
SOAT

Angular 2

3 jours

Introduction générale

Objectifs de la formation

- Qu'est ce qu'une **SPA**, pourquoi Angular
- Découvrir et utiliser **Typescript**
- Installer un **environnement** de dev / tests / déploiement
- **Lancer** une application Angular
- Organiser son application en **hiérarchie de composants**
- **Manipuler le DOM** lié aux composants
- Utiliser et créer des **services** pour la logique métier
- Mettre en place un **routeur** afin de naviguer dans l'application
- Gérer des **formulaires** de saisie
- Mettre en place un **flux unidirectionnel** de données
- **Tester** son application

Programme de la formation

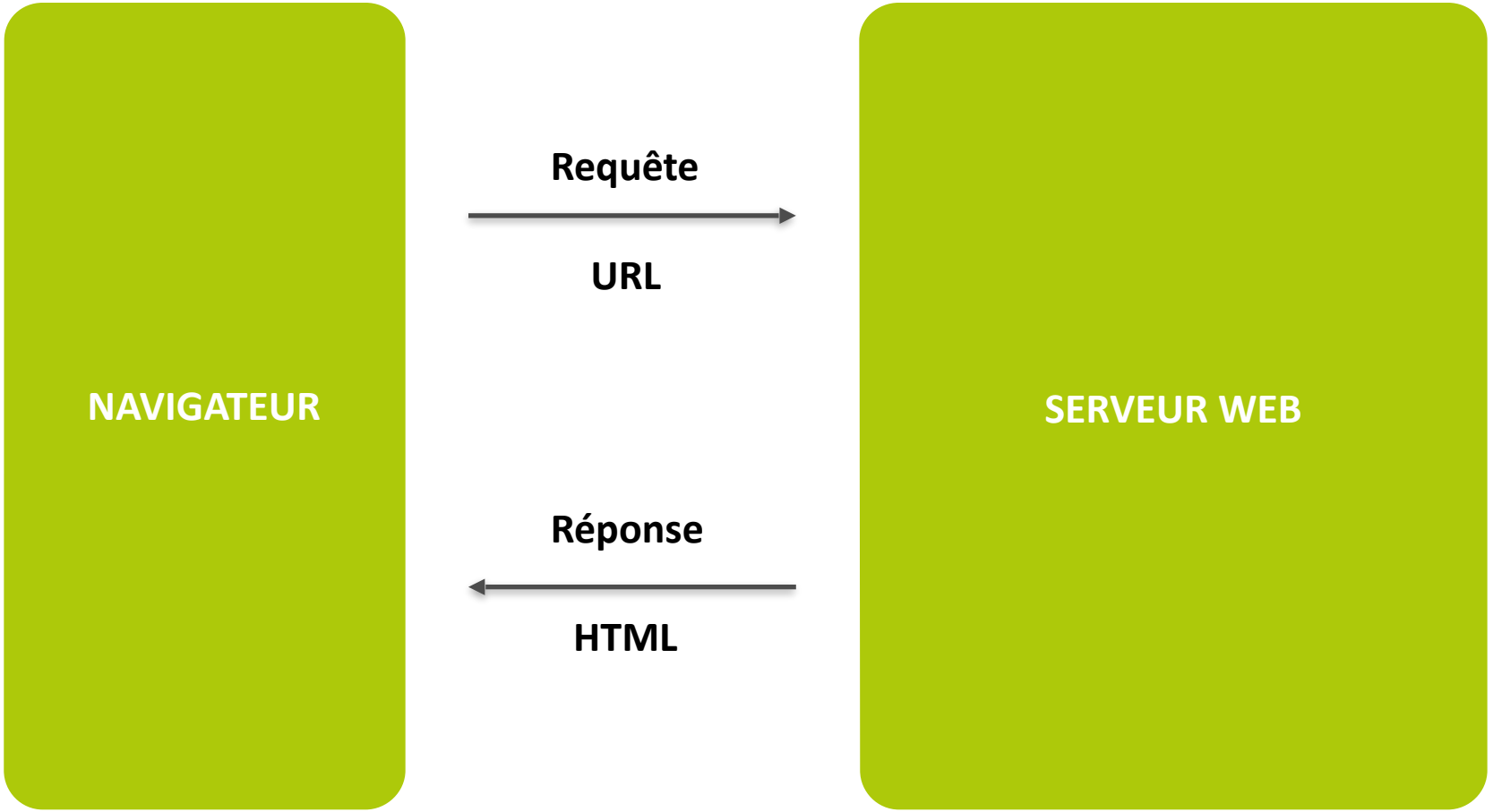
1. Single Page App, d'Angular 1 à Angular 2
2. ES5, ES6 et Typescript
3. Installer un environnement de développement
4. Bootstrapper Angular
5. Les Components
6. Les Pipes
7. Les Services
8. Le Router
9. Redux
10. HTTP
11. Components enfants et Directives
12. Les formulaires
13. Les tests unitaires
14. Liens et ressources

Single Page Application et Angular

Objectifs du chapitre

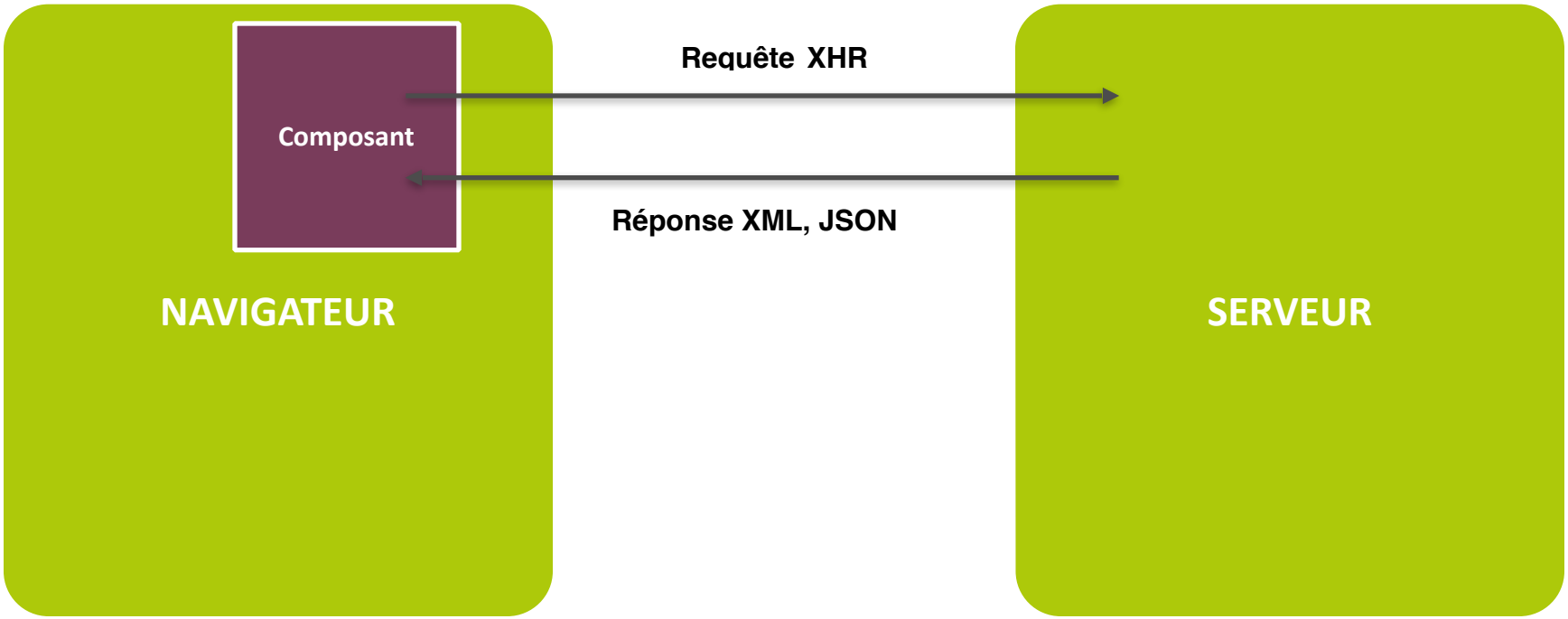
- Qu'est-ce qu'une **Single Page Application**
- Caractéristiques d'**Angular 1**
- Les concepts d'**Angular 2**

Sites web classiques



Ajax et Single Page Applications (1)

- AJAX = Asynchronous Javascript and XML
- Basé sur xmlHttpRequest (xhr)

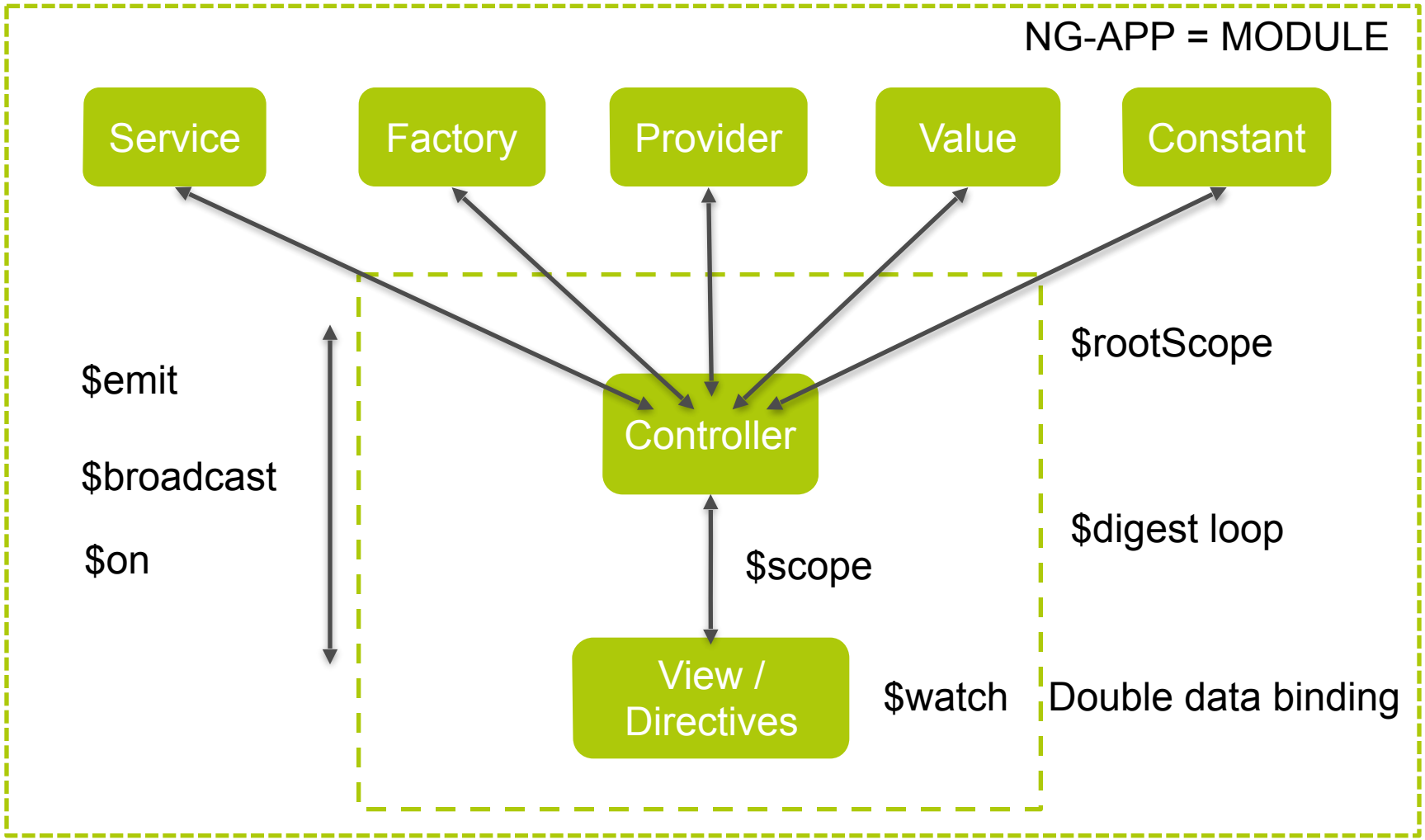


Ajax et Single Page Applications (2)

1. Eléments nécessaires chargés au **démarrage**
2. **Communication dynamique** : Le serveur ne renvoie que des éléments de la page
3. **Contexte** non perdu lors de la navigation

- Meilleure **expérience utilisateur**
- Indépendance Client / Serveur : **API REST**

