Principes et Commandes NodeJS - Angular

NodeJS

Principe :

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

Instalation de Maven :

Télécharger Maven sur http://maven.apache.org

L'installer à la racine du lecteur C:

Créer une variable d'environnement JAVA\_HOME - adresse du JDK (jdk.1.8...)

Mettre à jour la variable PATH avec l'adresse du repertoire bin de Maven (permet d'utiliser mvn en ligne de commande)

tester avec la commande : mvn --version

Nouveau projet Maven :

Lors de la génération d'un projet Maven, il faut définir obligatoirement :

-groupId: Dev => identifiant de l'organisme qui développe le projet

-artifactID: nomduprojet => nom du projet

-version: 1.0SNAPSHOT => version du projet

-package: dev => nom du fichier lors du packaging

Contenu du projet :

Fichier POM

gère les dépendances

/src/main/

code source

/src/test/

test unitaire JUnit

/src/main/resources

contiens les fichier .properties => fichier contenant les propriétés et leurs valeurs que l'on pourra utiliser dans le code Java

contiens les autres fichiers resources

/target

code compilé (.class et .jar)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Symbole | Signification | Code | Utilisation | |
| NodeJS |  | | | |
| en ligne de commande |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| 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 |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

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

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 |  | | | | | |
|  | ajout de la dépendance de JUnit dans le POM  Rq : JUnit est ajouté par défaut à la dépendance d'un projet Maven généré par Spring. | <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  </dependency>  </dependencies> | |  | |  |
|  | conventions de nommages ! | *dans src/main/*  public class MaClasse {  public String maMethodeA(String monParam1) {  instruction(s);  }    public void maMethodeB() {  instruction(s);  } | | *dans src/test/*  public class MaClasseTest {  @Test  public void testMaMethodeA() {  instruction(s) pour le test;  }  @Test  public void testMaMethodeB() {  instuction(s) pour le test;  }  @Test  public void testMaMethodeBAvecNull() {  instruction(s) pour le test;  }  } | |  |
|  |  |  | |  | |  |
| Angular-cli |  |  | |  | |  |
| @angular/cli | instalation d'Angular-cli en global | npm install -g @angular/cli | |  | |  |
| ng new | créer un nouveau projet Angular | ng new monProjet | |  | |  |
| ng serve | démare 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 |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
| Affichage |  |  | |  | |  |
| {{ }} | 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';  constructor() {}  } | | <p>  {{maProp1}}  </p>  *ou*  <p [textContent]="name"></p> | | ATTENTION : le constructeur doit être vide car il est généré automatiquement par Angular |
| [ ] | binding de propriété | @…  export class MonComposant {  fontWeight:string = 'bold';  color:string = 'green';  … | | <p [style.color]="'red'">Mon texte en rouge</p>  *ou*  <p [ngStyle]="{'color': 'red'}">Mon texte en rouge</p>  <p [ngStyle]="{'fontWeight': fontWeight, 'color': red}">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 }">Text mis en forme</p>  *fichier CSS :*  .txtRed {  color: red;  }  .backgroundBlue {  background-color: blue;  } | |  |
|  | création d'une nouvelle balise | import { Component } from '@angular/core';  export interface hasName {  name:string;  }  @Component({  selector: 'monComposant',  templateUrl: './adresseFichier.html',  styleUrls: ['./adresseFichier.css']  })  export class MonComposant {  getUser():hasName {  return {name:'Nom';};  }    constructor() {}  } | | <monComposant [user]="getUser()"></monComposant> | |  |
| [( )] | 2 way data binding  lie les valeurs du composant dans les 2 sens :   * TS -> HTML * HTML -> TS | @...  export class MonComposant {  maProp1:string = 'maValeurInitiale';  … | | <input [(ngModel)]="maProp1" placeholder="Ma valeur initiale" /> | | Rq: lorsque l'utilisateur remplira le champ, la valeur sera immédiatement mise à jour dans le TS. |
