

# Le développement FrontEnd avec JavaScript

## Module 6 - Les bases d'Angular



1

Les bases d'Angular

## Objectifs

- Maîtriser l'architecture d'Angular
- Être capable de créer une application Angular standard




2

## Historique

- 2010 :  ANGULARJS  
by Google

- Client riche
- Expérience utilisateur
- Basé sur JavaScript

- 2016 : 

- Refonte totale
- Basé sur TypeScript

Concurrents :   React

Base de  ionic



## Angular CLI

- Créer les projets
- Gérer les dépendances
- Construire les applications
- Tester les applications...

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



## Installation d'Angular CLI



```
npm install -g @angular/cli
```

Installation globale



## Les commandes d'Angular CLI

```
C:\Users\ >ng -help
```

Available Commands:

- add** Adds support for an external library to your project.
- build** (b) Compiles an Angular app into an output directory named dist/ at the given output path. Must be executed from within a workspace directory.
- config** Retrieves or sets Angular configuration values in the angular.json file for the workspace.
- doc** (d) Opens the official Angular documentation (angular.io) in a browser, and searches for a given keyword.
- e2e** (e) Builds and serves an Angular app, then runs end-to-end tests using Protractor.
- generate** (g) Generates and/or modifies files based on a schematic.
- help** Lists available commands and their short descriptions.
- lint** (l) Runs linting tools on Angular app code in a given project folder.
- new** (n) Creates a new workspace and an initial Angular app.
- run** Runs an Architect target with an optional custom builder configuration defined in your project.
- serve** (s) Builds and serves your app, rebuilding on file changes.
- test** (t) Runs unit tests in a project.
- update** Updates your application and its dependencies. See <https://update.angular.io/>
- version** (v) Outputs Angular CLI version.
- xi18n** Extracts i18n messages from source code.

For more detailed help run "ng [command name] --help"



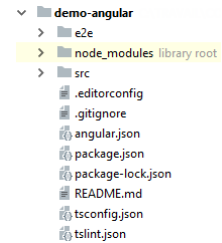
## Créer et exécuter un projet

- Créer un projet

```
ng new demo-angular
```

```
? Would you like to add Angular routing? No
```

```
? Which stylesheet format would you like to use? CSS
```



- Exécuter un projet

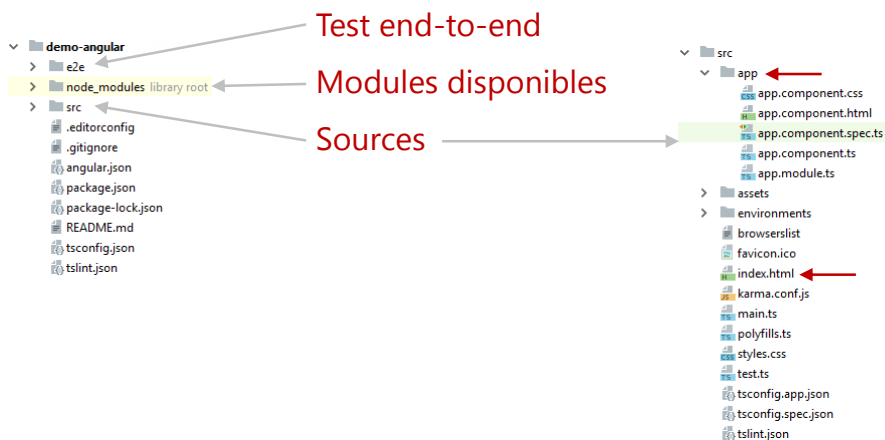
```
cd demo-angular
```

```
ng serve
```

<http://localhost:4200>



## Organisation d'un projet

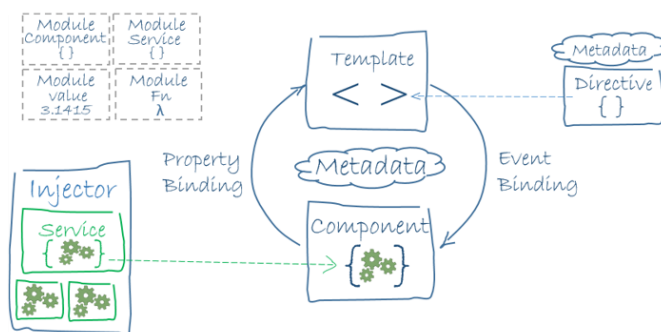


# Démonstration



9

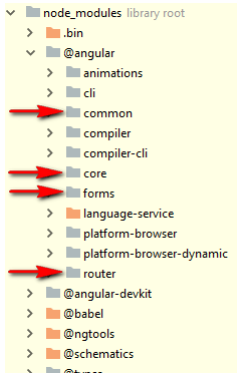
## Architecture générale



## Les bases d'Angular

# Les modules

- Arborescence



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

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

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

11

## Les bases d'Angular

# Les composants et templates

- Composant

```
@Component({
  selector: 'balise-perso',
  templateUrl: './pageHTMLAssociée.html',
  styleUrls: ['./fichierCSSAssocié.css']
})
export class MonComposant {
}
```

- Template

```
<div>
  <h1>
    Un contenu HTML
    relié aux variables
    et méthodes du composant
  </h1>
</div>
```

12

## La liaison de données

- La liaison unidirectionnelle (one-way binding)
  - Composant vers template
    - Interpolation
    - Liaison par propriété (property binding)
  - Template vers composant
    - Liaison par événement (event binding)
- La liaison bidirectionnelle (two-way binding)



## L'interpolation

- Syntaxe : `{{valeur}}`
- Caractéristiques :
  - Instruction TypeScript valide
  - Affichable
  - Peut faire référence aux variables et méthodes du composant

```
@Component({...})  
export class MonComposant {  
  valeur: string;  
}
```

→

```
<p>  
  {{valeur}}  
</p>
```

Liaison unidirectionnelle  
composant vers template



## La liaison par propriété

- Syntaxe : `<...[propriété]="valeur"/>`
- Caractéristiques :
  - La propriété est une propriété HTML ou Angular
  - La valeur est une instruction TypeScript valide
  - Affichable
  - Peut faire référence aux variables et méthodes du composant

```
@Component({...})
export class MonComposant {
  valeur: string;
}
```

→ `<p>`  
`<input [value]="valeur"/>`  
`</p>`

Liaison unidirectionnelle  
composant vers template



## La liaison par événement

- Syntaxe : `<... (événement)="méthode"/>`
- Caractéristiques :
  - La méthode est disponible sur le composant

```
@Component({...})
export class AppComponent {
  public onClick(): void {
    //traitement
  }
}
```

← `<button (click)="onClick()">`  
Cliquer  
`</button>`

Liaison unidirectionnelle  
template vers composant





## La liaison bidirectionnelle

- Syntaxe : `<... [(ngModel)]="variable"/>`

- Import : 

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

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

```
@Component({...})  
export class MonComposant {  
  valeur: string;  
}  
  
<input  
  type="text"  
  [(ngModel)]="valeur"/>
```

Liaison bidirectionnelle

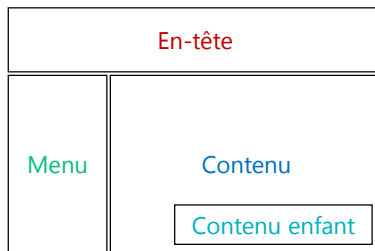


## Démonstration

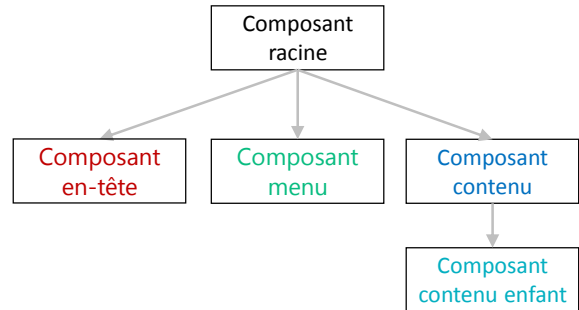


## Les vues

- Représentation graphique

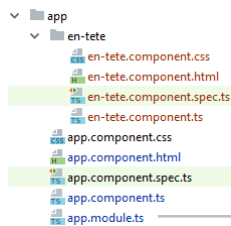


- Représentation par composants



## Créer un nouveau composant

ng generate component en-tete



```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
import { EnTeteComponent } from './en-tete/en-tete.component';

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



## Intégrer un composant dans un autre composant

- Composant enfant

- La classe TypeScript

```
@Component({  
  selector: 'app-en-tete',  
  ...  
})  
export class EnTeteComponent {  
  ...  
}
```

- Le template HTML

```
<p>  
  Bonjour depuis le composant enfant  
</p>
```

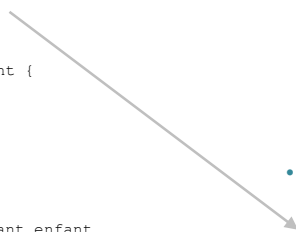
- Composant parent

- La classe TypeScript

```
@Component({  
  selector: 'app-root',  
  ...  
})  
export class AppComponent {  
  ...  
}
```

- Le template HTML

```
<!--Référence au composant enfant :-->  
<app-en-tete></app-en-tete>  
<!--...-->
```



## Passer des paramètres entre composants

- Déclaration d'un paramètre

- Variable membre du composant enfant
- Variable annotée avec `@Input()`

- Passage du paramètre

```
<app-en-tete [parametre]='VALEUR PASSEE EN PAREMETRE'></app-en-tete>  
<app-en-tete [parametre]="instruction TypeScript valide"></app-en-tete>
```



# Démonstration



## Les directives

- Disponibles sur le template
- Adaptent l'affichage au contexte
- Deux groupes de directives
  - Les directives structurelles
  - Les directives par attributs



## Directives structurelles

- Modifient la structure d'un document

- \*ngIf

```
<p *ngIf="instruction TypeScript retournant un booléen">  
  Affichage conditionné de la balise p.  
</p>
```



## Directives structurelles

- Modifient la structure d'un document

- \*ngForOf

```
<p *ngFor="let t of tableau">  
  Répétition de la balise p autant de fois  
  que d'éléments dans le tableau.  
  Élément courant : {{t}}  
</p>
```



## Directives structurelles

- Modifient la structure d'un document

- ngSwitch

```
<ul [ngSwitch]="maVariable">
  <li *ngSwitchCase="'valeur'">un contenu</li>
  <li *ngSwitchDefault>un autre contenu</li>
</ul>
```



## Directives par attributs

- Modifient les caractéristiques des éléments du DOM

- ngModel

- Composant

```
valeur: string;
```

- Template

```
<input type="text" [(ngModel)]="valeur"/>
```



## Directives par attributs

- Modifient les caractéristiques des éléments du DOM

- ngClass

- Fichier CSS

```
.classe1{  
  color: red;  
}  
.classe2{  
  font-size: xx-large;  
}
```

- Composant

```
classesAUtiliser: any = { 'classe1': true, 'classe2': false};
```

- Template

```
<div [ngClass]="classesAUtiliser">  
  Du contenu  
</div>
```



## Directives par attributs

- Modifient les caractéristiques des éléments du DOM

- ngStyle

- Composant

```
stylesAUtiliser: any = {  
  'font-style': true ? 'italic' : 'normal',  
  'font-weight': false ? 'bold' : 'normal'  
};
```

- Template

```
<div [ngStyle]="stylesAUtiliser">  
  Du contenu  
</div>
```



# Démonstration



# TP





## Les pipes

- Formatage des données pour l'affichage

- Pipes date et uppercase

```
<p>Aujourd'hui : {{aujourd'hui | date}}</p>
```

```
<p>Aujourd'hui : {{aujourd'hui | date:'dd/MM/yyyy'}}</p>
```

```
<p>Aujourd'hui : {{aujourd'hui | date:'dd MMMM yyyy' | uppercase}}</p>
```

- Pipes disponibles

 <https://angular.io/api?type=pipe>



## Démonstration



## Les services

- Les composants sont des contrôleurs frontaux
  - Interfaces
  - Actions utilisateurs
- Les services gèrent les traitements métier
  - Récupération d'informations sur le serveur
  - Authentification
  - Validation
  - Log
  - ...
- Objectif
  - Augmenter la modularité



## Créer et référencer un service

ng g service ReglesMetiers

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

@Injectable({
  providedIn: 'root'
})
export class ReglesMetiersService {
  constructor() { }
}
```

