Principes et Commandes NodeJS - TypeScript - Angular

NodeJS

Principe :

NodeJS permet d'utiliser du JavaScript en dehors d'un navigateur web.

Comme Chrome, il se base sur V8 pour interpréter et optimiser le code JS.

NPM = Node Package Manager. C'est le gestionnaire de projet JavaScript (équiv de ***Maven*** et ***Gradle*** pour Java)

Instalation de NodeJS :

* Via Nodist :

GitHub de Nodist : https://github.com/nullivex/nodist

Nodist permet d'instaler plusieurs versions de Node.JS sur une même machine et de sélectionner la version à utiliser.

Pour instaler Nodist et NodeJS : https://github.com/nullivex/nodist/releases

> télécharger et exécuter .exe

> "node -v" et "npm -v" en ligne de commande pour vérifier que cela a bien fonctionné et que la variable Path a été MàJ

* Ou instalation simple :

Télécharger NodeJS sur <https://nodejs.org/en/download/>

L'installer à la racine du lecteur C:

Vérifier que la variable d'environnement pointant sur le dossier 'npm' a bien été créée. Sinon la rajouter au Path.

Instalation de NPM et Angular-CLI

Dans l'invite de commande : (Rappel pour l'ouvrir : Window+R => cmd)

Installer la dernière version de NPM (*Node Package Manager*) en global avec la commande : npm install -g npm@latest

Installer le CLI (*Command Line Interface*) Angular en global avec la commande : npm install -g @angular/cli

Contenu du projet :

package.json

fichier de config (équiv. à *POM.xml* de Maven ou *build.gradle* de gradle) qui définit :

- nom (obligatoire)

- version (obligatoire)

- dépendances

- raccourcit en ligne de commandes ("script": {})

dossier ***node\_modules***

contient toutes les dépendances liées au projet et chargé par npm

ATTENTION : à ne pas charger sur **GitHub**

Détails importants sur le fonctionnement global et principe d'importation/exportation d'éléments d'un fichier .js à un autre :

Lors du chargement d'un fichier .js, Node place le contenu dans une fonction afin de préserver le global space name et de mettre à disposition 5 éléments accessibles partout dans l'appli de la manière suivante :

(function (exports, require, module, \_\_filename, \_\_dirname){

*contenu du fichier.js original*

});

exports => ref à module.exports

require => permet de charger les modules (depuis un autre fichiers .js ou depuis une dépendances incluses dans ***node\_modules***)

module => ref au module courant pour mettre à disposition l'élément module.export qui permet d'exporter des objets du fichier courant

\_\_filename => ref au fichier .js lui-même

\_\_dirname => ref au dossier courant lui-même

Ainsi, il est possible d'accéder aux éléments comme suit :

var monObjet = require('monmodule') importe le module contenu dans le dossier de dépendances node\_modules

var monObjet = require('./monFichier.js') importe les éléments déclarés module.exports = ... dans le fichier dans le même dossier

var monObjet = require('../monFichier.js') importe les éléments déclarés module.exports = ... dans le fichier dans le dossier parent

fichierA.js

module.exports = monObjet; => rend l'objet monObjet accessible (via un require('fichierA')) depuis un autre fichier JavaScript

fichierB.js

var monObjet = require('./fichierA'); => récupération de monObjet dans un autre fichier (ici dans le même dossier)

Dans la pratique, on aura le plus couramment :

|  |
| --- |
| import { Component, OnInit } from '@angular/core';  import { MonService } from 'src/services/mon-service.ts';  export class MonComponent {  constructor(public monService: MonService) {}  maMethode() {  return monService.faitQuelqueChose();  }  } |

Tests unitaires

Mocha gère les tests unitaires (equiv de ***JUnit*** en Java)

Ajouter la dépendance mocha au projet : npm install mocha --save

Créer un dossier test

Les fichiers sont nomes : **monFichier.spec.js**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Symbole | Signification | Code | | Utilisation | |
| NodeJS |  | | | | |
|  | en ligne de commande |  | |  |  |
| node | Lance l'application | node application.js | |  |  |
| npm init | initialise le projet | npm init -y | |  |  |
| npm install | télécharge la dépendance en local, cad dans le dossier node\_modules du projet | npm install monModule --save | |  | ATTENTION : doit être joué dans le dossier du projet lui-même ! |
|  | télécharge la dépendance en local en mode développeur | npm install monModule --save-dev | |  |  |
| -g | instaler la dépendance en global, cad dans Node et accessible par tous les projets | npm install -g monModule | |  |  |
|  | désinstalation de la dépendance globale | npm uninstall -g monModule | |  |  |
|  |  |  | |  |  |
|  | **TypeScript** |  | |  |  |
| typescript | instalation de TypeScript en global | npm install -g typescript | | tsc -v tsc --help |  |
| tsconfig.json | fichier qui définit les options de transpilation de TypeScript en JavaScript |  | |  |  |
|  |  |  | |  |  |
|  | Mocha = Tests unitaires |  | |  |  |
| mocha | lancement des tests unitaires  si dans package.json  "scripts": {  "test": "mocha" | npm node\_modules/.bin/mocha  *alors lancement des tests avec :*  mpn mocha | |  |  |
| chai | permet d'ajouter des fonctionnalités aux tests de base | npm install chai --save-dev | var expect = require('chai').expect;  describe('Math.addition', function() {  describe('#without arguments', function() {  it('should return 0 when have no arguments', function() {  expect(function() {  math.addition()}).to.not.throw();  expect(math.addition()).to.equal(0);  });  });  describe('#agrgument is not a number', function() {  it('should ignore argument that is not a number', function() {  expect(function() {  math.addition(2, 4, 'notNumber')}).to.not.throw();  expect(math.addition(2, 4, 'notNumber')).to.equal(6);  });  });  }); | |  |
|  |  |  | |  |  |

TypeScript

Principe :

Langage entre le langage type (comme Java) et le JavaScript.