| ( ) | binding d’événement  $event obligatoire pour déclarer l'évenement | @...  export class MonComposant {  onClick(event):void {  console.log('onClick', event);  }  … | | <div>  <button (click)="onClick($event)">Click me</button>  </div> | |  |
| # | déclaration de variable |  | <input type="text" #monInput placeholder="Saisir une valeur">  <div>La valeur de l'input est : {{monInput.value}}</div> | | |  |
| \* | directives structurelles | @...  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>  <!-- for -->  <ul>  <li \*ngFor="let skill of skills ; index as i">  {{i}} - {{skill}}  </li>  </ul>  <!-- 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> | | 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 |
| ? | paramètre optionnel | @...  export class MonComposant {  maProp1: 'valeur1';  constructor() {}  } | | <p [textContent]="user?.maProp2"></p> | | lorsque maProp2 est undefined, <p></p> est créé mais n'a pas de contenu sans lever d'erreur |
| | | | pipe  sorte de filtre permettant la mise en forme d'un élément à afficher | |  |  | | --- | --- | | 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 (une chaine vide sinon) | | | <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> | |  |
|  |  |  | |  | |  |
| @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 }} |
| @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');  }  } | | | | *utilisation dans le fichier HTML :*  **<p** appHighlight**>**Mon texte décoré</p> |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
| Communication entre composants |  |  | |  | |  |
| @Input() | charge un objet transmis par le composant parent | import { Component, Input } from '@angular/core';  @Component({  selector: 'monComposantEnfant',  templateUrl: './adresseFichier.html',  styleUrls: ['./adresseFichier.css']  })  export class **MonComposantEnfant** {    @Input()  maPropEnfant:{maPropParent:string};  @Input()  formation:Formation;    constructor() {}  } | | *HTML du* ***composant parent*** *:*  <**monComposantEnfant** [maPropParent]="getMaPropParent()"></monComposantEnfant>  *Exemple concret avec une liste :*  <ul>  <li \*ngFor="let formation of formations">  <**formation-item** **[formation]=**"formation"></formation-item>  </li>  </ul> | | => transmission de l'objet "formation" au composant enfant "formation-item" chargé d'affiché les détails d'un objet de type Formation |
| @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()  **customEvent**:EventEmitter<MonObjet> = new EventEmitter<MonObjet>();    constructor() {}  onClick(event): void {  this.**customEvent**.emit(this.**monObjet**);  } | | *HTML du* ***composant enfant*** *:*  <button (click)="**customEvent**($event)" >Cliquez ici</button>  *HTML du* ***composant parent*** *:*  <**monComposantEnfant**  [maPropParent]="getMaPropParent()"  (customEvent)="handleCustomEvent($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> | |  |
|  |  |  | |  | |  |
| 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 alors sont des singletons |  | |  | |  |
| @Injectable() | déclaration du provider (= classe service) | @Injectable()  export default class MonProvider {  ...  } | |  | | ATTENTION : nécessite de déclarer le provider dans ***app.module.ts*** manuellement |
| provideIn: | déclaration du provider sans avoir besoin de le déclarer manuellement dans *app.module.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*** | 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 :***  user:User;  httpService.getUser('pseudo')  .subscribe((result:any) => {  this.user = result;  }, () => {  console.log('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);  }));  }  } | | | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
| Cycle de vie du composant |  |  | |  | |  |
| ngOnChanges |  |  | |  | |  |
| 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')  ];  }  } | |  | |  |
| ngDoCheck |  |  | |  | |  |
| ngAfterContentInit |  |  | |  | |  |
| ngAfterContentChecked |  |  | |  | |  |
| ngAfterViewInit |  |  | |  | |  |
| ngAfterViewChecked |  |  | |  | |  |
| ngOnDestroy | lorsque le composant est détruit |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |

XXXX

Principe :

Système :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Symbole | Signification | Code | Utilisation | |
|  |  | | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |