



Angular

Utopios

Index

1. [Introduction](#)
2. [TypeScript](#)
3. [Sass](#)
4. [CSS 3 \(some tips\)](#)
5. [Architecture Générale](#) ✓
6. [Composants](#) ✓
7. [Tests](#)
8. [Pipes](#) ✓
9. [Formulaires](#)
10. [Services](#) ✓
11. [Functional Reactive Programming](#)
12. [@ngrx/store](#)
13. [Service Http](#) ✓
14. [Routeur](#) ✓
15. [Concepts avancés](#)
16. [Zone & Change Detection](#) ✗
17. [Angular & Performances](#)
18. [Angular Material](#) ✓
19. [PWA](#)
20. [Angular Universal](#) ✗
21. [Lazy Loading](#) ✓

Introduction

Angular - Présentation

- Framework créé par **Google** et annoncé en 2014
- Réécriture total du framework **AngularJS**
- Reprend certains concepts d'**AngularJS**
- Version beta annoncée le 23/10/2014
- Version officielle sortie en **2016**
- Programmation **orientée composant**
- Framework conçu pour être plus performant et optimisé pour les mobiles
- Application SPA
- Frameworks alternatifs : **VueJS** / **React** / ...
- <http://angular.io/>

NodeJS & NPM



- **Node** inclut un système de gestion des paquets : **npm**
- **npm** est un outil en ligne de commande (écrit avec Node.js)
- Il permet de télécharger les modules disponibles sur **npmjs.org**
- Il existe pratiquement depuis la création de Node.js
- C'est un canal important pour la diffusion des modules

NPM : commandes

- npm **init** : initialise un fichier package.json
- npm **install** : télécharge les modules et les placent dans `./node_modules`
- npm **install [package] --save** : ajoute une dépendance au projet
- npm **install [package] --save-dev** : ajoute une dépendance de dev au projet
- npm **install [package] --global** : ajoute une dépendance au user système
- npm **upgrade [package]** : met à jour les dépendances en suivant le package.json
- npm **uninstall [package]** : supprime une dépendance du projet
- npm **outdated** : liste les packages dont il existe une version plus récente

package.json

- **npm** se basent sur un fichier descripteur du projet
- **package.json** qui décrit précisément le module
- On y trouve différents types d'informations
 - Identification :
 - **name** : l'identifiant du module (unique, url safe)
 - **version** : doit respecter **node-semver**
 - Description : **description**, **authors**, ...
 - Dépendances : **dependencies**, **devDependencies**, ...
 - Cycle de vie : **scripts**, **main**, ...

package.json : dépendances

dependencies

La liste des dépendances nécessaires à l'exécution

devDependencies

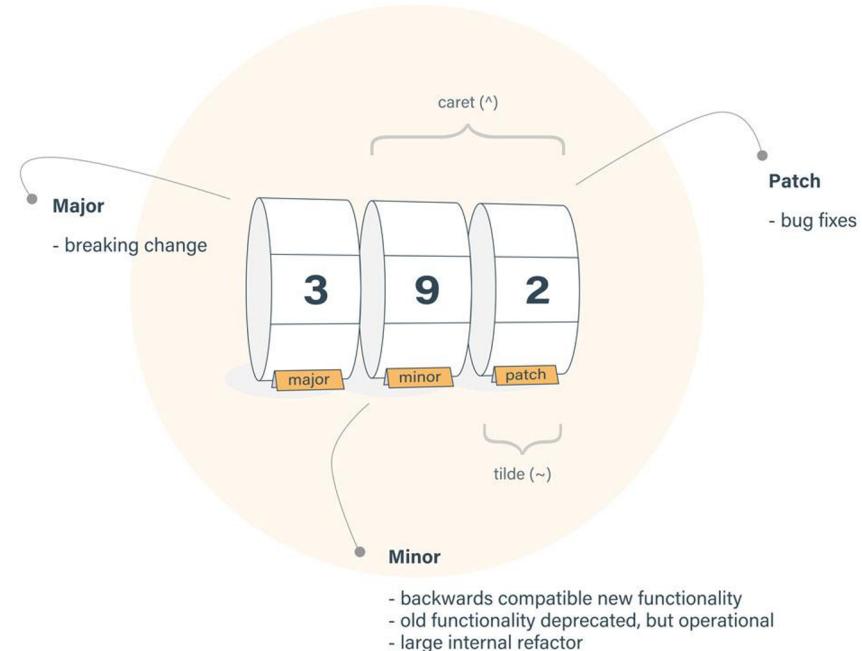
Les dépendances pour les développements (build, test...)

package.json : versions

- Les modules doivent suivre la norme **semver**
 - Structure : **MAJOR.MINOR.PATCH**
 - **MAJOR** : Changements non rétrocompatibles
 - **MINOR** : Changements rétrocompatibles
 - **PATCH** : Corrections d'anomalies rétrocompatibles
- Pour spécifier la version d'une dépendance
 - **version** : doit être exactement cette version
 - **~, ^** : approximativement, compatible
 - **major.minor.x** : **x** fait office de joker
 - Et bien d'autres : **>**, **<**, **>=**, ...
- Doc complète : <https://github.com/npm/node-semver>

SEMVER : SEMantic VERsioning

Symbol	Dependency	Versions	Changes
caret (^)	<code>^3.9.2</code>	<code>3.*.*</code>	<ul style="list-style-type: none">- backwards compatible new functionality- old functionality deprecated, but operational- large internal refactor- bug fix
tilde (~)	<code>~3.9.2</code>	<code>3.9.*</code>	<ul style="list-style-type: none">- bug fix



IDE : IDEA IntelliJ / Webstorm / PhpStorm

Plugins

- Plugin **Karma**
 - permet d'avoir les flèches pour lancer les TU
 - Code coverage disponible
- Plugin **NodeJS**

Raccourcis

- [ALT] + [CMD] + [L] : Format
- [ALT] + [CMD] + [O] : Organize imports
- **[F2] : Go to next error**
- **[ALT] + [ENTER] : Fix it !**



IDE : Visual Studio Code

Plugins

- Plugin `angular essentials`
- Plugin `Angular Language Service`
- Plugin `auto import`
- Plugin `typescript hero`
- Plugin `tslint` (installé de base)
- Plugin `editorconfig for VS Code` (installé de base)
- Plugin `gitlens`
- Plugin `bracket pair colorizer 2`
- Plugin `prettier` (ne pas oublier de configurer les simples quotes dans la config)

Raccourcis

- [ALT] + [SHIFT] + [F] : Format (by aprettier)
- [ALT] + [SHIFT] + [O] : Organize imports
- [CTRL] + [MAJ] + [K] : Supprimer une ligne
- [CTRL] + [;] : Fix error (on cursor)
- [CTRL] + [D] : Duplicate line

Astuce Google

Pour rechercher une information sur **Angular** (et non pas sur AngularJS), ajoutez à vos recherches **-angularjs**.



TypeScript



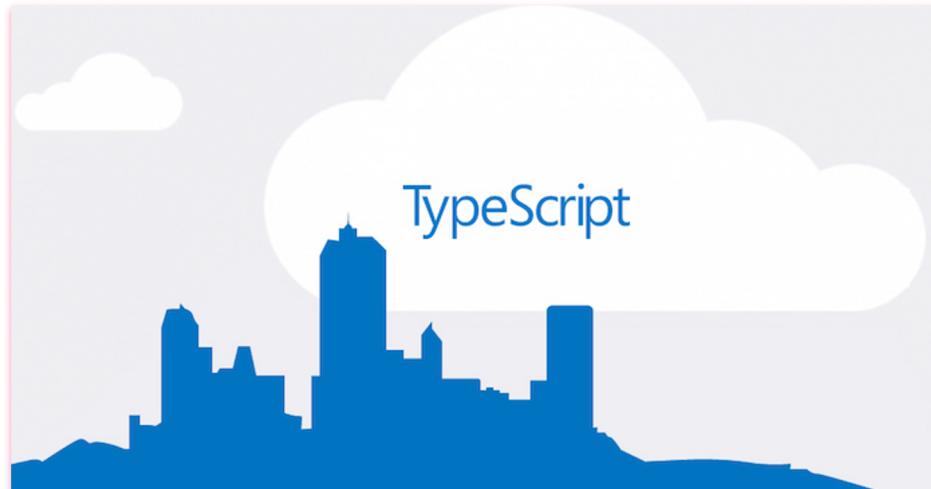
TypeScript

<https://www.typescriptlang.org>

Langage de programmation **libre** et **open source**

Développé (et maintenu) par **Microsoft**

1ere version sortie début 2012



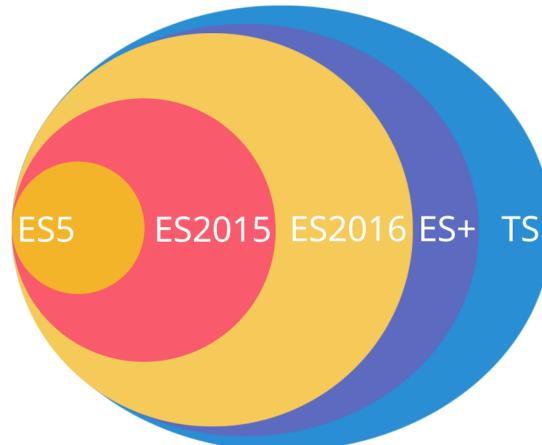
TypeScript

Superset typé de JavaScript qui est **transpilé en JavaScript**

Permet un **typage statique optionnel** des variables et des fonctions

Ajout de nouvelles fonctionnalités au langage JavaScript

Tout programme JavaScript est un programme TypeScript valide



Installation

1. Installation de **nodeJS** : <https://nodejs.org/fr/>
2. Installation de **TypeScript** :

```
FHE AFKL9DD Ö? LQHFK; JAHL
```

1. Créer un fichier **index.ts** contenant `let a = 'a';`
2. Transpilez avec **tsc** (`-w = --watch`)

```
LK; AF<=P} LK ÖÖ
```

TypeScript - Fonctionnalités

TypeScript ajoute aux fonctionnalités de l'**ES+** les éléments suivants :

Basic Types - **Variable Declarations** - **Interfaces** - **Classes** - **Functions** -
Generics - **Enums** - Type Inference - Type Compatibility - Advanced Types -
Symbols - Iterators and Generators - Modules - Namespaces - Namespaces and
Modules - Module Resolution - Declaration Merging - JSX - **Decorators** - Mixins
- Triple-Slash Directives

Types (basics)

Pour déclarer une variable :

```
var  variableName1: variableType = value;  
let  variableName2: variableType = value;  
const variableName3: variableType = value;
```

avec son type :

ßß : GGD=9F ~
D=L AK2JM=~ : GGD=9F ü >9DK=Ä

ßß FME: =J ~
D=L KAR=~ FME: =J ü T[SÄ

ßß KLJAF? ~
D=L F9E=~ KLJAF? ü à G à Ä

ßß 9JJ9Q í U FGL9LAGFK LJ9FKHADÍ =K <9FK D= E=^E= ; G= (9N91; JAHLí ~
D=L F9E=K~ KLJAF?í ü í à G à| à" QD9Fàí Ä
D=L F9E=K~ JJ9Q?KLJAF?£ ü í à G à| à" QD9Fàí Ä

ßß 9FQ ~
D=L FGL1MJ=~ 9FQ ü WUÄ

ßß KLJAF? GJ FME: =J
D=L 9?=~ KLJAF? " FME: =JÄ

Types (functions)

Comme en JavaScript, possibilité de créer des fonctions nommées ou anonymes

```
BB $GF; LAGF FGEEÍ =  
>MF; LAGF F9E=<$MF; LAGFí ì ~ NGA< ñ }}} ó  
  
BB $GF; LAGF 9FGFQE=  
D=L N9JA9: D= FGFQEGMK$MF; LAGF ü >MF; LAGFí ì ~ NGA< ñ }}} ó
```

Peut retourner une valeur grâce au mot clé return

Accès aux variables définies en dehors du scope de la fonction

```
let externalScope: number = 10;  
function add(localArg: number): number {  
    return localArg + externalScope;  
}
```

Types (functions parameters)

Une fonction peut prendre des paramètres :

```
function fn(name: string, forename: string) { }
```

Un **paramètre peut être optionel** (utilisation du caractère `?`)

Ordre de définition très important

Aucune implication dans le code JavaScript généré

Si non-défini, le paramètre aura la valeur **`undefined`**

```
function fn(name: string, forename?: string) { }
```

Arrays

Permet de manipuler un tableau d'objet

2 syntaxes pour créer des tableaux

```
// Syntaxe Littérale
let list: number[] = [1, 2, 3];

// Syntaxe utilisant le constructeur `Array`
let list: Array<number> = [1, 2, 3];
```

Ces 2 syntaxes aboutiront au même code JavaScript

Type Enum

Possibilité de définir un type pour expliciter un ensemble de données numériques

```
enum Music { Rock, Jazz, Blues };
let c: Music = Music.Jazz;
```

La valeur numérique commence par défaut à 0

Possibilité de surcharger les valeurs numériques :

```
enum Music { Rock = 2, Jazz = 4, Blues = 8 };
let c: Music = Music.Jazz;
```

Récupération de la chaîne de caractères associée à la valeur numérique

```
let style: string = Music[4]; //Jazz
```

Classes

- Système de classes et interfaces similaire à la programmation orientée objet
- Le code javascript généré utilisera le système de prototype
- Possibilité de définir un constructeur, des méthodes et des propriétés
- Propriétés/méthodes accessibles via l'objet **this**

```
class Animal {  
    constructor() {}  
}  
  
let animal = new Animal();
```

Classes - Propriétés

- Trois scopes disponibles :
 - **public** (par défaut)
 - **private**
 - **protected**
- Propriétés ajoutées sur l'objet en cours d'instanciation (**this**)
- Possibilité de définir des propriétés statiques (**static**)
- Tous les types sont supportés : types primitifs, fonctions, ...
- Propriété ajoutée au constructeur de l'objet
 - **constructor(public user: User)**

Classes - Propriétés

BB 4=JKAGF 2QH=1; JAHL T
; D9KK . D9Q=J ñ
HM DA; FA; CF9E=~ KLJAF?Ä
; GFKLJM LGJ í FA; CF9E=~ KLJAF?ì ñ
L@AK} FA; CF9E= ü FA; CF9E=Ä
ó
ó

BB 4=JKAGF 2QH=1; JAHL U í LJ9FKHADÍ =K <9FK D= E=^E= ; G<= (9N91; JAHLí
; D9KK . D9Q=J ñ
; GFKLJM LGJ í HM DA; FA; CF9E=~ KLJAF?ì ñ ó
ó

Classes - Accesseurs

- Possibilité de définir des accesseurs pour accéder à une propriété
- Utiliser les mots clés **get** et **set**
- Attention à l'espacement après les mots clé
- Nécessité de générer du code JavaScript compatible ES5

```
class Person {  
    private _secret: string;  
    get secret(): string {  
        return this._secret.toLowerCase();  
    }  
    set secret(value: string) {  
        this._secret = value;  
    }  
  
    let person = new Person();  
  
    person.secret = 'TEST';  
  
    console.log(person.secret); // => 'test'
```

Classes - Héritage

- Système d'héritage entre classes via le mot clé extends
- Si constructeur non défini, exécute celui de la classe parente
- Possibilité d'appeler l'implémentation de la classe parente via super
- Accès aux propriétés de la classe parente si public ou protected

```
class Person {  
    constructor() {}  
    speak(message: string) {  
        console.log(message);  
    }  
}  
  
class Child extends Person {  
    constructor() { super() }  
    cry() {}  
    speak(message: string) {  
        super.speak(message.toLowerCase());  
    }  
}
```

Interfaces

- Utilisées par le compilateur pour vérifier la cohérence des différents objets
- Aucun impact sur le JavaScript généré
- Système d'héritage entre interfaces
- Plusieurs cas d'utilisation possible
 - Vérification des paramètres d'une fonction
 - Vérification de la signature d'une fonction
 - Vérification de l'implémentation d'une classe

Interfaces - Implémentation d'une classe

- Cas d'utilisation le plus connu des interfaces
- Vérification de l'implémentation d'une classe
- Erreur de compilation tant que la classe ne respecte pas le contrat défini par l'interface

```
interface Musician {  
    play(): void;  
}  
  
class TrumpetPlay implements Musician {  
    play() {}  
}
```

Génériques

- Fonctionnalité permettant de créer des composants réutilisables
- Inspiration des génériques disponibles en **Java** ou **C#**
- Nécessité de définir un (ou plusieurs) paramètre(s) de type (**<T>**) sur la fonction/variable/classe/interface générique

```
function identity<T>(arg: T): T {
    return arg;
}

identity(5).toFixed(2); // Correct
identity('hello').toFixed(2); // Incorrect
identity(true);
```

Génériques

Possibilité de définir une classe générique

```
class Log<T> {
    log(value: T) {
        console.log(value);
    }
}

let numericLog = new Log<number>();
numericLog.log(5); // Correct
numericLog.log('hello'); // Incorrect
```

Modules (import / export)

Concept apparu en ES2015 nécessitant une compilation
(**babeljs** ou **webpack** par exemple)

```
export { nom1, nom2, ..., nomN };
export { variable1 as nom1, variable2 as nom2, ..., nomN };
export let nom1, nom2, ..., nomN; // fonctionne également avec var, function
export let nom1 = ..., nom2 = ..., ..., nomN; // également avec var, const

export default expression;
export default function (...) { ... } // également avec class, function*
export default function nom1(...) { ... } // également avec class, function*
export { nom1 as default, ... };

export * from ...;
export { nom1, nom2, ..., nomN } from ...;
export { import1 as nom1, import2 as nom2, ..., nomN } from ...;
```

```
import exportParDefaut from "mon-module";
import * as nom from "mon-module";
import { export } from "mon-module";
import { export as alias } from "mon-module";
import { export1 , export2 } from "mon-module";
import { export1 , export2 as alias2 , [...] } from "mon-module";
import exportParDefaut, { export [ , [...] ] } from "mon-module";
import exportParDefaut, * as name from "mon-module";
import "mon-module";
```

Décorateurs

- Les décorateurs TS sont des **annotations** qui s'appliquent sur des classes ou des attributs de classe.
- Il existe une proposition pour leur implémentation dans ECMAScript : <https://github.com/tc39/proposal-decorators>.
- Ce sont des fonctions capables d'étendre le comportement d'autres fonctions sans les modifier.
- Abondamment utilisé en Angular

@TYPES & *.d.ts!!!

Un fichier de définition (*.d.ts) :

- Décrit du code écrit en javascript (fonctions, type des paramètres, ...)
- Permet aux IDE de gérer l'auto-complétion sur des lib JS
- Sont stockés sur NPM dans le dossier “@types/<lib>” (depuis TS 2.0)

Exemple :

Pour récupérer le .d.ts de lodash il suffit d'inscrire dans la section ‘devDependencies’ votre packages.json : “@types/lodash.”

Sass

Sass

préprocesseur CSS

Un **métalangage**.

Un **outil** pour produire et maintenir des feuilles de styles CSS.



Pas de nouvelles possibilités.

Vous devez comprendre CSS.

Pourquoi un préprocesseur CSS ?

CSS reste **le roi de la mise en forme**, mais ...

- Structuration du code.
- Dépendances de mises en forme.
- Tâches répétitives.

Quid des préprocesseurs CSS ?

The logo for Sass, featuring the word "Sass" in a pink, flowing, cursive script font.The logo for Less, consisting of the word "less" in white, bold, sans-serif font, enclosed within a dark blue rectangular box with white borders. The opening brace is positioned above the letter "l" and the closing brace is positioned below the letter "s".The logo for Stylus, featuring the word "stylus" in a large, black, elegant, cursive script font. A horizontal line underlines the "u" and "s" in "stylus".

Pourquoi Sass



ionic



2 syntaxes

Sass supporte **2 syntaxes** :

- ***.sass** : syntaxe basée sur les indentations.
- ***.scss** : syntaxe basée sur la syntaxe originale CSS.



Transition douce avec ***.scss**.

Les fonctionnalités “de base”



- Imports
- Variables
- Nestings
- Opérations
- Fonctions
- Mixins
- Extends & Placeholders

Installation et Utilisation

Installation :

```
npm install -g sass
```

Transpiler un fichier scss en css :

```
sass style.scss style.css
```

Online playground :

<https://www.sassmeister.com/>

Avec Angular-cli, il est possible de créer un projet déjà entièrement configuré avec sass avec les paramètres suivants :

```
ng new myApp --style=scss
```

Cette configuration intègre même un mécanisme de watch permettant de transpiler automatiquement tous les fichiers scss modifiés en css.



Imports

Sass permet de découper le code source en plusieurs fichiers pour gagner en maintenabilité, un fichier peut en appeler un autre avec **@import**.



Il faut bien distinguer import CSS vs Sass

Ordre des déclarations

Un fichier partiel commence par _



```
body {  
    background: whitesmoke;  
}  
  
// Fichier contenant h1 {color: blue;}  
@import "partials/typo";
```

Variables

Il est possible de rendre plus dynamique la génération de CSS avec l'utilisation de **variables**. Celles-ci se déclarent et s'utilisent avec le symbole `$`.

Chaque variable peut prendre la valeur d'une propriété CSS :

`#9013FE`, `24px`, `right`, etc.



Il existe des types de données.

Il faut bien distinguer variable CSS vs Sass.

Variables



```
$purple: #9013FE;  
$default-padding: 24px;  
  
.title {  
  color: $purple;  
  font-size: 2em;  
  padding: $default-padding;  
}  
.box {  
  border-bottom: 1px solid $purple;  
  padding: $default-padding;  
}
```



```
.title {  
  color: #9013FE;  
  font-size: 2em;  
  padding: 24px;  
}  
.box {  
  border-bottom: 1px solid #9013FE;  
  padding: 24px;  
}
```

Nestings

Sass permet l'**imbrication des sélecteurs**, ceci afin de ne pas avoir à les répéter et ainsi gagner en productivité et en lisibilité.

En imbriquant un sélecteur dans un autre, Sass les chaînera. Pour un chaînage direct, on utilisera **&**.

Nestings



```
section {  
    border: 1px solid whitesmoke;  
    .alert {  
        color: red;  
    }  
    &.info {  
        color: yellow;  
    }  
}
```



```
section {  
    border: 1px solid whitesmoke;  
}  
  
section.alert {  
    colour: red;  
}  
  
section.info {  
    color: yellow;  
}
```

Nestings



```
.promo {  
    color: pink;  
}  
.promo p {  
    text-decoration: none;  
}  
.promo p:hover {  
    color: deeppink;  
}
```



Types de données

Pour l'affectation de variable, Sass est assez laxiste, en fait même si on ne les déclare pas explicitement, il dispose de différents types de données qui ont leur importance :

- **Chaînes de caractères**
- **Nombres**
- **Couleurs**
- **Listes**
- **Maps**



```
$string: 'left';
$number: 1em;
$colour: magenta;
$list: (blue, white, red);
$map: (
  'small': 767px,
  'medium': 992px,
  'large': 1200px,
);
```

Opérations

Il est possible de réaliser des opérations sur des données selon leur type :

- Chaînes de caractères : `+`
- Nombres : `+`, `-`, `/` et `*`
- Couleurs : `+`, `-`, `/` et `*`



```
$container-width: 960px;  
$num-columns: 3;  
.column {  
    width: $container-width / $num-columns;  
}
```

Listes et Maps ne supportent pas d'opérations mais des fonctions.

Opérations



```
.container {  
    width: 768px;  
}  
.box-main {  
    margin: 0 10px;  
    width: 748px;  
}  
.box-left, .box-right {  
    margin: 0 10px;  
    width: 364px;  
    display: inline-block;  
}
```



Fonctions

Puisque l'on peut faire des opérations, factorisons des traitements avec des fonctions.

Se déclare via `@function myFonction(...){...}` avec un `return`.

S'utilise via `myFonction(...)` en retournant une unique valeur (sans effets de bords).



```
@function sum($value1, $value2) {  
    @return $value1 + $value2;  
}  
h1 {  
    color: sum(blue, red);  
}
```

Fonctions natives

Sass dispose déjà de nombreuses fonctions “**prêtes à l'emploi**”.

Toutes les fonctions natives sont (bien) documentées sur le site officiel de Sass => <http://sass-lang.com/documentation/Sass/Script/Functions.html>



```
$colour: red;  
  
a {  
    colour: darken($colour, 10);  
}  
a:hover {  
    colour: lighten($colour, 10);  
}
```

Mixins

Les fonctions arrivent vite à leur limite puisqu'elles ne peuvent retourner qu'**une seule valeur**.

Qui plus est elles renvoient uniquement une valeur et non une **association propriété / valeur**.

Se déclare via **mixin myMixin(...){...}**.

S'utilise via **@include myMixin(...)** en retournant son contenu (sans effets de bords).

Mixins



```
@mixin text-truncate($width) {  
  overflow: hidden;  
  text-overflow: ellipsis;  
  white-space: nowrap;  
  width: $width;  
}  
.teaser {  
  @include text-truncate(300px);  
}  
.chapter {  
  @include text-truncate(450px);  
}
```



```
.teaser {  
  overflow: hidden;  
  text-overflow: ellipsis;  
  white-space: nowrap;  
  width: 300px;  
}  
.chapter {  
  overflow: hidden;  
  text-overflow: ellipsis;  
  white-space: nowrap;  
  width: 450px;  
}
```

Extends

L'héritage est une alternative aux mixins, il ne va pas dupliquer les propriétés CSS mais les regrouper.

Se déclare via `@extend` suivi du nom sélecteur dont l'on souhaite hériter.



Il n'est pas possible d'hériter d'un sélecteur chaîné
(`.containerer h1`).



```
.info {  
    border-radius: 4px; padding: 20px;  
}  
.alert {  
    @extend .info;  
    background: red;  
}
```

Extends



```
.button,  
.button-warn,  
.button-success {  
    background: white;  
}  
.button-warn {  
    border: 1px solid orange;  
}  
.button-success {  
    border: 1px solid green;  
}
```



Placeholders

Il s'agit d'une variante de **@extend** dans le sens où le sélecteur dont on souhaite hériter ne sera pas généré.

Se déclare via **%** suivi du nom sélecteur dont l'on souhaite hériter.



```
%info {  
    border-radius: 4px; padding: 20px;  
}  
.alert {  
    @extend %info;  
    background: red;  
}
```

Placeholders



```
ul li, ol li {  
    font-size: 1em;  
    padding: 8px 0;  
    line-height: 1.5em;  
}  
ol li {  
    color: green;  
}
```



Les fonctionnalités “avancées”



- Les arguments optionnels
- Les listes d'arguments
- L'interpolation
- Les structures de contrôle

Les arguments optionnels

Pour les fonctions et les mixins il est possible de rendre un **argument optionnel** en lui spécifiant une **valeur par défaut**.

Se déclare via : suivi de sa valeur au sein de la définition de la fonction ou du mixin.



Lors de l'appel Sass fera la correspondance selon leur ordre de déclaration. Pour parer cela il faut spécifier l'argument de la même manière qu' il est déclaré.

Les arguments optionnels



```
@function pxtorem($val: 24px, $base: 16px) {  
  @return ($val / $base) * 1rem;  
}  
  
h1 {  
  margin: pxtorem();  
  padding-bottom: pxtorem(32px);  
  padding-top: pxtorem($base: 32px);  
}
```

Listes d'arguments

Pour les fonctions et les mixins il est possible de passer des **listes d'arguments**

Se déclare via `...` après les autres arguments



Il ne peut y avoir qu'une seule liste d'arguments et il doit s'agir du dernier argument.