Le TypeScript est d'abord transpilé en JavaScript avant d'être chargé dans le navigateur ou le serveur.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Symbole | Signification | Code | Utilisation | |
|  |  | | | |
| Variables |  |  |  |  |
| let | variable remplaçant var | let monObjet;  let monObjet: MonObjet;  let monObjet: MonObjet = new MonObjet();  for(let i = 0; i < 10; i++) {  *instructions*  } | let monObjet: any;  let monNombre: number;  let maString: string;  let maListe: Array<any> = [ ];  let maListe: Array<any> = new Array<any>(); |  |
| const | constante, objet non mutable cad référence mémoire qui ne change pas, mais peut ajouter/suppr les propriétés dynamique comme avec une var | const monObjet: MonObjet = new MonObjet();  const maListe: Array<MonObjet> = [ ]; |  |  |
|  |  |  |  |  |
| String |  |  |  |  |
| ` `  ` ${ } ` | déclaration de chaine de caractère simplifiant la concaténation via les EL | let nb1 = 1, nb2 = 2;  let msg = 'Valeur de';  calculerSomme() {  return `${msg} la somme = ${nb1 + nb2}`;  } |  |  |
|  |  |  |  |  |
| Tuple |  |  |  |  |
| [ , , ] | permet de définir les types d'un nombre défini d'éléments dans un tableau | cont monTuple = [string, number, boolean];  monTuple[1] = 3;  monTuple[0] = 'Ma chaine'; | monTuple[1] = 'Ma chaine' *// => erreur* |  |
|  |  |  |  |  |
| Tableau |  |  |  |  |
| … | tableau contenant plusieurs paramètres | maFunction(arg1, …args) {  maFunction(arg1: string, …args: Array<string>) {  function f (x, y, ...arr) {  return (x + y) \* arr.length;  } | *appel à la fonction avec plusieurs paramètres :*  const valeur = f(1, 2, 3, 4, 5); *// => 6*  *appel à la fonction avec un tableau :*  const monTableau = ['a', 'b', 'c'];  const valeur = f(1, 2, 3, …monTableau); *// => 12* |  |
|  | clone les items d'un tableau dans un autre tableau | const params = ['hello', true, 7];  const other = [1, 2, ...params]; *// => other = [1, 2, 'hello', true, 7]* |  |  |
|  | clone les propriété d'un objet dans un autre objet | |  |  | | --- | --- | | var monObjetA = {  prop1: val1,  prop2: val2  } | var monObjetB = {  ...monObjetA,  prop3: val3  } | |  |  |
|  |  |  |  |  |
| Fonction |  |  |  |  |
|  | paramètres de fonctions avec des valeurs par défauts si non précisées lors de l'appel de la fct | function f (x, y = 2, z = 3) {  return x + y + z;  } |  |  |
| ? | paramètre facultatif | function f (x, y?, z?) {  const result = x;  if(y) { result += y);  if(z) { result += z);  return result;  } |  | ATTENTION : les paramètres facultatifs doivent obligatoirement être placés en dernier dans la liste des paramètres ! |
| Arrow function | (= lambda en Java) | Remarque : fonction anonyme mais ne créant pas de nouveau scope !!! | |  |
| => | 1 argument - retourne 1 valeur  plusieurs arguments - retourne 1 valeur  1 argument - retourne 1 objet  1 argument - réalise plusieurs instructions | (e => e + 1)  ((e, f) => e + f)  (e => ({prop1: val1, prop2: val2}))  (e => {  instruction(s);  return e;  }) |  |  |
| Objet |  |  |  |  |
|  | simplification d'initialisation de variable | var x = 1, y = 2;   |  |  |  | | --- | --- | --- | | monObjet = {  x,  y  } | ⇔ | monObjet = {  x = x,  y = y  } | |  |  |
|  | nom de propriété dynamique | function monCalcul(x, y) {  return x + y;  }  let monObjet = {  prop1: val1,  ['prop' + monCalcul(1, 1)]: val2  } |  |  |
|  | définition de paramètre implicitement | monObjet = {  constructor(public maProp1 : string, private maProp2 : number = 2) ; ⇔  } | monObjet = {  maProp1 : string ;  maProp2 : number = 2 ;  constructor(valeur1 : string, valeur2 : number) {  this.maProp1 = valeur1 ;  if(valeur2) {  this.maProp2 = valeur2 ;  }  }  } |  |
|  | définition de méthode implicitement | monObjet = {  maFonction(x, y) { ⇔  instruction(s);  } | monObjet = {  maFonction: function(x, y) {  instruction(s);  } |  |
|  | constructeur d'objet et héritage | class MonObjet {  constructor (id, x, y) {  this.id = id;  this.move(x, y);  }  move (x, y) {  this.x = x;  this.y = y;  }  } | class MonObjetB extends MonObjet {  constructor (id, x, y, z) {  super(id, x, y);  this.z = z;  }  } |  |
| get  set | définir des getteur et setter manuellement | class MonObjet {  private \_maProp1: string;  get maProp1(): string {  return this.\_maProp1;  }  set maProp1(value: string) {  if(value) {  this.\_maProp1 = value;  }  }  } | *utilisation des getters et setters* ***sans get et set*** *:*  monObjet.maProp1 = 'Ma chaine';  console.log(monObjet.maProp1); |  |
|  |  |  |  |  |
| interface |  |  |  |  |
|  | le nom de la prop sera défini lors de la création de l'objet | interface MonInterface {  maProp1: MonObjet;  maProp2: string;  [maPropDeNomInconnu: string]: any;  maPropOptionnelle1?: number;  maFonction(monParam1: string): void;  } |  |  |
| readOnly | équivalent de const pour les propriétés des objets | interface MonInterface {  maProp1: MonObjet;  readonly maProp2: string; |  |  |
|  |  |  |  |  |
| Fonction Générator |  | voir => https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/function\* | |  |
| fonction\* | fonctionne comme le mode debug | function\* monGenerator(i) {  *instruction(s);*  yield i + 1;  *instruction(s);*  yield i + 10;  return result;  }  var gen = monGenerator(10); *=> retourne 11*  gen.next(); *=> retourne 20*  gen.next(); *=> retourne result* |  |  |
|  |  |  |  |  |
| Destructuring | simplifie la syntaxe du code |  |  | Rq: à utiliser avec parcimonie car rend le code moins lisible et plus complexe à comprendre |
| const { } = | ne récupère que les valeurs des props qui m'intéressent de l'objet | const {prop1, prop5 = 'valDefaut', prop6} = maFctQuiRetourneUnObj(); |  | Rq: si prop5 n'est pas trouvée dans l'objet, elle prendra la valeur 'valDefaut' par défaut |
| const { prop: variable | récupère les valeurs et les place dans des constantes pouvant être utilisées directement | const {prop1: a, prop5: b, prop6 } = maFctQuiRetourneUnObj();  if(a > 5) {  ...  } |  |  |
|  | utilisation des propriétés d'un objet ou des éléments d'un tableau dirrectement | function({obj1, obj2}) {  console.log(obj1, obj2);  } | function([obj1, obj2]) {  console.log(obj1, obj2);  } |  |
|  |  |  |  |  |
| Window |  |  |  |  |
|  | changer le titre de l'onglet | window.parent.document.title = 'Nouveau titre'; |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **TypeScript** |  |  |  |
| Codes utiles |  |  |  |  |
|  |  |  |  |  |
| Fonctions outils | plusieurs fonctions outils pratiques | /\*\*  \* Fonction utilitaire determinant si l'objet passé en paramètre est null ou undefined.  \* @param value objet de n'importe quel type  \* @return true si l'objet est null ou undefined, false sinon.  \*/  export function isNullOrUndefined(value: any): boolean {  return value === null || value === undefined;  }  /\*\*  \* Fonction utilitaire déterminant si au moins un des objets passés en paramètre est null ou undefined.  \* @param values nombre variable d'objet de n'importe quel type  \* @return true si au moins un des objets est null ou undefined, false sinon.  \*/  export function anyIsNullOrUndefined(...values): boolean {  return values.some(value => isNullOrUndefined(value));  }  /\*\*  \* Fonction utilitaire déterminant si au moins un des tableaux/liste passés en paramètre est null ou undefined ou vide.  \* @param values nombre variable de tableaux de n'importe quel type  \* @return true si au moins un des tableaux est null ou undefined ou ne contenant aucun élément, false sinon.  \*/  export function anyIsNullOrEmpty(...tables: Array<Array<any>>): boolean {  return tables.some(table => isNullOrUndefined(table) || table.length === 0);  }  /\*\*  \* Fonction utilitaire déterminant si au moins une des chaines passées en paramètre est null ou undefined ou est une chaine vide.  \* @param values nombre variable de string  \* @return true si au moins une chaines est null ou undefined ou est une chaine vide, false sinon.  \*/  export function anyStringIsNullOrEmpty(...values: Array<string>): boolean {  return values.some(value => isNullOrUndefined(value) || value.trim().length === 0);  } | | |
| StringFormat | remplace des tags par des valeurs dans une chaine de caractère à la manière de String.format( ) en Java | /\*\*  \* stringFormat  \* Fonction utilitaire permettant de remplacer des tags dans une chaine de caractère par des valeurs  \* Format des tags : {0}, {1}, {2}, ...  \*  \* Exemple d'utilisation :  \* let maChaine = stringFormat('Ceci est un {0} d\'utilisation de la {1} utilitaire formatString avec {2} valeurs remplacées dans la chaine', 'exemple', 'fonction', 3);  \* console.log(maChaine);  \*  \* Résultat dans la console :  \* Ceci est un exemple d'utilisation de la fonction utilitaire formatString avec 3 valeurs remplacées dans la chaine  \*  \* @param formatString chaine de caractère contenant les tags  \* @param replacements valeurs à insérer dans la chaine  \*/  export function stringFormat(formatString: string, ...replacements: any[]): string {  replacements.forEach((replacement, index) => {  formatString = formatString.replace(`{${index}}`, replacement);  });  return formatString;  } | | |
| Map | convertir une map en un objet  convertir un objet JSON en une map | /\*\*  \* convertMapToObject  \* Fonction utilitaire convertissant une map de type <string, any> en un objet JavaScript  \*  \* @param map de type <string, any>  \* @returns {any} objet ayant pour propriété les clé de la map  \*/  export function convertMapToObject(map: Map<string, any>): any {  let object = {};  map.forEach((value, key) => object[key] = value);  return object;  }  /\*\*  \* convertJsonObjectToMap  \* Fonction utilitaire convertissant un jsonObjet en Map<string, any>  \*  \* @param jsonObject  \* @returns {Map<string, any>} ayant pour clé les propriété du jsonObjet  \*/  export function convertJsonObjectToMap(jsonObject: JSON): Map<string, any> {  let map: Map<string, any> = new Map<string, any>();  for (let key in jsonObject) {  map.set(key, jsonObject[key]);  }  return map;  } | | |
|  |  |  | | |
| Deserializable | désérialisation d'un JSON en un objet incluant les méthodes de la classe du même type | *-------------------------------------- interface Deserializable*  interface Deserializable {  getTypes(): Object;  }  */\* Interface servant à marquer les Beans pour la deserialisation avec la fonction deserialize(jsonObject, MonObjet)*  *Exemple d'utilisation :*  *export class CoordonneesDestinataire implements Deserializable {*  *destinataire: Destinataire;*  *adressesPostales: Array<AdressePostale>;*  *numTelelphone: number;*  *adressesCourriels: Array<string>;*  *coordonneeVerifiee: boolean;*  *public getTypes() {*  *return {*  *destinataire: Destinataire,*  *adressesPostales: Array,*  *adressesPostalesArray: AdressePostale,*  *adressesCourriels: Array*  *}*  *}*  *}*  *Dans le retour de getType() :*  *- seules les prorpiétés de type 'object' doivent être indiquées, pas les types primitifs*  *- toutes les propriétés de type Array doivent être indiquées*  *- mais seuls les Array contenant un type 'object' doivent EN PLUS être indiqués avec le format : 'nomProprieteArray: TypeContenuTableau'*  *Remarques :*  *~ les Beans doivent pouvoir être instanciés à vide (constructeur sans paramètre ou uniquement avec paramètres facultatifs !)*  *~ toutes les propriétés de type 'object' doivent implémenter Deserializable (dans l'exemple, Destinataire et AdressePostale doivent implémenter Deserializable)*  *Si un Bean ne contient que des types natifs, getType() retourne alors un objet vide.*  *export class AdressePostale implements Deserializable {*  *ligne1: string;*  *ligne2: string;*  *codePostal: number;*  *ville: string;*  *pays: string;*  *public getTypes() {*  *return {}*  *}*  *}*  *Exemple de déserialisation :*  *obtenirCoordonneesDestinataire(idDestinataire: string): Promise<CoordonneesDestinataire> {*  *return new Promise<CoordonneesDestinataire>((resolve, reject) => {*  *this.diffusionAdapter.getCoordDesinataire(idDestinataire).subscribe((response) => {*  *//déserialisation d'un objet*  *const obj: CoordonneesDestinataire = deserialize(response, CoordonneesDestinataire);*  *resolve(obj);*  *}, (error) => {*  *reject(error);*  *});*  *});*  *}*  *//ou déserialisation d'un tableau d'objet*  *const obj: Array<Document> = deserializeArray(response, Document);*  *\*/*  *----------------------------- méthodes utilitaires de deserialisation utilisant l'interface Deserializable :*  export function deserialize(jsonObject, clazz) {  const instance = new clazz(),  types = instance.getTypes();  for (let prop in jsonObject) {  if (isNullOrUndefined(jsonObject[prop])) {  continue;  }  if (typeof jsonObject[prop] !== 'object') {  instance[prop] = jsonObject[prop];  } else {  if (jsonObject[prop] instanceof Array) {  instance[prop] = deserializeArray(jsonObject[prop], types[prop.concat('Array')]);  } else {  instance[prop] = deserialize(jsonObject[prop], types[prop]);  }  }  }  return instance;  }  export function deserializeArray(jsonArray, arrayContentClazz) {  let table = new Array();  if (!isNullOrUndefined(jsonArray)) {  //s'il s'agit d'un tableau de primitifs, on copie simplement le tableau avec son contenu  if (isNullOrUndefined(arrayContentClazz)) {  table = jsonArray;  } else {  //pour un tableau d'objet, on déserialise chaque objet avant de l'ajouter au tableau  jsonArray.forEach(jsonObject => {  table.push(deserialize(jsonObject, arrayContentClazz));  });  }  }  return table;  } | | |
| ValidableBean |  | *-------------------------------------- interface ValidableBean*  interface ValidableBean {  isValid(): boolean;  }  *----------------------------- méthode utilitaire de validation des objets utilisant l'interface ValidableBean :*  /\*\*  \* Fonction utilitaire spécifique aux Beans implémentant l'interface ValidableBean.  \* Elle vérifie que tous les éléments passés en paramètres sont ni null ni undefined avant d'appeler la méthode isValid() pour chacun d'eux.  \* @param values nombre variable de bean de type ValidableBean  \* @return true si tous les beans sont valide selon leur propre méthode. False sinon.  \*/  export function areAllValid(...beans: Array<ValidableBean>) {  if (anyIsNullOrUndefined(…beans)) {  return false;  }  return beans.every(bean => bean.isValid());  } | | |
|  |  |  |  |  |

