOM : supprimer / afficher une partie du HTML

Possibilité de mettre en place un else via la directive

```
<ng-template #template_variable></ng-template>
```

2 </> Hidden

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

@Component({
  selector: 'deuxieme',
  template: `
    <div [hidden]="matieres.length > 0">
      Liste
    </div>
    <div [hidden]="matieres.length == 0">
      Liste
    </div>
  `
})
export class DeuxiemeComponent {
  matieres:Array<string | number> = [1, 2];
}
```

- Permet de masquer un ou plusieurs éléments html
- petite liste de préférence

3 </> ngSwitch

```

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

@Component({
  selector: 'deuxieme',
  template: `
    <ul class="nav nav-pills">
      <li [class.active]="ongletEnCours == 'onglet1'">
        <a (click)="ongletEnCours = 'onglet1'">Onglet 1</a>
      </li>
      <li [class.active]="ongletEnCours == 'onglet2'">
        <a (click)="ongletEnCours = 'onglet2'">Onglet 2</a>
      </li>
    </ul>
    <div [ngSwitch]="ongletEnCours">
      <div *ngSwitchCase="'onglet1'">Contenu Onglet1</div>
      <div *ngSwitchCase="'onglet2'">Contenu Onglet2</div>
      <div *ngSwitchDefault>Autre cas</div>
    </div>
  `)
export class DeuxiemeComponent {
  ongletEnCours : string = "onglet1";
  //onClick(valeur)
  //{
  //  this.ongletEnCours = valeur;
  //}
}

```

Rendu



4 </> ngFor

```

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

@Component({
  selector: 'deuxieme',
  template: `
    <ul>
      <li *ngFor="let a of posts ; index as i ; even as isEven ">
        {{i}} - {{ a.titre }}
        <span *ngIf="isEven"> - (pair)</span>
      </li>
    </ul>
  `)
export class DeuxiemeComponent {
  posts : {id:number,titre:string}[] = [
    {id : 1, titre : "Article1"},
    {id : 2, titre : "Article2"},
    {id : 3, titre : "Article3"},
  ];
}

```

- index: number: The index of the current item in the iterable.
- first: boolean: True when the item is the first item in the iterable.
- last: boolean: True when the item is the last item in the iterable.
- even: boolean: True when the item has an even index in the iterable.
- odd: boolean: True when the item has an odd index in the iterable.

Documentation officielle : [NgForOf](#)

Tuto intéressant : [ngFor index & Groupe](#)

5 </> ngFor trackBy

```

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

@Component({
  selector: 'deuxieme',
  template: `
    <button (click)="onClick()">Charger articles</button>
    <ul>
      <li *ngFor="let a of posts ; trackBy: postCheck ">
        {{ a.titre }}
      </li>
    </ul>
  `)
export class DeuxiemeComponent {
  posts : {id:number,titre:string}[];
  onClick()
  {
    return this.posts = [
      {id : 1, titre : "Article1"},
      {id : 2, titre : "Article2"},
      {id : 3, titre : "Article3"},
    ];
  }
  postCheck(index, a)
  {
    return a ? a.id : undefined ;
  }
}

```

Eviter de recharger intégralement une liste

[NgForOf#ngForTrackBy](#)

regarder dans le

6 </> ngClass

```
import { Component } from '@angular/core';
@Component({
  selector: 'deuxieme',
  template: `
    <i class="fa" [ngClass]="{
      'fa-address-book': isActive ,
      'fa-address-book-o': !isActive
    }" aria-hidden="true" [style.fontSize]="'size'>
  `
})
export class DeuxiemeComponent {
  isActive: boolean = true;
  size : string = "40px";
}
```



- version améliorée de `[class.fa-address-book]="isActive"` et `[class.fa-address-book-o]="!isActive"`
- `isActive` doit être un boolean
- <https://fontawesome.com/v4.7.0/icons/>

7 </> ngStyle

```
import { Component } from '@angular/core';
@Component({
  selector: 'deuxieme',
  template: `
    <p [ngStyle]="{
      'color' : s.color,
      'fontSize' : texte ,
      'fontWeight' : s.weight,
      'textAlign' : s.align
    }">
      {{ texte }}
    </p>
  `
})
export class DeuxiemeComponent {
  texte : string = "40px";
  s = {
    color : "#ed0cef",
    align : "center",
    weight : "bold"
  }
}
```

40px

version améliorée :

- `[style.color] = "s.color"`
- `[style.fontSize] = "texte"`
- `[style.fontWeight] = "s.weight"`
- `[style.textAlign] = "s.align"`

8 </> Safe Traversal Operateur ?

```
import { Component } from '@angular/core';
@Component({
  selector: 'deuxieme',
  template: `
    <p [ngStyle]="{
      'color' : s.color,
      'fontSize' : texte ,
      'fontWeight' : s?.weight
    }">
      {{ texte }}
    </p>
  `
})
export class DeuxiemeComponent {
  texte : string = "40px";
  s = {
    color : "#ed0cef",
    weight : null
  }
}
```

Angular Safe Navigation Operator



9 </> Créer ses propres Directives

Directive : créer un nouvel attribut personnalisé

1. `ng generate directive image`

2. Créer la directive :

```
//src/app/image.directive.ts
import { Directive, ElementRef } from '@angular/core';

@Directive({
  selector: '[appImage]'
})
export class ImageDirective {

  private _url : string = "https://via.placeholder.com/150?text=logo" ;

  constructor(@ElementRef() elHTML: ElementRef) {
    elHTML.nativeElement.src = this._url ;
    elHTML.nativeElement.addEventListener("click",function(){
      this.classList.toggle("border");
    });
  }
}
```

3. utiliser la directive :

```
//src/app/app.component.html
<div>
  <h1>Directive Attribut</h1>
  <figure>
    <img appImage alt="">
  </figure>
</div>
```

Rendu

Directive Attribut



Créer une span paramétrable par le parent

1. `ng generate directive span-parametree`

2. Créer la directive :

```
src/app/span-parametree.directive.ts
import { Directive, ElementRef, Input } from '@angular/core';

@Directive({
  selector: '[appSpanParametree]'
})
export class SpanParametreeDirective {

  @Input('appSpanParametree') color: string;

  constructor(private el: ElementRef) { }

  ngOnInit() {
    this.el.nativeElement.style.backgroundColor = this.color ;
    this.el.nativeElement.style['color'] = "white" ;
    this.el.nativeElement.style['padding'] = "5px" ;
    this.el.nativeElement.style['border-radius'] = "5px" ;
  }
}
```

3. Utiliser la directive :

```
src/app/app.component.html
<h1>Directive Attribut paramétré</h1>
<div>
  <p>
    un peu de code
    <span [appSpanParametree]="#090088">
      var j = 20 ;
    </span>
  </p>
</div>
```

Rendu

Directive Attribut paramétré

un peu de code `var j = 20 ;`

Créer une directive paramétrable par le parent + écouteur Evenement

1. `ng generate directive btn`

```
{lamb} ng generate directive btn
CREATE src/app(btn.directive.spec.ts (212 bytes)
CREATE src/app(btn.directive.ts (135 bytes)
UPDATE src/app/app.module.ts (787 bytes)
```

2. La Directive :

```
src/app/button.directive.ts
import { Directive, ElementRef
        , Input, HostListener } from '@angular/core';

@Directive({
  selector: '[appBtn]'
})
export class ButtonDirective {
  private clicked: boolean = false;

  @Input('appBtn') color: string;

  @HostListener('click')
  onClick() {
    if(this.clicked) {
      this.el.nativeElement.style.backgroundColor = this.color;
      this.el.nativeElement.style['color'] = "white";
      this.el.nativeElement.style['padding'] = "5px";
      this.el.nativeElement.style['border-radius'] = "5px";
      this.clicked = false;
    }
    else {
      this.el.nativeElement.style.backgroundColor = "white";
      this.el.nativeElement.style['color'] = "black";
      this.el.nativeElement.style['padding'] = "5px";
      this.el.nativeElement.style['border-radius'] = "5px";
      this.clicked = true;
    }
  }

  constructor( private el: ElementRef ) { }
}
```

3. Utilisation :

```
//src/app/app.component.html
<h1>Directive Attribut paramétré Event</h1>
<div>
  <button [appBtn]="#dbb6c6">Bouton html</button>
</div>
```

Rendu :

Directive Attribut paramétré Event



malik.h@webdevpro.net

Jour 1	Jour 2	Jour 3	Jour 4	Jour 5	TP
1. Angular ?? 2. Les Fondamentaux 3. Données et Événements	1. Composant réutilisable 2. Directives 3. Template-driven Forms	1. Reactive Forms 2. Consommer des Services HTTP 3. Routing et Navigation	1. Authentification et Autorisation	1. Déploiement	1. exos 2. fil rouge

Template Driven Forms

1 </> Formulaire Bootstrap

```
1. ng g c form-contact
```

```
2. //src/app/contact-form/contact-form.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'contact-form',
  templateUrl: './contact-form.component.html',
  styleUrls: ['./contact-form.component.css']
})
export class ContactFormComponent { }
```

```
3. //src/app/contact-form/contact-form.component.html
<form>
  <div class="form-group">
    <label for="email">Email :</label>
    <input type="email" class="form-control" id="email">
  </div>
  <div class="form-group">
    <label for="commentaire">Commentaire :</label>
    <textarea id="commentaire" class="form-control"></textarea>
  </div>
  <button type="submit" class="btn btn-default">Submit</button>
</form>
```

```
4. //src/app/app.component.html
<section class="container">
  <div class="row">
    <div class="col-sm-push-2 col-sm-8">
      <contact-form></contact-form>
    </div>
  </div>
</section>
```

Créer un nouveau module contenant un formulaire de contact :

```
(lamb) ng g c contact-form
CREATE src/app/contact-form/contact-form.component.html (27 bytes)
CREATE src/app/contact-form/contact-form.component.spec.ts (664 bytes)
CREATE src/app/contact-form/contact-form.component.ts (292 bytes)
CREATE src/app/contact-form/contact-form.component.css (0 bytes)
UPDATE src/app/app.module.ts (871 bytes)
```

Documentation sur Bootstrap

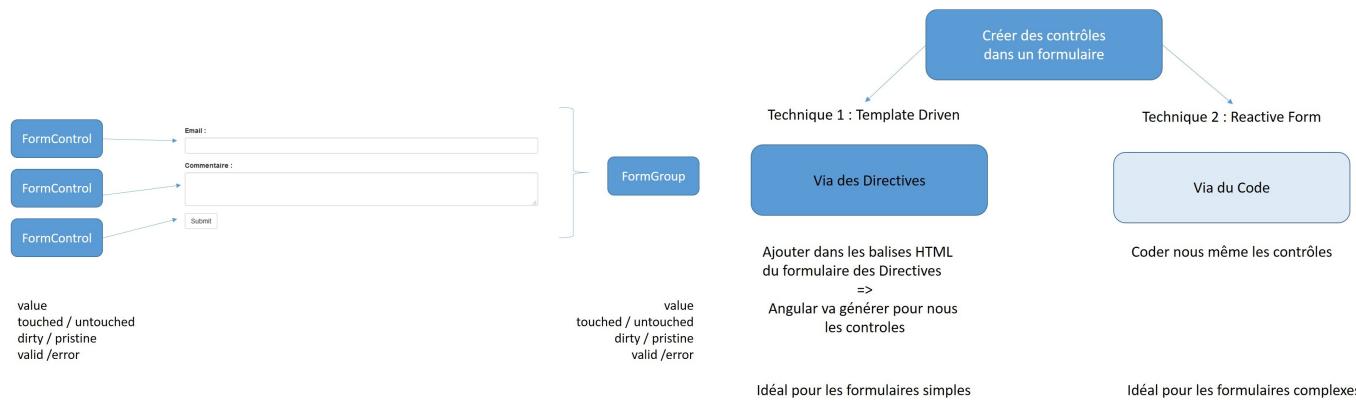
- Bootstrap Forms : [Forms](#)
- Bootstrap Inputs : [Inputs](#)
- Bootstrap Inputs 2 : [Inputs 2](#)
- Bootstrap Inputs Sizing : [Inputs Sizing](#)

Rendu :

Email :

Commentaire :

2 </> Template Driven Form Versus Reactive Form



3 </> ngModel - FormControl

