



Formation Angular

v2019-1

Planning



- JavaScript ES6+
- TypeScript
- Node et NPM
- Angular CLI
- ReactiveX
- Angular

JavaScript ES6+

JavaScript ES6+



Introduction

- EcmaScript, standard sur lequel se base le JavaScript
- Des grandes avancées / nouveautés par rapport à la précédente version
- Supporté par les navigateurs modernes (adieu IE, [voir plus](#))
- Pour de la retro-compatibilité, utilisation de **transpileur** ([Babel](#))



JavaScript ES6+



let

<https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/let>

Permet de déclarer une variable, ayant une porté limitée aux accolades parentes :

```
function getFullName(user) {  
  if (user.isWizard) {  
    let name = '🧙‍♂️ ' + user.name;  
    return name;  
  }  
  // name is not accessible here  
  return user.name;  
}
```

JavaScript ES6+



const

<https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/const>

Permet de déclarer une variable qui ne pourra pas changer de valeur / référence. On peut cependant changer le contenu d'un objet ou d'un tableau :

```
const user = {};
user.name = 'Toto'; // works
user = {'name': 'John'}; // error
```



JavaScript ES6+



Déclaration rapide d'objet

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Op%C3%A9rateurs/Initialiseur_objet#D%C3%A9finir_des_propri%C3%A9t%C3%A9s

Au lieu de répéter une clé / valeur portant le même nom, on peut écrire uniquement le nom de la valeur :

```
function getColor() {  
  const name = 'blue';  
  const color = 0x0000FF;  
  return { name, color }; // return { name: name, color: color };  
}
```



JavaScript ES6+



Affectations déstructurées

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Opérateurs/Affecter_par_décomposition

Permet de créer des variables à partir de propriétés d'objet ou de valeur tableau :

```
// object
const httpOptions = { timeout: 2000, isCache: true };
const { timeout, isCache } = httpOptions;
// array
const timeouts = [1000, 2000, 3000];
const [shortTimeout, mediumTimeout] = timeouts;
```



JavaScript ES6+



Paramètres fonction valeurs par défaut

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Fonctions/Valeurs_par_défaut_des_arguments

Permet d'initialiser des paramètres lors de l'appel de la fonction si aucune valeur n'est passée :

```
function multiplier(a, b = 1) {
  return a * b;
}
multiplier(5, 2); // 10
multiplier(5, 1); // 5
multiplier(5); // 5
```



JavaScript ES6+



Rest parameters

[https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Fonctions/paramètres_du reste](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Fonctions/param%C3%A8tres_du reste)

Permet de regrouper dans un tableau un nombre non défini de valeurs :

```
function maFonction(a, b, ...moreArguments) {
  for(const arg of moreArguments) {
    console.log('arg', arg);
  }
}

const runners = ['John', 'Maria', 'Pedro'];
const [winner, ...others] = runners; // others === ['Maria', 'Pedro'];
```



JavaScript ES6+



Spread operator

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Op%C3%A9rateurs/Syntaxe_d%C3%A9composition

Permet d'étaler/éclater les valeurs d'un objet ou tableau dans le parent :

```
const john = {id:1, name:'John'};  
const supaJohn = {id:null, ...john, name:'John with power', powered:true};  
// supaJohn === {id:1, name:'John with power', powered:true}
```

Attention cela redonne les mêmes références, ce n'est pas du deep clone.

JavaScript ES6+



Classes

<https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Classes>

Simplifie la création d'objet et de son héritage au lieu d'utiliser **prototype** :

```
class Rectangle {  
    constructor(width, height) {  
        this.width = width;  
        this.height = height;  
    }  
    calcArea() {  
        return this.width * this.height;  
    }  
}  
const square = new Rectangle(10, 10);
```



JavaScript ES6+



Extends

<https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Classes/extends>

Pour ceux à l'aise avec la POO, les classes peuvent hériter d'une autre classe :

```
class Animal {  
  constructor(name) { this.name = name; }  
  move(distanceInMeters) {  
    console.log(`${this.name} moved ${distanceInMeters}m.`);  
  }  
}  
class Snake extends Animal {  
  constructor(name) { super(name); }  
}  
const s = new Snake('Hitman le Cobra');  
s.move(10);
```



JavaScript ES6+



Promises

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Promise

Permet de "simplifier" le code asynchrone (appel http, ...) à la place des **callback** :

```
const promise1 = new Promise(function(resolve, reject) {
  setTimeout(function() {
    resolve('foo');
  }, 300);
});
promise1
  .then(function(value) {
    console.log(value); // foo
  })
  .catch(function(err) {
    console.log(err);
});
```



JavaScript ES6+



Promises All

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Promise/all

Pour exécuter en parallèle plusieurs Promises, puis avoir le résultat dans l'ordre de déclaration dans un tableau :

```
Promise.all([ http.get(url1), http.get(url2) ])
  .then(([ response1, response2 ]) => {
    // ...
  }).catch(error => {
    // at least one request failed
});
```



JavaScript ES6+



Async Await

<https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Op%C3%A9rateurs/await>

Utilisé aussi pour l'asynchrone, `await` permet d'attendre la résolution de la promise avant de passer à la ligne suivante. `await` ne peut être utilisé que dans une fonction `async` :

```
const promise = new Promise(/*...*/);
async function waiting() {
  try {
    const res = await promise;
    console.log(res);
  } catch(err){
    console.log(err);
  }
}
waiting();
```



JavaScript ES6+



Arrow function

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Fonctions/Fonctions_fl%C3%A9ch%C3%A9es

Comme une fonction anonyme, mais ne redéfinit pas le `this` :

```
const materials = ['Hydrogen', 'Helium', 'Lithium'];
console.log(materials.map(
  (material) => { return material.substr(0, 2) }
));
```



JavaScript ES6+



Template de string

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Litt%C3%A9raux_gabarits

Améliore l'insertion de variable dans une chaîne **string**. Attention tout espace ou retour à la ligne est pris en compte :

```
// const fullname = 'Hello ' + firstname + ' ' + lastname;  
const fullname = `Hello ${firstname} ${lastname}`;
```



JavaScript ES6+



Module import export

<https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/export>

<https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/import>

Permet d'importer des fonctions / valeurs dans un fichier JavaScript :

```
// fichier a.js
export function multiplier(a, b = 1) {
    return a * b;
}
```

```
// fichier b.js
import { multiplier } from './a';
multiplier(10,5);
```

```
// fichier c.js
import { multiplier } from './a';
multiplier(42);
```



JavaScript ES6+



Pour aller plus loin

- Le CSS n'est pas son fort, mais il maîtrise le JavaScript : [Dr. Axel Rauschmayer](#) (une version gratuite contenant 90% de la version complète est dispo au format HTML)
- Une série d'ebooks gratuits sur GitHub : [You Don't Know JS](#)



TypeScript

TypeScript



Introduction

- Création en 2012
- Développé par Microsoft
- Sur-ensemble de JavaScript (ES6+)
- Doit être compilé en JavaScript pour le navigateur
- Extension fichier .ts
- Typage des valeurs
- Documentation <https://www.typescriptlang.org/docs/>

TypeScript



Les types

<https://www.typescriptlang.org/docs/handbook/basic-types.html>

- Boolean `let a:boolean = true;`
- Number `let a:number = 42;`
- String `let a:string = 'hello';`
- Enum `enum Color {Red, Green, Blue}` `let c: Color = Color.Green;`
- Tableau `let a:Array<number>` OU `let a:string[]` OU `let a:[string, number]`
- Any `let a:any`