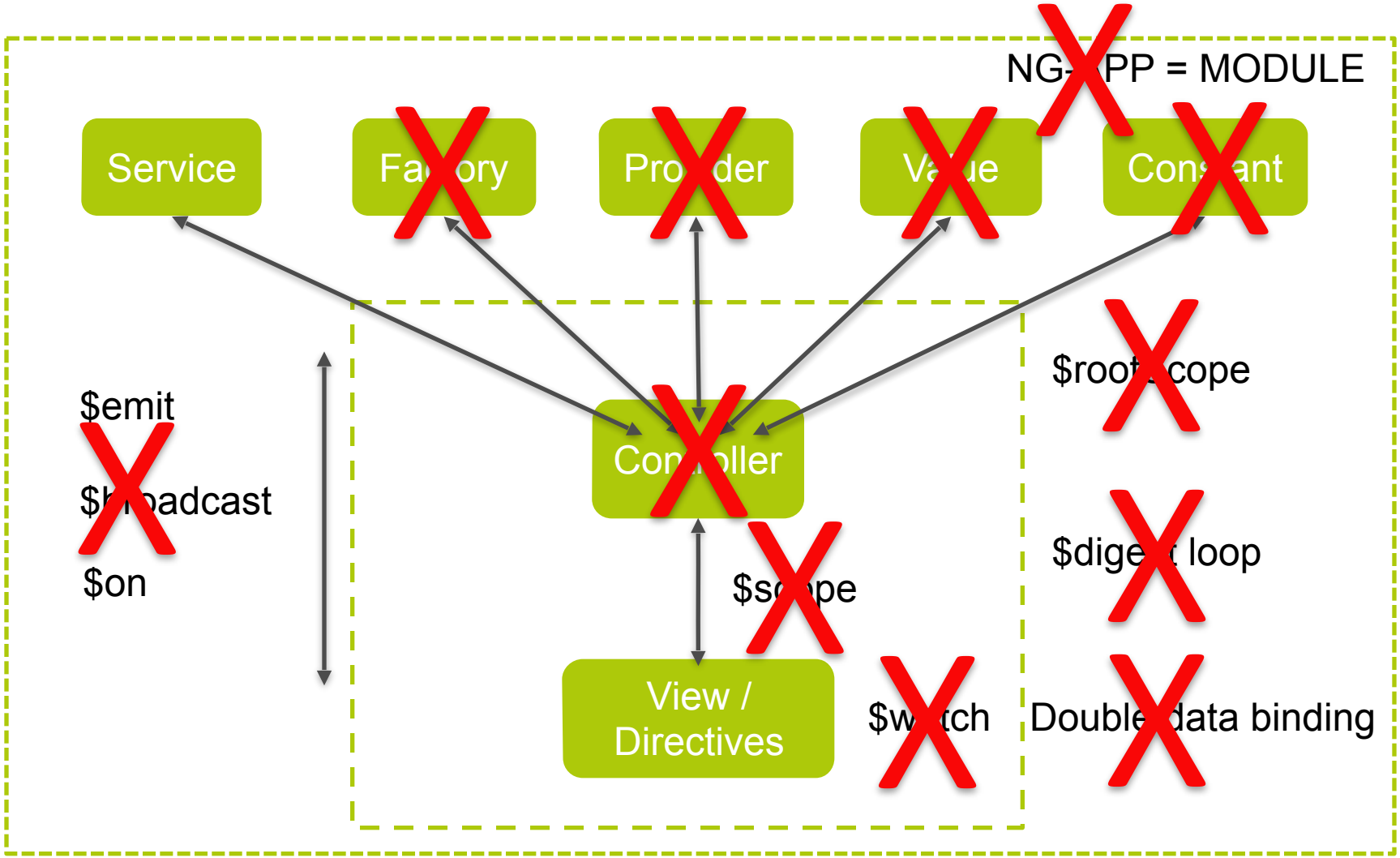
Angular 1 - Concepts



Angular 1 - Inconvénients

- **Couplage fort** Controller / View
- Double Data Binding **gourmand**
- Api de directive **complexe**
- **5 services** : lequel choisir ?
- **Difficulté** à intégrer des librairies tièrces (à cause de la $digest loop)
- Router natif : Fonctionnalités **simples**, utilisation du ui-router

Angular 1 - Evolution vers la v2



Angular 2 - Concepts

- Préconisation de langage = **TypeScript**
- **Modules ES6** : 1 fichier = 1 objet = 1 module
- Architecture de **components** = 1 classe de type **web component**
- 1 seul **service** = 1 classe
- Plus de double data binding : Détection du changement avec **Zone.js**
- Nouveau **router**, associé aux components
- **Observables** avec RxJs
- Préconisation de flux unidirectionnel des données => **Redux**
- Chargement d'angular dans un **web worker**
- **Server Side Rendering**

Angular 2 - Avantages

- **TypeScript** : encapsule les dernières versions de javascript
- Meilleure **performance**
- **Tooling** plus abouti
- Meilleure **scalabilité**
- Courbe **d'apprentissage** plus aisée
- **Routing** plus souple
- Simplification de la **gestion d'état** de l'application
- Solution au problème de **référencement** naturel (SEO)

Angular 2 - inconvénients

- Version **Beta** (documentation incomplète, fonctionnalités non implémentées
- Pas de **rétrocompatibilité** avec la v1 : migration complexe

En conclusion, Angular 2 c'est

- Un **nouveau** framework
- Une architecture front-end qui tend vers les **standards du futur**
- Des solutions pour une application **performante**

Site officiel : https://angular.io/

Javascript et TypeScript

Objectifs du chapitre

- Rappels ES5
- ES6
- ES7
- TypeScript

ES5 - Historique

- Norme publiée le 3 décembre 2009
- Compatibilité tous navigateurs et IE >= v9

ES5 - Variables et fonctions

- Langage typé. Le type n'est pas déclaré
- Type implicite dépendant de la valeur affectée

```
toto = 1            // variable globale de type number
var tutu = 'a'      // variable globale de type string
var temp = {}       // variable globale de type objet
```

- "var" implique une déclaration locale

```
toto = 1            // variable globale à window
var tutu = 'a'      // variable globale à window

function maFct() {
   glob = 'ma globale'      // variable globale (pas de mot clé var)
   var temp = {}            // variable locale
}
```

ES5 - Fonctions et paramètres

- Les paramètres de type Objet passés à une fonction sont passés par référence
- Les paramètres de type litéraux sont passés par valeur

```
(function() {

            var monJson = {}
            var tutu = 'a'

            maFct(tutu, monJson)

            function maFct(parmT, parmO) {

                        parmT = 'Titi'

                        parmO.valeur = 1

            } // les fonctions sont "hoistées"

            console.log(tutu)                 // "a"
            console.log(monJson.valeur)       // 1

})() // IIFE : module pattern
```

ES5 - Truthy Falsy

- Falsy : undefined, null, 0, false, NaN
- Truthy : autres valeurs

```
var a
if (!a) {
    console.log('a est falsy')
}

a = 'valeur'
console.log( a == true )         // true
console.log( a === true )        // false, === prend en compte le type de la variable

a = {}
console.log( a == true )         // true

a = []
console.log(a == true )          // true
console.log(a.length == true)    // false

a = null
console.log(a || 'valeur')       // "valeur"
console.log(a ? 'vrai' : 'faux'  // "faux"
```

ES5 - Closures

- Fonction déclarée dans une fonction
- La fonction enfant a accès aux variables locales de son parent

```
function init() {

    var nom = "SOAT"              // nom est une variable locale créée par init

    function afficheNom() {       // afficheNom() est une fonction interne, une closure
        alert(nom)                // afficheNom() utilise une variable de la fonction parente
    }

    afficheNom()
}

init()
```

ES5 - Objet JSON

```
var personne = {
    prenom: 'Laurent',
    nom: 'Dupont',
    age: 25,
    couleurYeux: 'bleu'
}

console.log(personne.prenom)     // affiche "Laurent"

console.log(personne['age'])     // affiche 25

var methodes = {

                getYeux: function(obj) {
                    return obj.couleurYeux
                },

                getPrenom: function(obj) {
                    return obj.prenom
                }
};

console.log( methodes.getYeux(personne) )       // affiche "bleu"
console.log( methodes.getPrenom(personne) )     // affiche "Laurent"
```

ES5 - new Object

```
var MyClass = function(parm2) {
                this.parm1 = null
                this.parm2 = parm2
};

MyClass.prototype.myMethod = function(valeur) {
                this.parm1 = valeur
                console.log(this.parm1)
                console.log(this.parm2)
};

var myObj = new MyClass('première valeur')

myObj.myMethod('autre valeur')

console.log(myObj.parm2)
```

- Attention : "this" correspond au contexte de l'objet appelant

ES6

- Norme publiée en juin 2015. Appelé aussi ES2015.
- Support partiel des navigateurs, nécessite l'utilisation d'un **transpiler**

ES6 - Variables et constantes

Affectation par Let

```
{
    let letAffectation = "ES6";
}
letAffection === "ES6"; // ERREUR la variable est définie hors score
```

Constante

```
{
     const PI = 3.1415926; // immutable et pas de hoisting

}
```

- Attention : Avec une constante de type objet json, les propriétés sont mutables

ES6 - Templates de chaîne

```
var personne = { prenom: "Etienne", nom: "Martin" }
let message = `

    Bonjour ${personne. prenom} ${personne.nom},

    bienvenue dans notre boutique
`


/*
Les retours charriot sont conservés:
Bonjour Etienne Martin,
bienvenue dans notre boutique
*/
```

ES6 - Classes

Déclaration

```
class Personne {
    constructor(nom, age) {
        this.nom = age
        this.vieillir(age)
    }

     vieillir(annees) {
          this.annees = this.annees || 0 + annees

     }

}

let toto = new Personne('Le Héro', 14)
```

Héritage

```
class Employee extends Personne {
    constructor(nom, age, fonction) {
        super(nom, age);
        this.fonction = fonction;
    }
}
```

ES6 - Initialisations et destructurations (1)

SOAT

Initialisation

```
function f (x, y = 5, z = 12) {

      return x + x + 12

}
f(23) === 42      // valeurs par defaut de y et z ont été utilisés
```

Paramètre REST

```
function f(guestStar, ...invites) {
      return guestStar + " et " + invites.length + "invités"

}
f("Alice", "Jean", "Boris") === "Alice et 2 invités"
```

Opérateur Spread

```
let nombre = "123456789";
let chiffres = [...nombre]; // Chiffres vaut [1,2,3,4,5,6,7,8,9]
```

ES6 - Initialisations et destructurations (2)

```
var x = [1, 2, 3, 4, 5]           // Array
var [y, z] = x                    // Affectation par décomposition
console.log(y)                    // 1
console.log(z)                    // 2
```

```
var o = {p: 42, q: true}
var {p, q} = o

console.log(p)                  // 42
console.log(q)                  // true

// Assigner de nouveaux noms
var {p: toto, q: truc} = o

console.log(toto)      // 42
console.log(truc)      // true
```

```
var toto = function(x) {
    return {x}
}

console.log( toto(12).x )        // retourne 12
```

ES6 - For … Of

```
var personnes = [
  {
    nom: "Alain Dupont",
    famille: {
      mère: "Isabelle Dupont",
      père: "Jean Dupont",
      sœur: "Laure Dupont"
    },
    âge: 35
  },
  {
    nom: "Luc Marchetoile",
    famille: {
      mère: "Patricia Marchetoile",
      père: "Antonin Marchetoile",
      frère: "Yann Marchetoile"
    },
    âge: 25
  }
]

for (var {nom: n, famille: { père: f } } of personnes) {
  console.log("Nom : " + n + ", Père : " + f)
}

// "Nom : Alain Dupont, Père : Jean Dupont"
// "Nom : Luc Marchetoile, Père : Antonin Marchetoile"
```

ES6 - Arrow functions (1)

```
let auCarre = valeur => valeur * valeur     // return implicite
auCarre(3) === 9
```

```
let incremente = valeur => {

    if (valeur <= 15)
          return valeur + 1

     }

    return valeur
}
incremente(12) === 13
```

ES6 - Arrow functions (2)

```
class MaClasse {
    constructor(x) {
         this.x = x
    }

    // La lambda garantit que this est l'instance de MaClasse

    callback = () => {
         return this.x
    }
}
```

ES6 - Promises

```
function msgAfterTimeout(msg, timeout) {
     return new Promise(resolve => {
          setTimeout(() => resolve(`-> ${msg}!`), timeout);

     })

 }

// Appel asynchrone
msgAfterTimeout("1er appel", 1000).then(() =>

     msgAfterTimeout("2eme appel", 500)).then(

          msg => console.log(`Après 1500ms ${msg}`)

)

// Affiche "Après 1500ms -> 2eme appel!"
```

ES6 - Modules

Fichier ./math.js = 1 module

```
export default function somme(x, y) { return x + y }

export function soustraction(x, y) { return x - y)

export const PI = 3.141593
```

Import global

```
import * as math from './math'
console.log('2π = ' + math.somme(math.PI, math.PI))
```

Import détaillé

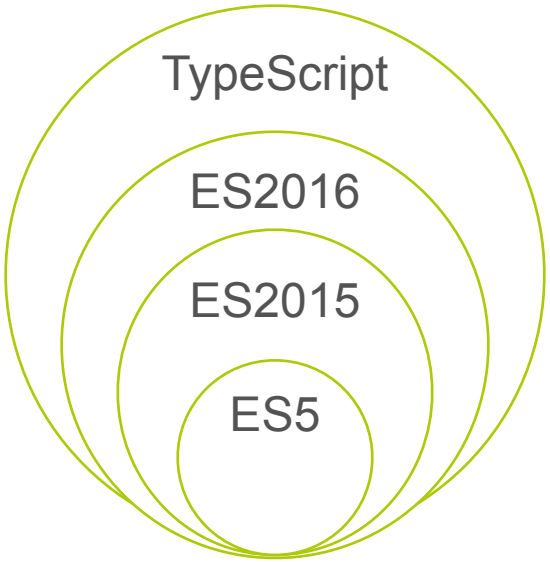
```
import somme, {soustraction} from './math'

console.log(somme(1, 2))

console.log(soustraction(2, 1))
```

ES7

- Norme en cours de conception.
- Appelée aussi ES2016.