```
//src/app/app.module.ts
import { ContactFormComponent } from './contact-form/contact-form.component';

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

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

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

```
//src/app/contact-form/contact-form.component.html
<form>
  <div class="form-group">
    <label for="email">Email :</label>
    <input type="email" class="form-control" id="email" name="email" ngModel #email="ngModel" (change)="onChangeEmail(email)">
  </div>
  <div class="form-group">
    <label for="commentaire">Commentaire :</label>
    <textarea name="commentaire" id="commentaire" class="form-control" ngModel></textarea>
  </div>
  <button type="submit" class="btn btn-default">Submit</button>
</form>
```

```
//src/app/contact-form/contact-form.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'contact-form',
  templateUrl: './contact-form.component.html',
  styleUrls: ['./contact-form.component.css']
})
export class ContactFormComponent {
  onChangeEmail(x)
  {
    console.log(x);
  }
}
```

- Pour pouvoir utiliser la directive `ngModel`, il faut au préalable d'importer dans le module root le module `FormsModule`
- Cette directive, `ngModel`, a déjà été vue : [Données et Événements > Two way Binding](#)
- Mais, ici, il ne faut pas utiliser la notation `[(ngModel)]="propriete class"` - Banana in a Box
- Elle s'écrit directement sans crochets et sans parenthèses

- Comme annoncé, Template Driven ajoutera dans le template HTML directement la Directive `ngModel` et ne pas oublier l'attribut `name` pour distinguer les différents contrôles du formulaire
- voir ce que contient `ngModel` via une variable de template `#email="ngModel"`



Documentation officielle : [NgModel](#)

4 </> Validation et messages d'erreur

```
//src/app/contact-form/contact-form.component.html
<form>
  <div class="form-group">
    <label for="email">Email :</label>
    <input type="email"
           class="form-control"
           id="email"
           name="email"
           ngModel
           #email="ngModel" (change)="onChangeEmail(email)">
    <div class="alert alert-danger" *ngIf="email.errors.required">Email requis</div>
    <div *ngIf="email.errors.pattern">Doit être de la forme nom@domaine.fr</div>
    <div *ngIf="email.errors.minLength">10 caractères minimum</div>
    <div *ngIf="email.errors.maxLength">250 caractères maximum</div>
  </div>
  <div class="form-group">
    <label for="commentaire">Commentaire :</label>
    <textarea
           class="form-control"
           id="commentaire"
           name="commentaire"
           #commentaire="ngModel"
           ngModel
           required
           minlength="10"
           maxlength="50"
           pattern="^([a-zA-Z.\\s]+)+@[a-zA-Z.-]+\\.[a-zA-Z]{2,4}$">
      <div class="alert alert-danger" *ngIf="commentaire.touched && !commentaire.valid">Commentaire requis</div>
    </div>
  </div>
  <button type="submit" class="btn btn-default">Submit</button>
</form>
```

- Driven Template Form : ajouter des attributs à la balise input qui vont être utilisés par la directive `ngModel`
- Affecter la variable de template ngModel : `#commentaire="ngModel"`
- liste des attributs disponibles pour la balise `<input>` : [W3C](#)



5 </> ngForm - FormGroup

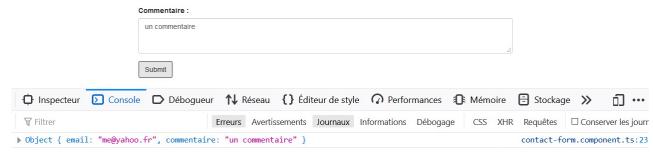
```
//src/app/contact-form/contact-form.component.html
<form #f=>ngForm (submit)="onFrmSubmit(f)">
  <div class="form-group">
    <label for="email">Email :</label>
    <input type="email"
      class="form-control"
      id="email"
      name="email"
      ngModel
      #email="ngModel" (change)="onChangeEmail(email)"
      required
      minlength="10"
      maxlength="50"
      pattern="[a-zA-Z-.%+-]+@[a-zA-Z.-]+\.[a-zA-Z]{2,4}$">
    <div *ngIf="email.errors.required" class="alert alert-danger">Email requis</div>
    <div *ngIf="email.errors.pattern">Doit être de la forme nom@domaine.fr</div>
    <div *ngIf="email.errors.minLength">10 caractères minimum</div>
    <div *ngIf="email.errors.maxLength">50 caractères maximum</div>
  </div>
  <div class="form-group">
    <label for="commentaire">Commentaire :</label>
    <textarea
      class="form-control"
      id="commentaire"
      name="commentaire"
      #commentaire="ngModel"
      ngModel
      required
      minlength="10"
      maxlength="50">
    </textarea>
    <div class="alert alert-danger" *ngIf="commentaire.touched && !commentaire.valid">
      <div *ngIf="commentaire.errors.required">Commentaire requis</div>
      <div *ngIf="commentaire.errors.minLength">10 caractères minimum</div>
      <div *ngIf="commentaire.errors.maxLength">250 caractères maximum</div>
    </div>
  </div>
  <button type="submit" class="btn btn-default">Submit</button>
</form>
```

```
//src/app/contact-form/contact-form.component.ts
import { Component } from '@angular/core';
```

```
@Component({
  selector: 'contact-form',
  templateUrl: './contact-form.component.html',
  styleUrls: ['./contact-form.component.css']
})
export class ContactFormComponent {
  onChangeEmail( x )
  {
    console.log(x);
  }

  onFrmSubmit(f)
  {
    //console.log(f)
    if(!f.valid)
    {
      console.log("des champs non conformes avant validation");
      return ;
    }

    console.log(f.value);
    // réaliser des traitements comme
    // enregistrer le contenu dans une BDD via une API
    // vider le formulaire
    // réaliser une redirection
  }
}
```



Documentation officielle : [NgForm](#)

6 </> Bloquer le bouton Submit tant que tout n'est pas conforme

```
<form #f="ngForm" (submit)="onFrmSubmit(f)">
<div class="form-group">
  <label for="email">Email :</label>
  <input type="email"
    class="form-control"
    id="email"
    name="email"
    ngModel
    #email="ngModel"
    (change)="onChangeEmail(email) "
    required
    minlength="10"
    maxlength="50"
    pattern="^([a-zA-Z_\\%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,4})$">
  <div class="alert alert-danger" *ngIf="email.touched && !email.valid">
    <div *ngIf="email.errors.required">Email requis</div>
    <div *ngIf="email.errors.pattern">Doit être de la forme nom@domaine.fr</div>
    <div *ngIf="email.errors.minLength">10 caractères minimum</div>
    <div *ngIf="email.errors.maxLength">250 caractères maximum</div>
  </div>
</div>
<div class="form-group">
  <label for="commentaire">Commentaire :</label>
  <textarea
    class="form-control"
    id="commentaire"
    name="commentaire"
    #commentaire="ngModel"
    ngModel
    required
    minlength="10"
    maxlength="50"
    ></textarea>
  <div class="alert alert-danger" *ngIf="commentaire.touched && !commentaire.valid">
    <div *ngIf="commentaire.errors.required">Commentaire requis</div>
    <div *ngIf="commentaire.errors.minLength">10 caractères minimum</div>
    <div *ngIf="commentaire.errors.maxLength">50 caractères maximum</div>
  </div>
</div>
<button type="submit" class="btn btn-default" [disabled]="!f.valid">Submit</button>
```

- liste des attributs disponibles pour la balise `<input>` : [W3C](#)

7 </> Checkbox, RadioBox et Liste déroulante

```
// utilise Bootstrap3
<form #f=>ngForm (submit)="onFrmSubmit(f)">
  <div class="form-group checkbox">
    <p>Recevoir la newsletter ??</p>
    <label>
      <input type="checkbox" value="isNewsletter" ngModel name="isNewsletter">Recevoir Newsletter
    </label>
  </div>
  <div class="form-group">
    <p>Vous êtes ?</p>
    <label classe="radio-inline">
      <input type="radio" name="sexe" value="1" ngModel required #sexe>Homme
    </label>
    <label classe="radio-inline">
      <input type="radio" name="sexe" value="2" ngModel required #sexe>Femme
    </label>
    <div class="alert alert-danger" *ngIf="!sexe.valid && sexe.touched">
      Champ Obligatoire
    </div>
  </div>
  <div class="form-group">
    <label for="pays"> Votre pays de résidence ?</label>
    <select class="form-control" id="pays" name="pays" ngModel required #pays="ngModel" >
      <option value=""> Sélectionner un pays
    </option>
    <option *ngFor="let p of paysList" [value]="p.id"> {{p.nom}}
    </option>
    </select>
    <div class="alert alert-danger" *ngIf="pays.touched && pays.value.length == 0">
      Veuillez sélectionner un pays
    </div>
  </div>
  <div class="form-group" (f.value | json)>
    <button type="submit" class="btn btn-default" [disabled]!="f.valid">
      Submit
    </button>
  </div>
</form>
```

une solution avec Bootstrap4

III Cas Pratiques

Réaliser le formulaire suivant :

Nom (*)	<input type="text"/>
Prénom (*)	<input type="text"/>
Mail (*)	<input type="text"/>
Numéro de téléphone (*)	<input type="text"/>
Poste souhaité (*)	<input type="text"/>

Documentation sur Bootstrap

- Bootstrap Forms : [Forms](#)
- Bootstrap Inputs : [Inputs](#)
- Bootstrap Inputs 2 : [Inputs 2](#)
- Bootstrap Inputs Sizing : [Inputs Sizing](#)

Rendu :

The screenshot shows a form with the following fields:

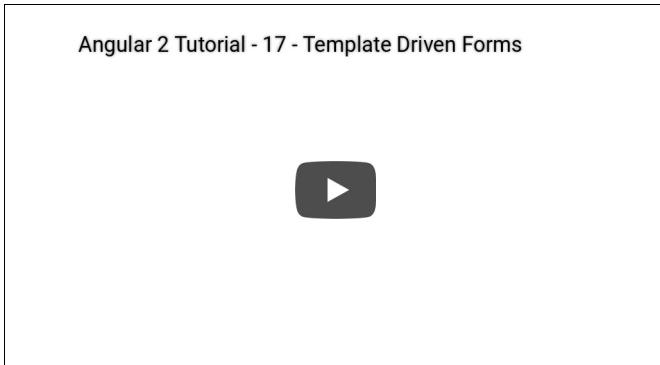
- A checkbox labeled "Recevoir la newsletter ??".
- A checkbox labeled "Recevoir Newsletter".
- A question "Vous êtes ?" followed by two radio buttons: "Homme" (selected) and "Femme".
- A dropdown menu labeled "Votre pays de résidence ?" with "France" selected.
- A submit button.