TypeScript



Interfaces

<https://www.typescriptlang.org/docs/handbook/interfaces.html>

Permet de décrire ce que doit contenir un objet :

```
interface exampleConfig {  
    id: number;  
    name: string;  
    isNew?: boolean; // optionnal  
}  
const cfg:exampleConfig = {id:0, name:'Hello'};
```



TypeScript



Implements

<https://www.typescriptlang.org/docs/handbook/classes.html>

Les classes TypeScript peuvent implémenter une interface :

```
interface CanRun {  
    run(meters: number): void;  
}  
class User implements CanRun {  
    run(meters) {  
        console.log(`you run ${meters}m`);  
    }  
}
```



TypeScript



Les modificateurs

<https://www.typescriptlang.org/docs/handbook/classes.html>

Dans la continuité de la POO, on peut modifier la portée d'un élément. On retrouve **public**, **private**, **readonly**, **protected**, **static** et **abstract**.

```
class Animal {  
    public name: string; // tout est public par défaut  
    readonly created: number = Date.now();  
    constructor() {}  
    private move() {} // ne sera pas accessible en dehors de la classe  
    protected run() {} // similaire au private, mais permet l'appel dans une classe héritée  
}
```

TypeScript



Les décorateurs

<https://www.typescriptlang.org/docs/handbook/decorators.html>

Dans un usage avancé de TypeScript, on peut modifier le comportement d'une classe ou fonction avec un décorateur. Angular utilise les décorateurs pour déclarer un **Component** par exemple. Cela peut s'apparenter à de l'héritage dynamique.

```
@Component({ selector: 'home' }) // lance la fonction Angular Component
class HomeComponent {
  // HomeComponent a été modifié par Component()
  constructor() {}
}
```



Node NPM



Node NPM



Node

- Exécute du JavaScript (sans navigateur)
- Basé sur le moteur V8 de Chrome
- Utilisé pour développer des serveurs web en JS
- Mais aussi pour l'outillage de développement front-end



Node NPM



NPM

- Node Package Manager
- Utilisé pour la gestion des dépendances
- Il y a les dépendances projet et dépendances de développement
- Utilise un `package.json` et `package-lock.json`



Node NPM



NPM

<https://docs.npmjs.com/cli-documentation/cli>

```
// pour initialiser un projet, créer un package.json  
npm init  
// va lire le package.json pour installer les dépendances  
npm install  
// va installer et enregistrer la dépendance projet  
npm install --save xxxxxxx  
// va installer et enregistrer la dépendance de développement  
npm install --save-dev xxxxxxx  
// va installer les dépendances avec les versions exactes décrites dans le package-lock.json  
// videra le dossier node_modules  
npm ci
```

Node NPM



Package.json

```
{  
  "name": "myApp",  
  "version": "1.0.0",  
  "description": "a wonderfull app",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  "author": "me",  
  "license": "MIT",  
  "dependencies": {  
    "project-dependency": "^4.17.5"  
  },  
  "devDependencies": {  
    "dev-dependency": "^3.9.1"  
  }  
}
```



Node NPM



Semver

De base les versions indiquent une version minimum et maximum à installer. NPM cherchera à installer la version la plus récente correspondante à la fourchette indiquée.

La norme Semver veut que le premier chiffre soit la version majeur (possiblement des cassures avec les versions antérieures). Le second est le chiffre des versions mineurs (modifications importantes mais non cassantes). Enfin le troisième chiffre, pour les patchs.

Pour mieux comprendre de manière interactive : <https://semver.npmjs.com/>

```
"dependencies": {  
  "mustache": ">2.3.0",  
  "decktape": "^2.9.2",  
  "socket.io": "~1.7.3",  
  "express": "4.16.2"  
}
```



The logo for Angular consists of a large, solid red square on the left side of the slide. Inside this square, there are several thin, light gray diagonal lines that radiate from the bottom-left corner towards the top-right. A vertical white line extends from the bottom edge of the red square up to its midpoint. At this midpoint, there is a small horizontal bar composed of two segments: a white segment on the left and a red segment on the right.

Angular

Angular



Introduction

- Sorti en septembre 2016
- Développé par Google
- Basé sur TypeScript
- Réécriture complète par rapport à Angular JS



Angular



Extension VSCode

Libre à vous d'utiliser l'éditeur de code que vous voulez, mais si vous êtes sous VSCode :

- Angular Language Service
- EditorConfig
- TSLint
- Debugger for Chrome

Angular



Angular CLI

<https://angular.io/cli>

- Facilite grandement la mise en place d'un projet
- Outil en ligne de commande
- `npm install -g @angular/cli`
- `ng new my-project`

Angular



TP

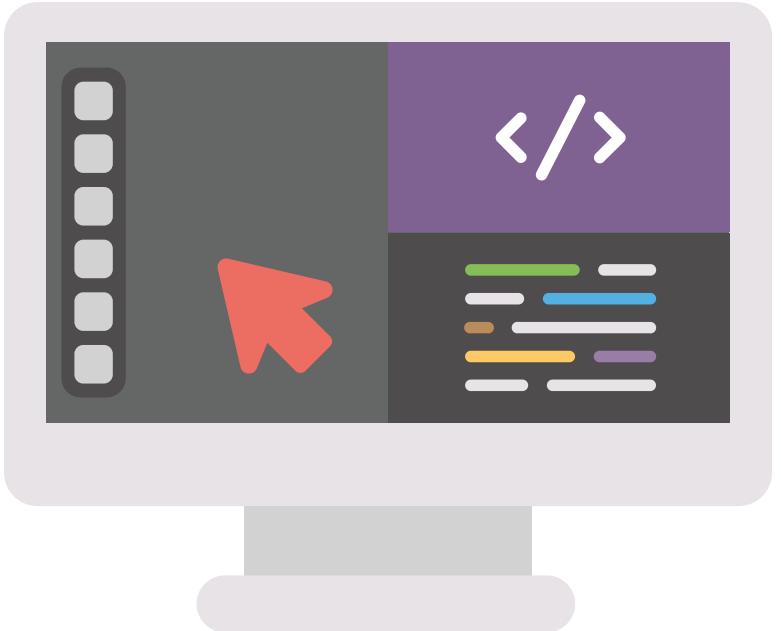
- Créez un projet

```
ng new my-project
```

- Lancez le server de développement

```
cd my-project
```

```
ng serve
```



Angular



Structure projet

- `e2e` pour y écrire vos scénario de test **end to end**
- `src/app` votre module et composant principal
- `src/assets` y placer toutes vos images et autres fichiers statiques
- `src/environments` vos variables d'environnements
- `src/index.html` la page html primaire
- `src/main.js` le script JS primaire
- `src/polyfill.js` pour activer les polyfills voulus
- `angular.json` la configuration de votre projet Angular

Angular



angular.json

Dans ce fichier .json vous pouvez déclarer vos assets, styles et scripts externes à vos composants :

```
"assets": [  
  "src/favicon.ico",  
  "src/assets"  
,  
  "styles": [  
    "src/styles.scss"  
,  
  "scripts": []]
```

Angular



Composant

Un composant se compose d'une partie html (la vue) :

```
<div><h1>Hello</h1></div>
```

D'une partie logique en TypeScript (le contrôleur) :

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-test-example', // pour la balise HTML
  templateUrl: './test-example.component.html', // la vue
  styleUrls: ['./test-example.component.scss'] // possible style CSS lié au composant
})
export class TestExampleComponent {
  constructor() { }
}
```



Angular



Composant

Une fois créé, il faut déclarer le composant dans le module qui l'utilisera :

```
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { TestExampleComponent } from './test-example/test-example.component';
@NgModule({
  declarations: [ AppComponent, TestExampleComponent ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Ensuite on peut l'utiliser en l'invoquant via sa balise HTML :

```
<app-test-example></app-test-example>
```



