

Angular 10

Christophe DUFOUR

Angular

- Framework Front JavaScript
- Développé par Google
- Première version: septembre 2016
- Réécriture de Angularjs (projet distinct)
- Site officiel: <https://angular.io/>
- Livré “avec le plein” (router, formulaires, httpclient, etc.)
- Utilise **typescript**
- Concurrents: React, Vuejs, Svelte



Avantages du Framework

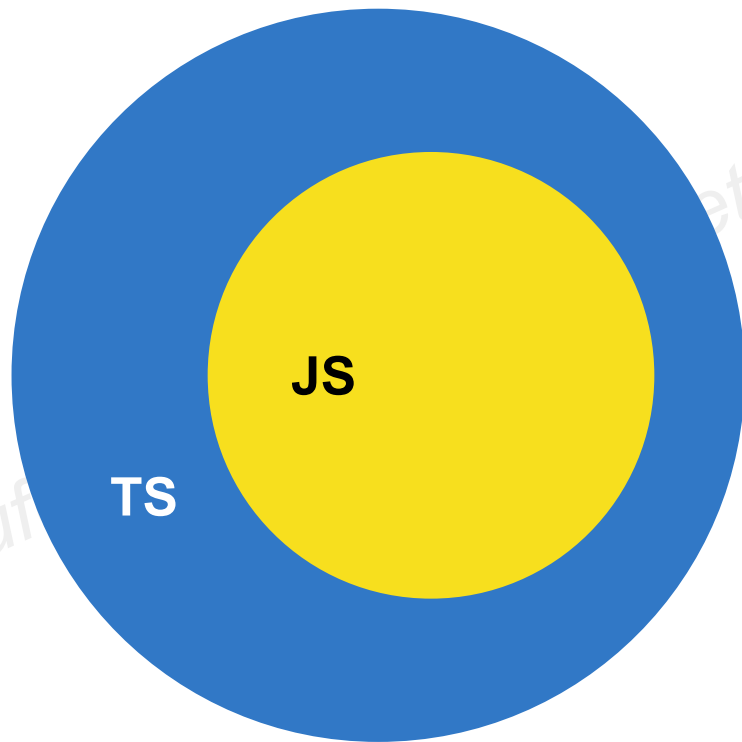
- Nombreuses fonctionnalités “out-of-the-box” (binding, routage, types, etc.)
- Cohérence
- Méthologie
- Facilite le découpage par lots et le travail collaboratif
- Scalabilité
- Modularité
- Rapidité de prise en main pour un nouveau développeur participant au projet (par rapport à du natif)
- Compatibilité (polyfill, etc.)

Typescript

- Langage open source créé par Microsoft
- Première version: février 2012
- “Sur-ensemble” de Javascript = JS + new features
- Ajoute des nouvelles fonctionnalités au JS pur comme:
 - Le typage statique. Exemple: `name: string = “Chris”`
 - L'utilisation de décorateurs de classe. Exemple: `@Injectable()`
 - La création d'interface pour typer des objets
 - L'organisation modulaire de l'application par mécanisme d'import/export
 - OO étendu: classes, constructeur, opérateurs de portée, etc.
- Site officiel: <https://www.typescriptlang.org/>
- Transpilation en JS standard (TS n'est pas directement interprétable par les navigateurs)



TypeScript



TypeScript

TS TypeScript Download Docs Handbook Community Playground Tools

Playground TS Config Examples What's New Browser Refresh Required

v4.0.5 Run Export

Syntaxe TS
Non interprétable par un navigateur web

Syntaxe ES5
Interprétable par un navigateur web

transpilation

```
1 interface Student
2   name: string;
3   grade: number;
4 }
5
6 const s1: Student = {name: "Chris", grade: 20};
7 const s2: Student = {name: "Toto", grade: 0};
8 const s3: Student = {name: "Manu", grade: 10};
9
10 const students: Student[] = [s1, s2, s3];
11 students.forEach((student) => console.log(student.name));
```

```
"use strict";
var s1 = { name: "Chris", grade: 20 };
var s2 = { name: "Toto", grade: 0 };
var s3 = { name: "Manu", grade: 10 };
var students = [s1, s2, s3];
students.forEach(function (student) { return console.log(student.name); });
```


Angular CLI (ng)

- Command Line Interface pour Angular
- Offre un gain de productivité en automatisant certaines tâches:
 - Création d'une nouvelle application
 - Génération d'un nouveau composant
 - Démarrage d'un serveur de développement

Angular CLI (ng)

- Installation: **npm install -g @angular/cli**
- Nouveau projet: **ng new projectName**
- Démarrage d'un serveur de développement: **ng serve [--port 4200]**
 - <http://localhost:4200/>
- Générer un nouveau composant:
 - **ng generate component componentName**
 - **ng g c componentName**

Port écouté par défaut
mais peut être modifié
par l'option --port



Angular CLI (ng)

```
chris@aston:~$ ng version
```



```
Angular CLI: 10.2.0
```

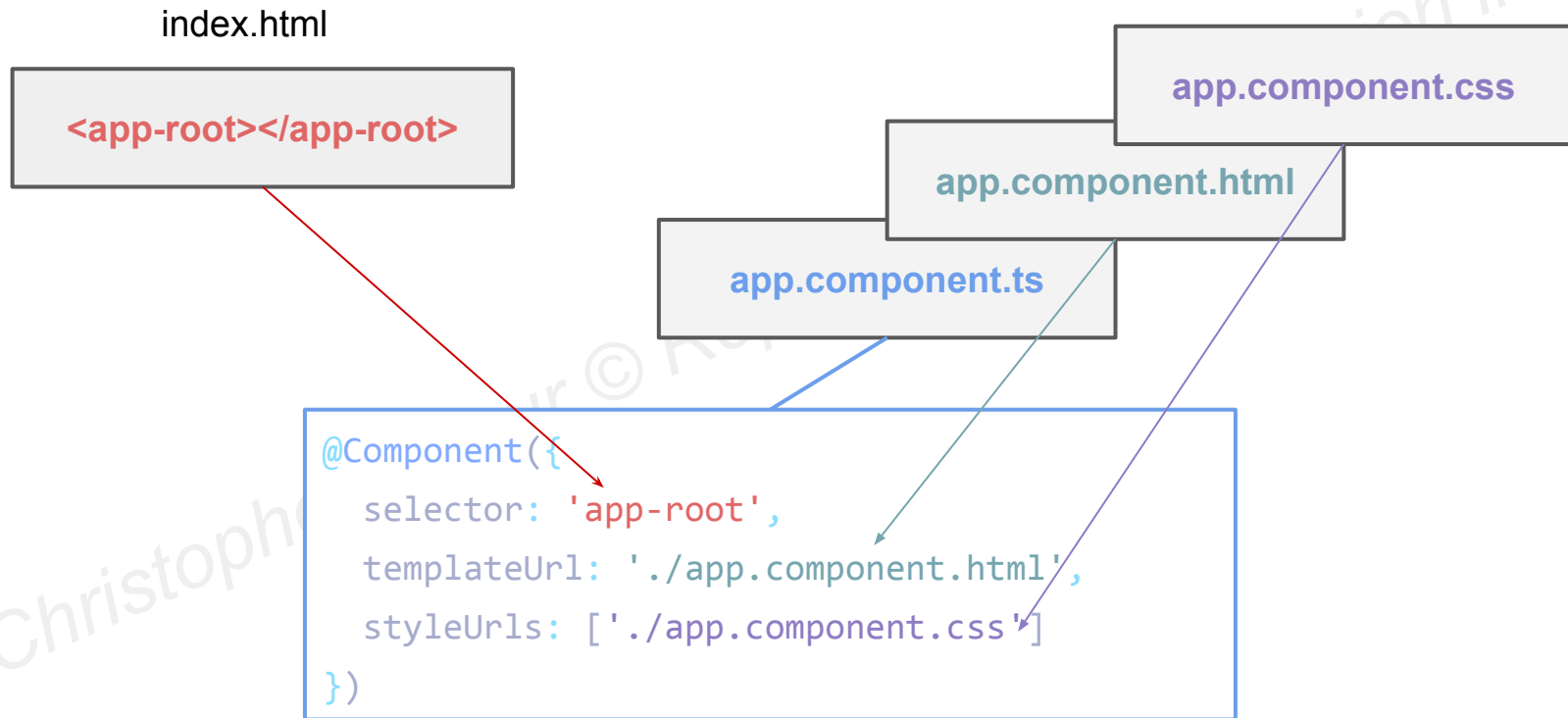
```
Node: 12.19.0
```

```
OS: linux x64
```

