Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Centre Universitaire de Mila
Institut des Mathématiques et Informatique

Chapitre 2 : Les Framework de développement Web

Département Informatique
s.meghzili@centre-univ-mila.dz

Plan

- **Chapitre 2 : Les frameworks de développement Web**
  - *Modèle **MVC** (**M**odèle **V**ue **C**ontrôleur)*
  - *Framework **Angular***
  - *Framework **Spring Boot***
  - *Framework **Hibernate** (Gestion des bases de données)*
  - *Architecture **Microservices***

Qu'est-ce qu'un Framework Web ?

- **Un framework web est un framework logiciel conçu pour simplifier votre vie de développement web.**

- **Des cadres existent pour vous éviter d'avoir à réinventer la roue et aider à atténuer certains des frais généraux lorsque la construction d'un nouveau site.**

Framework du développement web

Le framework

≈

« cadre de travail » ou « cadre de développement »

≈

« architecture prête à l'emploi »

≈

« ensemble d'outils constituant les fondations d'une application »

≈

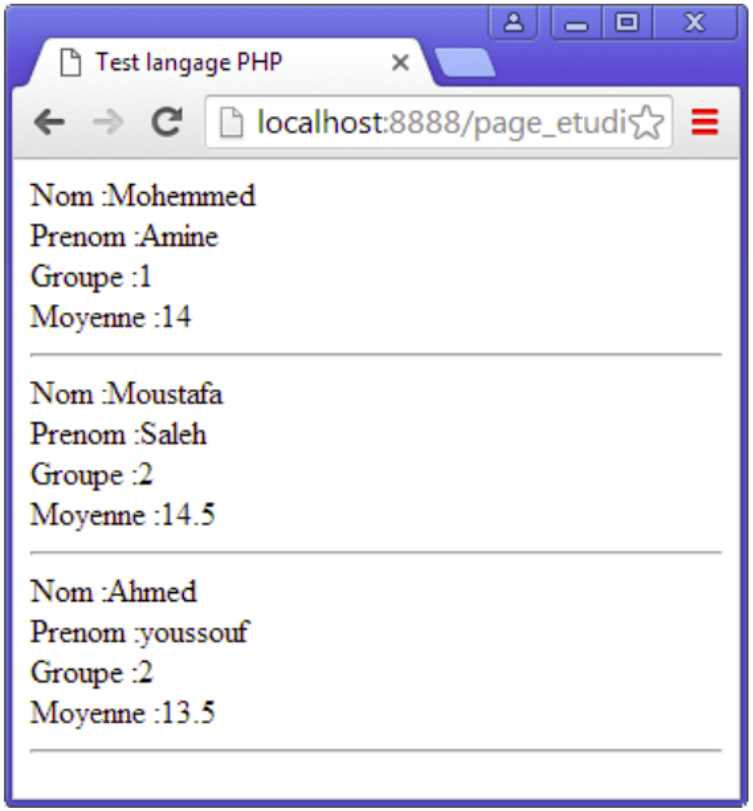
« c'est à la fois une sorte de boîte à outils et une méthodologie de travail »

Avantages des Framework

- *La rapidité de développement des projets grâce aux outilles disponibles*
- *Une très bonne organisation des projets*
- *Plusieurs frameworks sont basé sur l'architecture MVC, ils bénéficient donc de ses avantages*
- *Composants réutilisables*
- *Respecter les normes actuelles (sécurité , nouvelles technologies)*

Framework MVC : exemple introductif (1)

Pour commencer nous allons prendre un exemple d'un programme PHP qui permet d'afficher une liste d'étudiants a partir d'une base de données. Nous allons voir plusieurs version du programme et déterminer la version la plus efficace.



Framework MVC : exemple introductif (2)

```php
<?php
    $ma_connexion = mysql_connect ('localhost', 'root', '');   //Connexion au SGBD MySQL
    mysql_select_db ('universite', $ma_connexion) ; // selection de la base université
    //Préparer la requette SQL
    $sql = 'SELECT * FROM etudiants;';
    //Exécuter la requette SQL
    $resultat = mysql_query ($sql) or die ('Erreur SQL !'.$sql.'<br />'.mysql_error());

?>


<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
        <head>
            <title>Test langage PHP</title>
            <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
        </head>
<body>
    <?php
    //Percourir chaque tuple du résultat et l'afficher
    while($etud = mysql_fetch_row($resultat))
    {
    echo 'Nom :'.$etud['0'].'<br/>';
    echo 'Prenom :'.$etud['1'].'<br/>';
    echo 'Groupe :'.$etud['2'].'<br/>';
    echo 'Moyenne :'.$etud['3'].'<br/>';
    echo '<hr/>';
    }
    ?>
```

Version 1 : Un seul fichier

Framework MVC : exemple introductif (3)

```php
<?php
    $ma_connexion = mysql_connect ('localhost', 'root', '');   //Connexion au SGBD MySQL
    mysql_select_db ('universite', $ma_connexion) ; // selection de la base université
    //Préparer la requette SQL
    $sql = 'SELECT * FROM etudiants;';
    //Exécuter la requette SQL
    $resultat = mysql_query ($sql) or die ('Erreur SQL !'.$sql.'<br />'.mysql_error());
    mysql_close(); //fermer la connexion
    require ('vue1.php');
?>
```

« page_etudiants.php »

```php
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
        <head>
            <title>Test langage PHP</title>
            <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
        </head>
<body>
    <?php
    //Percourir chaque tuple du résultat et l'afficher
    while($etud = mysql_fetch_row($resultat))
    {
    echo 'Nom :'.$etud['0'].'<br/>';
    echo 'Prenom :'.$etud['1'].'<br/>';
    echo 'Groupe :'.$etud['2'].'<br/>';
    echo 'Moyenne :'.$etud['3'].'<br/>';
    echo '<hr/>';
    }
    ?>
</body>
</html>
```

« vue1.php »

Version 2 : deux fichiers

Framework MVC : exemple introductif (4)

```php
<?php
    function get_etudiants()
    {
    $ma_connexion = mysql_connect ('localhost', 'root', '');   //Connexion au SGBD MySQL
    mysql_select_db ('universite', $ma_connexion) ; // selection de la base université
    //Préparer la requette SQL
    $sql = 'SELECT * FROM etudiants;';
    //Exécuter la requette SQL
    $data = mysql_query ($sql) or die ('Erreur SQL !'.$sql.'<br />'.mysql_error());

    return $data;    // retourner le resultat extrait de la base de données
    }
?>
```

« modele_etudiants.php »
le modèle

Version 3 : trois fichiers

```php
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
        <head>
            <title>Test langage PHP</title>
            <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
        </head>
<body>
    <?php
    //Percourir chaque tuple du résultat et l'afficher
    while($etud = mysql_fetch_row($resultat))
    {
    echo 'Nom :'.$etud['0'].'<br/>';
    echo 'Prenom :'.$etud['1'].'<br/>';
    echo 'Groupe :'.$etud['2'].'<br/>';
    echo 'Moyenne :'.$etud['3'].'<br/>';
    echo '<hr/>';
    }
    ?>
</body>
</html>
```

« vue1.php »
la vue

```php
<?php
    require 'modele_etudiants.php';
    $resultat = get_etudiants();
    require 'vue1.php';
?>
```