Below the form, there is a note: "Remarque : liste des pays à afficher dans le menu déroulant".

Remarque : liste des pays à afficher dans le menu déroulant

- France
- Allemagne
- Italie

Vidéo sur le sujet à refaire



File input

Complete File Upload and Download Tutorial using Angular...



Ajouter Quill dans un formulaire

Using Quill JS Text Editor With Angular 7



malik.h@webdevpro.net

Jour 1	Jour 2	Jour 3	Jour 4	Jour 5	TP
1. Angular ??	1. Composant réutilisable	1. Reactive Forms	1. Authentification et Autorisation	1. Déploiement	1. exos
2. Les Fondamentaux	2. Directives	2. Consommer des Services HTTP			2. fil rouge
3. Données et Événements	3. Template-driven Forms	3. Routing et Navigation			

HTTP Service

1 </> JSON Placeholder - notre API

Nous allons avoir besoin d'un Web Service (API) pour pouvoir réaliser cette section
 Deux possibilités :

- Développer soi-même sa propre API par exemple via NodeJS et une BDD MongoDB et la mettre en ligne
- Utiliser un site internet qui va nous donner une API comme [jsonplaceholder](#)

Dans le cadre de cette section, nous allons utiliser la deuxième solution mais vous pouvez à tout moment changer l'url du webservice par l'url de votre API

Plusieurs points d'accès (url) / ressources sont proposés jsonplaceholder, nous allons utiliser celui dédié aux posts (articles) :

<https://jsonplaceholder.typicode.com/posts>

The screenshot shows the JSONPlaceholder API documentation. At the top, there's a navigation bar with links for Guide, GitHub, and My JSON Server. Below it, a banner says "Announcement: You can now support JSONPlaceholder on GitHub Sponsors!". The main content is divided into two sections: "Resources" and "Routes".

Resources: Describes JSONPlaceholder as a set of 6 common resources: posts, comments, albums, photos, todos, and users. It includes a table of resource counts: posts (100), comments (500), albums (100), photos (5000), todos (200), and users (10). A note states: "Note: resources have relations. For example: posts have many comments, albums have many photos, ... see below for routes examples."

Routes: States that all HTTP methods are supported. It lists the following routes:

Method	Path
GET	/posts
GET	/posts/{id}
GET	/posts/{id}/comments
GET	/comments/{postId}{id}
GET	/posts/{userId}{id}
POST	/posts
PUT	/posts/{id}
PATCH	/posts/{id}
DELETE	/posts/{id}

A note at the bottom says: "Note: you can view detailed examples [here](#)".

2 </> CRUD version 1

```
//src/app/app.module.ts
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';

import { HttpClientModule } from '@angular/common/http';

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

```
//src/app/app.module.html
<div>
  <app-articles></app-articles>
</div>
```

1. Charger le module natif `HttpClientModule` dans le module root
2. créer un nouveau composant :

```
{lamb} ng g c articles
CREATE src/app/articles/articles.component.html (23 bytes)
CREATE src/app/articles/articles.component.spec.ts (642 bytes)
CREATE src/app/articles/articles.component.ts (277 bytes)
CREATE src/app/articles/articles.component.css (0 bytes)
UPDATE src/app/app.module.ts (651 bytes)
```

```

//src/app/articles/articles.component.ts
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Component({
  selector: 'app-articles',
  templateUrl: './articles.component.html',
  styleUrls: ['./articles.component.css']
})
export class ArticlesComponent implements OnInit {

  private _url : string = "https://jsonplaceholder.typicode.com/posts";
  private _articles ;

  constructor(private http: HttpClient) { }

  // Read
  ngOnInit() {
    this.http.get(this._url)
      .subscribe( (response : Response) => {
        //console.log(response)
        this._articles = response ;
      });
  }

  // Create
  onSubmitArticle( f )
  {
    //console.log(f.value)
    const article = f.value;
    this.http.post(this._url,JSON.stringify(article))
      .subscribe((response : Response) => {
        // que f.value soit conforme ou pas => toujours OK avec JSONPlaceholder
        // console.log(response);
        article["id"] = response['id']
        this._articles.splice(0,0,article) ;
      })
  }

  // Update
  onUpdateArticle(article)
  {
    this.http.put(this._url + `/${article.id}`,JSON.stringify(article))
      .subscribe((response : Response) => {
        // que article soit conforme ou pas => toujours OK avec JSONPlaceholder
        // par contre pas si vous essayez de modifier un article crée par vous => erreur 500
        console.log(response);
        article.title = article.title + " Modifié!";
      })
  }

  onDeleteArticle(article)
  {
    this.http.delete(this._url + `/${article.id}`)
      .subscribe((response : Response) => {
        // que article soit conforme ou pas => toujours OK avec JSONPlaceholder

        console.log(response);
        let index = this._articles.indexOf(article);
        this._articles.splice(index,1) ;
      })
  }
}

//src/app/articles/articles.component.html
<!-- <pre>{{ _articles | json}}</pre> -->
<div>
  <form #f="ngForm" (submit)="onSubmitArticle(f)">
    <div class="form-group">
      <label for="title">Titre</label>
      <input type="text" name="title" id="title" class="form-control" ngModel>
    </div>
    <div class="form-group">
      <label for="body">Contenu</label>
      <textarea name="body" id="body" class="form-control" ngModel></textarea>
    </div>
    <div>
      <input type="submit" value="Ajouter nouvel article" class="btn btn-primary">
    </div>
  </form>
</div>
<hr>
<section *ngFor="let article of _articles">
  <h2>{{ article.title }}</h2>
  <p>{{ article.body }}</p>
  <footer>
    <button (click)="onUpdateArticle(article)" class="btn btn-warning">Mis à jour</button>
    &nbsp;
    <button (click)="onDeleteArticle(article)" class="btn btn-danger">Suppression</button>
  </footer>
<hr>
</section>

```

3 </> CRUD version 2

```
//src/app/app.module.ts
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';

import { HttpClientModule } from "@angular/common/http";
import { PostsService } from "./posts.service";

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

1. Créer un service

```
{lamb} ng g s posts
CREATE src/app/posts.service.spec.ts (328 bytes)
CREATE src/app/posts.service.ts (134 bytes)
```

2. le configurer dans le module root

```
//src/app/posts.service.ts
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class PostsService {

  private _url : string = "https://jsonplaceholder.typicode.com/posts";

  constructor(private http: HttpClient) { }

  getPosts()
  {
    return this.http.get(this._url) ;
  }

  deletePosts(id)
  {
    return this.http.delete(this._url + `/ ${id}`) ;
  }

  addPost(post)
  {
    return this.http.post(this._url,JSON.stringify(post)) ;
  }

  updatePost(post)
  {
    return this.http.put(this._url + `/ ${post.id}`,JSON.stringify(post)) ;
  }

}

import { Component, OnInit } from '@angular/core';
import { PostsService } from "../posts.service";

@Component({
  selector: 'app-articles',
  templateUrl: './articles.component.html',
  styleUrls: ['./articles.component.css']
})
export class ArticlesComponent implements OnInit {

  private _articles ;

  constructor(private service : PostsService) { }

  // Read
  ngOnInit() {
    this.service.getPosts()
      .subscribe( (response : Response) => {
        //console.log(response)
        this._articles = response ;
      });
  }

  // Create
  onSubmitArticle( f )
  {
    //console.log(f.value)
    const article = f.value;
    this.service.addPost(article)
      .subscribe((response : Response) => {
        // que f.value soit conforme ou pas  => toujours OK avec JSONPlaceholder
        // console.log(response);
        article["id"] = response['id']
        this._articles.splice(0,0,article) ;
      })
  }

  // Update
  onUpdateArticle(article)
  {
    this.service.updatePost(article)
      .subscribe((response : Response) => {
        // que article soit conforme ou pas  => toujours OK avec JSONPlaceholder
        // par contre pas si vous essayez de modifier un article crée par vous => erreur 500
        console.log(response);
        article.title = article.title + " Modifié!";
      })
  }

  onDeleteArticle(article)
  {
    this.service.deletePosts(article.id)
      .subscribe((response : Response) => {
        // que article soit conforme ou pas  => toujours OK avec JSONPlaceholder

        console.log(response);
        let index = this._articles.indexOf(article);
        this._articles.splice(index,1) ;
      })
  }
}
```

4 </> Gestion des erreurs

Maintenant que nous arrivons à réaliser un CRUD, il faut l'améliorer et prendre en compte les erreurs.
Dans une application utilisant le réseau, deux grandes familles d'erreurs possibles :

Documentation officielle : [guide/http](#)

Erreur inattendues

1. Serveur contenant l'API est arrêté (offline)
2. Connexion Internet est coupée
3. Autres erreurs, par exemple bug sur le serveur

Erreurs attendus

1. Erreur 404 : ressource non trouvée (Not Found)
2. Erreur 400 : requête du client n'est pas conforme (Bad Request)

5 </> Gestion des erreurs version 1

```
//articles/articles.component.ts
import { Component, OnInit } from '@angular/core';
import { PostsService } from "../posts.service";

@Component({
  selector: 'app-articles',
  templateUrl: './articles.component.html',
  styleUrls: ['./articles.component.css']
})
export class ArticlesComponent implements OnInit {

  private _articles ;
  constructor(private service : PostsService) { }

  // Read
  ngOnInit() {
    this.service.getPosts()
      .subscribe( (response : Response) => {
        //console.log(response)
        this._articles = response ;
      },(error)=>{
        console.log( "Erreur inattendue", error);
      });
  }

  // Create
  onSubmitArticle( f )
  {
    //console.log(f.value)
    const article = f.value;
    this.service.addPost(article)
      .subscribe((response : Response) => {
        // que f.value soit conforme ou pas => toujours OK avec JSONPlaceholder
        // console.log(response);
        article["id"] = response['id']
        this._articles.splice(0,0,article) ;
      },(error : Response) => {

        if(error.status == 400 )
        {
          console.log( "Erreur 400", error);
        }
        else
        {
          console.log( "Erreur inattendue", error);
        }
      });
  }

  // Update
  onUpdateArticle(article)
  {
    this.service.updatePost(article)
      .subscribe((response : Response) => {
        // que article soit conforme ou pas => toujours OK avec JSONPlaceholder
        // par contre pas si vous essayez de modifier un article crée par vous => erreur 500
        console.log(response);
        article.title = article.title + " Modifié!";
      },(error : Response) => {
        if(error.status == 400 )
        {
          console.log( "Erreur 400", error);
        }
        else if( error.status == 404)
        {
          console.log( "Erreur 404", error);
        }
        else
        {
          console.log( "Erreur inattendue", error);
        }
      });
  }

  onDeleteArticle(article)
  {
    this.service.deletePosts(article.id)
      .subscribe((response : Response) => {
        // que article soit conforme ou pas => toujours OK avec JSONPlaceholder

        console.log(response);
        let index = this._articles.indexOf(article);
        this._articles.splice(index,1) ;
      },(error : Response) => {

        if( error.status == 404)
        {
          console.log( "Erreur 404", error);
        }
        else
        {
          console.log( "Erreur inattendue", error);
        }
      });
  }
}
```