Angular

Principe :

Outil permettant de structurer un projet JavaScript ou TypeScript gérant l'affichage d'une application en autonomie côté client (SAP).

Il se base sur la création de composants visuels (HTML, CSS et code TypeScript) interagissant les uns avec les autres.

(equiv de Spring ou SpringBoot pour Java avec fichier de config, structure conventionnelle du projet, …)

SAP = Single Page Application

1 application = une seule page dont l'état varie en fonction des actions de l'utilisateur sans re-appeler le serveur

Modification de l'état de la page par JavaScript (DOM, CSS, ...)

Rq: l'URL est redéfinie en fonction de l'état de l'appli.

Angular-cli

*(CLI = Command Line Interface)*

Utilitaire en ligne de commande lié à Angular et permettant :

- créer projet

- créer des composants, des services

- packaging de l'appli

- gestion des environnements

- tests (plus besoin d'instaler Mocha, ... comme en pur Node.JS)

- confort de dev (watch des fichiers et actualisation à chaque changement)

=> architecture standardisée pour tous projets Angular (contrairement à AngularJS)

(equiv de Maven ou Gradle pour Java)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Symbole | Signification | Code | | | | | Utilisation | | | | | | | | |
| Angular |  | | | | | | | | | | | | | | |
|  |  |  | | | |  | | | | |  | | | | |
| Angular-cli |  |  | | | |  | | | | |  | | | | |
| @angular/cli | installation d'Angular-cli en global | npm install -g @angular/cli | | | |  | | | | |  | | | | |
| ng new | créer un nouveau projet Angular | ng new monProjet | | | | *définir le type de feuilles de style et supprimer les fichiers test*  ng new monProjet --style=scss --skip-tests=true | | | | |  | | | | |
| git init | Rq: après l'étape de création, créer un repo Git avec le contenu de ***.gitignore*** | *contenu de* ***.gitignore*** *:*  node\_modules  .idea  package-lock.json | | | |  | | | | |  | | | | |
| npm install | installation d'une dépendance dans le projet (= intégration au package.json)  ATTENTION : doit être lancée dans le dossier du projet lui-même ! | npm install maDependance@3.3.0 --save | | | *ex avec bootstrap :*  npm install bootstrap@3.3.7 --save  *puis dans angular.json :*  "styles": [  "../node\_modules/bootstrap/dist/css/bootstrap.css",  "styles.scss"  ] | | | | | | ATTENTION : la commande ajoute la dépendance au projet, mais pas à l'application. Pour cela il faut renseigner le dossier de la dépendance au bon endroit dans le fichier angular.json | | | | |
| ng serve | démarre l'application en mode de développement, cad rechargement de la page dans le navigateur à chaque modification des fichiers | ng serve | | | | *Affichage dans le navigateur :*  http://localhost:4200/ | | | | |  | | | | |
| ng generate | création d'un nouvel élément (component, service, pipe, directive, …) | ng generate component monComposant  ng generate service monService  ng generate pipe monPipePerso  ng generate directive maDirectivePerso | | | |  | | | | | Rq: =>  - création du dossier monComposant dans src/app/  - MàJ du fichier src/app/app.modules.ts | | | | |
|  |  |  | | | |  | | | | |  | | | | |
| internationnalisation | traduire le contenu des pages selon la localisation du client | ng add @angular/localize | | | |  | | | | |  | | | | |
| i18n | atribut permettant de taguer le texte à traduire | <h1 i18n="title">Application utilisateur</h1> | | | |  | | | | | Rq: si la localisation du client n’a pu être détectée, c’est le contenu de la balise qui s’affichera par défaut | | | | |
| ng extract-i18n | commande permettant d’extraire le contenu de tous les éléments tagués avec i18n | ng extract-i18n | | | | *Résultat de l’extraction dans un fichier :*  {  "locale": "fr-FR",  "translations": {  "title": "Application utilisateur"  } | | | | |  | | | | |
| "locales": {  "localize": true | ajout dans le paramétrage angular.json des fichiers de traduction pour chaque localité  passage de l’option ‘localize’ à true pour récupérer la localisation du client | {  "projects": {  "monApplication": {  "i18n": {  "sourceLocale": "fr-FR",  "locales": {  "us": "src/locale/messages.us.json"  }  },  "build": {  "options": {  "localize": true  }  }  }  }  } | | | |  | | | | |  | | | | |
| --configuration= | option permettant de démarrer le serveur en forçant la localisation | ng serve --configuration=us | | | |  | | | | |  | | | | |
|  |  |  | | | |  | | | | |  | | | | |
| json |  |  | | | |  | | | | |  | | | | |
| resolveJsonModule | paramètrage à ajouter au fichier tsconfig.json pour pouvoir importer et utiliser des fichiers .json en static | "compilerOptions": {  "resolveJsonModule": true | | | |  | | | | |  | | | | |
| import \* as | importation et utilisation d'un fichier .json | import \* as monObjet from 'src/assets/monFichier.json'; | | | | *ex de mock du retour d'un objet suite à une requête:*  getMonObjet(): Observable<any> {  return new Observable(suscriber => {  setTimeout(() => {  suscriber.next(monObjetJsonBouchon);  suscriber.complete();  }, 5000);  });  } | | | | | Rq: il est important d'appeler la méthode .complete() lorsque la totalité des données a été envoyée, en particulier si ces données sont contrôlées par un Resolver. Sans cela, le Resolver ne valide jamais la redirection ! | | | | |
|  |  |  | | | |  | | | | |  | | | | |
| Affichage |  |  | | | |  | | | | |  | | | | |
|  | **Data Binding** |  | | | |  | | | | |  | | | | |
| {{ }} | interpolation = accès à une propriété de l'objet TS du composant depuis le HTML | import { Component } from '@angular/core';  @Component({  selector: 'monComposant',  templateUrl: './adresseFichier.html',  styleUrls: ['./adresseFichier.css']  })  export class MonComposant {  maProp1 = 'valeur1';  maProp2 = 53;  isDisabled = true;  getMaProp2() {  return this.maProp2;  }  constructor() {}  } | | | | <p>{{maProp1}}</p>  <p>{{getMaProp2()}}</p> | | | | |  | | | | |
| [ ] | property binding / liaison par propriété | <p [textContent]="maProp1"></p>  <p [textContent]="getMaProp2()"></p>  <button [disabled]="isDisabled">Mon bouton</button> | | | | |  | | | | |
| ? | paramètre optionnel |  | | | | <p [textContent]="user?.name"></p> | | | | | lorsque la propriete name est undefined, <p></p> est créé mais n'a pas de contenu sans lever d'erreur  Rq: si l'on ne veut pas que <p></p> soit créé si name n'existe pas, il faut utiliser \*ngIf=" " *(voir plus bas)* | | | | |
| ( ) | event binding / liaison par évènement | @...  export class MonComposant {  onClick(event):void {  console.log('onClick', event);  }  … | | | | <div>  <button (click)="onClick($event)">Click me</button>  </div> | | | | | Rq : $event est facultatif. Utile si l'on souhaite récupérer l'évènement et ses propriétés pour le traitement | | | | |
| [( )] | 2 way binding / liaison double sens  lie les valeurs du composant dans les 2 sens :   * TS -> HTML * HTML -> TS | @...  export class MonComposant {  maProp1:string = 'maValeurInitiale';  monObjet: MonObjet = new MonObjet();  … | | | | <input [(ngModel)]="maProp1" placeholder="Ma valeur initiale" /> | | | | | Rq: lorsque l'utilisateur remplira le champ, la valeur sera immédiatement mise à jour dans le TS.  ATTENTION : La valeur de la propriété doit obligatoirement être initialisée (éventuellement avec un objet vide)  ATTENTION : utilisable uniquement si FormsModule est importé dans app.modules.ts  import { FormsModule } from '@Angular/forms';  @NgModule({  …  imports: [  FormsModule,  … | | | | |
|  | **Directives structurelles** |  |  | | | | | | | |  | | | | |
| # | déclaration de variable |  | <input type="text" #monInput placeholder="Saisir une valeur">  <div>La valeur de l'input est : {{monInput.value}}</div> | | | | | | | |  | | | | |
| \*ngIf | if | @...  export class MonComposant {  maProp1:string = 'valeur1';  skills:Array<string> = ['str1', 'str2', 'str3'];  … | | | | <!-- if -->  <div \*ngIf="maProp1.length !== 0">  <p>Propriété1 : {{maProp1}}</p>  </div>  <div \*ngIf="maProp1.length !== 0; else pDefault">  <p>Propriété1 : {{maProp1}}</p>  </div>  <ng-template **#pDefault**>  <p>Propriété1 absente !</p>  </ng-template> | | | | | Rq: si la condition est fausse, l'élément n'existe pas dans le DOM (il n'est pas seulement caché, il est supprimé du DOM) | | | | |
| \*ngFor | boucle for | <!-- for -->  <ul>  <li \*ngFor="let skill of skills ; index as i">  {{i}} - {{skill}}  </li>  </ul>  <user-component \*ngFor="let user of users" [userName]="user.name" [userLogin]="user.login"></user-component> | | | | | Rq: options possibles dans la boucle for :  **index** index de l'élément courant  **even** =true si l'élément a un index pair  **odd** =true si l'élément a un index impair  **first** =true si l'élément est le premier de la collection  **last** =true si l'élément est le dernier | | | | |
| \*ngSwitch | switch/case | <!-- switch/case -->  <div [ngSwitch]="maProp1.length">  <p \*ngSwitchCase="0">Prop1 is empty</p>  <p \*ngSwitchCase="1">Prop1 as one character</p>  <p \*ngSwitchDefault>Prop1 has many characters</p>  </div> | | | | |  | | | | |