Listes d'arguments



```
@mixin linear-gradient($direction, $gradients...) {
  background-image: linear-gradient($direction, $gradients);
}
.france {
  @include linear-gradient(to right, blue, white, red);
}
```

L'interpolation

Sass ne peut utiliser les variables en l'état en dehors des blocs de déclaration prévu à cet effet

Pour sortir du cadre, il faut passer par l'interpolation des variables qui se déclare par `#{...}`



```
$colour: red;  
.btn-#${$colour} {  
    background: $colour;  
}
```

L'interpolation



```
BodyLayout-maincontent {  
    float: left;  
    width: 75%;  
}  
BodyLayout-sidebar {  
    float: right;  
    width: 25%;  
}
```



Les structures de contrôle

Il est possible d'utiliser les structures de contrôle classiques comme dans des langages dynamiques :

- **if, else**
- **for**
- **while**
- **each**

if, else



```
@mixin contrast($colour) {
  @if lightness($colour) < 50% {
    color: white;
  } @else {
    color: black;
  }
}
$bg-color: green;
h1 {
  background: $bg-color;
  @include contrast($bg-color);
}
```

if, else



```
.h3 {  
    font-size: 3em;  
    text-shadow: 0.3em 0.3em blue;  
}  
.h4 {  
    font-size: 2.4em;  
    text-shadow: 0.24em 0.24em blue;  
}  
.h5 {  
    font-size: 2em;  
}
```



for



```
@for $i from 1 through 3 {  
  .column-#{$i} {  
    width: 100% / $i;  
  }  
}
```

for



```
.gray-0 { color: black; }
.gray-25 { color: #404040; }
.gray-50 { color: gray; }
.gray-75 { color: #bfbfbf; }
.gray-100 { color: white; }
```



while

The logo for the CSS preprocessor Sass, featuring the word "Sass" in a stylized, handwritten pink font.

```
$column: 3;  
@while $column > 0 {  
  .column-#{$column} {  
    width: 100% / $column;  
  }  
  $column: $column - 1;  
}
```

each

Sass

```
$alerts: (success, error);

@each $alert in $alerts {
    .alert-#{$alert} {
        background-image: url("/images/#{$alert}.png");
    }
}
```



```
.alert-success {
    background-image: url("/images/success.png");
}

.alert-error {
    background-image: url("/images/error.png");
}
```

each



```
.icon-left-kissing::before {  
    content: "K";  
    margin-left: 16px;  
}  
.icon-left-smiling::before {  
    content: "S";  
    margin-left: 16px;  
}  
.icon-left-fearful::before {  
    content: "F";  
    margin-left: 16px;  
}
```



Sass : Conclusion



- Facile à mettre en place
- Transition douce
- Intégré dans le CLI d'Angular
- Vous devez connaître CSS

Documentation officielle : <https://sass-lang.com/guide>

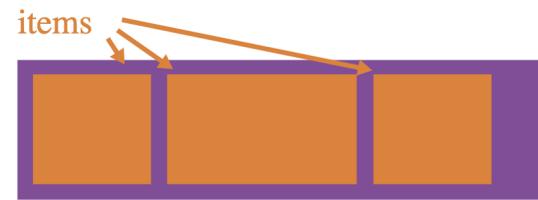
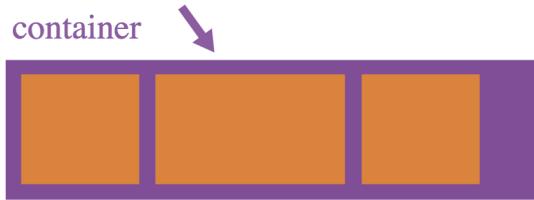
CSS 3 (some tips)



Flexbox memo

URL : <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

► Basics & Terminology



Properties for the Parent (flex container)

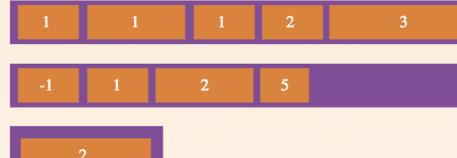
display

This defines a flex container; inline or block depending on the given value. It enables a flex context for all its direct children.

css

Properties for the Children (flex items)

order



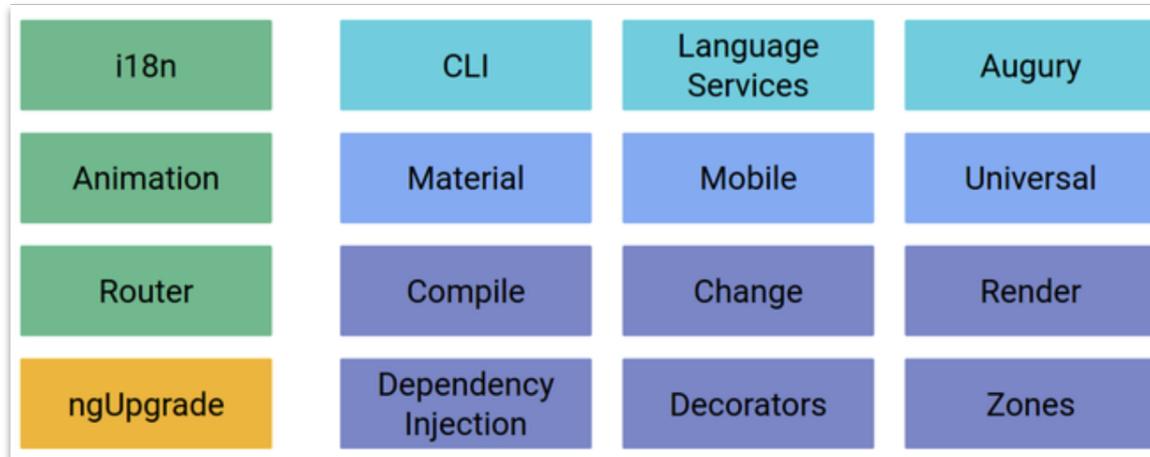
CSS

<https://medium.com/free-code-camp/the-css-handbook-a-handy-guide-to-css-for-developers-b56695917d11>

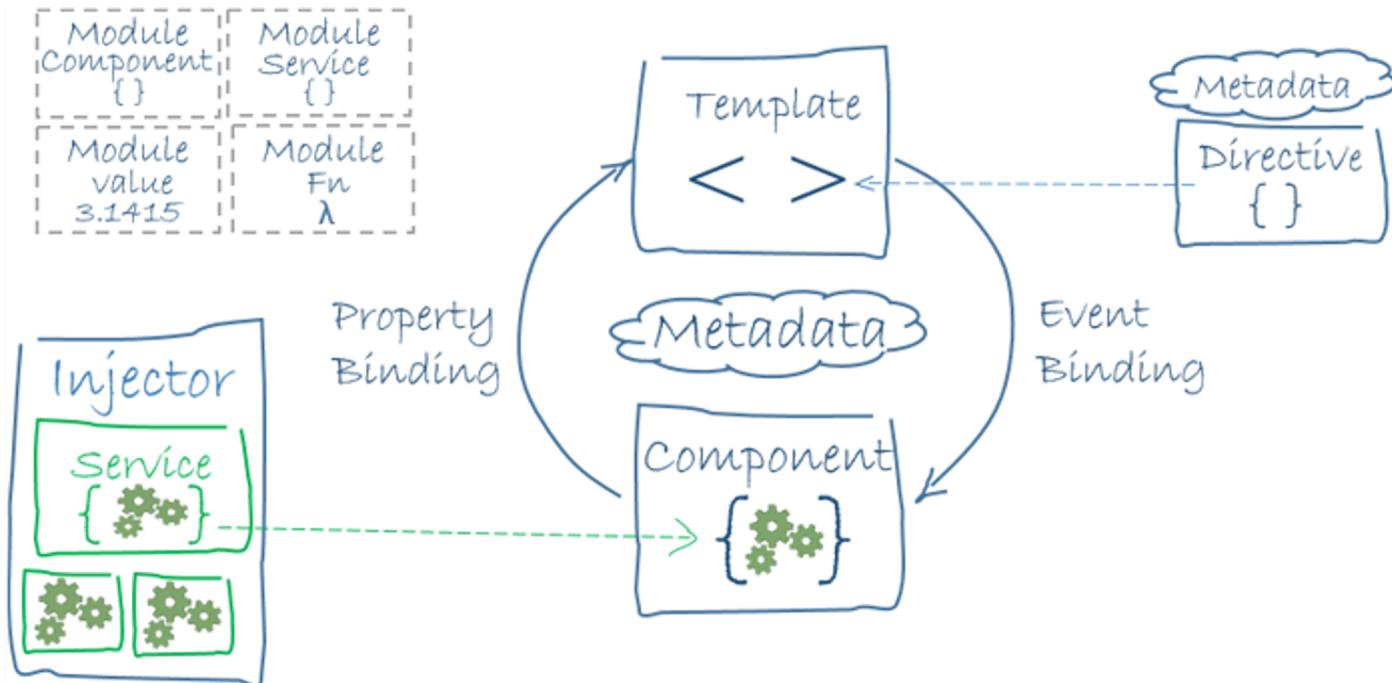
Architecture Générale

Angular, une plateforme

- Plus qu'un framework
- Écosystème riche
- Outilage destiné au développement



Angular, architecture générale



Angular, architecture générale

Module : regroupement d'un ensemble de fonctionnalités(composants, services, ...) sous un même namespace

Library Modules : @angular/core, @angular/http...

Composant : Elément graphique composé d'un template et d'une classe

Métadata : Moyen d'indiquer à Angular comment utiliser la classe

Directive : composant sans template (ngFor, ngIf, ...)

Service : Code métier implémenté dans des classes qui seront injectées dans les différents composants

Pipe : Elément permettant de formatter une donnée (équivalent au filter d'AngularJS)

Angular, architecture générale : exemple

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

@Component({
  selector: 'app',
  template: '<h1>{{value |UpperCase}}</h1>'
})
export class MyComponent {

  public value: string = 'test';

  constructor(private http: HttpClient){
  }

}
```

angular-cli

- <https://cli.angular.io/>
- Maintenu par l'équipe Google (basé sur le projet **ember-cli**)
- Permet de créer un projet, un service, un composant, ...
- Se configure avec le fichier `angular.json`



```
// Installed ?
ng --version

// Install
npm install -g @angular/cli

// Update by NPM
npm uninstall -g @angular/cli
npm install -g @angular/cli
```

```
> npm install -g @angular/cli
> ng new my-dream-app
> cd my-dream-app
> ng serve
```

Angular CLI

A command line interface for Angular

GET STARTED

Codelyzer

URL : <http://codelyzer.com/>

Set de règles pour **TsLint**.

Analyse statique du code pour les projets Angular développés en Typescript.

Déjà configuré dans les application générées avec angular-cli.

```
{  
  "rulesDirectory": [  
    "node_modules/codelyzer"  
  ],  
  "rules": {}  
}
```

tslint.json



Codelyzer ++

AirBnB

<https://www.npmjs.com/package/tslint-config-airbnb>

angular-cli : ng update

Commande du cli depuis Angular **v6**

ng update liste les mise à jour disponible ainsi que les commandes à exécuter

Name	Version	Command to update
@angular/cli	6.0.1 -> 6.0.5	ng update @angular/cli
@angular/core	6.0.2 -> 6.0.3	ng update @angular/core
rxjs	6.1.0 -> 6.2.0	ng update rxjs

Root module class : app.module.ts

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

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

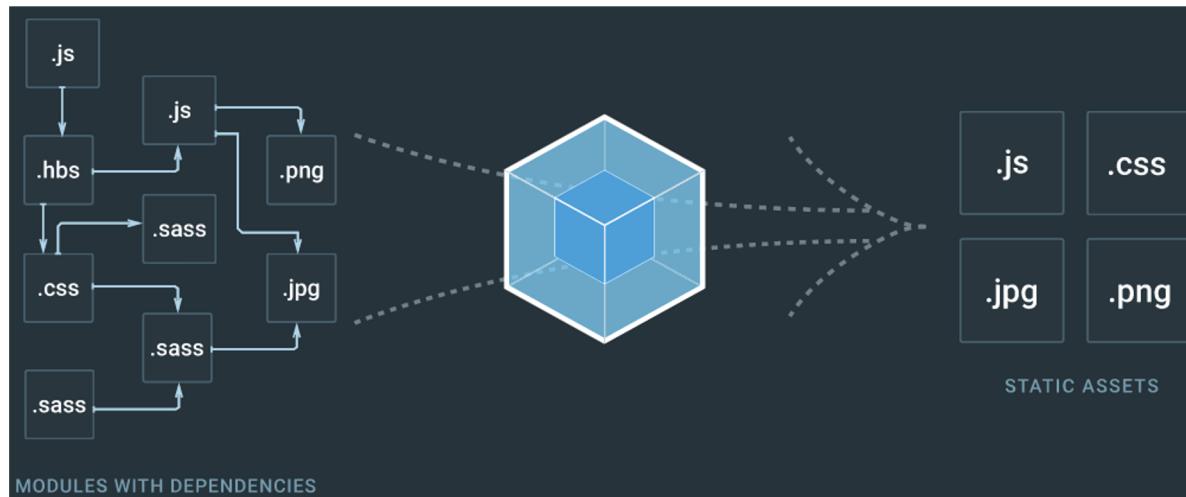
@NgModule({
  declarations: [ // Liste des éléments utilisés dans les vues (composants, pipes,...)
    AppComponent
  ],
  imports: [ // Liste des modules utilisés
    BrowserModule
  ],
  providers: [], // Liste des éléments injectables (services, resolvers, guards,...)
  bootstrap: [AppComponent], // Point d'entrée de l'application
  entryComponents: [] // Liste des composants non-rattachés au DOM (modals,...)
})
export class AppModule { }
```

>=6.X : Les services et autres éléments injectables peuvent se déclarer eux même avec l'attribut 'providedIn'.
=> Améliore l'efficacité du tree-shaking

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

Webpack

- Gestionnaire de modules
- Construit un graphe de toutes les dépendances de l'application
- Configuration dans un fichier de configuration ([webpack.config.js](#))
- Encapsulé dans le projet **Angular** (peut être extrait avec [ng eject](#))



lodash

<https://lodash.com/>



```
# Install with npm
npm install lodash --save
npm install @types/lodash --save-dev
```

```
// And in typescript file :
import * as _ from 'lodash';
```

Premières commandes

Créer un projet avec **angular-cli** :

```
ng new <appName> --prefix=<apn>
```

Builder l'application pour la production :

```
npm run lint && ng build --prod --base-href /<appName>/
```

=> à mettre dans la sections script du package.json



Arborescence

```
▼ └─ unicorn-ng ~/work-sources/formations/tmp/unicorn-ng
    ├ .idea
    ├ e2e
    └ src
        ├ app
        ├ assets
        └ environments
            ├ environment.prod.ts
            ├ environment.ts
            ├ browserslist
            ├ favicon.ico
            ├ index.html
            ├ karma.conf.js
            ├ main.ts
            ├ polyfills.ts
            ├ styles.scss
            ├ test.ts
            ├ tsconfig.app.json
            ├ tsconfig.spec.json
            └ tslint.json
        └ .editorconfig
    └ .gitignore
    └ angular.json
    └ package.json
    └ README.md
    └ tsconfig.json
    └ tslint.json
```



Montée de versions du framework (angular + cli + material)

<https://update.angular.io>

The screenshot shows a web browser displaying the Angular Update Guide at <https://update.angular.io>. The page has a blue header bar with the title "Angular Update Guide". Below the header, there is a form titled "Select the options matching your project:".

Angular Version: Two dropdown menus are shown side-by-side, both set to "6.1".

App Complexity: A horizontal row of three buttons: "Basic", "Medium" (which is highlighted in grey), and "Advanced".

Other Dependencies: A section containing two checkboxes:

- I use ngUpgrade for using AngularJS and Angular at the same time
- I use Angular Material

Package Manager: A horizontal row of two buttons: "npm" (which is highlighted in grey) and "yarn".

Show me how to update! A large blue button at the bottom of the configuration section.

Angular Update Guide | 6.1 > 7.0 for Medium Apps

Before Updating

- Switch from `HttpClientModule` and the `Http` service to `HttpClientModule` and the `HttpClient` service. HttpClient simplifies the default ergonomics (You don't need to map to json anymore) and now supports typed return values and interceptors. Read more on [angular.io](#).

JSON View

URL :

<https://chrome.google.com/webstore/detail/jsonview/chklaanhfefbnpoihckbnefhakgolnmc?hl=fr>



Composants

Les Web components

- Standard émergent définit dans 4 spécifications :
 - Custom elements
 - Shadow DOM : Encapsule le style et le JS du composant
 - <https://developer.mozilla.org/fr/docs/Web/HTML/Element/audio>
 - Template (`<template>`)
 - HTML imports : Permet d'importer des web components
- Polyfill : **web-component.js** (by Google, Mozilla et Microsoft)
- Bibliothèque par dessus :
 - Polymer (by Google)
 - X-tag (by Mozilla & Microsoft)

Les Composants

- En angular, une application est un arbre de composant
- Un composant est composé :
 - d'un **sélecteur** (un nom de balise)
 - d'un **template**
 - d'un **controller** (le contenu du composant lui même en typescript)
 - de feuilles de **styles** (CSS, SASS, SCSS, LESS, ...)
- Un composant est une classe avec le décorateur **@Component**
- La portée du style (CSS) d'un composant se limite au composant lui même
 - Il ne s'applique pas à ses fils, ses frères ou ses parents

Les Composants : exemple

```
// unicorn-card.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'unicorn-card',
  templateUrl: './unicorn-card.component.html',
  styleUrls: ['./unicorn-card.component.css']
})
export class UnicornCardComponent { }
```

Les Composants : template et interpolation

- Système d'**interpolation** grâce à la syntaxe `{{ expression }}`
- L'expression peut être :
 - une chaîne de caractère
 - la valeur d'une variable
 - la valeur rentrée d'une fonction (*mauvaise idée*)
- Cette expression sera convertie en **string** avant son affichage
- Une expression ne doit pas modifier l'état de l'application

Components & cli

```
npx ng generate component <componentName>  
npx ng g c <componentName>
```



Les Composants : les paramètres

Un composant peut avoir des **paramètres**

On utilise la syntaxe **[paramName]="variable"**
pour passer les paramètres au composant

Le composant définit
ses paramètres avec
le décorateur **@Input** :

```
// foo.component.ts
import { Component, OnInit, Input } from '@angular/core';

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

  @Input()
  public title: string;

  constructor() {}

  ngOnInit() {
    // Les paramètres sont utilisables ici, pas dans le constructeur
    console.log(this.title);
  }
}
```

```
// foo.component.html
<app-foo [title]="titleVar"></app-foo>
```

Les Composants : les événements natifs

Dans son template, un composant peut réagir aux évènements javascript

On utilise la syntaxe **(eventName)="actionOnEventSent()"**

```
<button (click)="doSomething()"></button>  
  
<unicorn (click)="doSomethingElse()"></unicorn>
```

Les Composants : les événements 2/2

Un composant peut émettre ses propres **événements**

```
// product-list.component.ts
<product (removed)="removeProductFromCart()"></product>
```

```
// cart-product.component.ts
import { Component, EventEmitter, Output } from '@angular/core';
```

```
@Component({
  selector: 'product',
  templateUrl: './product.component.html',
  styleUrls: ['./product.component.css']
})
export class ProductComponent {
```

```
  @Output()
  private removed = new EventEmitter();
```

```
  constructor() {}
```

```
  public remove(): void {
    this.removed.emit();
  }
}
```

```
// cart-product.component.html
<button (click)="remove()"></button>
```

Directive : Définition

- Directive = Composant sans template
- Directive = classe annotée avec `@Directive`
- Directive ≈ décorateur pour le HTML
- Doit avoir un sélecteur CSS (`selector`)
- Peut avoir des entrées (`inputs`)
- `@Component` hérite de `@Directive`
- Exemple :

```
<!-- test.html -->
<div logFooBar></div>
```

```
// log-foo-bar.directive.ts
import { Directive } from '@angular/core';

@Directive({
  selector: '[logFooBar]'
})
export class LogFooBarDirective {
  constructor() {
    console.log('Foo Bar');
  }
}
```

Directive : Sélecteurs

- Les sélecteurs peuvent être de différents types :
 - **footer** : un **élément**, comme c'est généralement le cas pour les composants.
 - **.alert** : une **classe**, mais c'est plutôt rare.
 - **[color]** : un **attribut**, ce qui est le plus fréquent pour une directive.
 - **[color=red]** : un **attribut** avec une **valeur** spécifique.
- ou une combinaison de ceux-ci :
 - **footer[color=red]** : désignera un élément footer avec un attribut color à la valeur red.
 - **[color],footer.alert** : désignera n'importe quel élément avec un attribut color, ou (,) un élément footer portant la classe CSS alert.
 - **footer:not(.alert)** : désignera un élément footer qui ne porte pas (:not()) la classe CSS alert.

Directives structurelles

- Chargées à partir de **BrowserModule**
- Elles commencent par le symbole *
- Angular fourni 3 directives structurelles nativement :
 - *ngIf
 - *ngFor
 - ngSwitch & *ngSwitchCase
- L'étoile indique que l'élément sera encapsulé dans un <**template**>

```
<!-- ngIf -->
<div *ngIf="user.enabled">...</div>

<!-- ngFor -->
<div *ngFor="let user of userList">
    {{ user | json }}
</div>

<!-- ngSwitch -->
<div [ngSwitch]="contentType">
    <div *ngSwitchCase="'type_a'">...</div>
    <div *ngSwitchCase="'type_b'">...</div>
    <div *ngSwitchCase="'type_c'">...</div>
</div>
```

Autres directives de templating : NgStyle

Directive permettant d'**ajouter du style CSS**

Prend un objet JSON en paramètre

```
<!-- my-component.component.html -->
<h1 [ngStyle]="{'font-size': size}">Title</h1>
```

Autres directives de templating : NgClass

La directive ngClass **ajoute ou enlève des classes CSS**.

Il existe trois syntaxes :

- `[ngClass]="'class1 class2'"`
- `[ngClass]=["'class1', 'class2']"`
- `[ngClass]="{'class1': isClass1, 'class2': isClass2}"` (la plus courante)

Variables locales

Il est possible de nommer localement un élément avec #

Cela permet d'y faire référence ailleurs dans le template

Exemples :

```
<input type="text" #name>
<button (click)="name.focus()">Focus the input</button>
```

Les Composants : template - résumé

- `{{}}` pour l'interpolation,
- `[]` pour le binding de propriété (`@Input`) ,
- `()` pour le binding d'événement (`@Output`) ,
- `[()]` pour le 2-ways data-binding (`<input>`) ,
- `#` pour la déclaration de variable,
- `*` pour les directives structurelles.

Tests



[https://angular.io/guide/
testing](https://angular.io/guide/testing)

[FR] <https://guide-angular.wishtack.io/angular/testing>

Tests

2 types de tests :

- **tests unitaires**

- Valide le fonctionnement d'une portion de code isolée
- isolé = indépendamment de ses dépendances
- Teste les méthodes des composants / services / pipes
- ***.spec.ts**

- **tests end-to-end** ou **e2e** ("de bout en bout")

- Emule une interaction utilisateur réelle
- Utilise un vrai navigateur
- ***.e2e-spec.ts**

Karma - Test runner



- <http://karma-runner.github.io/>
- Karma = ex-Testacular
- Développé par la team **Angular**
- Permet de :
 - lancer les tests dans un ou plusieurs navigateurs
 - lancer les TU à chaque modification de fichiers
- Est entièrement configuré à travers **angular.json**
- Se lance avec la commande : **npm run ng test**
- Peut se lancer en parallèle d'un **npm run ng serve**

Test unitaire



- Les TU sont très **rapides**, on peut en lancer plusieurs centaines par seconde
- Ils sont très utiles pour tester les cas limites
- Bibliothèque de rédaction de TU : **Jasmine** (<https://jasmine.github.io/>)
- En utilisant **angular-cli** pour générer des éléments, une structure de test est générée en parallèle
- Respectent la structure : **Given** / **When** / **Then**

Test unitaire - exemple

```
// user.model.ts
class User {
    constructor(public firstName: string,
                public lastName: string) { }
    getName() {
        return `${this.firstName} ${this.lastName.toUpperCase()}`;
    }
}
```

```
// user.model.spec.ts
describe('My first test suite', () => {
    it('should construct a User', () => {
        // Given
        const firstname = 'Bob';
        const lastname = 'Dylan';

        // When
        const user = new User(firstname, lastname);

        // Then
        expect(user.firstName).toBe('Bob');
        expect(user.lastName).toBe('Dylan');
        expect(user.getName()).toBe('Bob DYLAN');
        expect(user.getName()).not.toBe('Dick RIVERS');
    });
});
```

Test unitaire - expect

- La fonction **expect** peut être chaînée avec :
 - `expect(foo).toBe()`
 - `expect(foo).toBeLessThan()`
 - `expect(foo).toBeUndefined()`
 - `expect(foo).toBeDefined()`
 - `expect(foo).toBeNull()`
 - `expect(foo).toBeFalsy()`
 - `expect(foo).toBeTruthy()`
 - `expect(foo).toContain()`
 - `expect(foo).toThrowError()`
 - `expect(foo).toEqual()`
 - `expect(foo).toHaveBeenCalledWith()`
 - ...
- Exemples : <https://jasmine.github.io/2.8/introduction.html>

Injection de dépendance 1/2

Exemple (généré par **angular-cli**) :

```
import {TestBed, inject} from '@angular/core/testing';

describe('UserService', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      providers: [UserService]
    });
  });

  it('should be created', inject([UserService], (service: UserService) => {
    expect(service).toBeTruthy();
  }));
});
```

Injection de dépendance 2/2

Exemple avec la logique du **TestBed** externalisée :

```
import { TestBed, inject } from '@angular/core/testing';

describe('UserService', () => {
  let service: UserService;
  beforeEach(() => { TestBed.configureTestingModule({ providers: [UserService] }); });
  beforeEach(() => service = TestBed.get(UserService));

  it('should be created', () => {
    expect(service).toBeTruthy();
  });
});
```

Test unitaire - imports

Chaque test pouvant être exécuté indépendamment des autres, il est nécessaire d'importer tous les éléments nécessaires à son exécution dans la configuration du TestBed.

```
TestBed.configureTestingModule({
  imports: [HttpClientModule],
  providers: [UnicornService]
});
```

Headless

Il est possible d'exécuter les tests dans une version **headless** de **chrome** avec la configuration de karma suivant :

```
// karma.conf.js
...
browsers: ['ChromeHeadless'],
...
customLaunchers: {
  ChromeHeadless: {
    base: 'Chrome',
    flags: [
      '--headless',
      '--disable-gpu',
      // Without a remote debugging port, Google Chrome exits immediately.
      '--remote-debugging-port=9222',
    ],
  },
},
```

Exemple complet :

<https://gist.github.com/cvuorinen/543c6f72f8ec917ebfd596802d387aa3>

TestHostComponent

```
import {async, ComponentFixture, TestBed} from '@angular/core/testing';
import {UnicornCardComponent} from './unicorn-card.component';
import {Component, DebugElement, NO_ERRORS_SCHEMA} from '@angular/core';
import {Unicorn} from '../../../../../models/unicorn.model';
import {By} from '@angular/platform-browser';

describe('UnicornCardComponent', () => {
  let hostComponent: TestHostComponent;
  let component: UnicornCardComponent;
  let fixture: ComponentFixture<TestHostComponent>; // Toutes les metadata associées au composant (template, instance, ...)

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [UnicornCardComponent, TestHostComponent],
      // Tells the compiler not to error on unknown elements and attributes
      schemas: [NO_ERRORS_SCHEMA], // OU CA OU IMPORTER
    }).compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(TestHostComponent);
    hostComponent = fixture.componentInstance;
    hostComponent.unicorn = {id: 1} as Unicorn;
    component = queryCssSelector(fixture, 'uni-unicorn-card').componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });

  it('should have unicorn as Input', () => {
    // When
    const card: UnicornCardComponent = fixture.debugElement.query(By.css('uni-unicorn-card')).componentInstance;

    // Then
    expect(card.unicorn).toBe(hostComponent.unicorn);
  });
}

// ...
```

TestHostComponent

```
// ...

it('should set isYoung property at true on init', () => {
  // Given
  const unicorn: Unicorn = {id: 1, birthyear: new Date().getFullYear()} as Unicorn;
  hostComponent.unicorn = unicorn;

  // When
  fixture.detectChanges();

  // Then
  expect(component.isYoung).toBeTruthy();
});

it('should set isYoung property at false on init', () => {
  // Given
  const unicorn: Unicorn = {id: 1, birthyear: new Date().getFullYear() - 1} as Unicorn;
  hostComponent.unicorn = unicorn;

  // When
  fixture.detectChanges();

  // Then
  expect(component.isYoung).toBeFalsy();
});

@Component({
  template: `<uni-unicorn-card [unicorn]="unicorn" (deleted)="deleteFromList($event)"></uni-unicorn-card>`
})
class TestHostComponent {
  unicorn: Unicorn;
  deleteFromList = (unicorn: Unicorn) => {};
}

export function queryCssSelector(fixture: ComponentFixture<any>, selector: string): DebugElement {
  return fixture.debugElement.query(By.css(selector));
}
```

Test - Exercice



Objectif :

- Configurer Chrome Headless
- Créer un service `magic.service.ts`
- Créer une fonction "enchant" (si ca n'est pas déjà fait) pour enchanter un nom de licorne (une lettre sur deux en majuscule)
- **Tester cette fonction** (n'oubliez pas les cas limites)

Notes :

- Pour ne tester qu'une méthode (`focus`), remplacez "it" par "`fit`"
- Pour ne tester qu'une classe (`focus`), remplacez "describe" par "`fdescribe`"

```
import { TestBed } from '@angular/core/testing';
import { MagicService } from './magic.service';

fdescribe('MagicService', () => {

  let service: MagicService;
  beforeEach(() => TestBed.configureTestingModule({}));
  beforeEach(() => service = TestBed.get(MagicService));

  it('should be created', () => {
    expect(service).toBeTruthy();
  });

  it('should enchant unicorn name', () => {
    // Given
    const cases: any[] = [
      {originalName: 'Rainbow', expectedEnchantedName: 'RaInBoW'},
      {originalName: 'RAINBOW', expectedEnchantedName: 'RaInBoW'},
      {originalName: 'rainbow', expectedEnchantedName: 'RaInBoW'},
      {originalName: '', expectedEnchantedName: ''},
      {originalName: null, expectedEnchantedName: ''},
      {originalName: undefined, expectedEnchantedName: ''},
    ];

    // When
    cases.map(c => c.enchantedName = service.enchant(c.originalName));

    // Then
    cases.forEach(c => {
      expect(c.enchantedName).toEqual(c.expectedEnchantedName);
    });

  });
});
```



Tests e2e



Protractor

- <http://www.protractortest.org/>
- Même framework et même fonctionnement qu'en AngularJS (1.X)
- Lancent l'application dans un navigateur et simulent une utilisation
- Permet de tester l'application dans son ensemble
- Ils sont plus lents que les TU (plusieurs secondes par test)
- Souvent difficile de tester les cas limites
- Se mettent en place en **complément des TU**
- Se lance avec **npm run e2e**

Pipes

Les Pipes

Les pipes changent les valeurs affichées dans le template HTML

Similaire au **filter** dans **AngularJS**

Utilisation dans les templates HTML similaire à **AngularJS**

Pour appliquer un Pipe, il faut utiliser le caractère |

Pipes natifs

Pipes disponibles par défaut dans le framework ([@angular/common](#)):

- **LowerCasePipe**, **UpperCasePipe**
- **CurrencyPipe**, **DecimalPipe**, **PercentPipe**
- **DatePipe**, **JSONPipe**, **SlicePipe**
- **I18nPluralPipe**, **I18nSelectPipe**
- **AsyncPipe**



Pas de FilterPipe ni de OrderByPipe

Ils existaient en **AngularJS**, pas en **Angular**

Problèmes de performances

Impossible à minifier efficacement

Il est préférable de sortir les logiques de tri et de filtre dans le composant et d'exposer à la vue des **filteredFoos** ou des **sortedBars**

Utilisation dans le template

Les Pipes natifs sont directement utilisables dans les templates

Pour appliquer un Pipe, il faut utiliser le caractère |

Possibilité de chaîner les pipes les uns à la suite des autres

```
 {{ myVar | date | uppercase }}  
 //THURSDAY, OCTOBER 17, 1985
```

Certains pipes sont configurables

Séparation des paramètres par le caractère :

```
 {{ price | currency:'EUR':true }}
```

Écrire un pipe

- Définir une classe implémentant l'interface **PipeTransform**
- Implémenter la méthode **transform**
- Annoter la classe avec le décorateur **@Pipe**
- Exporter cette classe via **export**

```
import {isString, isBlank} from '@angular/core/src/facade/lang';
import {InvalidPipeArgumentError} from '@angular/common/src/pipes/invalid_pipe_argument_error';
import {PipeTransform, Pipe} from '@angular/core';

@Pipe({name: 'mylowercase'})
export class MyLowerCasePipe implements PipeTransform {
    transform(value: string, param1:string, param2:string): string {
        if (isBlank(value)) return value;
        if (!isString(value)) {
            throw new InvalidPipeArgumentError(MyLowerCasePipe, value);
        }
        return value.toLowerCase();
    }
}
```

Les Pipes - Utilisation dans le template

Les pipes externes nécessaires à votre application doivent :

- être définis dans un module importé par votre application (**ngModule**)
- être définis dans la propriété **declarations** du décorateur **ngModule** de votre application

```
// app.component.ts
import {Component} from '@angular/core';
@Component({
  selector: 'app',
  template: '<h2>{{'Hello World' | mylowercase}}</h2>'
})
export class App { }
```

```
// app.module.ts
import { NgModule } from '@angular/core';
import { MyLowerCasePipe } from '@angular/forms';

@NgModule({
  //...
  declarations: [MyLowerCasePipe],
  //...
})
export class AppModule { }
```

Les Pipes - Utilisation dans le component

Utilisation de l'injection de dépendances pour utiliser un **pipe**

Pas nécessaire d'utiliser un service **\$filter** ou une règle de nommage (**dateFilter**) comme en **AngularJS**

```
import {Component} from '@angular/core';
import {MyLowerCasePipe} from './mylowercase';
@Component({
  selector: 'app',
  providers: [MyLowerCasePipe]
})
class App {
  name:string;

  constructor(public lower:MyLowerCasePipe){
    this.string = lower.transform('Hello Angular');
  }
}
```

Pipes & cli

```
npm run ng generate pipe <pipeName>  
npm run ng g p <pipeName>
```



Magical name pipe



Créer un pipe :

- qui affiche le nom des licornes avec une lettre sur deux en majuscule
- sans utiliser une boucle "for"



```
import {Pipe, PipeTransform} from '@angular/core';
import * as _ from 'lodash';

@Pipe({
  name: 'magicalName'
})
export class MagicalNamePipe implements PipeTransform {
  transform(originalName: string): string {
    return originalName
      .split('')
      .map((char, idx) => idx % 2 ? char.toLowerCase() : char.toUpperCase())
      .join('');
  }
}
```

Les Pipes - pures et impures

Deux catégories de Pipes : **pure** et **impure**

Un **pipe** est **pure** par défaut

Les Pipes - pures

Exécuté seulement quand l'input du pipe subit un changement **pure**

- **changement de référence**
- changement d'une valeur primitive (boolean, number, string...)

