

Formation Angular

Romain Bohdanowicz

Twitter: @bioub

http://formation.tech/



Introduction

Présentations



- Romain Bohdanowicz
 Ingénieur EFREI 2008, spécialité en Ingénierie Logicielle
- Expérience
 Formateur/Développeur Freelance depuis 2006
 Plus de 10000 heures de formation animées
- Langages

Expert: HTML / CSS / JavaScript / PHP / Java Notions: C / C++ / Objective-C / C# / Python / Bash / Batch

- CertificationsPHP 5 / PHP 5.3 / PHP 5.5 / Zend Framework 1
- Particularités
 Premier site web à 12 ans (HTML/JS/PHP), Triathlète à mes heures perdues
- Et vous ?
 Langages ? Expérience ? Utilité de cette formation ?



ECMAScript 6

ECMAScript 6 - Introduction



- ECMAScript 6, aussi connu sous le nom ECMAScript 2015 ou ES6 est la plus grosse évolution du langage depuis sa création (juin 2015)
 http://www.ecma-international.org/ecma-262/6.0/
- Le langage est enfin adapté à des application JS complexes (modules, promesses, portées de blocks...)
- Pour découvrir les nouveautés d'ECMAScript 2015 / ES6 http://es6-features.org/

ECMAScript 6 - Compatibilité



- Compatibilité (novembre 2016) :
 - Dernière version de Chrome/Opera, Edge, Firefox, Safari : ~ 90%
 - Node.js 6 et 7 : ~ 90% d'ES6
 - Internet Explorer 11 : ~ 10% d'ES6
- Pour connaître la compatibilité des moteurs JS : http://kangax.github.io/compat-table/
- Pour développer dès aujourd'hui en ES6 et exécuter le code sur des moteurs plus anciens on peut utiliser des :
 - Compilateurs ou transpilateurs : Babel, Traceur, TypeScript... Transforment la syntaxe ES6 en ES5
 - Bibliothèques de polyfills : core-js, es6-shim, es7-shim... Recréent les méthodes manquante en JS

ECMAScript 6 - Portées de bloc



- let
 - · On peut remplacer le mot-clé var, par let et obtenir ainsi une portée de bloc
 - · La portée de bloc ainsi créée peut devenir une closure

```
for (var globalI=0; globalI<3; globalI++) {}
console.log(typeof globalI); // number

for (let i=0; i<3; i++) {}
console.log(typeof i); // undefined

// In 1s : 0 1 2
for (let i=0; i<3; i++) {
    setTimeout(() => {
        console.log(i);
      }, 10000);
}
```

ECMAScript 6 - Constantes



Constantes

- Il est désormais possible de créer des constantes
- Comme pour let, les variables déclarées via const on une portée de bloc
- Bonne pratique, utiliser const ou bien let lorsque ce n'est pas possible (plus jamais var)

```
if (true) {
  const PI = 3.14;
}

console.log(typeof PI); // undefined

const hello = function() {};

// SyntaxError: Identifier 'hello' has already been declared
  const hello = function() {};
```

ECMAScript 6 - Template literal



- Template literal / Template string
 - Permet de créer une chaine de caractères à partir de variables ou d'expressions
 - Permet de créer des chaines de caractères multi-lignes
 - Déclarée avec un backquote ` (rarement utilisé dans une chaine)

ECMAScript 6 - Fonctions fléchées



- Arrow Functions
 - Plus courtes à écrire : (params) => retour.
 - Si un seul paramètre, les parenthèses des paramètres sont optionnelles.
 - · Si le retour est un objet, les parenthèses du retour sont obligatoires.

```
const sum = (a, b) \Rightarrow a + b;
const hello = name => `Hello ${name}`;
const getCoords = (x, y) \Rightarrow (\{x: x, y: y\});
// ES5
// var sum = function (a, b) {
// return a + b;
// };
// var hello = function (name) {
// return 'Hello ' + name;
// };
// var getCoords = function (x, y) {
// return {
// x: x,
// y: y,
```

ECMAScript 6 - Fonctions fléchées



- Avec bloc d'instructions
 - Si les fonctions nécessitent plusieurs lignes, on peut utiliser un bloc { }
 - Le mot clé return devient alors obligatoire

```
const isWon = (nbGiven, nbToGuess) => {
  if (nbGiven < nbToGuess) {
    return 'Too low';
  }
  if (nbGiven > nbToGuess) {
    return 'Too high';
  }
  return 'Won !';
};
```

ECMAScript 6 - Fonctions fléchées



- Bonnes pratiques
 - Attention à ne pas utiliser les fonctions fléchées pour déclarer des méthodes!
 - Utiliser les fonctions fléchées pour les callback ou les fonctions hors objets
 - Utiliser les method properties pour les méthodes
 - Utiliser class pour les fonctions constructeurs

```
const globalThis = this;

const contact = {
    firstName: 'Romain',
    method1: () => { // Mauvaise pratique
        console.log(this === globalThis); // true
    },
    method2() { // Bonne pratique
        console.log(this === contact); // true
    }
};

contact.method1();
contact.method2();
```

ECMAScript 6 - Default Params



- Paramètres par défaut
 - Les paramètres d'entrées peuvent maintenant recevoir une valeur par défaut

```
const sum = function(a, b, c = 0) {
   return a + b + c;
};

console.log(sum(1, 2, 3)); // 6
console.log(sum(1, 2)); // 3

// ES5
// var sum = function(a, b, c) {
// if (c == undefined) {
// c = 0;
// }
// return a + b + c;
// };
```

ECMAScript 6 - Rest Parameters



- Paramètres restants
 - Pour récupérer les valeurs non déclarées d'une fonction on peut utiliser le REST Params
 - Remplace la variable arguments (qui n'existe pas dans une fonction fléchée)
 - La variable créé est un tableau (contrairement à arguments)
 - Bonne pratique : ne plus utiliser arguments

```
const sum = (a, b, ...others) => {
  let result = a + b;

  others.forEach(nb => result += nb);

  return result;
};
console.log(sum(1, 2, 3, 4)); // 10

const sumShort = (...n) => n.reduce((a, b) => a + b);
console.log(sumShort(1, 2, 3, 4)); // 10
```

ECMAScript 6 - Spread Operator



- Spread Operator
 - · Le Spread Operator permet de transformer un tableau en une liste de valeurs.

```
const sum = (a, b, c, d) => a + b + c + d;

const nbs = [2, 3, 4, 5];
console.log(sum(...nbs)); // 14

// ES5 :
    // console.log(sum(nbs[0], nbs[1], nbs[2], nbs[3]));

const otherNbs = [1, ...nbs, 6];
console.log(otherNbs.join(', ')); // 1, 2, 3, 4, 5, 6

// ES5 :
    // const otherNbs = [1, nbs[0], nbs[1], nbs[2], nbs[3], 6];

// Clone an array
const cloned = [...nbs];
```

ECMAScript 6 - Shorthand property



- Shorthand property
 - Lorsque l'on affecte une variable à une propriété (maVar: maVar), il suffit de déclarer la propriété

```
const x = 10;
const y = 20;

const coords = {
    x,
    y,
    };

// ES5
// const coords = {
    //    x:    x,
    //    y:    y,
    // };
```

ECMAScript 6 - Method properties



- Method properties
 - Syntaxe simplifiée pour déclarer des méthodes

```
const maths = {
   sum(a, b) {
     return a + b;
   }
};

console.log(maths.sum(1, 2)); // 3

// ES5
// const maths = {
   // sum: function(a, b) {
     // return a + b;
   // }
```

ECMAScript 6 - Computed Property Names



Computed Property Names
 Permet d'utiliser une expression en nom de propriété

```
let i = 0;
const users = {
  [`user${++i}`]: { firstName: 'Romain' },
  [`user${++i}`]: { firstName: 'Steven' },
};
console.log(users.user1); // { firstName: 'Romain' }
/* ES5
var i = 0;
var users = {};
users['user ' + (++i)] = { firstName: 'Romain' };
users['user ' + (++i)] = { firstName: 'Steven' };
console.log(users.user1); // { firstName: 'Romain' }
*/
```

ECMAScript 6 - Array Destructuring



- Déstructurer un tableau
 - · Permet de déclarer des variables recevant directement une valeur d'un tableau

```
// [1 , 2 , 3 ];
const [one, two, three] = [1, 2, 3];
console.log(one); // 1
console.log(two); // 2
console.log(three); // 3

// ES5
// var tmp = [1, 2, 3];
// var one = tmp[0];
// var two = tmp[1];
// var three = tmp[2];
```

ECMAScript 6 - Array Destructuring



- Déstructurer un tableau
 - Il est possible de ne pas déclarer un variable pour chaque valeur
 - Il est possible d'utiliser une valeur par défaut
 - Il est possible d'utiliser le REST Params

```
const [one, , three = 3] = [1, 2];
console.log(one); // 1
console.log(three); // 3

const [romain, ...others] = ['Romain', 'Jean', 'Eric'];
console.log(romain); // Romain
console.log(others.join(', ')); // Jean, Eric
```

ECMAScript 6 - Object Destructuring



- Déstructurer un object
 - Comme pour les tableaux il est possible de déclarer une variable recevant directement une propriété

```
// {x: 10 , y: 20 }
const {x: varX, y: varY} = {x: 10, y: 20};
console.log(varX); // 10
console.log(varY); // 20
```

ECMAScript 6 - Object Destructuring



- Déstructurer un object
 - Il est possible de nommer sa variable comme la propriété et d'utiliser shorthand property
 - Il est possible d'utiliser une valeur par défaut

```
const {x: x , y , z = 30} = {x: 10, y: 20};
console.log(x); // 10
console.log(y); // 20
console.log(z); // 30
```

ECMAScript 6 - Mot clé class



- Simplifie la déclaration de fonction constructeur
- Les classes n'existent pas pour autant en JavaScript, ce n'est qu'une syntaxe simplifiée (sucre syntaxique)
- Le contenu d'une classe est en mode strict

```
class Person {
  constructor(firstName) {
    this.firstName = firstName;
 hello() {
    return `Hello my name is ${this.firstName}`;
const instructor = new Person('Romain');
console.log(instructor.hello()); // Hello my name is Romain
// ES5
// var Person = function(firstName) {
    this.firstName = firstName;
// };
// Person.prototype.hello = function() {
// return 'Hello my name is ' + this.firstName;
// };
```

ECMAScript 6 - Mot clé class



- Héritage avec le mot clé class
 - Utilisation du mot clé extends pour l'héritage
 - Utilisation de super pour appeler la fonction constructeur parent et les accès au méthodes parents si redéclarée dans la classe

```
class Instructor extends Person {
  constructor(firstName, speciality) {
    super(firstName);
    this.speciality = speciality;
  }
  hello() {
    return `${super.hello()}, my speciality is ${this.speciality}`;
  }
}

const romain = new Instructor('Romain', 'JavaScript');
console.log(romain.hello()); // Hello my name is Romain, my speciality is
JavaScript
```



Modules ECMAScript

Modules ECMAScript - Introduction



- JavaScript à sa conception
 - Objectif: créer des interactions côté client, après chargement de la page
 - Exemples de l'époque :
 - Menu en rollover (image ou couleur de fond qui change au survol)
 - Validation de formulaire
- JavaScript aujourd'hui
 - Applications front-end, back-end, en ligne de commande, de bureau, mobiles...
 - Applications pouvant contenir plusieurs centaines de milliers de lignes de codes (Front-end de Facebook > 1 000 000 LOC)
 - Il faut faciliter le travail collaboratif, en plusieurs fichiers et en limitant les risques de conflit

Modules ECMAScript - Introduction

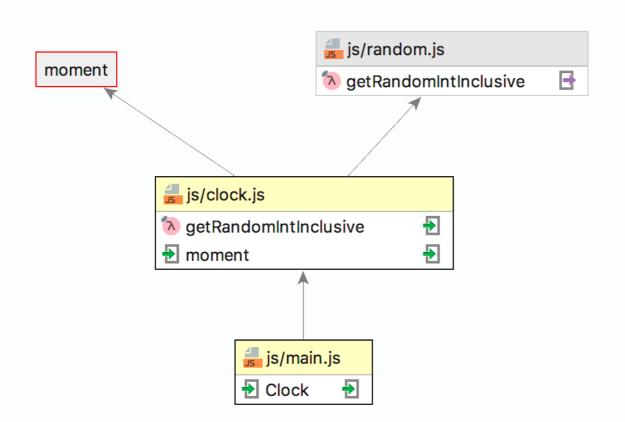


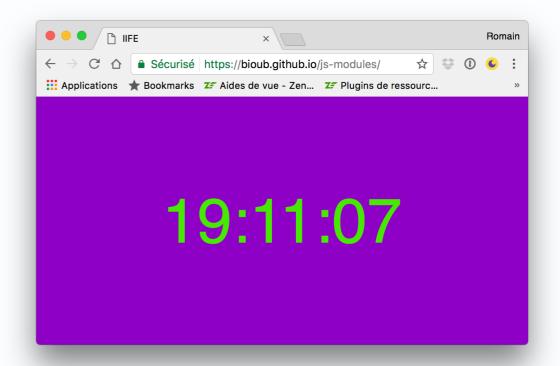
- Objectifs d'un module JavaScript
 - Créer une portée au niveau du fichier
 - Permettre l'export et l'import d'identifiants (variables, fonctions...) entre ces fichiers qui auront désormais leur propre portée
- Principaux systèmes existants
 - IIFE / Function Wrapper
 - CommonJS
 - AMD
 - UMD
 - SystemJS
 - ES6 (statiques mots clés import / export)
 - ESNext : import() (fonction asynchrone)

Modules ECMAScript - Introduction



Exemple utilisé pour la suite





- Le point d'entrée de l'application est le fichier main.js, qui dépend de Clock défini dans le fichiers clock.js, qui dépend lui même de getRandomIntInclusive du fichier random.js et moment définit dans le projet Open Source Moment.js
- Exemples: https://github.com/bioub/js-modules/
- Démo : https://bioub.github.io/js-modules/

Module ECMAScript - Concepts



Portée de modules

- Sans module la portée d'une fonction ou d'une variable déclarée dans un fichier serait globale.
- Avec les modules une fonction sera locale au fichier.