|  | **Directives par attribut** |  | | | |  | | | | |  | | | | |
| ngStyle |  | @…  export class MonComposant {  fontWeight:string = 'bold';  textColor:string = 'green';  … | | | | *sans le property binding :*  <p [style.color]="'red'">Mon texte en rouge</p>  *ou*  <p [ngStyle]="{'color': 'red'}">Mon texte en rouge</p>  *avec le property binding :*  <p [ngStyle]="{'fontWeight': fontWeight, 'color': textColor}">Texte en gras et vert</p> | | | | |  | | | | |
| ngClass | ajoute/supprime des valeurs à l'attribut class | @…  export class MonComposant {  isTxtRed:boolean = true;  isBackgroundBlue:boolean = false;  … | | | | <p [ngClass]="{'txtRed': isTxtRed, 'backgroundBlue': isBackgroundBlue }">Texte mis en forme</p>  *fichier CSS :*  .txtRed {  color: red;  }  .backgroundBlue {  background-color: blue;  } | | | | |  | | | | |
|  | **Directives personnalisée** |  | | | | | | | | |  | | | | |
| @Directive | définit une directive personnalisée | import { Directive, ElementRef, Renderer2 } from '@angular/core';  @Directive({  selector: '[appHighlight]'  })  export class HighlightDirective {  constructor(el: ElementRef, renderer: Renderer2) {  renderer.setStyle(el.nativeElement, 'color', 'green');  renderer.setStyle(el.nativeElement, 'text-decoration', 'underline');  }  }  -----------------------------------------  @Directive({  selector: '[appDynamicHighlight]'  })  export class DynamicHighlightDirective {  constructor(private el: ElementRef, private renderer: Renderer2) {}  @HostListener('mouseenter')  highlight() {  this.renderer.setStyle(this.el.nativeElement, 'background-color', 'green');  }  @HostListener('mouseleave')  unHighlight() {  this.renderer.setStyle(this.el.nativeElement, 'background-color', '');  }  } | | | | | | | | | *utilisation dans le fichier HTML :*  **<p** appHighlight**>**Mon texte décoré</p> | | | | |
|  | **Pipes pour la mise en forme** |  | | | |  | | | | |  | | | | |
| | | | pipe  sorte de filtre permettant la mise en forme d'un élément à afficher sans en modifier la nature | |  |  | | --- | --- | | pipe | affiche | | | json | les données en JSON | | | slice:1:5 | ca. ou éléments d'une collection compris entre 2 index | | | uppercase | en majuscule | | | lowercase | en minuscule | | | titlecase | 1re lettre de chaque mot en maj | | | number: | indique nb de chiffre ap virgule | | | percent | en pourcent | | | currency: | nb dans la devise indiquée | | | date | formate une date | | | async | affiche les données quand elles sont dispo | | | | | <p>{{monObjet | json }}</p>  <p> {{ monTitre | uppercase }}</p>  <p>Prix: {{monPrix | currency:'EUR'}}</p>  <p>Date: {{ currentDate | date }}</p>  <p>Date: {{ currentDate | date:'dd/MM/yyyy' }}</p>  <p>Date: {{ currentDate | date:'longDate' }}</p>  <p>{{maPromise | async }}</p>  *succession de pipes :*  <p>{{maPromise | async | date | uppercase }}</p> | | | | | ATTENTION : lors de la succession de plusieurs pipes, l'ordre est important ! | | | | |
| @Pipe | définit un pipe personnalisé | *Exemple de pipe perso qui affiche la 1re lettre de chaque mot en majuscule :*  import { Pipe, PipeTransform } from '@angular/core';  import \* as \_ from 'lodash';  /\*\*  \* Display the string element with the first letter of each word in uppercase  \* Usage:  \* value | inTitleCase  \* Exemple:  \* {{ 'my title' | inTitleCase}}  \* Display: My Title  \*/  @Pipe({  name: 'inTitleCase'  })  export class InTitleCasePipe implements PipeTransform {  transform(value: string, args?:any): string {  let words:Array<String> = \_.split(value, ' ');  let txt:string = '';  for(let word:string of words) {  txt += \_.upperFirst(word) + ' ';  }  return txt;  }  } | | | | | | | | | *utilisation dans le fichier HTML :*  <p>{{MonObjet.title | inTitleCase }}</p>  => args récupère les param facultatifs :  {{ monTitre | inTitleCase: 5 : 6 }} | | | | |
|  |  |  | | | |  | | | | |  | | | | |
| Communication entre composants |  |  | | | |  | | | | |  | | | | |
| @Input() | récupère la valeur d'une propriété transmise par le composant parent | import { Component, Input } from '@angular/core';  @Component({  selector: 'monComposantEnfant',  templateUrl: './adresseFichier.html',  styleUrls: ['./adresseFichier.css']  })  export class **MonComposantEnfant** {    @Input()  **maPropEnfant**:string;  @Input()  **maCompetence**:Competence;    constructor() {}  } | | | | *HTML du* ***composant parent*** *:*  <monComposantEnfant[maPropEnfant]="getMaPropParent()"></monComposantEnfant>  *Exemple concret avec une liste :*  <ul>  <li \*ngFor="let competence of competences">  <**monComposantEnfant** **[maCompetence]=**"competence"></monComposantEnfant >  </li>  </ul> | | | | | => transmission de l'objet "formation" au composant enfant "formation-item" chargé d'affiché les détails d'un objet de type Formation | | | | |
| @Input()  get | input avec getter et setter  dans le cas d'un input sur une variable présentant un getter et un setter, l'annotation @Input se place sur le getter ! | \_maVariable: any;  @Input()  get maVariable() {  return this.\_maVariable;  }  set maVariable(maValeur: any) {  this.\_maVariable = maValeur;  } | | | |  | | | | |  | | | | |
| @Output() | transmet un évènement du composant enfant à son composant parent | import { Component, Output, EventEmitter } from '@angular/core';  @Component({  selector: 'monComposant',  templateUrl: './adresseFichier.html',  styleUrls: ['./adresseFichier.css']  })  export class **MonComposantEnfant** {  **monObjet**:MonObjet = new MonObjet();  @Output()  **monEvent**:EventEmitter<MonObjet> = new EventEmitter<MonObjet>();    constructor() {}  **envoyerMonEvent**(event): void {  this.**monEvent**.emit(this.**monObjet**);  } | | | | *HTML du* ***composant enfant*** *:*  <button (click)="**envoyerMonEvent**($event)" >Cliquez ici</button>  *HTML du* ***composant parent*** *:*  <**monComposantEnfant**  [maPropParent]="getMaPropParent()"  (monEvent)="**handleMonEvent($event)**">  </monComposantEnfant>  *TS de l'élément parent :*  @...  export class MonComposantParent {  …  **handleCustomEvent(monObjet)** {  console.log('Objet transmis par le composant enfant :', monObjet);  }  } | | | | |  | | | | |
| @ViewChild | importe les propriétés et méthodes du composant enfant afin de pouvoir les utiliser directement dans le composant parent | *fichier TS composant parent :*  @ViewChild('**enfant**', {static: false})  monEnfant:**MonComposantEnfant**;  maMethodeParent:void {  valeur = monEnfant.maMethodeEnfant();  } | | | | *fichier HTML composant parent :*  <div>  <app-mon-composant-enfant **#enfant**></app-mon-composant-enfant>  </div> | | | | |  | | | | |
| <ng-content> | balise permettant la **projection de contenu** du parent vers l’enfant | *fichier HTML composant parent :*  <div>  **<app-mon-composant-enfant>**  <p>Contenu projeté dans le composant enfant</p>  **</app-mon-composant-enfant>**  </div> | | | | *fichier HTML composant enfant :*  <div>  <ng-content></ng-content>  </div> | | | | |  | | | | |
|  |  |  | | | |  | | | | |  | | | | |
| @HostListener | déclare un évenement du DOM à écouter | *@HostListener(‘click’)*  *onClick() {*  *console.log(‘Le bouton a été cliqué’) ;*  *}* | | | |  | | | | |  | | | | |
|  |  |  | | | |  | | | | |  | | | | |
| IOC | Inversion de contrôle = injection de dépendance |  | | | |  | | | | |  | | | | |
| providers: [] | dans le fichier ***app.modules.ts***  liste les providers, cad les classes services qui vont gérer l'injection dans les composants  => ces classes sont alors des singletons |  | | | |  | | | | |  | | | | |
| @Injectable() | déclaration du provider (= classe service) | *nécessitant de déclarer le provider dans* ***app.modules.ts*** *manuellement*  @Injectable()  export default class MonProvider {  ...  } | | | | *sans avoir à déclarer le provider dans* ***app.modules.ts***  @Injectable({  provideIn: 'root'  })  export default class MonProvider {  ...  } | | | | |  | | | | |
| constructor( ) | injection d'un provider dans un composant | construcor(private **api**:MonProvider) {} | | | |  | | | | |  | | | | |
|  |  |  | | | |  | | | | |  | | | | |
| Réseaux |  |  | | | |  | | | | |  | | | | |
| config | import de de HttpClientModule dans ***app.modules.ts*** | import { HttpClientModule } from '@angular/common/http';  …  imports: [  HttpClientModule  ], | | | |  | | | | |  | | | | |
