



Angular

Par Robin Delbaere

Qu'est ce qu'Angular ?

- Framework Javascript
- Développé par Google
- Single Page Application
- Basé sur TypeScript

D'autres Frameworks ?

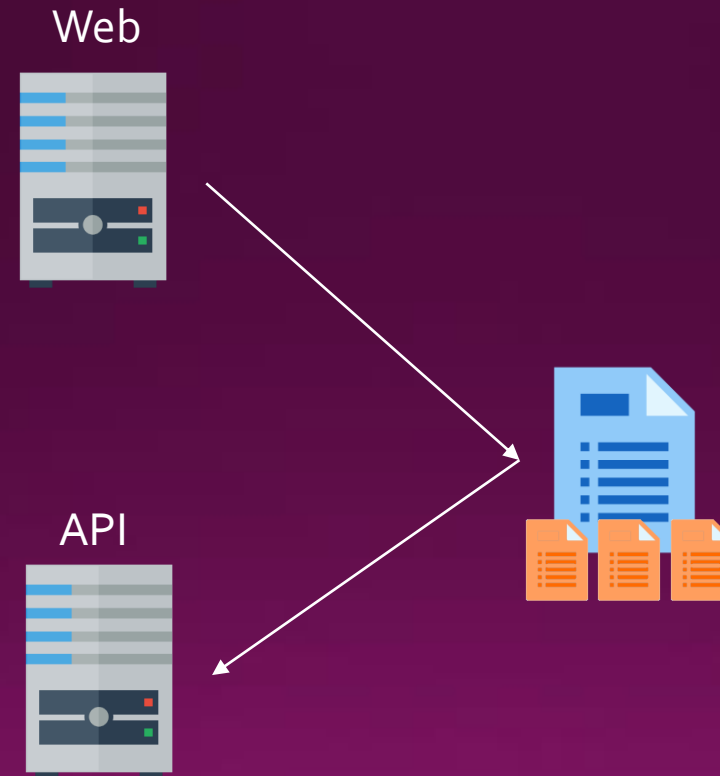
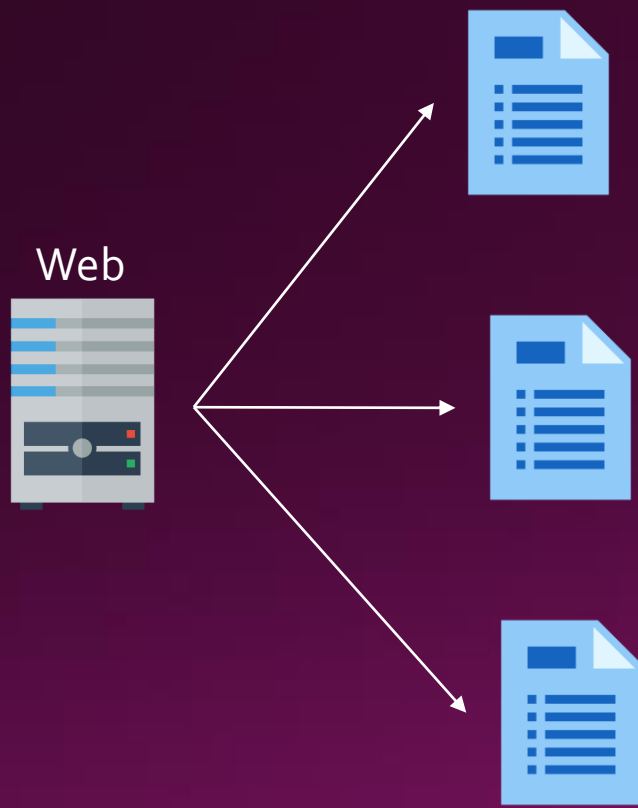


React



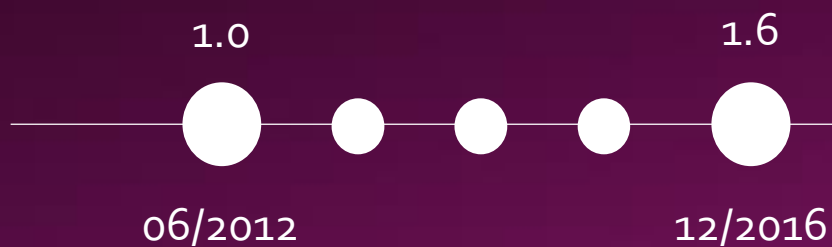
ember

Application Single Page

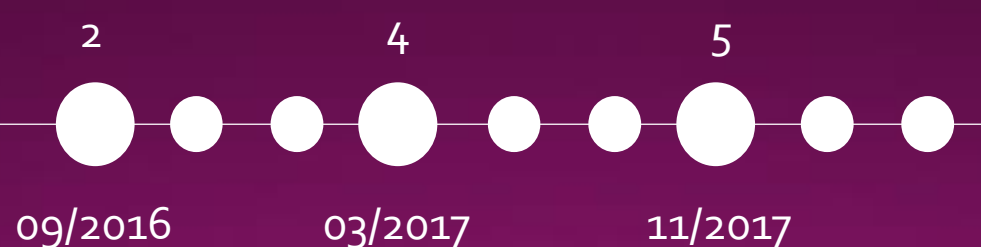


Historique

AngularJS



Angular



L'épreuve de force

- Avantages
 - Productivité
 - Documentation
 - Communauté
 - Exhaustif
 - Performance
- Inconvénients
 - Complexité
 - Intégration