« page_etudiants.php »
le contrôleur

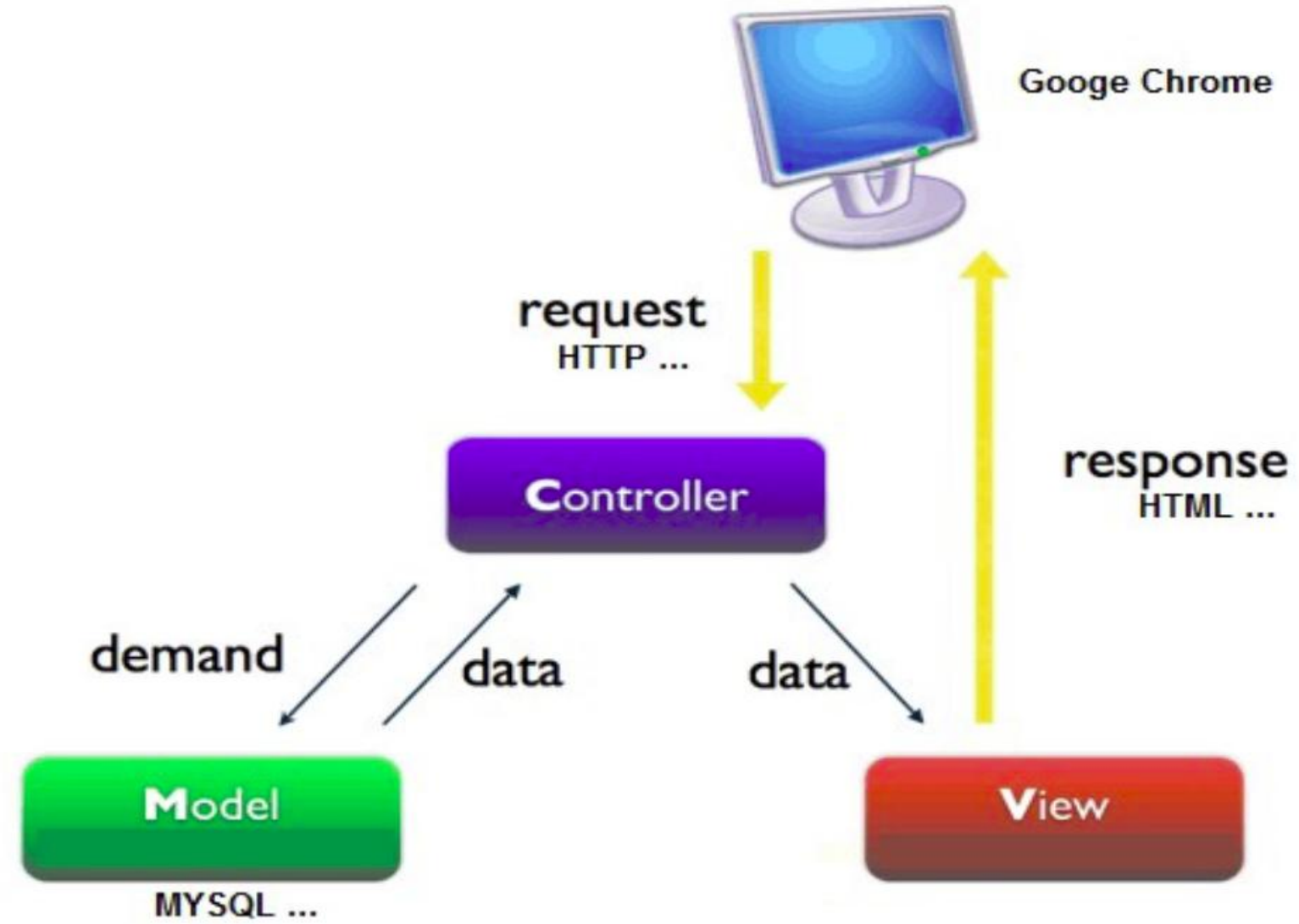
Framework MVC

Modèle : gestion des données de l'application

Vue : Affichage et interaction avec l'utilisateur

Contrôleur : gestion de la logique du code (il prend les décisions)

Modèle MVC



L'architecture MVC (Modèle-Vue-Contrôleur)

Avantages du MVC

- **Une conception claire et efficace**
- **Un gain de temps de maintenance et d'évolution du site (l'ajout et mise à jour des fonctionnalités est très facile)**
- **Une modularité qui offre une grande souplesse pour organiser le développement du site entre différents développeurs**
- **Une rapidité de développement**

Framework Angular

1. **Introduction**
2. **Installation et configuration**
3. **Data binding**
4. **Directives Angular**
5. **Exemple d'une application Angular:**
   - **Composants**
   - **Routes**
   - **Liens**
   - **Passage de paramètres**
   - **Exemple sur les opérations arithmétiques**
   - **Formulaires**
   - **Services**

Angular ?

- **Framework front end de Google permettant de créer des applications clientes web et mobile en HTML et en TypeScript**

- **Facilite la création de Single Page Application (SPA)**

- **Rends l'HTML dynamique**

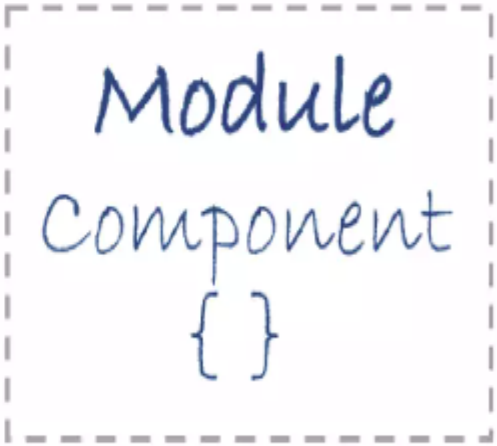
L'architecture d'Angular

Module

- C'est la base d'une application Angular
- Un module peut représenter le tout ou une partie de votre application

- Un module peut contenir les éléments suivants :
  - Component
  - Service
  - Directive
  - Pipe

- Un module peut être dépendant d'un ou plusieurs autre(s) module(s)
- Un module peut être partagé à d'autres modules



L'architecture d'Angular

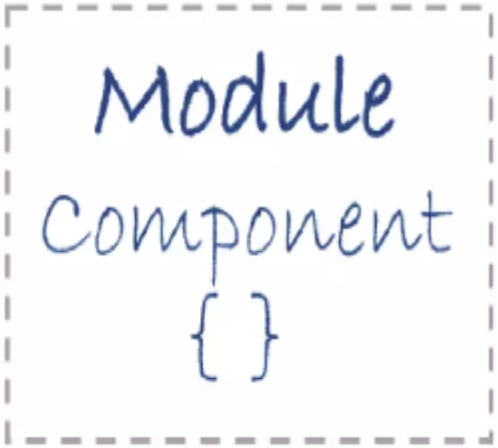
Component

- Dans Angular, tout est composant
- Un composant est une balise HTML personnalisée (ex: app-root, app-shop, app-login etc.)

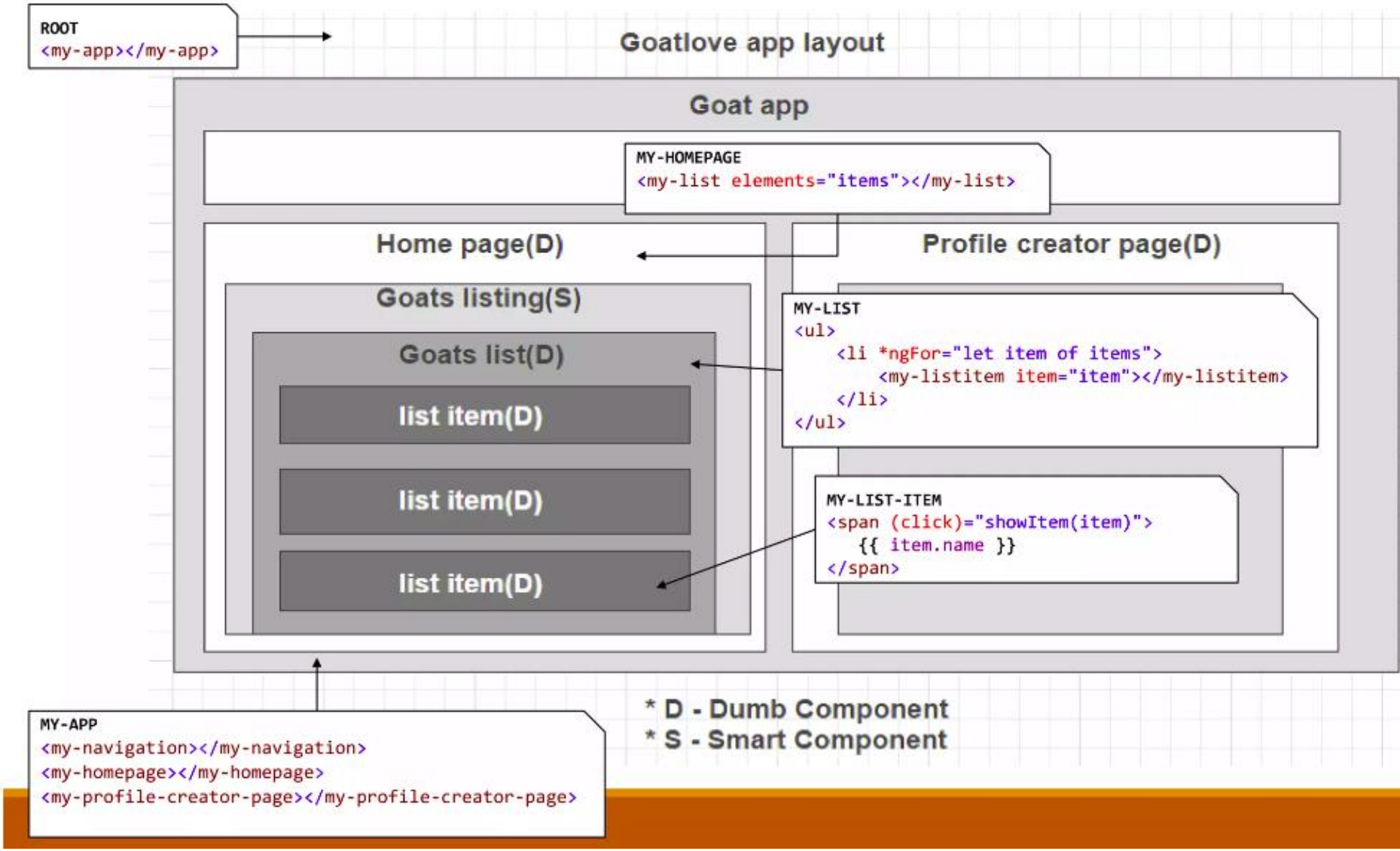
- Définit par :
  - 1 sélecteur (le nom de la balise)
  - 1 template
  - *1 ou plusieurs fichiers de style CSS (facultatif)*