TypeScript

- Open source, publié en octobre 2012, par **Microsoft**
- Superset de Javascript. Nécessite une "**transpilation**"
- Supporte toutes les versions de JS et apporte de **nouvelles** fonctionnalités
- **Angular 2** est développé en TypeScript : préconisé par Google



TypeScript - Décorateurs

- Annotation d'objet
- Ajout de metadata à ces objets

```
@Component({
     selector: 'app',
     providers: [NamesList],
     templateUrl: './app.html',
     directives: [RouterOutlet, RouterLink]
})
export class App {}
```

TypeScript - Portée (1)

```
export class App {
    nom                 // variable publique déclarée implicitement
    public prenom       // variable publique
    private adresse     // variable privée
    static compteur = 0 // variable statique
    protected protegee  // variable protected

    constructor(nom) {
        this.nom = nom      // construction obligatoire
        this.methode()
    }

    private methode = () => {
        // methode privée hoistée
    }

    public pubMethod() {
        // méthode publique
    }
}
```

TypeScript - Portée (2)

- **public** : variable ou méthode publique, accessible partout
- **private** : variable ou méthode privée, uniquement accessible dans l'instance de la classe
- **protected** : variable ou méthode protected, accessible dans la classe et dans les classes héritées (super())
- **static** : variable de classe indépendante des instances
- **abstract** : classe abstraite, non instanciable, pouvant être héritée

TypeScript - Types

```
// types primitifs : number, string, boolean, enum, void, null, undefined

const toto: number = 40
const tutu: string = 'literal'

enum state = { CONNECTING, CONNECTED }
if (state.CONNECTING === 'CONNECTING') {       // true
    ...
}

// any : type indéfini
const temp: any = {}
temp = 20

// Array types
let tableau: number[]       // tableau de numériques

// fonctions
let fct = (parm:number) : boolean => {
    return !!number
}
```

TypeScript - Interfaces

```
interface TypePersonne {
    nom: string,
    getAddress(): string
}

class Person implements TypePersonne {
    nom: string
    getAddress = () : string => {
           return 'adresse'
    }
}


interface Humain {
    age: number,
    taille: number
}
interface Femme extends Humain {
    maquillage: boolean
}

let monHumain: Femme = {
    age: 20,
    taille: 180,
    maquillage: true
}
```

TypeScript - TSD et TSC

- Fichiers TSD : ambiant type definition. Fichiers de définition des types d'une lib js
- TSC : programme de compilation des fichiers typescript, prenant en compte les fichiers TSD

```
/// <reference path="./typings/dom.d.ts"/>

import * from 'mylib'
```

En conclusion

- Google préconise l'utilisation de TypeScript pour Angular 2
- TypeScript ajoute de nouvelles fonctionnalités au javascript standard
- TypeScript suit et adapte les nouvelles normes JS

Installer un environnement

Objectifs du chapitre

- Utiliser NPM : Node Package Manager
- Configurer Webpack
- Installer les fichiers ts.d

Node et NPM

1. Installer Node.js

https://nodejs.org

2. Utiliser npm en ligne de commande

```
> npm init                              // Crée un fichier package.json

> npm install              // Installe le fichier package.json

> npm install library                   // Installe la librairie dans le répertoire courant

> npm install library -g  // Installe library globalement (nécessite d'être admin)

> npm install lib  --save               // Installe lib dans les dependencies du package.json

> npm install lib --save-dev            // Installe lib dans les devDependencies du package.json

> npm start                             // run du script start

> npm test                              // run du script test

> npm run deploy                        // run d'un autre script (mot clé run)
```

Fichier Package.json

```
{
  "name": "exemple",
  "version": "1.0.0",
  "description": "exemple",
  "main": "index.ts",
  "author": "ls",
  "license": "ISC",
  "dependencies": {
    "angular2": "2.0.0-beta.15",
    "mdi": "^1.5.54",
    "core-js": "^2.2.2",
    "rxjs": "5.0.0-beta.2",
    "zone.js": "~0.6.11",
    ...
  },
  "devDependencies": {
    "css-loader": "^0.23.1",
    "es6-promise": "^3.1.2",
    "es6-promise-loader": "^1.0.1",
    "es6-shim": "^0.35.0",
    "es7-reflect-metadata": "^1.6.0",
    ...
  },
  "scripts": {
    "start": "webpack-dev-server --progress --inline --content-base www/ --colors --port 9000 --watch",
    "deploy": "npm run cleandist && npm run webpack",
    "webpack": "webpack --config webpack.production.config.js",
    "cleandist": "rimraf dist/",
    "test": "karma start"
  }
}
```

Webpack - Configuration

Entry

- Import des **CSS**
- Import des **JS d'environnement** front (angular etc)

Output

- Génération d'un **bundle de dev**
- Ou génération d'un **bundle minifié de prod**

Sourcemaps

- Sources non compilées visibles sous **debug** dans les navigateurs

Webpack - Loaders

- Concaténation des CSS
- Compilation des fichiers LESS / SASS
- Compilation des fichiers Typescript
- Load des fichiers media (fontes, images…)
- postLoaders : fichiers spec et loader istanbul / reporting tests

Webpack - Plugins

HtmlWebpackPlugin

- Génération dynamique du fichier index.html

ProvidePlugin

- Déclaration de variables globales (ex jQuery)

DefinePlugin

- Utile pour définir des variables d'environnement (dev / prod)

Webpack - Scripts npm

- webpack-dev-server : Lance un server de test
- webpack : Commande de déploiement / build

Les fichiers ts.d

Installer typings

```
> npm install -g typings
```

Initialiser typings.json

```
> typings init
```

Installer des fichiers de définition

```
> typings install angular --ambient --save
```

typings.json

```
{
     "dependencies": {},
     "devDependencies": {},
     "ambientDependencies": {
       "angular": "github:DefinitelyTyped/DefinitelyTyped/angularjs/
angular.d.ts#1c4a34873c9e70cce86edd0e61c559e43dfa5f75"

     }

}
```

Dans les fichiers .ts

```
/// <reference path="./typings/browser.d.ts"/>
```

En conclusion

- NPM nous permet de gérer l'installation des dépendances de l'application
- NPM nous permet de lancer des scripts d'exécution, test et déploiement
- Webpack est un module loader (compilateur typescript, less…)
- Webpack gère un serveur de développement
- Webpack déploie un bundle applicatif

Documentation webpack : https://webpack.github.io/docs/

Bootstrap Angular

Objectifs du chapitre

- Importer les fichiers principaux
- Définir un premier composant pour l'application
- Démarrer Angular

Import des librairies et styles

./index.ts

```
/// <reference path="./typings/browser.d.ts" />

// import des styles - les fichiers less seront compilés via webpack
import './node_modules/mdi/css/materialdesignicons.min.css'
import './www/less/style.less'


// Polyfills javascript : supper es6 / es7 / zone
import 'core-js/es6'
import 'core-js/es7/reflect'
require('zone.js/dist/zone')

// Libs angular
import '@angular/common'
import '@angular/compiler'
import '@angular/core'
import '@angular/http'
import '@angular/platform-browser'
import '@angular/platform-browser-dynamic'
import '@angular/router-deprecated'

/**
*  Main App
*/
import './www/js/main'
```

Premier composant angular

./www/js/components/hello-world.component.ts

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

@Component({
  selector: 'hello-world',
  template: '<div>Hello World !</div>'
})

export default class HelloWorldComponent {}
```

Dans ./www/index-base.html

```
<body>
    <hello-world></hello-world>
</body>
```

Bootstrap angular

Dans ./www/js/main.ts

```
// import fonction bootstrap
import { bootstrap } from '@angular/platform-browser-dynamic/index'


// import du composant exporté
import HelloWorldComponent from './components/hello-world.component'


// bootstrap angular avec HelloWorldComponent
bootstrap(HelloWorldComponent, [])
```

Conclusion

- Les composants et fonctions angular sont importés dans l'app
- Bootstrap lance angular dans le composant défini en paramètre

TP 1

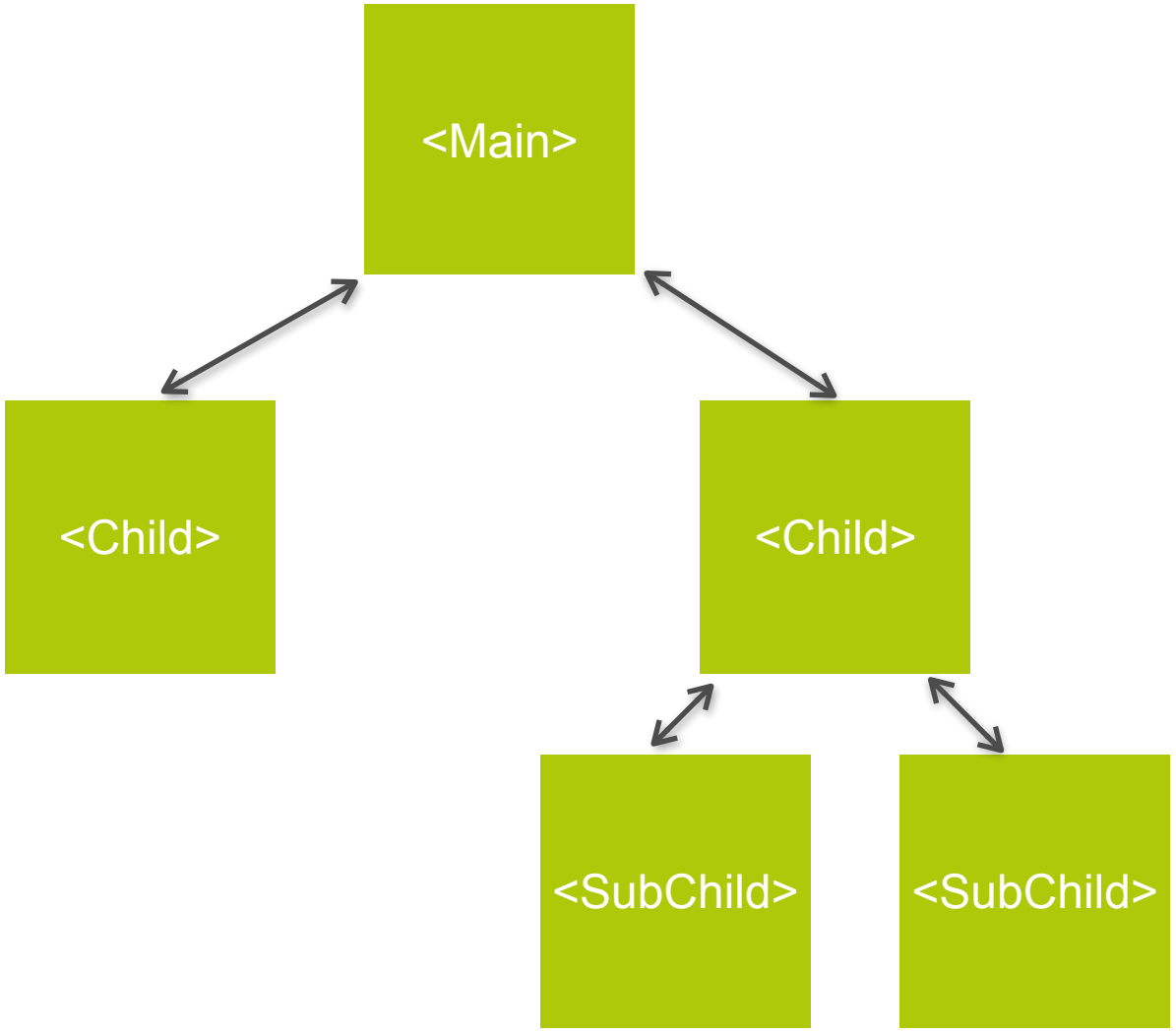
Installation, lancement de l'application

Components

Objectifs du chapitre

- Une hiérarchie de components
- Définir un component
- Manipulation des templates
- Directives angular

Hiérarchie de components



Définition des components

- Chaque component est une **classe**, associée à un **template**
- Le template est du **HTML**, auquel on ajoute des **fonctionnalités**
- Les variables et méthodes **publiques** du composant sont utilisées par le template
- Les composants incluent d'autres composants et forment une **hiérarchie**
- Les composants communiquent entre eux de manière **bi-directionnelle**
- Les **parents** envoient des **données** aux enfants
- Les **enfants** envoient des **évènements** aux parents

L'annotation @component

- @component décore une classe et la transforme en component
- Les variables et méthodes publiques sont exposées au template HTML

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

@Component({
  selector: 'my-component',                                    // nom du composant
  template: `                                                  // template
             <div>
               <h1>{{entree}}</h1>
               <p>{{node}}</p>
               <aside (click)="method('nouvelle valeur')">Click</aside>
             </div>
  `,
  inputs: ['entree']                                           // data input
})

export default class MyComponent {
  public node
  public entree
  private variablePrivee
  constructor() {
  }

  public method = (valeur) => {
    this.node = valeur
  }
}
```

```
<my-component [entree]="'initialisation'"></my-component>
```

@component : paramètres de base

selector	Identifiant du composant
template: `Hello {{nom}}` templateUrl: './my-component.html' template: require('./my-component.txt')	Template en ligne Template en fichier Template en fichier texte (webpack raw)
styles: ['.primary {color: red}'] styleUrls: ['my-component.css']	Styles en ligne Fichier css
directives: [SousComponent1, SousComponent2]	Liste des directives et component enfant du component
inputs	tableau de paramètres d'entrée
outputs	Tableau d'évènements en sortie

Syntaxe de template (1)

{{myContent}}	Interpolation de contenu
<img src="{{monImage}}">	Interpolation d'attribut
{{objet?.maProp}}	Si objet = undefined, interpolation ignorée
{{1 + 1}}	Expression calculée

Syntaxe de template (2)

<input [value]="nom">	Manipulation d'attribut
<div [class.selected]="isSelected">Item	Affectation de classe en fonction d'un booléen
<button [disabled]="isFormValid></button>	Attribut disabled en fonction d'un booléen
<img [src]="monUrl"> <img bind-src="monUrl">	Binding d'url
<p [style.color]="red"></p>	Affectation dynamique de style
<comp [valeur]="proprietePublique"></comp> <comp [valeur]="'litteral'"></comp>	**Attention** : Les littéraux doivent être passés entre côtes pour un paramètre de component. Sans côtes, angular attend une variable publique
<p [ngClass]="{selected: isTrue, invalid: isFalse}"></p>	Affectation dynamique de classes en fonction d'un booléen

Directives : *ngFor

- *ngFor permet de boucler un array
- Le contenu est répété à chaque itération
- Chaque itération est interpolable

Component

```
public items = [{nom: 'toto'}, {nom: 'titi'}, {nom: 'tutu'}]
```

Template

```
<article *ngFor="let item of items">
    {{item.nom}}
</article>
```

HTML

```
<article>toto</article>
<article>titi</article>
<article>tutu</article>
```

Directives : *ngIf

- *ngIf conditionne l'affichage du contenu en fonction d'un booléen
- Si le booléen est false, le contenu est absent du HTML

```
<p *ngIf="isTrue">
        Contenu visible
</p>

<p *ngIf="isFalse">
        Contenu non visible
</p>
```

Directives : [ngSwitch]

- Fonctionne comme un switch case javascript
- Contenu non présent si false

Template

```
<div [ngSwitch]="'literal'">
    <article *ngSwitchWhen="true">1</article>
    <article *ngSwitchWhen="'literal'">2</article>
    <article *ngSwitchDefault>autre</article>
</div>
```

HTML

```
<div>
    <article>2</article
</div>
```

Events

- Syntaxe : (event) ou on-{event}
- click, change, mouseenter, keyup, touchstart, focus, etc.