Ceci optimise les **performances** du mécanisme de détection de changement

Mais, pas dans les cas suivants :

- ajout/suppression d'un objet dans un tableau (la référence du tableau ne change pas)
- modification d'une propriété d'un objet (la référence de l'objet ne change pas)

Les Pipes - impure

Exécuté à chaque cycle du système de **Change Detection**

Pour définir un **Pipe** impure, mettre la propriété **pure** à **false**

```
@Pipe({
  name: 'myImpurePipe',
  pure: false
})
class MyImpurePipe {
  transform(value){ ... }
}
```



Les Pipes - AsyncPipe

async prend une **Promise** ou un **Observable** en paramètre

retourne la donnée émise

(**async** est un pipe **impure**)

```
@Component({
  selector: 'uni-demo-pipe-async',
  template: '{{ unicorns | async }}',
  styleUrls: ['./demo-pipe-async.component.css']
})
export class DemoPipeAsyncComponent {

  public unicorns: Observable<Unicorn[]> = this.unicornsService.getUnicorns();

  constructor(private unicornsService: UnicornsService) {}

}
```

Le JsonPipe

Le Json Pipe est un moyen simple pour debugger

Utilise **JSON.stringify** pour convertir la donnée en string

```
 {{ myMysteriousData | json }}
```

Pipes fonctions

Préférez toujours l'utilisation des pipes pour formater votre template.

Des **caches internes** sont gérés par Angular pour le rendu des pipes.

De manière générale, l'utilisation de fonctions dans le template est contre performant (elles sont exécutées à chaque cycle de détection du changement).

```
<h1>{{ format(data) }}</h1>
```

```
<h1>{{ data | format }}</h1>
```



Formulaires



Formulaires

- Angular propose **2 types** de formulaires :
 - 1 piloté par le template (adapté aux formulaires simples)
 - 1 piloté par le code
- Dans les deux cas, Angular crée une **représentation** du formulaire :
 - Automatiquement pour un formulaire piloté par le template
 - Manuellement pour un formulaire piloté par le code

Formulaire - FormControl

- Un champ du formulaire est toujours représenté par un **FormControl**
- Un **FormControl** a plusieurs attributs :

<code>value</code>	la valeur du champ
<code>valueChanges</code>	un Observable qui émet à chaque changement sur le champ
<code>valid</code>	si le champ est valide, au regard des contraintes et des validations qui lui sont appliquées
<code>invalid</code>	si le champ est invalide, au regard des contraintes et des validations qui lui sont appliquées
<code>errors</code>	un objet contenant les erreurs du champ
<code>dirty</code>	<code>false</code> , jusqu'à ce que l'utilisateur modifie la valeur du champ
<code>pristine</code>	<code>true</code> , tant que l'utilisateur n'a pas modifié le champ
<code>touched</code>	<code>false</code> jusqu'à ce que l'utilisateur soit entré dans le champ
<code>untouched</code>	l'opposé de <code>touched</code>

Formulaire - FormGroup

- Les **FormControl** peuvent être regroupés dans un **FormGroup**
- Un formulaire est aussi un **FormGroup**
- Un **FormGroup** a les mêmes propriétés qu'un **FormControl**, avec quelques nuances dans les valeurs

Formulaire - FormGroup : propriétés

value	la valeur du groupe. Pour être plus précis, c'est un objet dont les clés/valeurs sont les contrôles et leurs valeurs respectives
valueChanges	un Observable qui émet à chaque changement sur un contrôle du groupe
valid	si tous les champs sont valides, alors le groupe est valide
invalid	si l'un des champs est invalide, alors le groupe est invalide
errors	un objet contenant les erreurs du groupe, ou null si le groupe est entièrement valide. Chaque erreur en constitue la clé, et la valeur associée est un tableau contenant chaque contrôle affecté par cette erreur
dirty	false jusqu'à ce qu'un des contrôles devienne dirty
pristine	l'opposé de dirty
touched	false jusqu'à ce qu'un des contrôles devienne touched
untouched	l'opposé de touched

Formulaire piloté par le template

- Utilisation de plusieurs **directives** qui permettent à **Angular** de convertir les éléments du formulaire en **FormControl** et **FormGroup**
 - **<form>** ⇒ **FormGroup**
 - **<input>** ⇒ **FormControl**
 - **<input name="">** ⇒ **formControlName**
 - ...
- Ces directives sont dans le module **FormsModule** de **@angular/forms**

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

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

Formulaire piloté par le template : Exemple

- Exemple basé sur un mapping **unidirectionnel**
- Positionnement de **ngModel** sur les inputs
- Utilisation d'une variable locale **contactForm** pour envoyer les données au **ngSubmit**

```
<!-- add-contact.component.html -->
<h1>Formulaire</h1>

<form #addUnicornForm="ngForm"
(ngSubmit)="addUnicorn(addUnicornForm.value)">
  <input ngModel type="text" name="name"
placeholder="nom">
  <input ngModel type="number" name="birthyear"
placeholder="birthyear">
  <input ngModel type="number" name="weight"
placeholder="weight">
  <input type="submit">
</form>
```

```
// add-contact.component.ts
import {Component} from '@angular/core';
import {Unicorn} from
'../../models/unicorn.model';
import {UnicornsService} from
'../../services/unicorns.service';

@Component({
  selector: 'app-formulaire',
  templateUrl: './formulaire.component.html',
  styleUrls: ['./formulaire.component.scss']
})
export class FormulaireComponent {
```

Bananas in the box

`[ngModel]="varName"` permet un **mapping bidirectionnel**

```
<input name="username" [(ngModel)]="user.name" />
```

est l'équivalent de :

```
<input name="username" [ngModel]="user.name" (ngModelChange)="user.name = $event"
/>
```

La directive **ngModel** :

- Met à jour la valeur de l'input à chaque changement du modèle lié `user.username` ⇒ **[ngModel]="user.username"**
- Génère un événement depuis un output nommé **ngModelChange** à chaque fois que l'input est modifié par l'utilisateur, où l'événement est la nouvelle valeur ⇒ **(ngModelChange)="user.username=\$event"**

Validation et formulaire piloté par le template

4 directives sont fournies avec **Angular** :

- **required**
- **minlength**
- **maxlength**
- **email**

```
<input type="text"
      email
      required
      minlength="2"
      maxlength="10" />
```

Formulaire piloté par le code

Utilisation des directives présentes dans le module **ReactiveFormsModule** de **@angular/forms**

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

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

Formulaire piloté par le code : FormBuilder

FormBuilder est un **service** qui permet de créer des **FormControl** et des **FormGroup**

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

@Component({
  selector: 'app-add-contact',
  templateUrl: 'add-contact.component.html'
})
export class RegisterFormComponent {
  public contactForm: FormGroup;

  static isOldEnough(control: FormControl) {
    return (control.value >= 18) ? null : { tooYoung: true };
  }

  constructor(fb: FormBuilder) {
    this.contactForm = fb.group({
      username: ['', Validators.required],
      age: ['', [Validators.required, RegisterFormComponent.isOldEnough]]
    });
  }

  addContact() { console.log(this.userForm.value); }
}
```

Formulaire piloté par le code : le template

Utilisation des **directives** :

- **formGroup**
- **formControlName**

```
<form (ngSubmit)="addContact()" [formGroup]="contactForm">
  <label>name</label>
  <input type="text" formControlName="username">
  <div *ngIf="contactForm.get('username').dirty &&
    contactForm.get('username').hasError('required')">
    Name is required
  </div>
  <label>age</label>
  <input type="number" formControlName="age">
  <div *ngIf="contactForm.get('age').dirty && contactForm.get('age').hasError('required')">
    Age is required
  </div>
  <div *ngIf="contactForm.get('age').dirty && contactForm.get('age').hasError('tooYoung')">
    Age is too young
  </div>
  <button type="submit" [disabled]="contactForm.invalid">Add Contact</button>
</form>
```

Ternaires

```
// Option 1
```

```
let x;  
if (taille > 150) {  
    x = 'grand';  
} else {  
    x = 'petit';  
}
```

```
// Option 2 : ternaire
```

```
let x = (taille > 150) ? 'grand' : 'petit';
```

Services

Les services

- Les **services** sont des **classes** que l'on peut **injecter** dans d'autres classes
- **Angular** propose quelques services de base comme **Title** et **Meta**
- Il est également possible de créer ses propres services
- Exemple du service **Title** :

```
import { Component } from '@angular/core';
import { Title } from '@angular/platform-browser';

@Component({
  selector: 'mycompany-app',
  template: `<h1>My Company App</h1>`
})
export class MyCompanyAppComponent {
  constructor(private title: Title) {
    // Le service Title permet de récupérer le <title> de la page
    const name = title.getTitle().toUpperCase();
    // et de le modifier (le service créera la balise <title> si elle est absente
    title.setTitle(name);
  }
}
```

Créer son propre service

Créer un service équivaut à écrire une classe :

```
// contacts.service.ts
@Injectable({
  providedIn: 'root'
})
export class ContactsService {
  public list(): Contact[] {
    return [{ name: 'Bob' }, { name: 'Dylan' }];
}
```

Services & cli

```
npx ng generate service <serviceName>  
npx ng g s <serviceName>
```



Injectors

On peut injecter un service :

- de manière **globale** via le module principal **@NgModule**
- de manière **locale** via **@Component**

Généralement, on injecte les services dans le **@NgModule**

Les services sont injectés via le constructeur du parent et sont des **singlenton au sein du même injecteur**

Services

Dans une application **Angular**, les services servent principalement à 2 choses :

- Accéder aux API REST en utilisant le service **HttpClient**
- Utiliser le fait qu'ils soient des **singletons** pour partager des données entre **composants** séparés. Une alternative à ce point est l'utilisation d'un **Store** comme **NgRx**.

Services : Injection d'autres services

Si notre service utilise lui aussi d'autres services (comme **HttpClient**), il doit avoir le décorateur **@Injectable()**

C'est **@Injectable()** qui injecte les paramètres du constructeur

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

@Injectable({
  providedIn: 'root'
})
export class UnicornService {
  constructor(private http: HttpClient) {}
  public list(): Observable<Unicorn[]> {
    return this.http.get<Unicorn[]>('http://.../api/v1/unicorns');
  }
}
```

Service : private / public

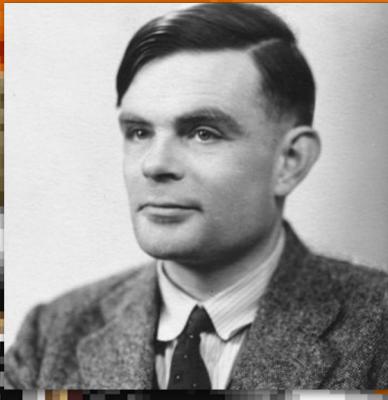
Les méthodes des services ne sont pas toujours accessibles de l'extérieur.

Une méthode **public** est accessible aux classes qui vont injecter le service

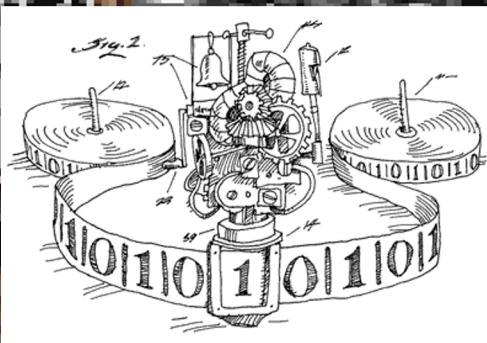
Une méthode **private** est uniquement accessible dans le service lui même

Functional Reactive Programming





Alan Turing
(1912-1954)



VS



DGFRG ! @M; @
i T\ SVÖT\ \ X\

λ

Programation fonctionnelle

<https://medium.freecodecamp.org/functional-programming-principles-in-javascript-1b8fc6c3563f>

Functional Reactive Programming

Programmation fonctionnelle

- = Programmation centrée sur l'usage de fonctions dont les paramètres sont immuables et la sortie unique
- ✗ Programmation impérative (for,...)

eg. **filter**, **map**, **reduce**, ...

Élève le niveau d'abstraction du code



Café-Craft (podcast) :

<https://overcast.fm/+KzBnKVfcE>

Programmation réactive

- = Programmation avec des **flux** de données asynchrones

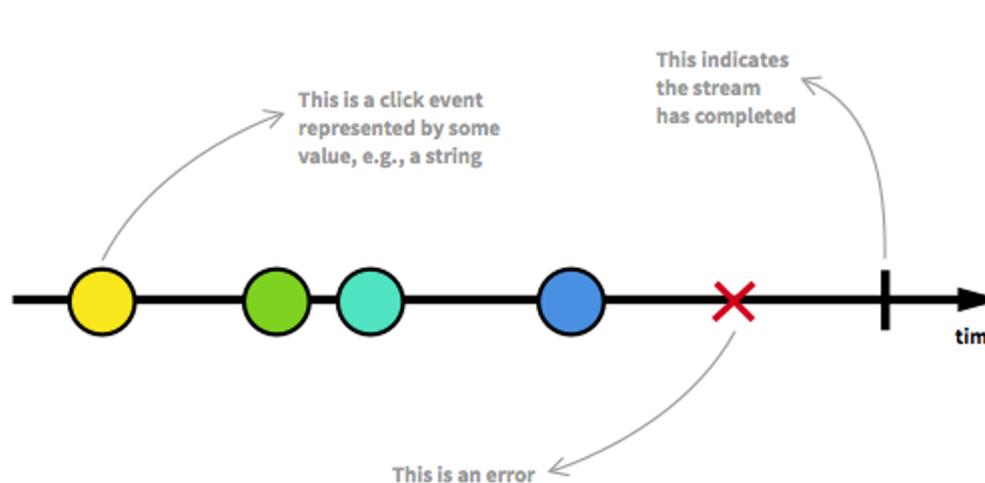
Les flux

- Un **flux** est une séquence d'événements
- en cours de production
- ordonnée dans le temps

Les flux

Trois types de signaux peuvent être émis par un flux :

- une valeur
- une erreur
- un signal « fin »



Angular & Programmation réactive : RxJS

rxjs v6





ReactiveX

An API for asynchronous programming
with observable streams

RxJS : The ReactiveX library for JavaScript

@benlesh => RxJS 5+ Lead, Software Engineer at Google, RxWorkshop.com.

Robert Penner
@robpenner

"#RxJS is #lodash for events."
— @benlesh

À l'origine en anglais

Netflix JavaScript Talks - RxJS Version 5
Ben Lesh explains what RxJS is, and why we use and love it at Netflix. He shares the motivations behind a new version of RxJS and how it was built from th...
youtube.com

RETTWEETS J'AIME
8 12

14:15 - 18 mars 2016

8 12

RxJS : 4 principaux concepts

- **Observable**
- **Observer**
- **Operator**
- **Subject**

RxJS : Observable

- **Observable** = **Flux** (coeur de RxJS)
 - séquence d'événements
 - en cours de production
 - ordonnée dans le temps
- Peut être créé à partir de nombreuses sources :

```
import {EMPTY, from, interval, merge, throwError, of} from 'rxjs';

const listStream      = of([1, 2, 3]);    // stream contenant [1, 2, 3]
const listStream      = from([1, 2, 3]);   // stream contenant 1 puis 2 puis 3
const timeStream      = interval(1000);    // stream émettant un évènement par seconde (1000ms)
const promiseStream   = from(fetch('ip.jsontest.com'));
const mergedStreams   = merge(listStream, timeStream);
const errorStream     = throwError('ERROR !'); // stream contenant 'ERROR !' et rien d'autre
const emptyStream     = EMPTY;
```

Observables : Hot & Cold

Hot Observable



- Produisent des valeurs même si le flux n'a pas de souscription.
- C'est le cas lorsque l'on crée un flux pour écouter les clics de l'utilisateur.
- Des valeurs sont produites même si l'on ne souscrit pas à cet observable.
- => vidéo live The Twitch logo, which consists of the word "twitch" in a purple, lowercase, sans-serif font with a white outline.

Cold Observable



- Ne produisent aucune valeur s'ils ne sont pas écoutés ou observés via la fonction **subscribe**
- C'est le cas des appels HTTP par exemple.
- => vidéo à la demande The Netflix logo, which is the word "NETFLIX" in a large, bold, red, sans-serif font.

Différences en Observables et Promesses

Les observables sont fainéants (**lazy**) et ne seront traités, chauds ou froids qu'à l'appel de la fonction **subscribe**.

La fonction **subscribe** renvoie un objet de type **Disposable** qui permettra d'arrêter l'écoute d'un flux (et l'arrêt de production de valeurs s'il s'agit d'un observable froid).

Promise	Observable
Exécutée dès sa création	lazy
non-annulable	annulable
Exécutée une seule fois (cache)	peut être rejoué (retry())

RxJS : Observer

Un **Observer** regroupe 3 callbacks, un pour chaque type d'évènements envoyés par un **Observable** : `next`, `error` et `complete`.

```
var observer = {
  next: x => console.log('Observer got a next value: ' + x),
  error: err => console.error('Observer got an error: ' + err),      // optionnel
  complete: () => console.log('Observer got a complete notification'), // optionnel
};
```

Pour utiliser un **Observer**,
il suffit de le passer à la fonction `subscribe` d'un **Observable** :

```
observable.subscribe(observer);
```

Généralement, un **Observer** est créé de manière **anonyme**.

RxJS : Observer / Exemples

Utilisation d'un Observer complet :

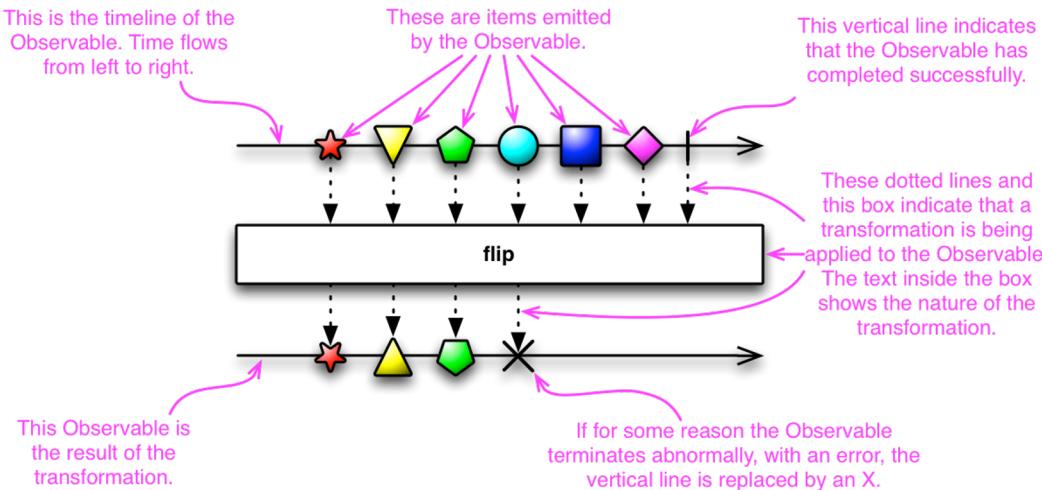
```
from(fetch('ip.jsontest.com')).subscribe({
  next: (value: Response) => {
    // ...
  },
  error: (error: Response) => {
    // ...
  },
  complete: () => {
    // ...
  }
});
```

Utilisation d'un Observer avec uniquement la méthode next :

```
from(fetch('ip.jsontest.com')).subscribe((value: Response) => {
  // ...
});
```

RxJS : Operators

- Permettent la **manipulation** d'Observables (flux)
- Retournent un autre **Observable**
- Liste : <http://reactivex.io/documentation/operators.html>



RxJS : Operators

Ressemblent à l'API de l'**Array** (et à celle de **Lodash**)

Sont les méthodes qui vont travailler sur les valeurs émises par l'**Observable**.

Permettent de **modifier** ces valeurs, de les **concaténer**, de les **filtrer**, etc.

Peuvent également être **combinés** afin de permettre des opérations complexes.

Chaining Operators

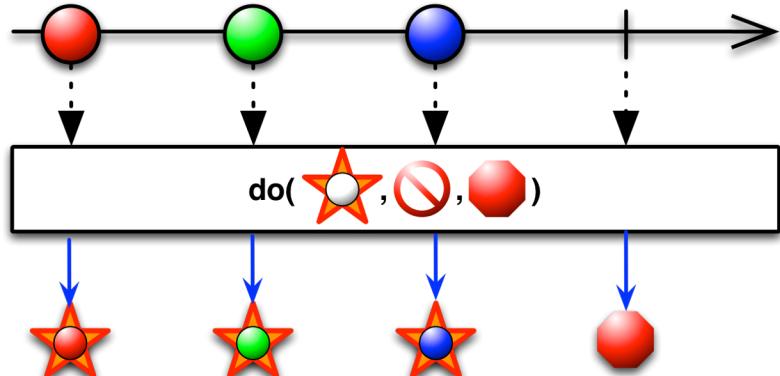
La plupart des opérateurs opèrent sur un **Observable** et renvoient un **Observable**.

Cela permet d'appliquer ces opérateurs l'un après l'autre, dans une chaîne.

Chaque opérateur dans la chaîne modifie l'**Observable** qui résulte du fonctionnement de l'opérateur précédent.

Operator : tap (ex: do)

Utile pour le **debug**



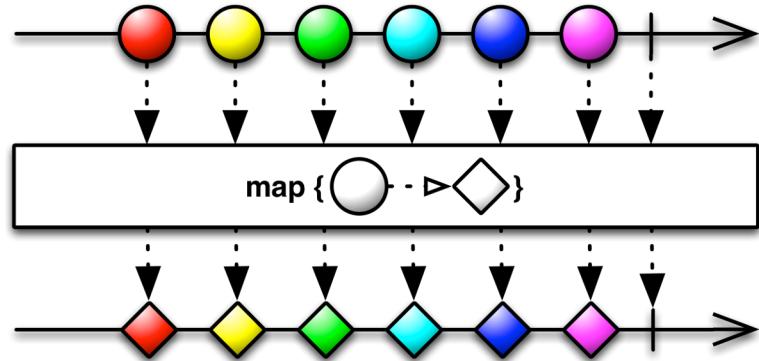
```
import {from} from 'rxjs';
import {tap} from 'rxjs/operators';

from(['a', 'b', 'c']).pipe(
  tap(e => {
    // `e` vaut successivement "a", "b" puis "c"
    console.log(e);
  }),
  // ...
).subscribe();
```

v6

Operator : map

Pour retourner une **nouvelle valeur** sur la base de l'événement reçu



```
import {from} from 'rxjs';
import {map} from 'rxjs/operators';

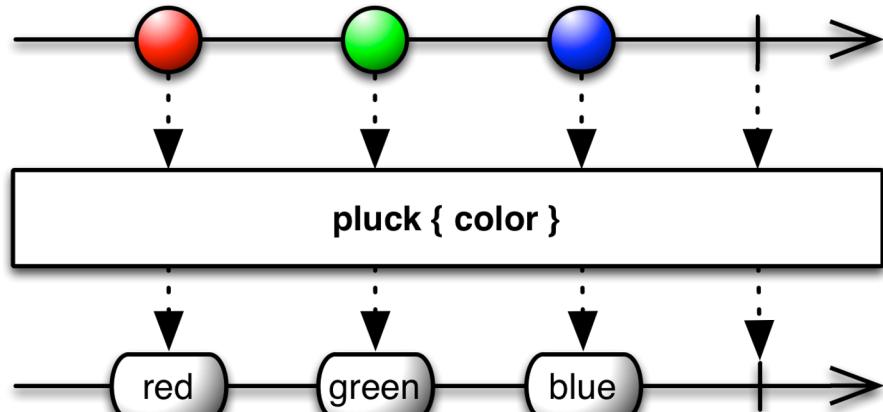
from(['a', 'b', 'c']).pipe(
  map((e: string) => {
    // `e` vaut successivement "a", "b" puis "c"
    return e.toUpperCase();
  }),
).subscribe(e => {
  console.log(e); // 'A', 'B', 'C'
});
```

v6

Operator : pluck

Retourne une propriété ou une sous-propriété des éléments du flux

Gère l'absence de propriété même dans le cas d'une arborescence



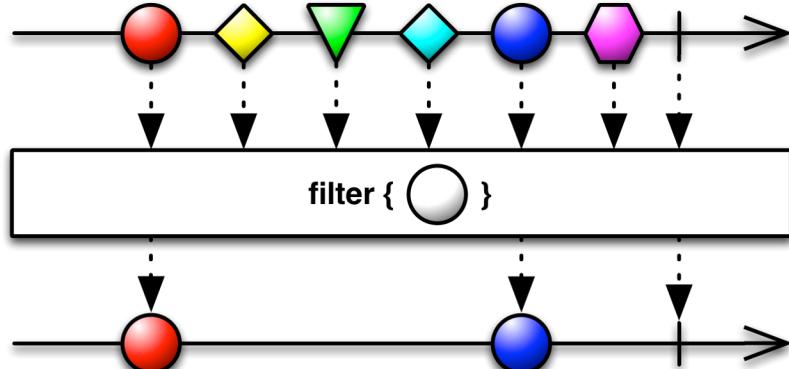
```
import {from} from 'rxjs';
import {filter} from 'rxjs/operators';

from([{name: 'bob', age: 32}, {name: 'dylan', age: 45}]).pipe(
  pluck('name'),
).subscribe(e => {
  console.log(e); // 'bob', 'dylan'
});
```

v6

Operator : filter

Pour **filtrer** certains éléments



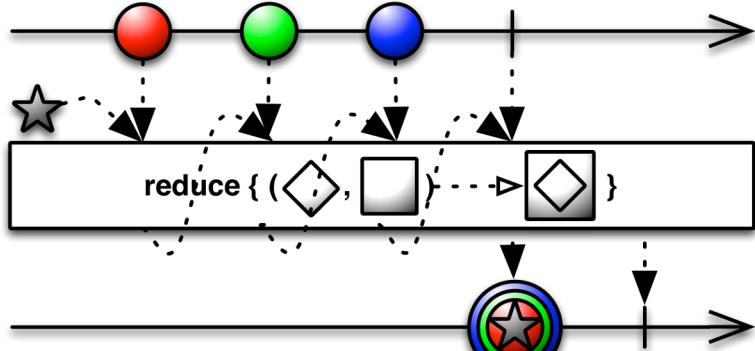
```
import {from} from 'rxjs';
import {filter} from 'rxjs/operators';

from(['A', 'b', 'C']).pipe(
  filter((e: string) => {
    // ne retourne que les lettres en majuscule
    return e === e.toUpperCase();
  }),
).subscribe(e => {
  console.log(e); // 'A', 'C'
});
```

v6

Operator : reduce

Pour appliquer successivement une opération et retourner le résultat à la fin du flux

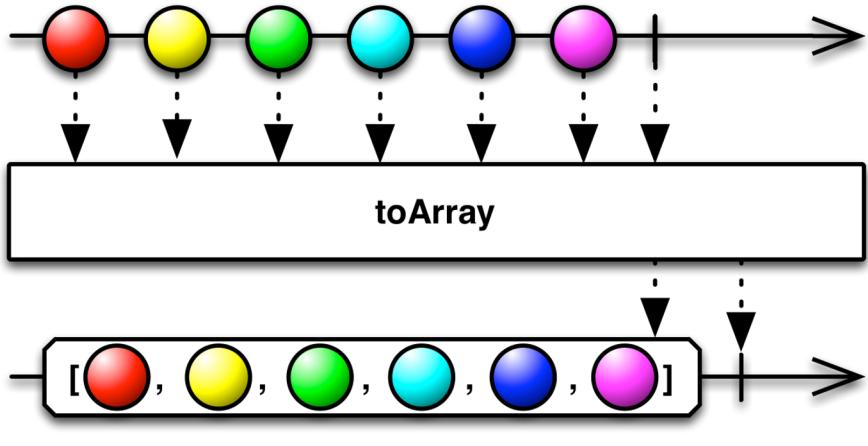


```
import {from} from 'rxjs';
import {map, reduce} from 'rxjs/operators';

from(['a', 'b', 'c']).pipe(
  map((e: string) => {
    return e.toUpperCase();
  }),
  // `reduce` agrège dans un tableau tous les éléments du flux
  reduce((acc: string[], e: string) => acc.concat(e), [])
  // dans ce cas : à le même effet qu'un "toArray()"
).subscribe(e => {
  console.log(e); // ['A', 'B', 'C']
});
```

Operator : toArray

Pour convertir
un flux de N éléments en
un tableau de N éléments



```
import {from} from 'rxjs';
import {toArray} from 'rxjs/operators';

from(['a', 'b', 'c']).pipe(
    // les éléments "a", "b" puis "c" passent successivement
    toArray(),
    // un seul élément contenant ['a', 'b', 'c'] passe
).subscribe(e => {
    console.log(e); // ['a', 'b', 'c']
});
```

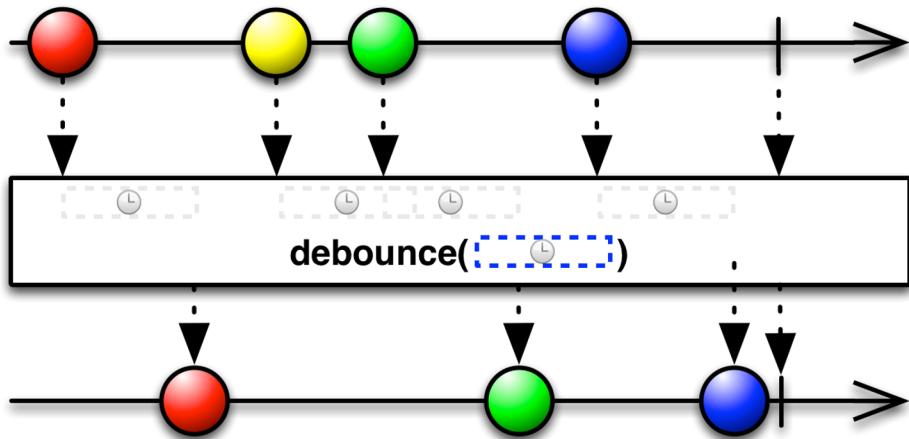
debounceTime

debounceTime() :

Ne laissez pas passer un évènement qu'après une période d'attente pendant laquelle aucun nouvel élément n'est reçu par l'opérateur.

distinctUntilChanged() :

Ne laissez pas passer l'évènement si il est identique à l'évènement précédent



```
import {Component} from '@angular/core';
import {FormBuilder, FormGroup} from '@angular/forms';
import {debounceTime, distinctUntilChanged} from 'rxjs/operators';

public myForm: FormGroup;
constructor(fb: FormBuilder) {
    this.myForm = fb.group({name: ''});
    this.myForm.get('name').valueChanges.pipe(
        debounceTime(300),
        distinctUntilChanged(),
    ).subscribe(val => {
        console.log(val);
    });
}
```

v6

Operator : flatMap

flatMap convertit un tableau de N éléments en un flux de N éléments

```
import {of} from 'rxjs';
import {flatMap, map, toArray} from 'rxjs/operators';

of(['a', 'b', 'c']).pipe(
  flatMap(e => e), // Permet de passer à map(): 'a' puis 'b' puis 'c'
  map((e: string) => e.toUpperCase()),
  toArray(),
).subscribe(e => {
  console.log(e); // ['A', 'B', 'C']
});
```



Operator : mergeMap

MergeMap permet, si la fonction en paramètre retourne un observable :
d'attendre son retour avant de passer à l'opérateur suivant

```
import {of} from 'rxjs';
import {flatMap, filter, mergeMap, map, toArray} from 'rxjs/operators';

let teams: Team[] = [];

this.http.get<Team[]>('https://.../teams').pipe(
  flatMap(e => e),
  filter((team: Team) => team.agile === true),
  mergeMap((team: Team): Observable<Team> =>
    // Attend que l'appel HTTP soit résolu avant de déclencher le toArray
    this.http.get('http://.../actualSprint?team=' + team.id).pipe(
      map((sprint: Sprint): Team => ({...team, actualSprint: sprint}));
    ),
    toArray()
  ).subscribe(teams => { this.teams = teams; });
)
```

v6

Exemple complet (v1)



Dans une fonction `getAllWithCapacityLabels()`, récupérez toutes les licornes, en utilisant la ressource '`http://localhost:3000/unicorns`'. Compléter la propriété `capacityLabels` de ces éléments grâce à l'utilisation de '`http://localhost:3000/capacities/${capacity}`'.

```
import {from} from 'rxjs';
import {flatMap, filter, mergeMap, map, toArray} from 'rxjs/operators';

this.http.get<Unicorn[]>('http://localhost:3000/unicorns').pipe(
  flatMap(e => e),
  mergeMap((unicorn: Unicorn): Observable<Unicorn> =>
    from(unicorn.capacities).pipe(
      mergeMap((capacityId: number): Observable<Capacity> =>
        this.http.get<Capacity>(`http://localhost:3000/capacities/${capacityId}`)),
      toArray(),
      map((capacities: Capacity[]): Unicorn => ({...unicorn, capacitiesObj: capacities}));
    )
  ),
  toArray()
).subscribe();
```

subscribe, subscribe, ... -> mergeMap !

Il est toujours possible de remplacer des subscribes imbriqués par l'utilisation d'un **mergeMap** (mieux, plus propre, plus élégant) :

Exple : récupérer la 1ere capacité de la 1ere licorne

```
http.get<Unicorn>('http://localhost:3000/unicorns/1').subscribe((unicorn: Unicorn) => {
  http.get(`http://localhost:3000/capacities/${unicorn.capacities[0]}`).subscribe((capacity: Capacity) => {
    console.log(capacity);
  });
});
```



```
http.get<Unicorn>('http://localhost:3000/unicorns/1').pipe(
  mergeMap((unicorn: Unicorn) => http.get(`http://localhost:3000/capacities/${unicorn.capacities[0]}`)),
).subscribe((capacity: Capacity) => {
  console.log(capacity);
});
```



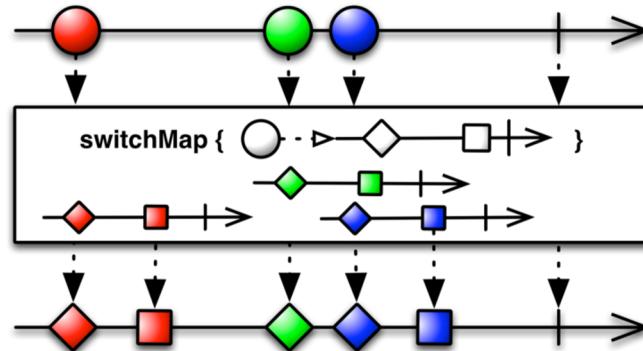
Operator : switchMap

L'opérateur **switchMap** se comporte comme le `mergeMap` à un détail prêt :

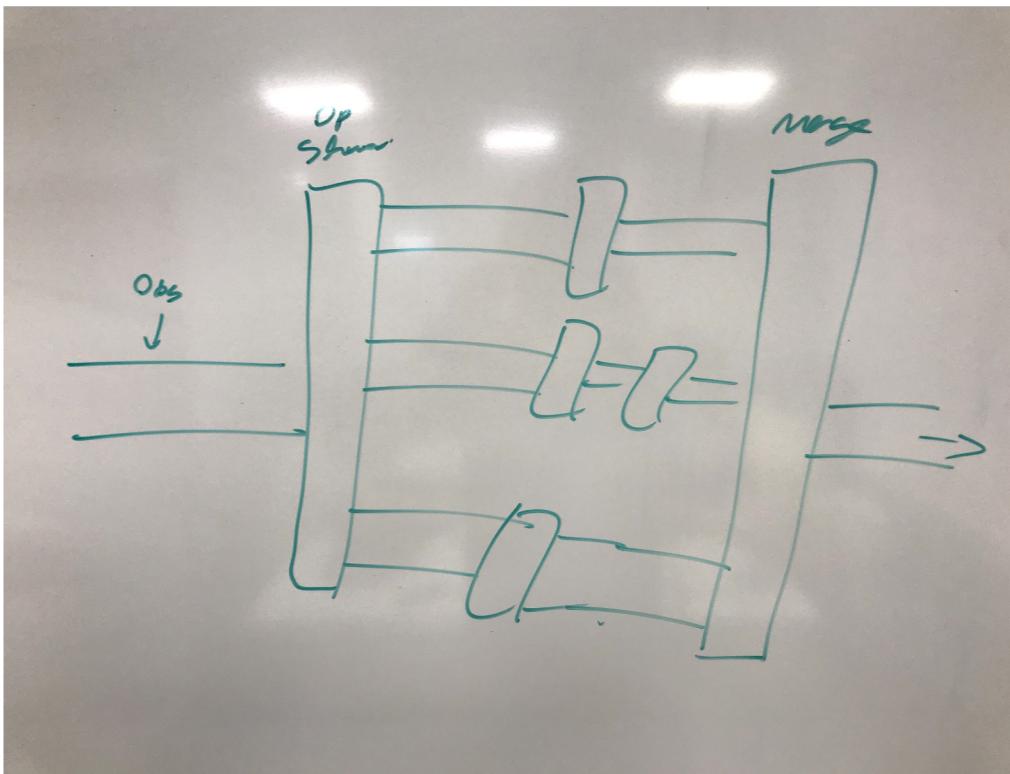
`switchMap` annule les précédentes requêtes HTTP en cours,
tandis que `mergeMap` les laisse toutes se terminer.

Exemple de cas d'utilisation du `switchMap` :

un champs de saisie semi-automatique dans lequel on souhaite ignorer tous les résultats issus de la saisie de l'utilisateur à l'exception du dernier.



Switch case de flux :



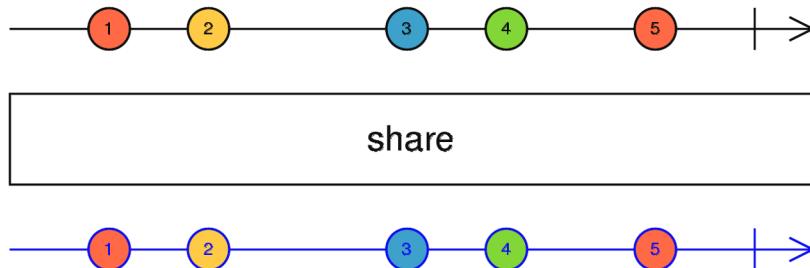
Operator : share

Renvoie un nouvel observable qui diffuse (**share**) l'observable d'origine.

Tant qu'il y aura au moins un abonné, cet observable émettra des données.

Lorsque tous les abonnés se seront désabonnés, il se désabonnera de la source Observable.

Parce que l'Observable est en multidiffusion, le flux devient **hot**.



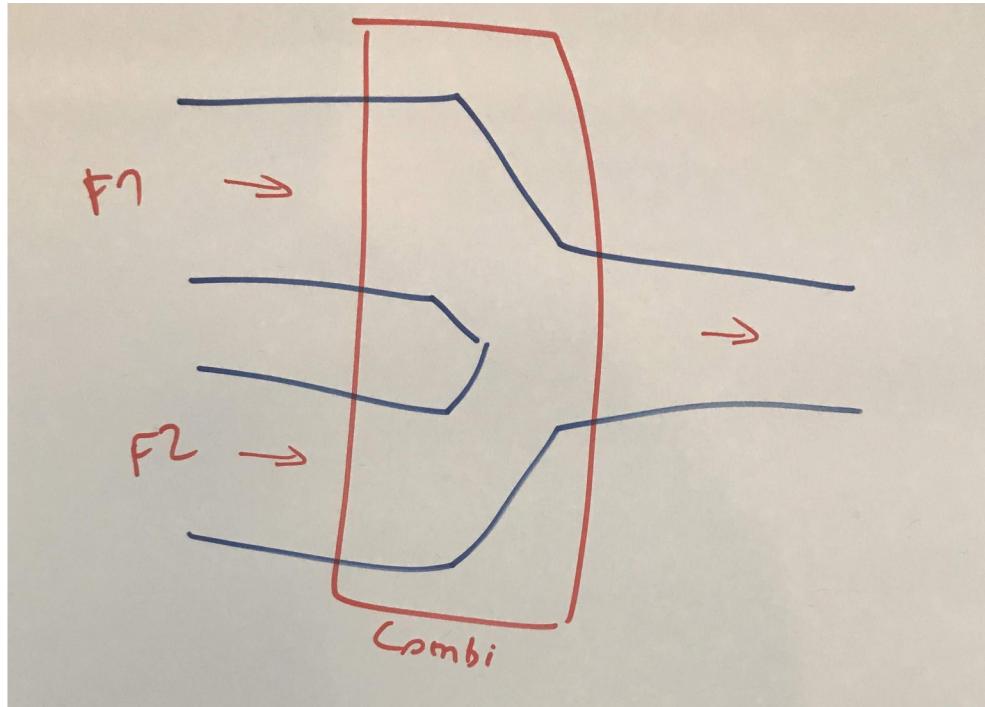
Opérateurs de combinaison

Il existe en RxJs 4 opérateurs pour combiner les données issues de plusieurs observables :

- `zip`
- `combineLatest`
- `withLatestFrom`
- `forkJoin`

1ère valeur

Les 4 opérateurs émettent leur première valeur lorsque tous les opérateurs d'entrée ont au moins émis une valeur.

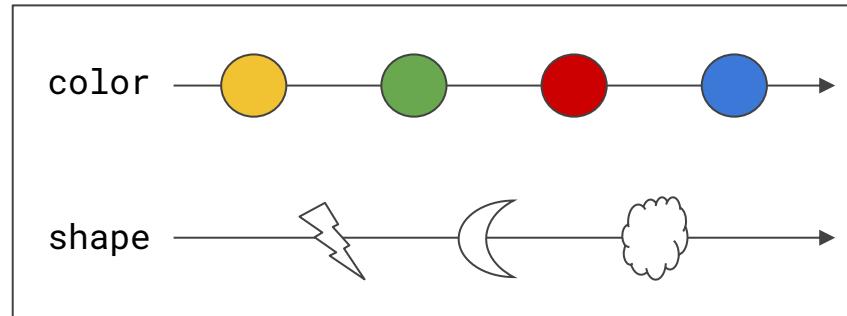


Combinaisons

Afin d'illustrer le fonctionnement de ces 4 opérateurs, nous allons observer leurs comportement à partir de cet exemple commun :

```
const colorSubject = new Subject<Color>();  
const shapeSubject = new Subject<Shape>();
```

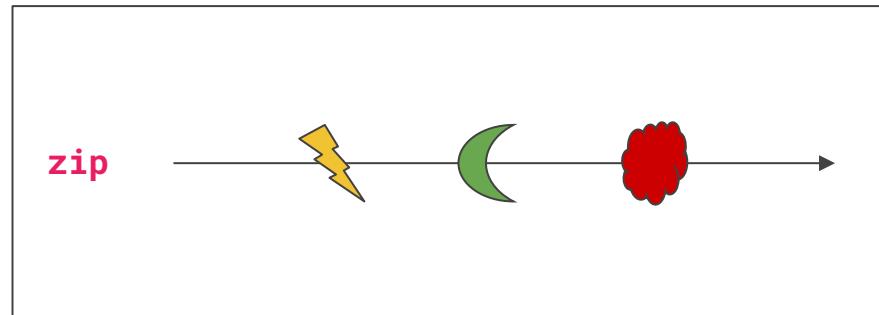
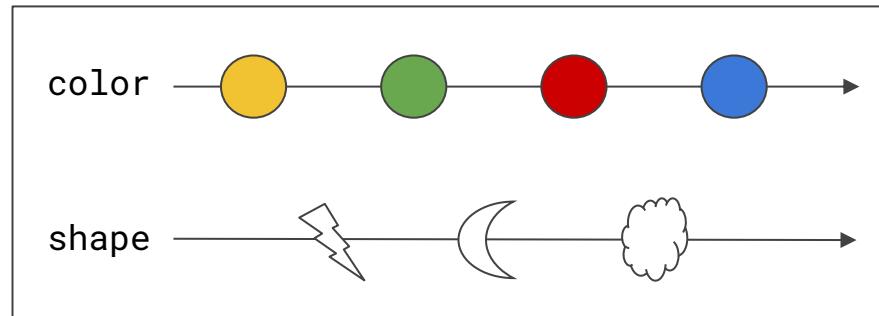
```
colorSubject.next(  );  
shapeSubject.next(  );  
colorSubject.next(  );  
shapeSubject.next(  );  
colorSubject.next(  );  
shapeSubject.next(  );  
colorSubject.next(  );
```



Operator : zip

L'opérateur **zip** combine plusieurs observables en générant une nouvelle valeur pour chaque ensemble de valeurs émises par l'ensemble des observables.

```
zip(colorSubject, shapeSubject)  
    .subscribe(([color, shape]) => { /* ... */ });
```



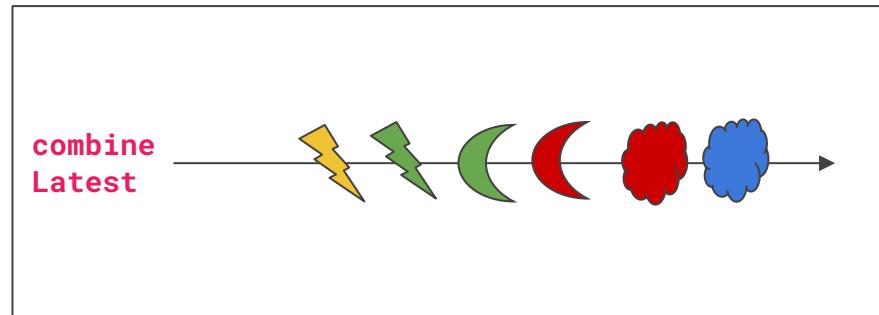
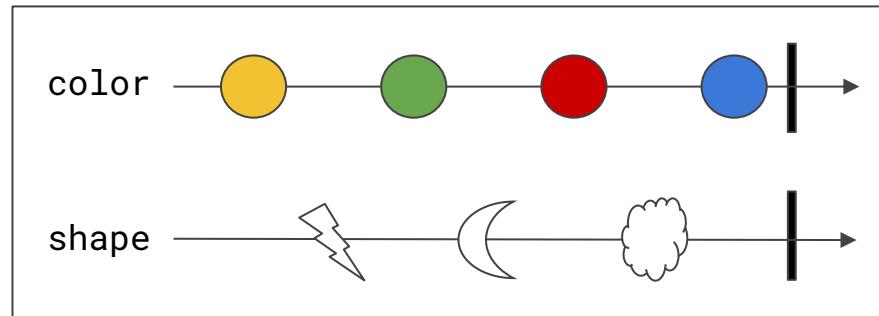
Operator : combineLatest

L'opérateur **combineLatest** combine plusieurs observables en générant une nouvelle valeur pour chaque nouvelle valeur émise par les observables en input.

```
combineLatest(colorSubject, shapeSubject)
    .subscribe(([color, shape]) => { /* ... */});
```

Cas d'utilisation :

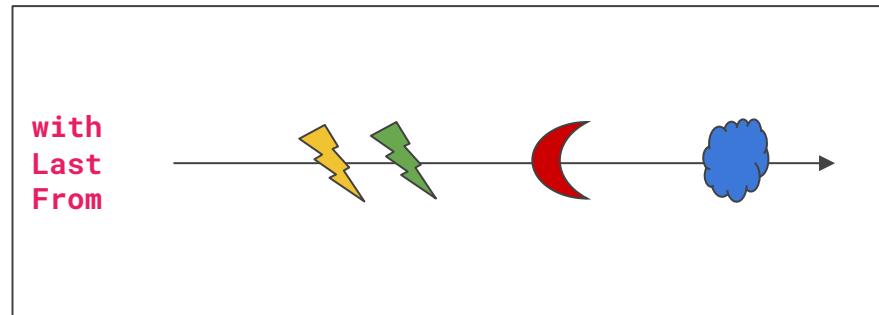
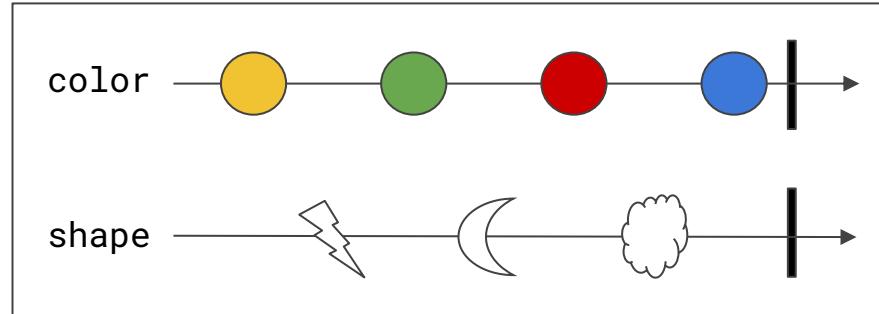
Combinaison de plusieurs inputs.valueChanges et déclenchement d'une action lors de la modification de l'un d'eux (exple: check de la cohérence entre CP et nom de ville)



Operator : withLastFrom

L'opérateur **withLastFrom** combine plusieurs observables en générant une nouvelle valeur pour chaque valeur émise par le 1er observable (master).

```
withLastFrom(colorSubject, shapeSubject)
    .subscribe(([color, shape]) => { /* ... */ });
```



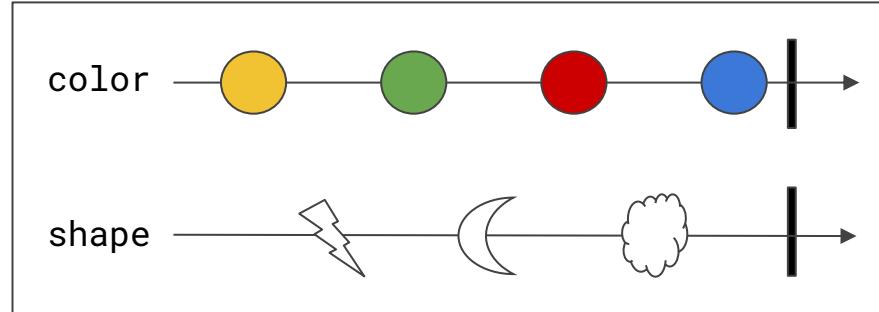
Operator : forkJoin

L'opérateur **forkJoin** combine plusieurs observables en générant une valeur lorsque tous les observables ont émis leurs signal de fin.

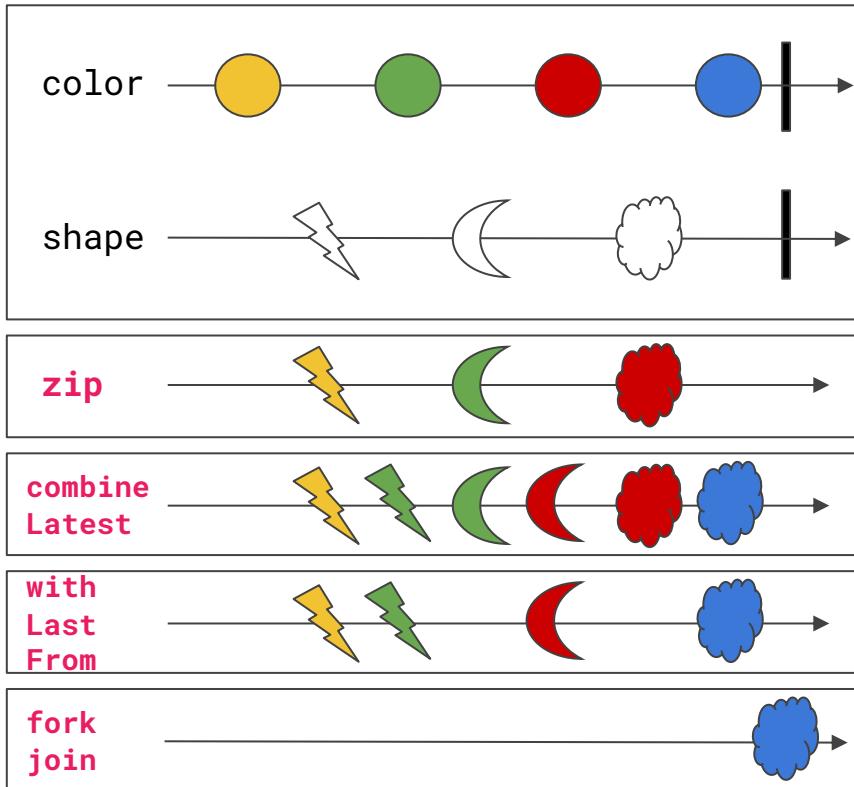
```
forkJoin(colorSubject, shapeSubject)
  .subscribe(([color, shape]) => { /* ... */ });
```

Cas d'utilisation classique :

Merge d'appels HTTP



Combinaisons : récap'



Mathematical Operators

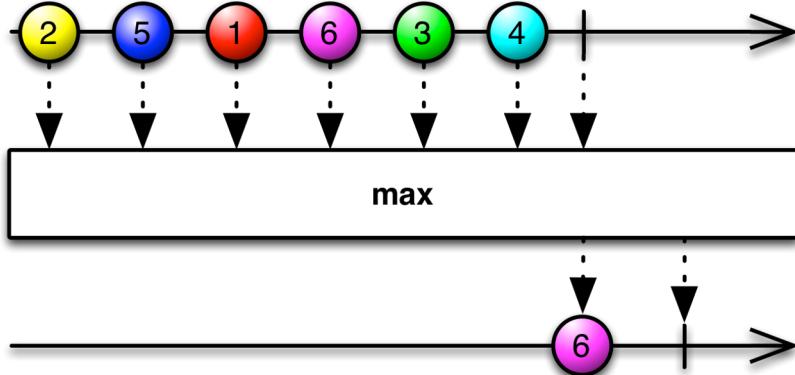
Count – compte le nombre d'éléments émis par la source Observable et n'émet que cette valeur

Max – détermine et émet l'élément à valeur maximale émis par un observable

Min – détermine et émet l'élément de valeur minimale émis par un observable

Sum – calcule la somme des nombres émis par un Observable et émet cette somme

Average – calcule la moyenne des nombres émis par un observable et émet cette moyenne

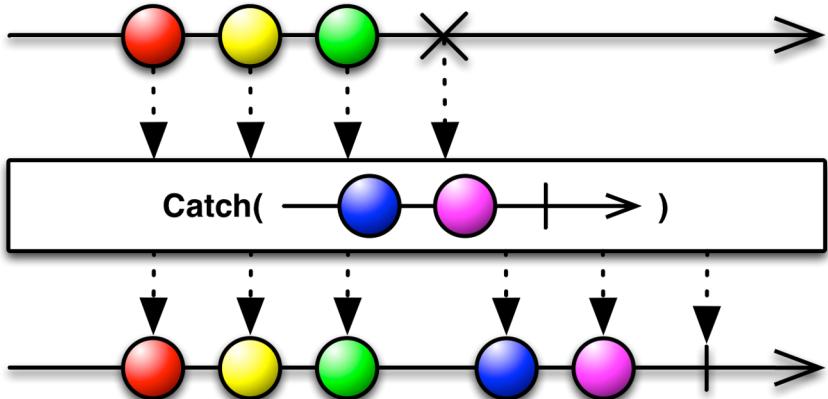


Operator : catchError

Les erreurs sont des évènements comme les autres.

```
v6  
import {from, throwError} from 'rxjs';
import {map, catchError} from 'rxjs/operators';

from(['a', 'b', 'c', 'd']).pipe(
  map((e) => {
    if (e.toUpperCase() === e) {
      throw new Error('already upper case !');
    }
    return e.toUpperCase();
}),
  catchError((e) => {
    // Faire qqch de cette erreur
    return throwError(e);
}),
  catchError((e) => {
    return of(e.message);
}),
).subscribe(e => {
  console.log(e); // ['A', 'B', 'already upper case !']
});
```



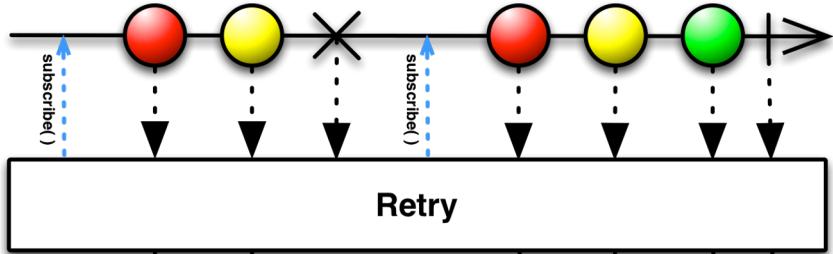
L'opérateur `catchError` prend un événement d'erreur en entrée et retourne un nouvel observable

L'observable retourné peut lui-même contenir un événement d'erreur grâce à `throwError`

Operator : retry

```
v6  
import {from, throwError} from 'rxjs';
import {map, catchError} from 'rxjs/operators';

from(fetch('https://unicorn-ng.com/unicorns')).pipe(
  retry(2),
).subscribe(e => {
  // ...
});
```



L'opérateur **retry** permet de rejouer un observable un nombre maximum de fois.

Ce nombre est défini par son unique paramètre: **count**.