| HttpClient | créer des requêtes HTTP  méthodes dispo :  get, post, put, delete, patch, head, jsonp | @Injectable()  export class MonHttpService {  constructor(private http: HttpClient) {}  BASE\_URL: string = 'https://monurl.com/';  USERS: string = 'users/';  getUser(login:string) {  return this.http.get(`${BASE\_URL}${USERS}${login}`);  }  postUser(user) {  return this.http.post((`${BASE\_URL}${ USERS}`, {firstname: user.firstname, lastname: user.lastname, pseudo: user.pseudo}));  }  deleteUser(id:number) {  params = new HttpParams().set('id', id);  return this.http.delete((((`${BASE\_URL}${ USERS}`, {params});  }  } | | | | ***Récupération des données retour via Observable :***  user:User;  httpService.getUser('pseudo')  .subscribe((result:any) => {  this.user = result;  }, (error) => {  console.log('erreur : ' + error);  }, () => {  console.log('complete');  });  ***=> ajout d'un objet JS (format JSON) dans le corps de la requête.***  ***=> requête avec paramètre(s)*** | | | | | Rq: la requête retourne un Observable qui peut éventuellement être transformé en Promise via .toPromise() | | | | |
|  |  |  | | | |  | | | | |  | | | | |
| Interceptor | equiv des filtres en JEE |  | | | |  | | | | |  | | | | |
| config | déclaration manuelle obligatoire de chaque interceptor dans ***app.modules.ts*** | providers: [  { provide: HTTP\_INTERCEPTORS, useClass: HeaderInterceptor, multi: true },  { provide: HTTP\_INTERCEPTORS, useClass: ErrorInterceptor, multi: true }  ], | | | | | | | | |  | | | | |
| HttpInterceptor | intercepteur  Rq: un intercepteur doit être crée manuellement (pas de ng generate interceptor) | import {Injectable} from '@angular/core';  import {HttpEvent, HttpHandler, HttpInterceptor, HttpRequest} from "@angular/common/http";  import {Observable} from "rxjs";  @Injectable()  export class HeaderInterceptor implements HttpInterceptor {  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {  // Le traitement est déclenché si la requête sur l'url monurl.com  if (req.url.includes('monurl.com')) {  // Ajout du header Content-Type  // => Il est important de noter qu'une requête est immuable, vous devez la cloner  const clone = req.clone({setHeaders: {'Content-Type': 'application/json'}});  return next.handle(clone);  }  return next.handle(req);  }  } | | | | | | | | |  | | | | |
|  | intercepteur permettant la gestion des erreur | import {Injectable} from '@angular/core';  import {HttpErrorResponse, HttpEvent, HttpHandler, HttpInterceptor, HttpRequest} from "@angular/common/http";  import {Observable, throwError} from "rxjs";  import {catchError} from "rxjs/operators";  @Injectable()  export class ErrorInterceptor implements HttpInterceptor {  constructor() {}  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {  return next.handle(req).pipe(  catchError((errorResponse: HttpErrorResponse) => {  if (errorResponse.status === 404) {  // TODO: Gérer l'erreur  }  return throwError(errorResponse);  }));  }  } | | | | | | | | |  | | | | |
|  |  |  | | | |  | | | | |  | | | | |
| Router | gère les états de la page et les URL associés sans faire d'appel réseaux  1 URL = 1 route = 1 état de l'appli | navigation prise en charge par framework JavaScript | | | |  | | | | |  | | | | |
| RouterModule | config dans ***app.modules.ts*** | import { RouterModule } from '@Angular/router';  import { ROUTES } from './app.router.module';  imports: [  RouterModule.forRoot(**ROUTES**),  ...  ], | | | |  | | | | |  | | | | |
| app.router.module.ts | module généré lors de la création du projet et gérant les routers | import { Routes } from '@Angular/router';  import …  export const **ROUTES**: Routes = [  { path: PATH\_HOME, component: HomeComponent, pathMatch: ‘full’ },  { path: PATH\_USER, canActivate: [MonGuard], component: UserComponent }, *//voir "Guard"*  { path: PATH\_USER\_DETAILS, canActivate: [MonGuard], resolve: {MonResolver}, component: UserDetailsComponent } *//voir "Resolver"*  { path: PATH\_ERROR, component: ErrorComponent },  { path: ADMIN, loadChildren: () => import(‘./admin/admin.module’).then(m => m.AdminModule) },  { path: '\*\*', redirectTo: PATH\_ERROR }  ] | | | | | | | | | Rq: les path sont relatifs, cad adresse suivant celle du serveur (*ex: localhost:4200/****users***)  Rq: l’option **pathMatch** indique si le path doit   * full = avoir une correspondance parfaite * prefix = URL commencer par le path   Rq: **redirectTo** redirige vers un path existant de manière transparente dans la barre d'adresse.  Rq: loadChildren permet de charger les modules en mode lazy => le module n’est pas chargé à l’ouverture de l’appli, mais uniquement si le path vers cette page est utilisé par l’utilisateur  ATTENTION : il faut supprimer l’import du module dans app.module.ts pour que le lazy mode fonctionne !  ATTENTION : le path '\*\*' doit obligatoirement être placé en dernier dans la liste !  ATTENTION : les path doivent être des constantes écrites en dur et non pas des concaténations de plusieurs constantes ! | | | | |
| app.routes.module.constantes.ts | fichier à créer manuellement pour stocker les constantes liées aux URL de l'appli | export const PATH\_HOME = '';  export const PATH\_USERS = 'users/:id';  export const PATH\_USERS\_RESOURCES = 'ressources'; | | | | | | |  | | ATTENTION : ces constantes ne doivent pas être placées dans le module router ci-dessus, sinon génère des boucles d'injection de dépendances infinies  Rq: dans un path :param permet de définir un path param variable pouvant être facilement récupéré dans le composant | | | | |
| <router-outlet> | balise à placer dans le composant principal et les composant ayant des routes et qui sera remplacée par le module qui convient selon l'URL demandé par l'utilisateur | <router-outlet></router-outlet> | | | |  | | | | |  | | | | |
| Routing module secondaires |  | *Routing module principal de l’application :*  const routes: Routes = [  {path: 'music', loadChildren: () => import ('./music/music.module').then(m => m.MusicModule)}  ];  @NgModule({  imports: [RouterModule.forRoot(routes)],  exports: [RouterModule]  })  export class AppRoutingModule { }  *Routing du sous-module :*  const routes: Routes = [  {  path: '', *// correspond au path ‘music’ défini dans le routing principal*  component: MusicComponent,  children: [  {path: 'album/:id', component: AlbumDetailComponent},  {path: 'artist/:id', component: ArtistDetailComponent} *//composants qui s’insereront dans la balise* ***<router-outlet>*** *présent dans dans le template music.component.html*  ]  } ,  {  path: '\*\*',  redirectTo: '' *//redirige vers le MusicComponent pour toutes les adresses imprévues*  }  ];  @NgModule({  imports: [RouterModule.forChild(routes)],  exports: [RouterModule]  })  export class MusicRoutingModule { } | | | | | | | | | ATTENTION : il ne peut y avoir qu’un seul **RouterModule**.forRoot(routes) dans l’application. Les routes des sous-modules doivent être déclarées avec **RouterModule**.forChild(routes)  Rq: children: [] permet de déclarer des composants liés à des sous-URL qui seront injectés dans le composant lié à l'URL parent si ce dernier contient une balise <router-outlet> | | | | |
| routerLink  [routerLink] | lien géré par le router sans passer par un appel réseau | *path simple*  <p><a href="" routerLink="/">Home</a></p>  <p><a href="" routerLink="/users">Users</a></p> | | | | *path avec plusieurs sous-parties => tableau + property binding*  <p><a href="" [routerLink]="['/']">Home</a></p>  <p><a href="" [routerLink]="['/users']">Users</a></p>  <p><a href="" [routerLink]="['/details', user.id]">{{user.name}}</a></p> | | | | | | | | | Rq: Le slash de début dans le chemin est nécessaire. S’il n’est pas inclus, RouterLink construit une  URL relativement au chemin courant. |
| router  LinkActive | ajoute une classe CSS pour mettre en valeur le lien actif (utile not. pour la barre de navigation) | <p><a href="" [routerLink]="['/']" routerLinkActive="nav\_\_item\_selected">Home</a></p> | | | | | | | | |  | | | | |
| Router  .navigate( ) | contrôle de la navigation par JavaScript | @...  export class MonComposant {  user: User;  constructor(private router: Router) {}  navigateToUser(event) {  this.**router.navigate(**[PATH\_USERS, this.user.id]**)**;  }  } | | | | <button (click)="**navigateToUser($event)**>{{user.name}}</button> | | | | |  | | | | |
| ActivatedRoute  .paramMap | récupération de l'URL de navigation et de ses paramètres  -> path params  -> query params | *https://www.monappli.com/mapage/users/user123?idDocument=doc987* | | | | | | | | | | | |  | |
| *de manière statique :*  @...  export class MonCOmponent {  idUser: number;  constructor(private route: ActivatedRoute) {}  getPathParams() {  this.idUser = Number( this.route.snapshot**.paramMap**.get('idUser')));  this.idDocument = this.route.snapshot.**queryParamMap**.get('idDocument');  }  } | | | | *de manière dynamique et asynchrone avec Observable :*  @...  export class MonCOmponent {  idUser: number;  constructor(private route: ActivatedRoute) {}  getPathParams() {  this.route**.paramMap**.suscribe((params: ParamMap) => {  this.idUser = Number(params.get('idUser'))  });  this.route**.queryParamMap**.subscribe((qParams: ParamMap) => {  this.idDocument = qParams.get('idDocument');  });  }  } | | | | | | | |
|  |  |  | | | |  | | | | |  | | | | |
| Guard | contrôle l'accès aux routes |  | | | |  | | | | |  | | | | |
| canActivate: [ ] | déclaration d'un ou plusieurs Guard(s) contrôlant la route | export const **ROUTES**: Routes = [  { path: PATH\_USER, canActivate: [MonGuard], component: UserComponent }  ] | | | | | | | | |  | | | | |