Le point sur ES6

- ECMAScript
 - Norme pour certains langages
 - Notamment Javascript
 - ECMAScript 5
 - Norme majoritairement supportée
 - ECMAScript 6
 - Egaleme^{nt} nommé ECMAScript 2015
 - En cours d'intégration
 - Utilisation d'un transpileur pour la compatibilité

Quelques nouveautés d'ES6

Le mot clé « let »

```
let number = 5;
```

Le mot clé « const »

```
const PI = 3.141592;
```

Valeur par défaut

```
function add( a, b = 1 ){  
  return a + b;  
}  
  
add( 5 );
```

Notion de classe

```
function Car( color ){  
  this.color = color;  
  this.engine = false;  
}  
  
Car.prototype.start = function(){  
  this.engine = true;  
}  
  
Car.prototype.stop = function(){  
  this.engine = false;  
}
```



```
class Car{  
  constructor( color ){  
    this.color = color;  
    this.engine = false;  
  }  
  
  start(){  
    this.engine = true;  
  }  
  
  stop(){  
    this.engine = false;  
  }  
}
```

Notion de d'héritage

```
class Crossover extends Car{  
  constructor( color, wheels ){  
    super( color );  
    this.wheels = wheels;  
  }  
}
```


ES6 - Les promesses

Callback

```
function loadUser( callback ){
  let user = true;
  if( user ){
    return callback( true );
  }else{
    return callback( false );
  }
}

loadUser( function( success ){
  console.log( success );
});
```

Pour aller plus loin

```
loadUser()
  .then( user => loadOrders( user ) )
  .then( orders => console.log( orders ) );
```

Promesse

```
function loadUser(){
  return new Promise( function( resolve, reject ){
    let user = true;

    if( user ){
      resolve( user );
    }else{
      reject( 'Une erreur est survenue !' );
    }
  });
}

loadUser().then( function( user ){
  console.log( user );
}, function( error ){
  console.log( error );
});
```

Et le TypeScript ?

- Métalangage Javascript
 - Surcouche au langage
 - Simplification & nouvelles fonctionnalités
 - Compilé en Javascript
 - Extension des fichiers en `.ts`

Typage des variables

```
let speed: number = 350;
let name: string = 'Tesla Model S';

let tesla: Car = new Car( name, speed );
let mustang: any = new Car( 'Mustang Shelby GT 500', 320 );

let cars: Array<Car> = [ tesla, mustang ];
```