- Représenté par :
  - 1 fichier Typescript (.ts)
  - 1 fichier HTML (.html)
  - *1 ou plusieurs fichiers de style CSS (facultatif)*

- Un component peut utiliser d'autres components



Angular: introduction



Outillage

```
npm install -g typescript
```

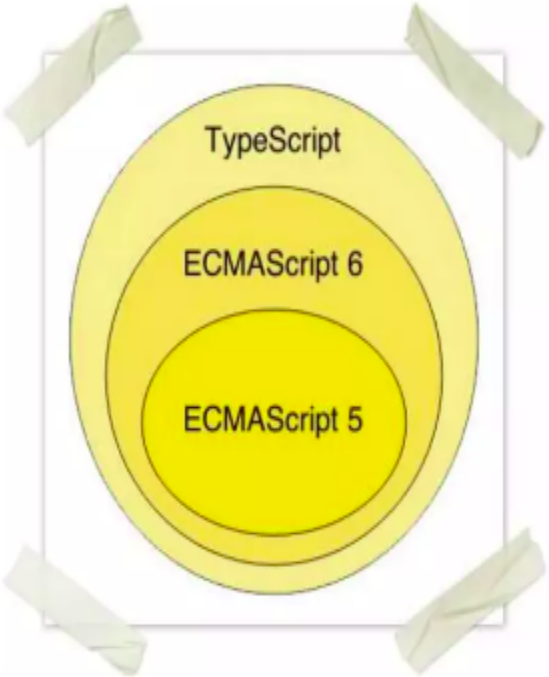
Typescript

Language de programmation libre et open-source développé par Microsoft

But initial : simplifier la production de code Javascript

Permet d'utiliser tous les concepts orienté objets
(classes, interfaces, héritages, public/private etc.)

Le code écrit en Typescript est transpilé en Javascript



Site officiel : https://www.typescriptlang.org

Qu'apporte TypeScript par rapport à Javascript : http://bit.ly/2rMQups

```
npm install -g @angular/cli
```

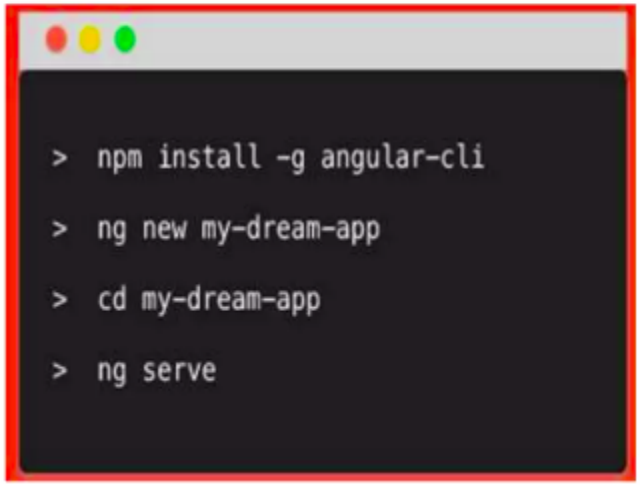
Outillage

Angular CLI

Outil en ligne de commande pour simplifier les tâches de développement avec Angular.

Fonctionnalités :

création d'un projet, génération de composants, exécution des tests, lancement du serveur, déploiement en production…



- **Installation et configuration:**

- Pour créer un nouveau projet nous devons bien évidemment disposer de certains éléments :

  - Node.js : La plateforme javascript
  - Angular CLI : L'outil fourni par Angular.
  - Visual Studio code : Un éditeur de code.

• Installation et configuration:

- **Etapes:**

1. **Installer le NodeJS en exécutant le fichier node-v16.14.2-x64.msi**
2. **Tester l'installation dans l'invite de commande, vous tapez : *npm -v***
3. **Installer l'éditeur du texte VS code**
4. **Installer Angular en utilisant la commande : npm install -g @angular/cli**
5. **Voir la version de Angular, vous tapez : *ng --v***
6. **Créer le premier projet Angular, par exemple : ng new *marchapp***
7. ***Exécuter ce projet en utilisant les commandes : cd / marchapp, ng serve***
8. **Ouvrir le navigateur puis vous tapez : *http://localhost:4200***
9. **Ouvrir ce projet en utilisant VSCode et explorer sa structure**

Angular : Configuration

- **.editorconfig** : configuration pour les éditeurs de code

- .**gitignore** : spécifie les fichiers intentionnellement non suivis que **Git** doit ignorer.
- **README.md** : documentation d'introduction pour l'application racine.

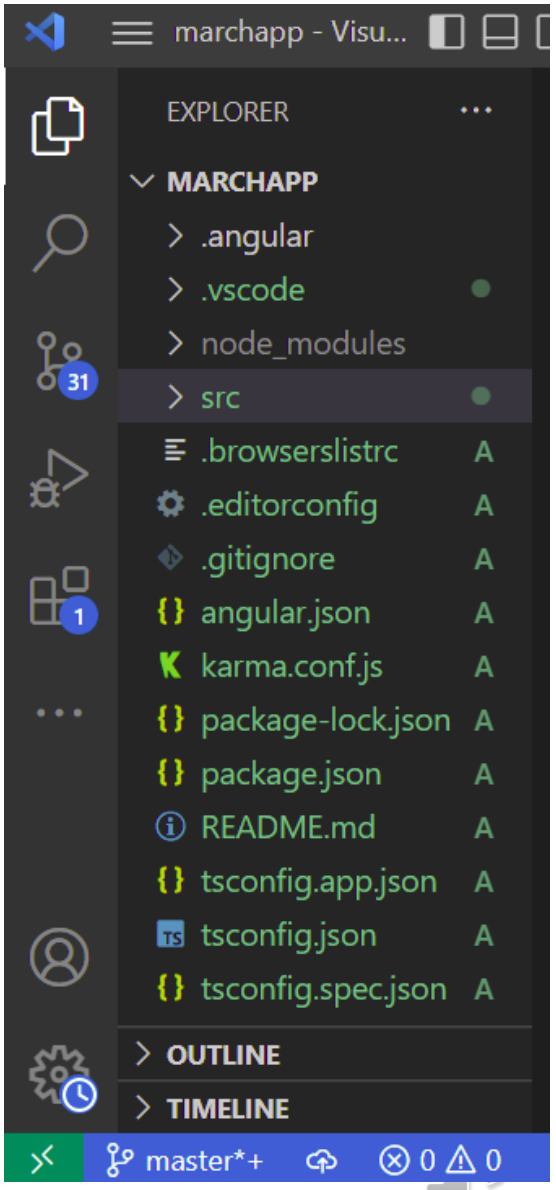
- **angular.json** : configuration par défaut de la **CLI** pour tous les projets de l'espace de travail, y compris les options de configuration pour les **outils** de génération, de service et de test utilisés par la CLI

- **package.json** : configure les dépendances de package ***npm*** qui sont disponibles pour tous les projets de l'espace de travail.