Service référencé  
pour l'ensemble  
de l'application

une instance unique



## Créer et référencer un service

```
import { ReglesMetiersService } from '../services/regles-metiers.service';

@NgModule({
  declarations: [
    AppComponent,
    EnteteComponent,
    providers: [ReglesMetiersService]
  ],
  ...
})
export class AppModule { }
```

Service référencé  
pour l'ensemble  
du module

Une instance unique



## Créer et référencer un service

```
import { Component, Input, OnInit } from '@angular/core';
import { ReglesMetiersService } from '../services/regles-metiers.service';

@Component({
  ...
  providers: [ReglesMetiersService]
})
export class MonComponent implements OnInit {
```

Service référencé  
pour le composant  
seulement

Une instance  
pour chaque instance  
du composant



## Injecter et utiliser un service

- Injection dans le constructeur
  - D'un composant
  - D'un autre service

```
constructor(private reglesMetiersService: ReglesMetiersService) { }  
  
uneMethode() {  
    this.reglesMetiersService.uneMethodeDuService();  
}
```

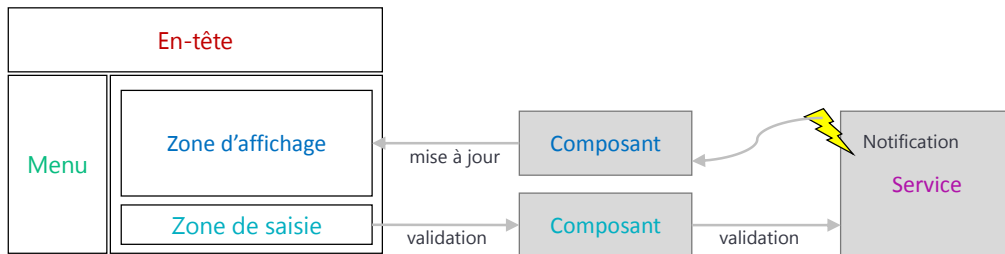


## Démonstration



## Programmation réactive

- Ou comment améliorer la communication entre les éléments ?



Observateur / Observable



## RxJS

- Basé sur RxJS (Reactive Extensions for JavaScript)
  - Programmation asynchrone
  - Propagation des changements
- `EventEmitter` : l'observable
- `Subscription` : l'observateur



## Mise en œuvre

### Observable

```
export class MonObservableService {  
  private datas: string[] = ['info1', 'info2'];  
  public dataEmetteur: EventEmitter<string[]> = new EventEmitter();  
  constructor() { }  
  
  public notifier(): void {  
    this.dataEmetteur.emit(this.datas.slice());  
  }  
  
  ajouter(valeur: string) {  
    this.datas.push(valeur);  
    this.notifier();  
  }  
}
```



### Observateur

```
export class MonObservateurComponent implements OnInit, OnDestroy {  
  tableau: string[];  
  constructor(private reglesMetierService: ReglesMetierService) {}  
  
  ngOnInit(): void {  
    this.reglesMetierService.dataEmetteur.subscribe((infos) => this.tableau = infos);  
  }  
  ngOnDestroy(): void {  
    this.reglesMetierService.dataEmetteur.unsubscribe();  
  }  
}
```



## Démonstration



# Formulaire côté template

## 1. Créer un formulaire classique

```
<form>
  <div class="form-group">
    <label class="center-block">Nom:</label>
    <input class="form-control" type="text"/>
  </div>
  <div class="alert alert-danger">
    Le nom est obligatoire
  </div>
  <button type="submit"
    class="btn-success">
    Enregistrer
  </button>
</form>
```