6 </> Gestion des erreurs version 2

```
//src/app/posts.service.ts
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { retry, catchError } from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
})
export class PostsService {

  private _url : string = "https://jsonplaceholder.typicode.com/posts";
  constructor(private http: HttpClient) { }

  getPosts()
  {
    return this.http.get(this._url)
      .pipe(
        retry(1),
        catchError(this.handleError)
      );
  }

  deletePosts(id)
  {
    return this.http.delete(this._url + `/ ${id}`)
      .pipe(
        retry(1),
        catchError(this.handleError)
      );
  }

  addPost(post)
  {
    return this.http.post(this._url, JSON.stringify(post))
      .pipe(
        retry(1),
        catchError(this.handleError)
      );
  }

  updatePost(post)
  {
    return this.http.put(this._url + `/ ${post.id}`, JSON.stringify(post))
      .pipe(
        retry(1),
        catchError(this.handleError)
      );
  }

  private handleError(error) {
    let errorMessage = '';
    if (error.error instanceof ErrorEvent) {
      // client-side error
      errorMessage = `Error: ${error.error.message}`;
    } else {
      // server-side error
      errorMessage = `Error Code: ${error.status}\nMessage: ${error.message}`;
    }
    console.log(errorMessage);
    return throwError(errorMessage);
  }
}
```

```
//articles/articles.component.ts
import { Component, OnInit } from '@angular/core';
import { PostsService } from '../posts.service';

@Component({
  selector: 'app-articles',
  templateUrl: './articles.component.html',
  styleUrls: ['./articles.component.css']
})
export class ArticlesComponent implements OnInit {

  private _articles;

  constructor(private service: PostsService) { }

  // Read
  ngOnInit() {
    this.service.getPosts()
      .subscribe((response: Response) => {
        this._articles = response;
      });
  }

  // Create
  onSubmitArticle(f) {
    //console.log(f.value)
    const article = f.value;
    this.service.addPost(article)
      .subscribe((response: Response) => {
        article['id'] = response['id'];
        this._articles.splice(0, 0, article);
      });
  }

  // Update
  onUpdateArticle(article) {
    this.service.updatePost(article)
      .subscribe((response: Response) => {
        article.title = article.title + " Modifié!";
      });
  }

  onDeleteArticle(article) {
    this.service.deletePosts(article.id)
      .subscribe((response: Response) => {
        let index = this._articles.indexOf(article);
        this._articles.splice(index, 1);
      });
  }
}
```

7 </> Héritage - un service générique

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { retry, catchError } from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
})
export class DataService {

  constructor( private _url : string , private http: HttpClient) { }

  getAll()
  {
    return this.http.get(this._url)
      .pipe(
        retry(3),
        catchError(this.handleError)
      );
  }

  delete(id)
  {
    return this.http.delete(this._url + `/${id}`)
      .pipe(
        retry(3),
        catchError(this.handleError)
      );
  }

  add(ressource)
  {
    return this.http.post(this._url,JSON.stringify(ressource))
      .pipe(
        retry(3),
        catchError(this.handleError)
      );
  }

  update(ressource)
  {
    return this.http.put(this._url + `/ ${ressource.id}` ,JSON.stringify(ressource))
      .pipe(
        retry(3),
        catchError(this.handleError)
      );
  }

  private handleError(error) {
    let errorMessage = '';
    if (error.error instanceof ErrorEvent) {
      // client-side error
      errorMessage = `Error: ${error.error.message}`;
    } else {
      // server-side error
      errorMessage = `Error Code: ${error.status}\nMessage: ${error.message}`;
    }
    console.log(errorMessage);
    return throwError(errorMessage);
  }
}

//src/app/posts.service.ts
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

import { DataService} from "./data.service";

@Injectable({
  providedIn: 'root'
})
export class PostsService extends DataService {
  constructor(http: HttpClient) {
    super("https://jsonplaceholder.typicode.com/posts", http) ;
  }
}

import { Component, OnInit } from '@angular/core';
import { PostsService } from "../posts.service";

@Component({
  selector: 'app-articles',
  templateUrl: './articles.component.html',
  styleUrls: ['./articles.component.css']
})
export class ArticlesComponent implements OnInit {

  private _articles ;

  constructor(private service : PostsService) { }

  // Read
  ngOnInit() {
    this.service.getAll()
      .subscribe( (response : Response) => {
        this._articles = response ;
      });
  }

  // Create
  onSubmitArticle( f )
  {
    this.service.add(f.value)
  }
}
```

8 </> Intercepteur

```
// https://scotch.io/bar-talk/error-handling-with-angular-6-tips-and-best-practices192
//src/app/http-error.interceptor.ts
import {
  HttpEvent,
  HttpInterceptor,
  HttpHandler,
  HttpRequest,
  HttpResponse,
  HttpErrorResponse
} from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { retry, catchError } from 'rxjs/operators';

export class HttpErrorInterceptor implements HttpInterceptor {
  intercept(request: HttpRequest, next: HttpHandler): Observable<HttpEvent> {
    return next.handle(request)
      .pipe(
        retry(1),
        catchError((error: HttpErrorResponse) => {
          let errorMessage = '';
          if (error.error instanceof ErrorEvent) {
            // client-side error
            errorMessage = `Error: ${error.error.message}`;
          } else {
            // server-side error
            errorMessage = `Error Code: ${error.status}\nMessage: ${error.message}`;
          }
          window.alert(errorMessage);
          return throwError(errorMessage);
        })
      )
  }
}

//src/app/app.module.ts
//...
import { HttpClientModule , HTTP_INTERCEPTORS } from "@angular/common/http";
import { PostsService } from "./posts.service";
import { HttpErrorInterceptor } from './http-error.interceptor';

@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,
    HttpClientModule
  ],
  providers: [
    ArticlesService ,
    PostsService,
    {
      provide: HTTP_INTERCEPTORS,
      useClass: HttpErrorInterceptor,
      multi: true
    }
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }

//src/app/data.service.ts
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class DataService {

  constructor( private _url : string , private http: HttpClient) { }

  getAll()
  {
    return this.http.get(this._url);
  }

  delete(id)
  {
    return this.http.delete(this._url + `/ ${id}`);
  }

  add(resource)
  {
    return this.http.post(this._url,JSON.stringify(resource));
  }

  update(resource)
  {
    return this.http.put(this._url + `/ ${resource.id}` ,JSON.stringify(resource));
  }
}
```

Jour 1	Jour 2	Jour 3	Jour 4	Jour 5	TP
1. Angular ?? 2. Les Fondamentaux 3. Données et Événements	1. Composant réutilisable 2. Directives 3. Template-driven Forms	1. Reactive Forms 2. Consommer des Services HTTP 3. Routing et Navigation	1. Authentification et Autorisation 2. Consommer des Services HTTP	1. Déploiement	1. exos 2. fil rouge

Routing et Navigation

1 </> Présentation

Pour mettre en place une navigation dans un projet Angular, il faut réaliser 3 actions :

1. Configurer les routes = lier une url à un composant
2. ajouter un router outlet = où afficher le composant dans la page pour un url donné
3. Ajouter des liens = le déclencheur

créer 6 composants

1. navbar : `(ng g c navbar)`
2. home : `(ng g c home)`
3. list-article : `(ng g c list-article)`
4. one-article : `(ng g c one-article)`
5. contact-form : `(ng g c contact-form)`
6. not-found : `(ng g c not-found)`

Le cas pratique pour illustrer

arborescence

```
page d'accueil
  |_ Liste des articles
  |
  |_ voir un article
  |
  |_ Contact

NotFound
```

Documentation officielle : [RouterModule](#)

navbar

```
//src/app/navbar/navbar.component.html
<nav class="navbar navbar-default">
  <div class="container-fluid">
    <div class="navbar-header">
      <a class="navbar-brand" href="#">Jour3</a>
    </div>
    <ul class="nav navbar-nav">
      <li class="active"><a href="#">Home</a></li>
      <li><a href="#">Liste articles</a></li>
      <li><a href="#">Contact</a></li>
    </ul>
  </div>
</nav>
```

list-article

```
//src/app/list-article/list-article.component.html
<div *ngFor="let article of articles">
  <h2>{{ article.titre }}</h2>
  <p><a href="#">Lire la suite ...</a></p>
  <hr>
</div>

//src/app/list-article/list-article.component.ts
import { Component, OnInit } from '@angular/core';
import { ArticlesService } from '../service/articles.service';

@Component({
  selector: 'app-list-article',
  templateUrl: './list-article.component.html',
  styleUrls: ['./list-article.component.css']
})
export class ListArticleComponent implements OnInit {

  articles ;

  constructor( private service : ArticlesService ) {

  }

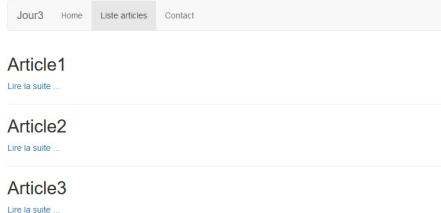
  ngOnInit() {
    this.articles = this.service.getArticles();
  }
}
```

Rendu Final :



Module NavBar de Twitter Bootstrap : [Tuto W3C](#)

Rendu Final :



one-article

```
//src/app/one-article/one-article.component.html
<h2>{{ article.titre }}</h2>
<p>{{ article.contenu }}</p>
<hr>
<h3>Laisser un commentaire :</h3>
<contact-form></contact-form>
<hr>
</div>
<button href="#" class="btn btn-primary" [disabled]="prevArticleId.length == 0">
    Article Précédent
</button>
-
<button href="#" class="btn btn-primary" [disabled]="nextArticleId.length == 0">
    Article Suivant
</button>
-
<button href="#" class="btn btn-warning">
    Retour à la liste des articles
</button>
</div>

//src/app/one-article/one-article.component.ts
import { Component, OnInit } from '@angular/core';
import { ArticlesService } from '../service/articles.service';

@Component({
  selector: 'app-one-article',
  templateUrl: './one-article.component.html',
  styleUrls: ['./one-article.component.css']
})
export class OneArticleComponent implements OnInit {

  private article ;
  private prevArticleId ;
  private nextArticleId ;

  constructor( private service : ArticlesService) {
  }

  ngOnInit() {
    this.article = this.service.getOneArticle(2)
    this.prevArticleId = 1 ;
    this.nextArticleId = 3 ;
  }
}
```

contact-form

```
//src/app/contact-form/contact-form.html
Reprendre le code dans
jour2 > Template-driven Forms > 6 Bloquer le bouton Submit