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

@Component({
  selector: 'my-component',
  template: `                                                    // template
             <div>
               <aside (click)="method('nouvelle valeur')">Click</aside> // appel méthode sur clic
             </div>
  `
})

export default class MyComponent {
  public node
  constructor() {}

  public method = (valeur) => {
    this.node = valeur
  }
}
```

https://developer.mozilla.org/en-US/docs/Web/Events

Events personnalisés / EventEmitter

- Objet angular : EventEmitter
- Evènement personnalisé : outputs

```
import {Component} from '@angular/core'
import ChildComponent from './child.component'
@Component({
  selector: 'parent-component',
  template: '<child-component (onActivate)="activate($event)"></child-component>',
  directives: [ChildComponent]
})

export default class ParentComponent {
  public onActivate = condition => {
    this.isActivated = condition
  }
}
```

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

@Component({
  selector: 'child-component',
  template: '<div (click)="setActif(true)">Activation</div>',
  outputs: ['onActivate']
})

export default class ChildComponent {
  public onActivate
  constructor() { this.onActivate = new EventEmitter() }
  setActif = condition => {
    this.onActivate.emit(condition)
  }
}
```

Annotations Input / Output

- Syntaxe alternative aux paramètres de @component
- Annotations @input et @output

```
import {Component, Input, Output, EventEmitter} from '@angular/core'

@Component({
  selector: 'parent-component',
  template: '<child-component (onActivate)="activate($event)"></child-component>'})

export default class ParentComponent {
  @Input monEntree
  @Output monAction = new EventEmitter()

  ...
}
```

Cycle de vie d'un composant - hooks

OnChanges	Exécutée lors de changements d'input
OnInit	Exécutée à l'initialisation d'un composant, après le premier OnChanges
OnDestroy	Exécutée lors de la désinstantiation d'un composant

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

@Component({
  selector: 'my-component',
  template: '<div>...</div>',
  inputs: ['entree']
})
export default class MyComponent implements OnInit, OnChanges, OnDestroy {
  public entree
  constructor() {
    ...                // avant hooks
  }
  ngOnInit() { ... }             // Initialisation
  ngOnDestroy() { ... }          // Désinstanciation
  ngOnChanges(objet) {           // A chaque changement d'input
    console.log(objet.entree.currentValue)
    console.log(objet.entree.previousValue)
  }
}
```

Conclusion

- (*) définit une modification de contenu = **directive structurelle**
- [xxx] définit un contrôle d'entrée
- (xxx) définit un contrôle de sortie

TP 2

Les premiers composants

Les Pipes

Objectifs du chapitre

- Découvrir et utiliser les pipes
- Créer ses pipes

Utiliser un pipe

- Les pipes permettent de modifier une expression dans le template
- Angular en fournit un certain nombre, on peut créer ses pipes
- Il est possible de chainer des pipes

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

@Component({
  selector: 'data-component',
  template: `
    <div>Nous sommes le {{maDate | date:"fullDate" | uppercase }}</div>
  ` // date du jour en majuscules
})

export default class DateComponent {
  maDate

  constructor() {
    this.maDate = new Date()
  }
}
```

Le pipe {date}

- expression | date[:format]
- expression de format date (objet) ou nb de millisecondes / utc

Quelques formats :

year	y (2015) yy (15)
month	MMM (Sep), MMMM (September) M (9), MM (09)
day	d (3) dd (03)
heure	H:m:s (13:3:7) HH:mm:ss (13:03:07)

https://angular.io/docs/ts/latest/api/common/DatePipe-class.html

Les pipes lowercase / uppercase

- expression | lowercase
- expression | uppercase

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

@Component({
  selector: 'x-component',
  template: `

    <p>{{texte | uppercase }}</p>      // MON TEXTE

    <p>{{texte | lowercase }}</p>      // mon texte

  `
})

export default class DateComponent {
  public texte = 'Mon Texte'
}
```

currency, percent, decimal

- expression | currency[:currencyCode[:symbolDisplay[:digitInfo]]]
- expression | percent[:digitInfo]
- expression | number[:digitInfo]
- digitInfo : {minIntegerDigits}.{minFractionDigits}-{maxFractionDigits}

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

@Component({
  selector: 'x-component',
  template: `

    <p>{{prix | currency:'Eur':true:'1.2-2' }}</p>     // €120.40

    <p>{{pourcent | percent:'1.2-2' }}</p>             // 12.00%

    <p>{{nombre | number:'3.2-2' }}</>                 // 045.80
    <p>{{nombre | number:'1.0-0' }}</>                 // 46
  `
})

export default class DateComponent {
  public prix = 120.4
  public pourcent = 0.121
  public nombre = 45.8
}
```

json, replace

- expression | json
- expression | replace:pattern:replacement

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

@Component({
  selector: 'x-component',
  template: `

    <p>{{ objet | json }}</p>     // {nom:'toto',prenom:'titi'}

    <p>{{ text | replace:'animal':'chien'}}    // Mon chien
  `
})

export default class DateComponent {
  public objet = {
    nom: 'toto'
    prenom: 'titi'
  }

  public text = 'Mon animal'
}
```

slice

- expression | slice:start[:end]

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

@Component({
  selector: 'x-component',
  template: `

    <p *ngFor="let lettre of tab | slice:1:3">{{lettre}}</p>        // <p>1</p><p>2</p>
    <p *ngFor="let lettre of [0,1,2,3] | slice:-1">{{lettre}}</p>   // <p>3</p>

    <p>{{ text | slice:0:3 }}</p>                    // <p>Mon</p>
    <p>{{ text | slice:-6:-2 }}</p>                  // <p>anim</p>

  `
})

export default class DateComponent {
  public tab = [0, 1, 2, 3]
  public text = 'Mon animal'
}
```

i18nSelect, i18nPlural

- expression | i18nSelect:mapping
- expression | i18nPlural:mapping

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

@Component({
  selector: 'x-component',
  template: `

    <p>{{ couleur | i18nSelect:couleurMap }}</p>   // <p>My blue Color</p>

    <p>{{ messages | i18nPlural:messageMap }}</p>  // <p>3 messages</p>
  `
})

export default class DateComponent {
  couleur = 'bleu'
  couleurMap = {
    bleu: 'My Blue Color',
    rouge: 'My Red Color'
  }

  messages = ['un', 'deux', 'trois']
  messageMap = {
    '=0': 'Aucun message',
    '=1': 'Un message',
    '=other': '# messages'
  }
}
```

async

- binding de valeur envoyée en asynchrone (promise etc.)
- expression | async

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

@Component({
  selector: 'message',
  template: '{{timer | async}}'
})

export default class DateComponent {
  public timer

  constructor() {
    this.timer = new Promise(function(resolve, reject) {
      setTimeout(() => {
        resolve('Hello après 10 secondes')
      }, 10000)
    })
  }
}
```

@pipe

- @pipe décore une classe, est dotée d'un nom
- pure : si false, exécuté à tout changement. si true, exécuté seulement si la data est immutable
- transform retourne la valeur calculée

```
import {Component} from '@angular/core'
import CutPipe from './cut.pipe'
@Component({
  selector: 'x-component',
  template: `
    <p>{{text | cut:2}}</>        // mo
  `,
  pipes: [CutPipe]
})

export default class DateComponent {
  text = 'mon texte'
}
```

```
import {Pipe, PipeTransform} from '@angular/core'

// pure true : chaque changement de valeur si littéral, ou référence si objet
@Pipe({name: 'cut', pure: true})
export default class CutPipe implements PipeTransform {
  transform(value, size) {
    return value.substr(0, size)
  }
}
```

En conclusion

- Les pipes permettent de **formatter** les expressions
- Attention à l'utilisation de fonctions dans les paramètres : possibilité de problèmes de **performance**, surtout si les pipes sont impures

TP 3

Multilingue avec i18nSelect

Les services

Objectifs du chapitre

- Créer un service
- Injecter un service

Pourquoi créer des services

SOAT

Problèmes

- Les components ont pour objectif de **gérer** un **template**
- Ils doivent se contenter de faire le **lien** entre le DOM et les données
- Les informations stockées dans le component sont **temporaires** : instanciations / désinstanciations fréquentes
- Les components sont indépendants les uns des autres : comment **partager** le code ou les données ?

Solutions

- Où **persister** des données ? dans des services
- Où **factoriser** du code réutilisable ? dans des services
- Les services sont des classes, de type **singleton**
- Ils sont **instanciés** soit au niveau de l'application, soir au niveau d'un component
- Les instances sont **accessibles** par les components enfants
- Ils sont liés aux components par un mécanisme d'**injection de dépendance**

Créer un service @Injectable

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

@Injectable()                                          // décoration
export default class MonService {
    public const CALCUL = 'MON CALCUL'                 // variable publique
    private maPrivate                                  // variable privée

    constructor() {
        this.maPrivate = 0
    }

    public calcul = valeur => {                        // méthode publique
        return this.maPrivate * valeur
    }

    public initialize = valeur => {
           this.maPrivate = valeur
    }
}
```

- Attention à ne pas oublier les parenthèses : **@Injectable()**

Instancier un service

- Instancié dès le lancement de l'application
- Injectable dans tous les components de l'application

```
import { bootstrap } from '@angular/platform-browser-dynamic/index'
import MonService from './mon-service'

bootstrap(AppComponent, [
  MonService
])
```

- Instancié au niveau d'un component : attribut **providers**
- Injectable dans le component et ses components enfant

```
import {Component} from '@angular/core'
import MonService from './mon-service'

@Component({
  selector: 'mon-component',
  template: '<div></div>',
  providers: [MonService]
})
```

Injecter un service dans un component

```
import {Component, OnInit} from '@angular/core'
import ChildComponent from './child.component'                    // import du ChildComponent
import MonService from './mon-service'                            // import du service

@Component({
  selector: 'parent-component',
  template: '<div></div>',
  providers: [MonService],                                        // instancie le service
  directives: [ChildComponent]
})
export default class ParentComponent implements OnInit {
    constructor(private monService: MonService) {}                // injecte le service
    ngOnInit() {
        this.monService.initialize(10)                            // utilise méthode du service
    }
}
```

```
import {Component, OnInit} from '@angular/core'
import MonService from './mon-service'                          // import du service

@Component({
  selector: 'child-component',
  template: '<div>{{monTitre}} = {{maValeur}}</div>'
})
export default class ChildComponent implements OnInit {
    public monTitre
    public maValeur
    constructor(private monService: MonService) {}              // injecte l'instance du service de ParentComponent
    ngOnInit() {
        this.monTitre = this.monService.CALCUL
        this.maValeur = this.monService.calcul(5)               // utilise le service
    }
}
```

Injecter un service dans un service

```
import {Injectable} from '@angular/core'
import MonService from './mon-service'                          // import du service

@Injectable()
export default class MonAutreService {
    constructor(private monService: MonService) {}              // injecte l'instance du service
    ...
}
```

- Attention à instancier le service injecté dans un component "parent"

OpaqueToken et provide()

- Par défaut le provider **instancie** une classe
- **provide()** permet d'utiliser d'autres types d'éléments
- **OpaqueToken** permet de créer un nouvel indentifiant : le **Token**

```
import {Component, provide, OpaqueToken} from '@angular/core'

// création des nouveaux token
const TITRE = new OpaqueToken('titre')
const MonAutreService = new OpaqueToken('autreservice')
...