Exercice - flux synchrone



Calculer l'**âge moyen** des licornes en utilisant la ressource '<http://localhost:3000/unicorns>'.

```
public getAverageAge(): Observable<number> {
  return this.http.get<Unicorn[]>('http://localhost:3000/unicorns').pipe(
    flatMap(e => e),
    pluck('birthyear'),
    map((birthyear: number): number => new Date().getFullYear() - birthyear),
    reduce((acc: any, age: number) => {
      return {count: acc.count + 1, sum: acc.sum + age};
    }, {count: 0, sum: 0}),
    map((ageCounter): number => ageCounter.count ? ageCounter.sum / ageCounter.count : 0)
  );
}
```

Exemple complet (v2)



Récupérer toutes les licornes, en utilisant la ressource '<http://localhost:3000/unicorns>'. Compléter la propriété `capacitiesLabels` de ces éléments grâce à l'utilisation de '<http://localhost:3000/capacities>'.

```
listWithCapacitiesLabels2(): Observable<Unicorn[]> {
    return forkJoin(
        this.list(),
        this.capacityService.list()
    ).pipe(
        map(([unics, capacites]): Unicorn[] =>
            unics.map((u: Unicorn): Unicorn => {
                return {
                    ...u, capacitiesLabels: u.capacities.map((c: number): string => {
                        return capacities.find((c2: Capacity) => c2.id === c).label;
                    })
                };
            });
        )
    );
}
```

Subjects

Subject

BehaviorSubject

ReplaySubject

AsyncSubject

RxJS : Subject

Un **Subject** :

- étend **Observable**, et peut être suivi par beaucoup d'**Observer**.
- est comme un **EventEmitter** : il conserve un registre de ses listeners.

```
import {Subject} from 'rxjs';

const subject = new Subject();
subject.subscribe((v) => console.log('observerA: ' + v));
subject.subscribe((v) => console.log('observerB: ' + v));

subject.next(1);
//CONSOLE: observerA: 1
//CONSOLE: observerB: 1

subject.next(2);
//CONSOLE: observerA: 2
//CONSOLE: observerB: 2
```



BehaviorSubject

Un **BehaviorSubject** est un Subject qui connaît son état actuel grâce à la méthode **getValue()**.

Un **BehaviorSubject** est utile pour représenter les valeurs "dans le temps".

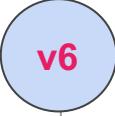
Par exemple, un flux d'événements d'anniversaires est un **Subject**, mais le flux de l'âge d'une personne serait un **BehaviorSubject**.

```
import {BehaviorSubject} from 'rxjs';

const behaviorSubject = new BehaviorSubject(0); // 0 is the initial value

behaviorSubject.next(1);
behaviorSubject.next(2);
behaviorSubject.next(3);

const actualValue = behaviorSubject.getValue();
console.log(actualValue);
//CONSOLE: 3
```



v6

BehaviorSubject

v6

```
import {BehaviorSubject} from 'rxjs';

const behaviorSubject = new BehaviorSubject(0); // 0 is the initial value

behaviorSubject.subscribe((v) => console.log('observerA: ' + v));
//CONSOLE: observerA: 0

behaviorSubject.next(1);
//CONSOLE: observerA: 1

behaviorSubject.next(2);
//CONSOLE: observerA: 2

behaviorSubject.subscribe((v) => console.log('observerB: ' + v));
//CONSOLE: observerB: 2

behaviorSubject.next(3);
//CONSOLE: observerA: 3
//CONSOLE: observerB: 3
```

ReplaySubject

Un **ReplaySubject** enregistre plusieurs valeurs venant d'un Observable et les répète aux nouveaux abonnés.

```
import {ReplaySubject} from 'rxjs';

const replaySubject = new ReplaySubject(3); // buffer 3 values for new subscribers

replaySubject.subscribe((v) => console.log('observerA: ' + v));
replaySubject.next(1);
//CONSOLE: observerA: 1
replaySubject.next(2);
//CONSOLE: observerA: 2
replaySubject.next(3);
//CONSOLE: observerA: 3
replaySubject.next(4);
//CONSOLE: observerA: 4

replaySubject.subscribe((v) => console.log('observerB: ' + v));
//CONSOLE: observerB: 2
//CONSOLE: observerB: 3
//CONSOLE: observerB: 4
replaySubject.next(5);
//CONSOLE: observerA: 5
//CONSOLE: observerB: 5
```

A diagram illustrating the behavior of a ReplaySubject. A light blue circle at the top contains the value 'v6'. Two lines descend from this circle to two separate circles below it. The left circle is light blue and contains the text 'observerA'. The right circle is light blue and contains the text 'observerB'.

v6

AsyncSubject

Un **AsyncSubject** représente le résultat d'une opération asynchrone.

La dernière valeur avant la notification **OnCompleted** ou l'erreur reçue via **OnError** est envoyée à tous les observateurs abonnés.

```
import {AsyncSubject} from 'rxjs';

const asyncSubject = new AsyncSubject();

asyncSubject.subscribe({ next: (v) => console.log('observerA: ' + v) });

asyncSubject.next(1);
asyncSubject.next(2);
asyncSubject.next(3);
asyncSubject.next(4);

asyncSubject.subscribe({ next: (v) => console.log('observerB: ' + v) });

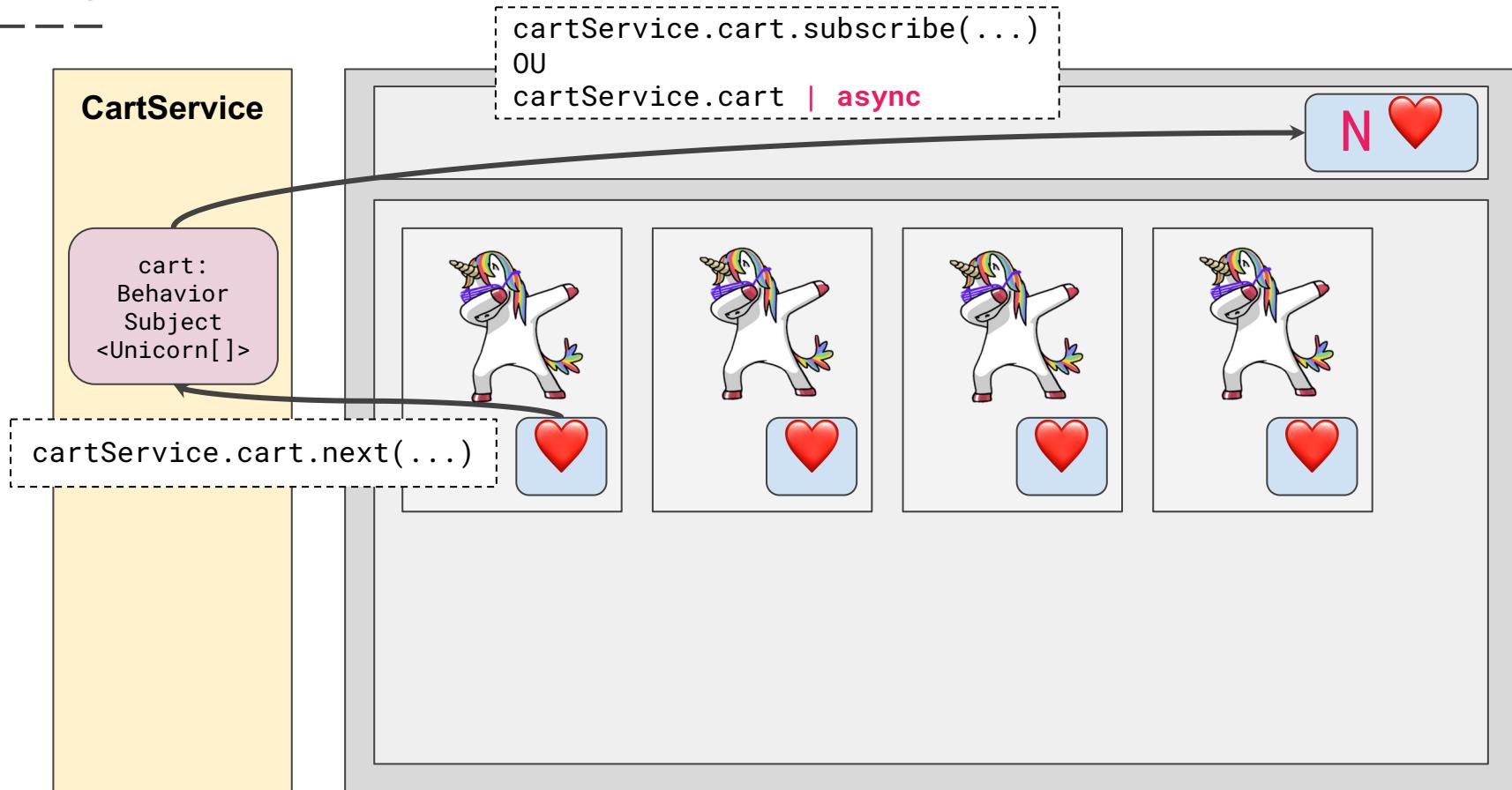
asyncSubject.next(5);
asyncSubject.complete();

//CONSOLE: observerA: 5
//CONSOLE: observerB: 5
```



v6

Subject & Service



Sources

- <http://reactivex.io/rxjs/manual/overview.html>
- <http://www.meanjs.fr/bien-demarrer-le-reactive-programming-en-general-et-rxjs-en-particulier/>
- <http://blog.ippon.fr/2016/11/16/rxjs/>
- <http://home.heere.com/tech-intro-programmation-reactive.html>
- <https://www.technologies-ebusiness.com/enjeux-et-tendances/rxjs-pour-les-humains>
- <https://antistatique.net/fr/nous/bloggons/2017/05/02/rxjs>
- <https://www.sitepoint.com/rxjs-functions-with-examples/>
- <http://rxmarbles.com/>
- <https://scotch.io/tutorials/rxjs-operators-for-dummies-forkjoin-zip-combinelatest-withlatestfrom>
- <https://itnext.io/polling-using-rxjs-b56cd3531815>
- <https://itnext.io/understanding-rxjs-operators-ea4bc93d56e>
- Café Craft : <https://overcast.fm/+KzBnKVfcE>

`@ngrx/store`



La gestion d'état - aka “State Management Pattern”

Flux (<https://facebook.github.io/flux/>)

Implémentation historique du pattern par Facebook en 2014

Redux (<https://redux.js.org/>)

Pattern né de Flux (simplifié : pas de dispatcher, un store unique)

Son implémentation pour **React** (react-redux) est devenu de facto un standard

Redux = Contraction de **Reducer** et de **Flux**

Vuex (<https://vuex.vuejs.org/>)

State Management Pattern pour **VueJS** inspiré de Flux et Redux.

NgRx (<https://github.com/ngrx/platform>)

State Management Pattern pour **Angular**

“State Management Pattern” : 3 implementations

- <https://github.com/ngxs/store> 2205★ github



- <https://github.com/ngrx/platform> 4796★ github

```
npm install @ngrx/store --save
```



- <https://github.com/angular-redux/store> 1348★ github



- <https://github.com/datorama/akita> 1306★ github
<https://netbasal.gitbook.io/akita/>

Akita



Akita : Une alternative à NGRX

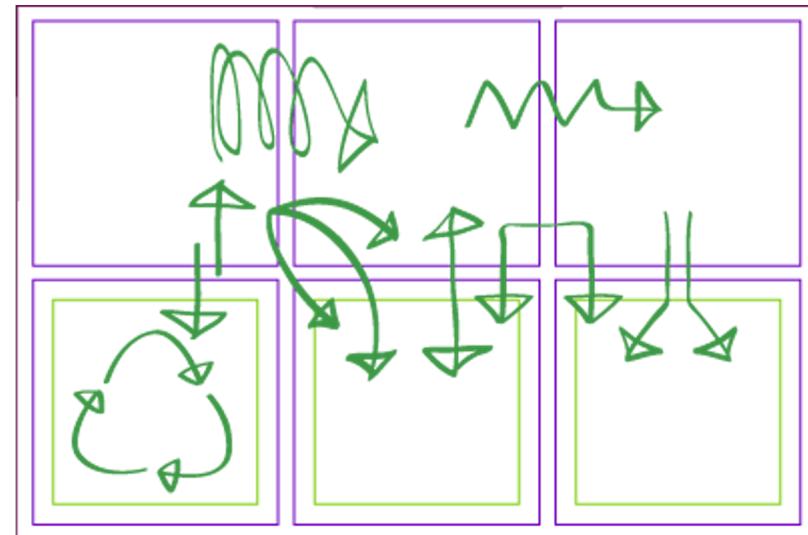
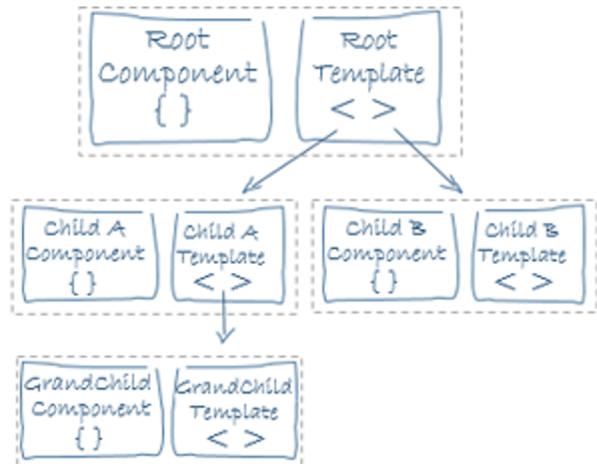
Akita



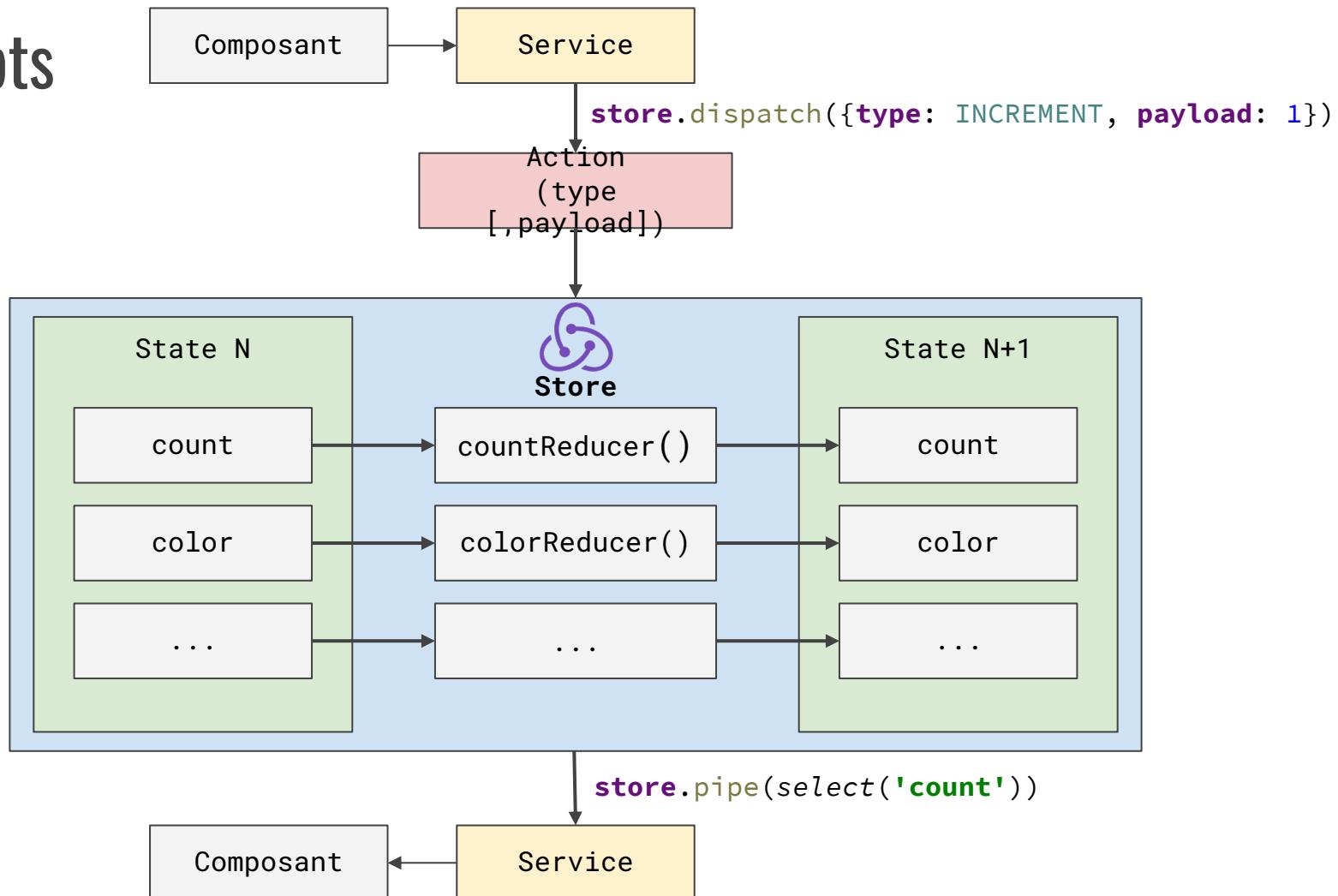
URL : <https://netbasal.gitbook.io/akita/>

Articles : <https://engineering.datorama.com/i-built-the-ngrx-demo-app-with-akita-heres-the-result-57f83fe92192>

Pourquoi NgRx ?



Concepts



Composant

La brique composant représente n'importe quel composant de votre application

```
// counter.component.ts
public count: Observable<number> = this.countService.count;

constructor(private countService: CountService) {
}

increment(value: number) {
    this.countService.increment(value);
}
```

```
// counter.component.html
<button (click)="increment(1)"> +1 </button>
<div>counter => {{ count | async }}</div>
```

Store

Store = ‘**single source of truth**’

Est un **singleton**

```
//app.module.ts

imports: [
  BrowserModule,
  StoreModule.forRoot({
    count: counterReducer,
    // ...
  }),
  // ...
],
```

Reducer

On crée 1 reducer par propriété dans le store

Un reducer est une fonction pure (mêmes paramètres -> même résultat)

Un reducer réagit à une action et mène à la production d'un nouvel état de l'application (state)

Un reducer gère des données **synchrones**

Bonnes pratiques :



- Garder le reducer aussi simple que possible
- Le reducer doit juste mettre à jour le **state** correspondant à sa donnée

Reducer : Exemple

```
export function counterReducer(countState: number = 0, action: CounterAction) {  
  
    console.log(action.type, countState);  
  
    switch (action.type) {  
        case INCREMENT:  
            return countState + action.value;  
        case DECREMENT:  
            return countState - action.value;  
        default:  
            return countState;  
    }  
}
```

Le reducer est **initialisé** (valeur métier) avec sa valeur par défaut (ici: **0**)

Action

Les actions sont des objets avec au minimum une propriété **type**

Les actions sont déclenchées par des composants à travers des services

Elles peuvent potentiellement porter de l'information (payload)

Lorsqu'elles sont déclenchées, elles sont traitées par **tous les reducers**

Bonnes pratiques :



- Utiliser une classe par action
- Toujours nommer le payload de l'action (ne pas l'appeler "payload")
- Définir toutes les actions associées dans un même fichier
(exple: count.actions.ts)

Action : Exemple

```
// count.actions.ts

import {Action} from '@ngrx/store';

export const INCREMENT = '[Counter] INCREMENT';
export const DECREMENT = '[Counter] DECREMENT';

export class IncrementCounter implements Action {
    readonly type = INCREMENT;
    constructor(public value: number) { }
}
export class DecrementCounter implements Action {
    readonly type = DECREMENT;
    constructor(public value: number) { }
}
export type CounterAction =
    IncrementCounter
    | DecrementCounter;
```

Service

```
// count.service.ts

import {Injectable} from '@angular/core';
import {Store, select} from '@ngrx/store';
import {IncrementCounter, DecrementCounter} from './store/counter.reducer';

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

  public count: Observable<number> = this.store.pipe(select('count'));

  constructor(private store: Store<AppState>) { }

  public increment(value: number) {
    this.store.dispatch(new IncrementCounter(value));
  }
  public decrement(value: number) {
    this.store.dispatch(new DecrementCounter(value));
  }
}
```

State

L'état de l'application (**state**) est défini par une **interface** qui possède autant de propriétés que le store n'a de reducer.

```
// src/app/store/app.state.ts
export interface AppState {
  count: number;
  color: string;
  // ...
}
```

Les états sont immutables (c.-à-d. que leurs valeurs ne changent pas après leurs créations). On peut illustrer cette propriété par la formule suivante :

newState = reducer(state, action)

Arborescence

```
app
|- store
  |- app.state.ts
  |- actions
    |- *.actions.ts
    |- ...
|- reducers
  |- *.reducer.ts
  |- ...
```

Devtool

```
npm install @ngrx/store-devtools --save-dev
```

```
// src/app/app.module.ts
import {StoreDevtoolsModule} from '@ngrx/store-devtools';

@NgModule({
  imports: [
    // ...
    StoreDevtoolsModule.instrument({
      maxAge: 10
    }),
  ]
})
```

Plugin Chrome “**redux-devtools**” :

<https://chrome.google.com/webstore/detail/redux-devtools/lmhkpmbekcpmknklioeibfkpmffibljd?hl=fr>

Devtool - prod ready

```
// src/app/app.module.ts
import {DevModule} from '../environments/environment';
@NgModule({
  imports: [
    // ...
    DevModule,
  ]
})
```

```
// src/environments/environment.ts
import {NgModule} from '@angular/core';
import {StoreDevtoolsModule} from '@ngrx/store-devtools';

export const environment = {
  production: false
};

@NgModule({
  imports: [StoreDevtoolsModule.instrument({
    maxAge: 10
  })],
})
export class DevModule {}
```

```
// src/environments/environment.prod.ts
import {NgModule} from '@angular/core';

export const environment = {
  production: true
};

// Laissez cette classe vide,
// pour que la compilation en prod
// fonctionne toujours
@NgModule({})
export class DevModule {}
```

Devtool

Permet de :

- voir les changements d'états
- annuler un changement d'état dans l'historique
- revenir à un état précédent
- exporter l'historique des changements d'états au format JSON
- importer un historique d'états

The screenshot shows the NgRx Store DevTools integrated into a browser's developer tools. The top navigation bar includes tabs for Elements, Console, Sources, Network, Performance, Memory, Application, Security, Audits, AdBlock, and Redux (which is currently selected). The left panel, titled 'Inspector', lists a history of state changes:

Action	State	Commit
@ngrx/store/init		7:08:09.67
[Counter] INCREMENT	+00:08.29	
[Counter] INCREMENT	+00:03.06	
[Counter] INCREMENT	+00:03.06	

Below this, there is a 'filter...' input field and a 'Commit' button. The right panel, titled 'NgRx Store DevTools', displays the 'Diff' tab. It shows a comparison between two states, with the first state being '4' and the second state being '6'. The 'Raw' tab is also visible. At the bottom, there are buttons for Dispatcher, Slider, Import, Export, Remote, and Settings.

Quelques liens sur le sujet

- <https://lainelouis.wordpress.com/2016/02/02/redux-explique-a-ma-mere/>
- <https://medium.com/@nioperas06/le-guide-pour-bien-d%C3%A9marrer-avec-ngrx-partie-1-7d18d3172269>
- <https://github.com/fausfore/ngrx-french-guide/>
- <https://github.com/ngrx/platform/blob/master/docs/store/README.md>
- <https://medium.com/city-pantry/handling-errors-in-ngrx-effects-a95d918490d9>
- Angular ngrx Redux Quick Start Tutorial : <https://youtu.be/f97IC0aekNU>
- <https://medium.com/@adrianfaciu/ngrx-tips-tricks-69feb20a42a7>
- <https://medium.com/supercharges-mobile-product-guide/angular-redux-the-lesson-weve-learned-for-you-93bc94391958>
- <https://itnext.io/ngrx-best-practices-for-enterprise-angular-applications-6f00bcd36d7>
- <https://medium.com/@andrew.kao/ngrx-complete-tutorial-without-pain-angular6-rxjs6-5511b8cb8dac>

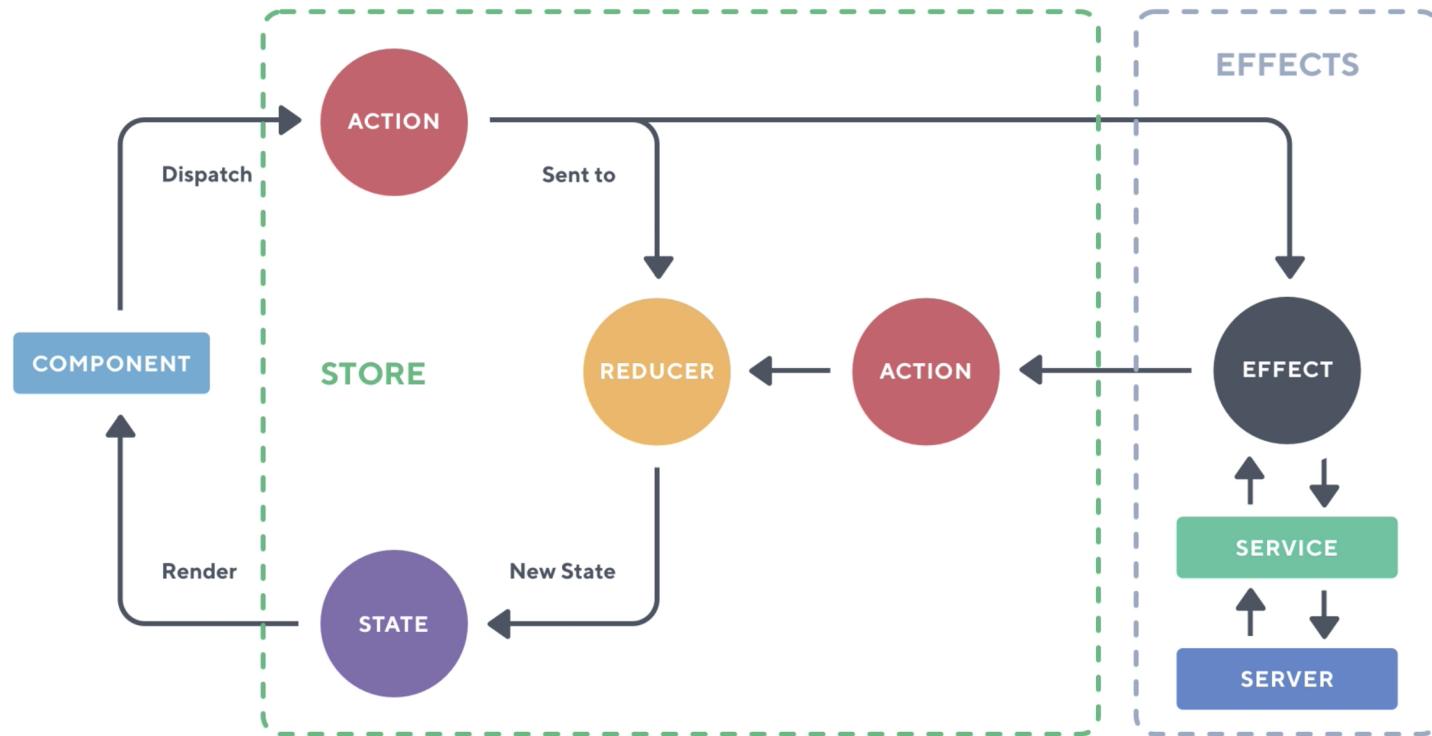
@Effect

Les side-effects gèrent ce qui est asynchrone
(car le reducer est toujours synchrone)

Un peu de lecture :

- <https://medium.com/@praveenpuglia/practical-ngrx-effects-tips-tricks-1935509c9fb6>
- <https://medium.com/@adrianfaciu/testing-ngrx-effects-3682cb5d760e>
- <https://github.com/ngrx/platform/blob/master/docs/effects/README.md>

@Effect 2/2



Ngrx : schematics

<https://github.com/ngrx/platform/blob/master/docs/schematics/README.md>

Install :

```
npm install @ngrx/schematics --save-dev  
ng config cli.defaultCollection @ngrx/schematics
```

Generate element :

```
npm run ng generate action User --spec  
npm run ng generate reducer ReducerName
```

Service HttpClient

Service HttpClient

- Angular fournit des services afin de communiquer via des requêtes HTTP
- Le module **HttpClientModule** a été introduit en **Angular 4.3**
- **HttpClientModule** remplace l'ancien module **HttpModule** d'**Angular** (non abordé ici)
- Il se trouve dans le package **@angular/common/http**
- Doit être importé après **BrowserModule** dans le module de l'application :

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';
@NgModule({
  imports: [
    BrowserModule,
    // ...
    HttpClientModule
  ],
})
export class AppModule { }
```

- Une fois **HttpClientModule** importé, vous pourrez injecter **HttpClient** dans **vos services**

Service HttpClient

- Basé sur le pattern **Observable** contrairement à AngularJS et ses Promises
- Plus grande flexibilité grâce aux différents opérateurs de **RxJS** : **retry**, ...
- **HttpClient** n'est pas imposé par **Angular**
 - on peut utiliser l'API **fetch** ou la lib AXIOS par exemple
- **Bonne pratique** :
 - implémenter les appels REST dans des **services**
 - ne jamais appeler la méthode **subscribe()** dans un service.

Méthodes

- Par défaut, le service **HttpClient** réalise des requêtes AJAX avec **XMLHttpRequest**
- Il propose plusieurs méthodes, correspondant au méthodes HTTP communes :

Méthode	Fonction
get	Récupération d'une (ou de plusieurs) ressource(s)
post	Création d'une ressource
put	Modification d'une ressource
delete	Suppression d'une ressource
patch	Modification partielle d'une ressource (peu utilisé)
head	Tester l'existence d'une ressource (code 200 / 404)
jsonp	(voir CORS)

HTTP - Utilisation dans un service

- Exemple simple d'un service utilisant **HttpClient**
 - Import d'**HttpClient** depuis le module `@angular/common/http`
 - Injection du service via le constructeur
- La méthode du service retourne un **Observable** du body de la réponse HTTP

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

@Injectable({
  providedIn: 'root'
})
export class ContactService {
  private baseUrl = 'https://.../v1/contacts';
  constructor(private http:HttpClient){ }
  getContacts(): Observable<Contact[]> {
    return this.http.get<Contact[]>(this.baseUrl);
  }
  getContact(contactId: string): Observable<Contact> {
    return this.http.get<Contact>(this.baseUrl, {
      params: new HttpParams()
        .set('contactId', contactId)
    });
  }
}
```

HTTP - Ajout de headers

Peut être utile pour certaines techniques d'authentification, comme JSON Web Token (aka **JWT**) :

```
const headers = { 'Authorization': `Bearer ${token}` };

http.get(`${this.baseUrl}/invoices`, { headers })
    .subscribe(invoices => {
        // retournera les factures visibles à l'utilisateur authentifié
        this.invoices = invoices;
    });
}
```

HTTP - Traitements sur l'Observable retourné

Méthodes : **retry**, **flatMap**, **filter**, **map**, **reduce**, ...