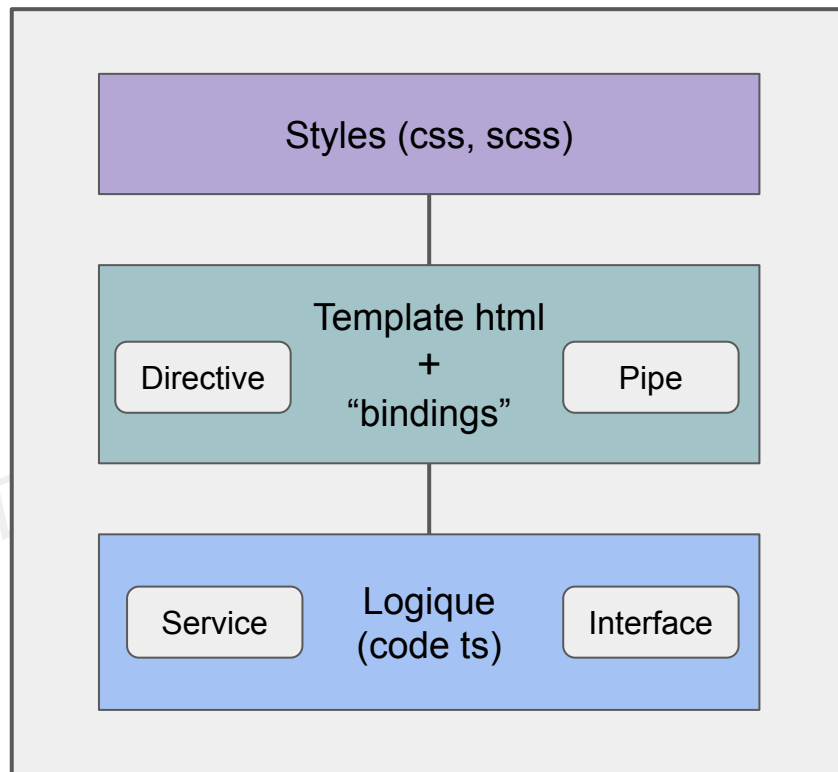
Composant

- Plus petite unité logique dans Angular
- Réutilisable
- Imbricable (nestable)
 - S'insère dans le template d'un composant de niveau supérieur
- Lié au DOM (template)
- Se décompose généralement en trois sections:
 - La logique: code ts
 - L'apparence (ui): code html statique + expressions dynamiques
 - Le style: code css/sass/scss,...
- Les trois sections peuvent être placées dans des fichiers distincts ou réunies dans le seul fichier .ts (décorateur @Component)

Composant en trois fichiers



Composant



Component sur un seul fichier (single-file component)

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

@Component({
  selector: 'app-message',
  template: `
    <h2>Message Component</h2>
    <p>{{ message }}</p>
  `,
  styles: ['h2 { color: orange }']
})
export class MessageComponent {
  message: string = "Coucou";
}
```

message.component.ts

UI embarqué dans le fichier .ts par la clé **template**. Utilisation des guillemets inversés (``) pour faciliter le formatage

Styles embarqués dans le fichier .ts par la clé **styles**.

Message Component

Coucou

Binding syntax

- `{{ }}` : binding de lecture, affiche une valeur string ou convertible en string définie dans la classe `.ts`
- `[]` : property binding, associe un attribut/propriété de balise à une valeur définie dans la classe `.ts`
- `()` : event binding, associe un événement à une méthode définie dans la classe `.ts`

Binding syntax

```
1 import { Component } from '@angular/core';
2
3 const CNT_INIT_VAL: number = 0;
4
5 @Component({
6   selector: 'app-counter',
7   templateUrl: './counter.component.html',
8   styleUrls: ['./counter.component.css']
9 })
10 export class CounterComponent {
11   title: string = "Counter Component";
12   count: number = CNT_INIT_VAL;
13
14   increment() {
15     this.count++;
16     if (this.count == 20) this.count = CNT_INIT_VAL;
17   }
18 }
```

counter.component.ts

```
1 <h2>{{ title }}</h2>
2 <button (click)="increment()">Increment</button>
3 <span [innerText]="count"></span>
4 <div *ngIf="count > 10 && count < 16">
5   ... ça fait beaucoup de clics !
6 </div>
```

counter.component.html

```
1 span {
2   margin: 2px;
3   font-weight: bold;
4 }
```

counter.component.css

Rendu navigateur

Counter Component

Increment 0

Classe typescript

```
1 import { Component } from '@angular/core';
2
3 const CNT_INIT_VAL: number = 0;
4
5 @Component({
6   selector: 'app-counter',
7   templateUrl: './counter.component.html',
8   styleUrls: ['./counter.component.css']
9 })
10 export class CounterComponent {
11   title: string = "Counter Component";
12   count: number = CNT_INIT_VAL;
13
14   increment() {
15     this.count++;
16     if (this.count == 20) this.count = CNT_INIT_VAL;
17   }
18 }
```

Import du décorateur

Constante

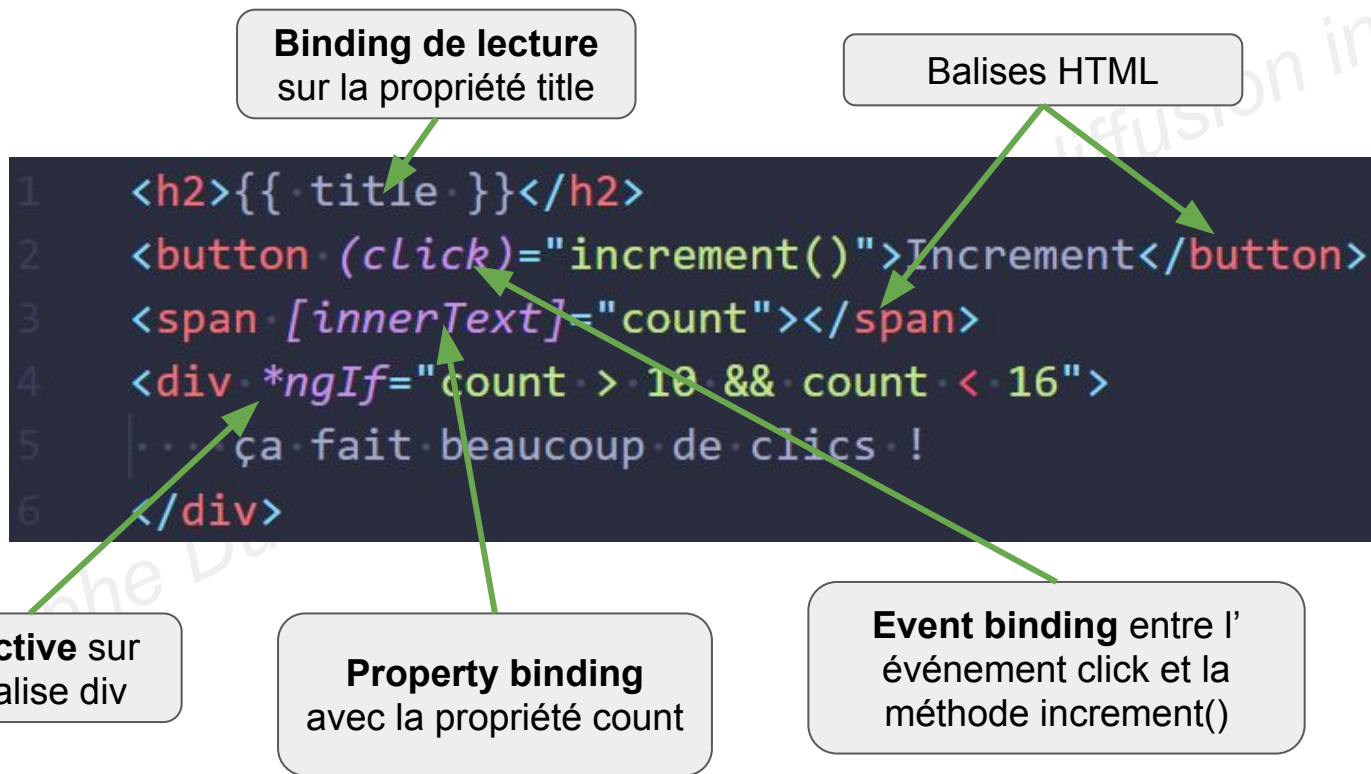
Décorateur de classe

Classe

Propriétés de la classe

Méthode de la classe

Template



Décorateur

- Fonction spécifique fournie par le framework (donc importable)
- Syntaxe: **@NomDuDecorateur()**, exemple: **@Component()**
- Finalité
 - Modifier le comportement d'une classe ou d'une propriété
 - Fournir des métadonnées à l'élément décoré afin de le paramétrer

Directive

- Attribut de balise visant à modifier/paramétrer l'apparence et ou le fonctionnement (comportement) de la balise ciblée
- “Built-in directives”: *ngIf, *ngFor, ngStyle, ngClass
- S'utilise uniquement dans le template html
- Le développeur peut créer ses propres directives

Directive *ngIf