@Component({
    ...,
    providers: [
        MonService,
        provide(MonAutreService, {useClass: MonAutreService}),
        provide(MonBeauService, {useExisting: MonService}),
        provide(TITRE, {useValue: 'Le titre de mon application'}),
        provide(Window, {useValue: window}),
        provide(ArticleService, {useFactory: fact(3), deps: [AutreService] })
    ]
})
export default class MaClasse {}

function fact(valeur) {
    return(autreService: AutreService) => {
        ...
    }
}
```

En conclusion

- Chaque composant possède un **Injector**
- La logique métier **doit** être développée dans les services

TP 4

Les services Toy et Translate

Le Router

Objectifs du chapitre

- Créer ses routes
- Les sous routes
- Passer des paramètres de route
- Les hooks de router

Principes du routing

- Changer la base href recharge le navigateur
- Changer la partie dynamique ne recharge pas le navigateur

- **pushState false** : HashLocationStrategy

http://www.monsite.fr/#partie-dynamique

- **pushState true** : PathLocationStrategy (default)

http://www.monsite.fr/partie-dynamique

```
<header
   <base href="/>
</header>
```

Configuration générale

- ROUTER_PROVIDERS : instancier les providers de route
- LocationStrategy : injecter le service LocationStrategy
- HashLocationStrategy : stratégie hash
- PathLocationStrategy : stratégie path, par défaut

```
import { bootstrap } from '@angular/platform-browser-dynamic/index'
import {ROUTER_PROVIDERS}  from '@angular/router-deprecated'
import {AppComponent}      from './app.component'

//
import {provide}           from '@angular/core'
import { LocationStrategy, HashLocationStrategy } from '@angular/common/index'

bootstrap(AppComponent, [
  ROUTER_PROVIDERS,
  provide(LocationStrategy, {useClass: HashLocationStrategy})
]);
```

@RouteConfig et ROUTER_DIRECTIVES

```
import {Component} from '@angular/core'
import {RouteConfig, ROUTER_DIRECTIVES} from '@angular/router-deprecated'

import ListeComponent from './liste.component'
import DetailComponent from './detail.component'

@Component({
  selector: 'app-component',
  directives: [ROUTER_DIRECTIVES],          // import des directives de Router
  template: 'app.html'
})

@RouteConfig([
  {
    path: '/liste',                         // chemin dynamique
    name: 'Liste',                          // nom de la route pour navigation
    component: ListeComponent,              // component associé à la route
    useAsDefault: true                      // route par défault
  },
  {
    path: '/detail/:id',                    // paramètre dynamique
    name: 'Detail',
    component: DetailComponent
  }
])

export default class AppContainer {}
```

Router-outlet, router-link et navigate

```
<div class="app">
    <nav>
        <a [router-link]=['Liste']>Liste</a>                // lien vers route
        <a [router-link]=['Detail', {id: 1}]>Detail</a>    // lien avec paramètre
    </nav>

    <router-outlet></router-outlet>                         // cible de la route

</div>
```

```
import {Router} from '@angular/router-deprecated'

export default class AppComponent {

    constructor(
        private router: Router
    ) {}

    goToListe() {
        this.router.navigate(['Liste'])                 // aller à la liste
    }

    goToDetail(id) {
        this.router.navigate(['Detail', {id}])          // aller au détail "id"
    }
}
```

Instanciation des components

- Accès à la route : Le component associé à la route est **instancié**
- Le component associé à la route précédente est **désinstancié**

=> Attention aux **fuites** mémoire

Récupérer les paramètres d'url

```
import {RouteParams} from '@angular/router-deprecated'

export default class DetailComponent {

    constructor(
        private routeParams: RouteParams              // injecter service RouteParams
    ) {}

    afficheDetail() {

        const monId = this.routeParams.get('id')      // récupérer le paramètre "id"

        console.log(`Mon Détail : ${monId}`)
    }
}
```

- Il est possible de passer des paramètres optionnels
- Le paramètre est passé en query string

```
this.router.navigate(['Detail', {id: 1, option: 'rouge'}])

// url appelée :
#/detail/1?option=rouge
```

Routes enfant

```
import {RouteConfig} from '@angular/router-deprecated'

@component(…)
@RouteConfig({
    path: '/parent/...',                    // sous route
    name: 'Parent',                         // nom de la route
    component: ParentComponent
})
export default class AppComponent {...}
```

```
import {RouteConfig} from '@angular/router-deprecated'

@component(...)
@RouteConfig({
    path: '/',
    name: 'Liste',
    component: ListeComponent
},
{
    path: '/:id',
    name: 'Detail',
    component: ListeComponent
})
export default class ParentComponent {...}
```

Naviguer vers les routes enfant

```
<div class="app">
    <nav>
        <a [router-link]=['Parent', 'Liste']>Liste</a>
        <a [router-link]=['Parent', 'Detail', {id: 1}]>Detail</a>
    </nav>

    <router-outlet></router-outlet>                    // cible de la route parent

</div>
```

```
<div class="parent">
    <header>
        <a [router-link]=['Detail', {id: 1}]>Detail</a>
    </header>

    <router-outlet></router-outlet>                    // cible de la route enfant

</div>
```

```
// du Parent

    goToDetail(id) {
        this.router.navigate(['Parent', 'Detail', {id}])
    }
```

Cycle de vie d'une route (1)

@CanActivate(() => { ... })class MyComponent() {}	Décorateur. Retourne un booléen ou une promise. Si false ou reject, pas d'accès à la route
routerOnActivate(current, prev) { ... }	Appelée à l'activation de la route
routerCanReuse(current, prev) { ... }	Retourne booléen ou promise Réutilisation instance ou création nouvelle instance
routerOnReuse(current, prev) { ... }	Appelé si réutilisation de l'instance
routerCanDeactivate(current, prev) { ... }	Retourne booléen ou promise Blocage du changement de route si false ou reject
routerOnDeactivate(current, prev) { ... }	Appelé à la désactivation de la route Peut attendre la résolution d'une promise

Cycle de vie d'une route (2)

```
import {CanActivate, OnActivate, OnDeactivate} from '@angular/router-deprecated'

@component(...)
@CanActivate(() => {
    return true                  // route activable
})
export default class MyComponent implements OnActivate, OnDeactivate {

    routerOnActivate() {
           console.log('route activée')
    }

    routerOnDeactivate() {
           console.log('route désactivée')
    }
}
```

En conclusion

- Le router permet de **naviguer** au sein de l'application
- Les routes permettent une **architecture** par component en profondeur
- **Attention** aux fuites mémoire aux changements de route

TP 5

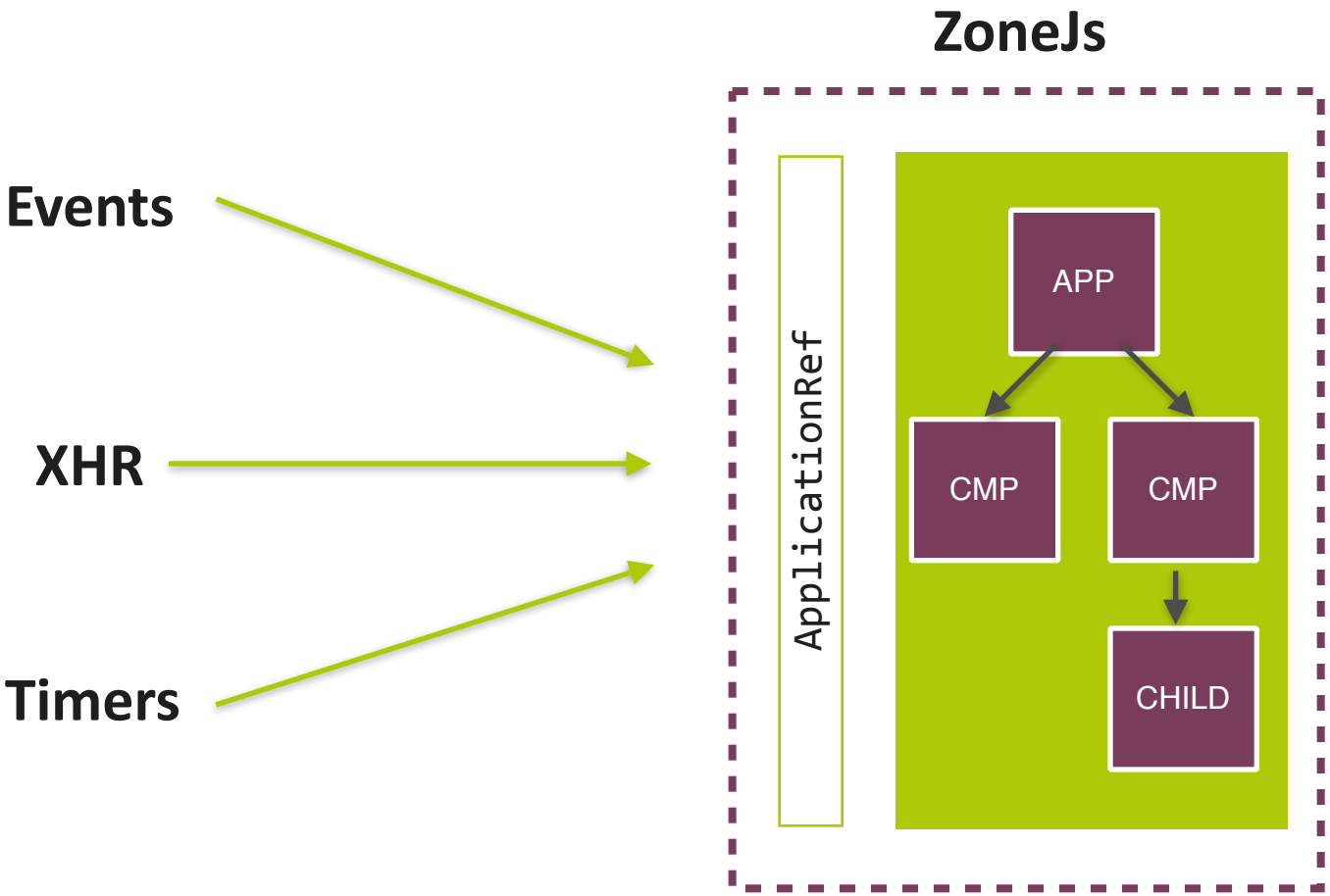
Mise en place des routes

Redux

Objectifs du chapitre

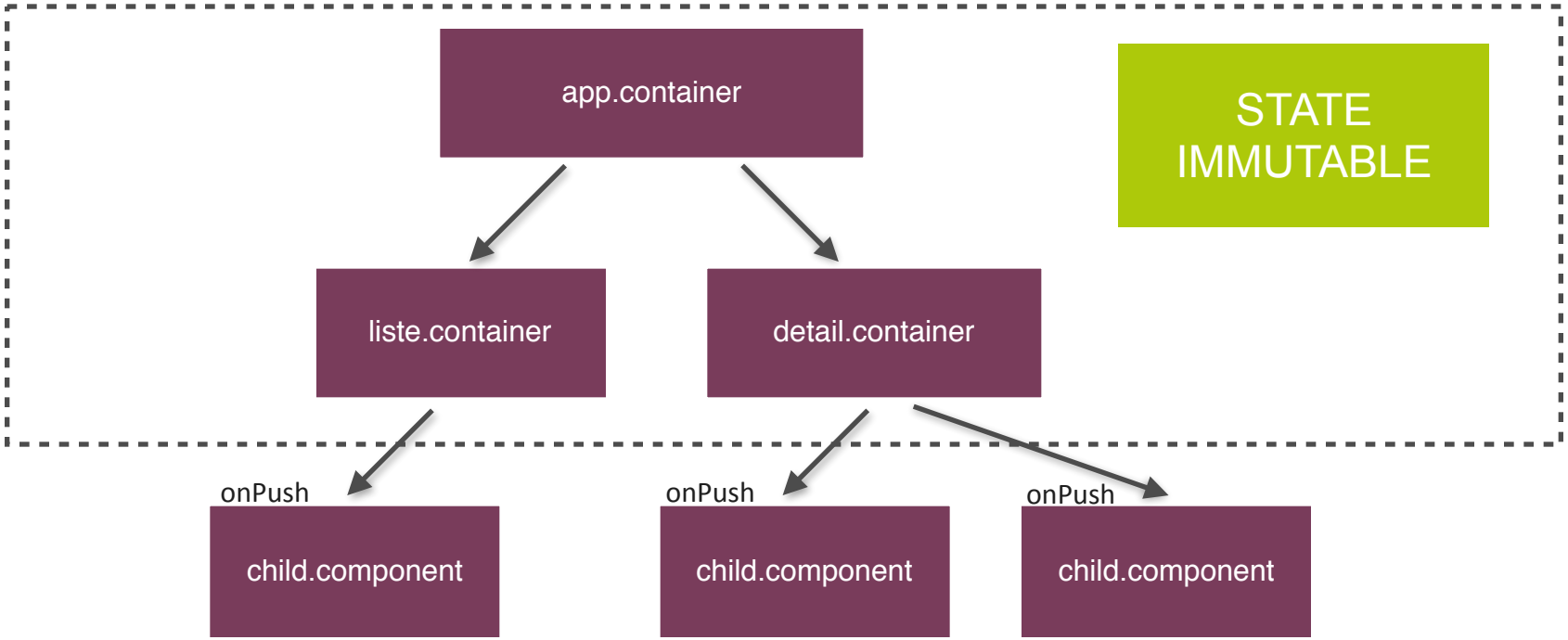
- ZoneJs : La détection du changement
- Découvrir les principes de flux unidirectionnel
- Découvrir Redux
- Utiliser ng2-redux

ZoneJs - Détection du changement



ChangeDetectionStrategy.onPush (1)

- ChangeDetectionStrategy.onPush = détecte uniquement les changements de **référence**
- => changement effectif si l'input est **immutable** (ou primitif)
- => **Containers** = composants logiques, manipulant les données, propres à l'application
- => **Components** = onPush, destinés à l'UI

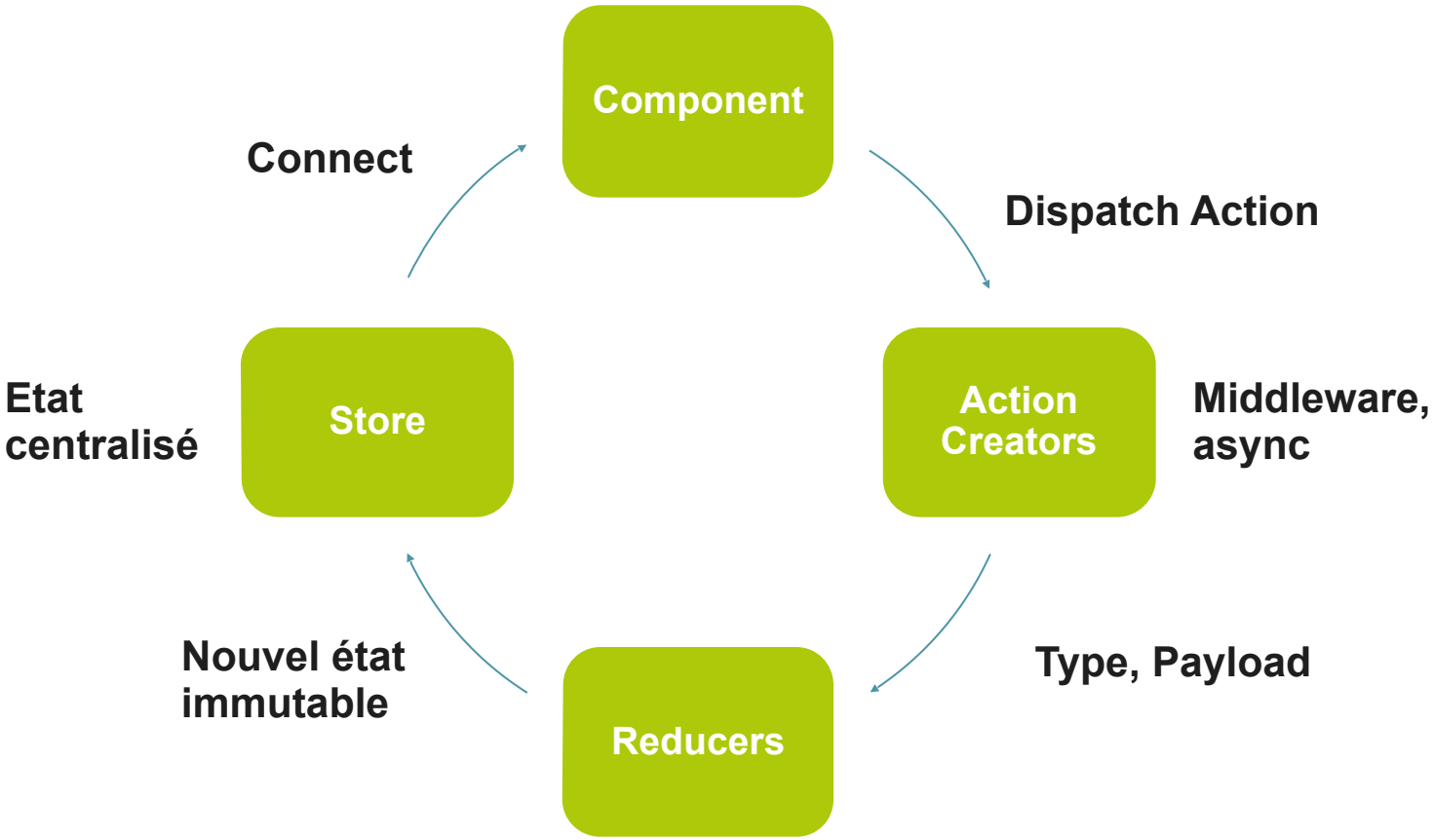


ChangeDetectionStrategy.onPush (2)

- Sans onPush, l'entrée **mutable** provoque la détection du changement à chaque nouvel event
- Input **immutable** = **performance**. Seule une nouvelle référence provoque le changement
- Communication au parent par **EventEmitter**