- **package-lock.json** : fournit des informations sur la version de tous les packages installés dans **node_modules** par le client npm.

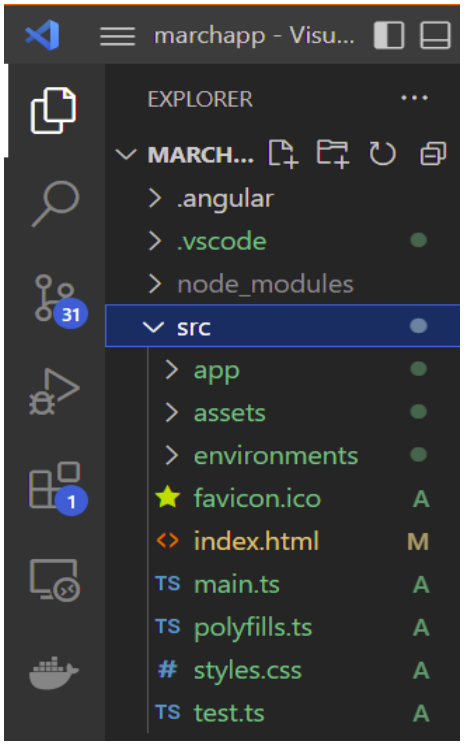
- **src/** : fichiers source pour le projet d'application de niveau racine.
- **node_modules/** : Fournit des packages **npm** à l'ensemble de l'espace de travail. Les dépendances **node_modules** à l'échelle de l'espace de travail sont visibles pour tous les projets.

- **tsconfig.json** : La configuration **TypeScript** de base pour les projets dans l'espace de travail. Tous les autres fichiers de configuration héritent de ce fichier de base.



Angular : dossier src/

- **app/** : contient les fichiers de **composants** dans lesquels la logique et les données de votre application sont définies.



- **assets/ :** Contient des **images** et d'autres fichiers d'actifs à copier tels quels lorsque vous créez votre application.
- **favicon.ico:** Une icône à utiliser pour cette application dans la barre de favoris.

- **index.html :** la page **HTML principale** qui est servie lorsqu'un internaute visite votre site. La CLI ajoute automatiquement tous les fichiers JavaScript et CSS lors de la création de votre application, vous n'avez donc généralement pas besoin d'ajouter manuellement des balises <script> ou <link> ici.

- **main.ts :** le point d'entrée principal de votre application. Compile l'application avec le compilateur JIT et démarre le module racine de l'application (**AppModule**) pour qu'il s'exécute dans le navigateur.

- **styles.css** : répertorie les fichiers CSS qui fournissent des styles pour un projet.

Angular : dossier /src/app/

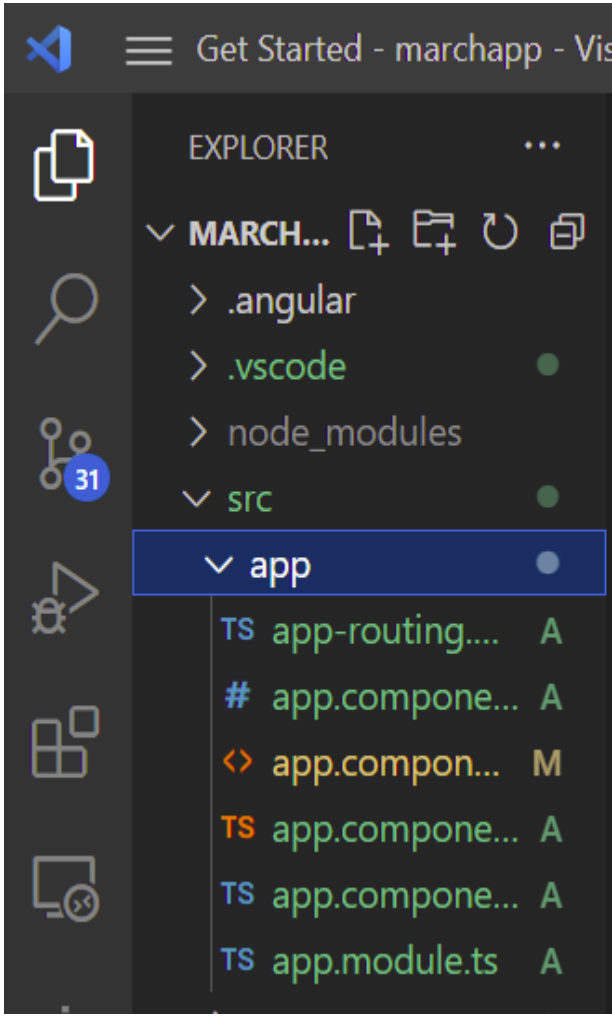
- ***app/app.component.ts*** : définit la logique du composant racine de l'application, nommé **AppComponent**. La vue associée à ce composant racine devient la racine de la hiérarchie des vues lorsque vous ajoutez des composants et des services à votre application.

- ***app/app.component.html*** : définit le modèle **HTML** associé à l'AppComponent racine.

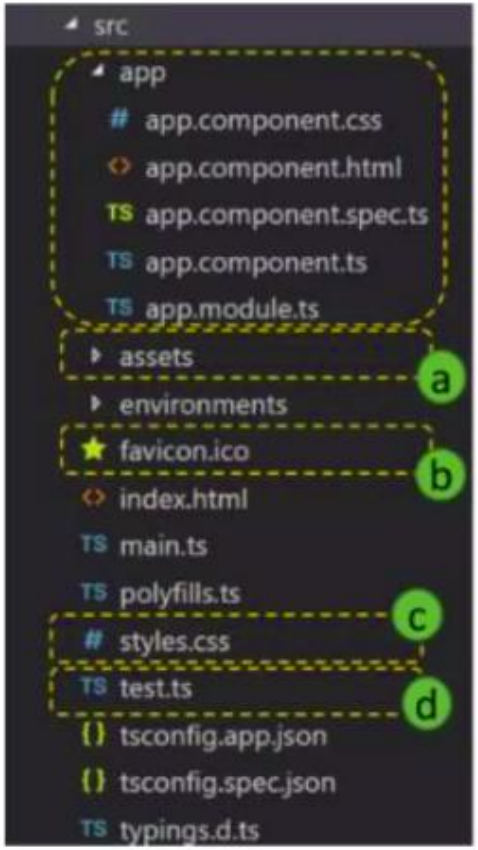
- ***app/app.component.css*** : définit la feuille de style **CSS** de base pour l'AppComponent racine.

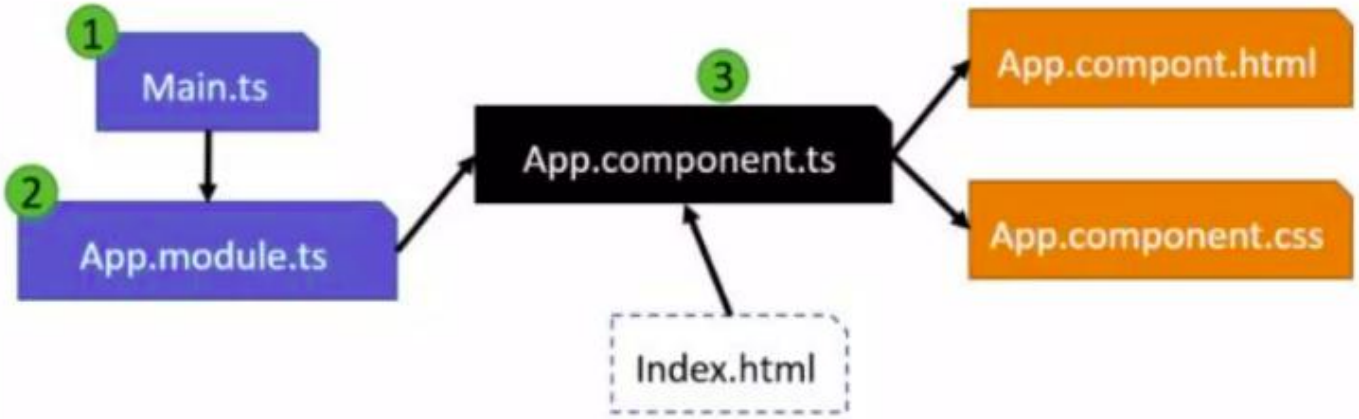
- ***app/app.component.spec.ts*** : définit un test unitaire pour l'AppComponent racine.