```
<balise *ngIf="booleanExprOrVal">  
  <!-- éléments enfants --!>  
</balise>
```

Si **booleanExprOrVal** vaut **true**, la balise, qui peut être une balise HTML standard (p, div, h2, etc.) ou bien un sélecteur de composant (app-login, app-player-list, etc.) sera insérée dans le DOM

Directive *ngFor

```
<balise *ngFor="let item of items">  
  <!-- éléments enfants --!>  
</balise>
```

La balise, qui peut être une balise HTML standard (p, div, h2, etc.) ou bien un sélecteur de composant (app-login, app-player-list, etc.) sera reproduite autant de fois qu'il y a d'éléments dans le tableau **items**.
item est une variable locale qui, à chaque itération prend pour valeur, la valeur de l'élément itéré.

Directive *ngFor avec index

```
<balise *ngFor="let item of items; let i=index">  
  <!-- éléments enfants --!>  
</balise>
```

L'ajout de l'expression **let i=index** permet de récupérer dans une variable locale **i**, la valeur de l'index d'itération. La variable vaut **0** à la première itération, puis **1**, et ainsi de suite

Directive ngStyle

```
<balise [ngStyle]="{  
  'cssProp1': boundProp1,  
  'cssProp2': boundProp2 }">  
  <!-- éléments enfants --!>  
</balise>
```

ngStyle prend pour valeur un objet ayant pour clés les propriétés CSS qu'on souhaite relier dynamiquement (bind) avec des propriétés de la classe (**boundProp1** et **boundProp2**)

Directive ngStyle variante

```
<balise [ngStyle]="prop">  
  <!-- éléments enfants --!>  
</balise>
```

```
prop: {} = {  
  "color": "green",  
  "text-decoration": "underline"  
}
```

ngStyle est reliée dynamiquement à la propriété de classe **prop**. Cette propriété est un objet contenant des définitions CSS

Directive ngClass