```
import {Component, ChangeDetectionStrategy, EventEmitter} from '@angular/core'

@Component({
  selector: 'mon-component',
  template: require('./mon.component.html'),
  inputs: ['maData'],
  outputs: ['monAction'],
  changeDetection: ChangeDetectionStrategy.OnPush
})

export default class MonComponent {
  public monAction

  constructor() {
    this.monAction = new EventEmitter()
  }

  select(data) {
    this.monAction.emit(data)
  }
}
```

Redux - flux unidirectionnel



Redux - Caractéristiques

- **Store** : Etat de toute l'application, centralisé, immutable
- **Actions** : la vue dispatche les actions utilisateur
- **Action creators** : centralisation des actions
- **Middleware** : Gestion asynchrone, lancement de fonctions avant ou après reduce
- **Reducers** : Retournent un nouvel état suite aux actions
- **Connect** : Les components souscrivent au store et se mettent à jour à chaque nouvel état

ng2-redux : implémentation de redux pour Angular 2

ngRedux - rootReducer

- Combine : combiner les reducers dans un rootReducer

```
import { combineReducers } from 'redux'

import userReducer from './user.reducer'
import autreReducer from './autre.reducer'

const rootReducer = combineReducers({
  userReducer,
  autreReducer
})

export default rootReducer
```

ngRedux - instancier le store

```
import { bootstrap } from '@angular/platform-browser-dynamic/index'

// imports redux

import { createStore, applyMiddleware } from 'redux'
import { provider } from ng2-redux
import thunk from 'redux-thunk'
import * as createLogger from 'redux-logger'


// création du store

const logger = createLogger()
const createStoreWithMiddleware = applyMiddleware(thunk, logger)(createStore);
const store = createStoreWithMiddleware(rootReducer)


// import du rootReducer

import rootReducer from './reducers/index'


// instancie le store au niveau de l'application

bootstrap(AppContainer, [
    provider(store)
])
```

ngRedux - Reducers

- Les reducers sont des fonction pures

```
import { USER } from '../actions/user.actions'

function userReducer(state: any = {}, action: any) {

  switch (action.type) {
    case USER.LOAD:
      return action.user || {}

    case USER.UPDATE:
      return action.user

    default:
      return state
  }

}

export default userReducer
```

ngRedux - Action Creators

```
import { Injectable } from '@angular/core'
import UserService from './user.service'

export const USER = {                                          // constante représentant chaque action
  LOAD_REQUEST: 'USER_LOAD_REQUEST',
  LOAD_RESPONSE: 'USER_LOAD_RESPONSE',
  LOGOUT: 'USER_LOGOUT'
}

@Injectable()
export default class UserActions {
    constructor(private userService: UserService) {}          // inject service

    login = credentials => {
        return (dispatch, getState) => {                       // thunk : retourne une fonction

            dispatch({                                         // dispatch synchone
              type: USER.LOAD_REQUEST
            })
            return this.UserService.getUser(credentials).subscribe(user => {
               dispatch({                                      // dispatch synchrone
                 type: USER.LOAD_RESPONSE,
                 user
               })
            }
        }
    }

    logout = () => {                                           // retour simple objet
      return {
        type: USER.LOGOUT
      }
    }
 }
```

ngRedux - connect

```
import { Component, Inject, OnDestroy } from '@angular/core'
import { NgRedux } from 'ng2-redux'
import UserActions from '../actions/user.actions'

@Component({
  selector: 'login-container',
  template: '...'
})
export default class LoginContainer implements OnDestroy {

  constructor(
    private ngRedux: NgRedux,
    private userActions: UserActions
  ) {}

  ngOnInit() {
    this.unsub = this.ngRedux.connect(this.mapStateToThis)(this)
  }
  ngOnDestroy() {
    this.unsub()
  }

  public login = credentials => {
    this.ngRedux.dispatch(this.userActions.login(credentials))
  }

  private mapStateToThis(state) {
    return {
      error: state.userReducer.error
    }
  }
}
```

En conclusion

- onPush optimise la **détection** de changement dans l'application
- redux adopte un flux unidirectionnel de données **immutables**
- L'état de l'application est **centralisé** en un seul endroit
- La séparation container / component permet le **partage** et la **réutilisation** de components UI

TP 6

On centralise l'état de l'application

Requêtes HTTP

Objectifs du chapitre

- Faire des requêtes au serveur avec HTTP
- Maitriser l'asynchrone : Des promises à RxJs

Http et promises (1)

- Le provider Http permet de faire des requêtes au serveur
- HTTP_PROVIDERS contient les services permettant de manipiuler http, dont Http
- Les promises permettent de gérer les appels asynchrones

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

@Component({
   ...
   providers: [HTTP_PROVIDERS]          // Instance HTTP_PROVIDERS
})

export default class ArticleComponent implements OnInit {

  public articles

  ngOnInit = () => {

    this.ArticleService.getArticles().then(          // résoulution de la promise
        res => {
           this.articles = res
        },                                           // retour en cas de succès
        error => { console.log(error) }              // retour en cas d'erreur
    )
  }
}
```

Http et promises (2)

- http.get permet de faire une requête de type GET
- then est exécuté en cas de succès
- Les erreurs sont attrapées par catch
- La méthode .json() transforme le résultat de la requête au format JSON

```
import {Injectable} from '@angular/core'
import {Http, Response} from '@angular/http'

@Injectable()
export default class ArticleService {
  constructor(public http: Http) {}                                          // injection Http

  getArticles = () => {
    return this.http.get('http://monserveur.com/api/articles')               // requête
      .toPromise()                                                           // traduction en promise
      .then((res: Response) => {
         console.log(res.status)                                             // code retour
         return res.json()                                                   // résultat
      })
      .catch(error => Promise.reject(error.message || 'Server error'))       // erreur
  }
}
```

Méthodes Http

http.get(url, options?)	Méthode GET
http.post(url, body, options?)	Méthode POST (ajout)
http.put(url, body, options?)	Méthode PUT (update)
http.delete(url, options?)	Méthode DELETE
http.patch(url, body, options?)	Méthode PATCH (update partiel)
http.head(url, options?)	Méthode HEAD (ne retourne pas de body)
http.request(url | Request, options?)	Tout type d'http request

Http RequestOptions et Headers

```
import {Injectable} from '@angular/core'
import {Http, Response, RequestOptions, Headers} from '@angular/http'

@Injectable()
export default class ArticleService {

  constructor(public http: Http) {}

  addArticle = () => {

    const headers = new Headers()
    headers.set('X-Token', 'Mon Token')

    const options = new RequestOptions({
        method: 'POST',
        body: JSON.stringify({data: 'hello'}),
        headers,
        search: 'toto=1'
    })

    return this.http.request('http://monserveur.com/api/article', options)
     ...
  }
}
```

Http Request

```
import {Injectable} from '@angular/core'
import {Http, Response, Request, RequestMethod} from '@angular/http'

@Injectable()
export default class ArticleService {

  constructor(public http: Http) {}

  getArticle = () => {

    const request = new Request({
        method: RequestMethod.Head,         // Get, Post, Put, Delete, Patch, Head
        search: 'toto=1',
        url: 'http://monserveur.com/api/article'
    })

    return this.http.request(request)
     ...
  }
}
```

Angular2 et RxJs

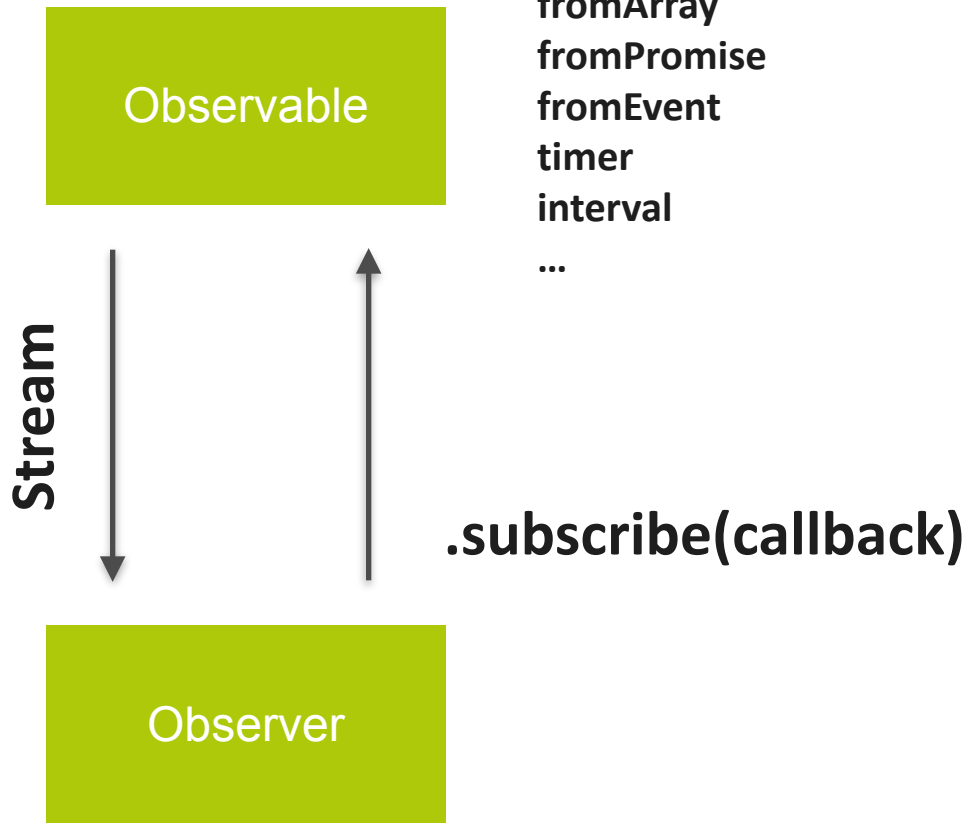
- Avec Angular2, les requêtes Http sont en fait des **Observables**
- Angular 2 dépend de RxJs : **Asynchronous Observable Pattern**
- **toPromise()** est une méthode de RxJs qui transforme un Observable en Promise

```
// RxJS
import 'rxjs/Observable'
import 'rxjs/add/operator/toPromise'
```

- On peut importer tout RxJs afin de **profiter de ses fonctionnalités**

```
// Tout RxJS
import 'rxjs/Rx'
```

Extensions RxJs : programmation fonctionnelle réactive



Http RxJs

```
import {Injectable} from '@angular/core'
import {Http, Response} from '@angular/http'

@Injectable()
export default class ArticleService {
  constructor(public http: Http) {}

  getArticles = () => {
    return this.http.get('http://monserveur.com/api/articles')                // requête
      .map(res => res.json())                                                 // stream
      .catch(error => Observable.throw(error._body || 'Server Error'))        // erreur
  }
}
```

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

@Component({
   ...
})

export default class ArticleComponent implements OnInit {

  public articles

  ngOnInit = () => {

    this.ArticleService.getArticles().subscribe(                    // souscrit à un observable
        res => { this.articles = res },                             // retour en cas de succès
        error => { console.log(error) }                             // retour en cas d'erreur
    )
  }
}
```

Jsonp (1)

- **CORS** : Cross Origin Resource Sharing, restriction d'autorisation
- **Jsonp** : Accepté par certains serveurs. Requête de type GET

```
import {Injectable} from '@angular/core'
import {Jsonp, URLSearchParams} from '@angular/http'

@Injectable()
export class WikipediaService {
  constructor(private jsonp: Jsonp) {}

  search = term => {
    let wikiUrl = 'http://en.wikipedia.org/w/api.php'

    // ?search=${term}&action=opensearch&format=json&callback=JSONP_CALLBACK
    const params = new URLSearchParams()
    params.set('search', term)
    params.set('action', 'opensearch')
    params.set('format', 'json')
    params.set('callback', 'JSONP_CALLBACK')

    return this.jsonp
               .get(wikiUrl, { search: params })
               .map(res => res.json()[1])
  }
}
```

Jsonp (2)