# Formulaire côté template

## 2. Nommer le formulaire

```
<form [formGroup]="angularForm">
  <div class="form-group">
    <label class="center-block">Nom:</label>
    <input class="form-control" type="text"/>
  </div>
  <div class="alert alert-danger">
    Le nom est obligatoire
  </div>
  <button type="submit"
    class="btn-success">
    Enregistrer
  </button>
</form>
```



## Formulaire côté template

### 3. Nommer les zones de saisie

```
<form [formGroup]="angularForm">
  <div class="form-group">
    <label class="center-block">Nom:</label>
    <input class="form-control"
      FormControlName="nom"
      type="text"/>
  </div>
  <div class="alert alert-danger">
    Le nom est obligatoire
  </div>
  <button type="submit"
    class="btn-success">
    Enregistrer
  </button>
</form>
```



## Formulaire côté template

### 4. Adapter l'affichage

```
<form [formGroup]="angularForm">
  <div class="form-group">
    <label class="center-block">Nom:</label>
    <input [ngClass]="{'form-control': true,
      'is-invalid': nomInvalide()}"
      FormControlName="nom"
      type="text"/>
  </div>
  <div *ngIf="nomInvalide()" class="alert alert-danger">
    Le nom est obligatoire
  </div>
  <button type="submit"
    class="btn-success" [disabled]="problemeValidation()">
    Enregistrer
  </button>
</form>
```





## Formulaire côté composant

### 5. Écrire le comportement

```
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
@Component({...})
export class FormulaireComponent {
  angularForm: FormGroup;
  constructor(private formBuilder: FormBuilder) {
    this.creerFormulaire();
  }
  private creerFormulaire() {
    this.angularForm = this.formBuilder.group({
      nom: ['', Validators.required]
    });
  }
  public nomInvalide(): boolean {
    return this.angularForm.controls.nom.invalid &&
      (this.angularForm.controls.nom.dirty ||
        this.angularForm.controls.nom.touched);
  }
  public problemeValidation(): boolean {
    return this.angularForm.pristine || this.angularForm.invalid;
  }
}
```



## Démonstration

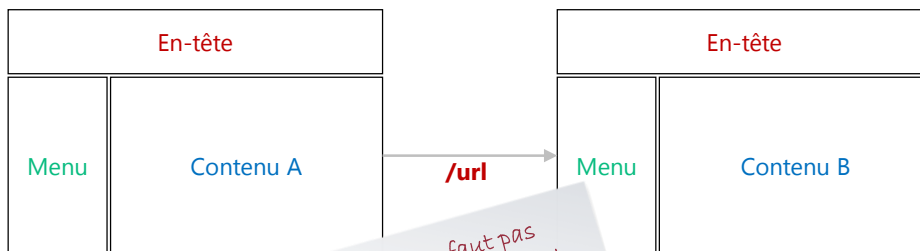


# TP



## Navigation / Routage

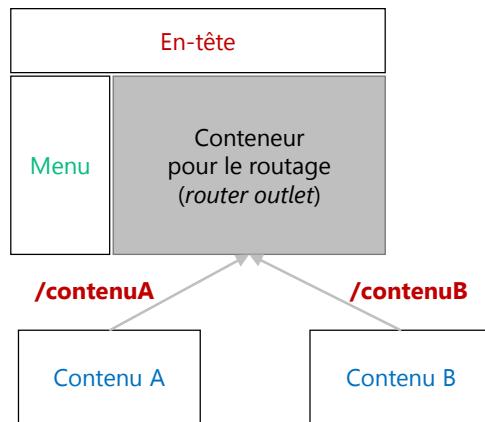
- Single Page Application
- Plusieurs vues



*Il ne faut pas  
un rechargement  
complet de la page*



## Navigation / Routage



## Navigation / Routage

### 1. Définir la zone modifiable dans le template

```
<app-en-tete></app-en-tete>
<app-menu></app-menu>
<app-contenu-a></app-contenu-a>
<router-outlet></router-outlet>
```



# Navigation / Routage

## 2. Définir l'URL racine



```
<!doctype html>
<html lang="en">
<head>
  <base href="/">
  ...
</head>
<body>
  <app-root></app-root>
</body>
</html>
```