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

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

  baseUrl = 'https://.../v1/contacts';

  constructor(private http:HttpClient){ }

  getContacts(): Observable<string> {
    return this.http.get<Contact[]>(this.baseUrl).pipe(
      retry(2), // Testera 3 fois en cas d'erreur d'API
      flatMap(e => e), // Converti un tableau de Contact en un flux de Contact
      filter((contact: Contact) => contact.active === true), // Bloque les contacts inactifs
      map((contact: Contact) => {
        contact.name = contact.name.toUpperCase();
        return contact;
      }),
      toArray()
    )
  }
}
```

v6

Intercepteur HTTP

- Une des raisons de la réécriture du module `Http` est l'ajout des intercepteurs
- Les intercepteurs permettent d'intercepter **des requêtes ou des réponses**
- Par exemple, si l'on veut intercepter toutes les requêtes pour ajouter un header particulier à certaines d'entre elles, on peut écrire un intercepteur comme celui-ci :

```
@Injectable()
export class MyAPIInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    // Si c'est une requête vers mon API
    if (req.url.includes('my.api.com')) {
      // on ajoute notre token d'authentification
      const clone = req.clone({ setHeaders: { 'Authorization': `token ${OAUTH_TOKEN}` } });
      return next.handle(clone);
    }
    // Si ce n'est pas une requête à destination de notre API, on passe au handler suivant
    return next.handle(req);
  }
}
```

```
providers: [
  { provide: HTTP_INTERCEPTORS, useClass: MyAPIInterceptor, multi: true }
]
```

Intercepteur HTTP - Use cases

Les intercepteurs HTTP servent généralement à :

- Gérer des problématiques de **sécurité** (en positionnant des tokens de manière transparente par exemple)
- Faire apparaître un **spinner** tant que des requêtes HTTP sont en cours
- **Logger** les erreurs remontées par vos API



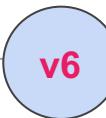
Intercepteur HTTP - Gestion des erreurs

Il est également **possible d'intercepter la réponse**, ce qui peut être pratique pour gérer les erreurs de façon générique :

```
@Injectable()
export class ErrorHandlerInterceptor implements HttpInterceptor {

    constructor(private router: Router, private errorHandler: ErrorHandler) {}

    intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
        return next.handle(req).pipe(
            // on catch les erreurs
            catchError((errorResponse: HttpErrorResponse) => {
                // si le status est Unauthorized
                if (errorResponse.status === 401) {
                    // on redirige vers la page de login
                    this.router.navigateByUrl('/login');
                } else {
                    // sinon, on notifie l'utilisateur
                    this.errorHandler.handleError(errorResponse);
                }
                return throwError(errorResponse);
            })
    );
}
```



CORS

- **CORS** : cross-origin resource sharing
- Mécanisme qui permet de charger des ressources depuis un **autre domaine**
- Deux pages ont la même origine si :
 - le **protocole** (http / https),
 - le **port** (si spécifié) et
 - l'**hôte** sont les mêmes pour les deux pages.
- Par défaut, l'**AJAX** est interdit vers des domaines différents
- 3 Solutions :
 - JSONP (solution 'old school' ne supportant que la méthode **GET**)
 - Configuration des serveurs HTTP (réécriture d'URLs)
 - Gestion des requêtes **OPTION** par le back proposant la ressource **REST**

Routeur

(1 matinée complète)



Router

- Prise en compte des différents cas d'utilisation : authentification, login, permissions, ...
- 3e implémentation depuis le début d'**Angular**

Router

- Router orienté **composant**
- Association d'un composant principal avec une URL de votre application
- Utilisation d'une méthode `RouterModule.forRoot` pour définir la configuration
- Utilisation de la directive `RouterOutlet` pour définir le point d'insertion
- Navigation entre les pages via la directive `RouterLink`
- Installation via :
 - **NPM** : `npm install --save @angular/router`

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

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

Router - forRoot

- Méthode permettant d'enregistrer de nouvelles routes
- Elle prend en paramètre un tableau de **RouterConfig**, qui correspond à un tableau de **Route**

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

export const routes: Routes = [
  { path: '', component: HomeComponent }, // path: '/'
  { path: 'contacts', component: ContactsComponent },
  { path: 'contact/:id', component: ContactComponent }
];

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

Router - Wildcard route

- Le routeur prend la 1ère route qui match avec le pattern du **path**
- On peut définir une **route par défaut** avec les wildcards "******"
- Permet généralement de définir une page en cas de mauvaise URL / route

```
const routes: Routes = [
  { path: '', component: HomeComponent }, // path: '/'
  { path: 'contacts', component: ContactsComponent },
  { path: 'contact/:id', component: ContactComponent },
  { path: '**', redirectTo: '' }
];
```

Router - RouterOutlet

- Directive à utiliser via l'élément **<router-outlet>**
- Permet de définir le point d'insertion dans le composant principal
- Le composant sera inséré en tant qu'enfant de l'élément sur lequel la directive **RouterOutlet** est utilisée
- Possibilité de définir la vue via un attribut **name** (utilisé pour définir plusieurs vues au même niveau)
- Possibilité de créer des vues enfant grâce à l'utilisation de **RouterOutlet** imbriquées

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

Router - RouterLink

- La directive **RouterLink** permet de naviguer d'une route à une autre
- **Alternative** : Utiliser la méthode **navigate** du service **Router** dans une méthode du composant. (routerLink l'utilise en interne)
- La directive **RouterLink** s'attend à un tableau de noms de routes, suivi par d'éventuels paramètres

```
<div>
  <h1>Hello {{message}}!</h1>
  <a [routerLink]=["/contacts"]>Link 1</a>
  <a [routerLink]=["/contact", 1]>Link 2</a>
  <router-outlet></router-outlet>
</div>
```

Router - Les liens href

- Utilisation via un attribut **routerLink**
- Configuration de la route désirée via ce même attribut **routerLink**
- Attribut **href** généré par le service **Location**
- Ajout de classes CSS si la route est active (directive **routerLinkActive**)

```
<li routerLinkActive="active" [routerLinkActiveOptions]="{exact:true}">
  <a routerLink="/">{{ 'tabs.home' | translate }}</a>
</li>
```

```
/* css file */
li {
  color: blue;
}
li.active {
  color: red;
}
```

Router - Récupération des paramètres d'URL (v1)

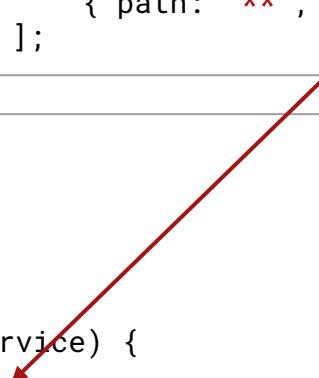
Utilisation du service **ActivatedRoute**

```
// app.component.html
<a [routerLink]=["'unicorn', 1]"></a>
```

```
const routes: Routes = [
  { path: '', component: HomeComponent }, // path: '/'
  { path: 'unicorns', component: UnicornsComponent },
  { path: 'unicorn/:id', component: UnicornComponent },
  { path: '**', redirectTo: '' }
];
```

```
@Component({ ... })
export class UnicornComponent {
  public unicorn: Unicorn;

  constructor(route: ActivatedRoute,
             unicornService: UnicornsService) {
    route.params.subscribe((params: Params) => {
      unicornService.get(params.id).subscribe(unicorn => this.unicorn = unicorn);
    });
  }
}
```



Router - Récupération des paramètres d'URL (v2)

Utilisation du service **ActivatedRoute**

```
// app.component.html
<a [routerLink]=["'unicorn', 1]"></a>
```

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

@Component({ ... })
export class UnicornComponent {

  public unicorn: Observable<Unicorn>;

  constructor(unicornService: UnicornsService,
             route: ActivatedRoute) {
    this.unicorn = route.params.pipe(
      mergeMap((params: Params) => unicornService.getById(params.id))
    );
  }
}
```

Router - Guards

Les guards sont des mécanismes qui permettent de sécuriser l'accès à une route. Il en existe 4 types :

CanActivate (et **CanActivateChild**) : Permet d'empêcher l'activation de la route. Il permet également de rediriger vers une autre route (comme une page d'erreur ou une page de connexion)

CanLoad : Permet d'empêcher le chargement du fichier JS du module applicatif concerné (lazy-loading) et l'accès à toutes ses routes filles. Ce guard ne peut être utilisé que sur les **routes ayant un attribut loadChildren**.

CanDeactivate : Utilisé pour empêcher de quitter la route actuelle. Peut être utile pour demander une confirmation avant de quitter une page.

Guards - Exemple

- Interface **CanActivate** : protège l'accès à une route
- Retourne un **boolean**, une **Promise<boolean>** ou un **Observable<boolean>**

```
import { Injectable } from '@angular/core';
import { CanActivate, Router, ActivatedRouteSnapshot } from '@angular/router';
import { AuthService } from '../shared/auth.service';

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

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
    if(this.authService.isLoggedIn()) {
      return true;
    }
    this.router.navigate(['/login']);
    return false;
  }
}
```

```
// fichier app/app-routing.module.ts
import { AdminComponent, AuthGuard } from './admin/';

export const AppRoutes: Routes = [
  {
    path: 'admin',
    component: AdminComponent,
    canActivate: [AuthGuard]
  }
];
```

Guards & cli

```
npx ng generate guard <guardName>  
npx ng g           g           <guardName>
```



Guards - Exercice



Créer un guard afin de ne permettre l'accès à la page unicorn-detail que si la licorne est née une année paire.

Dans le cas contraire, rediriger l'utilisateur vers la page unicorn-list

```
; 9F ; LAN9L=í  
F=PL~ ; LAN9L=<0GM_L=1F9HK@GL|  
KL9L=~ 0GM_L=J1L9L=1F9HK@GL| ~ - : K=JN9: D=¢: GGD=9F£ ñ
```

```
J=L MJF L@AK} MFA; GJF1=JNA; =} ?=L3FA; GJFí F=PL} H9J9EK} A<ì } HAHÍ  
HDM Cí à: AJL@=9Jàí |  
E9Hí : AJL@=9J~ FME: =Jí ü£ xí : AJL@=9J ™ Uí í |  
L9Hí ; 9F ; LAN9L=~ : GGD=9Fí ü£ ñ  
A> í x; 9F ; LAN9L=í ñ  
L@AK} JGML=J} F9NA?9L= Q3JDí î ààï ì Ä  
L@AK} KF9; C 9J} GH=Fí à▲ 4GMK F®à9N=R H9K D= <JGAL <= NGAJ ; =LL= H9?=} } } - 312 xàí Ä
```

ó

óí |

ó

Router - Resolver 1/2

- Mécanisme permettant de charger des données **avant** d'afficher la page
- Évite les incohérences lors du chargement d'une route

```
// contact.resolver.ts
import {ActivatedRouteSnapshot, Router, Resolve} from '@angular/router';

@Injectable({
  providedIn: 'root'
})
export class ContactResolver implements Resolve<Contact> {

  constructor(private contactService: ContactService,
              private router: Router) {}

  resolve(route: ActivatedRouteSnapshot): Observable<any> {
    const contactId = route.params['contactId'];
    return this.contactService.getContact(contactId).pipe(
      catchError(() => this.router.navigate(['/error']))
    );
  }
}
```

Router - Resolver 2/2

Le **resolver** doit être configuré dans la route

```
// app/app.routes.ts
export const AppRoutes: Routes = [
  {
    path: 'contact/:contactId',
    component: ContactComponent,
    resolve: {
      contact: ContactResolver
    }
  }
];
```

Dans le composant utilisant le **resolve**, la propriété **data** du **ActivatedRoute** contient le contenu des ressources

```
// contact.component.ts
constructor(private route: ActivatedRoute) {
  this.route.data.subscribe((data: Data) => {
    this.contact = data.contact;
  });
}
```

ici, **data.contact** correspond à la propriété contact définie dans la route

Router - Stratégies pour la génération des URLs

- **PathLocationStrategy** (stratégie par défaut)
 - Nécessite la définition de l'URL de base de votre application (**APP_BASE_HREF** ou **<base>**)

```
router.navigate(['contacts']); //example.com/app-ng/contacts
```

- **HashLocationStrategy**

```
router.navigate(['contacts']); //example.com#/contacts
```

- Possibilité de configurer l'implémentation à utiliser

```
import {HashLocationStrategy, LocationStrategy } from '@angular/common';

@NgModule({
  providers: [{ provide: LocationStrategy, useClass: HashLocationStrategy }]
})
export class AppModule { }
```

Router - Configuration de l'URL de base de l'application

- Définition d'un nouveau **provider** pour la constante **APP_BASE_HREF**
- Il sera utilisé lors de la génération des différentes URLs

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

@NgModule({
  providers: [{ provide: APP_BASE_HREF, useValue: '/my-app' }]
})
export class AppModule { }
```

- Possibilité de définir le **APP_BASE_HREF** lors du build :

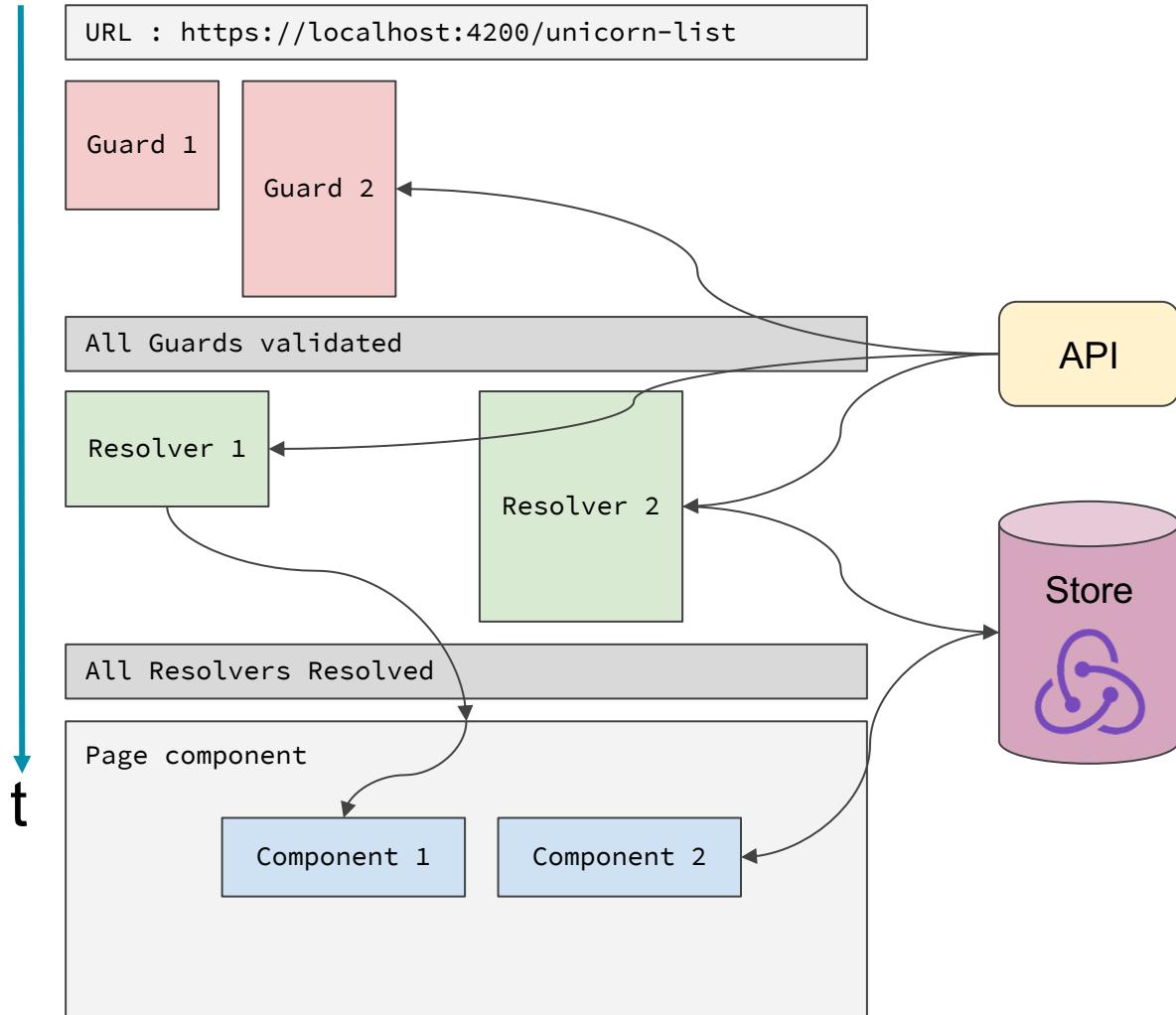
```
"scripts": {
  "build": "npm run lint && npm run ng -- build --prod --base-href /my-app/",
},
```



Guards, Resolvers & @ngrx

Lorsqu'un resolver charge une donnée gérée par le store, son action peut se limiter à mettre à jour le store avec les données de l'API (**resolver 2**).

Toutes les données n'étant pas gérées dans le store, le resolver peut également envoyer au composant de page les données reçues de l'API (**resolver 1**).



Passage d'infos entre guards et resolvers

```
@Injectable({
  providedIn: 'root'
})
export class OlderThanTenGuard implements CanActivate {
  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean> {
  return this.serviceUnicorn.getOne(next.params.id).pipe(
    tap((unicorn) => {
      next.data = {...next.data, unicorn};
    }),
    map((u: Unicorn) => u.age > 10)
  );
}
```

Concepts avancés

Configuration externalisée (properties.json)

```
// app.module.ts
import {APP_INITIALIZER, NgModule} from '@angular/core';
@NgModule({
    providers: [
        {
            provide: APP_INITIALIZER,
            useFactory: appInitializerFactory,
            deps: [ConfigService],
            multi: true
        }, // ...
    ]
})
export class AppModule {
}
export function appInitializerFactory(configService: ConfigService) {
    return configService.load();
}
```

```
// config.service.ts
@Injectable()
export class ConfigService {
    public config: any;
    load(): Promise<any> {
        return this.http.get<any>('resources/properties.json', {
            headers: new HttpHeaders().set('Cache-Control', 'no-cache')
        })
        .toPromise()
        .then(data => {
            this.config = data;
            return data;
        });
    }
}
```

Internationalisation : Angular I18N vs @ngx-translate

Why ngx-translate exists if we already have built-in Angular2+ i18n ?



Article sur le sujet :

<http://byteclub.fr/blog/angular-i18n.html>



ocombe commented on 3 Apr 2017 • edited ▾

Collaborator



Hello,

this is a good question, and as I am working on i18n in the Angular core team I am probably the best to answer this.

The idea behind this lib has always been to provide support for i18n until Angular catches up, after that this lib will probably be deprecated. For now, there are still a few differences between Angular i18n and this library:

- Angular only works with one language at a time, you have to completely reload the application to change the lang. The JIT support only means that it works with JIT, but you still have to provide the translations at bootstrap because it will replace the text in your templates during the compilation whereas this lib uses bindings, which means that you can change the translations at any time. The downside is that bindings take memory, so the Angular way is more performant. But if you use `OnPush` for your components you will probably never notice the difference
- Angular only supports using i18n in your templates for now, I'm working on the feature that will allow you to use it in your code, but it's still a work in progress. This lib works both in code and templates
- Angular supports either XLIFF or XMB (both are XML formats), whereas this lib supports JSON by default but you can write your own loader to support any format that you want (there's a loader for PO files for example)
- Angular supports ICU expressions (plurals and select), but this library doesn't
- Angular supports html placeholders including angular code, whereas this library only supports regular html (because it's executed at runtime, and not during compilation, and there is no `$compile` in Angular like there was in AngularJS)
- The API of this library is more complete because it is executed at runtime it can offer more things (observables, events, ...) which Angular doesn't have (but doesn't really need given that you can not change the translations)



146



11



9

I18N - @ngx-translate



ngx-translate

URL : <http://www.ngx-translate.com/>

Librairie de gestion de l'internationalisation (**I18N**)

Charge les traductions (*.json) en http avec **HttpClient**

Permet de changer de langue à chaud facilement

Doc : <https://github.com/ngx-translate/core>

```
npm install @ngx-translate/core @ngx-translate/http-loader
```

@ngx-translate : configuration 1/2

```
import {TranslateLoader, TranslateModule} from '@ngx-translate/core';
import {TranslateHttpLoader} from '@ngx-translate/http-loader';
import {HttpClient, HttpClientModule} from '@angular/common/http';

@NgModule({
    //...
    imports: [
        //...
        TranslateModule.forRoot({
            loader: {
                provide: TranslateLoader,
                useFactory: translateFactory,
                deps: [HttpClient]
            }
        }),
        HttpClientModule,
        //...
    ]
})
//...
export function translateFactory(http: HttpClient) {
    return new TranslateHttpLoader(http, 'assets/i18n/', '.json');
}
```

@ngx-translate : configuration 2/2

- Le plugin @ngx-translate nécessite de définir la locale à utiliser à l'initialisation ainsi que la locale par défaut.
- La locale par défaut sera utilisée si un libellé est absent de la locale sélectionnée.
- Cette configuration peut être faite lors de l'initialisation de l'application

```
// app.module.ts
import {TranslateService} from '@ngx-translate/core';
import {APP_INITIALIZER, LOCALE_ID} from '@angular/core';

export function getLocale(): string {
    // we can search on localstorage or elsewhere...
    return 'fr';
}

export function appInitializer(
    translateService: TranslateService) {
    return () => {
        translateService.setDefaultLang('fr');
        translateService.use(getLocale());
    });
}
```

```
// app.module.ts
@NgModule({
    //...
    providers: [
        { provide: LOCALE_ID, useFactory: getLocale },
        { provide: APP_INITIALIZER,
            useFactory: appInitializer, multi: true,
            deps: [TranslateService] }
    ],
    //...
})
```

@ngx-translate : Utilisation

```
// fr.json
{
  "actions": {
    "confirm": "Valider",
    "cancel": "Annuler"
  },
  "header": {
    "title": "Ma super app' de licorne !"
  }
}
```

```
// en.json
{
  "actions": {
    "confirm": "Confirm",
    "cancel": "Cancel"
  },
  "header": {
    "title": "My super unicorn app' !"
  }
}
```

```
// unicorn.component.html
<h1>{{ 'header.title' | translate }}</h1>
<button translate>actions.confirm</button>
```

La fonction `translateService.use('xx')` permet de changer la langue de l'application en allant chercher le fichier de traduction correspondant

Animations

```
@Component({
  animations: [
    trigger('enterAnimation', [
      state('inactive', style({
        height: 0,
        display: 'none',
        'background-color': 'inherit'
      })),
      state('active', style({
        height: '100vh',
        display: 'block',
        'background-color': 'rgba(255,255,255,0.97)'
      })),
      transition('inactive => active', animate('300ms ease-in')),
      transition('active => inactive', animate('300ms ease-out'))
    ])
  ],
  selector: 'lsa-cross-selling',
  templateUrl: './cross-selling.component.html',
  styleUrls: ['./cross-selling.component.scss']
})
export class CrossSellngComponent {
  crossSellngState = 'inactive';

  constructor() {}
  displayCrossSellng() { this.crossSellngState = 'active'; }
  hideCrossSellng() { this.crossSellngState = 'inactive'; }
}
```

```
<div [@enterAnimation]="'crossSellngState">
  ...
</div>
```



AOT vs JIT

JIT : Just in time -> compile l'application dans le navigateur au runtime

AOT : Ahead-of-Time -> compile l'application lors du build

AOT : Mode par défaut de compilation d'angular-cli (depuis la v5)

Le compilateur AOT (Angular Ahead-of-Time) convertit votre code HTML et TypeScript en code JavaScript efficace pendant la phase de construction avant que le navigateur télécharge et exécute ce code.

Avantages de la compilation AOT :

- Rendu plus rapide
- Moins de demandes asynchrones
- Taille des fichiers réduite
- Déetecter les erreurs de modèle plus tôt
- Meilleur sécurité



Cache strategy

Cache Angular (avec annotation) : <https://blog.usejournal.com/7d7ecea470d2>



Cache HTTP (général) : <https://developer.mozilla.org/fr/docs/Web/HTTP/Cache>

Cache-Control : <https://developer.mozilla.org/fr/docs/Web/HTTP/Headers/Cache-Control>

ETag : <https://developer.mozilla.org/fr/docs/Web/HTTP/Headers/ETag>

Documentation & compodoc



URL : <https://compoDocumentation.github.io/compoDocumentation/>

Ajout au projet :

```
npm install --save-dev @compoDocumentation/compoDocumentation
```

```
// package.json
"scripts": {
    // ...
    "compoDocumentation": "./node_modules/.bin/compoDocumentation -p src/tsconfig.app.json"
}
```

```
/** 
 * Performs a very special operation
 *
 * @example
 * Simply call it with 2 values
 * getResult(1, 2)
 *
 * @param {number} a First number
 * @param {number} b Second number
 * @returns The sum of a and b
 */
getResult(a: number, b: number) : number {
    let sum = a + b;
    this.storedValue = sum;
    return sum;
}
```

Build optimisation : source-map-explorer

Install Source Map Explorer :

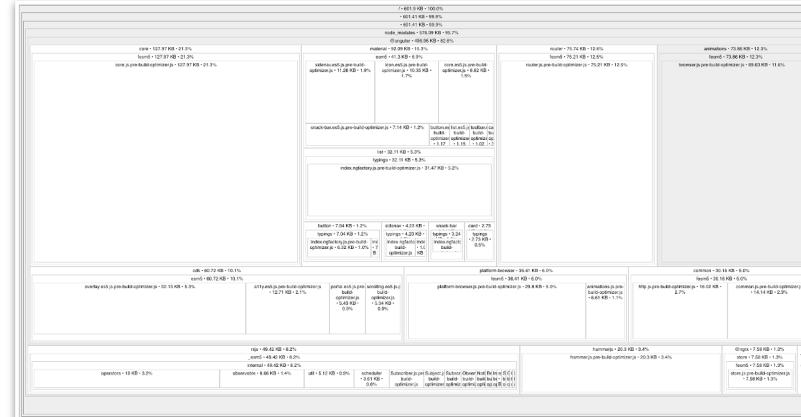
```
> npm install -g source-map-explorer
```

Build with source maps :

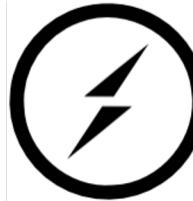
```
> npm run ng -- build --prod --source-map
```

Inspect your bundles :

```
> source-map-explorer ./dist/unicorn-ng/main*.js
```



Angular & Socket.io



socket.io

Une manière simple d'interagir avec un websocket est de le convertir en **Subject**.

Exemple d'implémentation avec une socket de **socket.io** :

npm install --save socket.io-client

```
// import * as io from 'socket.io-client';

private counterSocket: Subject<number>;

public getCounterSocket(): Subject<number> {
    if (!this.counterSocket) {
        const socket = io('http://localhost:3100/count');
        const observable = Observable.create((subscriber: Subscriber<number>) => {
            socket.on('count', (data: number) => {
                console.log(`Received data ${data} from server`);
                subscriber.next(data);
            });
            return () => socket.disconnect();
        });
        const observer = {
            next: (data: Object) => socket.emit('message', JSON.stringify(data)),
        };
        this.counterSocket = Subject.create(observer, observable);
    }
    return this.counterSocket;
}
```



Zone & Change Detection

Contexte

Les frameworks JS fonctionnent sur la même macro-structure et répondent aux critères suivants :

- Faciliter les interactions avec les événements applicatifs
- Mettre à jour l'état de l'application
- Rafraîchir le DOM

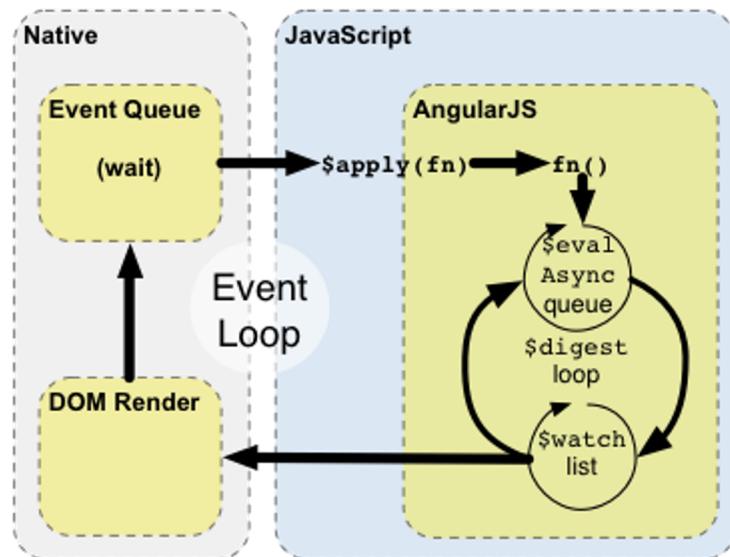
Cependant, chaque framework à une approche différente pour y arriver.

Change detection & AngularJS

En AngularJS, la détection des changements se fait à travers l'utilisation des directives ou services :

- ng-click
- ng-model
- \$http
- \$timeout

(ou utiliser `$scope.$apply()` pour forcer le mécanisme de détection dans le cas de librairies externes par exemple)



Change detection & Angular : Zone

Angular utilise Zone pour déclencher l'action de d'identifier un changement

Projet développé par l'équipe d'Angular

N'impose plus l'utilisation de services propres au framework (\$http, ...)

URL projet : <https://github.com/angular/zone.js>

Présentation par Brian Ford (team Google) :

https://www.youtube.com/watch?v=3IqtmUscE_U

Async Javascript

// Code