Import / Export

- Une fonction ou un objet pouvant servir dans un autre fichier il faudra l'exporter.
- Cela va créer l'API public du fichier (accessible de l'extérieur).
- Un autre fichier devra importer les fonctions utilisées

Mode Strict

 Les modules ECMAScript sont par défaut en mode strict, il n'est donc pas nécessaire d'écrire 'use strict'; en début de fichier.

Module ECMAScript - Export



Pour exporter une variable ou une fonction on utilise le mot clé export

```
export const getRandom = function() {
  return Math.random();
};
export const getRandomArbitrary = function(min, max) {
  return Math.random() * (max - min) + min;
};
export const getRandomInt = function(min, max) {
 min = Math.ceil(min);
 max = Math.floor(max);
  return Math.floor(Math.random() * (max - min)) + min;
};
export const getRandomIntInclusive = function(min, max) {
 min = Math.ceil(min);
 max = Math.floor(max);
  return Math.floor(Math.random() * (max - min + 1)) + min;
};
```

Module ECMAScript - Export



Il est également possible d'exporter en une seule fois en fin de fichier

```
const getRandom = function() {
  return Math.random();
};
const getRandomArbitrary = function(min, max) {
  return Math.random() * (max - min) + min;
};
const getRandomInt = function(min, max) {
 min = Math.ceil(min);
 max = Math.floor(max);
  return Math.floor(Math.random() * (max - min)) + min;
};
const getRandomIntInclusive = function(min, max) {
 min = Math.ceil(min);
 max = Math.floor(max);
  return Math.floor(Math.random() * (max - min + 1)) + min;
};
export { getRandom, getRandomArbitrary, getRandomInt, getRandomIntInclusive };
```

Module ECMAScript - Import



- Pour importer on utilise le mot clé import, associé à des accolades et le nom du fichier (l'extension est optionnelle)
- Lorsque que le fichier fait partie du projet, il est obligatoire de préfixer le fichier par ./ ou ../
- Les modules ECMAScript ne peuvent être importée que statiquement en début de fichier. Pour des imports dynamiques il faut utiliser les modules CommonJS ou Dynamic Import (ESNext)

```
import { getRandomIntInclusive } from './random';

class Clock {
    // ...

    update() {
        let r = getRandomIntInclusive(0, 255);
        let g = getRandomIntInclusive(0, 255);
        let b = getRandomIntInclusive(0, 255);
        // ...
    }

    // ...
}
```

Module ECMAScript - Tree Shaking



- Les imports étant statiques, des bundlers (bibliothèques de build) comme webpack ou Rollup peuvent analyser le code et éliminer du build les exports non importés
- Le build final ressemblera ainsi à :

```
const getRandomIntInclusive = function(min, max) {
 min = Math.ceil(min);
 max = Math.floor(max);
  return Math.floor(Math.random() * (max - min + 1)) + min;
};
class Clock {
 // ...
 update() {
    let r = getRandomIntInclusive(0, 255);
    let g = getRandomIntInclusive(0, 255);
    let b = getRandomIntInclusive(0, 255);
```

Module ECMAScript - Export/Import par défaut



- Il est possible de définir un export par défaut lorsqu'on a qu'une seule valeur à importer ou une valeur principale à importer
- Pour exporter on ajoute le mot clé default

```
export default class Clock {
   // ...
}
```

Pour importer il faudra ne pas utiliser d'accolades

```
import Clock from './clock';
let clockElt = document.querySelector('.clock');
let clock = new Clock(clockElt);
clock.start();
```

 Certains développeurs conseillent d'éviter les exports par défaut : https://basarat.gitbooks.io/typescript/docs/tips/defaultIsBad.html

Module ECMAScript - Imports avancés



 On peut renommer un import, par exemple dans le cas où 2 identifiants auraient le même nom :

```
import { render as renderDom } from 'react-dom';
import { App } from './App';
renderDom(<App />, document.getElementById('root'));
```

On peut également importer tous les exports dans un objet :

```
// serviceWorker.js
export function register(config) {
    // ...
}
export function unregister() {
    // ...
}
```

```
import * as serviceWorker from './serviceWorker';
serviceWorker.unregister();
```



TypeScript

TypeScript - Introduction



- TypeScript : JavaScript + Typage statique
 - TypeScript est un langage créé par Microsoft, construit comme un sur-ensemble d'ECMAScript
 - Pour pouvoir exécuter le code il faut le transformer en JavaScript avec un compilateur
 - A quelques exceptions près et selon la configuration, le JavaScript est valide en TypeScript
 - Le principal intérêt de TypeScript est l'ajout d'un typage statique

TypeScript - Installation



- Installation
 - npm install -g typescript
- Création d'un fichier de configuration
 - tsc --init
- Compilation
 - tsc



Le principal intérêt de TypeScript est l'introduction d'un typage statique

```
const lastName: string = 'Bohdanowicz';
const age: number = 32;
const isTrainer: boolean = true;
```

- Types basiques:
 - boolean
 - number
 - string



- Avantages
 - Complétion

Détection des erreurs

```
const firstName: string = 'Romain';

function hello(firstName: string): string {
  return `Hello ${firstName}`;
}

hello({
  firstName: 'Romain',
});
```



Tableaux

```
const firstNames: string[] = ['Romain', 'Edouard'];
const colors: Array<string> = ['blue', 'white', 'red'];
```

Tuples

```
const email: [string, boolean] = ['romain.bohdanowicz@gmail.com', true];
```

Enum

```
enum Choice {Yes, No, Maybe}

const c1: Choice = Choice.Yes;
const choiceName: string = Choice[1];
```

Never

```
function error(message: string): never {
  throw new Error(message);
}
```



Any

```
let anyType: any = 12;
anyType = "now a string string";
anyType = false;
anyType = {
  firstName: 'Romain'
};
```

Void

```
function withoutReturn(): void {
  console.log('Do someting')
}
```

Null et undefined

```
let u: undefined = undefined;
let n: null = null;
```

TypeScript - Assertion de type



Le compilateur ne peut pas toujours déterminer le type adéquat :

```
const formElt = document.querySelector('form.myForm');
const url = formElt.action; // error TS2339: Property 'action' does not exist on
type 'Element'.
```

Il faut alors lui préciser, 3 syntaxes possibles

```
let formElt = <HTMLFormElement> document.querySelector('form.myForm');
const url = formElt.action;
```

```
let formElt = document.querySelector<HTMLFormElement>('form.myForm');
const url = formElt.action;
```

```
let formElt = document.querySelector('form.myForm') as HTMLFormElement;
const url = formElt.action;
```

TypeScript - Inférence de type



TypeScript peut parfois déterminer automatiquement le type :

```
const title = 'First Names';
console.log(title.toUpperCase());

const names = ['Romain', 'Edouard'];
for (let n of names) {
   console.log(n.toUpperCase());
}
```

TypeScript - Interfaces



- Pour documenter un objet on utilise une interface
 - Anonyme

```
function helloInterface(contact: {firstName: string}) {
  console.log(`Hello ${contact.firstName.toUpperCase()}`);
}
```

Nommée

```
interface ContactInterface {
   firstName: string;
}

function helloNamedInterface(contact: ContactInterface) {
   console.log(`Hello ${contact.firstName.toUpperCase()}`);
}
```

TypeScript - Interfaces



- Les propriétés peuvent être :
 - optionnelles (ici lastName)
 - en lecture seule, après l'initialisation (ici age)
 - non déclarées (avec les crochets)

```
interface ContactInterface {
    firstName: string;
    lastName?: string;
    readonly age: number;
    [propName: string]: any;
}

function helloNamedInterface(contact: ContactInterface) {
    console.log(`Hello ${contact.firstName.toUpperCase()}`);
}
```

TypeScript - Classes



- Quelques différences avec JavaScript sur le mot clé class
 - On doit déclarer les propriétés
 - On peut définir une visibilité pour chaque membre : public, private, protected

```
class Contact {
  private firstName: string;

  constructor(firstName: string) {
    this.firstName = firstName;
  }

  hello(): string {
    return `Hello my name is ${this.firstName}`;
  }
}

const romain = new Contact('Romain');
console.log(romain.hello()); // Hello my name is Romain
```

TypeScript - Classes



- Une classe peut
 - Hériter d'une autre classe (comme en JS)
 - Implémenter une interface
 - Être utilisée comme type

```
interface Writable {
  write(data: string): void;
}

class FileLogger implements Writable {
  write(data: string): Writable {
    console.log(`Write ${data}`);
    return this;
  }
}
```

TypeScript - Génériques



Permet de paramétrer le type de certaines méthodes

```
class Stack<T> {
  private data: Array<T> = [];
  push(val: T) {
    this.data.push(val);
  pop(): T {
    return this.data.pop();
  peek(): T {
    return this.data[this.data.length - 1];
const strStack = new Stack<string>();
strStack.push('html');
strStack.push('body');
strStack.push('h1');
console.log(strStack.peek().toUpperCase()); // H1
console.log(strStack.pop().toUpperCase()); // H1
console.log(strStack.peek().toUpperCase()); // BODY
```

TypeScript - Décorateurs



- Permettent l'ajout de fonctionnalités aux classes ou membre d'une classe en annotant plutôt que via du code à l'utilisation
- Norme à l'étude en JavaScript par le TC39
 https://github.com/tc39/proposal-decorators
- Supporté de manière expérimentale en TypeScript
- Pour activer leur support il faut éditer le tsconfig.json ou passer une option au compilateur

```
{
  "compilerOptions": {
    "target": "es5",
    "experimentalDecorators": true
  }
}
```

TypeScript - Décorateurs



Décorateur de classes

```
'use strict';
function Freeze(obj) {
  Object.freeze(obj);
@Freeze
class MyMaths {
  static sum(a, b) {
    return Number(a) + Number(b);
try {
  MyMaths['substract'] = function(a, b) {
    return a - b;
  };
catch(err) {
 // Cannot add property substract, object is not extensible
  console.log(err.message);
```

TypeScript - Décorateurs



Décorateur de propriétés

```
import 'reflect-metadata';
const minLengthMetadataKey = Symbol("minLength");
function MinLength(length: number) {
  return Reflect.metadata(minLengthMetadataKey, length);
function validateMinLength(target: any, propertyKey: string): boolean {
  const length = Reflect.getMetadata(minLengthMetadataKey, target, propertyKey);
  return target[propertyKey].length >= length;
class Contact {
 @MinLength(7)
  protected firstName;
  constructor(firstName: string) {
   this.firstName = firstName;
  isValid(): boolean {
    return validateMinLength(this, 'firstName');
const romain = new Contact('Romain');
console.log(romain.isValid()); // false
```



RxJS

RxJS - Introduction



- Vers le milieu des années 2000, des bibliothèques comme Bluebird se créent et implémentent le concept de promesse (API Promise)
- L'API Promise devient natif en 2015 avec ES6
- Les promesses permettent de simplifier l'écriture de code de code asynchrone mais ne fonctionnent que pour les callbacks asynchrones appelées une seule fois (setTimeout, requête AJAX, accès à la plupart des bases de données et des systèmes de fichier...)
- Pour répondre aux besoins des callbacks asynchrone appelées plusieurs fois on pourrait remplacer les promesses par des Observables (setInterval, WebSockets, Workers, Evénements Souris / Clavier, Changement d'URL, Bindings...)
- RxJS est une bibliothèque qui implémente le concept tout en ajoutant des fonctionnalités supplémentaires (operators, subjects...)
- ▶ L'API Observable est dans le processus pour faire partie de la norme JavaScript

RxJS - Promise vs Observable



Création et déclenchement d'une Promise (native ECMAScript)

```
function timeout(delay) {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve(delay);
    }, delay);
  });
}
timeout(1000)
  .then((delay) => console.log(delay + 'ms'));
```

```
$ node timeout.js
1000ms
```

RxJS - Promise vs Observable



Création et déclenchement d'une Observable (avec RxJS)

```
function interval(delay) {
  return new Observable((observer) => {
    const intervalId = setInterval(() => {
      observer.next(delay);
    }, delay);
    return () => {
      clearInterval(intervalId);
    };
  });
const interval$ = interval(1000);
const subscription = interval$
  subscribe((delay) => console.log(delay + 'ms'));
setTimeout(() => {
  subscription.unsubscribe();
}, 4500);
```

```
$ node interval.js
1000ms
1000ms
1000ms
1000ms
```