//src/app/contact-form/contact-form.ts
Reprendre le code dans
jour2 > Template-driven Forms > 5 ngForm - FormGroup
```

Rendu Final :

Article1
contenu article 1

Laisser un commentaire :

Email :

Commentaire :

Submit

Article Précédent Article Suivant Retour à la liste des articles

2 </> Configurer les routes

```
//src/app/app.module.ts

import { NavbarComponent } from './navbar/navbar.component';
import { HomeComponent } from './home/home.component';
import { ListArticleComponent } from './list-article/list-article.component';
import { ContactFormComponent } from './contact-form/contact-form.component';
import { OneArticleComponent } from './one-article/one-article.component';
import { NotFoundComponent } from './not-found/not-found.component';

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from "@angular/forms";
import { RouterModule } from "@angular/router";

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

@NgModule({
  declarations: [
    AppComponent,
    ContactFormComponent,
    NavbarComponent,
    HomeComponent,
    ListArticleComponent,
    OneArticleComponent,
    NotFoundComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    RouterModule.forRoot([
      {path: '', component: HomeComponent},
      {path: 'articles/:id', component: OneArticleComponent},
      {path: 'articles', component: ListArticleComponent},
      {path: 'contact', component: ContactFormComponent},
      {path: '**', component: NotFoundComponent}
    ])
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

- Importer le module Angular Router
- Paramétriser les routes de l'application = lier un url à un composant
 - Dans la section imports du décorateur `@NgModule`
 - Ajouter `RouterModule.forRoot()`
 - Cette méthode prend un JSON comme argument
 - chaque item du JSON est de la forme suivante
`{path : 'url', component: UnComposantComponent}`
- Attention à l'url de la page :
 1. d'accueil "
 2. pour afficher un et des article `articles 'articles/:id'` : `:id` est un paramètre
 3. Noter qu'il faut mettre la route la plus spécifique en premier
 4. si rien n'est trouvé : *** à mettre en dernière position

Documentation officielle : [RouterModule](#)

3 </> Ajouter un router outlet

```
//src/app/app.component.html
<section class="container">
  <div class="row">
    <div class="col-sm-push-2 col-sm-8">
      <nav></nav>
    </div>
    <div class="col-sm-push-2 col-sm-8">
      <router-outlet></router-outlet>
    </div>
  </div>
</section>
```

Remarque

- * pour la directive navbar = à l'intérieur
- * pour la directive router-outlet = à la suite

Documentation officielle : [RouterOutlet](#)

4 </> Ajouter des Liens - RouterLink

```
//src/app/navbar/navbar.component.html
<nav class="navbar navbar-default">
  <div class="container-fluid">
    <div class="navbar-header">
      <a class="navbar-brand" routerLink="/">Jour3</a>
    </div>
    <ul class="nav navbar-nav">
      <li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
        <a routerLink="/">Home</a>
      </li>
      <li routerLinkActive="active">
        <a routerLink="/articles">Liste articles</a>
      </li>
      <li routerLinkActive="active">
        <a routerLink="/contact">Contact</a>
      </li>
    </ul>
  </div>
```

Pour les routes sans paramètre : `routerLink="/url"`

* Single Page Application (SPA) : l'application ne se charge qu'une seule fois

Pour générer la class `active`, le RouterModule fournit une directive qui permet de savoir sur quel url nous sommes `routerLinkActive`

Pour les routes avec un paramètre, il faut utiliser la syntaxe suivante :

1. `[routerLink]` : routerLink avec la syntaxe property binding `[]`
2. `=`
3. une expression avec deux éléments : `['/url'], paramètre url`

Documentation officielle : [RouterLink](#)

5 </> Récupérer paramètre d'url

```
//src/app/one-article/one-article.component.ts
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { ArticlesService } from './service/articles.service';

@Component({
  selector: 'app-one-article',
  templateUrl: './one-article.component.html',
  styleUrls: ['./one-article.component.css']
})
export class OneArticleComponent implements OnInit {

  private article;
  private prevArticleId;
  private nextArticleId;

  constructor(
    private service: ArticlesService,
    private route: ActivatedRoute,
    private router: Router
  ) {}

  ngOnInit() {
    this.route.paramMap
      .subscribe((params) => {
        //console.log(params);
        const id = params.get('id');
        const article = this.service.getOneArticle(id);
        if(article != null) {
          this.article = article;
          this.prevArticleId = this.service.previousOneArticleId(id);
          this.nextArticleId = this.service.nextOneArticleId(id);
        }
        else {
          this.router.navigate(['/not-found']);
        }
      });
  }
}
```

Pour récupérer le paramètre de l'url :

- Importer le service `ActivatedRoute`
- Dans `ngOnInit() {}` Utilisation de la `this.route.paramMap` - Observable, pour récupérer le paramètre de route `:id`
- Il dispose d'une méthode `subscribe()` dont l'argument est une fonction
- Observable rappelle beaucoup les Promise

```
//src/app/one-article/one-article.component.html
<h2>{{ article.titre }}</h2>
<p>{{ article.contenu }}</p>
<hr>
<h3>Laisser un commentaire :</h3>
<contact-form></contact-form>
<hr>
<div>
  <button
    [routerLink]=["'/articles', prevArticleId ]"
    class="btn btn-primary"
    [disabled]="prevArticleId.length == 0">
    Article Précédent
  </button>
  -
  <button
    [routerLink]=["'/articles',nextArticleId]"
    class="btn btn-primary"
    [disabled]="nextArticleId.length == 0">
    Article Suivant
  </button>
  -
  <button
    routerLink="/articles"
    class="btn btn-warning">
    Retour à la liste des articles
  </button>
</div>
```

6 </> Cas du multi paramètre

```
//src/app/app.module.ts
...
@NgModule({
  imports: [
    BrowserModule,
    FormsModule,
    RouterModule.forRoot([
      {path: '', component: HomeComponent},
      {path: 'articles/:id/:titre', component: OneArticleComponent},
      {path: 'articles', component: ListArticleComponent},
      {path: 'contact', component: ContactFormComponent},
      {path: '**', component: NotFoundComponent}
    ])
  ],
})

```

```
//src/app/list-article/list-article.component.html
<div *ngFor="let article of articles">
  <h2>{{ article.titre }}</h2>
  <p>
    <a [routerLink]="/articles", article.id,article.titre">
      Lire la suite ...
    </a>
  </p>
  <hr>
</div>
```

7 </> Url avec query

Rendu

- pour setter dans le template :

```
[queryParams]={query1: 'valeur', query2:'valeur'}
```

- pour récupérer dans la composant :

```
this.route.queryParams.subscribe( (query) => {})
```

- `queryParams` vient lui aussi de `ActivatedRoute`

```
//src/app/list-article/list-article.component.html
<div>
  <button
    routerLink="/articles"
    [queryParams]="{ order:btn.order }"
    [innerText]="btn.txt"
    class="btn btn-primary"
    (click)="onTrie()"
  >
</button>
</div>
<div *ngFor="let article of articles">
  <h2>{{ article.titre }}</h2>
  <p>
    <a [routerLink]="/articles", article.id ]>
      Lire la suite ...
    </a>
  </p>
  <hr>
</div>

//src/app/list-article/list-article.component.ts
import { Component, OnInit } from '@angular/core';
import { ArticlesService } from '../../service/articles.service';
import { ActivatedRoute } from '@angular/router';

@Component({
  selector: 'app-list-article',
  templateUrl: './list-article.component.html',
  styleUrls: ['./list-article.component.css']
})
export class ListArticleComponent implements OnInit {

  articles;

  btn = {
    txt: "Ordre décroissant",
    order: "desc"
  }

  constructor(
    private service: ArticlesService,
    private route: ActivatedRoute
  ) {}

  ngOnInit() {
    this.route.queryParams.subscribe( (query) => {
      this.articles = this.service.getArticles(query.order);
    })
  }

  onTrie()
  {
    this.route.queryParams.subscribe( (query) => {
      if(query.order == "desc")
        {
          this.btn = {
            txt: "Ordre croissant",
            order: "asc"
          };
        }
      else
        {
          this.btn = {
            txt: "Ordre décroissant",
            order: "desc"
          };
        }
    })
  }
}

//src/app/service/articles.service.ts
...
getArticles(order?:string)
{
  if(order != undefined) return this._articles.reverse();
  return this._articles;
}
...
```

III Cas Pratiques

Une page avec une pagination

Réaliser le composant suivant :



Remarques :

- Utilisez le composant `pagination` de twitter bootstrap pour la mise en forme
- Lorsque l'on va cliquer sur l'un bouton de la pagination, l'url de la page va être complété d'une query (`?page=[numero-page]`)
- Créer un service disposant de deux méthodes :
 - `getAll()` : récupérer l'ensemble des ressources
 - `getFromTo()` : récupérer l'ensemble les articles à afficher sur la page

[une solution](#)

8 </> Url avec paramètre et query simultanément subscribe à plusieurs Observable

```
//src/app/list-article/list-article.component.html
<div>
  <button
    routerLink="/articles"
    [queryParams]="{{ order:btn.order }}"
    [innerText]="'btn.txt'"
    class="btn btn-primary"
    (click)="onTrie()"
  >
</button>
</div>
<div *ngFor="let article of articles">
  <h2>{{ article.titre }}</h2>
  <p>
    <a [routerLink]="/articles", article.id ]"
      [queryParams]="{{showForm:article.comment }}"
    Lire la suite ...
  </a>
  </p>
  <hr>
</div>

//src/app/one-article/one-article.component.html
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router' ;
import { ArticlesService } from "../service/articles.service" ;
import { combineLatest } from "rxjs" ;

@Component({
  selector: 'app-one-article',
  templateUrl: './one-article.component.html',
  styleUrls: ['./one-article.component.css']
})
export class OneArticleComponent implements OnInit {

  private article ;
  private prevArticleId : number | string;
  private nextArticleId : number | string;
  private isCommentOpen ;

  constructor(
    private service : ArticlesService ,
    private route: ActivatedRoute , private router: Router) {
  }

  ngOnInit() {
    combineLatest([
      this.route.paramMap,
      this.route.queryParamMap
    ]).subscribe(( combined ) => {
      //console.log(combined)

      const id = combined[0].get('id');
      const article = this.service.getOneArticle(id);
      if(article != null)
      {
        this.article = article ;
        this.prevArticleId = this.service.previousOneArticleId(id) ;
        this.nextArticleId = this.service.nextOneArticleId(id) ;
      }
      else
      {
        this.router.navigate(['/not-found'])
      }
    })
  }
}
```

malik.h@webdevpro.net

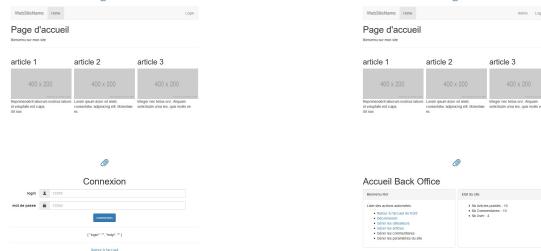
Jour 1	Jour 2	Jour 3	Jour 4	Jour 5	TP
1. Angular ??	1. Composant réutilisable	1. Reactive Forms	1. Authentification et Autorisation	1. Déploiement	exos
2. Les Fondamentaux	2. Directives	2. Consommer des Services HTTP			fil rouge
3. Données et Événements	3. Template-driven Forms	3. Routing et Navigation			