Angular



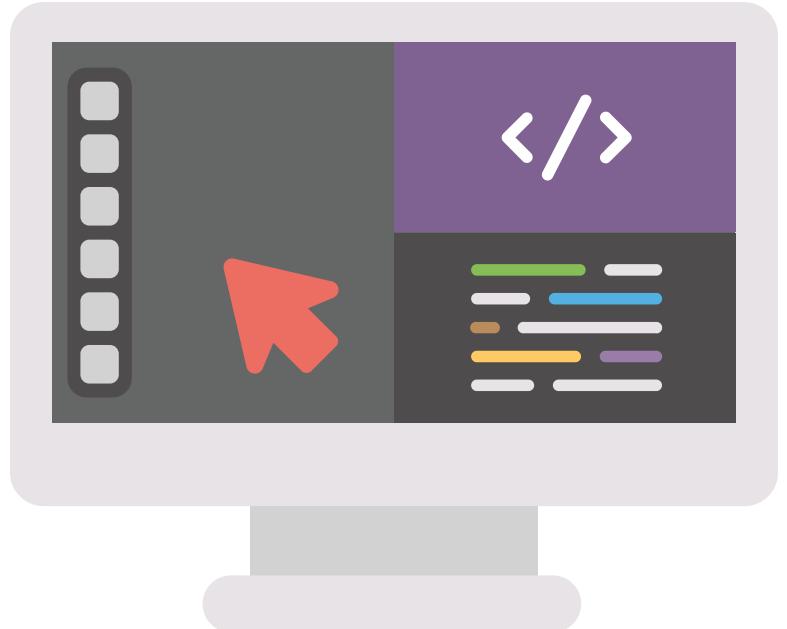
TP

- Créez un nouveau composant
(Angular CLI est votre ami)

```
ng generate component
```

- Insérez le composant pour le voir apparaître sur votre page

```
http://localhost:4200
```



Angular



Interpolation

<https://angular.io/guide/template-syntax#interpolation->

Permet d'afficher une valeur du contrôleur dans la vue :

```
<!-- .html -->
<div>{{ expression }}</div>
<h1>{{ title }}</h1>
```

```
// .ts
export class TestExampleComponent {
  expression = 'Bonjour';
  title = 'Le titre !';
}
```

Angular



Property binding

<https://angular.io/guide/template-syntax#property-binding--property->

On peut aussi **binder** une valeur à un attribut HTML :

```
<!-- .html -->
<img [src]="itemImageUrl">
<button [disabled]={!isValid}>Save</button>
```

```
// .ts
export class TestExampleComponent {
  itemImageUrl = '../img/logo.svg';
  isValid = true;
}
```



Angular



Event binding

<https://angular.io/guide/template-syntax#event-binding-event>

Dans l'autre sens (vue vers contrôleur) on peut binder des events :

```
<!-- .html -->
<button (click)="deleteHero()">Delete hero</button>
```

```
// .ts
export class TestExampleComponent {
  deleteHero() { /* ... */ }
}
```

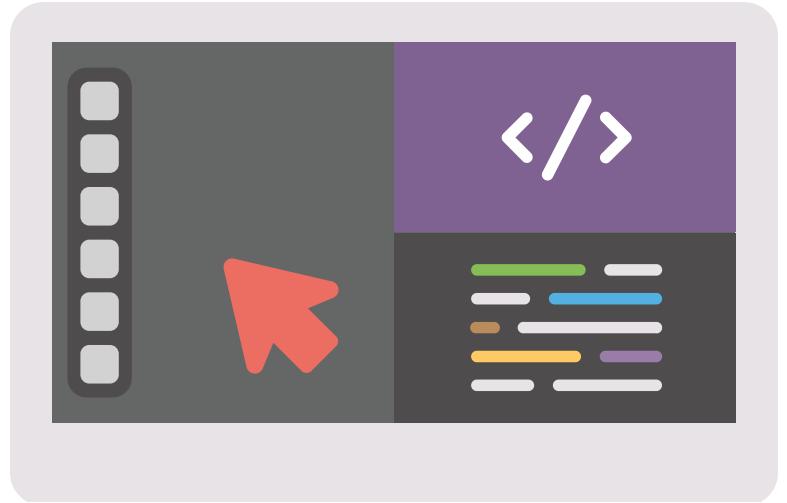


Angular



TP

- Dans le composant déjà créé, affichez un titre `{{ hello }}`
- En dessous, affichez la date du chargement de la page
- Ajoutez un bouton qui, au click, actualisera la date



Angular



Two-ways binding / Directive NgModel

<https://angular.io/guide/template-syntax#two-way-binding--->

<https://angular.io/api/forms/NgModel>

Pour envoyer et recevoir dans la vue des informations on utilisera généralement `ngModel`

```
<!-- .html -->
<input type="text" [(ngModel)]="userInput">
<!-- similaire via bind + fonction : -->
<input type="text" [ngModel]="userInput" (ngModelChange)="doSomething($event)">
```

```
// ! ATTENTION ! ngModel require un import dans votre module, voir slide suivante
// .ts
export class TestExampleComponent {
  userInput = 'valeur par défaut'; // sera mis à jour via la saisie dans l'input HTML
  doSomething(value) { console.log(value); }
}
```



Angular



Two-ways binding / Directive NgModel

<https://angular.io/guide/template-syntax#two-way-binding--->

<https://angular.io/api/forms/NgModel>

ngModel fait partie du module FormsModule. Il faut l'importer pour pouvoir l'utiliser :

```
// app.module.ts
import { FormsModule } from '@angular/forms';
@NgModule({
  imports: [
    /*...*/
    FormsModule
  ]
})
```



Angular



Attribute Directive NgClass

<https://angular.io/api/common/NgClass>

Permet de d'ajouter/enlever des classes CSS dynamiquement :

```
<!-- .html -->
<div [ngClass]="currentClasses">Div avec class assignées directement</div>
<div [ngClass]="'first': isTest">Div avec class 'first' sous condition</div>
```

```
// .ts
export class TestExampleComponent {
  currentClasses = ['highlight', 'valid'];
  isTest = true;
}
```



Angular



Attribute Directive NgStyle

<https://angular.io/api/common/NgStyle>

Similaire au NgClass, il permet d'injecter du style CSS. De préférence, utilisez NgClass pour un code/structure plus propre.

```
<!-- .html -->
<div [ngStyle]="'font-style': myVar">...</div>
```

```
// .ts
export class TestExampleComponent {
  myVar = '3rem';
}
```



Angular



Structural Directive NgIf

<https://angular.io/api/common/NgIf>

Permet d'ajouter ou enlever tout l'élément (et son contenu) selon une condition/variable :

```
<!-- .html -->
<div *ngIf="show">Text to show</div>
```

```
// .ts
export class TestExampleComponent {
  show = true;
}
```

Angular



Structural Directive NgForOf

<https://angular.io/api/common/NgForOf>

Permet de boucler sur un tableau pour afficher ses valeurs :

```
<!-- .html -->
<li *ngFor="let item of items; trackBy: trackByFn">{{item.name}}</li>
```

```
// .ts
interface MyItems { id: number; name: string; }

export class TestExampleComponent {
  items: MyItems[] = [{id:0, name:'toto'}, {id:1, name:'blup'}];
  // optimise le rendu DOM lors des mises à jours de `items`
  trackByFn(i: number, el: MyItems): number { return el.id; }
}
```