RxJS - Promise vs Observable



Différences

	Promise (ECMAScript)	Promise (Bluebird)	Observable (RxJS)
Déclenchement du code asynchrone	Au moment de l'appel	Au moment de l'appel	Au moment de la souscription
Nombre d'appels asynchrones	Un	Un	Plusieurs
Etats	succès (then), erreur (catch)	succès (then), erreur (catch)	succès (next), erreur (error), fin (complete)
Annulation	A implémenter	Prévue	Prévue
Combinaisons / Transformations prévues	3 (Promise.all, Promise.race, Promise.allSettled)	~ 20 méthodes de création ou transformation	~130 méthodes de création ou transformation



```
import { Component, OnInit, EventEmitter } from '@angular/core';
import { HttpClient } from '@angular/common/http'
import { of } from 'rxjs/observable/of';
import { catchError, debounceTime, distinctUntilChanged, filter, map, switchMap, tap } from 'rxjs/
operators';
@Component({
  selector: 'my-app',
  templateUrl: './app.component.html',
export class AppComponent implements OnInit {
  public users$;
  public selectedUser;
  public loading;
  public typeahead = new EventEmitter<string>();
  constructor(private httpClient: HttpClient) {}
  ngOnInit() {
   this.users$ = this.typeahead.pipe(
      filter((term) => term.length >= 3),
      distinctUntilChanged(),
      debounceTime(200),
      tap(() => this.loading = true),
      switchMap(
        (term) => this.httpClient.get<any>(`https://api.github.com/search/users?q=${term}`).pipe(
          catchError(() => of({items: []})),
          map(rsp => rsp.items),
          tap(() => this.loading = false),
```



- Dans l'exemple précédent, typeahead est un observable qui fournit dans le temps les valeurs saisies dans un champ
- filter
 L'observable résultant n'émettra une valeur que si la condition est vérifiée

distinctUntilChanged
 L'observable résultant n'émettra une valeur que si elle diffère de la précédente



debounceTime

L'observable résultant n'émettra une valeur que s'il se produit une pause de n millisecondes entre 2 valeurs

tap

L'observable est identique au précédent, le callback est appelé pour chaque valeur mais n'influe pas sur l'observable (side-effect)

(affiche également 5 fois 'Hello' dans la console)



map

L'observable résultant verra ses valeurs transformées par le callback de map

switchMap

Combine 2 observables, le second étant créé à partir d'une valeur provenant du premier

(les requêtes 2 et 3 sont annulées car la 3 a commencée avant le retour de la seconde et la 4 avant le retour de la 3)

RxJS - Allez plus loin



What's with the Subjects in RxJS 5
 https://samvloeberghs.be/posts/whats-with-the-subjects-in-rxjs5



Zone.js

Zone.js - Introduction



- Angular inclus une bibliothèque développée par Google appeler Zone.js
- Zone.js permet d'intercepter automatiquement des callbacks asynchrone et d'exécuter du code avant ou après
- Angular s'en sert principalement dans 3 contextes :
 - Lancer son algo de détection de changement après une requête AJAX, un changement de route ou toute autre opérations asynchrone
 - Intercepter les erreurs pouvant se produire dans les callbacks asynchrones
 - Lors des tests automatisés, marquer la fin du tests à l'issue de l'exécution du code synchrone et asynchrone
- Il est possible de gagner en performance en supprimant Zone.js et en lançant la détection de changement manuellement
- Installation npm install zone.js

Zone.js - Exemples



Exécuter du code après le dernier callback asynchrone

```
require('zone.js');

const myZone = Zone.current.fork({
   onHasTask(delegate, current, target, hasTaskState) {
     if (!hasTaskState.microTask && !hasTaskState.macroTask) {
        console.log('DONE');
     }
   },
});

myZone.run(() => {
   setTimeout(() => {
      console.log('setTimeout 1');
   }, Math.floor(Math.random() * 1001));

setTimeout(() => {
      console.log('setTimeout 2');
   }, Math.floor(Math.random() * 1001));
});
```

```
setTimeout 1
setTimeout 2
DONE
```

Zone.js - Exemples



Exécuter du code avant ou après chaque callback asynchrone

```
require('zone.js');
const myZone = Zone.current.fork({
  onInvokeTask: (parentZoneDelegate, currentZone, targetZone, task) => {
    console.log('async call');
    return parentZoneDelegate.invokeTask(targetZone, task);
   // angular -> $digest
   // dt.detectChanges();
});
myZone.run(() => {
  setTimeout(() => {
   console.log('setTimeout 500ms');
 }, 500);
  setTimeout(() => {
   console.log('setTimeout 800ms');
 }, 800);
});
```

```
async call
setTimeout 500ms
async call
setTimeout 800ms
```

Zone.js - Exemples



Intercepter les erreurs dans les callbacks asynchrones

```
require('zone.js');
const myZone = Zone.current.fork({
  onHandleError: (parentZoneDelegate, currentZone, targetZone, error) => {
    console.log(error.message);
});
myZone.run(() => {
  setTimeout(() => {
    throw new Error('Error in async callback : setTimeout 300ms');
  }, 300);
  setTimeout(() => {
    console.log('setTimeout 500ms');
  }, 500);
  setTimeout(() => {
    throw new Error('Error in async callback : setTimeout 800ms');
  }, 800);
});
```

```
Error in async callback : setTimeout 300ms
setTimeout 500ms
Error in async callback : setTimeout 800ms
```



Angular CLI

Angular CLI - Introduction



- Angular introduit un programme en ligne de commande permettant d'interagir avec l'application :
 - créer un projet
 - builder
 - lancer le serveur de dev
 - générer du code
 - générer les fichiers de langue
 - •

Angular CLI - Introduction



- Installation
 - npm install -g @angular/cli
- Documentation
 - https://cli.angular.io/
 - https://github.com/angular/angular-cli/wiki
 - ng help
 - ng help COMMANDE

Angular CLI - Création d'un projet



- Création d'un projetng new CHEMIN_VERS_MON_PROJET
- Autres options
 - --skip-commit: ne fait pas de commit initial
 - --routing: créer un module pour les routes (Single Page Application)
 - --prefix : change le préfixe des composant (par défaut app)
 - --style: change type de fichier CSS (css par défault ou sass, scss, less, stylus)
 - --service-worker: ajouter un service worker pour le mode hors-ligne
- Créer un projet sans installer
 npx -p @angular/cli ng new CHEMIN_VERS_MON_PROJET

Angular CLI - Squelette



- angular.json
 Fichier de configuration du programme ng, permet de renommer des répertoires, des fichiers
- e2e
 Test End to End (qui pilotent le navigateur)
- src/appLe code source de l'application
- tsconfig.json
 Configuration du compilateur TypeScript
- tslint.json
 Configuration des conventions de code

```
-- README.md
                             angular.json
— e2e
                                    protractor.conf.js
                                                                  src

    — app.e2e-spec.ts

— app.po.ts

                                  node_modules
                              package.json
                             src
                                                                   app

    □ app.component.css

    — app.component.html

    □ app.component.ts

                                                                      — app.module.ts
                                     ├─ assets
                                   ── browserslist
                                  ─ environments
                                                                     ─ environment.prod.ts
                                                                     — environment.ts
                                    ├─ favicon.ico

    index.html
    index.html

                                  ─ karma.conf.js
                                   ├─ main.ts
                                  ─ polyfills.ts

    ─ styles.css

                                   ├─ test.ts

    tsconfig.app.json
    in the second control of the second
                                 tsconfig.json
```

Angular CLI - Squelette



- src/assets
 Les fichiers statiques non-buildés (images...)
- src/browserslist
 Les navigateurs ciblés (pour autoprefixer)
- src/environmentsConfiguration de l'application
- src/index.html src/main.ts
 Points d'entrées de l'application
- src/polyfills.tsChargement des polyfills (core-js, ...)
- src/style.cssCSS global
- src/karma.conf.js src/test.ts
 Configuration des tests

```
— README.md
               angular.json
     -- e2e
                         protractor.conf.js
                   ├─ src

→ app.e2e-spec.ts

— app.po.ts

                   node_modules
               package.json
           - src
                                     app

    □ app.component.css

    — app.component.html

    □ app.component.ts

                                      — app.module.ts
                         — assets
                            browserslist
                   ─ environments
                                      ─ environment.prod.ts
                                      — environment.ts
                       — favicon.ico
                         — index.html
                   ─ karma.conf.js
                   ├─ main.ts
                   ─ polyfills.ts
                  ├─ styles.css
                         — test.ts

    tsconfig.app.json
    tsconfig.app.json
    in the second content of the second
                  tsconfig.json
```

Angular CLI - Build



- Compiler l'application Angular
 - ng build
- Options intéressantes :
 - --prod: minifie le code avec UglifyJS et active les options --aot, -environment=prod, --extract-css, --build-optimizer...
 - --environment=NOM: permet de charger un fichier de configuration particulier (staging, test...)
 - --vendor-chunk: pour que le code de node_modules soit dans un fichier séparé

Angular CLI - Build



Gains d'un build avec --prod

```
Angular 4 : ng build
Date: 2017-11-02T09:02:41.042Z
Hash: 1d2842c3e0ac46a944f0
Time: 6349ms
chunk {inline} inline.bundle.js, inline.bundle.js.map (inline) 5.83 kB [entry] [rendered]
chunk {main} main.bundle.js, main.bundle.js.map (main) 18.1 kB {vendor} [initial] [rendered]
chunk {polyfills} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 199 kB {inline} [initial] [rendered]
chunk {styles} styles.bundle.js, styles.bundle.js.map (styles) 11.3 kB {inline} [initial] [rendered]
chunk {vendor} vendor.bundle.js, vendor.bundle.js.map (vendor) 1.98 MB [initial] [rendered]
Angular 5 : ng build
Date: 2017-11-02T09:07:47.401Z
Hash: d1a929eaad03e8e746bb
Time: 4937ms
chunk {inline} inline.bundle.js, inline.bundle.js.map (inline) 5.83 kB [entry] [rendered]
chunk {main} main.bundle.js, main.bundle.js.map (main) 17.9 kB [initial] [rendered]
chunk {polyfills} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 199 kB [initial] [rendered]
chunk {styles} styles.bundle.js, styles.bundle.js.map (styles) 11.3 kB [initial] [rendered]
chunk {vendor} vendor.bundle.is, vendor.bundle.is.map (vendor) 2.29 MB [initial] [rendered]
Angular 4: ng build --prod
Date: 2017-11-02T09:03:29.639Z
Hash: cb067f695303856c2315
Time: 6228ms
chunk {0} polyfills.14173651b8ae6311a4b5.bundle.js (polyfills) 61.4 kB {4} [initial] [rendered]
chunk {1} main.f5677287cea9969f6fb6.bundle.js (main) 8.39 kB {3} [initial] [rendered]
chunk {2} styles.d41d8cd98f00b204e980.bundle.css (styles) 0 bytes {4} [initial] [rendered]
chunk {3} vendor.43700a281455e3959c70.bundle.js (vendor) 217 kB [initial] [rendered]
chunk {4} inline.6b5a62abf05dcccf24d7.bundle.js (inline) 1.45 kB [entry] [rendered]
Angular 5 : ng build --prod --vendor-chunk
Date: 2017-11-02T09:10:58.444Z
Hash: cf4dd52226e15e33c748
Time: 11487ms
chunk {0} polyfills.ad37cd45a71cb38eee76.bundle.js(polyfills) 61.1 kB [initial] [rendered]
chunk {1} main.2c7fbf970f7125d9617e.bundle.js (main) 7.03 kB [initial] [rendered]
chunk {2} styles.d41d8cd98f00b204e980.bundle.css (styles) 0 bytes [initial] [rendered]
chunk {3} vendor.719fe92af8c44a7e3dac.bundle.js (vendor) 167 kB [initial] [rendered]
chunk {4} inline.220ce59355d1cb2bcb28.bundle.js (inline) 1.45 kB [entry] [rendered]
```

Angular CLI - Serveur de développement



- Lancer le serveur de dev
 - ng serve
- Options intéressantes :
 - --port: changer le port
 - --target=production: sert les fichiers dans la config de prod

Angular CLI - Générateurs



- Générateurs
 - Angular CLI contient un certains nombre de générateurs : application, class, component, directive, enum, guard, interface, module, pipe, service, universal, appShell
- Dry run
 Chaque générateur peut se lancer avec l'option --dry-run ou -d qui va afficher le résultat de la commande sans rien créer, sachant qu'il n'y a pas de retour automatique possible une fois les fichiers créés.
- Afficher la doc d'un générateur
 - ng help generate NOM_DU_GENERATEUR

Angular CLI - Générateurs



- Générer un module
 - ng generate module CHEMIN_DEPUIS_APP
 - ng g m CHEMIN_DEPUIS_APP
- Autres options
 - --routing: génère un 2e module pour les routes
 - --flat: ne créé pas de répertoire

Angular CLI - Générateurs



- Générer un composant
 - ng generate component CHEMIN_DEPUIS_APP
 - ng g c CHEMIN_DEPUIS_APP
- Autres options
 - --flat : ne créé pas de répertoire
 - --export : ajoute une entrée dans les exports du module

Angular CLI - Tests & lint



- Lancer les tests Karma + Jasmine
 - ng test
- Lancer les tests Protractor
 - ng e2e
- Vérifier les conventions de code
 - ng lint
 - ng lint --fix --type-check



Composants

Composants - Introduction



- 2 parties
 - code TypeScript
 - template
- Compilation
 - Les 2 sont compilés dans un code optimisé pour la VM JavaScript
 - Le template peut être compilé en JIT (par le browser) ou en AOT (au moment du build)

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

@Component({
    selector: 'my-hello',
    template: 'Hello {{name}}',
})
export class HelloComponent {
    public name = 'Romain';
}
```

Composants - Lifecycle Hooks



- Les composants peuvent implémenter les interfaces et leurs méthodes suivantes seront appelées automatiquement :
 - OnChanges / ngOnChanges(): lorsque qu'un changement se produit au niveau d'un input binding.
 - OnInit / ngOnInit() : une fois que le composant reçoit ses propriétés @Input et affiche les premiers input bindings.
 - OnDestroy / ngOnDestroy(): juste avant la destruction du component/directive.
 Il faut s'y désabonner des Observable ou événement pour éviter les fuites mémoires.
 - DoCheck / ngDoCheck() à chaque lancement de la détection de changement, permet d'identifier des changements que ngOnChanges ne peut détecter.

Composants - Lifecycle Hooks



- ngAfterContentInit()
 - Respond after Angular projects external content into the component's view / the view that a
 directive is in.
 - Called once after the first ngDoCheck().
- ngAfterContentChecked()
 - Respond after Angular checks the content projected into the directive/component.
 - Called after the ngAfterContentInit() and every subsequent ngDoCheck().
- AfterViewInit / ngAfterViewInit()
 - Respond after Angular initializes the component's views and child views / the view that a directive
 is in.
 - Called once after the first ngAfterContentChecked().
- AfterViewChecked / ngAfterViewChecked()
 - Respond after Angular checks the component's views and child views / the view that a directive is
 in.
 - Called after the ngAfterViewInit and every subsequent ngAfterContentChecked().

Composants - Lifecycle Hooks



Exemple

```
import { Component, OnDestroy, OnInit } from '@angular/core';
@Component({
  selector: 'hello-lifecycle',
  template: `
    {{ now | date: 'HH:mm:ss' }}
export class LifecycleComponent implements OnInit, OnDestroy {
 public now = new Date();
 private intervalId: number;
  ngOnInit() {
    this.intervalId = setInterval(() => {
      this.now = new Date();
    }, 1000)
  ngOnDestroy() {
    clearInterval(this.intervalId);
```



Templates

Templates - Introduction



Templates

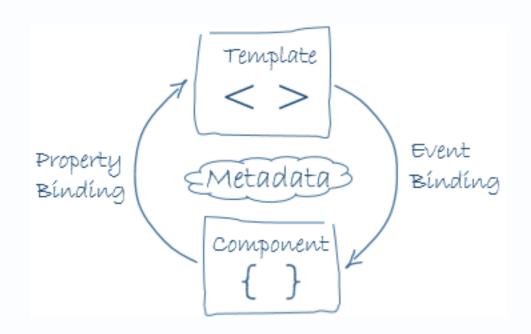
- Comme dans AngularJS, on décrit l'interface de manière déclarative dans des templates
- Chaque template est compilé par le compilateur d'Angular, soit en amont (mode AOT pour Ahead Of Time Compilation), soit dans le browser (mode JIT pour Just In Time Compilation)
- Les templates sont ainsi transformé en du code optimisé pour la VM/Moteur JavaScript

Templates - Data binding



Data binding

- Sans data binding ce serait au développeur de maintenir les changements à opérer sur le DOM à chaque événement
- Dans jQuery par exemple, cliquer sur un bouton peut avoir pour conséquence de rafraîchir une balise, de lancer un indicateur de chargement...
- Avec Angular le développeur décrit l'état du DOM en fonction de propriétés qui constitue le Modèle, ainsi un événement n'a plus qu'



```
[property] = "value"

(event) = "handler"

[(ng-model)] = "property"
```

Templates - Property Binding



Invité

| HelloAngular

C i localhost:4200

 2 syntaxes
 Pour synchroniser le DOM avec le modèle (les propriétés publiques du composant dans Angular)

bind-nomDeLaPropDuDOM="propDuComposant"

[nomDeLaPropDuDOM]="propDuComposant"

Templates - Expressions



 Dans un property binding il est possible d'utiliser des noms de propriétés ou des expressions, sauf les expressions ayant des effets de bords :

```
affectations (=, +=, -=, ...)
   new
                                                                HelloAngular
                                                                                      Invité
                                                              C i localhost:4200
                                                                                     ⊕ :
   expressions chainées avec ; ou ,
                                                         3 + 4 = 7

    incrementation et décrémentation (++ et --)

                                                         Hello Romain!
import { Component } from '@angular/core';
@Component({
selector: 'hello-prenom',
template:
  <div>3 + 4 = <span [textContent]="3 + 4"></span></div>
  <div>Hello <span [textContent]="prenom + '!'"></span></div>
export class PrenomComponent {
public prenom = 'Romain';
```

Templates - Interpolation



- Interpolation
 - Plutôt que bind-innerHTML sur une balise span, on peut utiliser la syntaxe aux doubles accolades {{ }}
 - A privilégier car cette syntaxe échappe les entrées, évitant ainsi que des balises contenues dans les entrées se retrouvent dans le DOM (faille XSS)

```
import { Component } from '@angular/core';
                                                                        HelloAngular
@Component({
                                                                  ← → C (i) localhost:4200
                                                                                          ⊕ :
  selector: 'hello-interpolation',
                                                                  Hello Romain
  template:
                                                                  Hello Romain
<div>Hello <span [innerHTML]="prenom"></span></div>
                                                                  Hello <script>alert("XSS")</script>
<div>Hello {{prenom}}</div>
<div>Hello {{xssAttack}}</div>
})
export class InterpolationComponent {
  public prenom = 'Romain';
  public xssAttack = '<script>alert("XSS")</script>';
```

Templates - Event Binding



Invité

C i localhost:4200

Prénom: Roma

Hello Roma

2 syntaxes

Pour synchroniser le DOM avec le modèle (les propriétés publiques du composant

dans Angular), on utilise des événements

- on-nomEvent="expression"
- (nomEvent)="expression"

```
import { Component } from '@angular/core';
@Component({
  selector: 'hello-event-binding',
  template:
<div>Prénom : <input on-input="updatePrenom($event)"></div>
<div>Prénom : <input (input)="updatePrenom($event)"></div>
<div>Prénom : <input (input)="prenom = $event.target.value"></div>
<div>Hello {{prenom}}</div>
})
export class EventBindingComponent {
  public prenom = '';
 public updatePrenom(e) {
this.prenom = e.target.value;
```

Templates - Two way data-binding

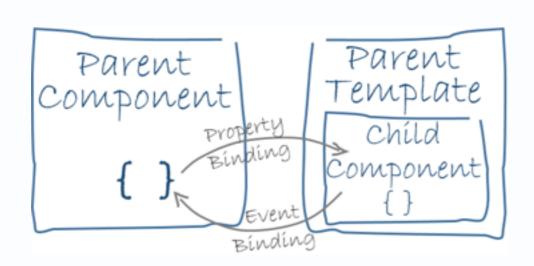


- AngularJS proposait des bindings dans les 2 sens (two way data-binding)
- Angular propose une syntaxe similaire, mais en réalité le binding se fait toujours en 2 temps, d'abord les events bindings, puis les property bindings
- Pour utiliser les 2 sur une ligne il faut :
 - Avoir un property binding
 - Avoir un event binding du même nom suffixé par change et qui affecte \$event à la variable lié au property binding
- Exemple
 - [ngModel]="prenom" (ngModelChange)="prenom = \$event"
 - [(ngModel)]="prenom"
- Pour moyen mnémotechnique : [()] = BANANA IN A BOX

Templates - Communication inter-composant



- Pour communiquer entre un composant parent et un composant enfant (imbriqués l'un dans l'autre) on utilise des bindings
- Pour passer des valeurs on utilise des property bindings, la propriété du composant doit alors utiliser le décorateur @Input
- Pour remonter des valeurs au parent on utilise des event bindings, la propriété du composant doit alors utiliser le décorateur @Output et doit être de type EventEmitter



Templates - Communication inter-composant



Exemple extrait de ng-select : https://github.com/ng-select/ng-select

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

@Component({
    selector: 'ng-select'
})
export class NgSelectComponent {

@Input() items: any[] = [];

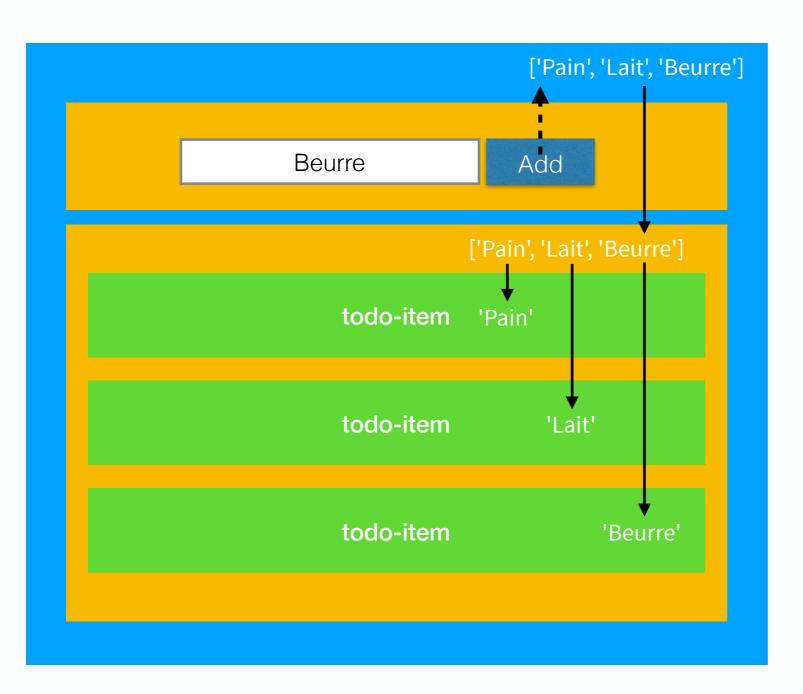
@Output('add') addEvent = new EventEmitter();

select(item) {
    // ...
    this.addEvent.emit(item.value);
    // ...
}
```

```
<ng-select [items]="chains" (add)="addChain($event)"></ng-select>
```

Template - Exercice





- Créer un nouveau projet
 TodoList avec prefix todo et style
 scss
- Créer 3 composants : Form, List,
 Item
- Créer un tableau dans App
- Passer le tableau à List via un property binding
- Passer chaque élément du tableau à Item via un property binding
- Au submit du form du composant Form, transmettre la valeur saisie au parent via un Event Binding (ajouter au tableau dans App)



Modules

Modules - Introduction



- 2 notions de modules
 - NgModule (class décorée avec @NgModule)
 - Module ES6 (import / export de fichiers)
- Jusqu'à la RC d'Angular 2, la notion de NgModule n'existait pas Voir TodoMVC : https://github.com/tastejs/todomvc/tree/gh-pages/examples/angular2
- Intérêt d'avoir des NgModules :
 - Pouvoir importer un ensemble de composants / directives / pipes...
 - Pour configurer la portée d'un service
 - Permettre de charger des blocs de code par lazy-loading (après le chargement initial)

•

Modules - Principaux Modules



Principaux Modules

- AppModule : le module racine
- CommonModule: le module qui inclus toutes les directives Angular de base comme NgIf, NgForOf, mais aussi les pipes comme DatePipe, AsyncPipe...
- BrowserModule: exporte *CommonModule* et contient les services permettant le rendu DOM, la gestion des erreurs, la modification des balises *title* ou *meta...*
- FormsModule: le module qui permet la validation des formulaires, la déclaration de la directive ngModel...
- HttpClientModule: contient les composants pour les requêtes HTTP
- RouterModule: permet de manipuler des routes (associer des composants à des URL)
- Open-Source
 La première chose à faire après l'installation d'une bibliothèque Angular via npm sera d'importer un module

Modules - Déclaration



Déclaration

- Pour qu'un composant, directive ou pipe existe dans l'application il faut le déclarer dans un module
- Ne jamais déclarer 2 fois la même classe dans 2 modules différents

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { HelloComponent } from './hello/hello.component';

@NgModule({
    declarations: [
        AppComponent,
        HelloComponent,
        ],
        imports: [
        BrowserModule
    ],
        bootstrap: [AppComponent]
})
export class AppModule { }
```

Modules - Erreur courante



Erreur courante

Si un composant, directive ou pipe n'est pas déclaré dans un module, ou bien que le module dans lequel il est déclaré n'est pas importé par le module qui l'utilise