Aller plus loin avec le typage

```
class Car{
    private name: string;
    private speed: number;

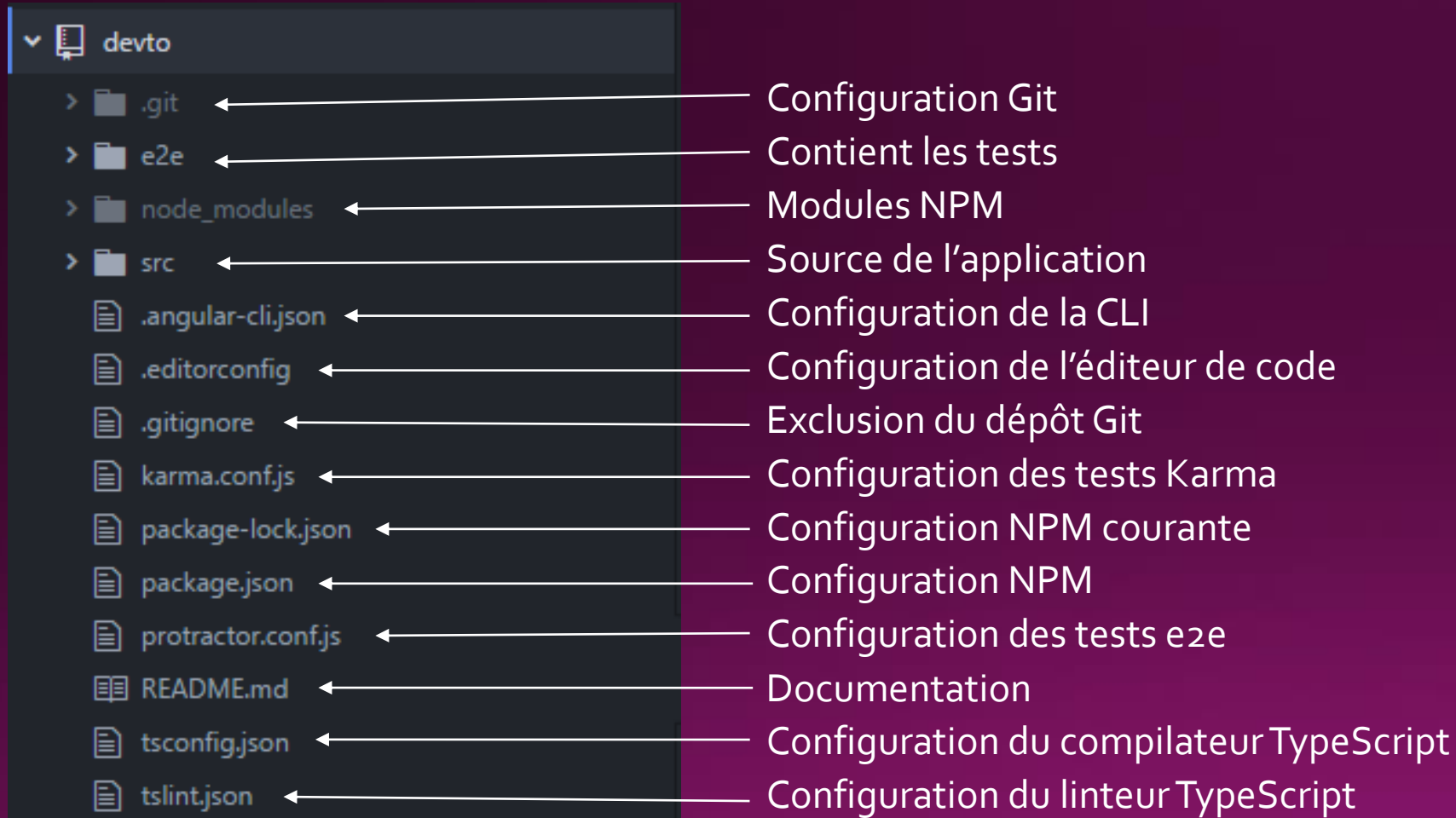
    constructor( name: string, speed: number ){
        this.name = name;
        this.speed = speed;
    }

    getSpeed(): number{
        return this.speed;
    }
}
```

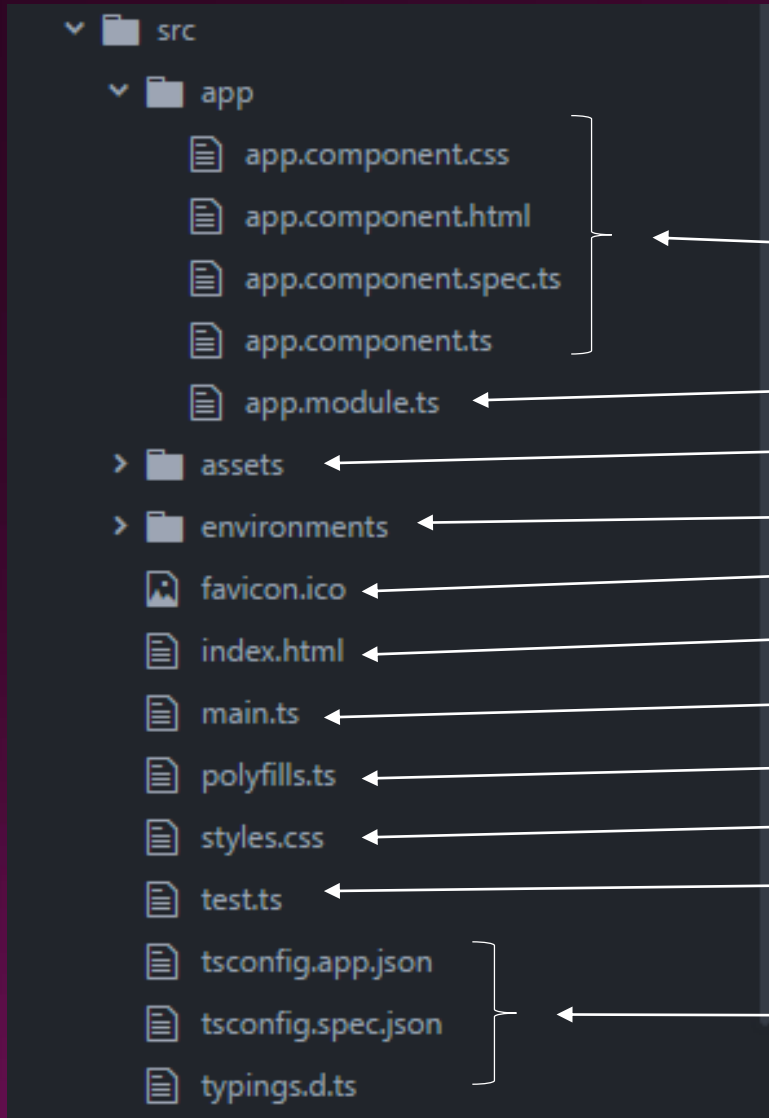
Installation

- Installer NodeJS
 - <https://nodejs.org>
 - Installer Angular CLI
 - `λ npm install -g @angular/cli`
-
- Créer un nouveau projet
 - `λ ng new devto`
 - Lancer l'application
 - `λ ng serve --open`