- ***app/app.module.ts*** : définit le module racine, nommé **AppModule**, qui indique à Angular comment assembler l'application. Déclare initialement uniquement AppComponent. Au fur et à mesure que vous ajoutez des composants à l'application, ils doivent **<u>être déclarés ici</u>**.

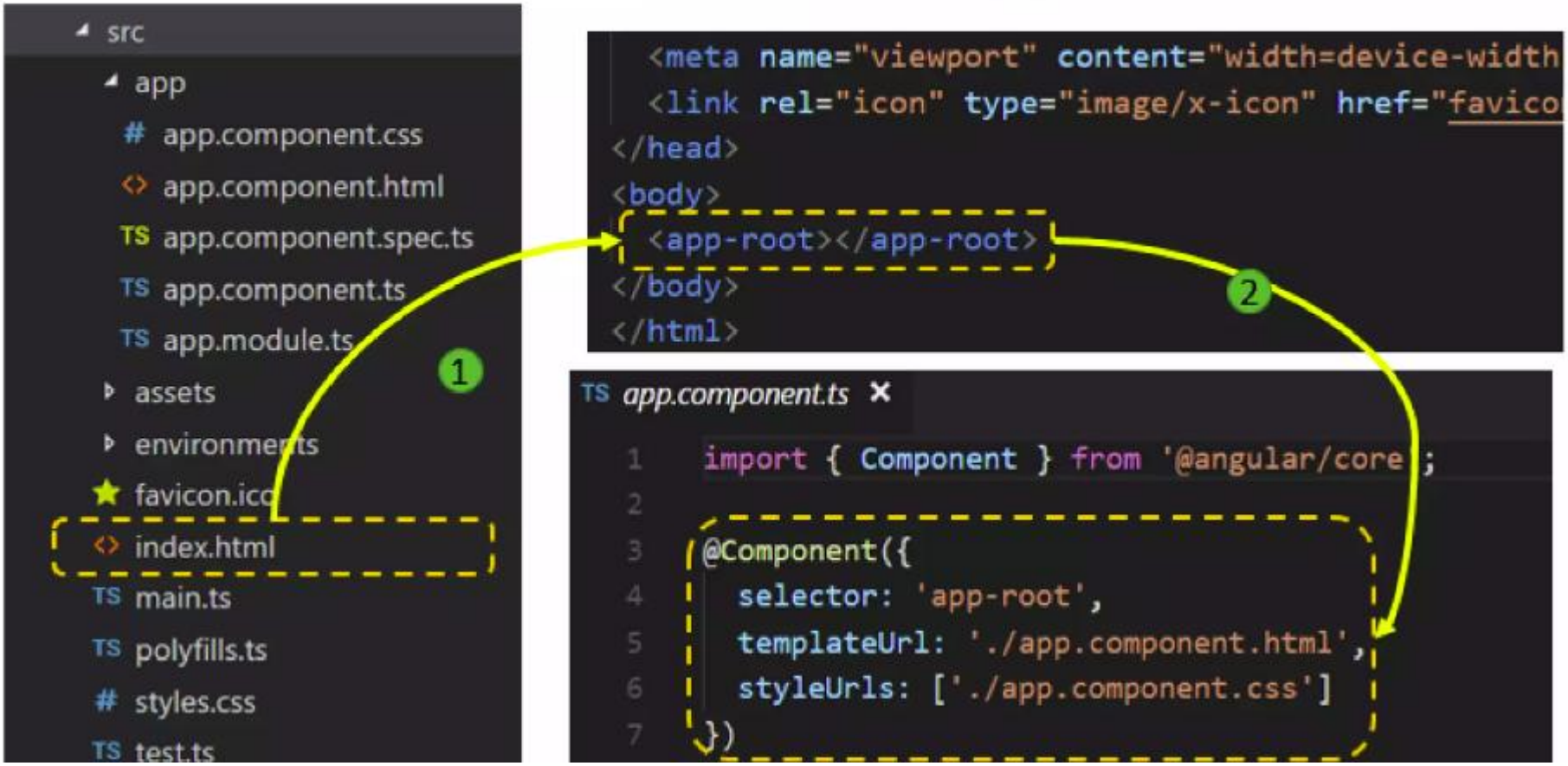


STRUCTURE PROJET ANGULAR DETAILLÉE





SELECTEUR


src
app
app.component.css
app.component.html
app.component.spec.ts
app.component.ts
app.module.ts
assets
environments
favicon.ico
index.html
main.ts
polyfills.ts
styles.css
test.ts
1
<meta name="viewport" content="width=device-width
<link rel="icon" type="image/x-icon" href="favico
</head>
<body>
<app-root></app-root>
</body>
</html>
2
app.component.ts
import { Component } from '@angular/core';
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})

Angular: Data Binding

- *Insertion dynamique de donnée dans l'application*
- ***One-way** data binding injecte du contenu dans la **vue** :*

```
<h1>{{name }}</h1>
```

- ***Property** binding :*

```
<button [disabled] = « ! isValid »> Valider </button>
```

- ***Two-way** data binding :*

```
<input [( name )] = « newName » />
```

- ***Event** binding :*

```
<button (click) = 'envoyer()' > Envoyer </button>
```

Directives Angular

- *Interaction direct avec le DOM de la page **HTML***
- ***Ajout – suppression – modification** des éléments au cours de l'exécution de la page*
- ***NgIf** rend **oui** ou **non** un élément **HTML***

```
<div *NgIf= « condition »> Hello World </div>
```

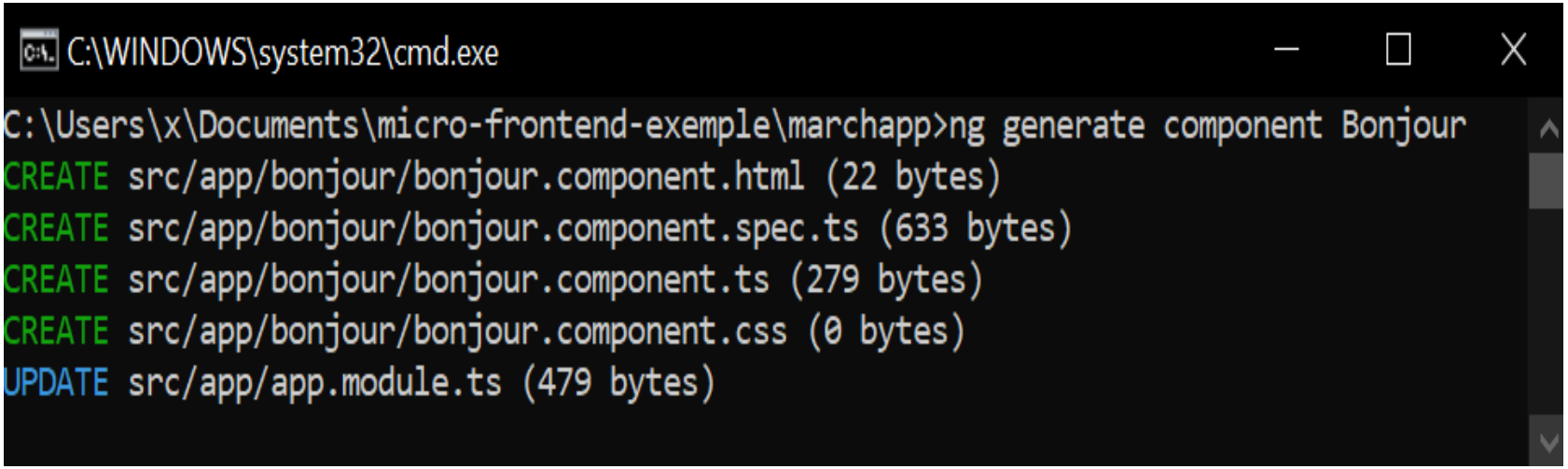
- ***NgFor** itère sur une **collection** afin d'appliquer un Template*

```
<li *ngFor = « let idea of ideas » > …… </li>
```

Angular: les composants

- *Exemple de Création du composant Bonjour* :

  ng generate component Bonjour



Angular: les routes (1)