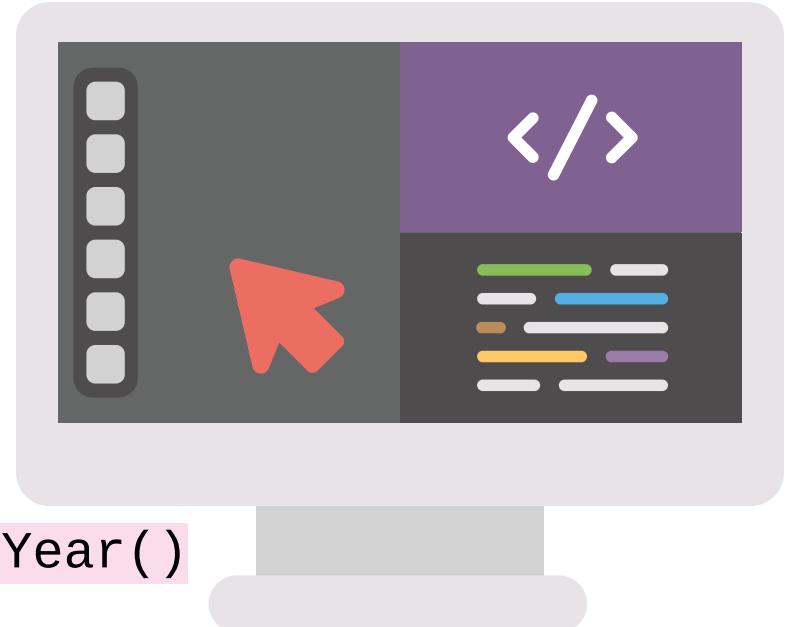
Angular



TP

Dans le composant qui affiche la date :

- Ajoutez un `input type="date"`
- La date du chargement de la page doit s'afficher dedans
- Si l'utilisateur change la valeur de l'input, il faut appeler `checkIfPastYear()`
- `checkIfPastYear()` doit alors changer une valeur `isPastYear`
- Créez deux `div`, l'une avec comme texte `Année courante`, l'autre `Année passée`
- Selon `isPastYear`, affichez la bonne div
- Dans la div `Année passée`, listez les années entre notre année courante et celle choisie
- Dans cette liste, les dates paires doivent avoir la classe CSS `yearPair`



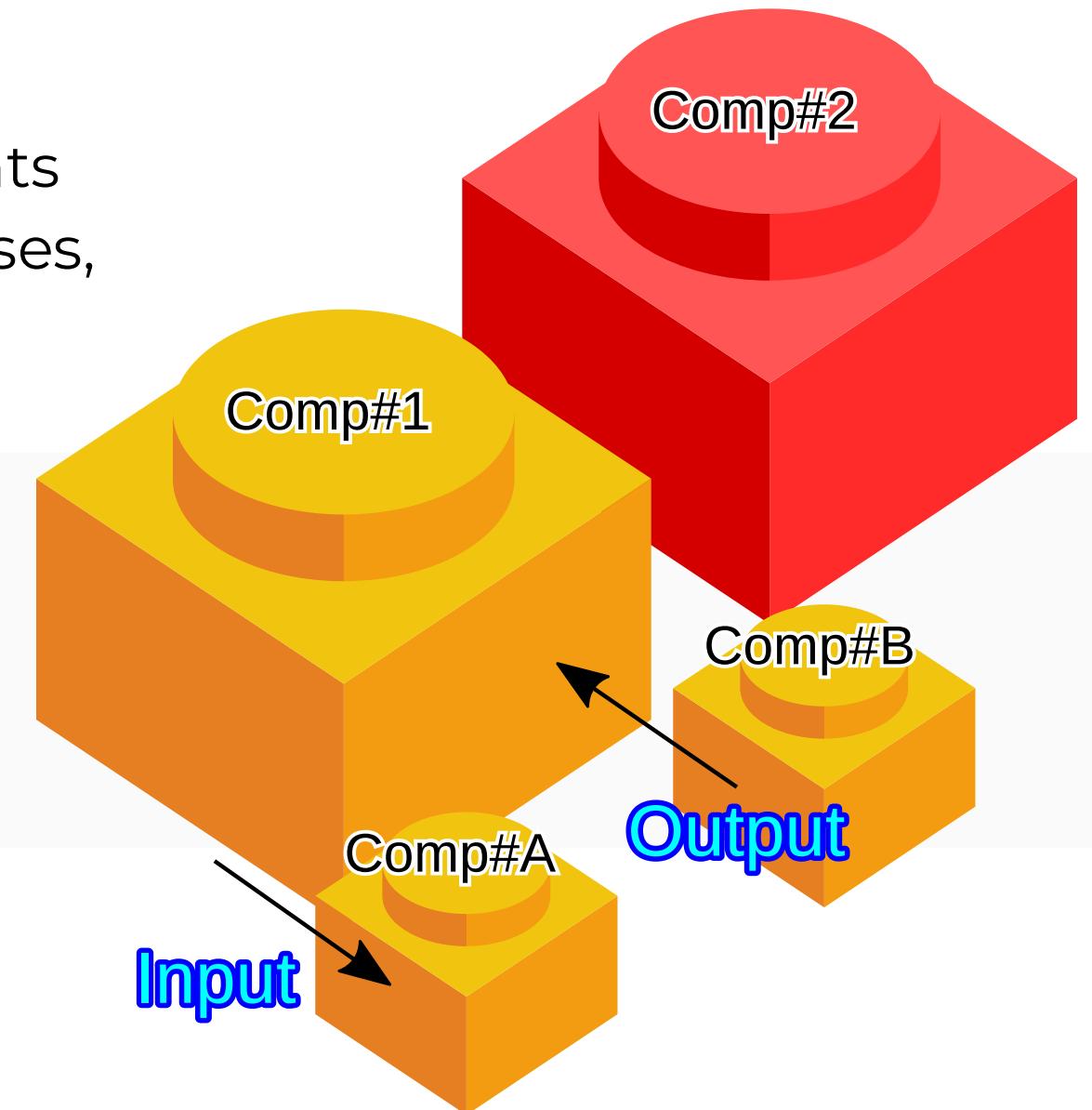
Angular



Imbrication composants

Dans une application Angular, les composants s'imbriquent entre eux, tels des poupées russes, parents - enfants. On peut vite arriver à une structure pyramidale.

```
<comp1>
  <compA></compA>
  <compB></compB>
</comp1>
<comp2></comp2>
```



Angular



Input (parent)

<https://angular.io/guide/template-syntax#input-and-output-properties>

Les inputs vont permettre de transmettre une valeur à un composant enfant tel un attribut HTML. Voici la partie parent :

```
<!-- .html composant parent -->
<div>
  <composant-enfant [title]="titleForChild"></composant-enfant>
</div>
```

```
// .ts composant parent
export class ComponentParent {
  titleForChild = 'Bonjour !';
}
```



Angular



Input (enfant)

<https://angular.io/guide/template-syntax#input-and-output-properties>

La partie enfant qui reçoit :

```
<!-- .html composant enfant -->
<div>
  <h1>{{ title }}</h1>
</div>
```

```
// .ts composant enfant
export class ComponentChild {
  @Input() title: string;
}
```

Angular



Output (parent)

<https://angular.io/guide/template-syntax#input-and-output-properties>

Les outputs vont permettre de lancer une fonction du parent lorsque l'enfant émet un événement :

```
<!-- .html composant parent -->
<div>
  <composant-enfant (sayHello)="doSomething($event)"></composant-enfant>
</div>
```

```
// .ts composant parent
export class ComponentParent {
  doSomething(ev) { /*...*/ }
}
```

Angular



Output (enfant)

<https://angular.io/guide/template-syntax#input-and-output-properties>

Pour déclarer et lancer un événement :

```
<!-- .html composant enfant -->
<div>
  <button type="button" (click)="sayHello.emit('Hi ! Bonjour !')">Emit event</button>
</div>
```

```
// .ts composant enfant
export class ComponentChild {
  @Output() sayHello = new EventEmitter<string>();
}
```





Lifecycle des composants

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

Les composants ont un cycle de vie, de la création à la destruction. Plusieurs fonctions peuvent être lancées si le composant les implémente. Par ordre de lancement :

- `constructor` lancé au tout début de l'initialisation de la classe TypeScript.
- `ngOnChanges` lancé à la création puis sera relancé à chaque modification d'un `@Input`.
- `ngOnInit` lancé une seule fois à la création. Contrairement au `constructor`, ici on a accès aux `@Input`.
- `ngOnDestroy` lancé à la suppression du DOM du composant.