```
a();  
setTimeout(b, 0);  
setTimeout(c, 0);  
d();
```

// Run order

```
a  
d  
b (async)  
c (async)
```

Et si on voulait savoir combien de temps ça prend ?

// Code

```
start();  
a();  
setTimeout(b, 0);  
setTimeout(c, 0);  
d();  
stop();
```

// Run order

```
// start  
a  
d  
// stop  
b // missed!  
c // missed!
```

Ca ne fonctionne pas !

Async Javascript avec Zone 1/3

t



a();
d();

b();

c();

Async Javascript avec Zone 2/3

t



```
start();  
a();  
d();  
stop();
```

```
start();  
b();  
stop();
```

```
start();  
c();  
stop();
```

Async Javascript avec Zone 3/3

```
onZoneEnter();  
a();  
d();  
onZoneLeave();
```

```
onZoneEnter();  
b();  
onZoneLeave();
```

```
onZoneEnter();  
c();  
onZoneLeave();
```

t



Permet ainsi de tester, à chaque sortie de zone, que le modèle est identique à celui enregistré en entrée de zone.

Dans le cas contraire, la vue est mise à jours.

Angular & Performances





- MAJ Version
- Change detector
- track by *ngFor
- debounce / fields
- cache PWA / app shell
- lazy loading
- pipe async (if possible)
- unsubscribe onDestroy
- IE 11 specific
- images sizes
- pipes pure (in caches)
- call methods on templates
- budgets (bundle sizes)
- Virtual scroll ?

Maintenez votre version d'Angular à jour

Tuto officiel :

<https://update.angular.io/>

Généralement, cela se limite à la commande :

```
npx ng update @angular/core @angular/cli @angular/material --next
```

Change detector

Le but du **Change detector** est de détecter qu'un changement à eu lieu dans l'état de l'application (variable du modèle) et de le rendre visible sur l'interface.

~ Pascal Precht

Change Detector

Le modèle peut être modifié par 3 choses :

1. **Events** - click, submit, hover, ...
2. **XHR** - Récupération de données depuis un serveur distant
3. **Timers** - setTimeout(), setInterval()

Les 3 sources d'altération de l'état étant toutes **asynchrones**,

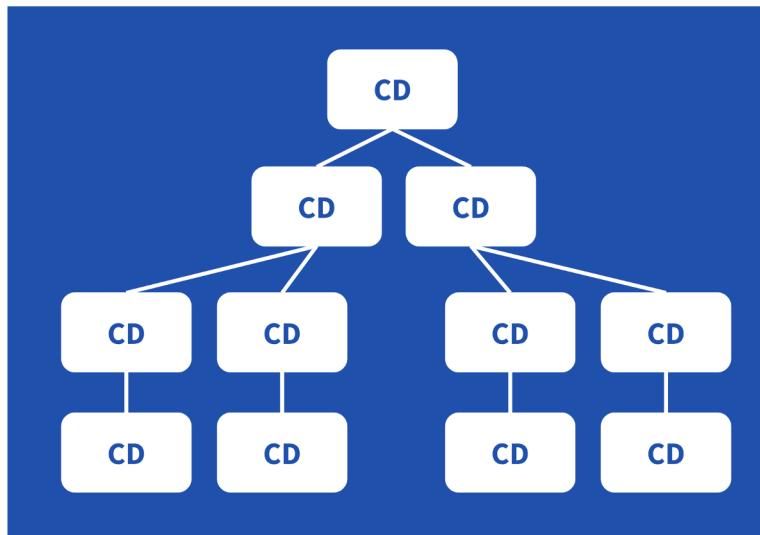
nous pouvons déduire que, chaque fois qu'une opération asynchrone est effectuée, l'état de notre application **peut** avoir changé.

C'est à ce moment que ~~quelqu'un~~ doit dire à Angular de mettre à jour la vue.

ZoneJS

1 Component = 1 Change Detector

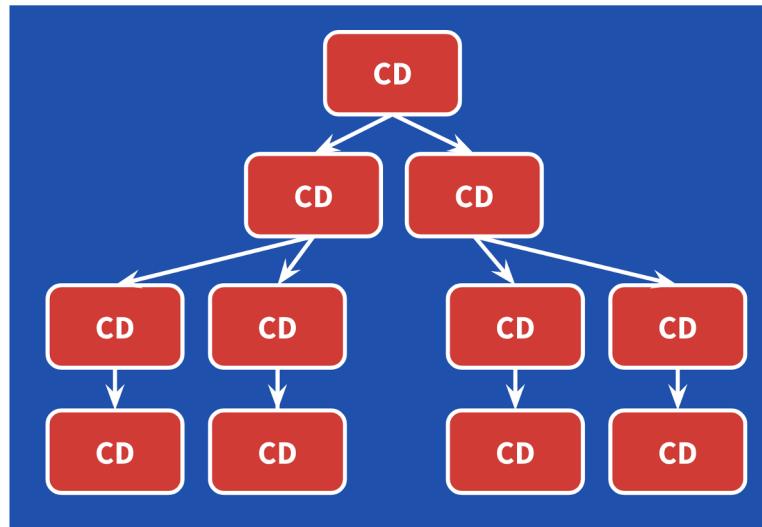
Chaque composant possède son propre détecteur de changement



Change Detectors Tree

Arborescence de composants => Arborescence de Change Detectors

La détection de changement à toujours lieu depuis la racine et se propage jusqu'aux feuilles.



Expression has changed after it was checked.

La détection de changement devient stable après un seul passage.

Si l'un de nos composants provoque des effets secondaires supplémentaires après la première analyse, Angular génère une erreur.

Le fameux: '**Expression has changed after it was checked.**'

NB : Ce check ne se produit qu'en développement.

L'idée étant d'éviter un décalage entre le modèle et la vue en production.



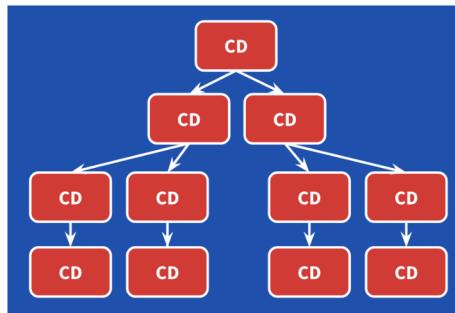
ChangeDetectionStrategy.Default

Chaque composant a sa propre ChangeDetectionStrategy.

Il existe 2 stratégies :

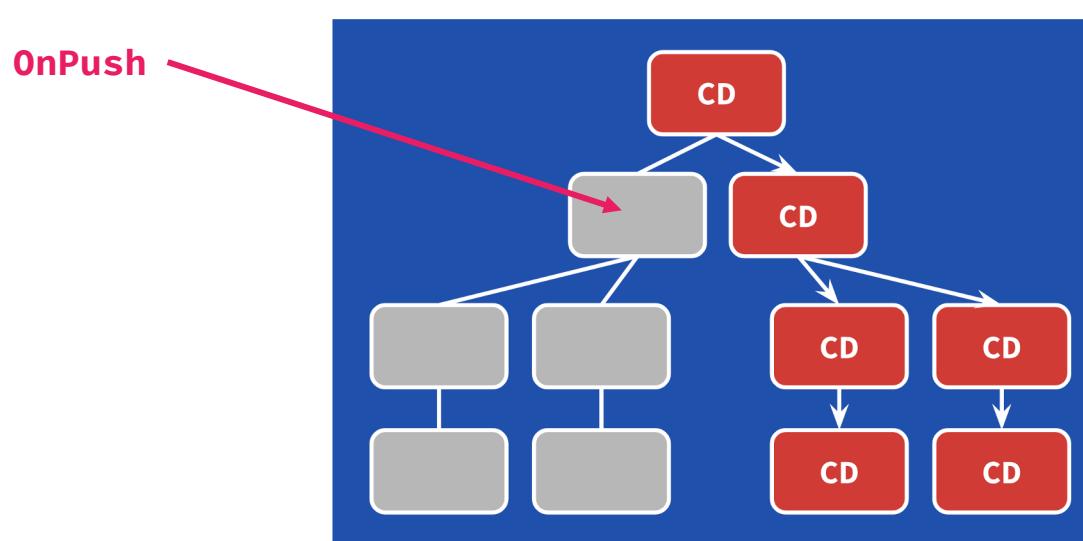
- Default
- OnPush

En mode **Default**, à chaque évènement asynchrone (clic, submit, XHR response, setTimeout, ...) -> Angular lance une action de change detection à partir du noeud root sur toute l'arborescence des composants



ChangeDetectionStrategy.OnPush

Passer un composant en stratégie **OnPush** permet d'ignorer la détection des changements pour son change detector et son sous arbre associé.



ChangeDetectionStrategy.OnPush

En mode **OnPush**,

on ne met à jour le composant automatiquement que si l'un des **@Input** a changé.



Il faut que la référence de l'objet en input aie changée (immutabilité)



Pour faciliter la mise en place il vaut mieux partir des feuilles

```
@Component({
  selector: 'app-po-and-quantities',
  templateUrl: './po-and-quantities.component.html',
  styleUrls: ['./po-and-quantities.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class PoAndQuantitiesComponent implements OnInit, OnChanges {
  ...
}
```

ChangeDetectionStrategy.OnPush : les Observables

Attention : La référence des Observables ne change jamais.

Si un composant à un Observable en **@Input** et qu'il est en stratégie OnPush, il va falloir indiquer au ChangeDetector que la vue doit être checkée.

Solution : ChangeDetectorRef & **markForCheck()**

```
● ● ●

constructor(private ordersValidationService: OrdersValidationService,
            private translateService: TranslateService,
            private cd: ChangeDetectorRef) {
}

private fetchSupports() {
    this.ordersValidationService.fetchPriceTicketTypes().subscribe(supports => {
        this.priceTickets = supports;
        this.cd.detectChanges();
    });
}
```

Change Detection : `detach()`

Peu importe la stratégie adoptée, il est possible de réaliser un `detach()` afin de ne plus être relié au mécanisme de détection des changement (même via les `@Inputs`)

Après un `detach`, il est toujours possible de détecter manuellement les changements (`detectChanges()`) ou de se rattacher (`reatach()`)

```
constructor(private cd: ChangeDetectorRef) {  
}  
  
ngAfterViewInit() {  
    this.cd.detach();  
}  
  
update() {  
    this.cd.detectChanges();  
}
```

Quand et quoi ?

Changement de haute fréquence :

(exple: récupération de données boursières live)

- `detach()`
- rafraîchissement dans un timer (ttes les X secondes)

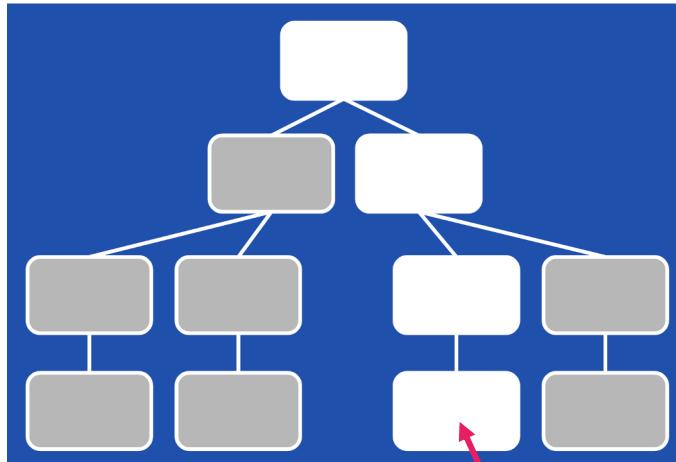
Changement de nombreuses expressions :

- Push strategy, immutability, observable
- Start from the leaf

markForCheck VS detectChanges

cd.markForCheck() :

Met le composant courant et ses parents en mode "à checker" même si ceux-ci sont en strategy **OnPush**.

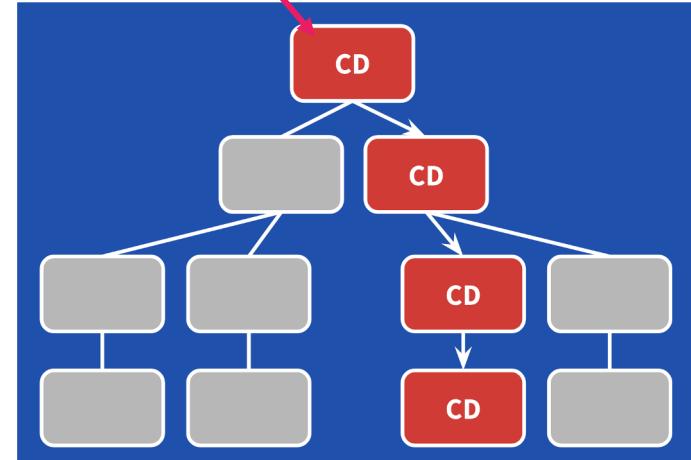


markForCheck

cd.detectChanges() :

Lance un cycle de détection de changement à partir du composant courant, quelque soit sa stratégie.

detectChanges



Change Detection : Quelques liens sur le sujet

Change detection (article de référence) :

<https://blog.thoughtram.io/angular/2016/02/22/angular-2-change-detection-explained.html>

Change detection (autres articles) :

<https://medium.com/@bencabanes/angular-change-detection-strategy-an-introduction-819aaa7204e7>

https://medium.com/@debug_mode/understating-angular-change-detection-with-example-4da0d998e3dd

<https://blog.bitsrc.io/understanding-change-detection-strategies-in-angular-d4ca7744085a>

track by *ngFor

```
● ● ●  
@Component({  
  selector: 'app',  
  template: `<ul>  
    <li *ngFor="let item of items; trackBy: trackById">{{item.name}}</li>  
  </ul>`  
})  
class AppComponent {  
  Items = [  
    {  
      id: 1,  
      name: 'item 1'  
    }, {  
      id: 2,  
      name: 'item 2'  
    },  
    ...  
  ];  
  trackById(index, item) {  
    return item.id;  
  }  
}
```

Quand ?

- gros volume de données
- Des changements sur la liste d'éléments

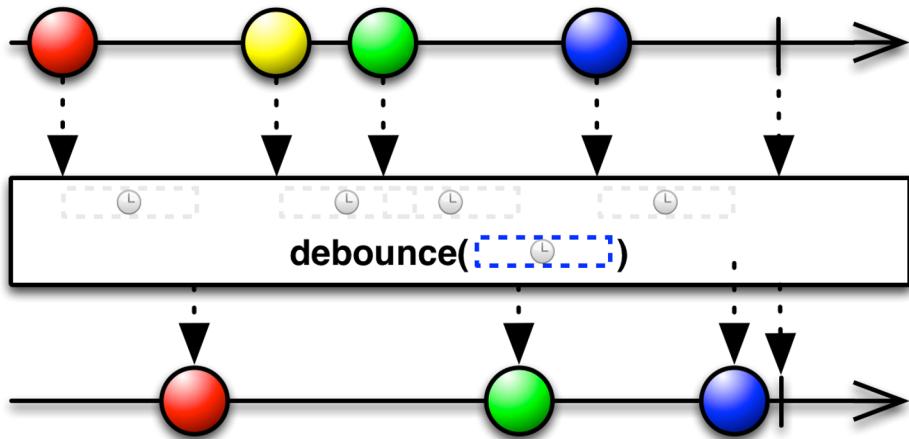
debounceTime

debounceTime() :

Ne laissez pas passer un évènement qu'après une période d'attente pendant laquelle aucun nouvel élément n'est reçu par l'opérateur.

distinctUntilChanged() :

Ne laissez pas passer l'évènement si il est identique à son état précédent



```
import {Component} from '@angular/core';
import {FormBuilder, FormGroup} from '@angular/forms';
import {debounceTime, distinctUntilChanged} from 'rxjs/operators';

public myForm: FormGroup;
constructor(fb: FormBuilder) {
  this.myForm = fb.group({name: ''});
  this.myForm.get('name').valueChanges.pipe(
    debounceTime(300),
    distinctUntilChanged(),
  ).subscribe(val => {
    console.log(val);
  });
}
```

v6

PWA : Cache & App Shell



[https://angular.io/guide/
service-worker-intro](https://angular.io/guide/service-worker-intro)

Angular & PWA

```
npm run ng add @angular/pwa
```



Configuration des caches

Les caches se configurent via le fichier **ngsw-config.json**

2 types d'éléments peuvent être mis en caches :

- Des fichiers statiques (html, css, js, ...)
- Des appels aux API

Cache des fichiers statiques

Les fichiers statiques à mettre en cache sont listés dans la propriété **assetGroups** du fichier **ngsw-config.json**

```
{
  "index": "/index.html",
  "assetGroups": [
    {
      "name": "app",
      "installMode": "prefetch",
      "resources": {
        "files": [
          "/favicon.ico",
          "/index.html",
          "/*.css",
          "/*.js"
        ]
      }
    },
    {
      "name": "assets",
      "installMode": "lazy",
      "updateMode": "prefetch",
      "resources": {
        "files": [
          "/assets/**",
          "/*.(eot|svg|cur|jpg|png|webp|gif|otf|ttf|woff|woff2|ani)"
        ]
      }
    }
  ]
}
```



Cache des appels aux API

Les réponses des appels aux API sont listées dans la propriété **dataGroups** du fichier **ngsw-config.json**

```
export interface DataGroup {
  name: string;
  urls: string[];
  version?: number;
  cacheConfig: {
    maxSize: number;
    maxAge: string;
    timeout?: string;
    strategy?: 'freshness' |
    'performance';
  };
}
```

Cache des appels aux API : 2 stratégies

Le cache des appels API peut être configuré via **2 stratégies** :

performance :

Si la réponse existe dans le cache, la version mise en cache est utilisée et aucune demande réseau n'est faite.

- **maxSize**: nombre d'éléments max en cache
- **maxAge** : durée de vie d'un élément dans le cache

freshness :

Fait systématiquement un appel réseau pour récupérer les données. Si le timeout est dépassé et que la donnée est en cache, la donnée est renvoyée à partir du cache. Lors du retour de l'API, le cache sera mis à jour.

- **maxSize**: nombre d'éléments max en cache
- **maxAge** : durée de vie d'un élément dans le cache
- **timeout**: durée à partir de laquelle le cache est utilisé à la place de l'appel réseau.

Cache des appels aux API : Les URLs

Les URLs définies ne doivent pas prêter à confusion.

Si une URL peut être interprétée comme valide par plusieurs règles, son résultat ne sera pas mis en cache.

=> Pensez à terminer vos URL par un **\$** si nécessaire et à utiliser *****, ******, **?** et **!**

****** : correspond à 0 ou plusieurs segments de chemin

***** : correspond à 0 ou plusieurs caractères à l'exclusion de /

? : correspond exactement à un caractère à l'exclusion de /

\$: indique la fin

! : désigne l'inverse du segment indiqué

```
"urls": [
    "**/nomenclature/universes$$",
    "**/nomenclature/universes/*/families$$",
    "**/nomenclature/universes/*/families/*/*products$$",
    "**/nomenclaturereferences/search$$",
    "**/nomenclature/products$$"
],
```

SW Cache tips

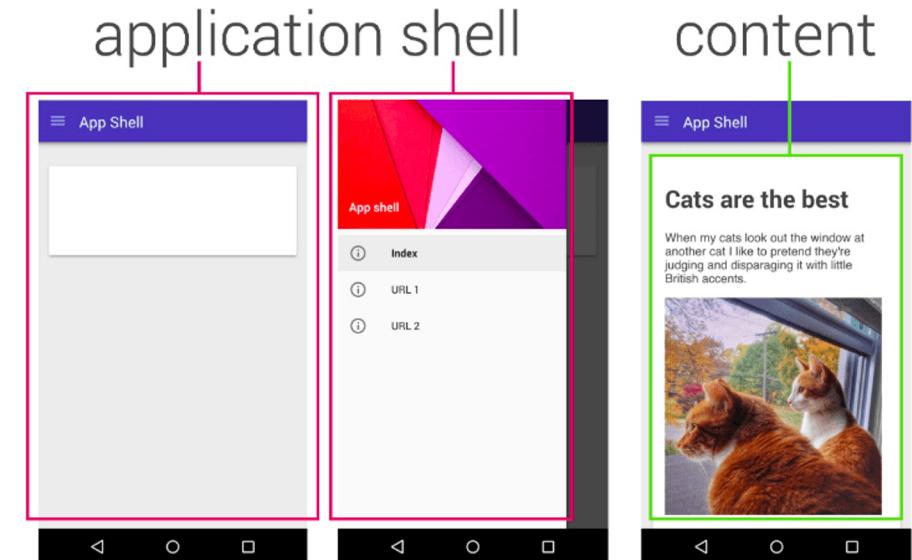
Quand vous souhaitez tester la mise en place des caches par Angular, il est conseillé d'utiliser la **navigation privée** de votre navigateur pour vous assurer que le service worker ne lira pas à partir d'un état antérieur non utilisé, ce qui peut provoquer un comportement inattendu.



app-shell

=> <https://angular.io/guide/app-shell>

- On met la partie statique/structure de l'application en cache afin de la servir plus rapidement.
- On peut générer le premier rendu directement dans le index.html pour afficher un rendu avant le démarrage d'angular.
- On peut utiliser Angular Universal afin de générer un index html côté serveur.



Cached shell loads **instantly** on repeat visits.

Dynamic content then populates the view

Observables, templates & pipe async

pipe async

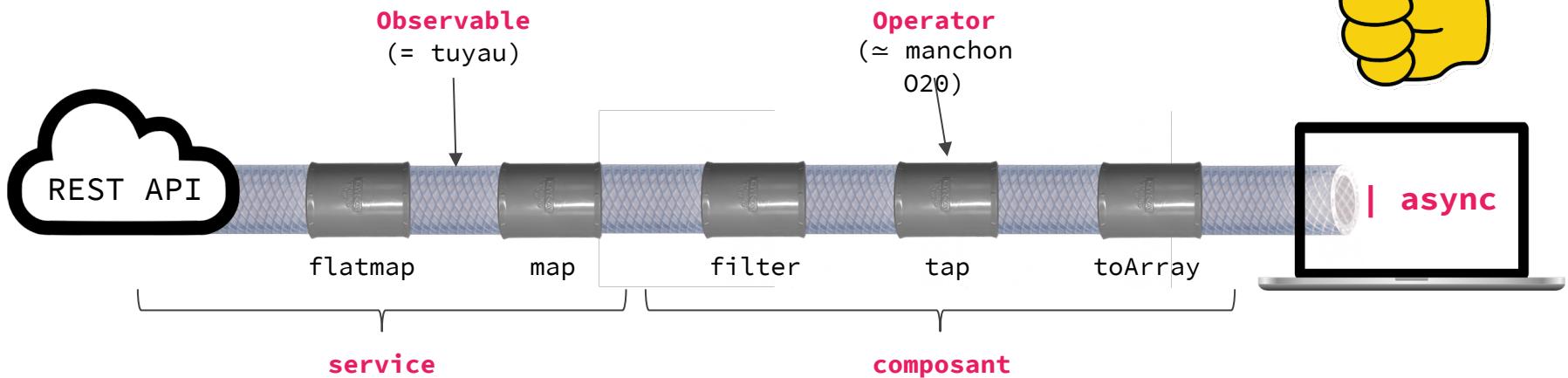
Les observables peuvent être comparés à des tuyaux dans lesquels passent les données.

Une bonne pratique en Angular consiste à amener ces tuyaux le plus près possible du template (HTML).

Tant que l'application n'est pas amenée à modifier à posteriori les informations issues du pipe, l'observable peut être acheminé jusqu'à la vue pour en afficher le contenu (i.e. tant que les données issues du pipe ne sont pas éditables).

Dans cette configuration, l'idéal est d'afficher la valeur reçue par l'observable via un **pipe async**.

pipe async : exemple



pipe async : exemple

```
@Component({
  selector: 'uni-demo-pipe-async',
  template: '{{ unicorns | async }}',
  styleUrls: ['./demo-pipe-async.component.css']
})
export class DemoPipeAsyncComponent {

  public unicorns: Observable<Unicorn[]> = this.unicornsService.getUnicorns();

  constructor(private unicornsService: UnicornsService) {}

}
```

pipe async : nglf & ngFor



```
<span *ngIf="(word$ | async)?.length > 9; else shortWord">
  Long word: {{ word$.value }}
</span>

<ng-template #shortWord>
  Short word: {{ word$.value }}
</ng-template>
```



```
<ul>
  <li *ngFor="let city of cities$ | async">
    Name: {{ city.name }},
    Population: {{ city.population }},
    Elevation: {{ city.elevation }}</li>
</ul>
```

Source : <https://alligator.io/angular/async-pipe/>

Memory Leaks



Memory Leaks

En Angular, la cause la plus répandue de **fuite mémoire** est liée aux Observables qui ne sont pas fermés à la destruction des composants.

Les fuites de mémoire viennent de la destruction et de la recréation de nos composants, lorsque nous ne nettoyons pas les subscriptions existantes. En recréant nos composants, nous ajoutons de plus en plus de subscriptions, d'où la fuite de mémoire...

Il existe 2 cas de figure :

- Les Observables avec un **subscribe()** manuel
- Les Observables appelés via un **pipe async**

Si tu subscribe(), tu unsubscribe() !



```
@Component({ ... })
export class MyComponent implements OnDestroy {

    public time: Date;
    private timeSub: Subscription;

    constructor() {
        this.timeSub = interval(1000).pipe(
            tap(i => console.log(i)),
            map(() => new Date()),
        ).subscribe(time => {
            console.log('sub' + time);
            this.time = time;
        });
    }

    ngOnDestroy(): void {
        this.timeSub.unsubscribe();
    }
}
```

pipe async :



L'utilisation du **pipe async** dans la vue permet à Angular de se désinscrire automatiquement de l'observable (unsubscribe) lorsque le composant est détruit (et c'est un bon début ^^).



Sources :

- <https://alligator.io/angular/async-pipe/>
- <https://blog.angularindepth.com/d8f9aa42f6a0>

Observable : break execution flow

Se désinscrire des observables souscrit est indispensable que ce soit :

- automatiquement dans le cas d'un pipe async
- manuellement dans le cas d'un subscribe manuel

Cependant, cela n'empêche pas les opérateurs d'être sollicités tant que la source de l'observable continue d'envoyer des données.

Une méthode élégante pour remédier à ce problème consiste à utiliser un Subject qui servira de déclencheur à la fermeture de tous les observables du composant une fois le composant détruit.

Observable : break execution flow

```
@Component({ ... })
export class MyComponent implements OnDestroy {

  public time: Observable<Date>;
  private ngUnsubscribe = new Subject<void>();

  constructor() {
    this.time = interval(1000).pipe(
      takeUntil(this.ngUnsubscribe),
      tap(i => console.log(i)),
      map(() => new Date())
    );
  }

  ngOnDestroy(): void {
    this.ngUnsubscribe.next();
    this.ngUnsubscribe.complete();
  }
}
```



lazy loading (de module)

Permet de ne pas charger toute l'application directement mais d'uniquement charger les modules au moment de leur utilisation.

```
● ● ●

export const ROUTES: Routes = [
  {
    path: '',
    pathMatch: 'full',
    loadChildren: './app.module#AppModule',
  },
  {
    path: 'dashboard',
    loadChildren: './pages/dashboard/dashboard.module#DashboardModule',
  },
  {
    path: 'ordersValidation',
    loadChildren: './pages/orders-validation/orders-validation.module#OrdersValidationModule',
  },
  ...
]
```

Pipes Fonctions on templates

Préférez toujours l'utilisation des pipes pour formater votre template.

Des **caches internes** sont gérés par Angular pour le rendu des pipes.

De manière générale, l'utilisation de fonctions dans le template est contre performant (elles sont exécutées à chaque cycle de détection du changement).

```
<h1>{{ format(data) }}</h1>
```

```
<h1>{{ data | format }}</h1>
```



Pipes impures & OnPush strategy

Note : En mode **OnPush**, les pipe impures sont moins problématiques (pas de cycle de détection de changement = pas de réévaluation).