- Principe du **SPA (Single Page Application)**
  - charger une **seule** page (index.html)
  - y présenter, au besoin, tout composant de l'application

- Problème :

Comment **naviguer** entre les différents **composants**, en changeant l'**URL** du navigateur, en modifiant l'historique du navigateur et en gardant **l'état** de l'interface utilisateur synchronisé **?**

- **Solution** : **Routage**

- **Objectif** : faire correspondre un chemin (**Path**) donnée de l'URL à un **composant** et afficher sa **vue** précise

Angular: les routes (2)

- **Système de routage : Les routes dans Angular sont gérés par le fichier «app-routing.module.ts »**

app-routing.module.ts M

src > app > app-routing.module.ts > routes

```ts
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { BonjourComponent } from './bonjour/bonjour.component';
const routes: Routes = [
  { path: 'bonjour', component: BonjourComponent},
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

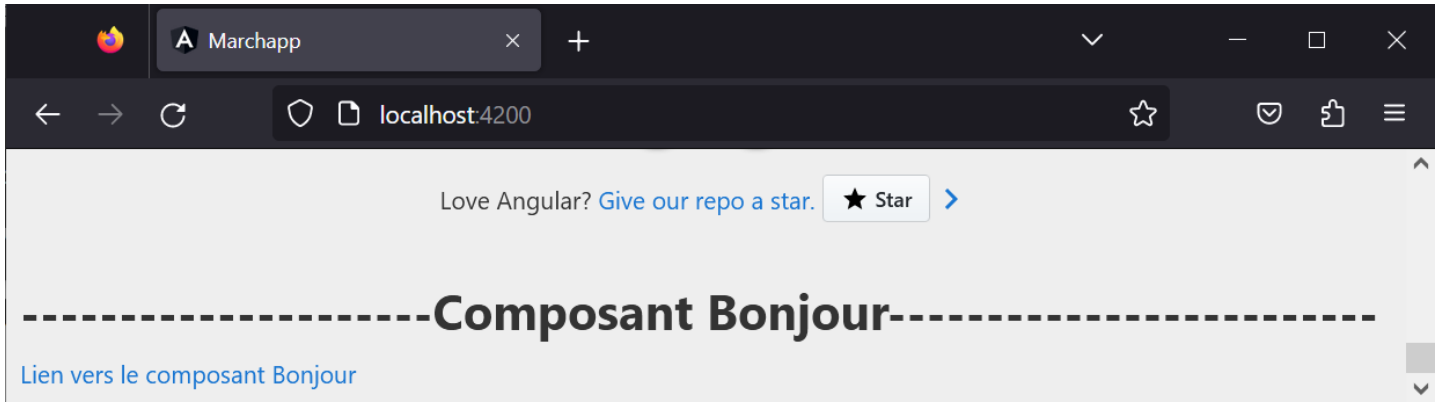
Angular: les liens

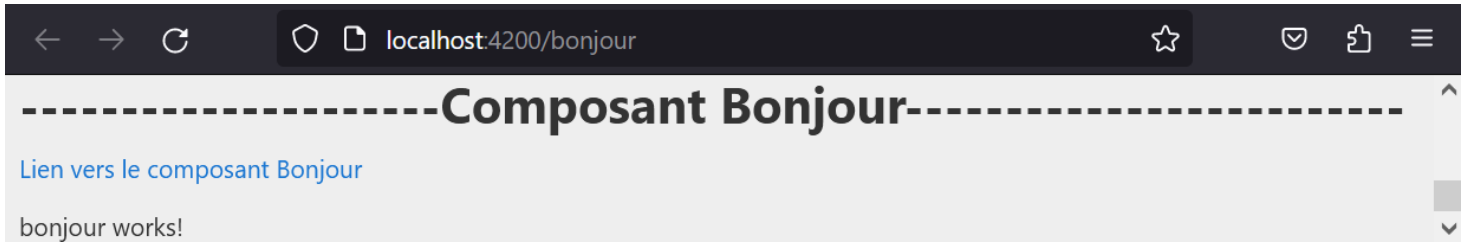
Exemple : On veut créer un lien vers le composant Bonjour : « /bonjour », pour cela on ajoute au fichier « app.component.html » le code suivant :

app.component.html M

src > app > app.component.html > ...

```
474   </div>
475   <div><h1>--------------------Composant Bonjour-----------------------</h1>
476     <a href="/bonjour" target="_blank" rel="noopener"> Lien vers le composant Bonjour
477
478     </a>
479   </div>
```





Angular : Passage de paramètres (1)

- **Exemple :**

On veut calculer la some de param1+param2 , **On modifie le fichier de routes :**

TS app-routing.module.ts M

src > app > TS app-routing.module.ts > AppRoutingModule

```
1   import { NgModule } from '@angular/core';
2   import { RouterModule, Routes } from '@angular/router';
3   import { BonjourComponent } from './bonjour/bonjour.component';
4   const routes: Routes = [
5     { path: 'bonjour/:param1/:param2', component: BonjourComponent},
```

Puis, on modifier les deux fichier Bonjour.component.ts (contrôleur) et Bonjour.component.html (Vue):

TS bonjour.component.ts U ● <> bonjour.component.html U ●

src > app > bonjour > <> bonjour.component.html > div > p

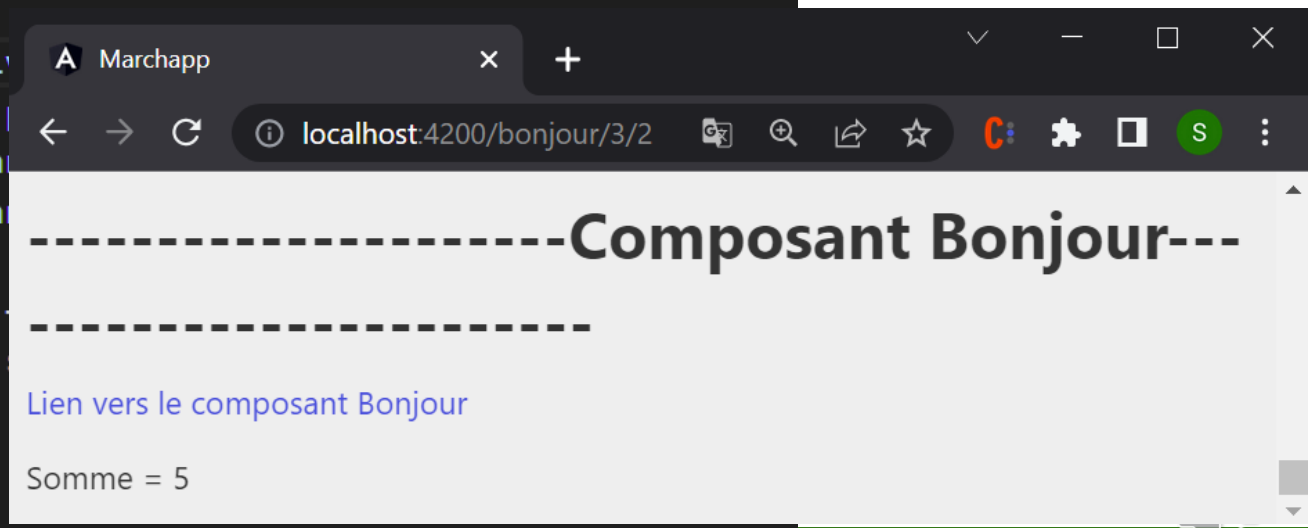
```
1   <div>
2     <p>Somme = {{somme}}</p>
3   </div>
```

Angular : passage de paramètres (2)

bonjour.component.ts U

src > app > bonjour > TS bonjour.component.ts > BonjourComponent > constructor

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

@Component({
  selector: 'app-bonjour',
  templateUrl: './bonjour.component.html',
  styleUrls: ['./bonjour.component.css']
})

export class BonjourComponent implements OnInit {
somme :number=0;

constructor(private acti
    this.activatedRoute.
      let p1:number = pa
      let p2:number = pa

        this.somme= +p1
        console.log("La
      }
      );
  }
```



Angular: exemple des opération arithmétique (1)

- *On veut améliorer notre exemple en ajoutant d'autres types d'opération : Soustraction (-), multiplication (*) et la division (/) de deux nombres.*