```
Uncaught Error: Template parse errors:
'app-title' is not a known element:
1. If 'app-title' is an Angular component, then verify that it is part of this NgModule.
2. If 'app-title' is a Web Component then add 'CUSTOM_ELEMENTS_SCHEMA' to the '@NgModule.schemas' of this component to suppress this message.
```

Modules - Bonnes pratiques



- Bonnes pratiques
 Extrait du Style Guide Angular:
 https://angular.io/guide/styleguide#application-structure-and-ngmodules
 - Créer un dossier core global
 Contiendra la déclaration de tous les services mono-instanciés (singleton) et composants, pipes ou directives utilisés uniquement dans le module racine (AppModule)
 - Créer un module SharedModule global
 Contiendra la déclaration de tous les services multi-instanciés et composants, pipes ou directives utilisés dans différents modules

Modules - Exports



- Utilisé une déclaration dans un autre module
 - Il est fréquent de vouloir utiliser une déclaration (Composant, Pipe, Directive) dans un autre module dans lequel il est déclaré (ex : ng-select déclaré dans NgSelectModule)
 - Dans ce cas il faut qu'il soit déclaré puis exporté dans un NgModule
 - Ce NgModule devra ensuite être importé par chaque NgModule dans lesquels sera utilisé cette déclaration

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { SelectComponent } from './select/select.component';

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

Modules - Imports



- Pour importer une déclaration exportée, on importe le module contenant l'export
- La déclaration devient alors accessible dans n'importe quel composant du module
- Dans notre exemple AppComponent a maintenant accès à SelectComponent dans son template

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { SharedModule } from './shared.module';

@NgModule({
    declarations: [
        AppComponent,
        ],
        imports: [
        BrowserModule,
        SharedModule,
        ],
        bootstrap: [AppComponent]
})
export class AppModule { }
```

Modules - Exports de modules



- Il est courant d'exporter un module pour factoriser les imports
- Exemple: si SharedModule exporte CommonModule et FormsModule, je n'ai plus qu'à importer SharedModule dans ContactsModule pour pouvoir utiliser NgIf et NgModel

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import { SelectComponent } from './select/select.component';
@NgModule({
  declarations: [
    SelectComponent,
   // en important Common, on peut utiliser NgIf dans SelectComponent
  imports: [
    CommonModule,
  // si on importe Shared on a accès à SelectComponent mais aussi NgIf, NgModel...
  exports: [
    CommonModule,
    FormsModule,
    SelectComponent,
export class SharedModule { }
```

Modules - Exports et services



- En important un module, on importe ses déclarations mais également ses services (dont on a décrit l'instanciation avec les providers)
- Le fait d'importer 2 fois un service provoque la réinstanciation du service, c'est à dire qu'il y aura 2 instances dans l'application
- Or il est fréquent de vouloir une seule instance, il faut alors importer le module dans AppModule
- Si le module contient également des déclarations il est courant de retrouver 2 méthodes forRoot, forChild

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

const routes: Routes = [];

@NgModule({
    // forRoot contient le provider de Router, ActivatedRoute...
    imports: [RouterModule.forRoot(routes)],
    exports: [RouterModule]
})
export class AppRoutingModule { }
```

Modules - Exports et services



- Exemple avec RouterModule
 - On peut importer RouterModule.forRoot dans AppModule et avoir ainsi une instance unique de Router dans toute l'application
 - On peut importer RouterModule.forChild pour enregistrer de nouvelles routes sans réimporter le service Router
 - On peut importer RouterModule pour importer les directives comme RouterLink ou RouterOutlet

```
@NgModule({
    declarations: ROUTER_DIRECTIVES,
    exports: ROUTER_DIRECTIVES,
})
export class RouterModule {
    static forRoot(routes: Routes, config?: ExtraOptions) {
        return {
            ngModule: RouterModule,
                providers: [ROUTER_PROVIDERS, provideRoutes(routes)],
            };
    }
    static forChild(routes: Routes) {
        return {ngModule: RouterModule, providers: [provideRoutes(routes)]};
    }
}
```



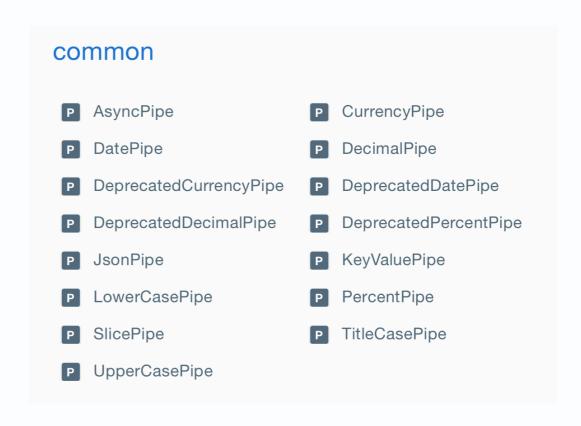
Pipes

Pipes - Introduction



- Pour pouvoir formater proprement des valeurs
- Dans AngularJS on parlait de filtres
- Les Pipes sont définis dans CommonModule (il faut donc que le composant qui les utilise soit défini dans un Module qui importe CommonModule ou BrowserModule)
- Utilisation

```
{{ now | date:'HH:mm:ss' }}
```



Pipes - Création d'un Pipe



- Les Pipes doivent être déclarés dans un module
- Eventuellement exportés pour pouvoir être utilisés ailleurs
- Implémentent PipeTransform
- Le premier paramètre de *transform* est la valeur à transformer
- Il est possible de passer des paramètres supplémentaires (ex : format de Date)

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

@Pipe({
    name: 'duration',
})

export class DurationPipe implements PipeTransform {
    transform(seconds: number, args?: any): any {
        const h = String(Math.floor(seconds / (60 * 60))).padStart(2, '0');
        const m = String(Math.floor((seconds % (60 * 60)) / 60)).padStart(2, '0');
        const s = String(seconds % 60).padStart(2, '0');
        return `${h}:${m}:${s}`;
}
```



Services

Services - Introduction



- Les Services sont des objets utilitaires associés à des concepts informatiques (Client HTTP, Conversion Euro Dollars, Navigation vers un autre composant routé...), par opposition aux Value Objects qui représente des valeurs (String, Date, Model...)
- Angular contient une fonction appelée Injecteur qui permet de retrouver un service facilement sans avoir à le créer directement
- Il faut alors expliquer au framework comment doit se créé le service lorsqu'il est demandé dans un certain contexte
- L'injecteur Angular se base sur le typage statique de TypeScript pour déterminer l'objet à injecter



Un service doit être décoré avec @Injectable

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

@Injectable()
export class UserService {
    users = [{
        id: 1,
            name: 'Romain',
            city: 'Paris',
        }, {
        id: 2,
            name: 'Steven',
            city: 'Besançon',
        }];
}
```



- Pour indiquer à Angular comment créer le service on utilise un provider au niveau d'un NgModule
- useClass permet d'indiquer à Angular qu'il saura résoudre toutes les dépendances de ce service (dans ce cas on peut passer directement le nom de la classe plutôt qu'un provider)

```
import { NgModule } from '@angular/core';
import { SharedModule } from '../shared/shared.module';
import { UsersComponent } from './users/users.component';
import { UserService } from './user.service';
@NgModule({
  declarations: [
    UsersComponent,
  imports: [
    SharedModule,
  providers: [
    { provide: UserService, useClass: UserService },
    // équivalent à :
   UserService
export class UsersModule { }
```



- Pour récupérer le service, il suffit de le demander dans le constructeur d'une classe
- Rappelons nous qu'en TypeScript, utiliser public, protected ou private devant un paramètre du constructeur permet de l'affecter directement à la propriété du même nom

```
import { Component, OnInit } from '@angular/core';
import { User } from '../user.model';
import { Observable } from 'rxjs';
import { UserService } from '../user.service';
@Component({
  selector: 'app-users-list',
  templateUrl: './users-list.component.html',
  styleUrls: ['./users-list.component.scss']
})
export class UsersListComponent implements OnInit {
  users$: Observable<User[]>;
  constructor(protected userService: UserService) { }
 ngOnInit() {
    this.users$ = this.userService.getAll$();
```



Un service peut lui même recevoir des dépendances

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { User } from './user.model';

@Injectable()
export class UserService {

  constructor(protected httpClient: HttpClient) { }

  getAll$(): Observable<User[]> {
    return this.httpClient.get<User[]>('/users');
  }
}
```



Les dépendances peuvent cependant ne pas être résolue automatiquement (ici url)

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { User } from './user.model';

@Injectable()
export class UserService {

  constructor(protected httpClient: HttpClient, protected url: string) { }

  getAll$(): Observable<User[]> {
    return this.httpClient.get<User[]>(this.url);
  }
}
```

Services - Singleton



- Dans ce cas on pourra utiliser une fabrique avec useFactory
- La fabrique n'étant pas décorée avec @Injectable, il faudra lui donner ses Token d'Injection avec deps

```
export function userServiceFactory(httpClient: HttpClient) {
   return new UserService(httpClient, '/users');
}

@NgModule({
   declarations: [
     UsersComponent,
   ],
   imports: [
     SharedModule,
   ],
   providers: [
     { provide: UserService, useFactory: userServiceFactory, deps:
[HttpClient] },
   ],
})
export class UsersModule { }
```

Services - Singleton



- Pour garantir qu'un service n'ait qu'une seule instance il faut le déclarer dans AppModule ou dans un module qui ne sera inclus que par AppModule
- Depuis Angular 6 il est possible de déclarer un service directement dans le décorateur @Injectable, 'root' signifiant AppModule

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { User } from './user.model';

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

   constructor(protected httpClient: HttpClient) { }

   getAll$(): Observable<User[]> {
     return this.httpClient.get<User[]>('/users');
}
```

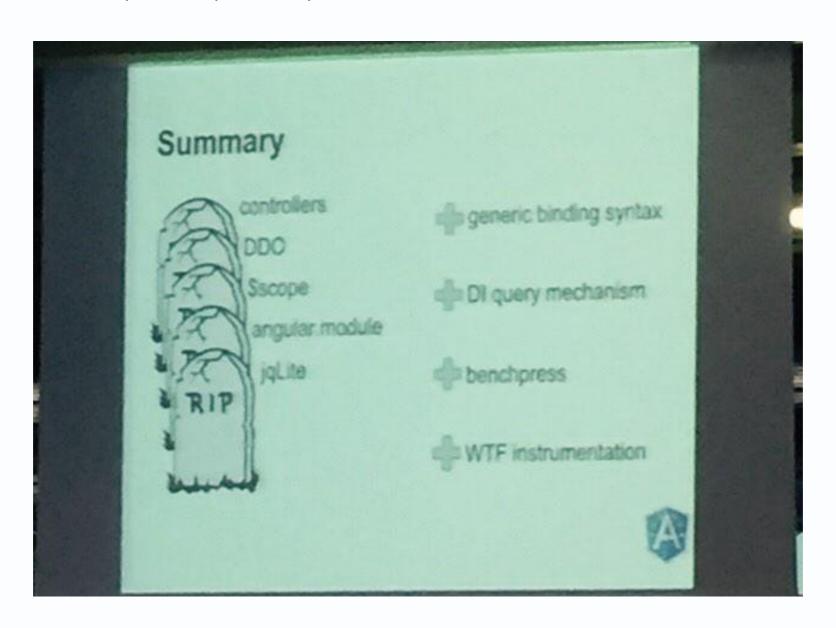


Migrer depuis AngularJS

Migrer depuis AngularJS - Introduction



- AngularJS proposait plusieurs architectures, MVC, MVVM, on parlait dans la documentation de MVW pour Model View Whatever (Model Vue Peut-importe)
- Angular a été annoncé à Paris en octobre 2014
 https://twitter.com/bioub/status/525198150873931776



Migrer depuis AngularJS - Composants



- Les composants ont été introduit dans AngularJS 1.5 (février 2016)
- Angular 2 sort en octobre 2016 avec son architecture autour des composant
- Pour pouvoir migrer il faut donc réarchitecturer son applications autour de composants
- Afin de faciliter l'utilisation de composants AngularJS dans Angular et inversement, les développeurs ont créé ngUpgrade
- ngUpgrade permet le temps de migrer entièrement son application AngularJS d'avoir une application hybride
- On peut au choix décider de prendre un composant AngularJS et de l'upgrader vers Angular ou bien de downgrader des composants Angular
- Un wiki est dédié à la migration https://github.com/angular/ngMigration-Forum/wiki



Mettre à jour Angular

Mettre à jour Angular - Introduction



- Angular depuis la v2 respecte le versionnage sémantique https://semver.org/
- Avant de migrer vers une version majeure
 - Lire le changelog <u>https://github.com/angular/angular/blob/master/CHANGELOG.md</u>
 - Suivre les recommandation de l'Update Guide <u>https://angular-update-guide.firebaseapp.com/</u>

Mettre à jour Angular - Outdated



Vérifier qu'on est à jour ou non : npm outdated

MacBook-Pro:AddressBookAngular romain\$ npm outdated				
<u>Package</u>	<u>Current</u>	<u>Wanted</u>	<u>Latest</u>	<u>Location</u>
@angular/animations	4.4.6	4.4.6	5.2.9	address-book-angular
@angular/cli	1.4.9	1.4.9	1.7.3	address-book-angular
@angular/common	4.4.6	4.4.6	5.2.9	address-book-angular
@angular/compiler	4.4.6	4.4.6	5.2.9	address-book-angular
@angular/compiler-cli	4.4.6	4.4.6	5.2.9	address-book-angular
@angular/core	4.4.6	4.4.6	5.2.9	address-book-angular
@angular/forms	4.4.6	4.4.6	5.2.9	address-book-angular
@angular/http	4.4.6	4.4.6	5.2.9	address-book-angular
@angular/language-service	4.4.6	4.4.6	5.2.9	address-book-angular
@angular/platform-browser	4.4.6	4.4.6	5.2.9	address-book-angular
@angular/platform-browser-dynamic	4.4.6	4.4.6	5.2.9	address-book-angular
@angular/router	4.4.6	4.4.6	5.2.9	address-book-angular
@ngx-translate/core	7.2.2	7.2.2	9.1.1	address-book-angular
@types/jasmine	2.5.54	2.5.54	2.8.6	address-book-angular
@types/node	6.0.90	6.0.102	9.4.7	address-book-angular
codelyzer	3.2.2	3.2.2	4.2.1	address-book-angular
core-js	2.5.1	2.5.3	2.5.3	address-book-angular
jasmine-core	2.6.4	2.6.4	3.1.0	address-book-angular
jasmine-spec-reporter	4.1.1	4.1.1	4.2.1	address-book-angular
karma	1.7.1	1.7.1	2.0.0	address-book-angular
karma-chrome-launcher	2.1.1	2.1.1	2.2.0	address-book-angular
karma-coverage-istanbul-reporter	1.3.0	1.4.2	1.4.2	address-book-angular
karma-jasmine	1.1.0	1.1.1	1.1.1	address-book-angular
karma-jasmine-html-reporter	0.2.2	0.2.2	1.0.0	address-book-angular
protractor	5.1.2	5.1.2	5.3.0	address-book-angular
rxjs	5.5.1	5.5.7	5.5.7	address-book-angular
ts-node	3.2.2	3.2.2	5.0.1	address-book-angular
tslint	5.7.0	5.7.0	5.9.1	address-book-angular
typescript	2.3.4	2.3.4	2.7.2	address-book-angular
zone.js	0.8.18	0.8.20	0.8.20	address-book-angular

Mettre à jour Angular - Migrer



- Angular CLI a une commande update depuis la v1.7
- Pour migrer Angular CLI:npm i @angular/cli@latest
- Mettre à jour Angular ng update
- Vers la prochaine version pour tester son code ng update --next

Mettre à jour Angular - Bonnes pratiques



- Lancer la migration dans une branche, lancer les tests automatisés et/ou refaire des tests manuels
- Avec git, sur les commandes : checkout, pull, merge, rebase Lancer npm install
- Peut s'automatiser avec des hooks git
- Utiliser husky pour versionner ces scripts dans package.json https://github.com/typicode/husky



Immuabilité

Immuabilité - Introduction



- Lors de la modification d'un objet, le changement peut-être muable en modifiant l'objet d'origine ou immuable en créant un nouvel objet
- Les algorithmes de détections de changements préfèreront les changements immuables, ayant ainsi juste à comparer les références plutôt que l'ensemble du contenu de l'objet
- Exemple, en JS les tableaux sont muables, les chaines de caractères immuables