Authentification & Autorisation

1 </> Présentation

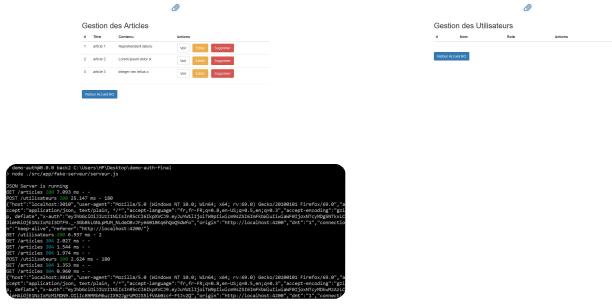
rendu final [demo-auth-final.zip](#)

- télécharger & dézipper
- [npm i](#)
- [ng start](#)
- Ouvrir un autre terminal
- [npm run back2](#) - [json-server](#) - [npm](#)
- lancer votre navigateur à l'adresse suivante : <http://localhost:4200>



Sommaire

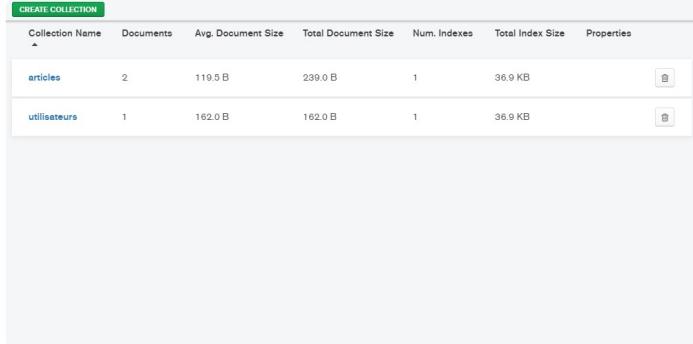
1. Architecture
2. JSON Web Tokens / local Storage
3. Connexion
4. Déconnexion
5. Afficher masquer des Elements
6. Afficher masquer des Elements en fonction du rôle
7. Utilisateur Actuel
8. CanActivate Interface
9. Redirection après connexion
10. Protection des routes



2 </> Back Office Node

starter Node : [node-blog.zip](#)

- Télécharger et dézipper
- [nodemon index.js](#)
- Lancer un autre terminal
- [mongod](#) : lancer la base de donnée MongoDB
- Lancer [postman](#)
- Lancer [MongoDBCompass](#)



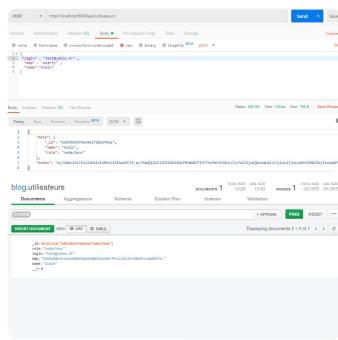
Description de l'API

Utilisateur :

- Créer un compte traducteur :**

méthode : POST
url : http://localhost:5000/api/utilisateurs
body :

```
{
  "login": "test@yahoo.fr",
  "mdp": "azerty",
  "name": "Alain"
}
```

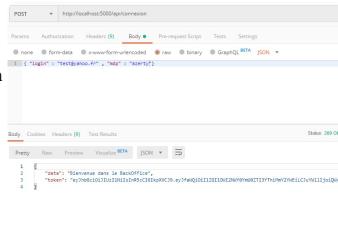


Connexion :

- Connexion au back office :**

méthode : POST
url : http://localhost:5000/api/connexion
body :

```
{
  "login": "test@yahoo.fr",
  "mdp": "azerty"
}
```



3 </> Starter Angular

starter Angular : [angular-auth-starter.zip](#)

1. `npm i`
2. `ng serve --open`

le résultat final de cette section est disponible [ici](#)

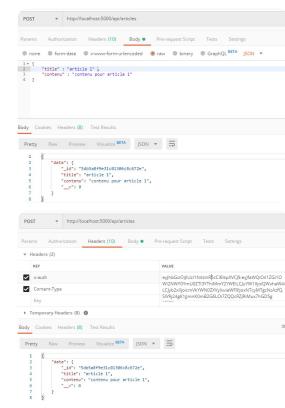
Générer le starter soit même

Articles :

- Créer un article :**

méthode : POST
url : http://localhost:5000/api/articles
head : **x-auth : valid JSONWEBTOKEN**
body :

```
{
  "title": "article 1",
  "contenu": "contenu pour article 1"
}
```



- Lire un article :**

méthode : GET
url : http://localhost:5000/api/articles/_id

- Lire tous les articles :**

méthode : GET
url : http://localhost:5000/api/articles

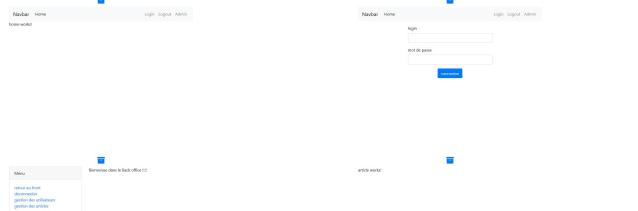
- Supprimer un article :**

méthode : DELETE
head : **x-auth : valid JSONWEBTOKEN**
url : http://localhost:5000/api/articles/_id

- Modifier un article :**

méthode : PUT
head : **x-auth : valid JSONWEBTOKEN**
url : http://localhost:5000/api/articles/_id

```
{
  "title": "article modifié",
  "contenu": "contenu pour article modifié"
}
```



4 </> Formulaire de connexion

```
src/app/front/login/login.component.html
<app-navbar></app-navbar>
<form
  class="col-sm-6 offset-sm-3 my-3"
  #formConnexion="ngForm"
  (submit)="onConnexion($event, formConnexion)"
>
  <div class="form-group">
    <label for="login">login</label>
    <input
      type="text"
      class="form-control"
      id="login"
      name="login"
      ngModel
      #login="ngModel"
    />
  </div>
  <div class="form-group">
    <label for="mdp">mot de passe</label>
    <input
      type="text"
      class="form-control"
      id="mdp"
      name="mdp"
      ngModel
      #mdp="ngModel"
    />
  </div>
  <div class="form-group text-center">
    <input type="submit" class="btn btn-primary" value="connexion" />
  </div>
</form>
<div class="text-center">
  {{ formConnexion.value | json }}
</div>
<div
  *ngIf="invalidLogin"
  class="alert alert-danger"
  role="alert"
>
  {{ errorMessage }}
</div>
```

5 </> Login Logout

- `npm i @auth0/angular-jwt`
- `ng g s service/auth`

```
//src/app/service/auth.service.ts
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Router } from '@angular/router';

import { map } from 'rxjs/operators';

import { JwtHelperService } from '@auth0/angular-jwt';

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  constructor(private http: HttpClient, private router: Router) {}

  url = 'http://localhost:5000/api/connexion';

  // d'habitude => CRUD

  login(credentials) {
    // appel à une API
    // observable
    // opérateur d'observable
    return this.http.post(this.url, credentials).pipe(
      map(resp => {
        if (resp['token']) {
          localStorage.setItem('token', resp['token']);
          return true;
        } else {
          return false;
        }
      })
    );
  }

  logout() {
    localStorage.removeItem('token');
  }

  logoutBack() {
    localStorage.removeItem('token');
    this.router.navigate(['/login']);
  }

  isLoggedIn() {
    let token = localStorage.getItem('token');

    if (!token) {
      return false;
    }

    // si on a un token en localstorage vérifier qu'il est bien conforme
    const helper = new JwtHelperService();
    const isExpired = helper.isTokenExpired(token);

    return !isExpired;
  }
}
```

```
src/app/front/login/login.component.ts
import { Component, OnInit } from '@angular/core';
import { AuthService } from './service/auth.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {
  constructor(private authService: AuthService, private router: Router) {}

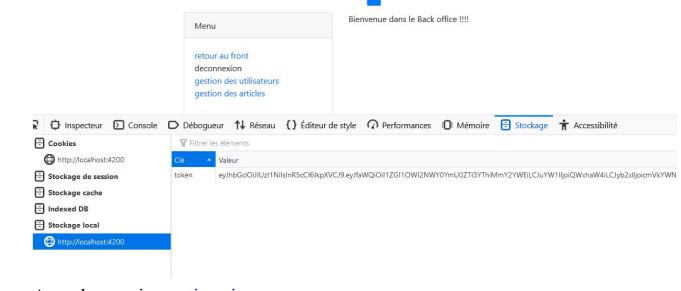
  ngOnInit() {}

  invalidLogin = false;
  errorMessage = "";

  onConnexion(Sevent, form) {
    Sevent.preventDefault();
    const credentials = form.value;

    // envoyer à un service qui va appeler notre projet node
    this.authService.login(credentials).subscribe(result => {
      if (result) {
        this.router.navigate(['/admin/home']);
      }
    },
    error => {
      this.invalidLogin = true;
      this.errorMessage = error.error.msg;
    });
  }
}
```

• **localStorage - [Documentation sur MDN](#)**



- Asynchrone via [reactivex.io](#)
- Librairie client pour traiter un JsonWebToken : [npmjs.com/@auth0/angular-jwt](#)

6 </> NavBar Front & Back

```

import { Component, OnInit } from '@angular/core';
import { AuthService } from './service/auth.service';

@Component({
  selector: 'app-navbar',
  templateUrl: './navbar.component.html',
  styleUrls: ['./navbar.component.css']
})
export class NavbarComponent implements OnInit {
  constructor(private authService: AuthService) {}

  ngOnInit() {}
}

//src/app/navbar/navbar.component.html
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <a class="navbar-brand" routerLink="/">Navbar</a>

  <div class="collapse navbar-collapse">
    <ul class="navbar-nav mr-auto">
      <li class="nav-item active">
        <a class="nav-link" routerLink="/">Home</a>
      </li>
    </ul>
    <ul class="nav navbar-nav navbar-right">
      <li class="nav-item" *ngIf="!authService.isLoggedIn()">
        <a class="nav-link" routerLink="/login">Login</a>
      </li>
      <li class="nav-item" *ngIf="authService.isLoggedIn()">
        <a class="nav-link" (click)="authService.logout()">Logout</a>
      </li>
      <li class="nav-item" *ngIf="authService.isLoggedIn()">
        <a class="nav-link" routerLink="/admin/home">Admin</a>
      </li>
    </ul>
  </div>
</nav>

src/app/app.module.ts
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
// ...
import { AuthService } from './service/auth.service';
// ...
@NgModule({
  declarations: [],
  imports: [],
  providers: [AuthService],
  bootstrap: [AppComponent]
})
export class AppModule {}

//src/app/admin/dashboard/dashboard.component.ts
import { Component, OnInit } from '@angular/core';
import { AuthService } from './service/auth.service';

@Component({
  selector: 'app-dashboard',
  templateUrl: './dashboard.component.html',
  styleUrls: ['./dashboard.component.css']
})
export class DashboardComponent implements OnInit {
  constructor(private authService: AuthService) {}

  ngOnInit() {}
}

//src/app/admin/dashboard/dashboard.component.html
<section class="row">
  <div class="col-5">
    <div class="card">
      <div class="card-header">
        Menu
      </div>
      <div class="card-body">
        <ul class="list-unstyled">
          <li><a routerLink="/">retour au front</a></li>
          <li><a (click)="authService.logoutBack()">deconnexion</a></li>
          <li><a routerLink="/admin/user">gestion des utilisateurs</a></li>
          <li><a routerLink="/admin/article">gestion des articles</a></li>
        </ul>
      </div>
    </div>
    <div class="col">
      <p>Bienvenue dans le Back office !!!!</p>
    </div>
  </div>
</section>