  - *La Première modification est dans le fichier app-routing.module.ts en ajoutant le paramètre 3: typeop:*

```
4  const routes: Routes = [
5    { path: 'bonjour/:param1/:param2/:typeop', component: BonjourComponent},
6  ];
```

  - *Puis, on modifie les deux fichier Bonjour.component.ts (contrôleur) et Bonjour.component.html (Vue):*

TS bonjour.component.ts U <> bonjour.component.html U

src > app > bonjour > <> bonjour.component.html > div

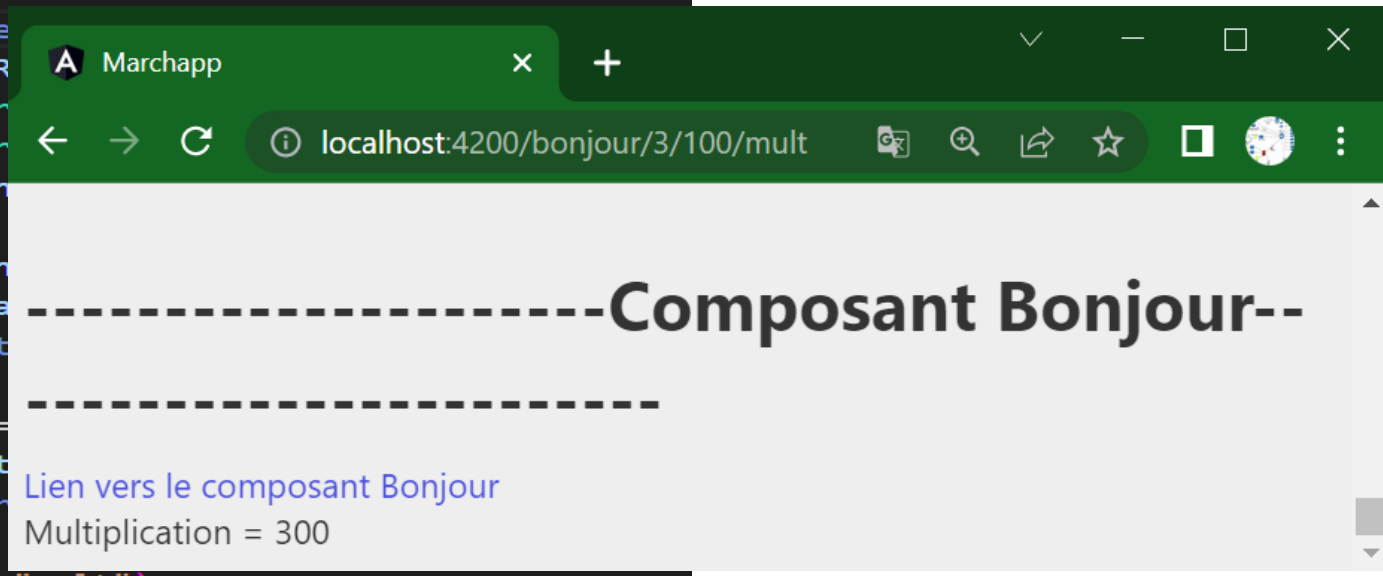
```
1  <div *ngIf="isSome">Somme = {{resultat}}</div>
2  <div *ngIf="isDiv">Division = {{resultat}}</div>
3  <div *ngIf="isSous">Soustraction = {{resultat}}</div>
4  <div *ngIf="isMult">Multiplication = {{resultat}}</div>
```

Angular : exemple des opération arithmétique (2)

TS bonjour.component.ts U

src > app > bonjour > TS bonjour.component.ts > BonjourComponent >

```
export class BonjourComponent implements OnInit {
resultat :number=0;
isSome : boolean=false;
isSous : boolean=false;
isMult : boolean=false;
isDiv : boolean=false;
typeoperation:any;

constructor(private
    this.activatedR
      let p1:number
      let p2:number
      let operation

      if (operation
     { this.resulta
      this.isSome=t
    }
    if (operation =
    { this.resultat
     this.isSous=tr
   }
   if (operation =="mult")
   { this.resultat= this.Multiplication(p1, p2);
    this.isMult=true ;
  }
  if (operation =="div")
  { this.resultat= this.Division(p1, p2);
   this.isSous=true ;
```



Angular: les formulaires (1)

- Dans Angular, il existe deux types de formulaires :
  - Les formulaires réactifs **(ReactiveFormsModule)**
  - Les formulaires template-driven **(FormsModule)**

- les formulaires template-driven sont **plus simples** à mettre en place. Mais , on vous recommande d'utiliser les formulaires **réactifs** car ils sont plus **puissants** et plus **flexibles**.

Angular: les formulaires (2)

FormsModule est un **module** Angular qui contient une série de **directives** et de **services** qui vous permettent de créer et de gérer des formulaires dans votre application Angular. Il contient notamment les directives suivantes :

- **NgForm** : lie automatiquement un **objet au formulaire** pour sa **validation** et son **traitement**
- **NgModel :** lier la valeur d'un **champ** de formulaire à une **propriété** d'un composant.
- **.value :**
  - propriété de l'objet lié au formulaire
  - objet ayant pour propriétés les noms des champs enregistrés
  - chaque nom de champ stocke la valeur du champ
- **NgSubmit : détecte la soumission du formulaire**
- **NB : FormsModule** est à importer depuis **@angular/forms** et doit être ajouté dans la section imports de **app.module.ts**

Angular: les formulaires (3)

exemple : ajouter un étudiant

- Générer le composant addStudent
- Dans le template de AddStudent, coder le formulaire

```
<form (ngSubmit)="handleAdd(myForm)" #myForm="ngForm" >
  <div> Prenom : <input name="prenom" ngModel ></div>
  <div> Nom : <input name="nom" ngModel > </div>
  <div> <button type="submit" >Ajouter</button> </div>
</form>
```

NB : #myForm="ngForm" assignation d'un objet NgForm à la variable locale myForm associée au formulaire

- Dans la classe de AddStudent, ajouter

```
@Output() studentToAdd:EventEmitter<any> = new EventEmitter();
handleAdd(myForm:NgForm){
  this.studentToAdd.emit(myForm.value);
  myForm.reset();
}
```

NB : importer NgForm à partir de @angular/forms

Angular: les formulaires (4)

Exemple : On veut créer une FORM qui contient le champ nom dans le composant Bonjour : « /bonjour », pour cela on ajoute au fichier « Bonjour.component.html » le code suivant :

bonjour.component.html U

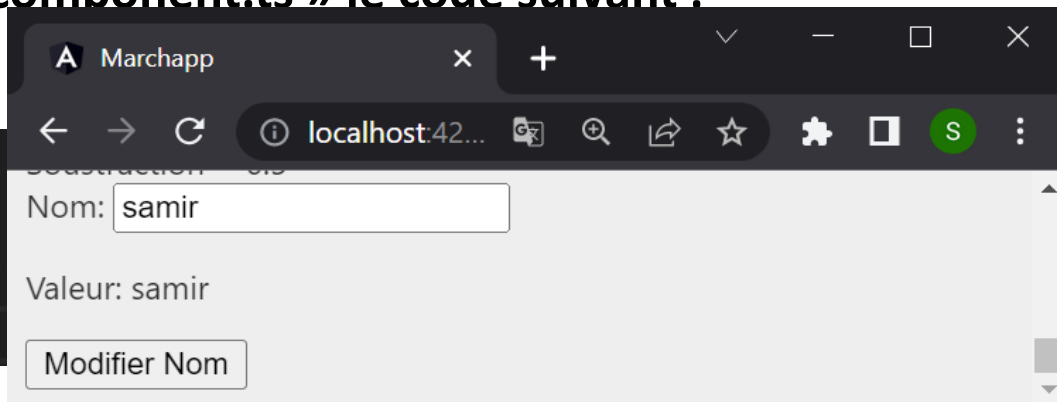
src > app > bonjour > bonjour.component.html > div > label

```
5    <div>
6    <label for="name">Nom: </label>
7    <input id="name" type="text" [formControl]="name">
8    <p>Valeur: {{ name.value }}</p>
9    <button type="button" (click)="updateName()">Modifier Nom</button>
10   </div>
```

On import : `import { FormControl } from '@angular/forms';`

Et on ajoute au fichier Bonjour.component.ts » le code suivant :

```
51   name = new FormControl('');
52     updateName() {
53     this.name.setValue('samir'
54   }
```



Angular :validation des formulaires (1)

Permet de vérifier la cohérence les données d'un formulaire avant de les sauvegarder dans la base de données et cela en définissant un ensemble de règle à respecter.