Structure



Structure « src »



Définition du composant principal

Définition des composants

Assets du projet

Configuration des environnements

Icône pour navigateur

Point d'entrée

Cœur de l'application

Normalisation

CSS générique

Cœur des tests

Configuration du compilateur TypeScript

Pratique

- Installer Angular CLI
- Créer un nouveau projet
- Lancer votre application

Composants

- Architecture orienté composant
- Basé sur les Web Components
- Orchestré par `AppComponent`

app.component.ts

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

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

app.component.html

```
<h1>{{ title }}</h1>

<p>
  Lorem ipsum dolor sit amet, consectetur adipisicing elit
</p>
```

app.component.css

```
h1{
  color: blue;
  font-size: 22px;
}
```

Créer un composant

- Générer un composant
 - `ng generate component mementos`
- Utiliser un composant
 - `<app-mementos></app-mementos>`

app.component.html

```
<h1>{{ title }}</h1>
<p>
  Lorem ipsum dolor sit amet, consectetur adipisicing elit
</p>

<app-mementos></app-mementos>
```

mementos.component.ts

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

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

  constructor(){}

  ngOnInit(){}
}
```


Cycle de vie d'un composant

- Les étapes

Nom	Description
ngOnChanges	Avant « ngOnInit » et lors d'une modification des propriétés
ngOnInit	Lors de l'initialisation
ngDoCheck	Permet de modifier le comportement de la détection des changements
ngAfterViewInit	Après l'initialisation de la vue
ngOnDestroy	Lors de la destruction

- Utilisation

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

@Component({
  selector: 'app-mementos',
  templateUrl: './mementos.component.html',
  styleUrls: ['./mementos.component.css']
})
export class MementosComponent implements OnInit, OnDestroy {
  private mementos: Array<any>;

  constructor(){}

  ngOnInit(){
    this.mementos = [ 'HTML', 'CSS', 'Javascript', 'PHP' ];
  }

  ngOnDestroy(){
    this.mementos = [];
  }
}
```

Pratique

- Créer un composant pour lister les films disponible
- Créer un tableau de film à l'initialisation du composant
- Intégrer ce composant dans la composant racine

Les templates – L'interpolation

- Afficher la propriété d'une classe dans un template

mementos.component.ts

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

@Component({
  selector: 'app-mementos',
  templateUrl: './mementos.component.html',
  styleUrls: ['./mementos.component.css']
})
export class MementosComponent implements OnInit {
  private title: string = 'Les mementos';
  private description: string = 'Retrouvez nos mementos pour divers langages';
  private picture: string = 'https://encrypted-tbn0.gstatic.com/...';
  private mementos: Array<any>;

  ngOnInit(){
    this.mementos = [ 'HTML', 'CSS', 'Javascript', 'PHP' ];
  }
}
```

mementos.component.html

```
<h2>{{ title }}</h2>

<p>
  {{ description }}
</p>

<img [src]="picture" />
```

Les templates – Les évènements

- Lier un événement

```
<img [src]="picture" (click)="clickPicture()" />
```

- Définir la fonction

```
clickPicture(){  
    console.log( 'Picture clicked!' );  
}
```

Les templates – Les variables

- Définir une variable

```
<input type="text" #field />
```

- Utiliser la variable

```
<input type="text" #field (keyup)="0" />  
<p>{{ field.value }}</p>
```

- Transmettre la valeur

```
<input type="text" #field (keyup)="showValue( field.value )" />
```

```
showValue( value: string ){  
    console.log( value );  
}
```

Les templates – La directive « ngFor »

- Répéter une portion du template

```
<div class="mementos">  
  <div class="memento" *ngFor="let memento of mementos">  
    {{ memento }}  
  </div>  
</div>
```

Les templates – La directive « ngIf »

- Conditionner l’affichage

```
<div *ngIf="isConnected == true">  
  Vous êtes connecté.e  
</div>
```

```
private isConnected: boolean = true;
```

Pratique

- Afficher la liste des films dans le template
- Afficher le nombre de film disponible
 - Si aucun film n'est disponible afficher un message spécifique
- Afficher une alerte contenant le nom du film au clique sur celui-ci

Les directives

- Classe avec l'annotation `@Directive`
- Interagir avec un élément HTML
- Type de directive
 - Composant
 - Directive d'attribut
 - Directive structurelle

Créer une directive

- Générer une directive

- `λ ng generate directive mementoActive`

- Utiliser la directive