```
<balise [ngClass]="{  
  'className 1': booleanVal1,  
  'className 2': booleanVal2}">  
  <!-- éléments enfants --!>  
</balise>
```

ngClass prend pour valeur un objet dont les clés sont des noms de classes présentes dans l'application/composant et dont les valeurs associées sont des booléens (par retour d'expression ou binding avec propriétés de la classe). En fonction de la valeur booléenne qui lui est associée, la classe sera ou ne sera pas attribuée à la balise

Module

- Un module vise à réunir différents éléments (**composants**, directives, interfaces, etc.) de manière logique
- Une “grosse” application sera décomposée en plusieurs modules
- Il existe des modules natifs (ex: HttpClientModule, FormsModule)
- Le développeur peut créer et organiser ses propres modules
- Avantages:
 - Organisation
 - Lisibilité
 - Réutilisabilité
 - Maintenabilité
- `ng generate (g) module moduleName`

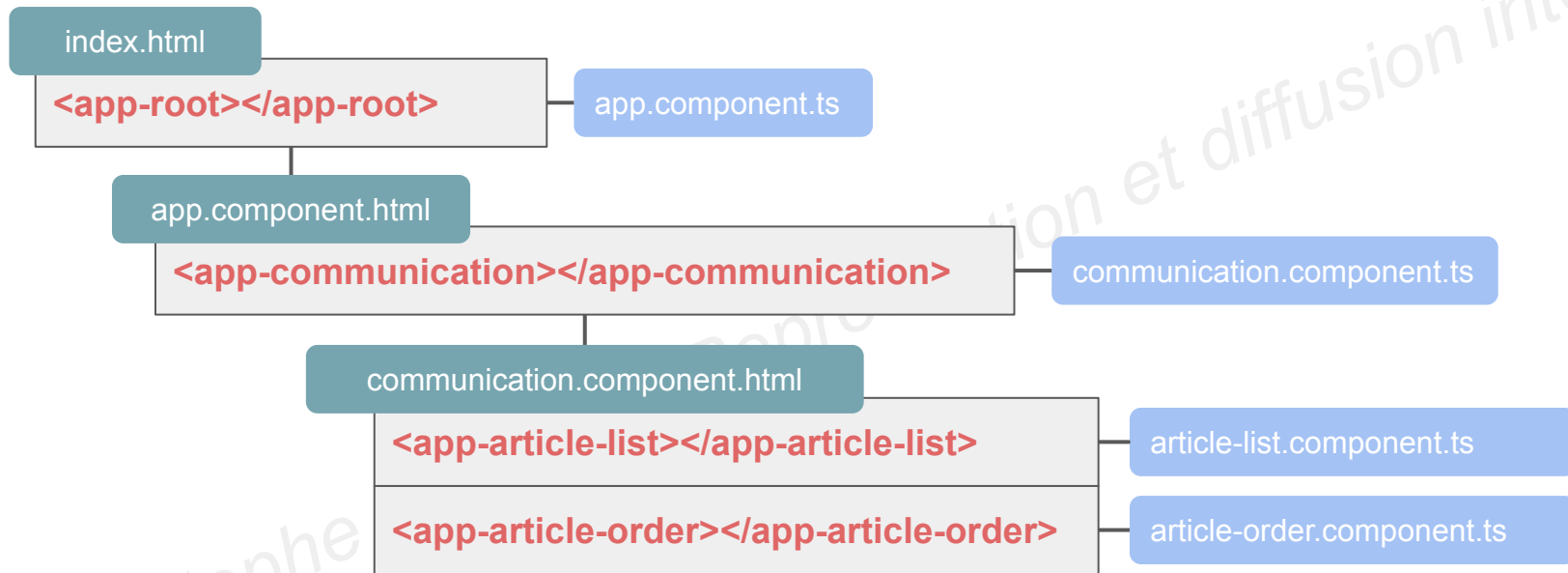
Module

Le module *communication.module.ts* réunit 3 composants: Article List Component, Article Order Component, CommunicationComponent. Il contient également un fichier *interfaces.ts* définissant des interfaces utiles au module



CommunicationComponent, enregistré dans le tableau “exports”, sert de “point d’entrée” au module. Le sélecteur (balise) associé (app-communication) pourra être inséré dans le template d’un composant d’un autre module (ayant importé communication.module). L’arbre des composants (composants enfants insérés dans son template) sera chargé également.

Arbre des composants (Component Tree)



`app.component` est **parent** de `communication.component`

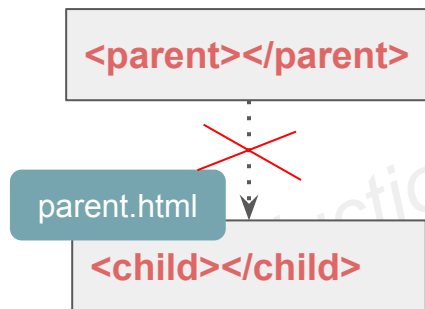
`communication.component` est **parent** de `article-list.component` et `article-order.component`

`communication.component` est **enfant** de `app.component`

`article-list.component` et `article-order.component` sont **frères** (siblings), ils ont le même parent direct.

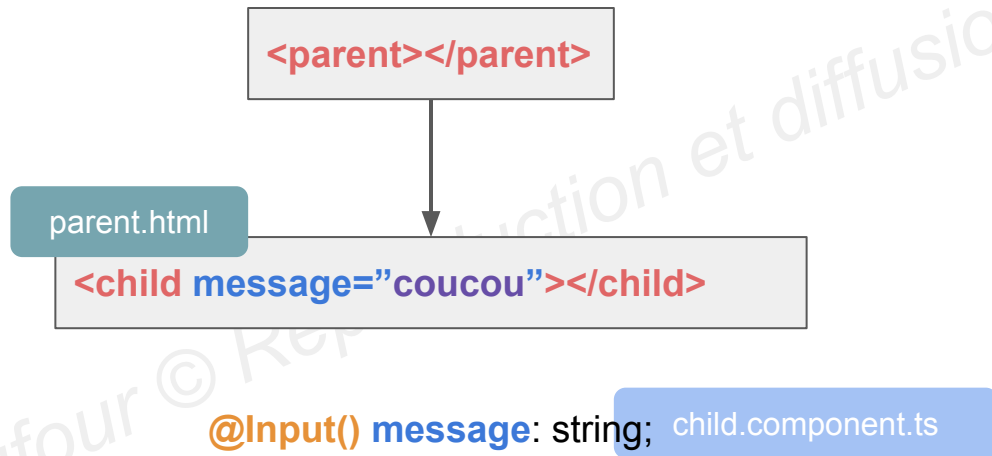
Présentation réalisée par christophe DUFOUR. Reproduction et diffusion interdites sans autorisation.

Communication entre composant



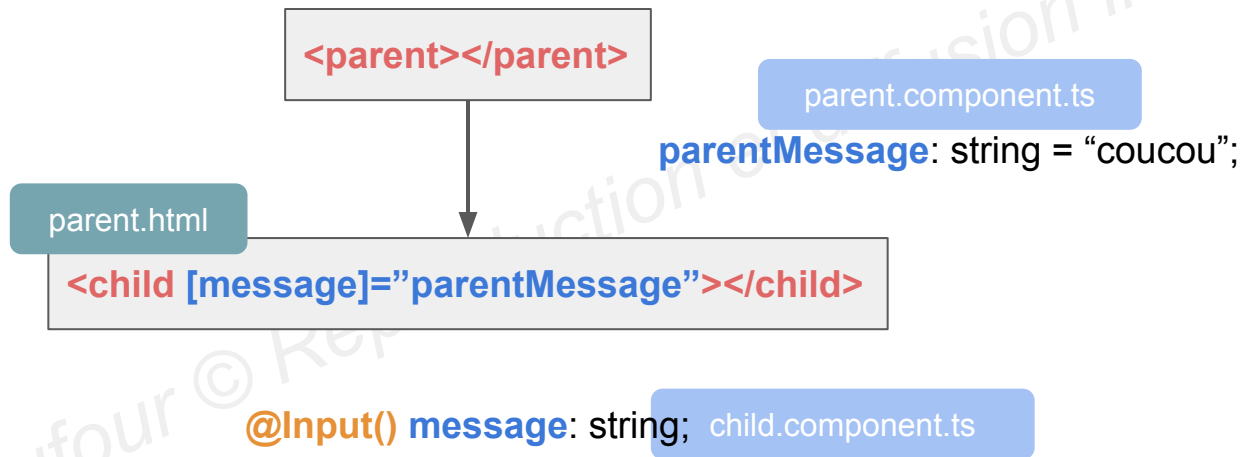
Aucune communication possible entre le composant **parent** et le composant **child**. Child est fermé sur lui-même et n'offre au contexte parent aucun moyen de lui fournir des données. Les éventuelles propriétés internes au composant enfant ne sont pas accessibles au parent.

Communication entre composant



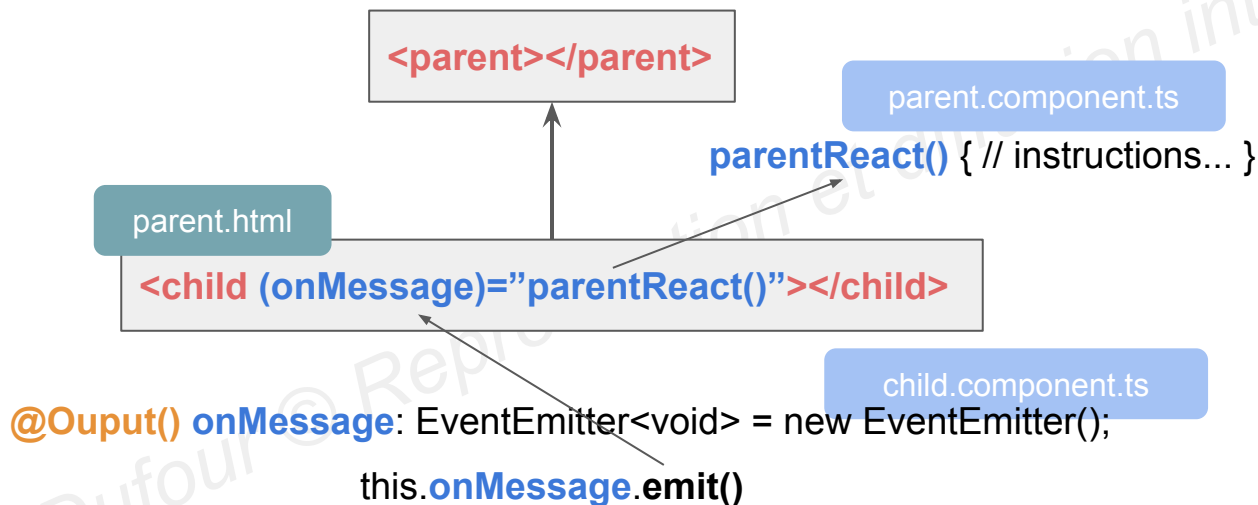
La propriété **message** du composant **child** est accessible en écriture par le composant parent, grâce au décorateur **@Input()**
L'attribut de balise **message** permet au parent de transmettre des données à son enfant, ici la chaîne de caractères statique **"coucou"**

Communication descendante sans binding



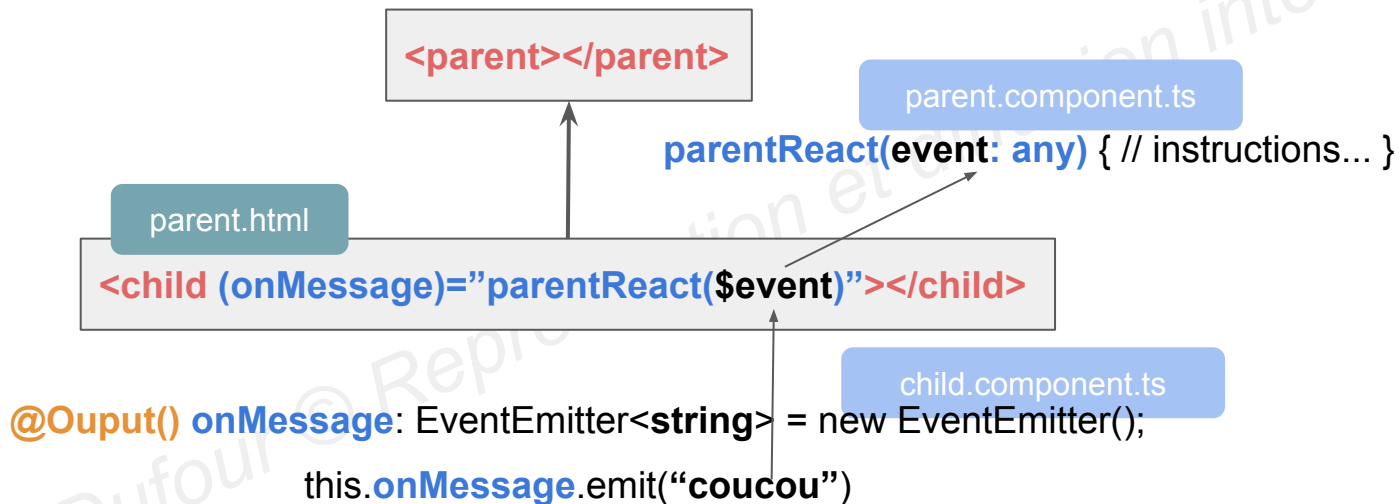
La propriété `message` du composant `child` est liée dynamiquement, grâce au **Property Binding**, avec la propriété `parentMessage` du composant `parent`.
Tout modification de la propriété `parentMessage` entraîne la mise à jour automatique de la propriété `message`.

Communication ascendante



La propriété `onMessage` du composant `child`, décorée `@Output()` et de type `EventEmitter` permet au composant enfant d'envoyer un signal au `parent`, grâce à la méthode `emit()` de l'objet de type `EventEmitter`. Le composant parent peut lier dynamiquement l'événement `onMessage` à l'une de ses méthodes (ici, `parentReact`), grâce à l'**Event Binding**. L'événement `onMessage` se déclenche dès que `this.onMessage.emit()` est appelée. Ici, aucune valeur n'est transmise lors du déclenchement de l'événement.

Communication ascendante



La propriété `onMessage` du composant `child` a été déclarée comme `EventEmitter` sur un type `string`. Le déclenchement de l'événement est accompagné d'une transmission d'une valeur ("`coucou`" ici) que la parent peut récupérer dans le paramètre `$event` de la callback (méthode `parentReact`) associée à l'événement

Communication entre composants frères

```
<parent></parent>
```

parent.component.ts

this.parentMsg = 'coucou'

```
<childOne  
  (onMessage)="parentMsg = 'coucou'"  
></childOne>
```

childOne.component.ts

this.onMessage.emit()