| CanActivate | une Guard est un service implémentant CanActivate et sa méthode canActivate( ) renvoyant un booléen permettant ou non l'accès à la/les route(s) qu'elle contrôle | import { ActivatedRouteSnapshot, CanActivate, RouterStateSnapshot, Router } from '@angular/router';  import { Observable } from 'rxjs/Observable';  @Injectable()  export class MonGuard implements CanActivate {  constructor(private router: Router, private monService: MonService) {}    canActivate(route: ActivatedRouteSnapshot,  state: RouterStateSnapshot): Observable<boolean> | Promise<boolean> | boolean {  if(this.monService.monTest()) {  return true;  } else {  this.router.navigate(['/pageErreurOuAuthentification']);  }  }  } | | | | | | | | | Rq: une Guard étant un service, elle doit être déclarée dans ***app.modules.ts*** dans les **providers** | | | | |
|  |  |  | | | |  | | | | |  | | | | |
| Resolver | ne donne accès à la route que lorsque les informations ont bien étés récupérées (depuis un appel au serveur par exemple) | | | | |  | | | | |  | | | | |
| resolve: { }  resolve: [ ] |  | export const **ROUTES**: Routes = [  { path: PATH\_USER, canActivate: [MonGuard], component: UserComponent },  { path: PATH\_USER\_DETAILS, canActivate: [MonGuard], resolve: {MonResolver}, component: UserDetailsComponent }  ] | | | | | | | | | Rq: peut aussi se placer dans un tableau | | | | |
| Resolve | un Resolver est un service implémentant Resolve et sa méthode resolve( ) renvoyant une observable et ne permettant l'accès à la route uniquement lorsque les données ont été récupérées | import { Resolve, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';  import { Observable } from 'rxjs';  @Injectable()  export class MonResolver implements Resolve<any> {  constructor(private monServiceAdapter: MonServieAdapter) {}  resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<any> | Promise<any> | any {  return monServiceAdapter.faireAppelBackPourRecupererInfosInObservable();  }  } | | | | | | | | | Rq: un Resolver étant un service, il doit être déclarée dans ***app.modules.ts*** dans les **providers** | | | | |
| this.route.data.subscribe( ) | récupération des données retournées par le Resolver depuis le composant | import { ActivatedRoute } from '@angular/router';  @Component({…})  export class MonComponent implements OnInit {  dataDuResolver: any;  constructor(private route: ActivatedRoute) { }  ngOnInit(): void {  this.route.data.subscribe(data => this.dataDuResolver = data);  }  } | | | | | | | | |  | | | | |
|  |  |  | | | |  | | | | |  | | | | |
| Cycle de vie du composant |  |  | | | |  | | | | |  | | | | |
| constructor | à la création du composant |  | | | |  | | | | | Rq: à cet instant, les données issues des parents transmises par @Input() NE SONT PAS encore chargées, les champs ont leur valeurs par défaut => appels services back impossible ! A faire dans ngOnInit() | | | | |
| *ngOnChanges* | à chaque changement des données du composant |  | | | |  | | | | | ATTENTION : méthode à éviter car appelée trop souvent (avant même ngOnInit()) ce qui peut provoquer des diminutions de performances | | | | |
| ngOnInit | lorsque le composant est initialisé | import { Component } from '@angular/core';  @Component({  selector: 'monComposant',  templateUrl: './adresseFichier.html',  styleUrls: ['./adresseFichier.css']  })  export class MonComposant {  mesObjets:Array<MonObjet> = [ ];  constructor() {}  ngOnInit() {  this.mesObjets = [  new MonObjet('Mon objet 1'),  new MonObjet('Mon objet 1')  ];  }  } | | | |  | | | | | Rq: à cet instant, les données issues des parents transmises par @Input() sont bien chargées => appels services back à faire dans ngOnInit() | | | | |
| ngDoCheck |  |  | | | |  | | | | |  | | | | |
| ngAfterContentInit | lorsque les éléments projetés sont initialisés |  | | | |  | | | | |  | | | | |
| ngAfterContentChecked | lorsque les éléments projetés ont été vérifiés |  | | | |  | | | | |  | | | | |
| ngAfterViewInit | lorsque les éléments enfants ont été initialisés |  | | | |  | | | | |  | | | | |
| ngAfterViewChecked | lorsque les éléments enfants ont été vérifiés |  | | | |  | | | | |  | | | | |
| ngOnDestroy | lorsque le composant est détruit |  | | | |  | | | | | Rq: très utile pour se désabonner des observables utilisés dans le composant ! | | | | |
|  |  |  | | | |  | | | | |  | | | | |
| RxJS | Programmation réactive | traitement asynchrone des données | | | | voir => https://rxjs-dev.firebaseapp.com/api | | | | |  | | | | |
| Promise | objet qui représente une valeur qui peut être disponible maintenant ou plus tard, ou jamais. |  | | | | Rq: on y attache des callback qui sont appelés en fonction du succès ou de l'échec de la réponse   |  |  | | --- | --- | | Etats des promises :  - pending = en attente  - fulfilled = tenue, succés  - rejected = rompue, erreur  - settled = acquitée, traitement terminé | 3 chemins possibles :   * pending * pending -> fulfilled -> settled * pendig -> rejected -> settled | | | | | | | | | | |
|  | exemple général de déclaration d'une Promise | const maPromise = new Promise((resolve, reject) => {  if(true) {  resolve(monObjetResult);  } else {  reject();  }  }); | | | | | | *le timeout permet de simuler l'attente des données :*  const maPromise = new Promise((resolve, reject) => {  setTimout(  () => { resolve(monObjetResult);},  5000);  }); | | | |  | | | |
| .then( )  .catch( ) | abonnement à une promise | maPromise  .then((monObjetResult) => { *instruction(s) en cas de succés*; }) *=> définit la fonction* ***resolve***  .catch(() => { *instruction(s) en cas d'échec;* }); *=> définit la fonction* ***reject*** | | | | | | | | |  | | | | |
| Promise.all([ ])  async / await | promises chainées  si l'une des promise de la liste est rejected, alors on passe dans .catch | *promises indépendantes :*  Promise.all([  faireQqc(),  faireAutreChose(),  faireUnTroisiemeTruc()  ])  .then(()=>{})  .catch(e=>{}); | | *promises dépendantes du résultat de la précédente :*  async function toto() {  try {  let result = await faireQqc();  let newResult = await faireQqcAutre(result);  let finalResult = await faireUnTroisiemeTruc(newResult);  return finalResult;  } catch(error) {  failureCallback(error);  }  }  toto().then(finalResult => console.log(finalResult)); | | | | | | | => récupération du résultat à l'extérieur de la fonction async | | | | |
| async / await | permet aussi le lancement de plusieurs callback en même temps et retourne la réponse quand les résultats sont tous récupérés |  | | async function add(x) {  var a = resolveAfter2Seconds(20);  var b = resolveAfter5Seconds(30);  return x + await a + await b;  }  add(10).then(value => {console.log(value);}); | | | | | | | => affiche 60 après 5 secondes. | | | | |
|  |  |  | | | |  | | | | |  | | | | |
| Observable | objet permettant d'être informé des changements de valeur d'une variable dans le temps |  | | | |  | | | | | ATTENTION : il faut toujours prévoir de se désabonner d'une observable pour ne pas que le code se joue à l'infini (problèmes de mémoire et de performance !) | | | | |
|  | exemple général de déclaration d'une observable | const MonObservable$ = new Observable(suscriber => {  suscriber.next(1);  suscriber.next(2);  setTimout(() => {  suscriber.next(3);  suscriber.complite();  }, 1000);  } | | | | *Observable incrémentant une valeur toute les 2 s :*  const counter = Observable.interval(2000); | | | | | Rq: par convention on suffixe une variable observable avec $  Rq: il est important d'appeler la méthode .complete() lorsque la totalité des données a été envoyée, en particulier si ces données sont contrôlées par un Resolver (voir partie *Router*). Sans cela, le Resolver ne valide jamais la redirection ! | | | | |
| .subscribe( ) | s'abonne aux éléments retournés par l'observable et leur applique un traitement | MonObservable$.subscribe({  next(x) { console.log('got value ' + x); },  error(err) { console.error('something wrong occurred: ' + err); },  complete() { console.log('done'); }  }); | | | | MonObservable$.subscribe(  (value) => { console.log('Value = ' + value); },  (error) => { console.error('something wrong occurred: ' + err); },  () => { console.log('done'); }  ); | | | | | ATTENTION : lorsque le traitement est terminé**, il faut se désinscrire** pour ne pas avoir de fuite de mémoire !  ***=> voir exemple ligne suivante !*** | | | | |
| Subscription  .subscribe( )  .unsubscribe() | objet stockant la souscription à une observable afin de pouvoir se désinscrire | **Syntaxe complète à privilégier !!!**  export class MonComponent implements OnInit, OnDestroy {  maSubscription: Subscription;  ngOnInit() {  this.maSubscription = MonObservable$.subscribe(  (value) => { console.log('Value = ' + value); },  (error) => { console.error('something wrong occurred: ' + err); },  () => { console.log('done'); }  );  }  ngOnDestroy() {  this.maSubscription.unsubscribe();  }  } | | | | | | | | |  | | | | |