```
<div class="memento" *ngFor="let memento of mementos" mementoActive>
  {{ memento }}
</div>
```

- Modifier le style de l'élément

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

constructor( private el: ElementRef ){
  this.el.nativeElement.style.backgroundColor = 'red';
}
```

memento-active.directive.ts

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

@Directive({
  selector: '[mementoActive]'
})
export class MementoActiveDirective{

  constructor(){}

}
```

Les directives – Événements

- Utilisation de l'annotation `@HostListener`

```
@HostListener('mouseenter') onMouseEnter(){
    this.setBackground( 'red' );
}

@HostListener('mouseleave') onMouseLeave(){
    this.setBackground( 'white' );
}

setBackground( color ){
    this.el.nativeElement.style.backgroundColor = color;
}
```

Pratique

- Créer une directive pour réaliser un effet au survol d'un film

Les pipes

- Transformer une donnée
- Utiliser un pipe

```
private current: Date = new Date();
```

```
<p>{{ current|date }}</p>
```

- Passer un paramètre au pipe

```
<p>{{ current|date:"dd/MM/yy" }}</p>
```

- Chainer les pipes

```
<p>{{ current | date:"dd MMMM yyyy" | uppercase }}</p>
```

Nom	Description
date	Formater la date
uppercase	Passer en majuscule
lowercase	Passer en minuscule
currency	Affichage des devises
json	Convertir en chaine

Créer un pipe

- Générer un pipe
 - `ng generate pipe mementoDifficulty`
- Données et affichage

```
ngOnInit(){  
  this.mementos = [  
    { name: 'HTML', difficulty: 1 },  
    { name: 'CSS', difficulty: 1 },  
    { name: 'Javascript', difficulty: 2 },  
    { name: 'PHP', difficulty: 2 },  
    { name: 'Symfony', difficulty: 3 },  
  ];  
}
```

```
{{ memento.name }}  
<span>  
  {{ memento.difficulty | mementoDifficulty }}  
</span>
```

```
<span [innerHTML]="memento.difficulty | mementoDifficulty">  
</span>
```

memento-difficulty.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';  
  
@Pipe({  
  name: 'mementoDifficulty'  
})  
export class MementoDifficultyPipe implements PipeTransform {  
  
  transform(value: any, args?: any): any {  
    return null;  
  }  
}
```

- Modifier la transformation

```
transform(value: any): string {  
  if( value == 1 ){  
    return 'Facile';  
  }else if( value == 2 ){  
    return 'Moyen';  
  }else {  
    return 'Difficile';  
  }  
}
```

Pratique

- Ajouter la date de sortie au format internationale ainsi que la note dans les données du film
- Afficher cette date au format français
- Créer un pipe pour convertir la note en étoile

Les routes – Configuration

- Simulation d'une navigation « classique »
- Configurer les routes dans `app.module.ts`

```
import { RouterModule, Routes } from '@angular/router';
```

```
const routes: Routes = [  
  { path: 'memento', component: MementosComponent },  
  { path: 'memento/:id', component: MementoComponent }  
];
```

```
@NgModule({  
  imports: [  
    BrowserModule, RouterModule.forRoot( routes )  
  ],  
  // ...  
})
```

app.component.html

```
<h1>{{ title }}</h1>
```

```
<router-outlet></router-outlet>
```


Les routes – Redirections

- Dans un template

```
<div routerLink="/memento/{{ memento.id }}">
  {{ memento.name }}
  <span>
    {{ memento.difficulty | mementoDifficulty }}
  </span>
</div>
```

- Dans un component

```
import { Router } from '@angular/router';

constructor( private router: Router ){

selectMemento( id: number ){
  console.log("la");
  this.router.navigate( [ '/memento', id ] );
}
```

```
<div (click)="selectMemento( memento.id )">
  {{ memento.name }}
  <span>
    {{ memento.difficulty | mementoDifficulty }}
  </span>
</div>
```

Les routes – Paramètres

- Récupérer la route

```
import { ActivatedRoute } from '@angular/router';  
constructor( private route: ActivatedRoute ){}  
  
```

- Récupérer le paramètre

```
let id = this.route.snapshot.params['id'];  
  