```
<childTwo  
  [message]="parentMsg"  
></childTwo>
```

childTwo.component.ts

this.message = 'coucou'

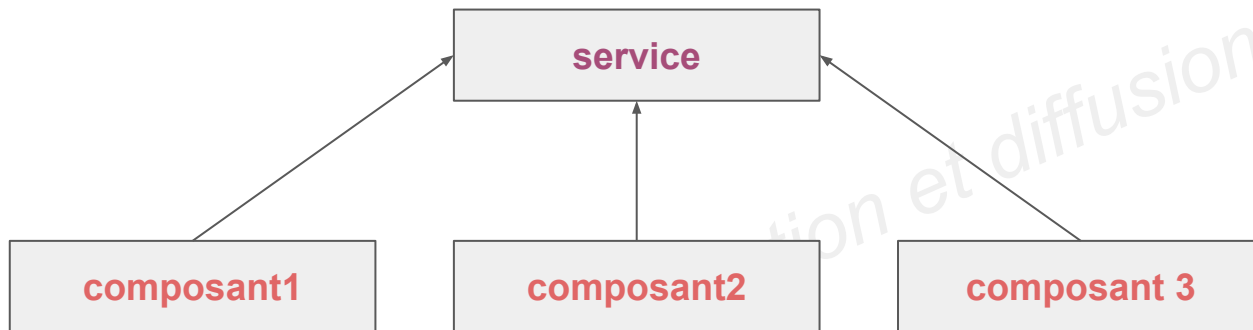
childOne et **childTwo**, frères, ne peuvent pas communiquer directement, leur parent commun, leur sert d'interface de communication.

L'événement **onMessage** émis par **childOne** modifie la propriété **parentMsg** du composant **parent**, ce qui met également à jour la propriété **message** de **childTwo** dynamiquement liée avec la propriété **parentMsg**

Service

- Classe décorée **@Injectable**
- Mutualise, centralise des traitements et des données à destination de composants et/ou d'autres services
- S'utilise par injection de dépendance (instanciation dans le constructeur de la classe en ayant besoin)
- Evite la redondance de code
- Sert souvent d'intermédiaire avec la couche d'accès aux données
- N'a pas vocation à interagir avec l'UI (rôle dévolu aux composants)
- `ng generate service serviceName`

Service: schéma



3 composants utilisant le même service

Service: déclaration

fruit.service.ts

```
1 import { Injectable } from '@angular/core';
2
3 @Injectable({
4   providedIn: 'root'
5 })
6 export class FruitService {
7   private fruits: string[] = ["cerise", "pomme", "kiwi"];
8
9   constructor() {}
10
11   getFruits() {
12     return this.fruits;
13   }
14 }
```

Propriété privée, non directement accessible depuis une autre classe

Méthode publique (par défaut), getter retournant le tableau de fruits.

Méthode invocable par les objets instanciant la classe FruitService

Service: utilisation

app.component.ts

```
1 import { Component } from '@angular/core';
2 import { FruitService } from '../services/fruit.service';
3
4 @Component({
5   selector: 'root',
6   templateUrl: './app.component.html',
7   styleUrls: ['./app.component.css']
8 })
9 export class AppComponent {
10   title = 'intro';
11   fruits: string[];
12
13   constructor(private fruitService: FruitService) {
14     this.fruits = this.fruitService.getFruits();
15   }
16 }
```

Import du service

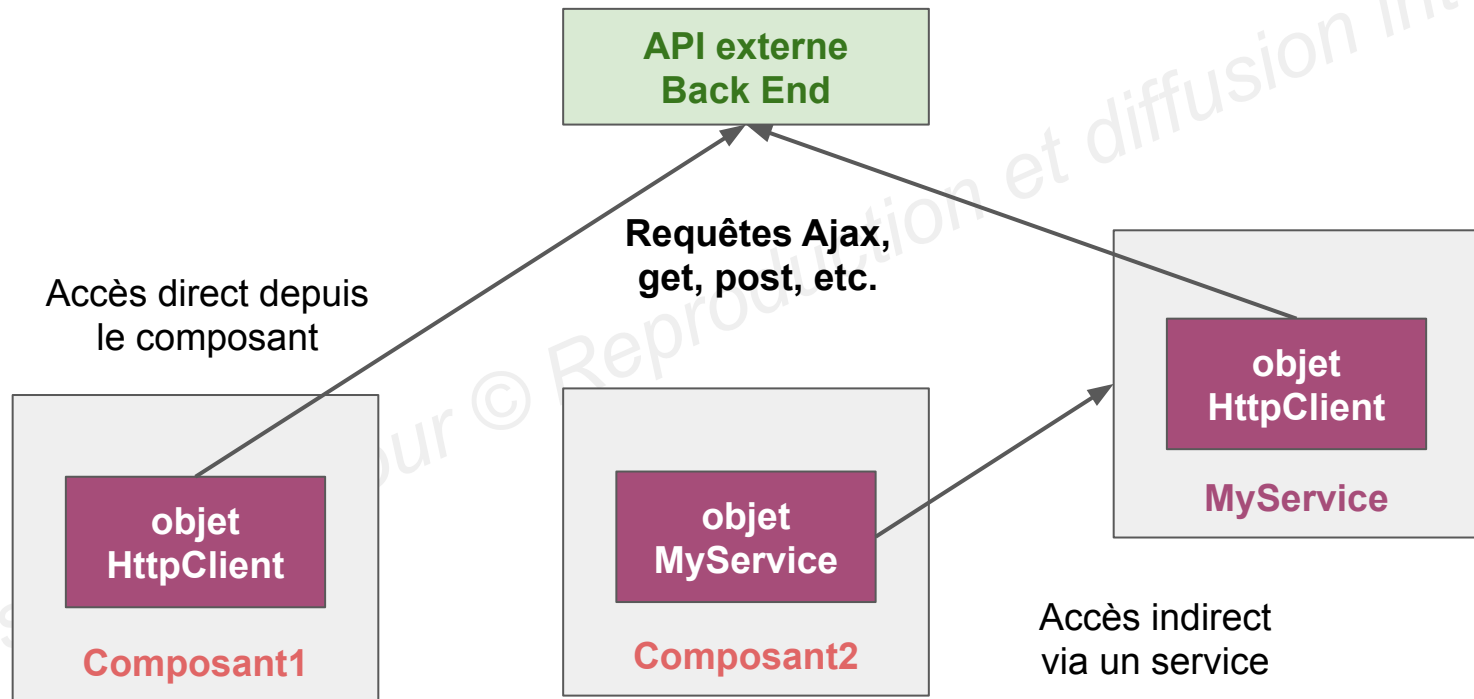
Injection de dépendance.
Une propriété *fruitService* est créée à la construction du composant par instantiation de la classe *FruitService*

Invocation de la méthode publique **getFruits()** exposée par l'objet *this.fruitService*

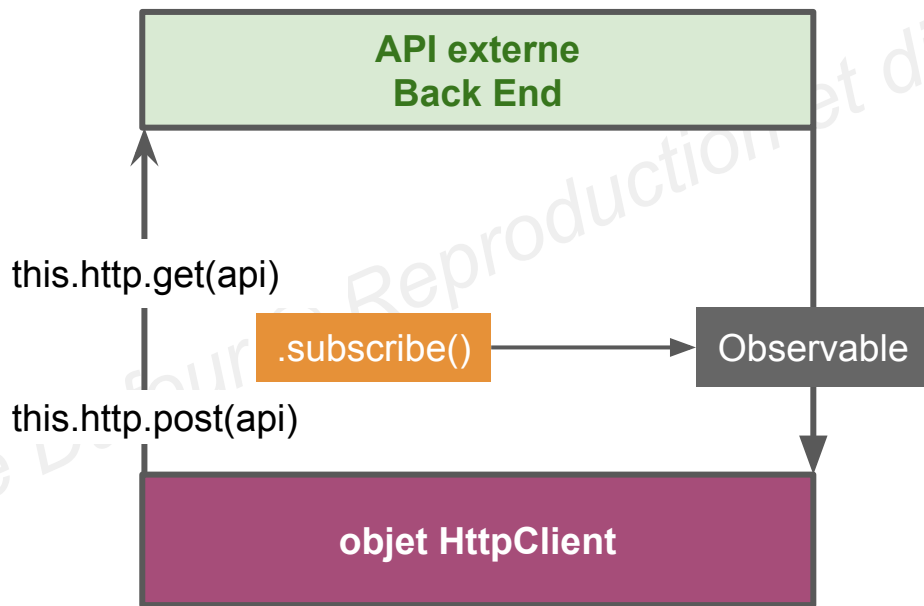
Requêtes Ajax

- Se font via le module **HttpClientModule** (@angular/common/http)
- Ce module expose le service **HttpClient** à instancier dans les composants et ou services ayant besoin d'effectuer des requêtes ajax
- Un objet de type HttpClient permet de faire des requêtes grâce aux méthodes suivantes: get, post, patch, etc. Exemple: this.httpClient.get(url);
- L'exécution de ces méthodes renvoie un objet de type **observable**
- Les requêtes étant par nature **asynchrone**, c'est dans le corps de la méthode de **souscription** - `.subscribe()` - à l'observable retourné qu'on peut accéder à la réponse du serveur

Requêtes Ajax: schéma



Requêtes Ajax: schéma



Requêtes Ajax: exemple

ajax.component.ts