| .pipe( ) | ajoute un ou plusieurs traitements successifs sur les données retournées par l'observable ou la promise **avant de les utiliser** | MonObservable$  .pipe(**filter(**elt => elt !== null**)**)  .pipe(**map(**elt => elt.sousElt**)**)  .subscribe(  (value) => { console.log('Value = ' + value); },  (error) => { console.error('something wrong occurred: ' + err); },  () => { console.log('done'); }  **)**; | | | | *exemple d'opérateurs :*   |  |  | | --- | --- | | filter( ) | *filtrer* | | map( ) | *transformer* | | throttleTime( ) | *impose un délai min entre 2 valeurs* | | reduce( ) | *rassemble toutes les valeurs*  *(ne retourne la valeur finale)* | | scan( ) | *rassemble toutes les valeurs*  *(retourne chaque étape du calcul)* | | takeUntil( ) | *définit un timeout pour l’observable* | | | | | | Rq: il existe de nombreux opérateurs de pipes. Voir site RxJS | | | | |
| forkJoin( ) | opérateur créant une observable qui renverra en une seule fois les données issues de plusieurs observables sous forme de Map  (ex : utile si traitement nécessite le retour de plusieurs API) | const joinedObservable = forkJoin({  firstResult: monObservable1.subscribe(),  secondResult: monObservable2.subscribe()  }); | | | |  | | | | | Rq: il existe de nombreux opérateurs de création d’observables. Voir site RxJS | | | | |
|  |  |  | | | |  | | | | |  | | | | |
| Subject | objets gérant la synchronicité des données entre les composants au sein même de l'appli |  | | | |  | | | | |  | | | | |
|  | au lieu d'émettre l'objet lui-même, le service expose un objet Subject équivalent à une Observable qui contient cet objet et les composants qui le consomment s'abonnent à ce Subject grâce à un objet Subscription  (en n'oubliant pas   * d'émettre à chaque changement de la donnée dans le service * de se désabonner à la destruction de chaque composant consommateur) | export class **MonService** {  **private** maData: any;  maDataSubject = new Subject<any>();  emitMaDataSubject() {  this.maDataSubject.next(this.maData);  }  traitementSurMaData() {  instruction(s);  this.emitMaDataSubject();  }  autreTraitementSurMaData() {  instruction(s);  this.emitMaDataSubject();  }  } | | | | export class **MonComponent** implement OnInit, OnDestroy {  maData: any;  maDataSubscription: Subscription;  ngOnInit() {  this.maDataSubscription = this.monService.maDataSubject.subscribe(  (maData: any) => {  this.maData = maData;  },  (error) => { traitement des erreurs; }  );  this.monService.emitMaDataSubject();  }  ngOnDestroy() {  this.maDataSubscription.unsuscribe();  }  } | | | | | | | Rq: cela introduit un niveau d'abstraction intermédiaire et complexifie le code, mais il s'agit d'une bonne pratique qui limite les problèmes de synchronisation des données et les éventuels bugs qui pourraient y être lié en particulier lorsque l'appli devient complexe | | |
|  |  | *Autre stratégies permettant de unsuscribe toutes les subscriptions en une seule étape*  *et sans les délcarer en paramétres :*  export class **MonComponent** implement OnInit, OnDestroy {  maData: any;  **isDestroyed:** Subject<void> = new Subject();  ngOnInit() {  this.maDataSubscription = this.monService.maDataSubject  .pipe(takeUntil(this.isDestroyed))  .subscribe(  (maData: any) => {  this.maData = maData;  },  (error) => { traitement des erreurs; }  );  this.monService.emitMaDataSubject();  }  ngOnDestroy() {  this.isDestroyed.next() ; *//émet pour clore toutes les souscriptions*  this.isDestroyed.complet() ; *//clos le subject*  }  } | | | | | | | | | | |  | | |
| BehaviorSubject | subject avec une valeur initiale par défaut | private maDataSubject: BehaviorSubject<MaData> = new BehaviorSubject<MaData>(this.maData); | | | |  | | | | |  | | | | |
|  |  |  | | | |  | | | | |  | | | | |
| Formulaires |  |  | | | |  | | | | |  | | | | |
| FormsModule | config dans ***app.modules.ts*** | imports: [  FormsModule,  ...  ], | | | |  | | | | |  | | | | |
| Coté template | géré côté HTML |  | | | |  | | | | |  | | | | |
|  | récupération des données lors du clic sur le bouton submit | <form (ngSubmit)="handleSubmit(userForm.value)" #userForm="ngForm">  <input type="text" name="email" ngModel />  <input type="submit" value="Valider" />  </form> | | | | | | | | |  | | | | |
|  | récupération des données dynamiquement en même temps qu'elles sont entrées par l'utilisateur  (lors du clic sur submit, les données sont déjà à jour dans l'objet user coté TS) | <form (ngSubmit)="handleSubmit()" #userForm="ngForm">  <input type="text" name="email" [(ngModel)]="user.email" require #emailModel="ngModel"/>  <div \*ngIf="emailModel.dirty && emailModel.hasError('email')">  Le champ email n'est pas valide  </div>  <div \*ngIf="emailModel.dirty && emailModel.hasError('required')">  Le champ email est requis  </div>  <input type="submit" value="Valider" [disabled]="userFormModel.invalid" />  </form> | | | | | | | | |  | | | | |
| Coté code | géré côté TypeScript |  | | | |  | | | | |  | | | | |
| ReactiveFormsModule | config dans ***app.modules.ts*** | imports: [  FormsModule,  ReactiveFormsModule  ...  ], | | | |  | | | | |  | | | | |
| FromControl | fournit des méthodes permettant de controler la valeur du champ | <form (ngSubmit)="handleSubmit()" [formGroup]="userForm">  <div>  <label>Email:</label> *↓ :* *voir Style*  <input type="text" formControlName="email" class="input-angular" />  <div \*ngIf="emailCtrl.dirty && emailCtrl.hasError('email')">  Le champ email est invalide !  </div>  <div \*ngIf="emailCtrl.dirty && emailCtrl.hasError('required')">  Le champ email est requis !  </div>  </div>  <input type="submit" value="Login" [disabled]="userForm.invalid"/>  </form>  @...  export class CodeFormComponent {  emailCtrl: FormControl;  passwordCtrl: FormControl;  userForm: FormGroup;  constructor(fb: FormBuilder, private router: Router, …) {  *//Création des controles*  this.emailCtrl = fb.control('', [Validators.email, Validators.required]);  this.passwordCtrl= fb.control('', [Validators.required, isPrefixValidator]); *voir validator personnalisé : ↑*  *//Création du groupe*  this.userForm = fb.group({  email: this.emailCtrl,  password: this.passwordCtrl  });  }  handleClear() {  this.emailCtrl.setValue('');  this.passwordCtrl.setValue('');  }  handleSubmit() {  const newUser = new User();  newUser.email = this.userForm.value['email'];  newUser.password = this.userForm.value['password'];  this.monUserService.addNewUser(newUser);  this.router.navigate(['/users]);  }  } | | | | | | | |  | Rq: pour des formulaires dynamique avec ajout de champs ou de groupes de champs à la demande de l'utilisateur   * voir tableau "*Compléments pour les Formulaires complexes"* en fin de document. | | | | |
| FormGroup | = ensemble de **FormControl**  mêmes méthodes mais valables pour le groupe de champs entier | |  |  | | --- | --- | | monFormControl | | | **.value** | la valeur du champ. | | .valueChanges | un Observable qui émet à chaque modification du champ | |  |  | | .valid | = true si champ valide | | .invalid | = true si champ n'est pas valide | |  |  | | .errors | récupération des erreurs du champ | | **.hasError** | fonction qui permet de connaitre si le champ a une erreur donnée. Prend le nom de l’erreur en paramètre | |  |  | | .dirty | = false jusqu’à ce que l’utilisateur modifie la valeur du champ | | .pristine | = true jusqu’à ce que l’utilisateur modifie la valeur du champ | |  |  | | .touched | = false tant que l’utilisateur n’a pas pris le focus sur le champ | | .untouche | = true tant que l’utilisateur n’a pas pris le focus sur le champ | | | | | | |
| Validator | permet de valider un champ  utilisable dans les 2 techniques (template et code) | Validators  .required  .minLength(nb)  .maxLength(nb)  .email()  .pattern(maRegEx)  .min(nb)  .max(nb) |  | | | | |
| Validator personnalisé | validator personnalisé | import { FormControl } from '@angular/forms';  import \_ from 'lodash';  export const PREFIX: string = 'nk';  export const PREFIX\_SEPARATOR: string = '-';  /\*\*  \* La valeur du champ commence-t-elle par nk-  \* @param{FormControl} control  \* @returns {{isPrefix: boolean}}  \*/  export function isPrefixValidator(control: FormControl) {  const prefix = \_.split(control.value, PREFIX\_SEPARATOR, 1);  console.log('Préfixe :', prefix[0]);  return (\_.isArray(prefix) && PREFIX === prefix[0]) ? null : { isPrefix: true}; *//l'erreur levée sera "isPrefix"*  } | | | | | | | | |  | | | | |