```
const firstName = 'Romain';
firstName.toUpperCase();
console.log(firstName); // Romain

const firstNames = ['Romain'];
firstNames.push('Edouard');
console.log(firstNames.join(', ')); // Romain, Edouard
```



Ajouter à la fin

```
const firstNames = ['Romain', 'Edouard'];
function append(array, value) {
  return [...array, value];
}

const newfirstNames = append(firstNames, 'Jean');
console.log(newfirstNames.join(', ')); // Romain, Edouard, Jean
console.log(firstNames === newfirstNames); // false
```

Ajouter au début

```
const firstNames = ['Romain', 'Edouard'];
function prepend(array, value) {
  return [value, ...array];
}

const newfirstNames = prepend(firstNames, 'Jean');
console.log(newfirstNames.join(', ')); // Jean, Romain, Edouard
console.log(firstNames === newfirstNames); // false
```



Ajouter à un indice donné



Modifier un élément



Supprimer un élément

Immuabilité - Objet



Ajouter un élément

```
const contact = {
  firstName: 'Romain',
  lastName: 'Bohdanowicz',
};

function add(object, key, value) {
  return {
    ...object,
    [key]: value
  };
}

const newContact = add(contact, 'city', 'Paris');
console.log(JSON.stringify(newContact));
// {"firstName":"Romain","lastName":"Bohdanowicz","city":"Paris"}
console.log(contact === newContact); // false
```

Immuabilité - Objet



Modifier un élément

```
const contact = {
    firstName: 'Romain',
    lastName: 'Bohdanowicz',
};

function modify(object, key, value) {
    return {
        ...object,
        [key]: value
    };
}

const newContact = modify(contact, 'firstName', 'Thomas');
console.log(JSON.stringify(newContact));
// {"firstName":"Thomas","lastName":"Bohdanowicz"}
console.log(contact === newContact); // false
```

Immuabilité - Objet



Supprimer un élément

```
const contact = {
  firstName: 'Romain',
  lastName: 'Bohdanowicz',
};

function remove(object, key) {
  const { [key]: val, ...rest } = object;
  return rest;
}

const newContact = remove(contact, 'lastName');
console.log(JSON.stringify(newContact));
// {"firstName":"Romain"}
console.log(contact === newContact); // false
```



- Pour simplifier la manipulation d'objets ou de tableaux immuables, Facebook a créé Immutable.js
- Installation
 npm install immutable



Ajouter à la fin

```
const immutable = require('immutable');

const firstNames = immutable.List(['Romain', 'Edouard']);

const newfirstNames = firstNames.push('Jean');

console.log(newfirstNames.join(', ')); // Romain, Edouard, Jean
console.log(firstNames === newfirstNames); // false
```

Ajouter au début

```
const immutable = require('immutable');

const firstNames = immutable.List(['Romain', 'Edouard']);

const newfirstNames = firstNames.unshift('Jean');

console.log(newfirstNames.join(', ')); // Jean, Romain, Edouard
console.log(firstNames === newfirstNames); // false
```



Ajouter à un indice donné

```
const immutable = require('immutable');

const firstNames = immutable.List(['Romain', 'Edouard']);

const newfirstNames = firstNames.insert(1, 'Jean');

console.log(newfirstNames.join(', ')); // Romain, Jean, Edouard
console.log(firstNames === newfirstNames); // false
```



Modifier un élément

```
const immutable = require('immutable');

const firstNames = immutable.List(['Romain', 'Edouard']);

const newfirstNames = firstNames.set(1, 'Jean');

console.log(newfirstNames.join(', ')); // Romain, Jean
console.log(firstNames === newfirstNames); // false
```



Supprimer un élément

```
const immutable = require('immutable');

const firstNames = immutable.List(['Romain', 'Edouard']);

const newfirstNames = firstNames.delete(1);
console.log(newfirstNames.join(', ')); // Romain
console.log(firstNames === newfirstNames); // false
```

Immuabilité - Immutable.js Map



Ajouter un élément

```
const immutable = require('immutable');

const contact = immutable.Map({
    firstName: 'Romain',
    lastName: 'Bohdanowicz',
});

const newContact = contact.set('city', 'Paris');
console.log(JSON.stringify(newContact));
// {"firstName":"Romain","lastName":"Bohdanowicz","city":"Paris"}
console.log(contact === newContact); // false
```

Immuabilité - Immutable.js Map



Modifier un élément

```
const immutable = require('immutable');

const contact = immutable.Map({
    firstName: 'Romain',
    lastName: 'Bohdanowicz',
});

const newContact = contact.set('firstName', 'Thomas');
console.log(JSON.stringify(newContact));
// {"firstName":"Thomas","lastName":"Bohdanowicz"}
console.log(contact === newContact); // false
```

Immuabilité - Immutable.js Map



Supprimer un élément

```
const immutable = require('immutable');

const contact = immutable.Map({
   firstName: 'Romain',
   lastName: 'Bohdanowicz',
});

const newContact = contact.remove('lastName');
console.log(JSON.stringify(newContact));
// {"firstName":"Romain"}
console.log(contact === newContact); // false
```



Détection de changement

Détection de changement - Introduction

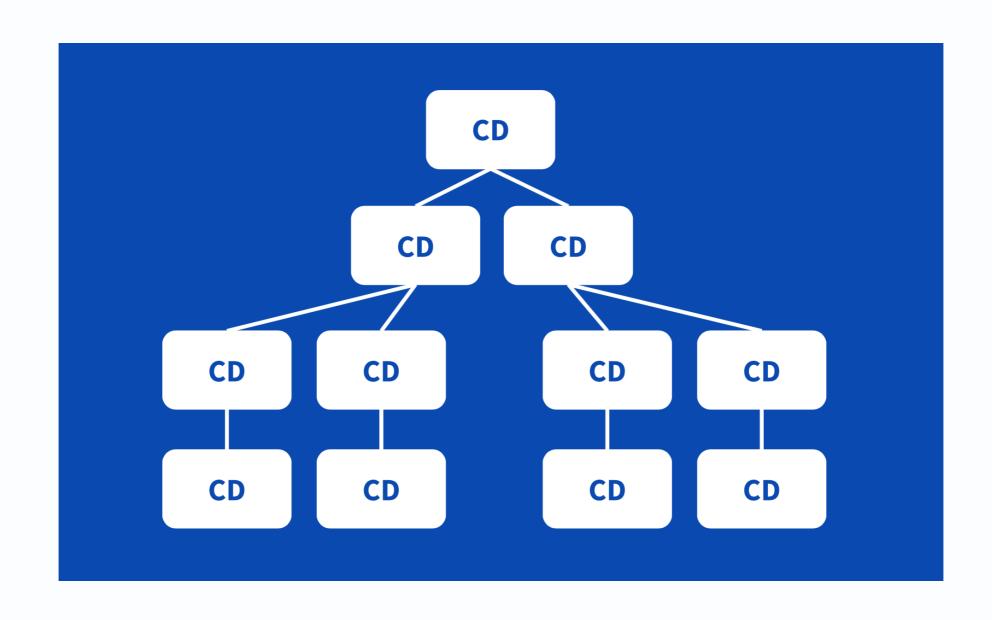


- Dans Angular la détection de changement permet de rafraîchir les property bindings des composants
- On peut lancer les algorithmes de détection de changement de 3 façons :
 - Les events bindings (click), (submit)...
 - Les opérations asynchrones grâce à Zone.js
 - Manuellement via ChangeDetectionRef

Détection de changement - Déclenchement



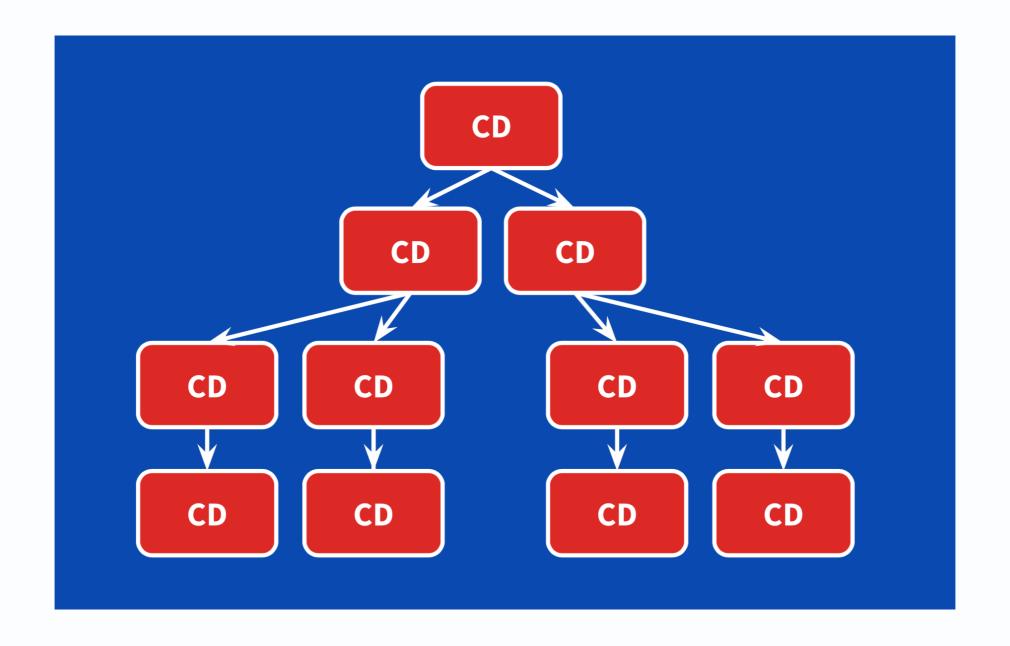
- Chaque composant vient avec son propre algorithme de détection qui est généré au moment de la compilation du template (en JIT ou AOT)
- Le code généré est optimisé pour la VM JavaScript et lance des comparaison du type ancienneValeur === nouvelleValeur ou ancienneValeur.prop === nouvelleValeur.prop



Détection de changement - Lancement



 Lorsque que la détection de changement est lancée, l'ensemble des algorithmes de des composants sont lancés du composant racine vers les composants les plus lointains



Détection de changement - Interruption



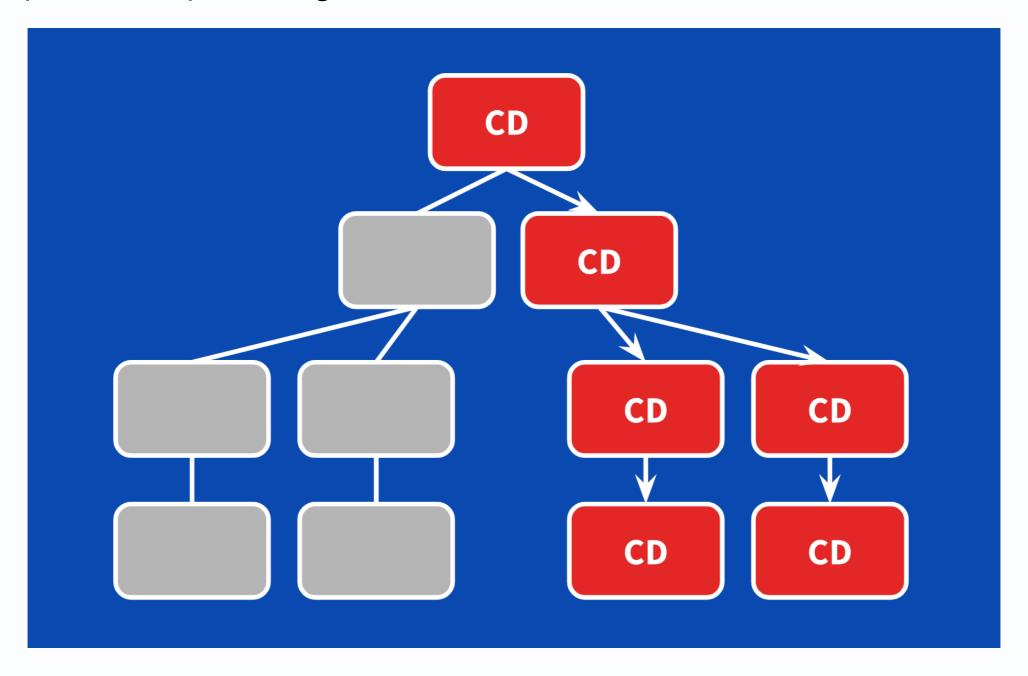
- Pour éviter que la détection de changement soit lancée sur un composant et ses descendants, on peut :
 - Utiliser la stratégie OnPush
 - Utiliser ChangeDetectionRef (Manuel)

Détection de changement - OnPush



OnPush

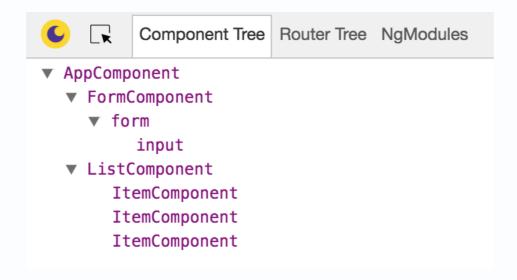
 Avec la stratégie OnPush on peut faire en sorte qu'un composant ne dépende que de ses Input (changement immuable)



Détection de changement - OnPush



Exemple
 https://gitlab.com/angular-avance/TodoAngular





ChangeDetectorRef
 Permet de contrôler soit même la détection de changement