```
1 import { HttpClient } from '@angular/common/http';
2 import { Component, OnInit } from '@angular/core';
3 import { Todo } from '../interfaces';
4
5 const API: string = "https://jsonplaceholder.typicode.com/todos";
6
7 @Component({
8   selector: 'app-ajax',
9   templateUrl: './ajax.component.html',
10  styleUrls: ['./ajax.component.css']
11 })
12 export class AjaxComponent implements OnInit {
13   public todos: Todo[] = [];
14
15   constructor(private http: HttpClient) {}
16
17   ngOnInit(): void {
18     this.http
19       .get(API)
20       .subscribe((todo: Todo) => {
21         this.todos.push(todo);
22       })
23   }
24 }
```

Import du service HttpClient

URL de l'API externe

Injection du service HttpClient

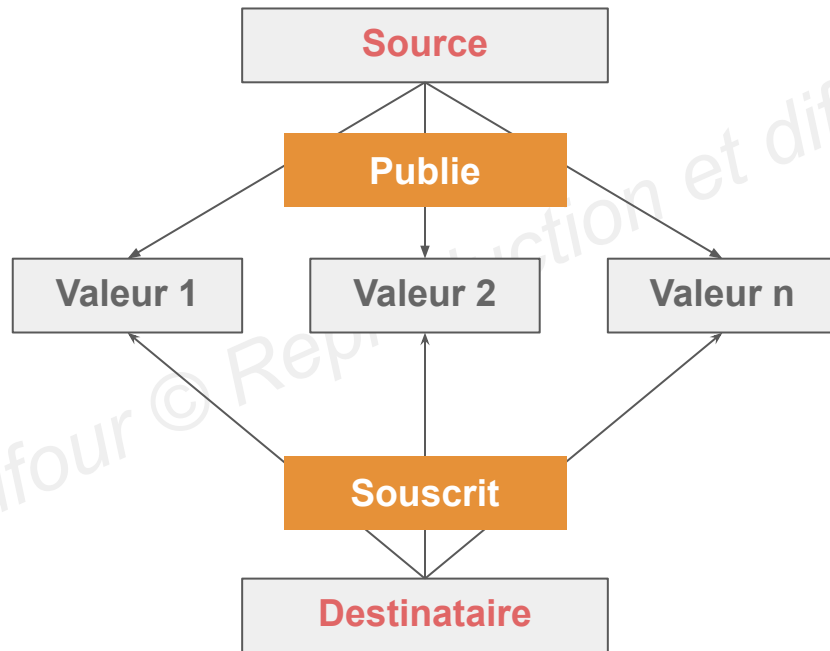
Requête en GET
renvoie un Observable

Suscription à l'Observable
retourné et accès à la
réponse du serveur (l'objet
todo) dans la callback

Rxjs et Observables

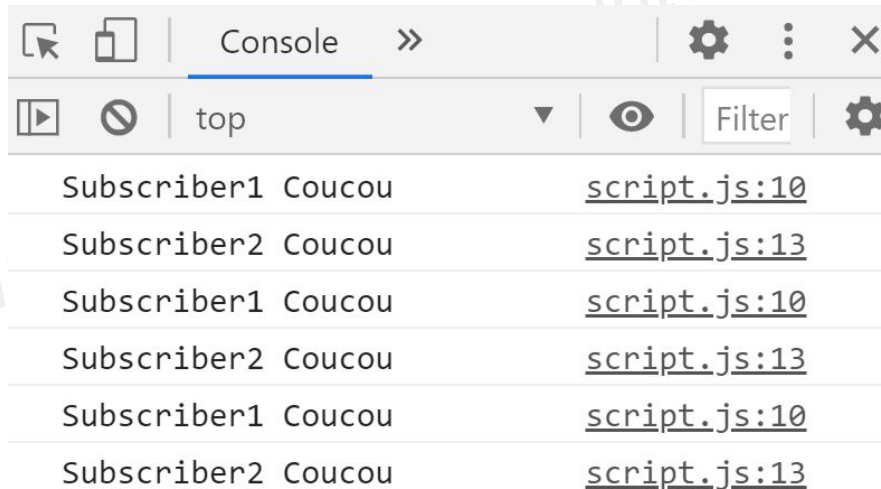
- Angular utilise rxjs à différents endroits (requêtes http, routing; etc.)
- Bibliothèque indépendante, implémentation JS des **Reactive eXtensions**
- Site officiel: <http://reactivex.io/>
- Approche basée sur le modèle de conception (design pattern) **PUB/SUB**
- Principe: une entité (source) publie/émet des valeurs, une ou plusieurs entités (destinataires) souscrivent, s'abonnent aux valeurs émises
- Diverses opérations - transformation, filtrage, etc., peuvent être réalisées et chaînées entre la source et ses souscripteurs
- Principes de la programmation fonctionnelle
- Bibliothèque complète offrant de nombreux outils (fonctions de création, opérateurs, etc.)

Le modèle PUB/SUB



Exemple hors Angular

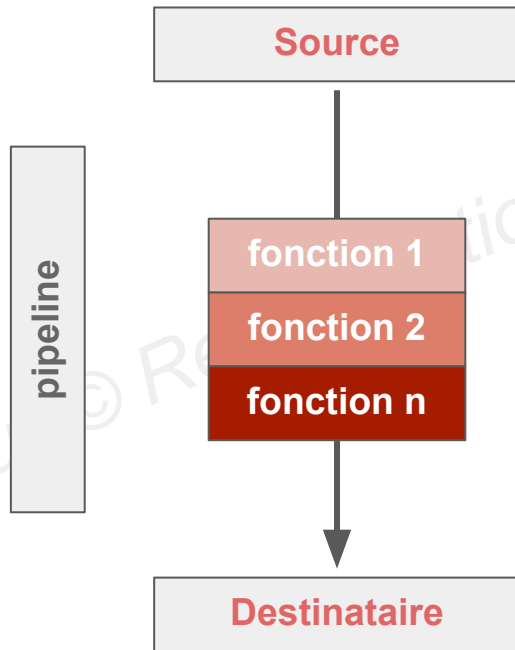
```
2 | const { Observable } = rxjs;
3 |
4 | const source$ = Observable.create((observer) => {
5 |   ...setInterval(() => observer.next("Coucou"), 2000);
6 | });
7 |
8 | const subscriber1 = source$.subscribe(res => {
9 |   ...console.log("Subscriber1", res);
10 | });
11 | const subscriber2 = source$.subscribe(res => {
12 |   ...console.log("Subscriber2", res);
13 | });
```



| | |
|--------------------|------------------------------|
| Subscriber1 Coucou | script.js:10 |
| Subscriber2 Coucou | script.js:13 |
| Subscriber1 Coucou | script.js:10 |
| Subscriber2 Coucou | script.js:13 |
| Subscriber1 Coucou | script.js:10 |
| Subscriber2 Coucou | script.js:13 |

La méthode **.create()** de l'objet Observable renvoie une source émettant (par la méthode **.next()**) la valeur "coucou" toutes les 2 secondes.
Deux destinataires souscrivent à **source\$** grâce à la méthode **.subscribe()**.
Les valeurs émises par la source sont reçues en entrée de la callback de chacune des souscriptions.

Pipeline d'un stream observable

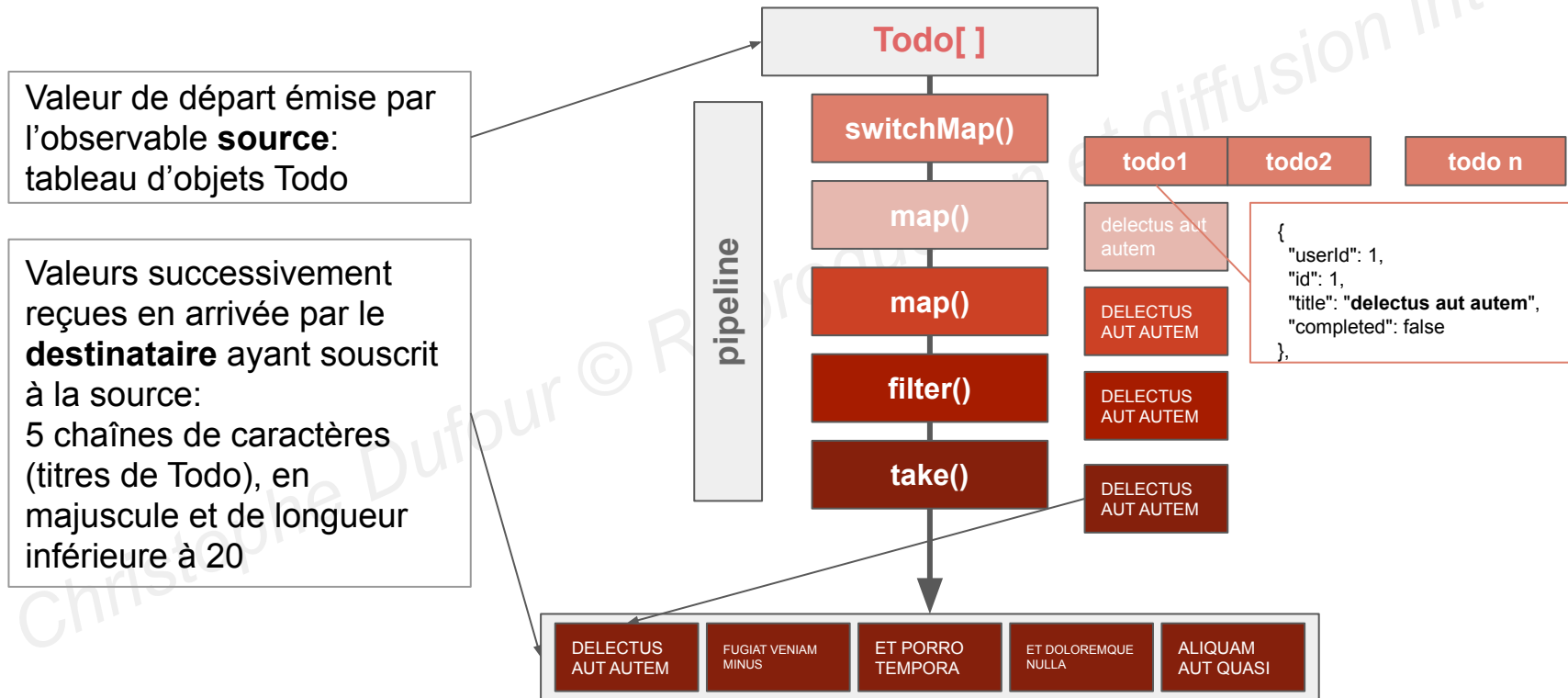




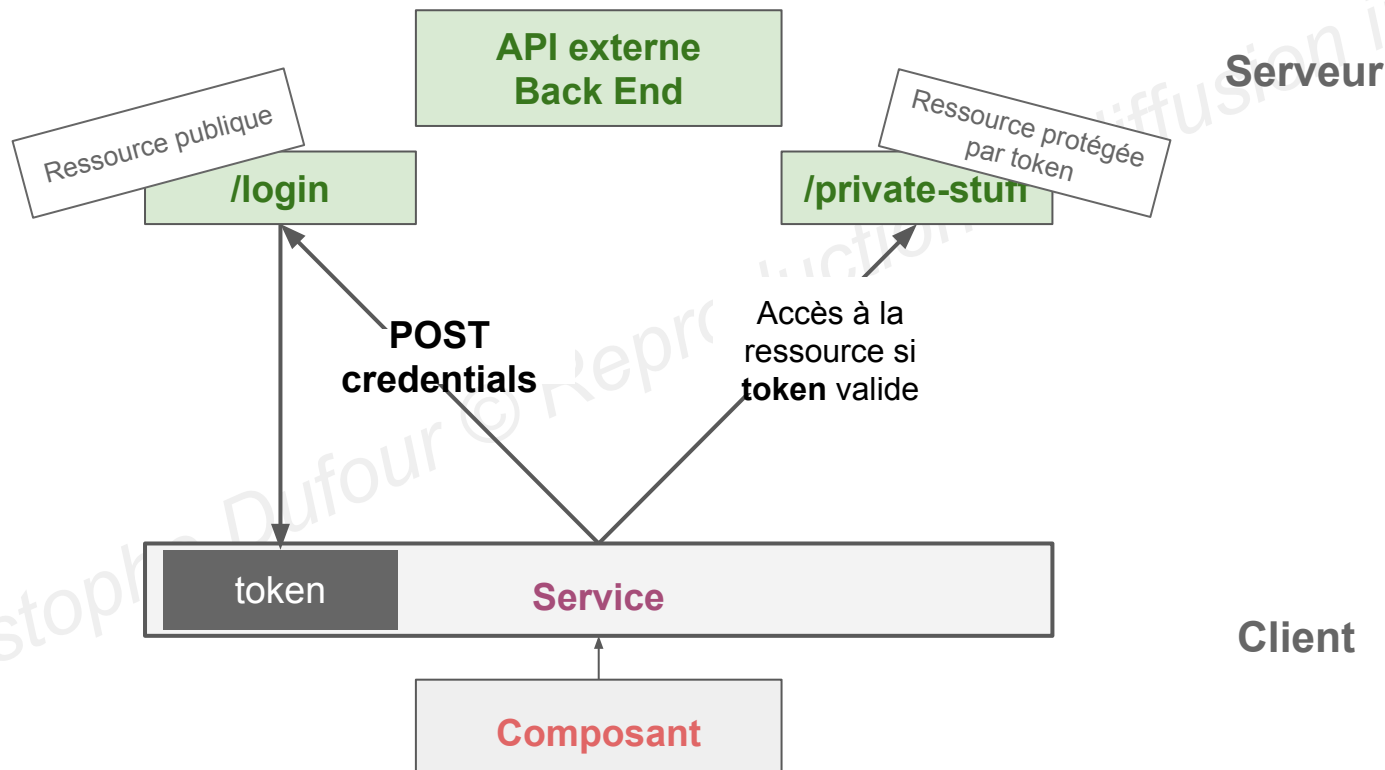
```
<p *ngFor="let title of titles">{{ title }}</p>
```

ALIQUM AUT QUASI

Pipeline d'un stream observable



Login: schéma



Formulaires

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

Reactive forms (model driven)

- accès direct et explicite au modèle sous-jacent
- comparativement plus robuste, réutilisable et testable

Template-driven forms

- repose sur des directives du template
- création implicite d'objets sous-jacents
- plus basique, utile pour l'ajout d'un formulaire simple dans une application

Formulaires

Setup of form model

Explicit, created in component class

Implicit, created by directives

Data model

Structured and immutable

Unstructured and mutable

Data flow

Synchronous

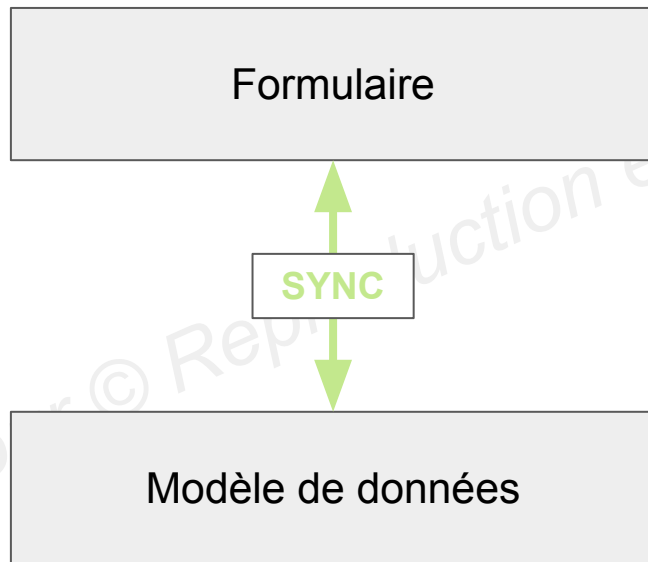
Asynchronous

Form validation

Functions

Directives

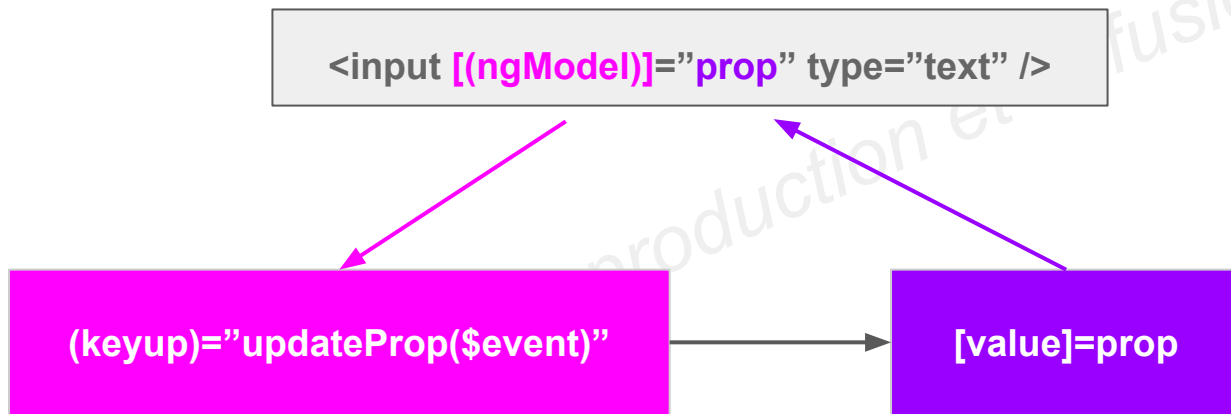
Formulaires



La directive ngModel (formulaire)

- Requiert l'import du module **FormsModule** (@angular/forms)
- Syntaxe: [(ngModel)]="prop"
- Offre un rapide mécanisme de "Two-way data binding"
- Permet de lier dynamiquement un champ de formulaire de manière bidirectionnelle: event binding (click, change, keyup, etc.) + property binding sur la valeur du champ mis à jour
- Exemple: ngModel appliquée à un champ input, écouter par défaut l'événement (Keyup) et met à jour la valeur du champ dès que l'événement se produit

ngModel: schema



ngModel: exemple