D'autres fonctions sont disponibles telles que `ngDoCheck`, `ngAfterContentInit`, `ngAfterContentChecked`, `ngAfterViewInit` et `ngAfterViewChecked`.

Angular



TP

- Scindez le composant en deux, un parent, l'autre enfant
- Une partie avec l'input date qui recoit en `@Input` la valeur initiale
- En `@Output` le composant doit retourner si l'année est
 - passé | courant | futur
- Dans le composant parent, affichez la liste des `div` selon la valeur reçue



Angular



Pipe

<https://angular.io/guide/pipes>

Les pipes sont des fonctions autonomes, généralement pures, recevant des paramètres et retournant une valeur en conséquence. On peut créer ses propres Pipes : `ng generate pipe`

Liste des Pipes fournis par Angular <https://angular.io/api?type=pipe>

```
<!-- The hero's birthday is 02/04/2050 -->
<p>The hero's birthday is {{ birthday | date:"dd/MM/yyyy" }}</p>

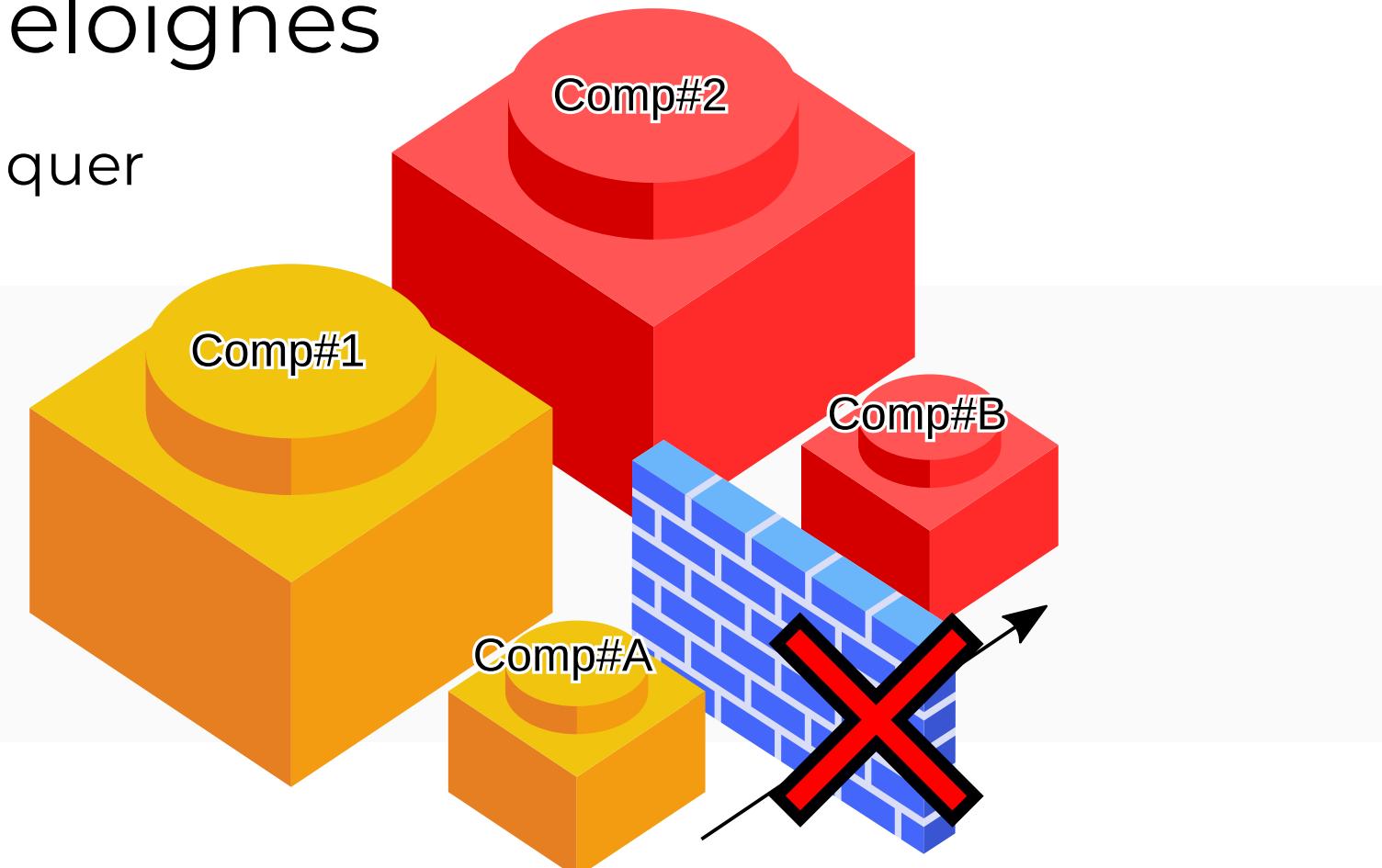
<!-- Total panier : €1,011.00 -->
<div>Total panier : {{ price | currency:"EUR" }}</div>
```



Communication Composants éloignés

Les composants A et B ne peuvent communiquer directement (pas de lien de parenté).

```
<comp1>
  <compA></compA>
</comp1>
<comp2>
  <compB></compB>
</comp2>
```