```
export declare abstract class ChangeDetectorRef {
   abstract markForCheck(): void;
   abstract detach(): void;
   abstract detectChanges(): void;
   abstract checkNoChanges(): void;
   abstract reattach(): void;
}
```

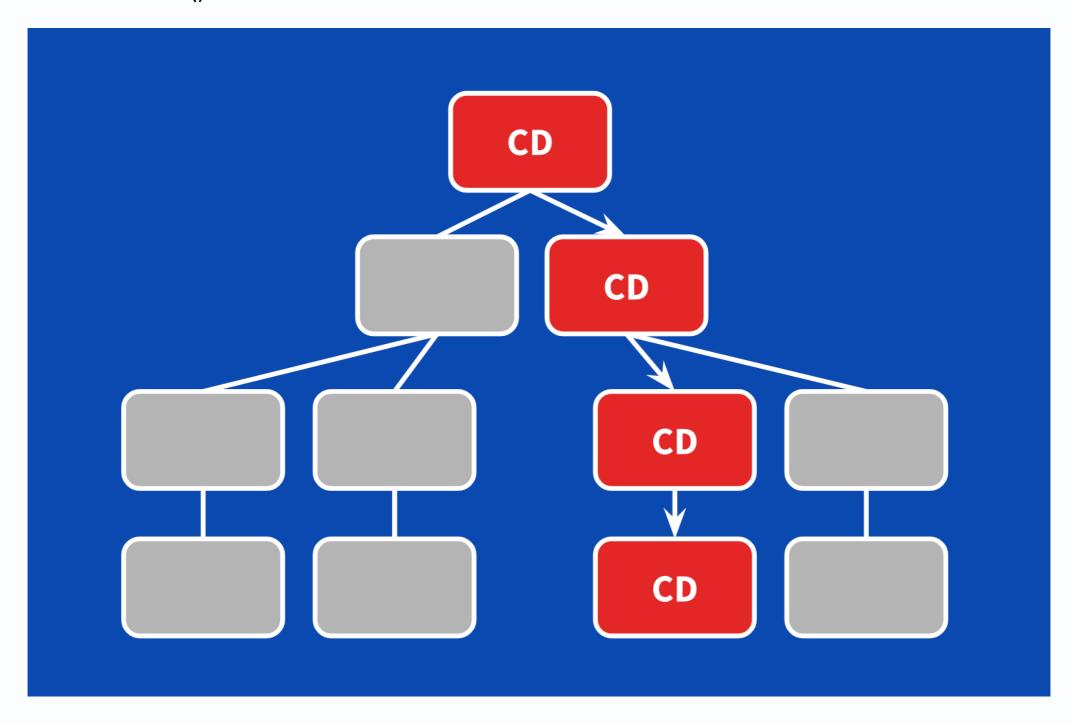


markForCheck()
 Permet de tagger le composant et tous ses ancêtres OnPush comme étant à checker

```
import { ChangeDetectionStrategy, ChangeDetectorRef } from '@angular/core';
@Component({
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class ContactsShowComponent implements OnInit {
  public contact: Contact;
  constructor(private cd: ChangeDetectorRef) {}
  ngOnInit() {
    this.contactService.getAll()
      .subscribe(contact => {
        this.contact = contact;
        this.cd.markForCheck();
      });
```



markForCheck()





- detach()
 Désactive la détection du changement de ce composant
- reattach()
 Réactive la détection du changement de ce composant
- detectChanges()
 Lance la détection du changement manuellement

```
import { ChangeDetectionStrategy, ChangeDetectorRef } from '@angular/core';
@Component()
export class ContactsShowComponent implements OnInit {
 public contact: Contact;
  constructor(private cd: ChangeDetectorRef) {}
  ngOnInit() {
    this.cd.detach();
    this.contactService.getAll()
      .subscribe(contact => {
        this.contact = contact;
        this.cd.detectChanges();
      });
```

Détection de changement - Resources



- Angular Change Detection Explained <u>https://blog.thoughtram.io/angular/2016/02/22/angular-2-change-detection-explained.html</u>
- Everything you need to know about change detection in Angular https://blog.angularindepth.com/everything-you-need-to-know-about-change-detection-in-angular-8006c51d206f
- How to test OnPush components
 https://medium.com/@juliapassynkova/how-to-test-onpush-components-c9b39871fe1e
- JS web frameworks benchmark
 http://www.stefankrause.net/js-frameworks-benchmark7/table.html



Créer sa bibliothèque

Créer sa bibliothèque - Introduction



- Deux options pour créer sa propre bibliothèque
 - Utiliser @angular/compiler et @angular/compiler-cli
 - Utiliser ng-packagr
- Avantages de ng-packagr
 - Respecte le format <u>Angular Package</u>
 - Créé des versions ESM ES6, ESM ES6, and UMD
 - Code compatible Angular CLI, Webpack, or SystemJS
 - Definitions et metadata(.d.ts, .metadata.json)
 - Points d'entrées secondaires : @my/foo, @my/foo/testing, @my/foo/bar
 - Converti les templates et les styles en inline
 - Fonctionnalités CSS incluses : SCSS, LESS, Stylus, Autoprefixer, PostCSS

Créer sa bibliothèque - Création



Créer son propre package

```
package-lock.json
package.json
src
index.ts
ui-copyright.component.ts
ui-copyright.module.ts
tsconfig.json
```

Le fichier index.ts exporte tout ce qui doit pouvoir être importé (Modules, Services, Interfaces, Classes, Injectors Tokens...)

```
export * from './ui-copyright.module';
```

Créer sa bibliothèque - Création



package.json minimal

```
"name": "@formation.tech/ui-copyright",
"version": "0.0.1",
"main": "dist/index.js",
"typings": "dist/index.d.ts",
"dependencies": {
  "@angular/common": "^5.2.9",
  "@angular/core": "^5.2.9",
  "rxjs": "^5.5.7"
},
"devDependencies": {
  "@angular/compiler": "^5.2.9",
  "@angular/compiler-cli": "^5.2.9",
  "typescript": "^2.7.2"
},
"scripts": {
  "build": "ngc"
```

Créer sa bibliothèque - Création



tsconfig.json minimal

```
"compilerOptions": {
    "module": "es2015", // modules exportés en ES6
    "moduleResolution": "node", // rend accessible node_modules si module es2015
    "target": "es5", // code exporté en ES5
    "sourceMap": false, // génère les fichiers map pour le debug ou non
    "declaration": true, // génère les fichiers .d.ts pour la complétion
    "experimentalDecorators": true, // support des décorateurs @Component ...
    "outDir": "dist", // dossier de destination
    "rootDir": "src", // dossier à builder
    "lib": [
        "dom", // reconnait les types du DOM : Console, Node, Document...
        "es2015" // reconnait les types ES6 : Map...
]
}
```

Lancer ensuite le compilateur npm run build

Créer sa bibliothèque - ngPackagr



- Installation npm i ngPackager -D
- Ajouter la config au package.json

```
{
  "name": "@formation.tech/ui-horloge",
  "version": "0.0.1",
  "license": "MIT",
  "scripts": {
     "build": "ng-packagr -p package.json"
  },
  "ngPackage": {
     "lib": {
        "entryFile": "public_api.ts"
     },
     "dest": "../lib/ui-horloge"
  }
}
```

Lancer le build npm run build

Créer sa bibliothèque - ngPackagr



Package généré

```
bundles
  formation.tech-ui-horloge.umd.js
   - formation.tech-ui-horloge.umd.js.map
   - formation.tech-ui-horloge.umd.min.js
   - formation.tech-ui-horloge.umd.min.js.map
esm2015
  formation.tech-ui-horloge.js
 — formation.tech-ui-horloge.js.map
esm5
  formation.tech-ui-horloge.js
    formation.tech-ui-horloge.js.map
formation.tech-ui-horloge.d.ts
formation.tech-ui-horloge.metadata.json
node_modules
package.json
public_api.d.ts
src
      - ui-horloge
           - ui-horloge.component.d.ts
           - ui-horloge.module.d.ts
    typings.d.ts
```

Créer sa bibliothèque - Installation



- Pour installer un paquet on peut :
 - créer un lien symbolique : npm install ../ma-lib
 - déployer sur git : npm install
 - · déployer sur npm public ou privé : npm install ma-lib
- Attention depuis Angular 5 il faut ajouter l'option preserveSymlinks au moment du build (en CLI --preserve-symlinks ou dans le .angular-cli.json) en cas d'installation locale

```
{
  "defaults": {
    "build": {
        "preserveSymlinks": true
     }
  }
}
```

Créer sa bibliothèque - Bonnes pratiques



- Supprimer le node_modules de la bibliothèque après son build
- Ne pas générer les ngfactory.js "angularCompilerOptions": { "skipTemplateCodegen": true }
- Compiler la version la plus basse possible (un module compilé en Angular 5 ne fonctionnera pas dans Angular 4)

Créer sa bibliothèque - Resources



- How to build and publish an Angular module <u>https://medium.com/@cyrilletuzi/how-to-build-and-publish-an-angular-module-7ad19c0b4464</u>
- angular-cli-lib-example
 https://github.com/jasonaden/angular-cli-lib-example
- Building an Angular 4 Component Library with the Angular CLI and ng-packagr <u>https://medium.com/@nikolasleblanc/building-an-angular-4-component-library-with-the-angular-cli-and-ng-packagr-53b2ade0701e</u>
- Distributing an Angular Library The Brief Guide <u>http://blog.mgechev.com/2017/01/21/distributing-an-angular-library-aot-ngc-types/</u>



Angular Universal

Angular Universal - Introduction



- Permet de faire un rendu côté server (SSR)
- 3 intérêts:
 - Faciliter le référencement (SEO)
 - Améliorer les performances sur mobile et machines peu performantes
 - Afficher la première page plus rapidement
- Angular CLI 1.6+ facilite fortement la mise en place
- 3 moteurs
 - Express (Node.js)
 - Hapi (Node.js)
 - ASP.NET

Angular Universal - Pièges



- Choses à anticiper
 - Les Web APIs ne seront pas dispo côté serveur (Window, Document, ...) bien qu'ils soient en partie réimplémentés
 - Les requêtes AJAX vont s'exécuter 2 fois, côté serveur puis côté client, créer un cache pour l'éviter
 - Les requêtes AJAX doivent être absolues
 - Eviter ou bannir setTimeout et surtout setInterval
 - Certains modules ne fonctionneront pas, ou nécessiteront des changements, ex ngx-translate :

```
import { TranslateLoader, TranslateModule } from '@ngx-translate/core';
import { TranslateHttpLoader } from '@ngx-translate/http-loader';
import { UniversalTranslateLoader } from '@ngx-universal/translate-loader';