```
<input  
  [(ngModel)]="credentials.email"  
  type="text" placeholder="Email" />  
<br>  
<input  
  [(ngModel)]="credentials.password"  
  type="password" placeholder="Password">  
<br>  
<select [(ngModel)]="status">  
  <option value="admin">Administrateur</option>  
  <option value="anonymous">Anonyme</option>  
</select>
```

Placé sur un input de type text, **ngModel** met à jour la propriété *email* de l'objet *credentials* à l'événement **keyup**

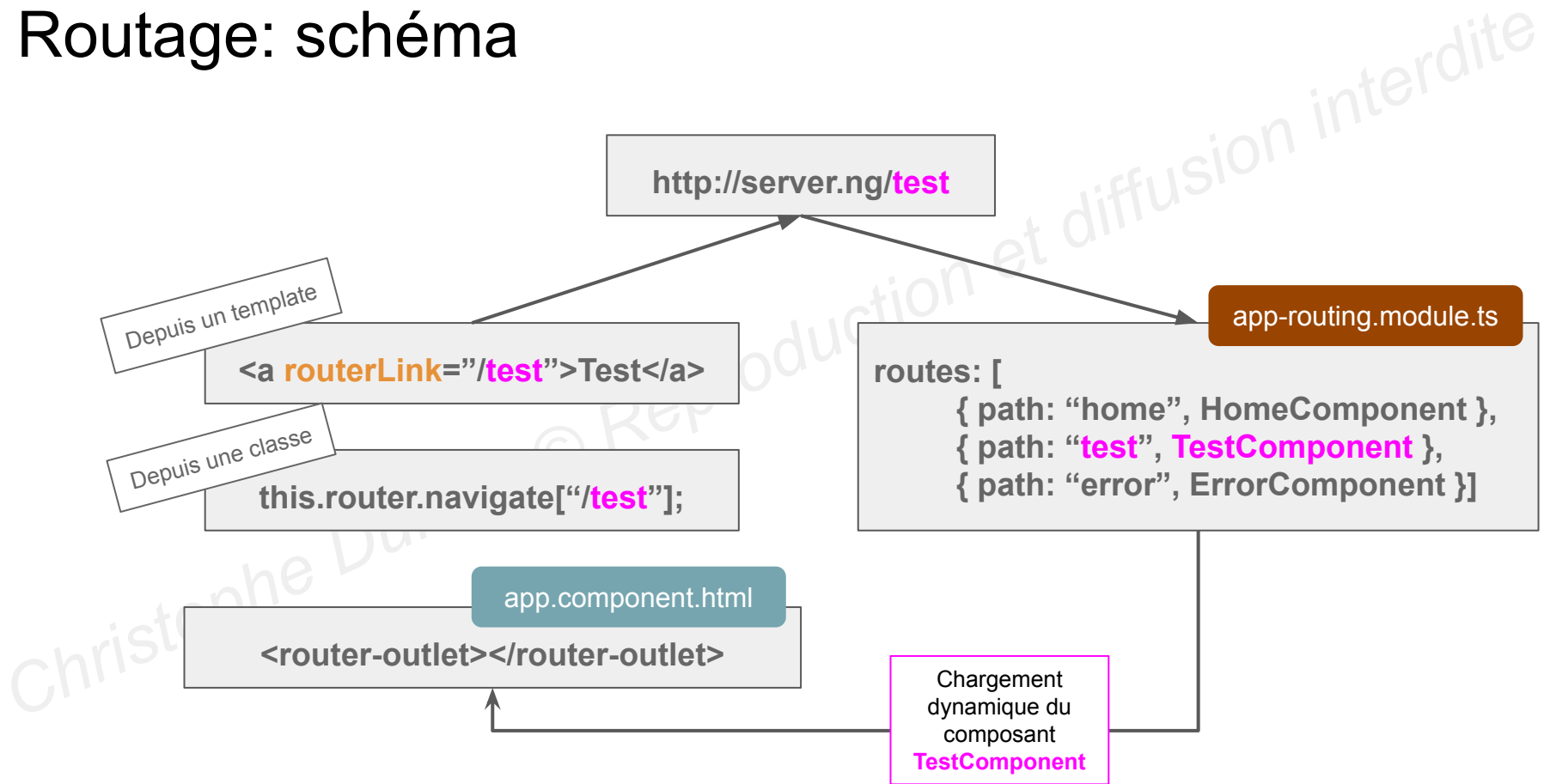
```
credentials: Credentials = {  
  email: "toto@gmail.com",  
  password: "bonjour"  
};  
status: string = "anonymous";
```

Placé sur un champ select, **ngModel** met à jour la propriété *status* à l'événement **change**

Routage

- Les fonctionnalités de routage Angular nécessitent l'import du module **RouterModule** (`@angular/router`)
- La méthode **.forRoot()** de ce module de lui fournir le table de routage à prendre en compte
- Les routes sont un tableau d'objets disposant a minima des clés **path** et **component** destinées à faire le lien entre un segment d'URL et le composant à charger en conséquence
- Le composant **<router-outlet></router-outlet>** sert de “coquille” aux composants chargés dynamiquement par routage

Routage: schéma



Routage: exemple

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { FirstComponent } from './first/first.component';
import { SecondComponent } from './second/second.component';
import { UserComponent } from './user/user.component';

const routes: Routes = [
  { path: "**", component: FirstComponent},
  { path: "first", component: FirstComponent},
  { path: "second", component: SecondComponent},
  { path: "user/:id", component: UserComponent}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})

export class AppRoutingModule { }
```