```

Les routes – Une page d'erreur

- Générer un composant pour gérer les erreurs

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

@Component({
  selector: 'app-not-found',
  template: '<h2>404 - Not Found</h2>',
})
export class NotFoundComponent{}
```

- Intercepter toutes les routes avec l'opérateur 

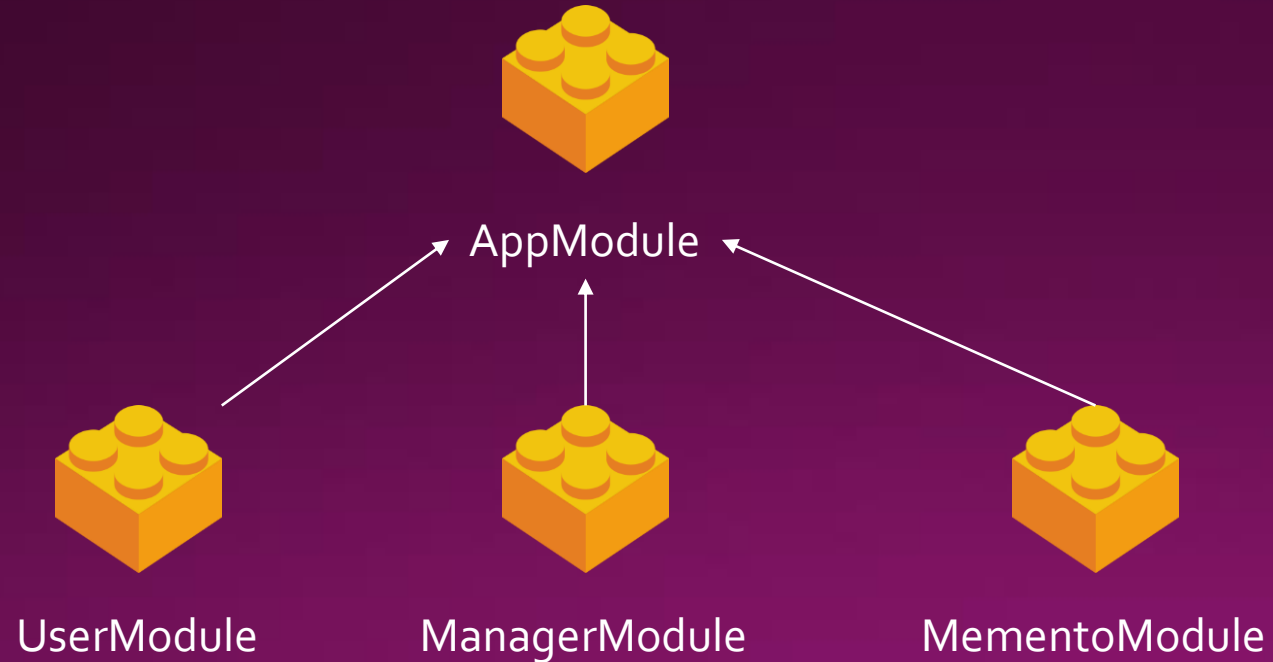
```
const routes: Routes = [
  { path: 'memento', component: MementosComponent },
  { path: 'memento/:id', component: MementoComponent },
  { path: ' ** ', component: NotFoundComponent },
];
```

Pratique

- Créer un composant pour visualiser un film
- Mettre en place les routes pour les pages suivantes
 - Page de présentation du service
 - Page listant les films
 - Page de visualisation d'un film
- Mettre en place un design de base

Les modules

- Architecture modulaire
- Un module racine
- Un module par fonctionnalité



Créer un module

- Générer un module

- `ng generate module user`

user.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

@NgModule({
  imports: [
    CommonModule
  ],
  declarations: [],
  providers: [],
  exports: []
})
export class UserModule { }
```

app.module.ts

```
import { UserModule } from '../user/user.module';

@NgModule({
  imports: [
    BrowserModule, UserModule, RouterModule.forRoot( routes )
  ],
  declarations: [
    AppComponent,
    MementosComponent,
    MementoActiveDirective,
    MementoDifficultyPipe,
    MementoComponent,
    NotFoundComponent
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Pratique

- Créer un module pour les films
- Déplacer tous les éléments relatifs aux films à l'intérieur
- Créer un module pour la gestion des utilisateurs
 - Un composant pour la connexion
 - Un composant pour l'inscription
 - Une page regroupant ces 2 composants

Les services

- Factoriser le code
- Centraliser la gestion des données
- Isoler la partie métier

Créer un service

- Générer un service

- `λ ng generate service memento`

- Enregistrer le service

*.module.ts

```
import { MementoService } from './memento.service';
```

```
@NgModule({  
  // ...  
  providers: [MementoService],  
  // ...  
})
```

memento.service.ts

```
import { Injectable } from '@angular/core';  
  
@Injectable()  
export class MementoService {  
  
  constructor() { }  
  
}
```



```
export class MementoService {  
  private mementos: Array<any>;  
  
  constructor(){  
    this.mementos = [  
      { id: 1, name: 'HTML', difficulty: 1 },  
      { id: 2, name: 'CSS', difficulty: 1 },  
      { id: 3, name: 'Javascript', difficulty: 2 },  
      { id: 4, name: 'PHP', difficulty: 2 },  
      { id: 5, name: 'Symfony', difficulty: 3 },  
    ];  
  }  
  
  getAll(){  
    return this.mementos;  
  }  
}
```

Utiliser le service

- Importer

```
import { MementoService } from '../memento.service';
```

- Injecter

```
constructor(  
    private router: Router,  
    private mementoService: MementoService  
){}  

```

- Consommer