// AoT requires an exported function for factories
export function translateFactory(platformId: any, httpClient: HttpClient):
TranslateLoader {
   const browserLoader = new TranslateHttpLoader(httpClient);
   return new UniversalTranslateLoader(platformId, browserLoader, 'dist-server/assets/i18n');
}
```

Angular Universal - Test de plate-forme



- Exécuter du code sur une plateforme spécifiquement (client ou serveur)
 - Utiliser l'Injection Token PLATFORM_ID et les méthodes isPlatformBrowser et isPlatformServer

```
import { PLATFORM_ID } from '@angular/core';
import { isPlatformBrowser, isPlatformServer } from '@angular/common';

constructor(@Inject(PLATFORM_ID) private platformId: Object) { ... }

ngOnInit() {
   if (isPlatformBrowser(this.platformId)) {
      // Client only code.
      ...
   }
   if (isPlatformServer(this.platformId)) {
      // Server only code.
      ...
   }
}
```

Angular Universal - Création



- Rendre son application universelle ng generate universal [nom] ng generate universal serverApp
- La commande *generate universal* fait des changements dans l'application et créé une seconde application dans le .angular-cli.json
- Pour builder les 2 apps
 ng build --prod && ng build --prod --app server-app --output-hashing=none

Angular Universal - Serveur Express



 Installer express et le moteur universal pour express npm i express @nguniversal/express-engine

```
require('zone.js/dist/zone-node');
const path = require('path');
const express = require('express');
const ngUniversal = require('@nguniversal/express-engine');
const appServer = require('./dist-server/main.bundle');
const app = express();
app.use(express.static(path.resolve(__dirname, 'dist')));
app.engine('html', ngUniversal.ngExpressEngine({
  bootstrap: appServer.AppServerModuleNgFactory
}));
app.set('view engine', 'html');
app.set('views', 'dist');
app.use((req, res) => {
  res.render('index', { req, res });
});
app.listen(8000, () => {
  console.log(`Listening on http://localhost:8000`);
});
```

Angular Universal - TransferState



- Avec un rendu côté serveur, les requêtes vont s'exécuter 2 fois, côté serveur puis client
- Outre le problème de performance, les données vont "flasher"



Angular Universal - TransferState



- Configuration
 - Importer ServerTransferStateModule dans AppServerModule
 - Importer BrowserTransferStateModule dans AppModule
 - Dans un Composant :

```
import { makeStateKey, TransferState } from '@angular/platform-browser';
const RESULT_KEY = makeStateKey('contacts');

export class ContactService implements ContactServiceInterface {
   public contacts;

public getList$(): Observable<Contact[]> {
    if (this.transferState.hasKey(RESULT_KEY)) {
      const res = Observable.of(this.transferState.get<Contact[]>(RESULT_KEY, null));
      this.transferState.remove(RESULT_KEY);
      return res;
    } else {
      this.transferState.onSerialize(RESULT_KEY, () => this.contacts);
      return this.http.get<Contact[]>(`${environment.apiServer}/contacts`)
      .pipe(tap(contacts => this.contacts = contacts));
    }
}
```

Angular Universal - TransferState



Voir aussi

- TransferHttpCacheModule pour se simplifier les transferts HTTP: https://github.com/angular/universal/tree/master/modules/common
- Pour ngx-translate: https://github.com/ngx-translate/core/issues/
 754#issuecomment-353616515

Angular Universal - Resources



- ng-seed/universal https://github.com/ng-seed/universal
- Using TransferState API in an Angular v5 Universal App https://blog.angularindepth.com/using-transferstate-api-in-an-angular-5-universal-app-130f3ada9e5b
- Angular server-side rendering in Node with Express Universal Engine
 https://medium.com/@cyrilletuzi/angular-server-side-rendering-in-node-with-express-universal-engine-dce21933ddce



Progressive Web Apps

PWA - Introduction

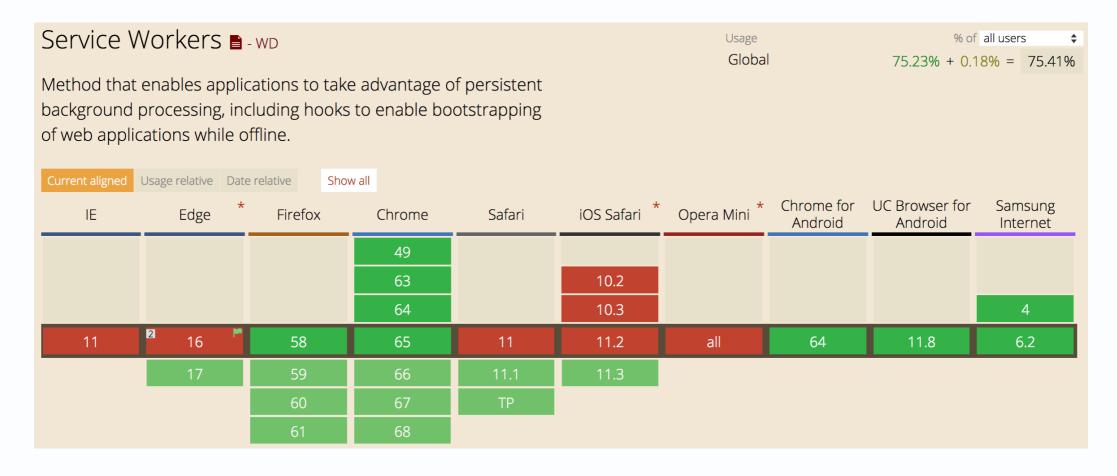


- Intérêt des Progressive Web Apps
 - Un temps de chargement considérablement réduit
 - Une utilisation sans connexion Internet
 - Elles sont Responsive, donc compatibles avec n'importe quel système d'exploitation et n'importe quel support (pc, tablette, mobile)
 - Pas d'installation requise
 - Les PWA sont accessibles depuis une URL ou directement depuis une icône sur l'écran d'accueil du mobile
 - Elles ne prennent pas de place dans la mémoire du mobile
 - Elles sont sécurisées (protocole HTTPS)
 - · Une expérience immersive grand écran, semblable aux applications natives.

PWA - Introduction



- Service Worker vs AppCache
 - Depuis quelques années les navigateurs implémentaient un API appelé
 AppCache permettant de décrire dans un fichier manifest les resources à garder
 en cache pour une utilisation hors ligne
 - Un Service Worker permet l'exécution du code dans un thread séparé, améliorant les performances lors d'appel à des données et permettant de les retrouver en mode hors ligne



PWA - Mise en place



- Installationnpm i @angular/service-worker
- Angular CLI
 Ajouter dans l'app cliente
 "serviceWorker": true
- Créer un fichier src/ngsw-config.json (peut se générer à partir du CLI ngsw-config)
- Importer dans AppModule
 ServiceWorkerModule.register(
 '/ngsw-worker.js', {
 enabled: environment.production
 },
),

```
"index": "/index.html",
"assetGroups": [{
  "name": "app",
  "installMode": "prefetch",
  "resources": {
    "files": [
      "/favicon.ico",
      "/index.html"
    "versionedFiles": [
      "/*.bundle.css",
      "/*.bundle.js",
      "/*.chunk.js"
  "name": "assets",
  "installMode": "lazy",
  "updateMode": "prefetch",
  "resources": {
    "files": [
      "/assets/**"
```

PWA - Manifest



Créer un fichier src/manifest.json (l'ajouter dans les assets dans .angular-cli.json)

```
{
  "short_name": "Address Book",
  "name": "Address Book Angular Avancé",
  "start_url": "/",
  "theme_color": "#212529",
  "background_color": "#f8f9fa",
  "display": "standalone",
  "orientation": "portrait",
  "icons": [
      {
            "src": "/assets/icons/android-chrome-512x512.png",
            "sizes": "512x512",
            "type": "image/png"
      }
    ]
}
```

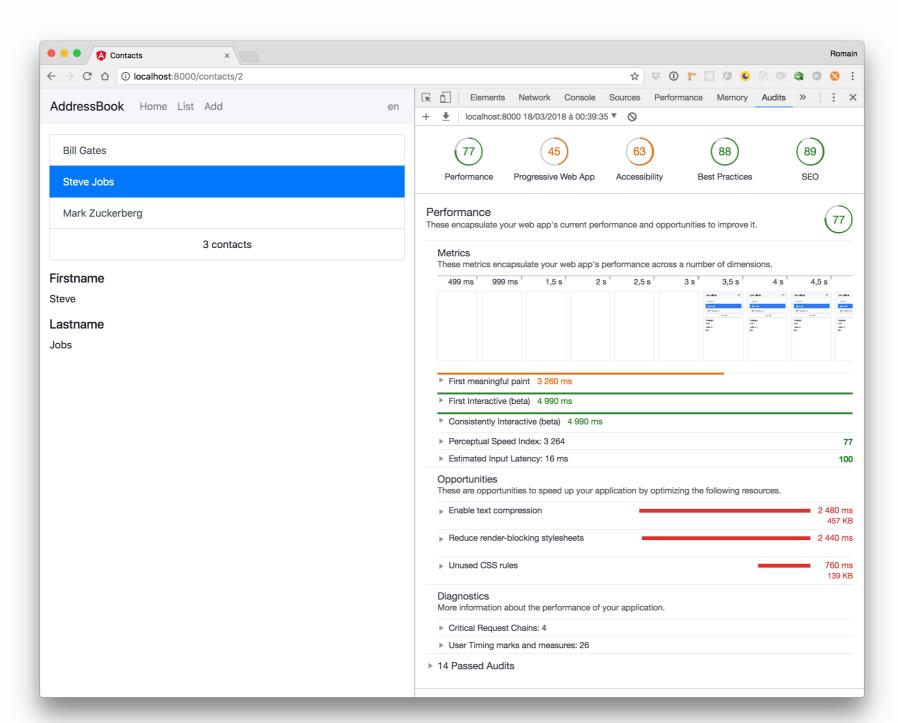
Ajouter au fichier index.html

```
<link rel="manifest" href="/manifest.json">
<meta name="theme-color" content="#212529"/>
```

PWA - Lighthouse

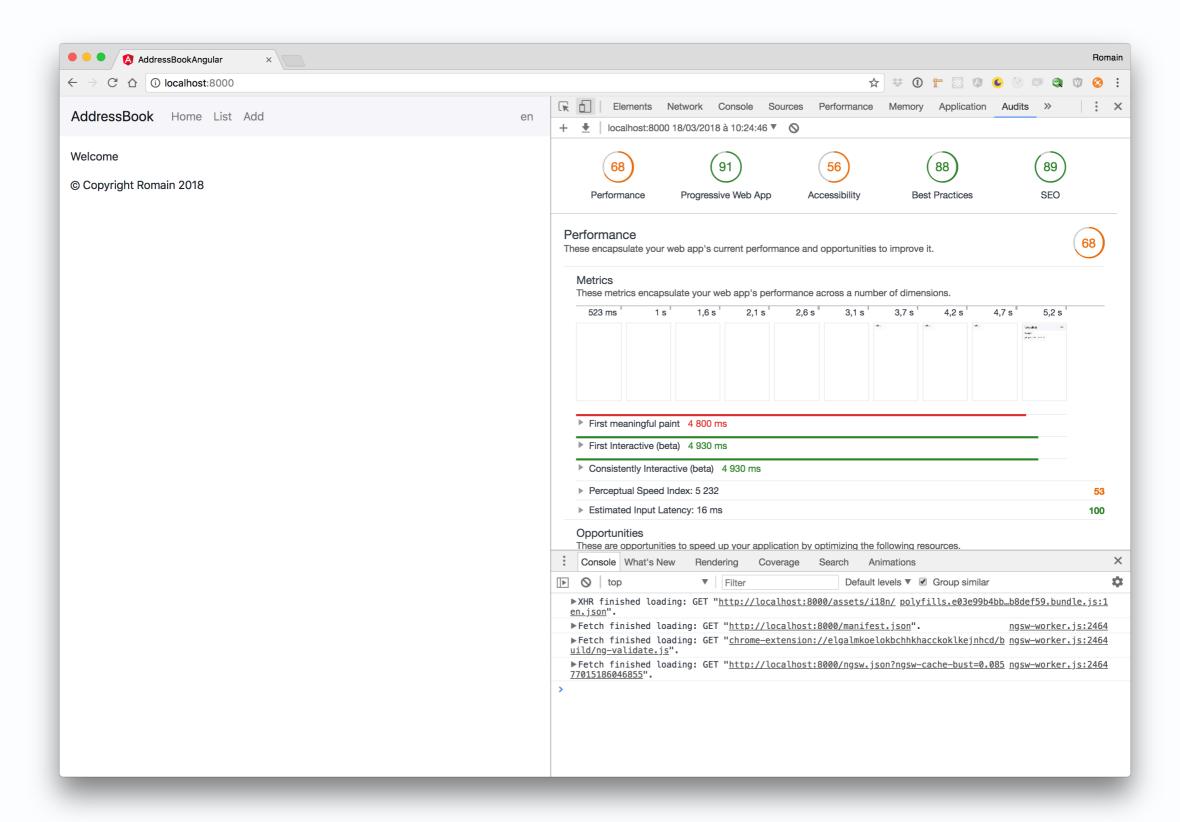


 Lighthouse
 Chrome inclus un outil permettant de savoir si une application fait le nécessaire pour se dire PWA



PWA - Lighthouse





PWA - Resources



- Service worker configuration
 https://angular.io/guide/service-worker-config
- Service Workers in Angular With @angular/service-worker
 https://alligator.io/angular/service-workers/