Composant chargé dans
<router-outlet></router-outlet>
lorsqu'une correspondance ("match") est
détectée entre l'url requise par la directive
routerLink et la table de routage

```
<nav>
  <ul>
    <li><a routerLink="/first">First Component</a></li>
    <li><a routerLink="/second">Second Component</a></li>
    <li><a routerLink="/user/1">User 1</a></li>
    <li><a routerLink="/user/2">User 2</a></li>
  </ul>
</nav>
```

Route avec paramètre

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

@Component({
  selector: 'app-user',
  template: `<h2>UserId: {{ userId }}</h2>`
})
export class UserComponent implements OnInit {
  userId: string = "";
  constructor(private route: ActivatedRoute) { }
  ngOnInit(): void {
    this.route.paramMap.subscribe((params) => {
      this.userId = params.get("id");
    })
  }
}
```

<app-user>

<h2>UserId: 5</h2>

</app-user>

Dans le
navigateur

http://server.ng/user/5

app-routing.module.ts

```
routes: [
  { path: "home", HomeComponent },
  { path: "user/:id", UserComponent },
  { path: "error", ErrorComponent }]
```

L'objet *router* de type **ActivatedRoute** permet d'accéder à la propriété Observable **paramMap**. La souscription fournit un objet exposant lui-même une méthode **get(paramName)** qui nous retourne la valeur associée au paramètre url spécifié, ici **"id"**.