```

7 </> AuthGuardService

```

src/app/app.module.ts
// ...

• ng g s service/auth-guard

src/app/service/auth-guard.service.ts
import { Injectable } from '@angular/core';
import { CanActivate, Router } from '@angular/router';
import { AuthService } from './auth.service';

@Injectable({
  providedIn: 'root'
})
export class AuthGuardService implements CanActivate {
  constructor(private authService: AuthService, private router: Router) {}

  canActivate(route) {
    if (this.authService.isLoggedIn()) {
      return true;
    }

    this.router.navigate(['/login']);
    return false;
  }
}

Documentation Angular : CanActivate

```

8 </> Redirections

```

//src/app/front/login/login.component.ts
import { Component, OnInit } from "@angular/core";
import { AuthService } from "../../service/auth.service";
import { Router, ActivatedRoute } from "@angular/router";

//src/app/service/auth-guard.service.ts
import { Injectable } from '@angular/core';
import { CanActivate, RouterStateSnapshot } from '@angular/router';
import { AuthService } from './auth.service';

@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {

  constructor(private authService: AuthService, private router: Router) {}

  canActivate(route, state: RouterStateSnapshot) {
    if(this.authService.isLoggedIn()) return true;

    this.router.navigate(
      ['/login'],
      { queryParams: { return_url: state.url } }
    );
    return false;
  }
}

@Component({
  selector: "app-login",
  templateUrl: "./login.component.html",
  styleUrls: ["./login.component.css"]
})
export class LoginComponent implements OnInit {
  constructor(
    private authService: AuthService,
    private router: Router,
    private route: ActivatedRoute
  ) {}

  ngOnInit() {}

  invalidLogin = false;
  errorMessage = "";
  onConnexion($event, form) {
    $event.preventDefault();
    const credentials = form.value;

    // envoyer à un service qui va appeler notre projet node
    this.authService.login(credentials).subscribe(
      result => {
        if (result) {
          let returnUrl = this.route.snapshot.queryParamMap.get(
            "return_url"
          );
          this.router.navigate([
            returnUrl || "/admin/home"
          ]);
        } else this.invalidLogin = true;
      },
      error => {
        this.invalidLogin = true;
        this.errorMessage = error.error.msg;
      }
    );
  }
}

```

9 </> Nom de l'utilisateur connecté

```
//src\app\service\auth.service.ts
//...
export class AuthService {
//...
    getCurrentUser() {
        let token = localStorage.getItem('token');

        if (!token) return null;

        const helper = new JwtHelperService();
        return helper.decodeToken(token);
    }
}

//src\app\admin\dashboard\dashboard.component.ts
import { Component, OnInit } from '@angular/core';
import { AuthService } from '../../../../../service/auth.service';

@Component({
    selector: 'app-dashboard',
    templateUrl: './dashboard.component.html',
    styleUrls: ['./dashboard.component.css']
})
export class DashboardComponent implements OnInit {
    constructor(private authService: AuthService) { }

    name = this.authService.getCurrentUser().name;
    role = this.authService.getCurrentUser().role;

    ngOnInit() {
    }
}

//src\app\admin\dashboard\dashboard.component.html
//...
<p>Bienvenue {{ name }} dans le Back office !!!!</p>
```



10 </> Créer un nouvel article depuis le Back Office avec authentification

```
//src\app\admin\article\article.component.html
<form
  #formCreateArticle="ngForm"
  (submit)="onCreate($event, formCreateArticle)"
>
  <div class="form-group">
    <label for="title">Titre</label>
    <input
      type="text"
      name="title"
      class="form-control"
      ngModel
      #title="ngModel"
    />
  </div>
  <div class="form-group">
    <label for="contenu">Contenu</label>
    <textarea
      name="contenu"
      id="contenu"
      rows="10"
      class="form-control"
      ngModel
      #contenu="ngModel"
    ></textarea>
  </div>
  <div class="form-group">
    <input
      type="submit"
      class="btn btn-secondary"
      value="Créer"
    />
  </div>
</form>
<div
  *ngIf="success"
  class="alert alert-success"
  role="alert"
>
  Article créé !
</div>
<div *ngIf="error" class="alert alert-danger" role="alert">
  {{ errorMsg }}
</div>
<hr />
<div class="text-center">
  Retour au Dashboard
</div>
```

- [ng g s service/articles](#)

```
//src\app\front\login\login.component.ts
import { Component, OnInit } from '@angular/core';
import { AuthService } from '../../service/auth.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {
  constructor(
    private authService: AuthService,
    private router: Router
  ) {}

  ngOnInit() {}

  isValid = false;
  errorMessage = '';
  onConnexion($event, form) {
    $event.preventDefault();
    const credentials = form.value;

    // Envoyer à un service qui va appeler notre projet node
    this.authService.login(credentials).subscribe(
      result => {
        if (result) {
          this.router.navigate(['/admin/home']);
        }
      },
      error => {
        this.isValid = true;
        this.errorMessage = error.error.msg;
      }
    );
  }
}
```

```
import { Injectable } from '@angular/core';
import {
  HttpClient,
  HttpHeaders
} from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class ArticlesService {
  url = "http://localhost:5000/api/articles";

  constructor(private http: HttpClient) {}

  create(data) {
    let token = localStorage.getItem("token");
    const requestOptions = {
      headers: new HttpHeaders({
        "x-auth": token
      })
    };
    return this.http.post(this.url, data, requestOptions);
  }

  getAll() {
    return this.http.get(this.url);
  }
}
```

The screenshot shows a simple Angular component for creating an article. It consists of a form with two text inputs: one for the title and one for the content. Both inputs have placeholder text ('Titre' and 'contenu' respectively). Below the inputs is a blue 'Créer' button. In the bottom right corner of the page, there is a yellow 'Retour au Dashboard' button.

- Composant Bootstrap Alert
- Composant Bootstrap Form

11 </> Homepage du site

```
//src\app\front\home\home.component.ts
import { Component, OnInit } from '@angular/core';
import { ArticlesService } from './../../../service/articles.service';

@Component({
  selector: "app-home",
  templateUrl: "./home.component.html",
  styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnInit {
  articles;
  constructor(private articleService: ArticlesService) {}

  ngOnInit() {
    this.articles = this.articleService
      .getAll()
      .subscribe(result => {
        this.articles = result["data"];
      });
  }
}

//src\app\front\home\home.component.html
<app-navbar></app-navbar>
<section>
  <article *ngFor="let article of articles">
    <h2>{{ article.title }}</h2>
    
    <p>{{ article.contenu }}</p>
  </article>
</section>
```

```
section {
  display: grid;
  grid-template-columns: repeat(2, 1fr);
  grid-gap: 10px;
}

img {
  max-width: 100%;
}
p {
  white-space: pre-wrap;
}
```



malik.h@webdevpro.net

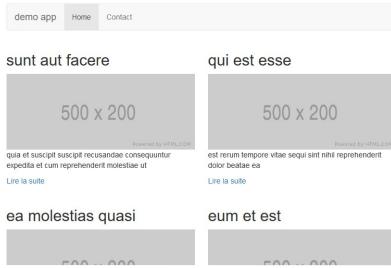
Jour 1	Jour 2	Jour 3	Jour 4	Jour 5	TP
1. Angular ??	1. Composant réutilisable	1. Reactif Forms	1. Authentification et Autorisation	1. Déploiement	1. exos
2. Les Fondamentaux	2. Directives	2. Consommer des Services HTTP			2. fil rouge
3. Données et Événements	3. Template-driven Forms	3. Routing et Navigation			

Mettre en ligne un projet Angular

1 </> Présentation de l'application à déployer

Dans cette dernière section, nous allons déployer une application Angular que j'ai réalisée :

1. Télécharger [demo-app.zip](#)
2. Dézipper le dossier
3. Dans le dossier dézippé, lancez la commande `(npm i)` qui va créer le dossier `node_modules` et télécharger toutes les librairies
4. Puis lancer `(ng serve -o)` pour voir l'aspect de l'application :
 - o Contient 3 pages : une page d'accueil + une page de contact + une page d'article
 - o Utilise un web service : [jsonplaceholder](#)
 - o Utilise un Pipe personnalisé : [MorePipe](#)



2 </> Techniques de déploiement

Avant de mettre notre projet sur un hébergeur, nous allons lister l'ensemble des techniques qui vont être mis en oeuvre pour optimiser le déploiement :

- Minification : supprimer les espaces / sans de ligne / commentaires
 - Uglification : remplacer les variables / fonctions / Class pour qu'elles aient le moins de caractères possibles
 - Bundling : concaténer l'ensemble des fichiers d'un même langage dans un seul
 - Eliminer le code mort : supprimer toutes les fonctions / class / variables non utilisées par notre application finale (mais potentiellement utiles lors de la phase de développement)
 - Ahead Of Time (AOT) Compilation : tous les fichiers html / css / ts / js vont être transformé en Javascript ready to read par les navigateurs.
- Toute la phase de transformation html / css / ts en bundle (lors de la phase de développement) n'est pas présente sur l'application que nous allons déployer sur l'hébergeur

`ng build --prod`

Toutes ces techniques d'optimisations vont être réalisées via **une seule commande**

3 </> ng build --prod

- Lancer la commande `(ng build --prod)`
- regarder le contenu du dossier demo-app
- un nouveau dossier **dist** (distribuable) a été créé
- regarder le contenu des différents fichiers : [index.html](#), les bundles ainsi que les fichiers [*.map](#)
- Enfin, remarquer que certains fichiers dispose d'un **hash** dans leur nom pour éviter que le navigateur les mettent en cache

```
C:\Users\HP\Desktop\demo-app>ng build --prod
chunk {0} runtime-es2015.85f895af57b038f1e5b4.js (runtime) 2.82 kB [entry] [rendered]
chunk {1} main-es2015.dd4a0ec8499e96a6fffb.js (main) 439 kB [initial] [rendered]
chunk {2} polyfills-es2015.0fe6949bc5ff4ab784062.js (polyfills) 64 kB [initial] [rendered]
chunk {3} polyfills-es5-es2015.a83ac866abc867bfd530.js (polyfills-es5) 222 kB [initial] [rendered]
chunk {4} styles.aaccee5669538ad468e3.css (styles) 112 kB [initial] [rendered]
Date: 2019-09-19T19:40:55.234Z - Hash: d1db9ece491157d605f2 - Time: 35555ms
Generating ES5 bundles for differential loading...
ES5 bundle generation complete.

<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>DemoApp</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <link rel="stylesheet" href="styles.aaccee5669538ad468e3.css">
</head>
<body>
  <app-root></app-root>
<script src="polyfills-es5.a83ac866abc867bfd530.js" nomodule defer></script><script src="polyfills-es2015.0fe6949bc5ff4ab784062.js" type="module"></script><script src="runtime-es2015.85f895af57b038f1e5b4.js" type="module"></script><script src="main-es2015.dd4a0ec8499e96a6fffb.js" type="module"></script><script src="runtime-es5.85f895af57b038f1e5b4.js" nomodule defer></script><script src="main-es5.dd4a0ec8499e96a6fffb.js" nomodule defer></script></body>
</html>
```