|  | validator personnalisé avec paramètre | import { FormControl } from '@angular/forms';  import \_ from 'lodash';  export const PREFIX\_SEPARATOR: string = '-';  /\*\*  \* La valeur du champ commence-t-elle par le préfixe attendu  \* @param{expectedPrefix} prefixe attendu  \* @param{FormControl} control  \* @returns {{isPrefix: boolean}}  \*/  export function parametrableValidator(expectedPrefix: string) {  return function (control: FormControl) {  const prefix = \_.split(control.value, PREFIX\_SEPARATOR, 1);  return (\_.isArray(prefix) && prefix[0] === expectedPrefix) ? null : { isPrefix: true}; *//l'erreur levée sera "isPrefix"*  }  } | | | | | | | | | Rq: pour un créer un validateur personnalisé dont le paramètre dépend d'une donnée variable ou de la valeur d'un autre controleur   * voir tableau "*Compléments pour les Formulaires complexes"* en fin de document. | | | | |
| Style | changer le style d'un champ selon sa validité  FormControl ajoute une classe sur le champ à chaque changement de validité | *sur l'élément HTML :*  class="input-angular">  *code CSS associé :*  .input-angular {  background-color: rgba(205, 208, 214, 0.56);  color: #3e3c3c;  }  */\* sans espace = si possède les 3 classes \*/*  .input-angular.ng-invalid.ng-dirty {  border: 2px red solid;  border-radius: 2px;  }  .input-angular.ng-valid {  border: 2px green solid;  border-radius: 2px;  } | | | | *selon l'état, ajout de la classe :*  .ng-valid  .ng-invalid  .ng-dirty  .ng-pristine  .ng-touched  .ng-untouched | | | | |  | | | | |
| evt clavier |  |  | | | |  | | | | |  | | | | |
| keyup | capture les évenement clavier dans un formulaire | *HTML*  <mon-formulaire #monFormulaire (keyup)="controlerActionClavier($event)">  *//sensible uniquement à la touche "Entrée" sans //avoir à filtrer dans la méthode*  <mon-formulaire #monFormulaire (keyup.enter)="validerFormulaire()"> | | | | *TypeScript*  controlerActionClavier(event) {  *//si action sur la touche "Entrée"*  if(event.keyCode === 13) {  *//traitement*  }  } | | | | |  | | | | |
| controles sur les formGroup |  |  | | | |  | | | | |  | | | | |
| .disable()  .enable() | sur un FormGroup  rend tous les champs d'un formulaire innaccessible à l'utilisateur  ATTENTION : rends impossible la récupération de ses données !  sur un FormControl  rend **le champ du formulaire innaccessible** à l'utilisateur  ATTENTION : la valeur reste disponible pour le traitement mais pas le controle de sa validité | */\*\**  *\* formGroupDisabled*  *\* Méthode utilitaire permettant de passer tous les formControl d'un formGroup en disabled ou en enabled*  *\* Remarque : evite d'utiliser formGroup.disable() qui rend les valeurs des formControl inaccésibles*  *\**  *\* @param disabled*  *\* @param formGroups*  *\*/*  *export function formGroupDisabled(disabled: boolean, ...formGroups: Array<FormGroup>) {*  *formGroups.filter(formGroup => !isNullOrUndefined(formGroup)).forEach(formGroup => {*  *if (disabled) {*  *Object.keys(formGroup.controls).forEach(ctrl => formGroup.get(ctrl).disable());*  *} else {*  *Object.keys(formGroup.controls).forEach(ctrl => formGroup.get(ctrl).enable());*  *}*  *});*  *}* | | | | | | | | |  | | | | |
|  |  |  | | | |  | | | | |  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Compléments pour les Formulaires complexes | |  |  |  |
| Validator  dépendant d'une valeur variable | lier le paramètre d'un validator personnalisé avec la valeur d'une variable : monChampCtrl1  lier le paramètre d'un validator personnalisé avec la valeur d'un autre champ contrôlé par un autre validateur : monChampCtrl2 | | | Rq: techniques obsolètes ! A corriger pour ne pas utiliser ngOnChange() et détruire les souscriptions dans ngOnDestroy() ! |
|  | public maVariable: any;  public monChampCtrl1: FormControl;  public monChampCtrl2: FormControl;  constructor(fb: FormBuilder) {  this.monChampCtrl1 = fb.control('valeur initiale', [monValidator(this.maVariable)]);  this.monChampCtrl2 = fb.control('valeur initiale', [monValidator(this.monChampCtrl1.value)]);  this.monChampCtrl1.valueChanges  .pipe(takeUntil(this.unsubscribe))  .subscribe(value => {  this.monChampCtrl2.setValidators([distributionValidator(this.monChampCtrl1.value)]);  this.monChampCtrl2.updateValueAndValidity();  });  }  ngOnInit() {  this.monChampCtrl1.setValue('valeur initiale');  this.monChampCtrl1.setValidator([monValidator(this.maVariable)]);  }  ngOnChanges() {  this.monChampCtrl1.setValidator([monValidator(this.maVariable)]);  this.monChampCtrl1.updateValueAndValidity();  } | |  |
| FormArray | ajout dynamique de champs avec contrôle | <form (ngSubmit)="handleSubmit()" [formGroup]="userForm">  <div formArrayName="hobbies">  <h3>Entrer vos hobbies :</h3>  <div class="form-group" \*ngFor="let hobbyCtrl of **hobbiesFArray**.conctrols; index as i">  <input type="text" class="form-control" [formControlName]="i" />  </div>  <button type="button" (click)="**onAddHobby()**">Ajouter un hobby</button>  …  </form>  ----------------------------------------------  @...  export class CodeFormComponent {  pseudoCtrl: FormControl;  **hobbiesFArray**: FormArray;  userForm: FormGroup;  constructor(private fb: FormBuilder, private router: Router, …) {  *//Création des controles*  this.pseudoCtrl = fb.control('', [Validators.required]);  *//Initialisation des tableaux* |*//ou avec des valeurs initiales :*  this.**hobbiesFArray** =fb.array([ ]); |fb.array([this.fb.control(maValeur, [Validators.required]), …])    *//Création du groupe*  this.userForm = fb.group({  pseudo: this.pseudoCtrl,  **hobbies**: this.**hobbiesFArray**  });  }  **onAddHobby() {**  **const newHobbyCtrl = this.fb.control(**''**, [Validators.required]);**  **this.hobbiesFArray.push(newHobbyCtrl);**  **}**  handleSubmit() {  const newUser = new User();  newUser.pseudo = this.userForm.value['pseudo'];  newUser.hobbies = this.userForm.value['**hobbies**'] ? this.userForm.value['hobbies'] : [ ];  this.monUserService.addNewUser(newUser);  this.router.navigate(['/users]);  }  } | |  |
| FormGroup  FormArray | association des FormGroup et FormArray pour ajouter des groupes de N champs de manière dynamique | <form [formGroup]="**ingredientsForm**">  <div formArrayName="**ingredients**">  <table>  <tr \*ngFor="let ingredient of **ingredientsFormArray**.controls; index as i">  <div [formGroup]="**ingredientsFormArray.controls**[i]">  <td>  <input type="text" class="form-control" formControlName="**nom**" />  </td>  <td>  <input type="text" class="form-control" formControlName="**quantite**" />  </td>  <td>  <input type="text" class="form-control" formControlName="**unite**" />  </td>  </div>  </tr>  <tr>  <td colspan="3">  <button type="button" (click)="onAddIngredient()">Ajouter un ingrédient</button>  </td>  </tr>  </table>  </div>  </form>  ----------------------------------------------  @Component({…})  export class IngredientsFormulaireComponent implements OnInit {  @Input()  ingredients: Array<any>;  **ingredientsFormArray**: FormArray;  **ingredientsForm**: FormGroup;  constructor(private fb: FormBuilder) {  this.**ingredientsFormArray** = fb.array(this.initIngregientsFormGroupList(this.ingredients));  *// ^ initialisation ici avec un* Array<FormGroup>  this.**ingredientsForm** = fb.group({  **ingredients**: this.**ingredientsFormArray**  })  }  initIngregientsFormGroupList(ingredients: Array<any>): Array<FormGroup> {  const ingredientsFormGroupList = [];  for (let ingredient of ingredients) {  const nomCtrl = this.fb.control(ingredient.nom, []);  const quantiteCtrl = this.fb.control(ingredient.quantite, []);  const uniteCtrl = this.fb.control(ingredient.unite, []);  const ingredientForm = this.fb.group({  **nom**: nomCtrl,  **quantite**: quantiteCtrl,  **unite**: uniteCtrl  });  ingredientsFormGroupList.push(ingredientForm);  }  return ingredientsFormGroupList;  }  ngOnInit(): void {  }  onAddIngredient() {  const nomCtrl = this.fb.control('', []);  const quantitesCtrl = this.fb.control('', []);  const unitesCtrl = this.fb.control('', []);  const ingredientForm = this.fb.group({  **nom**: nomCtrl,  **quantite**: quantitesCtrl,  **unite**: unitesCtrl  });  this.**ingredientsFormArray**.push(ingredientForm);  }  getIngredients() {  return this.**ingredientsForm**.value;  }  } | | formGroup principal  formArray contenant les sous-groupes  récupération de chaque sous-groupe (FormGroup) grace à l'index  utilisation du sous-groupe (FormGroup) de manière classique en faisant appel aux controleurs qu'il contient  seul le FormGroup principal et le FormArray qu'il va contenir sont définit en tant que variables.  Les sous-groupes et les controleurs seront définis de manière dynamique  ici le FormArray est initialisé avec un Array<FormGroup> et non un Array<FormControler> comme ci-dessus !  initialisation facultative de la liste des ingrédients à partir de données transmissent au composant  le sous-groupe "ingredientForm" contient 3 controleurs => 3 champs par ligne dans le tableau des ingrédients  ajout dynamique d'un nouvel ingrédient (cad nouvelle ligne de 3 champs) à la demande de l'utilisateur  méthode permettant la récupération des données brutes depuis un composant parent via un ViewChild() par exemple *(méthode pouvant être complétée ou aditionnées de méthodes testant la validité du FormGroup principal ou d'un prétraitement sur les valeurs avant de les retourner)* |
|  |  |  |  |  |