```
@Pipe({
  name: 'calculate',
  pure: true
})
export class CalculatePipe {
  transform(num: number) {
    return fibonacci(num);
  }
}
```

Images sizes

Pensez toujours à valider le poids des images que vous intégrez dans vos projets. Des tailles excessives peuvent fortement altérer l'expérience des utilisateurs avec un réseau limité.



budgets (bundle size)

Il est possible de définir des tailles de livrables limites à surveiller lors de la phase de build. Ces limites se spécifient dans le fichier **angular.json** dans la section **budgets**.

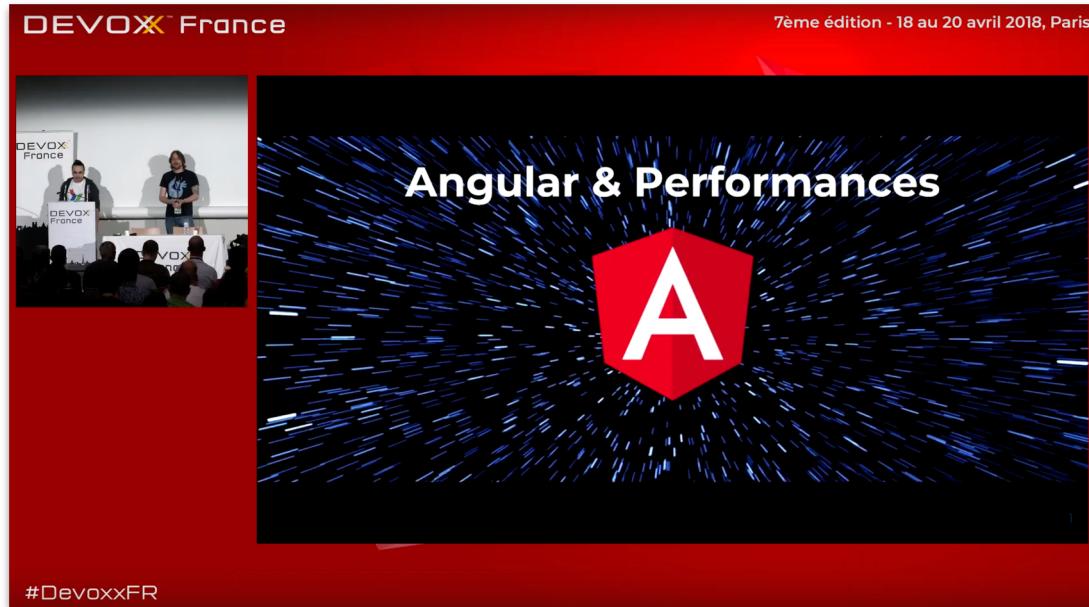
```
// angular.json
"configurations": {
  "production": {
    ...
    "budgets": [
      {
        "type": "initial",
        "maximumWarning": "2mb",
        "maximumError": "5mb"
      }
    ],
    ...
  }
}
```

Récapitulatif :

<https://medium.com/@spp020/44-quick-tips-to-fine-tune-angular-performance-9f5768f5d945>

Vidéo sympa (FR 2018) :

<https://youtu.be/ZxZ0v5wop0s>



Angular Material



Angular Material

Angular Material se s'installe grâce à une seule commande :
(depuis angular 6)

npx ng add @angular/material



Utilisation des composants

Les composants d'Angular-Material sont répartis dans des modules séparés afin de limiter l'impact sur la taille de l'application une fois buildée.

Afin de pouvoir utiliser un composant, il faut donc l'importer dans la définition du @NgModule.

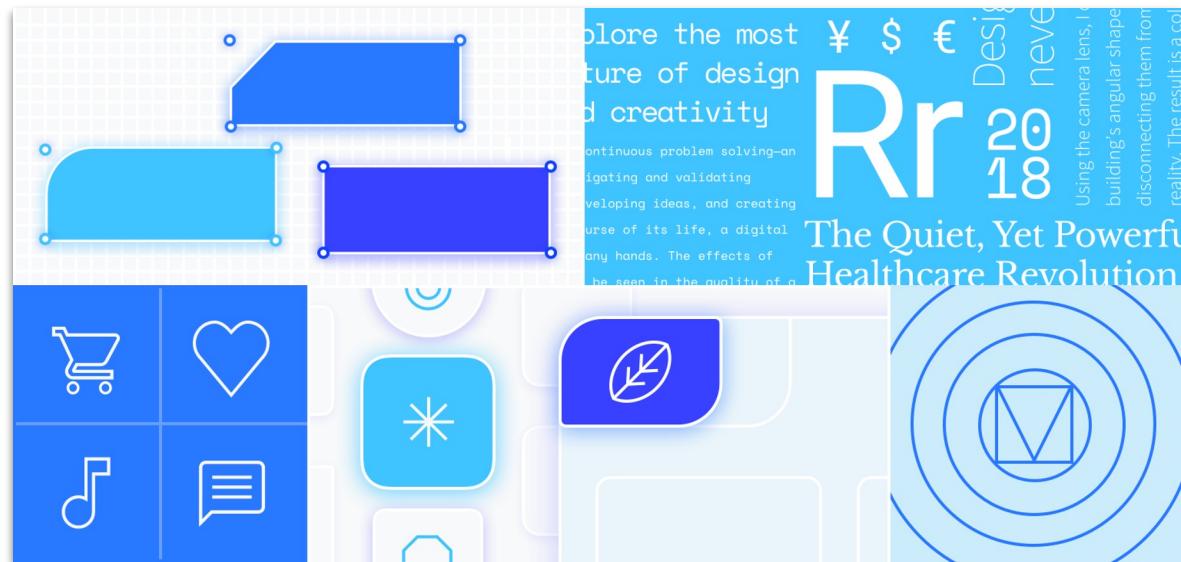
```
import {MatButtonModule} from '@angular/material';

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

Angular Material

Implémentation pour Angular de la spécification Material (<https://material.io/>)

Doc : <https://material.angular.io/>



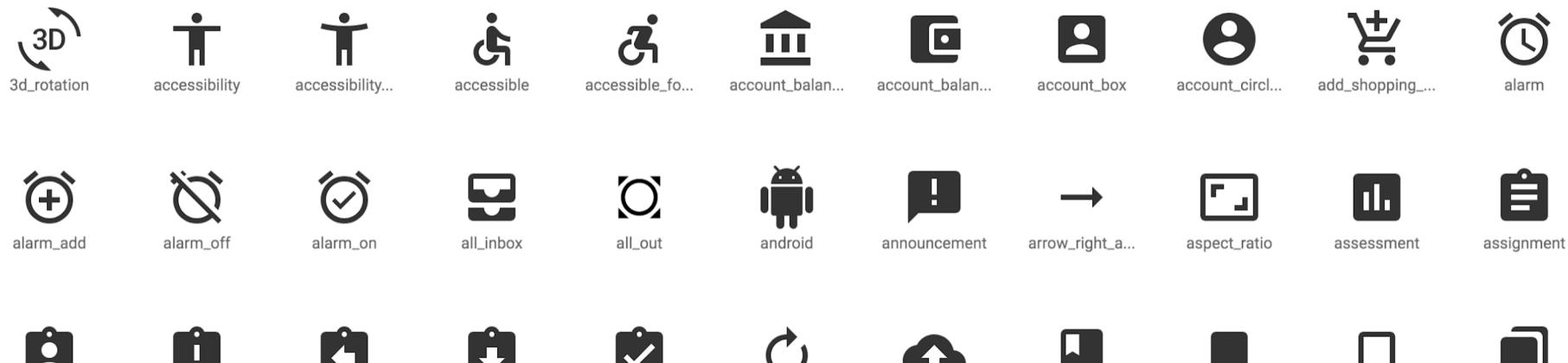
Material Icons

Configuré lors de l'ajout de @angular/material

Liste des icônes : <https://material.io/tools/icons>

Utilisation :

1. Importer le module **MatIconModule**
2. <**mat-icon**>account_box</**mat-icon**>



Theming

Configuration des couleurs : <https://material.angular.io/guide/theming>

Il existe des thèmes pré-configurés. Pour les utiliser, il suffit de les importer dans le fichier style.scss (1 seul au choix) :

- `@import "~@angular/material/prebuilt-themes/deeppurple-amber.css";`
- `@import "~@angular/material/prebuilt-themes/indigo-pink.css";`
- `@import "~@angular/material/prebuilt-themes/pink-bluegrey.css";`
- `@import "~@angular/material/prebuilt-themes/purple-green.css";`

Green 50	#E8F5E9
100	#C8E6C9
200	#A5D6A7
300	#81C784
400	#66BB6A
500	#4CAF50
Light Green 50	#F1F8E9
100	#DCEDC8
200	#C5E1A5
300	#AED581
400	#9CCC65
500	#8BC34A
Lime 50	#F9FBEB
100	#F0F4C3
200	#E6EE9C
300	#DCE775
400	#D4E157
500	#CDDC39

schematics

Les schematics permettent d'étendre les fonctionnalités de base d'angular-cli

Material propose quelques schematics pour créer des **composants pré-configurés** en une seule commande

Ces schematics sont **packagés avec Angular Material** (pas d'installation spécifique)

Doc : <https://material.angular.io/guide/schematics>

Exemples :

- `npx ng generate @angular/material:material-nav --name=nav`
- `npx ng generate @angular/material:material-dashboard`
- `npx ng generate @angular/material:material-table`



Extending the CLI with Schematics



new



generate



update



add

component
directive
pipe
service
...

>=V6



@angular/cdk

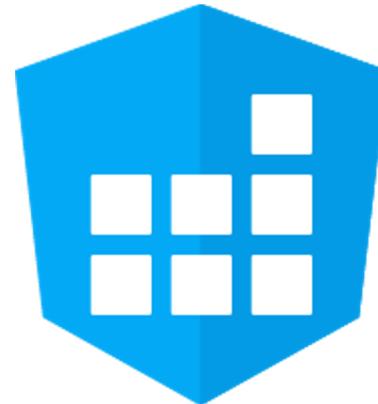
CDK : Component Dev Kit

URL : <https://material.angular.io/cdk>

Ensemble d'outils pour développer ses propres composants graphiques

Les composants Angular Material s'appuient sur ce package

Librairie **bas-niveau**.



CDK : Parce que tout le monde n'a pas le style “Material”



<http://ng.ant.design>



<https://js.devexpress.com/>



<https://akveo.github.io/nebular/>



Clarity



INFRAGISTICS



ngBootstrap



Lightning

@angular/cdk

Présentation au AngularMIX 2017 (EN) : <https://www.youtube.com/watch?v=kYDLlfpTLEA>

Slides : <http://g.co/ng/mix17-cdk>

Les CDK est découpé en packages :

- **Accessibility** : outils pour améliorer **l'accessibilité**
- **Bidirectionality** : gère les changements de **disposition LTR / RTL**
- **Layout** : fournit des utilitaires pour créer des interfaces utilisateur réactives qui réagissent aux **modifications de la taille de l'écran**
- **Observers** : fournit la directive **MutationObserver** construites au-dessus des observateurs de la plate-forme Web native pour **détecter un changement**
- **Overlay** : fournit un moyen d'ouvrir les **panneaux flottants** sur l'écran
- **Portal** : système flexible pour rendre le contenu dynamique dans une application.
- **Scrolling** : outils pour les directives qui réagissent aux événements de **défilement**

PWA



Origine du concept

PWA = Progressive Web App (Terme Marketing, pas une nouvelle technologie)

Origine : Article de Alex Russell (software engineer at Google) d'aout 2015 :
<https://medium.com/@slightlylate/progressive-apps-escaping-tabs-without-losing-our-soul-3b93a8561955>



Alex Russell [Follow](#)

Dragging the web platform into the mid-naughties, one spec at a time. Progressive Web Apps are my jam.

Aug 10, 2015 · 6 min read

Progressive Web Apps: Escaping Tabs Without Losing Our Soul

It happens on the web from time to time that powerful technologies come to exist without the benefit of marketing departments or slick packaging. They linger and grow at the peripheries, becoming old hat to a tiny group while

PWA : Les 10 points clefs



Linkable - Accessible via une **URL** (facilement partageable)

Secure - Une PWA doit être en **HTTPS**

Fresh - Tout le temps **à jours** (web)

Discoverable - **Indexable** par les moteurs de recherche

Responsive

Content is like water

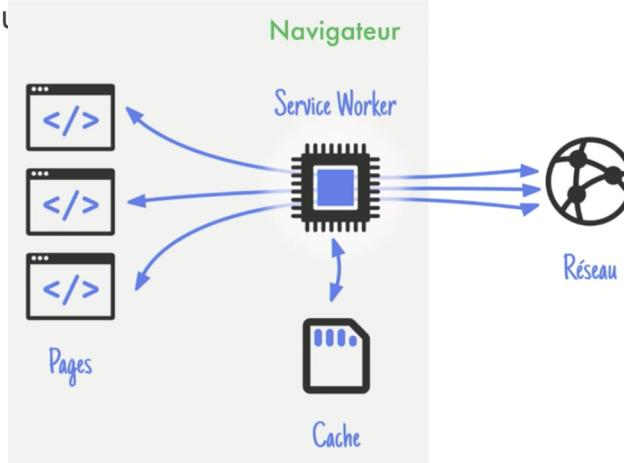


"We can't design a new experience on every platform from scratch every time. Content is like water; it takes many forms and flows into all these different containers."

Josh Clark, designer & developer

Connectivity Indépendant

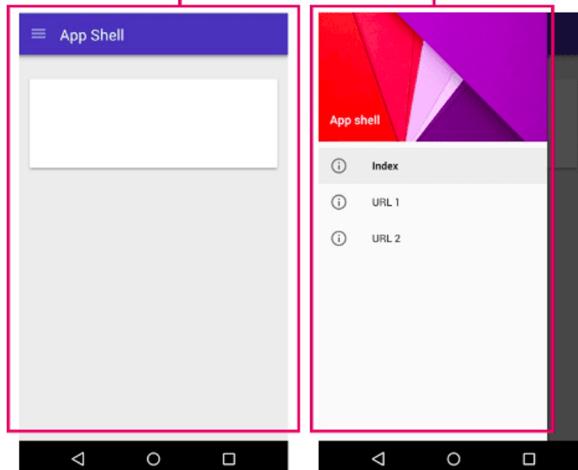
- Service Workers
 - HTTPS Only
 - Thread Indépendant
 - Spec W3C : <https://w3c.github.io/ServiceWorker/>
 - Cookbook Mozilla : <https://serviceworke.rs/>
 - <https://jakearchibald.com/2014/offline-cookbook/>
 - <https://jakearchibald.github.io/isserviceworkerready/> -> Yes ^^
 - Tutorial : <https://pwa-workshop.js.org/>
- API Cache (Map de requêtes)
- API Fetch



App-like Interactions

App Shell <https://developers.google.com/web/fundamentals/architecture/app-shell>

application shell



Cached shell loads **instantly** on repeat visits.

content



Dynamic content then populates the view

Angular & app-shell

```
ng generate app-shell --client-project my-app --universal-project server-app
```

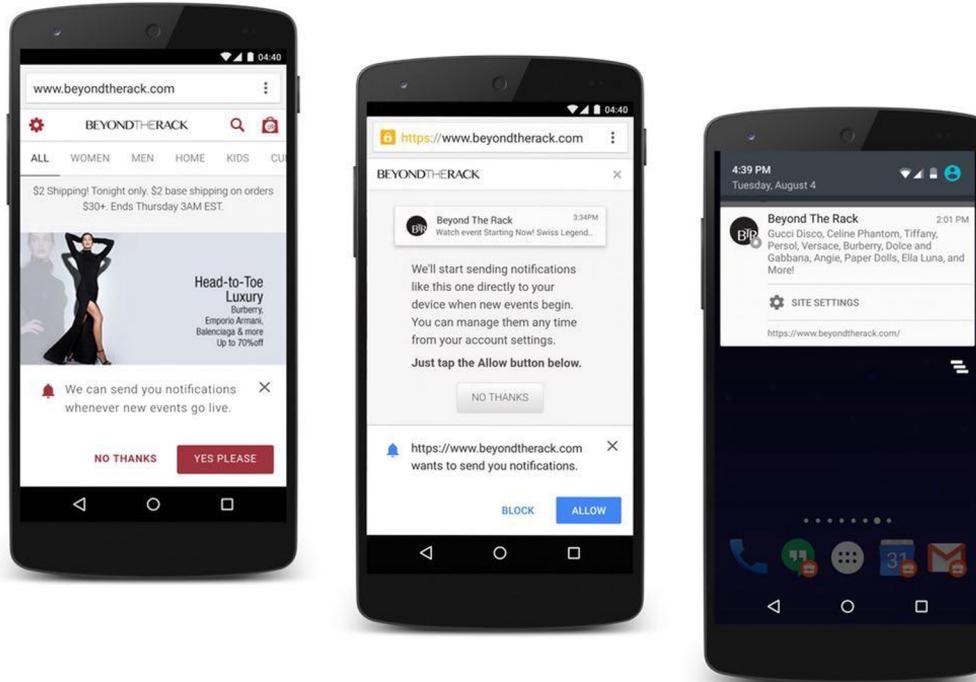
(doc : <https://github.com/angular/angular-cli/wiki/stories-app-shell>)



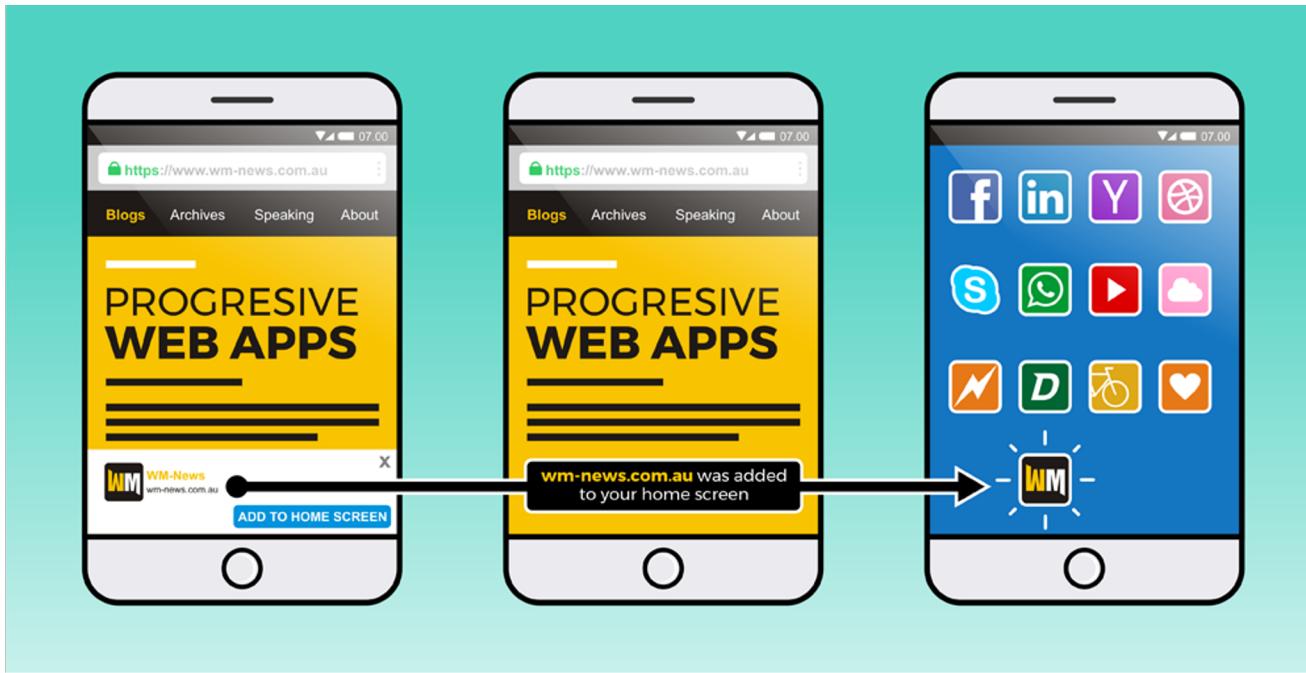
Re-engageable

Notifications (même avec le browser éteint)

Demo : <https://serviceworkers.push-simple.html>



“Installable”



Progressive

Progressive : fait allusion à la façon dont ces pages web peuvent évoluer :

- en partant d'une page web « normale »,
- que l'on enrichit **progressivement** de fonctionnalités



Toutes ces fonctionnalités doivent être pensées comme potentiellement non supportées par le navigateur de l'utilisateur.

Tester sa PWA avec Google Lighthouse

<https://github.com/GoogleChrome/lighthouse>



Etat des lieux des features

Support variable en fonction des navigateurs

<https://whatwebcando.today/>

The screenshot shows the homepage of the "What Web Can Do Today" website. The header features a smartphone icon with the HTML5 logo and a globe icon, followed by the title "What Web Can Do Today" and the subtitle "Can I rely on the Web Platform features to build my app? An overview of the device integration HTML5 APIs". Below the header is a legend: a green checkmark for "Feature available in your current browser" and a red X for "Feature not available in your current browser". The main content is organized into six categories, each with a list of features and their availability status:

Category	Available Features (Green)	Not Available (Red)
Camera & Microphone	AUDIO & VIDEO CAPTURE, ADVANCED CAMERA CONTROLS, RECORDING MEDIA, REAL-TIME COMMUNICATION	
Native Behaviors	LOCAL NOTIFICATIONS, PUSH MESSAGES, HOME SCREEN INSTALLATION, FOREGROUND DETECTION, PERMISSIONS	
Seamless Experience	OFFLINE MODE, BACKGROUND SYNC, PAYMENTS, CREDENTIALS	INTER-APP COMMUNICATION
Advertisement	Progress' Kendo UI advertisement	
Operating System	OFFLINE STORAGE, FILE ACCESS	
Location & Position	GEOLOCATION	GEOFENCING

[https://angular.io/guide/
service-worker-intro](https://angular.io/guide/service-worker-intro)

Angular & PWA

Certaines caractéristiques des PWA peuvent être pré-configurées avec le cli :

- Mise en cache hors connexion grâce aux [service workers](#) afin de rendre l'application disponible hors ligne
- Génère le [Manifeste d'application](#) pour définir l'apparence de l'application (écran de démarrage, icônes, nom, plein écran ou non)
- Configuration de la fonction d'écran d'accueil pour que l'application (Web) soit accessible comme n'importe quelle autre application native
- Activation des [notifications Web](#) pour permettre d'envoyer des notifications en arrière-plan à l'utilisateur, tout comme les applications mobiles natives!

Article sur le sujet : <https://blog.angulartraining.com/progressive-web-apps-for-angular-6-and-beyond-f7e4b9a2f9fa>

Angular & PWA

```
npm run ng add @angular/pwa
```



PWA - Quelques liens

Quelques liens utiles :

- <https://developer.mozilla.org/fr/Apps/Progressive>
- <https://progressive-web-apps.fr/definition-progressive-web-apps-pwa>
- <https://thierrygustin.com/progressive-web-app-wordpress/>
- <https://www.search-foresight.com/the-next-big-thing-progressive-web-apps/>
- <https://www.youtube.com/watch?v=dSKp-76Ur6E&index=2&list=PLTBioAEvUEj0oFF2v71IPvu8iFBQYYNEt>

Quelques exemples :

- <https://www.pokedex.org/>
- <https://pwa.rocks/>

Workshop :

- <https://pwa-workshop.js.org/>

Angular Universal



ANGULAR UNIVERSAL

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

Angular Universal

URL du projet : <https://universal.angular.io/>

Le projet Universal a été conçu autour de 3 axes :

Meilleure performance perçue

Les premiers utilisateurs de votre application verront instantanément une vue rendue par le serveur, ce qui améliorera grandement les performances perçues et l'expérience utilisateur globale. Selon les recherches de Google, la différence de seulement 200 millisecondes dans les performances de chargement de la page a un impact sur le comportement de l'utilisateur.

Optimisé pour les moteurs de recherche

Bien que Googlebot explore et rende les sites les plus dynamiques, de nombreux moteurs de recherche attendent du code HTML. Le pré-rendu côté serveur est un moyen fiable, flexible et efficace pour garantir que tous les moteurs de recherche puissent accéder à votre contenu.

Aperçu du site

Permet de s'assurer que Facebook, Twitter et toutes les autres applications de médias sociaux affichent correctement une image de prévisualisation de votre application.

Universal to platform-server ?



Le projet Universal est maintenant intégré dans le noyau d'Angular depuis sa version 4 dans le package “platform-server”.

=> <https://github.com/angular/angular/tree/master/packages/platform-server>

Le projet universal en tant que tel (<https://github.com/angular/universal>) est deprecated depuis.

The Angular Universal project consists of the base platform API and the surrounding tools that enables developer to do server side rendering (or pre-rendering) of Angular applications. The platform API part has been merged into Angular core as of 4.0.



Foxandxss commented on 3 Jul 2017

Member

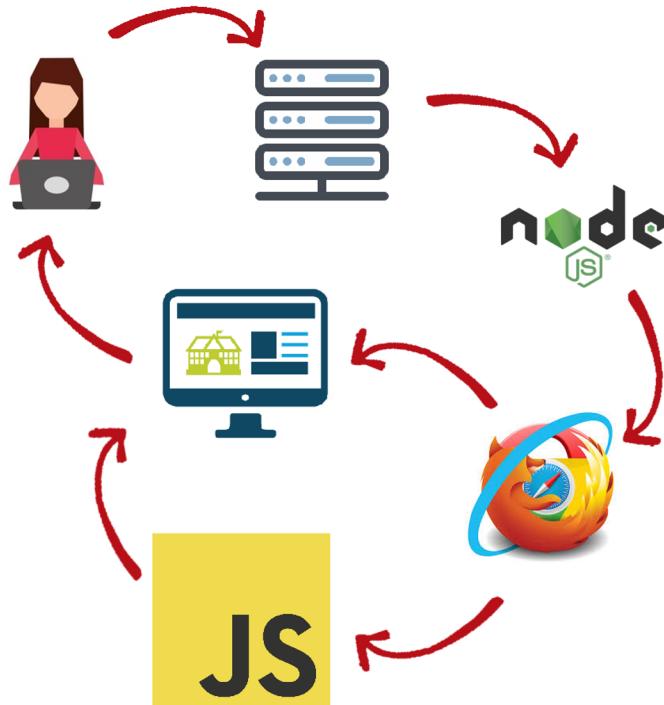


No, you are still confused.

What you understand as "angular universal" (that repo you are pointing to me) has been integrated into the core of angular. You can do server render with angular right now **without** the need of that repo which is now deprecated.



Universal : fonctionnement



1. L'utilisateur appelle la page
2. La requête arrive au serveur
3. NodeJS génère du code HTML et l'envoie au navigateur
4. Le navigateur affiche la vue depuis le HTML et l'affiche immédiatement à l'utilisateur, simultanément JavaScript est en cours d'exécution
5. Lorsque JavaScript termine le bootstrap de l'application, il modifie la vue rendue en HTML avec l'application Angular
6. L'utilisateur voit complètement l'application Angular interactive

Universal : Configuration

Doc officielle :

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

Exemple complet officiel :

<https://angular.io/generated/zips/universal/universal.zip>

Universal avec angular-cli :

<https://medium.com/sohoffice/angular-universal-an-adventure-9d969d401072>



Configurer Universal à partir d'un projet existant créé avec angular-cli

<https://github.com/angular/angular-cli/wiki/stories-universal-rendering>

Lazy Loading



Lazy Loading - Concept

Lazy loading = diviser l'application en plusieurs modules Angular.

Chaque module :

- est chargé à la demande
- définit ses propres **routes**, **composants**, **pipes** et **services**
- est packagé dans un bundle (fichier JavaScript)
- chaque bundle est distinct du bundle principal de l'application (fichiers séparés)

Lazy Loading - Objectifs

Le **lazy loading** est principalement utile pour :

- Améliorer les **performances** de chargement de l'application (📱 mobile)
- **Sécuriser** les pages critiques de l'application (page d'admin, ...)



Lazy Loading - Lien avec le routeur principal

- Le module principal définit une route pour accéder au module lazy-loadé.
- Le router ne télécharge le bundle du module à lazy-loader que lorsque l'utilisateur navigue vers cette route.
- Une fois le module chargé, les composants, services et routes filles sont ajoutés à l'application dynamiquement

```
// app-routing.module.ts

import {NgModule} from '@angular/core';
import {Routes, RouterModule} from '@angular/router';
import {UnicornListComponent} from './unicorn-list/unicorn-list.component';

const routes: Routes = [
  {path: '', component: UnicornListComponent},
  {path: 'admin', loadChildren: './admin/admin.module#AdminModule'}
];

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

Lazy Loading - Les modules lazy loadés

- Le module fils n'importe pas le BrowserModule.
- À la place, il importe CommonModule.
- Le module fils définit ses propres routes dans un fichier séparé
- Le routeur étant un singleton, au lieu d'utiliser RouterModule.forRoot(), le module fils utilise RouterModule.forChild()

Lazy Loading - Les modules lazy loadés

Angular-cli permet de créer un module fils lazy loadé :

```
npm run ng -- generate module admin --routing
```

```
// admin-routing.module.ts

import {NgModule} from '@angular/core';
import {Routes, RouterModule} from '@angular/router';
import {AdminComponent} from './admin.component';

const routes: Routes = [
  {path: '', component: AdminComponent}
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class AdminRoutingModule { }
```

```
// admin.module.ts

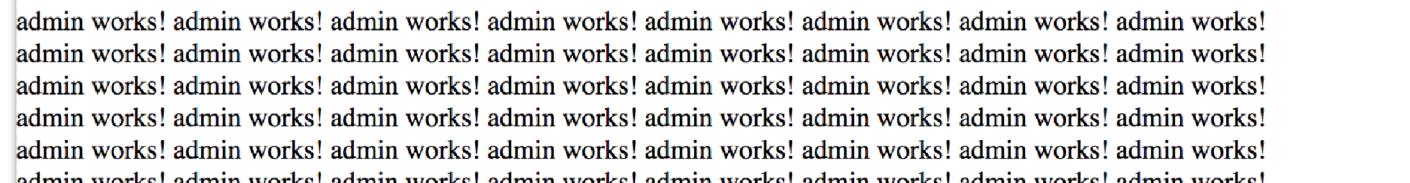
import {AdminComponent} from './admin.component';
import {NgModule} from '@angular/core';
import {CommonModule} from '@angular/common';
import {AdminRoutingModule} from './admin-routing.module';

@NgModule({
  imports: [
    CommonModule,
    AdminRoutingModule,
  ],
  declarations: [AdminComponent]
})
export class AdminModule {}
```

Lazy Loading - Network capture

← → C i localhost:4200/admin

Unicorn-ng

admin works!
admin works! admin works! admin works! admin works! admin works! admin works! admin works! admin works!
admin works! admin works! admin works! admin works! admin works! admin works! admin works! admin works!
admin works! admin works! admin works! admin works! admin works! admin works! admin works! admin works!
admin works! admin works! admin works! admin works! admin works! admin works! admin works! admin works!


Elements Console Sources Network **Performance** Memory Application Security Audits Redux

● ⚡ 🎥 Group by frame Preserve log Disable cache Offline Online ▾

Filter Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other

Name	Method	Status	Type	Initiator	Size	Time
runtime.js	GET	200	script	admin	8.1 KB	
polyfills.js	GET	200	script	admin	222 KB	
styles.js	GET	200	script	admin	132 KB	
vendor.js	GET	200	script	admin	6.9 MB	
main.js	GET	200	script	admin	45.6 KB	
admin-admin-module.js	GET	200	script	bootstrap:143	11.2 KB	



Vous l'aurez compris,
pour le **lazy-loading**,
nous allons devoir
organiser notre
application en **modules**...