```
import { Component }         from '@angular/core'
import { JSONP_PROVIDERS }   from '@angular/http'
import { Observable }        from 'rxjs/Observable'
import { WikipediaService } from './wikipedia.service'

@Component({
  selector: 'my-wiki',
  template: `
    <input (keyup)="search(terme.value)">
    <article *ngFor="let item of items | async">{{item}}</article>
  `,
  providers:[JSONP_PROVIDERS, WikipediaService]
})

export class WikiComponent {
  constructor (private wikipediaService: WikipediaService) {}
  public items

  search (terme) {
    this.items = this.wikipediaService.search(terme)
  }
}
```

En conclusion

- Les requêtes doivent être faites **dans des services**, jamais dans les components
- RxJS ou Promise ? RxJs = librairie importante, **problème de poids** ?
- Plus loin avec **RxJs** : https://github.com/ReactiveX/rxjs

TP 7

Toys et langues en Http

Components, concepts avancés

Objectifs du chapitre

- Directives et manipulation du DOM
- Composants enfant

@directive

3 types de directives dans Angular2

- **Components** : directive avec un template
- Directives de type **attribut** : ex [ngClass] = changent le contenu
- Directives **structurelles** : ex *ngFor = changent la structure

@Directive de type attribut (1)

- **selector** : element, [attribut], .class, et :not()
- **ElementRef** : référence à l'élément HTML

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

@Directive({
    selector: '[myHighlight]'               // selecteur CSS
})

export default class HighlightDirective {
    constructor(private el: ElementRef) {
       el.nativeElement.style.backgroundColor = 'yellow'
    }
}
```

- Utilisation de la directive dans le template

```
@component({
    selector: 'mon-component',
    template: '<div myHightlight>Je brille</div>'
    directives: [HighlightDirective]
})
```

@Directive de type attribut (2)

- host / @HostListener : binding d'évènement
- @input('alias') variable : input de paramètre
- @input() set : fonction de transformation de l'input

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

@Directive({
    selector: '[myBorder]',
    host: {
      '(mouseenter)': 'onMouseEnter()'                // bind events
    }
})

export default class HighlightDirective {
    private width
    private elem
    @input('myBorder') color                          // alias
    @input() set myWidth(width) {                     // set
           this.width = width
    }
    constructor(private el: ElementRef) { this.elem = el.nativeElement }

    onMouseEnter() { this.highlight(true) }
    @HostListener('mouseleave') onMouseLeave() { this.highlight(false) }

    private highlight(cond) {
       this.elem.style.border = cond ? '${this.width}px solid ${this.color}' : 'none'
    }
}
```

Directive structurelle et template

- Action sur la structure du DOM
- Basé sur <template> : HTML5
- *{directive} : écriture simplifiée

```
<p *ngIf="condition">                          // avec *
  visible si condition true
</p>

<template [ngIf]="condition">                  // avec <template>
  <p>
    visible si condition true
  </p>
</template>
```

Créer une directive structurelle

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

@Directive({
    selector='[monSi]'
})

export default class SiDirective {

    constructor(
      private templateRef: TemplateRef,                // accès au template
      private viewContainer: ViewContainerRef          // référence à l'élément
    ) {}

    @Input() set mySi(condition) {
      if (condition) {
        this.viewContainer.createEmbeddedView(this.templateRef)   // créer le template
      } else {
        this.viewContainer.clear()                                // supprime le template
      }
    }

}
```

```
<p *monSi="condition">
  visible si condition true
</p>
```

Communication parent enfant par variable locale

- Lien parent enfant par l'#identifiant
- Communication par le template uniquement

```
@Component({
    selector: 'child-component'
    template: '...'
})
export default class ChildComponent {
    public action() {
        console.log('action enfant')
    }
    public message = 'Click enfant'
}
```

```
@Component({
    selector: 'parent-component'
    template: `
        <div (click)="child.action()">{{child.message}}</div>
        <child-component #child></child>                          // #child
    `,
    directives: [ChildComponent]
})
export default class ParentComponent {

}
```

ViewChild (1)

```
@Component({
    selector: 'child-component'
    template: '...'
})
export default class ChildComponent {
    public action(message) {
        console.log(message)
    }
    public message = 'Click enfant'
}
```

```
import { Component, AfterViewInit, ViewChild } from '@angular/core'

@Component({
    selector: 'parent-component'
    template: '...',
    directives: [ChildComponent]
})
export default class ParentComponent implements AfterViewInit {

    @ViewChild(ChildComponent)
    private childComponent

    onClick() {
        this.childComponent.action(this.childComponent.message)
    }

}
```

ViewChild (2)

- **@ViewChild()** permet de lier parent et enfant
- Le parent peut utiliser les **propriétés et méthodes publiques** de l'enfant
- **AfterViewInit()** : Enfant accessible en get après son initialisation
- **AfterViewChecked()** : Appelé à chaque mise à jour de l'enfant

ViewChildren et QueryList

```
@Component({
    selector: 'child-component'
    template: '...'
})
export default class ChildComponent {}
```

```
import { Component, AfterViewInit, ViewChildren, QueryList } from '@angular/core'

@Component({
    selector: 'parent-component'
    template: '<child-component *ngFor="let item in tbs"></child-component>',
    directives: [ChildComponent]
})
export default class ParentComponent implements AfterViewInit {
    public tbs = [0, 1, 2, 3]

    @ViewChildren(ChildComponent)
    private childComponents: QueryList<ChildComponent>

    ngAfterViewInit() {

            console.log(this.childComponents.length)

    }
}
```

Projection de contenu : ng-content (1)

- Le contenu du component est projeté dans ng-content

```
@Component({
    selector: 'my-component'
    template: `
        <h1>Mon titre</h1>
        <ng-content></ng-content>
    `
})
export default class ChildComponent {}
```

```
<my-component>
    <footer>Mon footer</footer>
</my-component>

Résultat :

<h1>Mon titre</h1>
<footer>Mon footer</footer>
```

Projection de contenu : ng-content (2)

- select : selecteur CSS permettant de choisir le contenu projeté

```
@Component({
    selector: 'my-component'
    template: `
        <h1>Mon titre</h1>
        <ng-content select="header"></ng-content>
        <ng-content select="footer"></ng-content>
    `
})
export default class ChildComponent {}
```

```
<my-component>
    <header>Mon header</header>
    <section>Ne sert à rien</section>
    <footer>Mon footer</footer>
</my-component>


Résultat :

<h1>Mon titre</h1>
<header>Mon header</header>
<footer>Mon footer</footer>
```

ng-content et hooks

- Avec **@ContentChild** et **@ContentChildren**, accès au content enfant
- Hooks de content : **AfterContentInit** et **AfterContentChecked**

```
@Component({
    selector: 'child-component'
    template: '<div>toto</div>'
})
export default class ChildComponent {
    methode() { ... }
}
```

```
@Component({
    selector: 'my-component'
    template: `
        <h1>Mon titre</h1>
        <ng-content select="header"></ng-content>
   `
})
export default class MyComponent {

    @ContentChild(ChildComponent)
    private childComponent

    AfterContentInit() {
         this.childComponent.methode()
    }
}
```

En conclusion

- Les **directives** permettent de personnaliser le comportement d'un élément
- **ViewChild** et **ContentChild** permettent une communication plus souple entre composants

TP 8

Des containers aux composants

Les formulaires

Objectifs du chapitre

- Créer un formulaire
- Gérer les erreurs de saisie
- Soumettre un formulaire

Balises HTML

- Utilisation des balises standard : input, textarea, select, form, button

```
<form #identification>

    <label for="user">User</label>
    <input type="text" #user>

    <label for="password>Password</label>
    <input type="password" #password>

    <label for="application">Application</label>
    <select #application>
        <option>Choix application</option>
    </select>

    <button>Envoyer</button>

</form>
```

ngModel et directives de formulaire

- [(ngModel)] : double-data binding entre template et component
- FORM_DIRECTIVES : directives nécessaires au fonctionnement du formulaire

```
import {Component} from '@angular/core'
import {FORM_DIRECTIVES} from '@angular/common/index'

@Component({
  selector: 'login-component',
  template: `

    <form #loginForm>

           <input type="text" #loginField [(ngModel)]="user.login">

    </form>

  `,
  directives: [FORM_DIRECTIVES]
})

export default class LoginComponent {

  public user = {
    login: null
  }

}
```

NgForm et submit

- NgForm es tune classe qui ajoute des fonctionnalités : #idForm="ngForm"
- (submit) permet de soumettre le formulaire

```
<form #identification="ngForm" (submit)="envoi()">

    ...

</form>
```

```
export default class LoginComponent {

  constructor(private idService) {}

  public user = {
    login: null
  }

  envoi = () => {
    this.idService.login(this.user)
  }
}
```

Validators

- Validation de la saisie
- #idField="ngForm" : Lier le champ au formulaire

```
<form #identification="ngForm" (submit)="envoi()">

  <input type="text"
         required                          // requis
         maxlength="12"                    // longueur max
         minlength="0"                     // longueur min
         pattern="[0-9]+"                  // regExp
         #nombre="ngForm">                 // champ lié au NgForm

  <select required>
    <option value="">Votre choix</option>
    <option *ngFor="let p of choix" [value]="p">{{p}}</option>
  </select>

</form>
```

Validité des champs et formulaire (1)

Propriété	Classe CSS	Description
valid	ng-valid	Champ ou form valide
invalid	ng-invalid	Champ ou form invalide
pristine	ng-pristine	Champ non modifié
dirty	ng-dirty	Champ modifié
touched	ng-touched	Perte de focus
untouched	ng-untouched	Focus non perdu

Validité des champs et formulaire (2)

```
<form #f="ngForm" (submit)="envoi()">

    <input type="text" [(ngModel)]="user.login"
        required
        #login="ngForm">

    <input type="text" [(ngModel)]="user.login"
        required
        [readonly]="login.pristine"
        #password="ngForm">

    <button [disabled]="f.form.invalid">Envoi</button>
</form>
```

```
.ng-valid[required] {
  border: 1px solid green;
}
.ng-invalid.ng-dirty[required] {
  border: 1px solid red;
}
button[disabled] {
  opacity: 0.2;
}
```

Checkbox et radio

```
<p>
   Newsletter :
   <input type="radio" name="news" (change)="user.news = 'oui'"> Oui
   <input type="radio" name="news" (change)="user.news = 'non'"> Non
   <input type="hidden"
        #news="ngForm"
        ngControl="news"
        required
        [(ngModel)]="user.news">
</p>
<p>{{user.news}}</p>

<p>
   <input type="checkbox" #check (change)="user.check = check.checked">
</p>
<p *ngIf="user.check">bam</p>
```

En conclusion

- NgForm fournit les directives de contrôle du formulaire
- Les validateurs permettent de contrôler le formulaire
- (submit) soumet la saisie

TP 9

Paiement des achats

Les tests unitaires

Objectifs du chapitre

- Configurer Karma
- Jasmine
- Stratégies de test des objets angular

Tests unitaires - Les outils

- Karma : moteur de lancement des tests et reports
- Lancement dans un browser, ou dans PhantomJS (headless browser)
- Langage d'assertion : Jasmine, Behaviour Driven Development (BDD)
- @angular : Différentes libraires de tests

Karma

package.json

```
"scripts": {
   "test": "karma start"
}

>> npm test
```

karma.conf.js

```
module.exports = function (config) {
  config.set({

    basePath: './',                     // répertoire de base des fichiers utilisés
    frameworks: ['jasmine'],            // framework de test : jasmine, mocha etc.
    files: [                            // liste des fichiers de test
      { pattern: 'spec.ts' }
    ],
    exclude: [],                        // liste des fichiers à exclure
    preprocessors: {                    // liste des preprocesseurs : compilation, coverage etc.
      'spec.ts': ['webpack'],
      'www/js/**/!(*.spec)+(.ts)': ['coverage', 'webpack']
    },
    webpack: webpackConfig,             // fichier de config webpack à importer
    reporters: ['spec', 'coverage'],    // report en console
    coverageReporter: {                 // reporting en sortie
      reporters: []
    },
   autoWatch: true,                     // surveille les modifications de fichier
   browsers: ['Chrome'],                // browsers de lancement de test. ie Chrome, PhantomJS...
   singleRun: false                     // tests en une passe ou en continu
 })
}
```

Jasmine - Assertions

- **Describe** : Bloc d'un élément à tester
- **BeforeEach** / **AfterEach** : code exécuté avant/après chaque spec.
- **BeforeAll / AfterAll** : code exécuté avant / après la suite de tests
- **It** : Bloc de spécification à tester
- **Expect** : Condition de vérification du résultat

```
describe('MonObjet', () => {

    beforeEach(() => {

    })

    it('Should do something',() => {
       expect(true).toBe(true)
    })

    it('Should add some value',() => {
       expect(false).not.toBe(true)
    })

    afterEach(() => {

    })

})
```

Jasmine : Les matchers

toBe()	Egalité stricte (ie ===)
toEqual()	Egalité (ie ==)
toMatch()	RegExp
toBeDefined() / toBeUndefined()	Défini ou undefined
toBeNull()	est null
toBeTruthy() / toBeFalsy()	truthy / falsy
toContain()	contient un élément dans un tableau
toBeLessThan() / toBeGreaterThen()	Inférieur ou supérieur
toThrow()	A lancé une exception
not	Négation (ex : not.toBe(true))

Jasmine : Les spies

- **Spy** : mock d'une méthode d'objet

spyOn	spyOn(MonObjet, 'maMethode')
.and.returnValue()	spyOn(MO, 'ma').and.returnValue('resultat')
.and.callFake()	spyOn(MO, 'ma').and.callFake(() => { return true })
.and.callThrough()	spyOn(MO, 'ma').and.callThrough()
.toHaveBeenCalled()	expect(MO.ma).toHaveBeenCalled()
.tohavebeenCalledWith()	expect(MO.ma).toHaveBeenCalledWith('une valeur')
jasmine.createSpyObj()	jasmine.createSpyObj(MO, ['methode1', 'methode2', 'methode3'])