```
ngOnInit(){  
    this.mementos = this.mementoService.getAll();  
}
```

Pratique

- Créer un service pour centraliser la gestion des films

Les formulaire

- Utilisation du module `@angular/forms`
- Deux approches différentes
 - Piloté par le template avec `FormModules`
 - Piloté par le composant avec `ReactiveFormsModule`

Créer un formulaire

- Créer un modèle
- Configurer le module
- Générer un composant

- `λ ng generate component register --module user`

user.module.ts

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

const routes: Routes = [
  { path: 'register', component: RegisterComponent },
];
```

```
imports: [
  CommonModule, RouterModule.forChild( routes ), FormsModule
],
```

user.ts

```
export class User{
  id: number;
  username: string;
  password: string;
  role: string;
  created: Date;
}
```

register.component.ts

```
import { Component, OnInit } from '@angular/core';
import { User } from '../user';

@Component({
  selector: 'user-register',
  templateUrl: './register.component.html',
})
export class RegisterComponent implements OnInit{
  private user: User = new User();
  private roles: Array<string>;

  ngOnInit(){
    this.roles = ['Etudiant', 'Développeur', 'Manager'];
  }
}
```

Le template du formulaire

- Configuration du formulaire
 - Créer une variable `#registerForm="ngForm"`
 - Intercepter l'envoi `(ngSubmit)="register()"`
- Liaison avec le modèle
 - `[(ngModel)]="user.username"`

```
<form #registerForm="ngForm" (ngSubmit)="register()">
  <input type="text" [(ngModel)]="user.username" name="username" placeholder="Nom d'utilisateur">

  <input type="password" [(ngModel)]="user.password" name="password" placeholder="Mot de passe">

  <select [(ngModel)]="user.role" name="role">
    <option *ngFor="let role of roles" [value]="role">{{ role }}</option>
  </select>

  <button type="submit">S'inscrire</button>
</form>
```

```
register(){
  console.log( 'Inscription de : ' + this.user.username );
}
```

Validation du formulaire

- Utilisation des attributs HTML5

```
<input type="text" [(ngModel)]="user.username"
      name="username" placeholder="Nom d'utilisateur" required>
```

```
<input type="password" [(ngModel)]="user.password" name="password"
      placeholder="Mot de passe" required pattern=".{6,}">
```

- Désactiver le bouton d'envoi en cas d'erreur

```
<button type="submit" [disabled]="!registerForm.form.valid">S'inscrire</button>
```

- Afficher un message d'erreur

```
<input type="text" [(ngModel)]="user.username" #username="ngModel"
      name="username" placeholder="Nom d'utilisateur" required>
```

```
<div *ngIf="username.invalid && username.dirty">
  Vous devez saisir un nom d'utilisateur
</div>
```

Pratique

- Créer un modèle pour les films
- Créer un composant d'ajout de film
- Créer un modèle d'utilisateur
- Créer le formulaire d'inscription

Communiquer avec une API

- Utilisation du module '@angular/common/http'
- Importer le module

*.module.ts

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

```
imports: [  
  // ...  
  HttpClientModule,  
],
```

Utilisation du module Http

- Injecter le module

```
import { HttpClient } from '@angular/common/http';  
  
constructor( public http: HttpClient ){}  

```

- Effectuer un appel GET

```
this.http.get( 'https://api.exemple.come/memento' )  
  .toPromise()  
  .then( data => console.log( data ) );
```

- Effectuer un appel POST


```
this.http.post( 'https://api.exemple.come/memento', memento )  
  .toPromise()  
  .then( data => console.log( data ) );
```

Consommer une API

- Dans le service

```
getAll(){  
    return this.mementos;  
}
```


```
getAll(){  
    return this.http.get(  
        'https://api.exemple.come/memento'  
    ).toPromise();  
}
```



- Dans le composant

```
ngOnInit(){  
    this.mementos = this.mementoService.getAll();  
}
```

```
ngOnInit(){  
    this.mementoService.getAll()  
        .then( mementos => { this.mementos = mementos; } );  
}
```



Gérer les erreurs HTTP

- Utilisation de la méthode `.catch()`

```
this.http.get(  
    'https://api.exemple.come/memento'  
)  
.toPromise()  
    .then( () => console.log( 'Succesed' ) )  
    .catch( () => console.log( 'Failed' ) );
```

Mettre en place une API de test

- Installer json-server

```
λ npm install -g json-server
```

- Créer un fichier data.json

```
{  
  "memento" : [  
    { "id": 1, "name": "HTML", "difficulty": 1 },  
    { "id": 2, "name": "CSS", "difficulty": 1 },  
    { "id": 3, "name": "Javascript", "difficulty": 2 },  
    { "id": 4, "name": "PHP", "difficulty": 2 },  
    { "id": 5, "name": "Symfony", "difficulty": 3 }  
  ]  
}
```

- Lancer le serveur

```
λ json-server --watch data.json
```

Pratique

- Mettre en place un API de test pour les films
- Modifier le service pour charger les films depuis l'API