4 </> Variable d'environnement

```
//src/environnements/environnement.ts
export const environment = {
  production: false,
  bg : "green"
};
```

```
//src/environnements/environnement.prod.ts
export const environment = {
  production: true,
  bg : "#f8f8f8"
};
```

```
//src/app/front/navbar/navbar.component.ts
import { Component, OnInit } from '@angular/core';
import { environment } from '../../../../../environments/environment';

@Component({
  selector: 'app-navbar',
  templateUrl: './navbar.component.html',
  styleUrls: ['./navbar.component.css']
})
export class NavbarComponent implements OnInit {

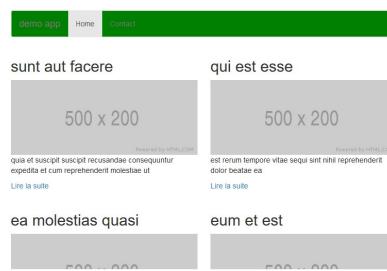
  bgColor = environment.bg;
  constructor() { }

  ngOnInit() {
  }

//src/app/front/navbar/navbar.component.html
<nav class="navbar navbar-default" [style.backgroundColor]="bgColor">
  <div class="container-fluid">
    <div class="navbar-header">
      <a class="navbar-brand" routerLink="/">demo app</a>
    </div>
    <ul class="nav navbar-nav">
      <li routerLinkActive="active" [routerLinkActiveOptions]="{{ exact: true }}>
        <a routerLink="/">Home</a>
      </li>
      <li routerLinkActive="active">
        <a routerLink="/contact">Contact</a>
      </li>
    </ul>
  </div>
```

utilisation des variables d'environnement pour distinguer rapidement un environnement de production finalisé et un environnement de test :

`ng serve -o`



`ng serve --prod -o`



5 </> tslint



Angular est livré avec un outil qui permet de normaliser le codage (et ainsi faciliter la passage d'une développeur à un autre) : [tslint](#)

- voir dans le fichier `package.json` la présence de la librairie dans la propriété `devDependencies`
- voir le fichier `tslint.json` à la racine du projet
- `ng lint` : voir l'ensemble de lignes qui ne sont pas conformes à la norme de codage
- `ng lint --fix` : réaliser toutes les rectifications simples sur le code

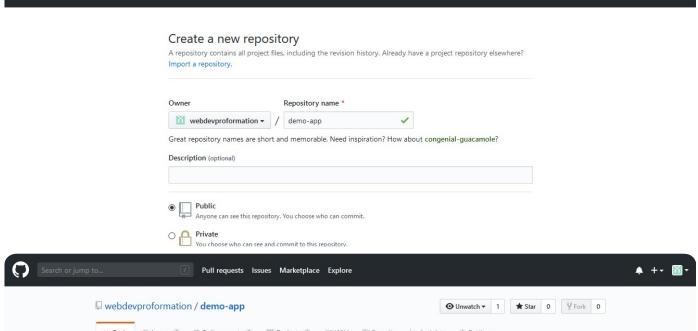
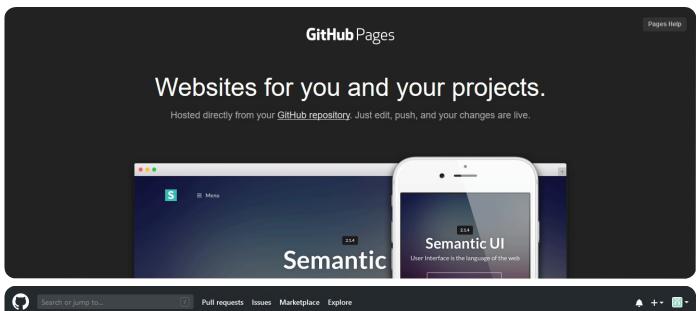
```
C:\Users\HP\Desktop\demo-app>ng lint
Linting "demo-app"...

ERROR: C:/Users/HP/Desktop/demo-app/src/app/app.module.ts:11:30 - " should be '
ERROR: C:/Users/HP/Desktop/demo-app/src/app/app.module.ts:12:29 - " should be '
ERROR: C:/Users/HP/Desktop/demo-app/src/app/app.module.ts:13:34 - " should be '
ERROR: C:/Users/HP/Desktop/demo-app/src/app/app.module.ts:15:33 - " should be '
ERROR: C:/Users/HP/Desktop/demo-app/src/app/front/article/article.component.ts:3:43 - " should be '
ERROR: C:/Users/HP/Desktop/demo-app/src/app/front/article/article.component.ts:13:13 - " should be '
ERROR: C:/Users/HP/Desktop/demo-app/src/app/front/article/article.component.ts:14:12 - " should be '
ERROR: C:/Users/HP/Desktop/demo-app/src/app/front/article/article.component.ts:17:4 - Missing semicolon
ERROR: C:/Users/HP/Desktop/demo-app/src/app/front/article/article.component.ts:19:22 - expected whitespace before colon
parameter
ERROR: C:/Users/HP/Desktop/demo-app/src/app/front/article/article.component.ts:19:42 - trailing whitespace
ERROR: C:/Users/HP/Desktop/demo-app/src/app/front/article/article.component.ts:20:38 - trailing whitespace
ERROR: C:/Users/HP/Desktop/demo-app/src/app/front/article/article.component.ts:21:3 - misplaced opening brace
ERROR: C:/Users/HP/Desktop/demo-app/src/app/front/article/article.component.ts:25:32 - missing whitespace
ERROR: C:/Users/HP/Desktop/demo-app/src/app/front/article/article.component.ts:25:32 - missing whitespace
ERROR: C:/Users/HP/Desktop/demo-app/src/app/front/article/article.component.ts:25:33 - trailing whitespace
ERROR: C:/Users/HP/Desktop/demo-app/src/app/front/article/article.component.ts:27:62 - missing whitespace
ERROR: C:/Users/HP/Desktop/demo-app/src/app/front/article/article.component.ts:27:62 - missing whitespace

C:\Users\HP\Desktop\demo-app>ng lint --fix
Linting "demo-app"...
Fixed 1 error(s) in C:/Users/HP/Desktop/demo-app/src/environments/environment.ts
Fixed 1 error(s) in C:/Users/HP/Desktop/demo-app/src/app/navbar/navbar.component.ts
Fixed 26 error(s) in C:/Users/HP/Desktop/demo-app/src/app/service/articles.service.ts
Fixed 4 error(s) in C:/Users/HP/Desktop/demo-app/src/app/front/home/home.component.ts
Fixed 8 error(s) in C:/Users/HP/Desktop/demo-app/src/app/front/contact/contact.component.ts
Fixed 4 error(s) in C:/Users/HP/Desktop/demo-app/src/app/front/footer/footer.component.ts
Fixed 4 error(s) in C:/Users/HP/Desktop/demo-app/src/app/pipe/more.pipe.ts
Fixed 16 error(s) in C:/Users/HP/Desktop/demo-app/src/app/front/article/article.component.ts
Fixed 4 error(s) in C:/Users/HP/Desktop/demo-app/src/app/app.module.ts
Fixed 1 error(s) in C:/Users/HP/Desktop/demo-app/src/environments/environment.prod.ts

ERROR: C:/Users/HP/Desktop/demo-app/src/app/front/home/home.component.ts:12:3 - variable name must
PascalCase or UPPER_CASE
ERROR: C:/Users/HP/Desktop/demo-app/src/app/service/articles.service.ts:17:11 - variable name must
PascalCase or UPPER_CASE
```

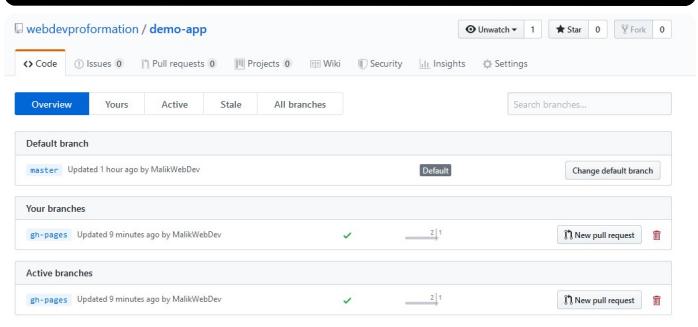
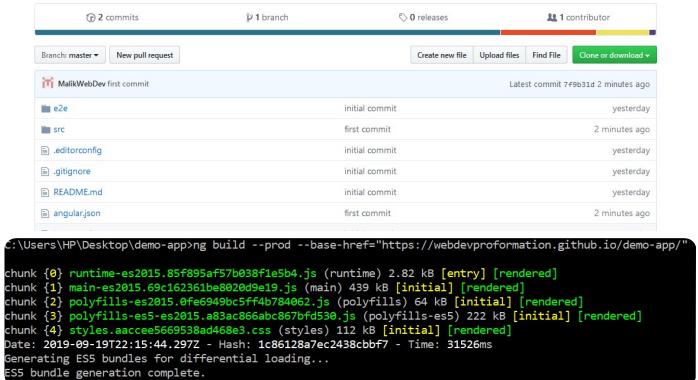
6 </> Déploiement sur une page github



Enfin nous allons mettre en ligne notre projet sur une page github :
<https://pages.github.com/>

1. se créer un compte sur [github](#)
2. se connecter et créer un repository
3. commandes git :
 - o `git init`
 - o `git add *`
 - o `git commit -m "lancement projet"`
 - o `git remote add origin https://github.com/votre-nom-git/demo-app.git`
 - o `git push -u origin master`
 - o le terminal vous demande votre login et mot de passe du compte github
4. Installer (`npm i -g angular-cli-ghpages`, [plus d'info sur npmjs.com](#))
5. (`ng build --prod --base-href="https://votre-nom-git.github.io/repository/"`
 (attention aux doubles quote et au slash final))
6. (`ngh --dir dist/[PROJECTNAME]`) ou
`angular-cli-ghpages --dir dist/[PROJECTNAME]`
7. <https://votre-nom-git.github.io/repository>
8. ou faire un script dans `package.json` à exécuter dans un gitbash
9. une nouvelle branch est créée intitulée gh-pages

```
{
  "name": "demo-app",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e",
    "deploy" : "ng build --prod --base-href=\"\` https://webdevproformation.github.io/demo-app/\`"
    && ngh --dir dist/demo-app"
  },
}
```



malik.h@webdevpro.net