Jasmine : Asynchrone

- beforeEach, beforeAll, afterEach, afterAll, it retournent une fonction à exécuter en fin d'appel asynchone
- done() et done.fail() stoppent l'appel asynchrone

```
describe('MonObjet', () => {

    it('Should do something', done => {

        setTimeout(() => {

            expect(doSomething).toBe(true)

            done()                                  // exécuté en fin d'async

        }, 1000)

        if (uneErreur) {
            done.fail()
        }

    })

})
```

Instancier et injecter les providers

- **beforeEachProviders** : Instancier les providers utilisés lors du test
- **inject** : Injecter les providers pour chaque test

```
import { beforeEachProviders, beforeEach, inject } from '@angular/core/testing'

import MonService from './mon-service'

describe('MonService', () => {

   let monService

   beforeEachProviders(() => [                  // instanciation des providers
     MonService
   ])

   beforeEach(inject([MonService], _serv => {   // injection du service avant tests
       monService = _serv
   })
})
```

Tester un service

- Instancier et injecter le service
- Appeler ses méthodes et vérifier leur bon fonctionnement

```
@Injectable()
export default class MonService {
   private status
   public setStatus = value => { this.status = value }
   public getStatus = () => this.status
}
```

```
import { beforeEachProviders, beforeEach, inject } from '@angular/core/testing'
import MonService from './mon-service'

describe('MonService', () => {
   let monService

   beforeEachProviders(() => [MonService])
   beforeEach(inject([MonService], _service => {
       monService = _service
   }))

   it('Should return hello') {
      monService.setStatus('hello')
      expect(monService.getStatus()).toBe('hello')
   }
})
```

Spy des providers

- utiliser **spyOn** : pour simuler l'appel d'une méthode d'un service

```
export default class MonService {
   constructor(private statusService: StatusService) {}
   public getStatus = () => { return this.statusService.getAwesomeStatus() }
}
```

```
import { beforeEachProviders, beforeEach, inject } from '@angular/core/testing'
import MonService from './mon-service'
import StatusService from './status-service'

describe('MonService', () => {
   let monService
   let statusService

   beforeEachProviders(() => [ MonService, StatusService ])

   beforeEach(inject([ MonService, StatusService ], (_mon, _status) => {
      monService = _mon
      statusService = _status
      spyOn(statusService, 'getAwesomeStatus').and.returnValue(true)
   })

   it('Should return true', () => {
      expect( statusService.getAwesomeStatus ).toHaveBeenCalled()
      expect( monService.getStatus() ).toBe(true)
   })
})
```

Mocker des providers

- créer des **fake providers :** si une unité à tester injecte un service

```
export default class StatusService {
   constructor(private autreService: AutreService) {}  // ce service n'est pas à tester ici
   ...
}
```

```
export default class MonService {
   constructor(private statusService: StatusService
   public getStatus() { return this.statusService.getAwesomeStatus() }
}
```

```
import { beforeEachProviders, beforeEach, inject } from '@angular/core/testing'
import { provide } from '@angular/core'

import MonService from './mon-service'
import StatusService from './status-service'

class MockService {}

describe('MonService', () => {

   beforeEachProviders(() => [
     MonService,
     StatusService,
     provide(AutreService, { useClass: MockService })   // utilisation d'un mock
   ])

})
```

Tester un reducer

- Fonction pure : facile à tester

```
export default function monReducer(state, action) {

    switch(action.type)

        case: MON.ACTION
            const newstate = JSON.parse(JSON.stringify(state))
            return Object.assign({}, newstate, {ajout: true})

    ...
}
```

```
import monReducer from './mon-reducer'

describe('monReducer', () => {

   it('Should return true') {
      const state = monReducer({}, { action: MON.ACTION })

      expect(state.ajout).toBe(true)
   }

})
```

Tester une action redux - Le code

```
export default class UserActions {

  constructor(private userService: UserService) {}              // service à injecter

  login = credentials => {                                      // méthode à tester
    return dispatch => {

      dispatch({ type: USER.LOGIN_REQUEST })

      this.userService.login(credentials).then(user => {        // méthode à "spyer"
         dispatch({
            type: USER.LOGIN_RESPONSE,
            user
         })
      })
    }

  }

  logout = () => {                                              // méthode à tester
    return {
        type: USER.LOGOUT
    }
  }
}
```

Tester une action - Le logout

```
import { beforeEachProviders, beforeEach, inject } from '@angular/core/testing'
import { HTTP_PROVIDERS } from '@angular/http'

import UserActions from './user.actions'
import UserService from './user.service'

describe('UserActions', () => {
    let userActions, userService

    beforeEachProviders(() => [
        HTTP_PROVIDERS,
        UserActions,
        UserService
    ])

    beforeEach(inject([ UserActions, UserService ], (_a, _s) => {
        userActions = _a
        userService = _s
   }))

    it('Should logout', () => {
        const result = userActions.logout()
        expect(result).toEqual({type: 'USER_LOGOUT'})
    })

})
```

Tester une action - Le login

```
. . .
const mockService = () => {
    return Promise.resolve({name: 'superman'})
}

describe('UserActions', () => {
    let userActions, userService
    let redux = { dispatch: () => {}}

    beforeEachProviders(() => [ HTTP_PROVIDERS, UserActions, UserService ])
    beforeEach(inject([ UserActions, UserService ], (_a, _s) => {
        userActions = _a
        userService = _s
        spyOn(userService, 'login').and.callFake(mockService)
        spyOn(redux, 'dispatch')
    }))

    it('Should login', done => {
        userActions.login('toto')(redux.dispatch).then(() => {
           expect(redux.dispatch).toHaveBeenCalledTimes(2)
           expect(redux.dispatch).toHaveBeenCalledWith({ type: USER.LOGIN_REQUEST })
           expect(userService.login).toHaveBeenCalledWith('toto')
           expect(redux.dispatch).toHaveBeenCalledWith({
              type: USER.LOGIN_RESPONSE,
              user: {name: 'superman'}
           })
           done()
        })
    })
})
```

Tester les requêtes Http

- **XHRBackend** : crée une instance de connection http
- **MockBackend** : mock http de test

```
import { beforeEachProviders, beforeEach, afterEach, inject } from '@angular/core/testing'
import { MockBackend } from '@angular/http/testing'
import { provide } from '@angular/core'
import { HTTP_PROVIDERS, XHRBackend, Response } from '@angular/http'
import MyService from './my.service'

describe('MyService', () => {
  let service, mock

  beforeEachProviders(() => [ MyService, HTTP_PROVIDERS,
     provide(XHRBackend, { useClass: MockBackend })
  ])
  beforeEach(inject([XHRBackend, Myservice], (_xhr, _serv) => {
    service = _serv
    mock = _xhr
  }))

  it('Should get some data', () => {
    let response = [{ jouet: 'Bombe Atomique' }]
    mock.connections.subscribe(connection => {
      connection.mockRespond(new Response({body: JSON.stringify(response)}))
    })
    service.getData().subscribe(data => {
      expect(data.length).toBe(1)
    })
  })
})
```

Tester un component par injection

```
import { Component, OnInit } from '@angular/core'
@Component({
   ...
})
export default class MonComponent implements OnInit {
  public variable
  ngOnInit() {
    this.variable = 'init'
  }
}
```

```
describe('MonComponent', () => {
  let mon

  beforeEachProviders(() => [ MonComponent ])

  beforeEach(inject([MonComponent], _cmp => { mon = _cmp }))

  it('Should be initialized', () => {
    mon.ngOnInit()
    expect(mon.variable).toBe('init')
  }
```

- Comment gérer la detection du changement ?
- Comment gérer les actions asynchrones ?
- => trop limité

TestComponentBuilder et async

```
import { beforeEachProviders, inject, async } from '@angular/core/testing'
import { TestComponentBuilder } from '@angular/compiler/testing'

describe('MonComponent', () => {
  let mon

  beforeEachProviders(() => [ TestComponentBuilder, MonComponent ])

  beforeEach(async(inject([TestComponentBuilder], _tcb => {
   tcb = _tcb
  })))

  it('Should change state', done => {                        // test async

    tcb.createAsync(MonComponent).then(fixture => {          // createAsync retourne promise

      const instance = fixture.componentInstance             // instance objet
      const element = fixture.nativeElement                  // element html

      instance.mon.objet = { selected: true }
      fixture.detectChanges()                                // détection du changement

      expect(element.querySelector('article').className).toBe(true)

      done()                                                 // async done
    })
  })
})
```

Tester NgRedux dans un Component (1)

- Comment tester la bonne mise à jour du message ?

```
import { Component, OnInit, OnDestroy } from '@angular/core'
import { NgRedux } from 'ng2-redux'

@Component({
  selector: 'mon-container',
  template: '<section><div>{{message}}</div></section>'
})
export default class MonContainer implements OnInit, OnDestroy {
  public message
  private unsub

  constructor(
    private ngRedux: NgRedux
  ) {}

  ngOnInit() {
    this.unsub = this.ngRedux.connect(this.mapStateToThis)(this)
  }
  ngOnDestroy() {
    this.unsub()
  }

  private mapStateToThis(state) {
    return {
      message: state.messageReducer                    // action type: MESSAGE.SEND
    }
  }
}
```

Tester NgRedux dans un Component (2)

```
import { beforeEachProviders, inject, async } from '@angular/core/testing'
import { TestComponentBuilder } from '@angular/compiler/testing'
import MonContainer from './header.container'
import { MESSAGE } from './actions/message.actions'

import { NgRedux } from 'ng2-redux'
import store from '../helpers/redux.helper' // construit le store

describe('MonContainer', () => {
  let redux, tcb, monContainer

  beforeEachProviders(() => [ TestComponentBuilder, MonContainer, store() ])

  beforeEach(async(inject([TestComponentBuilder, MonContainer, NgRedux], (t, m, r) => {
    tcb = t
    monContainer = m
    redux = r
  })))

  it('Should send a message', done => {
    return tcb
      .overrideTemplate(MonContainer, '<div>{{message}}</div>')              // simplifie le template cible
      .createAsync(MonContainer).then(fixture => {                           // createAsync component

          const instance = fixture.componentInstance                         // récup instance
          redux.dispatch({ type: MESSAGE.SEND, message: 'Le Message' })      // dispatch message

          fixture.detectChanges()                                            // changement
          expect(instance.message).toBe('Le Message')                        // test

          done()                                                             // fin async

    }).catch(e => done.fail(e))                                              // en cas d'erreur
  })
})
```

Tester un pipe

- Exécuter la méthode **transform()** du pipe

```
import { Pipe, PipeTransform } from '@angular/core'
@Pipe({
  name: 'upper'
})
export default class UpperPipe implements PipeTransform {
  transform(value) {
    return value.toUpperCase()
  }
}
```

```
import { beforeEachProviders, inject } from '@angular/core/testing'
import UpperPipe from '../pipes/upper.pipe'

describe('UpperPipe', () => {
 let pipe

  beforeEachProviders(() => [ UpperPipe ])

  beforeEach(inject([UpperPipe], p => {
    pipe = p
  }))

  it('Should return an uppercase', () => {
    expect(pipe.transform('')).toEqual('')
    expect(pipe.transform('hello')).toEqual('HELLO')
  })
})
```

Tester un EventEmitter

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

@Component({
  selector: 'mon-component',
  template: '...',
  outputs: ['onUpdate']
})
export default class MonComponent implements OnInit {
  public onUpdate

  constructor() {
    this.onUpdate = new EventEmitter()
  }

  update(data) {
    this.onUpdate.emit(data)
  }
}
```

```
it('Should update', done => {
    monComponent.update('bouh')
    monComponent.onUpdate.subscribe(res => {
      expect(res).toBe('bouh')
      done()
    })
})
```

Tester les routes

DEPRECATED

```
describe('Test router', () => {
  var location, router

  beforeEachProviders(() => [
    RouteRegistry,                          // service contenant définition des routes
    provide(Location, {useClass: SpyLocation}),                  // spy du service Location
    provide(Router, {useClass: RootRouter}),
    provide(ROUTER_PRIMARY_COMPONENT, {useValue: App})           // map du RouteConfig
  ])

  beforeEach(inject([Router, Location], (r, l) => {
    router = r                  // service router
    location = l                // service location
  }))

  it('Should navigate to Home', done => {
    router.navigate(['Home']).then(() => {            // navigue vers Home
      expect(location.path()).toBe('/home')           // l'url doit être /home
      done()
    }).catch(e => done.fail(e));
  })

  it('should redirect to Home', done => {
    router.navigateByUrl('/bidon').then(() => {       // navigue une url bidon
      expect(location.path()).toBe('/home')           // redirection vers /home
      done()
    }).catch(e => done.fail(e));
  })
})
```

En conclusion

- Les tests assurent un code de qualité et sans erreur
- Les tests permettent le contrôle des non regressions
- Les tests sont une documentation de spécifications

TP 10

Des tests !

Projets et liens

Angular2 de tous les côtés

Angular2 universal : Server Side Rendering

https://github.com/angular/universal

Lancer Angular 2 dans un web worker

https://github.com/alexpods/angular2-universal-starter

Angular2 meteor : Angular full-stack

http://www.angular-meteor.com/

Angular2 côté mobile

Ionic 2 = Angular2 + cordova + natif

http://ionic.io/2

NativeScript + Angular2

https://www.nativescript.org/

Ressources

https://angular.io/

http://angularjs.blogspot.fr/

http://ngmodules.org/modules?query=angular2

https://github.com/angular/material2

http://redux.js.org/index.html

https://github.com/angular-redux/ng2-redux

https://github.com/ngrx/store

https://github.com/ReactiveX/rxjs

http://rxmarbles.com/

FIN