Authentication

- Utilisation des *Guards*
- Gérer des scénarios de navigation
- Renvoi un booléen pour autoriser ou non l'accès
- Fonctionnement de manière synchrone et asynchrone
- Différents types des *Guards*

Mettre en place un *Guard*

- Générer un *guard*
 - `λ ng generate guard auth`
- Injecter dans le module racine

```
import { AuthGuard } from './auth.guard';  
providers: [AuthGuard],
```

- Configurer pour une route

```
const routes: Routes = [  
  // ...  
  { path: 'memento/:id', component: MementoComponent, canActivate: [ AuthGuard ] },  
  // ...  
];
```

auth.guard.ts

```
import { Injectable } from '@angular/core';  
import { CanActivate } from '@angular/router';  
  
@Injectable()  
export class AuthGuard implements CanActivate{  
  canActivate(){  
    console.log( 'Firewall' );  
    return true;  
  }  
}
```


Exemple de *Guard*

```
import { Injectable } from '@angular/core';
import { CanActivate, Router } from '@angular/router';

@Injectable()
export class AuthGuard implements CanActivate{
  private isLoggedIn: boolean = false;

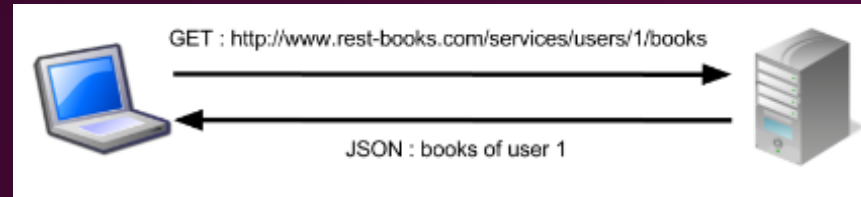
  constructor( private router: Router ){}

  canActivate():boolean{
    if( this.isLoggedIn ){
      return true;
    }

    this.router.navigate( [ 'register' ] );
    return false;
  }
}
```

Authentification sur une API

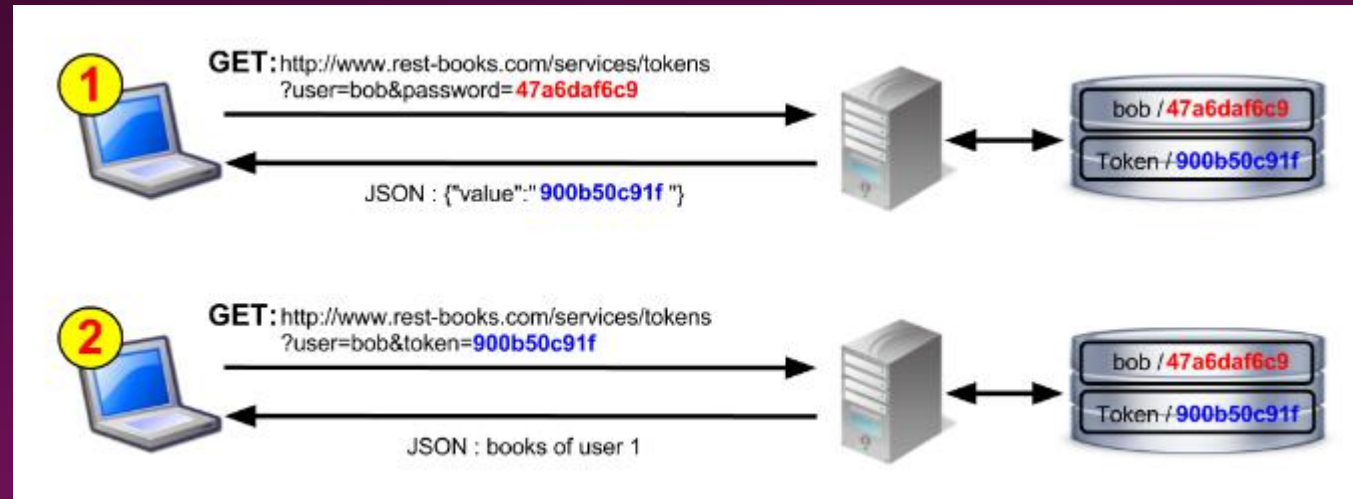
- Ouvert



- Par Identifiant



- Par token



Les points clés

- Utilisation d'un « salt »
- Protocole « https »
- Expiration des tokens
- Signature de la requête (HMAC)

Pratique

- Mettre en place un Guard pour la page de détail d'un film
- Bonus : Authentification complète
 - Créer un service pour gérer l'authentification
 - Mettre en place le formulaire de connexion
 - Mettre en place un bouton de déconnexion

Déployer votre application

- Compiler pour un environnement de test
 - `λ ng build`
- Compiler pour un environnement de production
 - `λ ng build --env=prod`
- L'application compilé se trouve vers le dossier « /dist »
 - Pointer votre serveur web sur le dossier « /dist »
 - Si votre application n'est pas à la racine
 - `λ ng build --env=prod --base-href=/apps/devto`