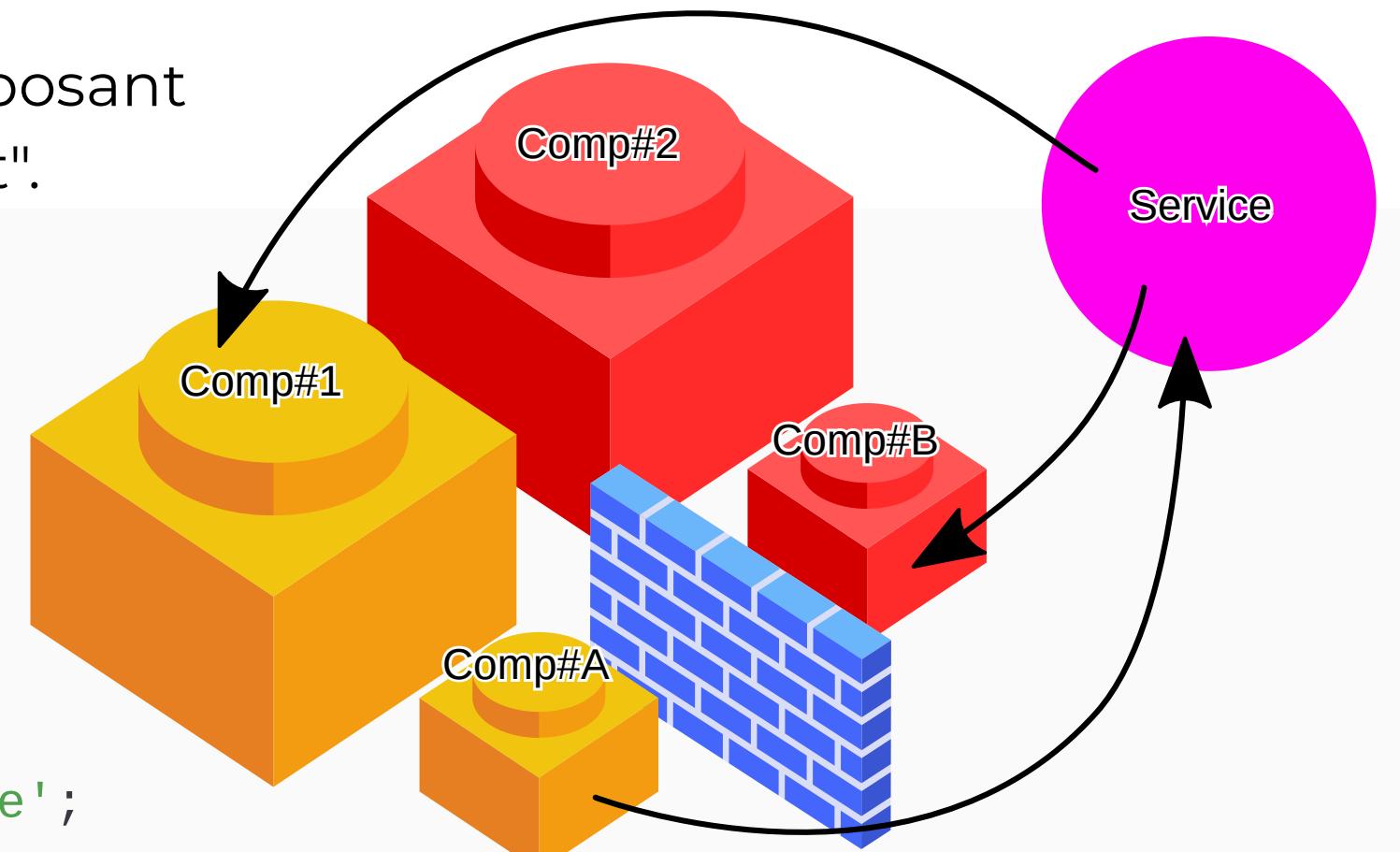


Communication Composants éloignés

Avec un service externe, n'importe quel composant peut communiquer / écouter un "événement".

```
<comp1>
  <compA></compA>
</comp1>
<comp2>
  <compB></compB>
</comp2>
```

```
// dans les contrôleurs des composants
import { HelloService } from './hello.service';
/* injection du service */
```



Angular



Service

<https://angular.io/tutorial/toh-pt4>

<https://angular.io/guide/singleton-services>

Un service est une instance de classe, à référence unique (singleton) dans un module.

Par la suite dans les contrôleurs de composants, on pourra l'injecter / importer sa référence pour appeler ses méthodes, lire ses valeurs ou écouter des "événements".

```
ng generate service
```

```
import { Injectable } from '@angular/core';
@Injectable({ providedIn: 'root' }) // déclaration de son module simplifié depuis Angular 6
export class HelloService {
  constructor() {}
}
```



Angular



Service Injection

```
// dans un contrôleur composant
import { Component, OnInit } from '@angular/core';
import { HelloService } from 'src/app/hello.service'; // import du service
@Component({
  selector: 'app-example',
  templateUrl: './example.component.html',
  styleUrls: ['./example.component.scss']
})
export class ExampleComponent implements OnInit {
  constructor(private helloService: HelloService) {} // injection du service
  ngOnInit() {
    this.helloService.sayHello(); // appel du service
  }
}
```



Angular



TP

- Utilisez un Pipe pour afficher la date sélectionnée de notre précédent composant
- Créez un service et utilisez le pour y stocker la valeur de la date sélectionnée





ReactiveX

ReactiveX



Introduction

- Librairie pour "faciliter" la programmation asynchrone
- Utilise des **observables** et **subjects**, proches des événements JavaScript
- Fournit une multitude de fonctions pour filtrer / concaténer / modifier ce qui est reçu des **observables**
- Utilisé dans Angular (par exemple pour le client HTTP)
- Librairie multi langages (Java, C#, Python, ...)
- <http://reactivex.io/> et le site dédié à la version JavaScript <https://rxjs.dev/>

ReactiveX



Exemple

```
const rate = 1000;
const button = document.querySelector('button');
let count = 0;
let lastClick = Date.now() - rate;

button.addEventListener('click', () => {
  // évite le spam des clicks, 1sec de tempo
  if ((Date.now() - lastClick) >= rate) {
    count++;
    console.log(`Clicked ${count} times`);
    lastClick = Date.now();
  }
});
```

```
import { fromEvent } from 'rxjs';
import { throttleTime, scan } from 'rxjs/operators';
const rate = 1000;
const button = document.querySelector('button');

fromEvent(button, 'click').pipe(
  // fait la tempo des 1sec
  throttleTime(rate),
  // stock pour nous le count
  scan((count) => { return count + 1, 0 })
)
.subscribe((count) => {
  console.log(`Clicked ${count} times`)
});
```



ReactiveX



Fonctions Opérateurs

- Liste complète <https://rxjs.dev/api?type=function>
- Mais y en a beaucoup (trop) ! Il y a une "aide" pour trouver celui dont tu as besoin
<https://rxjs.dev/operator-decision-tree>
- `filter` similaire au filter Array JS
- `distinctUntilChanged` filtre prédéfini, stop si la valeur est identique à la précédente
- `map` similaire au map Array JS
- `fromEvent` et `from` pour convertir en Observable
- `throttleTime` et `debounceTime` pour temporiser les déclenchements
- `forkJoin` similaire au Promise.all

ReactiveX



Subject

<https://rxjs.dev/api/index/class/Subject>

Exemple de création puis souscription d'un "événement" :

```
import { Subject } from 'rxjs';
const mySubject = new Subject();
// 
// plus tard...
//
mySubject.next('hello !');
```

```
// quelque part (autre ou même fichier)
//
mySubject.subscribe((data) => {
  console.log(`observer: ${data}`);
});
// observer: hello !'
```

Il peut y avoir plusieurs souscriptions, tous vont recevoir l'information. Utile dans un service qui va émettre une donnée à qui l'a souscrit.

ReactiveX



BehaviorSubject

<https://rxjs.dev/api/index/class/BehaviorSubject>

Très proche du Subject, il est initialisé avec une valeur par défaut. De plus, à la souscription, il retourne la valeur courante.

```
import { BehaviorSubject } from 'rxjs';
const myBehaviorSubject = new BehaviorSubject('coucou');
myBehaviorSubject.subscribe((value) => {
  console.log('sub A', value);
});
myBehaviorSubject.next('pwet');
myBehaviorSubject.subscribe((value) => {
  console.log('sub B', value);
});
myBehaviorSubject.next('hello');
```

```
// sub A coucou
// sub A pwet
// sub B pwet
// sub A hello
// sub B hello
```



Angular (suite)

Angular



HTTP

<https://angular.io/tutorial/toh-pt6>

Angular fournit un client HTTP pour faire vos requêtes HTTP.

```
// dans votre app.module.ts
import { HttpClientModule } from '@angular/common/http';
@NgModule({
  declarations: [/*...*/],
  imports: [
    BrowserModule,
    HttpClientModule,
    /*...*/
  ]
})
```



Angular



HTTP

<https://angular.io/tutorial/toh-pt6>

Le client HTTP est accessible sous forme de service.

```
// dans votre composant ou service

import { HttpClient } from '@angular/common/http';
import { catchError } from 'rxjs/operators';

// injection de la dépendance
constructor(private http: HttpClient) {}
```



Angular



HTTP

<https://angular.io/tutorial/toh-pt6>

Exemple d'une requête GET

```
this.http.get<any>('https://reqres.in/api/users?page=2')
  .pipe(
    // ici vos operators
    catchError((err) => {
      // ici votre gestion d'erreur
      // on throw l'erreur pour stopper l'Observable
      throw err;
    })
  )
  // si pas de subscribe, pas d'exécution de la requete HTTP !
  .subscribe((data) => {
    console.log(data);
  });
}
```





HTTP Interceptor

<https://angular.io/guide/http#intercepting-requests-and-responses>

En usage avancé mais assez récurrent, HTTP Interceptor vous aidera !