# Navigation / Routage

## 3. Définir les routes et les composants associés



```
import {RouterModule, Routes} from '@angular/router';
...
const appRoutes: Routes = [
  {path: 'contenuA', component: ContenuAComponent},
  {path: 'contenuB', component: ContenuBComponent},
  {path: '', redirectTo: '/contenuA', pathMatch: 'full'},
  {path: '**', component: InconnuComponent}
];

@NgModule({
  ...
  imports: [
    RouterModule.forRoot(
      appRoutes
    )
  ]
})
export class AppModule { }
```



## Navigation / Routage

### 4. Naviguer au travers de liens

```
<a routerLink="/contenuA">Contenu A</a>  
<a routerLink="/contenuB">Contenu B</a>
```



## Navigation / Routage

### 5. Naviguer programmatiquement

```
export class ContenuBComponent {  
  constructor(private router: Router) { }  
  
  onClickRetour() {  
    this.router.navigate(['/contenuA']);  
  }  
}
```



## Navigation / Routage

### 6. Gérer les URL paramétrables

#### a) Définir la route

```
const appRoutes: Routes = [...,{path: 'detail/:id', component: DetailComponent}];
```

#### b) Définir les liens

```
<a routerLink="/detail/1">Détail 1</a>  
<a routerLink="/detail/{{idDetail}}">Détail {{idDetail}}</a>
```

```
this.router.navigate(['detail', this.idDetail]);
```

#### c) Lire le paramètre

```
export class DetailComponent implements OnInit {  
  identifiantRecu: number;  
  
  constructor(private route: ActivatedRoute) { }  
  
  ngOnInit() {this.identifiantRecu = +this.route.snapshot.paramMap.get('id');}  
}
```



## Démonstration



## Les services web REST

- Interroger un serveur HTTP
- Injection d'un objet HttpClient

```
import {HttpClient} from '@angular/common/http';  
  
export class CrudHttpService {  
  constructor(private http: HttpClient) { }  
}
```

- Traitement asynchrone

```
Observable<any>
```



## Les services web REST

- Réalisation d'une requête de type GET

```
public obtenir(): Observable<any>  
{  
  return this.http.get("http://localhost:8080/infos");  
}
```

- Utilisation

```
this.crudHttpService.obtenir()  
  .subscribe(  
    (value)=>this.datas=value,  
    (error)=>console.log("Erreur: " +error),  
    ()=>console.log("Fin")) ;
```



## Les services web REST

- Réalisation d'une requête de type POST

```
public ajouter(data: any): Observable<any>
{
    return this.http.post("http://localhost:8080/infos", data);
}
```

- Utilisation

```
this.crudHttpService.ajouter("des infos").subscribe();
```



## Les services web REST

- Traitement des données avant restitution
- Subject : observateur et observable

```
//Création d'un subject
var sujet : Subject<any> = new Subject<any>();

//Observation d'un subject (c'est un observable)
sujet.subscribe((value) => console.log("Le sujet me dit : " + value),
    (error) => console.log("Erreur : " + error),
    ()=>console.log("Fin de l'écoute"));

//On lui envoie des données (c'est un observateur)
sujet.next("bonjour tout le monde");
sujet.complete();
```





## Les services web REST

- Évolution d'une requête de type GET

```
public obtenir(): Observable<string[]>
{
    var sujet : Subject<string[]> = new Subject<string[]>();
    this.http
        .get<string[]>("http://localhost:8080/infos")
        .subscribe(
            (tab) =>{
                tab.forEach((value, index) => tab[index]=tab[index].toUpperCase());
                sujet.next(value)
            },
            (error)=>sujet.error(error),
            () => sujet.complete()
        );
    return sujet;
}
```



## Démonstration



TP



## Conclusion

- Vous connaissez les fondamentaux d'Angular :
  - Les composants
  - Les templates
  - Les pipes
  - Les directives
  - Les services
  - La gestion des formulaires
  - La navigation et le routage
  - L'usage des services web REST