- Pour ajouter à toute requête un header, par exemple de token d'auth

<https://angular.io/guide/http#set-default-headers>

- Pour catch un code retour HTTP `401` pour logger et / ou rédiriger vers la page de login

<https://angular.io/guide/http#logging>

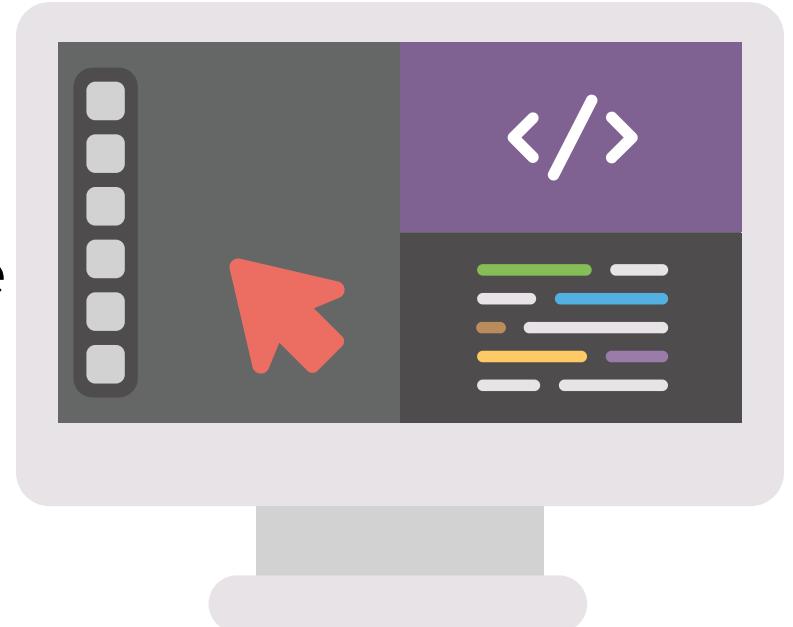
Angular



TP

- Créez un nouveau composant sans lien de parenté avec les autres
- Ce composant doit écouter / afficher la date stockée dans le service
- Ce composant doit récupérer les films via l'API **TheMovieDB** selon l'année recue

[https://api.themoviedb.org/3/discover/movie?
api_key=0411f3dca4189da999b2f9d32ad5cd39&page=1&year=1990](https://api.themoviedb.org/3/discover/movie?api_key=0411f3dca4189da999b2f9d32ad5cd39&page=1&year=1990)



- Affichez les résultats dans le composant

Angular



Routeur

<https://angular.io/guide/router>

Le router va permettre de :

- Naviguer de page en page
- Charger / décharger des composants selon les pages
- Passer des paramètres dans l'url
- Protéger l'accès à une page

Angular



Routeur

<https://angular.io/guide/router>

Déclaration du router, généré via Angular CLI

```
// app-routing.module.ts
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
const routes: Routes = [
/* déclaration de vos routes */
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```



Angular



Routeur

<https://angular.io/guide/router>

Import dans le module principal

```
// app.module.ts
/* ... */
import { AppRoutingModule } from './app-routing.module';
@NgModule({
  imports: [
    /* ... */
    AppRoutingModule
  ]
})
```



Angular



Routeur

<https://angular.io/guide/router>

Déclaration de la cible HTML du routing.

Cela indique l'endroit où les composants seront chargés selon l'url.

```
<!-- app.component.html -->
<router-outlet></router-outlet>
```



Angular



Routeur

<https://angular.io/api/router/Route>

Déclaration des routes

```
// app-routing.module.ts
export const routes: Routes = [
  { path: '', pathMatch: 'full', redirectTo: '/home' },
  { path: 'home', component: HomeComponent },
  { path: 'users', component: UsersComponent },
  { path: 'user/:userId', component: UserComponent }
];
```



Angular



Routeur

<https://angular.io/api/router/RouterLink>

Déclaration des liens

```
<!-- HTML -->
<a href="" [routerLink]=["/user", user.id]>See user</a>
```

```
// TypeScript
import { Router } from '@angular/router';
/* ... */
constructor(private router: Router) {}
/* ... */
this.router.navigate(['/user', this.user.id]);
```



Angular



Routeur

<https://angular.io/api/router/ActivatedRoute>

Récupérer les paramètres

```
import { ActivatedRoute } from '@angular/router';
/* ... */
constructor(private route: ActivatedRoute) {}
ngOnInit() {
  this.route.paramMap.subscribe((params: ParamMap) => {
    const id = params.get('userId');
    /* ... */
  });
}
```



Angular



Routeur

<https://angular.io/api/router/Route>

Routes "enfants"

```
export const routes: Routes = [
{ path: 'user/:userId', component: UserComponent,
children: [
  { path: '', pathMatch: 'full', redirectTo: 'info' },
  { path: 'info', component: UserInfoComponent },
  { path: 'config', component: UserConfigComponent },
  { path: 'bonus', component: UserBonusComponent }
]
}
];
```



Angular



Routeur

<https://angular.io/api/router/Route>

Routes "enfants"

```
<!-- app-user-component -->
<router-outlet></router-outlet>
```



Angular



Guards

<https://angular.io/api/router/CanActivate>

Tel un gardien, permet de bloquer ou non un accès à une route.

! Cela ne garantit aucune sécurité ! C'est au back / api d'avoir la vraie sécurité.

Déclaration d'un guard, doit retourner true ou false :

```
@Injectable({ providedIn: 'root' })
export class LoggedInGuard implements CanActivate {
  constructor(private router: Router, private userService: UserService) { }
  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
    if (this.userService.isLoggedIn()) { return true; }
    else { this.router.navigate(['/login']); return false; }
  }
}
```



Angular



Guards

<https://angular.io/api/router/Route>

Utilisation d'un guard dans une route :

```
import { LoggedInGuard } from 'src/app/logged-in.guard';
/* ... */
export const routes: Routes = [
  { path: 'user', component: UserComponent, canActivate: [LoggedInGuard] }
];
```

Angular

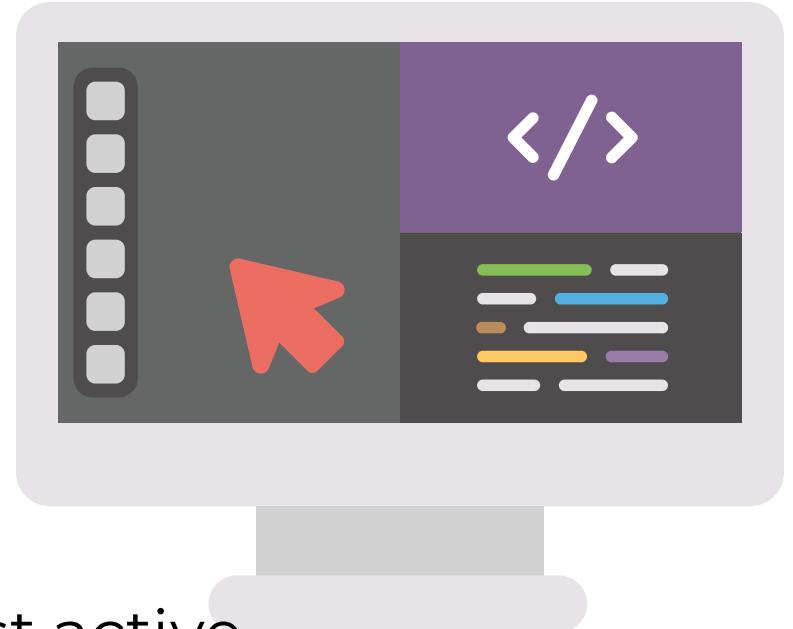


TP

- Créez deux routes : l'une pour afficher nos précédents travaux, l'autre pour afficher un composant vierge
- Créez un composant "Menu" pour aller d'une route à l'autre

Bonus :

- Changez le style CSS des liens "Menu" si la route correspondante est active
- Ajoutez un guard sur la route du composant vierge, on doit avoir une chance sur deux d'y accéder



Angular



Reactive Forms

<https://angular.io/guide/reactive-forms>

Permet de gérer / valider un formulaire HTML sans ngModel

! Cela ne garantit aucune sécurité ! C'est au back / api d'avoir la vraie sécurité.

```
// app.module.ts
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
@NgModule({
  /* ... */
  imports: [
    FormsModule,
    ReactiveFormsModule // ajout de la dépendance Reactive Forms
  ]
})
```

Angular



Reactive Forms

<https://angular.io/guide/reactive-forms>

Déclaration des `FormControl` et `validators`.

```
import { FormControl, FormBuilder, FormGroup, Validators } from '@angular/forms';
/* ... */

export class ExampleComponent {
  passwordCtrl: FormControl; // pour chaque input
  userForm: FormGroup; // pour chaque formulaire
  constructor(fb: FormBuilder) {
    this.passwordCtrl = fb.control('valeur par défaut', [Validators.minLength(6)]);
    this.userForm = fb.group({
      password: this.passwordCtrl
    });
  }
  submitForm() { console.log(this.userForm.value); }
}
```



Angular



Reactive Forms

<https://angular.io/guide/reactive-forms>

Partie HTML

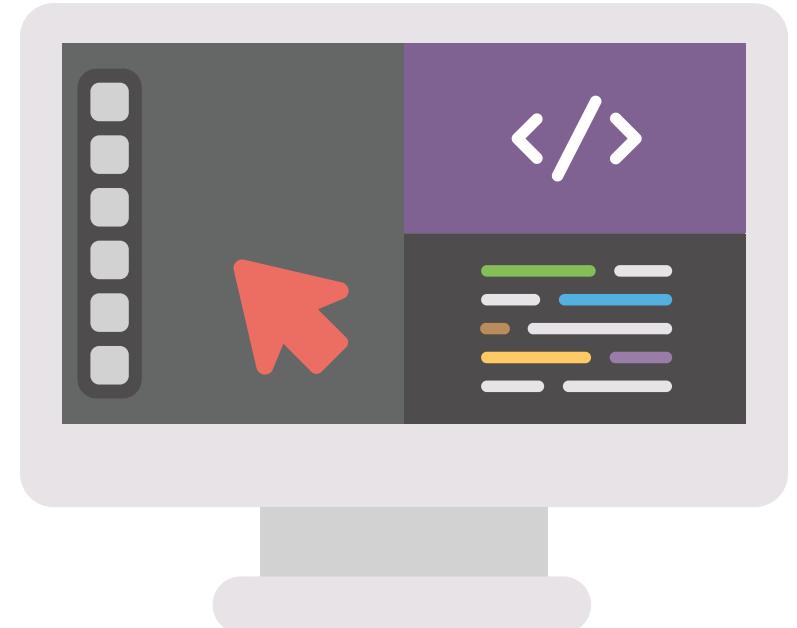
```
<form (ngSubmit)="submitForm()" [formGroup]="userForm">
  <label>Password</label>
  <input type="password" formControlName="password">
  <!-- affiche un message si l'input a été touché par l'utilisateur (dirty)
       et si il y a l'erreur (hasError) -->
  <div *ngIf="passwordCtrl.dirty && passwordCtrl.hasError('minlength')">
    Password should be 6 characters min
  </div>
  <button type="submit" [disabled]="userForm.invalid">Register</button>
</form>
```

Angular



TP

- Dans le composant vide précédent, créez un formulaire
- Il doit comporter un champ `username` et un champ `password`
- Les deux champs sont requis
- `password` n'est valide que si il fait 6 caractères minimum
- Affichez bien les messages d'erreurs en conséquence



Bonus :

- Faire que `password` ne soit valide que s'il y a au moins une majuscule, une minuscule, un chiffre et un caractère spécial.

Angular



Test unitaire

<https://angular.io/guide/testing>

- Intégré via Angular CLI `ng test`
- Utilise Jasmine
- Teste tous les `**.spec.ts` (`ng generate`)
- On peut tester facilement les fonctions pures

Angular



Test unitaire

<https://angular.io/guide/testing>

Example d'un Pipe qui doit transformer la première lettre en majuscule :

```
@Pipe({name: 'titlecase', pure: true})
export class TitleCasePipe implements PipeTransform {
  transform(input: string): string {
    return input.length === 0 ? '' :
      input.replace(/\w\S*/g, (txt => txt[0].toUpperCase() + txt.substr(1).toLowerCase() ));
  }
}
```



Angular



Test unitaire

<https://angular.io/guide/testing>

Le test correspondant :

```
// .spec.ts
describe('TitleCasePipe', () => {
  let pipe = new TitleCasePipe();
  it('transforms "abc" to "Abc"', () => {
    expect(pipe.transform('abc')).toBe('Abc');
  });
});
```



Angular



Test e2e

<https://angular.io/guide/testing>

- Intégré via Angular CLI `ng e2e`
- Utilise **Protractor**
- Permet de tester des scénarios de navigation dans l'application
- Utilise les scripts qui sont dans `/e2e/`

Angular



Test e2e

<https://angular.io/guide/testing>

Exemple de mini scénario

```
// .e2e-spec.ts
describe('Protractor Demo App', function() {
  it('should have a title', function() {
    browser.get('http://localhost:4200/');
    element(by.id('gobutton')).click();
    expect(browser.getTitle()).toEqual('Welcome to Hello World!');
  });
});
```



Angular



Build

<https://angular.io/guide/build>

Pour lancer le build via Angular CLI `ng build --prod --configuration production`

Utilise l'environnement `production` décrit dans `angular.json`. Vous pouvez rajouter autant de configurations que vous voulez.

```
"configurations": {  
  "production": {  
    "fileReplacements": [  
      {  
        "replace": "src/environments/environment.ts",  
        "with": "src/environments/environment.prod.ts"  
      }  
    ],
```



Angular



Build

<https://angular.io/guide/build>

Pour accéder aux variables d'env, un simple `import` suffit :

```
import { environment } from 'src/environments/environment'; // toujours cibler ce fichier
/* ... */
if(environment.production) {
  /* ... */
}
```



Angular



Build

<https://angular.io/cli/build>

Configuration production par défaut :

```
// angular.json
"optimization": true, // optimization, minify
"outputHashing": "all", // cache busting filename
"sourceMap": false, // sourcemap for debug
"extractCss": true, // generate .css files
"namedChunks": false, // filename for lazy load
"aot": true, // ahead of time compilation
"extractLicenses": true, // separate licences to a file
"vendorChunk": false, // separate vendor to a new bundle
"buildOptimizer": true, // aot optimization
```



Angular



Vers l'infini et au delà

<https://www.youtube.com/watch?v=TJrybmdC81U>

- Internationalisation i18n <https://angular.io/guide/i18n>
- Server Side Rendering <https://angular.io/guide/universal>
- PWA <https://angular.io/guide/service-worker-intro>



Ceci est un chat déguisé en Buzz l'éclair





Des questions ?





Merci à vous !