- Quelques **règles** de validation : **required**, **number**, **email**, **maxlength**, **minlength**, **pattern**
- Qlq **propriétés** de validation d'un **formulaire** ou d'un **champ**
  - **valid** : vraie si toutes les règles sont respectées
  - **invalid** : vraie si au moins une des règles est violée
  - **touched** : vraie si l'élément a été visité
  - **pristine** : vraie si l'élément n'a pas été modifié
  - **dirty** : vraie si l'élément a été modifié
  - **errors.<regle>** : vraie si la règle de **validation** est violée
- Syntaxes d'accès: **nomFormulaire.propriété** et **nomChamp.propriété** avec **#nomFormulaire="ngForm"** et **#nomChamp**="**ngModel**"
- Afficher tout message d'erreur avec la directive **NgIf**

Angular : validation des formulaires (2)

```
<form (ngSubmit)="handleAdd(myForm)" #myForm="ngForm" >
    <div>
        Prenom : <input name="prenom" ngModel required minlength="3" #prenom="ngModel" >
        <span style="color:red" *ngIf="prenom.invalid && (prenom.dirty || prenom.touched)
        ">
          <span *ngIf="prenom.errors.required">Le prenom est obligatoire</span>
          <span *ngIf="prenom.errors.minlength">Le prenom doit avoir au moins 3 lettres</
          span>
        </span>
    </div>
    <div>
        Nom : <input name="nom" ngModel required #nom="ngModel" >
        <span style="color:red" *ngIf="nom.invalid && (nom.dirty || nom.touched)">
          <span *ngIf="nom.errors.required">Le nom est obligatoire</span>
        </span>
    </div>
    <div>
      <button type="submit" [disabled]="myForm.invalid" >Ajouter</button>
    </div>
</form>
```

Angular : Services (1)

- **Pbl** : comment isoler certaines données et fonctionnalités **réutilisables** ?

Exemple : communiquer avec le backend

- **Sol** : enveloppées ces données et fonctionnalités réutilisables dans des classes **détachées** de l'interface
- **Service** : classe **TypeScript** décorée par **@Injectable** dont l'objectif est d'organiser et de **partager** le **code métier**
- Exemple de service natif : **HttpClient** pour exécuter des requêtes **AJAX**
- Commande de création :[***ng generate service*** *nomService*]
- Il est fourni (provide), par défaut, à la « **root injector** » pour que sa **réutilisation** dans l'application soit globale
- **singleton** : instancié une seule fois et cette instance est ensuite utilisée partout

Angular : Services (2)

Communiquer avec un serveur: présentation de **HttpClient**

- **HttpClient** : service qui facilite la communication avec un serveur **HTTP** distant via l'objet **XMLHttpRequest**
- **HttpClient** dispose des méthodes **get()**, **post()**, **put()**, **patch()** , **delete()** pour effectuer des requêtes HTTP
- Chacune des méthodes construit un « Observable » qui attend un abonnement (**.subscribe()**) pour exécuter la requête
- **get()** permet de retourner le type de l'objet attendu avec la syntaxe suivante : **.get<***typeAttendu***>()**
- **Prérequis** d'utilisation de HttpClient
  - **Importer HttpClientModule** dans **AppModule**
  - **Injecter HttpClient** dans un service de l'application
  - **HttpClientModule** et ***HttpClient*** sont dans **@angular/common/http**

Etude de cas 1:
Gestion des utilisateurs Frontend

Gestion des utilisateurs Frontend

Création des modules pour notre application :

1- Générer les composants suivants:

- **ng generate component Userr1**
- **ng generate component add-userr**

2- Générer le service suivant:

- **ng generate service services/httpClient**

3- Générer le modèle Userr1:

- **ng g class models/Userr1 --type=model**

Création des modules pour notre application :

```
C:\WINDOWS\system32\cmd.exe

C:\Users\x\Documents\micro-frontend-exemple\marchapp>ng generate component Userr1
CREATE src/app/userr1/userr1.component.html (21 bytes)
CREATE src/app/userr1/userr1.component.spec.ts (626 bytes)
CREATE src/app/userr1/userr1.component.ts (275 bytes)
CREATE src/app/userr1/userr1.component.css (0 bytes)
UPDATE src/app/app.module.ts (640 bytes)

C:\Users\x\Documents\micro-frontend-exemple\marchapp>
C:\Users\x\Documents\micro-frontend-exemple\marchapp>ng generate component add-userr
CREATE src/app/add-userr/add-userr.component.html (24 bytes)
CREATE src/app/add-userr/add-userr.component.spec.ts (641 bytes)
CREATE src/app/add-userr/add-userr.component.ts (286 bytes)
CREATE src/app/add-userr/add-userr.component.css (0 bytes)
UPDATE src/app/app.module.ts (732 bytes)

C:\Users\x\Documents\micro-frontend-exemple\marchapp>
C:\Users\x\Documents\micro-frontend-exemple\marchapp>ng generate service services/httpClient
CREATE src/app/services/http-client.service.spec.ts (378 bytes)
CREATE src/app/services/http-client.service.ts (139 bytes)

C:\Users\x\Documents\micro-frontend-exemple\marchapp>
C:\Users\x\Documents\micro-frontend-exemple\marchapp>ng g class models/Userr1 --type=model
CREATE src/app/models/userr1.model.spec.ts (160 bytes)
CREATE src/app/models/userr1.model.ts (24 bytes)
```

Etude de cas 1: Paramétrage app.module.ts

TS app.module.ts M

src > app > TS app.module.ts > AppModule

```ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { BonjourComponent } from './bonjour/bonjour.component';
import { ReactiveFormsModule } from '@angular/forms';
import { Userr1Component } from './userr1/userr1.component';
import { AddUserrComponent } from './add-userr/add-userr.component';
import { NgbModule } from '@ng-bootstrap/ng-bootstrap';

@NgModule({
  declarations: [
    AppComponent,
    BonjourComponent,
    Userr1Component,
    AddUserrComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    AppRoutingModule,
    ReactiveFormsModule,
    FormsModule,
    NgbModule
```

Etude de cas 1: Création du modèle Userr1

TS userr1.model.ts U

src > app > models > TS userr1.model.ts > ...

```ts

export class Userr1 {
    constructor(
        public  userName:string,
        public  nom:string,
        public  prnom:string,
        public  dateDeNaissance: Date,
        public  lieuDeNaissance:string,
        public  motPasse:string,
        public  role:number,
        public  numService:number,
        public  numUser:number ) {}
    }
```

Etude de cas 1: Création du contrôleur Userr1

userr1.component.ts

src > app > userr1 > userr1.component.ts > Userr1Component > handleSuccessfulResponse

```
11  @Injectable()
12  export class Userr1Component implements OnInit {
13    @Input() userrs:Userr1[];
14    RoleList:any[];
15     x: boolean= false;
16     user: Userr1 = new Userr1(null,null, null, null, null, null, null, null, null);
17    constructor( private httpClientService:HttpClientService, private router: Router) { }
18
19    ngOnInit() {
20      this.httpClientService.getUserrs().subscribe( response =>this.handleSuccessfulResponse(response));
21      this.httpClientService.getRoles().subscribe(courrierras => this.RoleList = courrierras);
22    }
23    deleteUserr(employee: Userr1): void {
24      this.httpClientService.deleteUserr(employee)
25        .subscribe( data => {
26          this.userrs = this.userrs.filter(u => u !== employee);
27        })
28    };
29    updateUserr(): void {
30
31      this.httpClientService.updateUserr(this.user.numUser, this.user )
32      .subscribe( data => {
33        alert("User updated successfully."); });
34    };
35
36    editUserr(employee: Userr1){
37      this.x=!(this.x);
38      this.user=employee;    }
39
40  handleSuccessfulResponse(response)
41  {
42      this.userrs=response;}
```

Etude de cas 1: Création de la vue Userr1 (1)

-Afficher la liste des utilisateur

userr1.component.html

src > app > userr1 > userr1.component.html > div.col-md-6

```
<div class="col-md-6">
  <h2 class="text-center">Liste des Utilisateurs</h2>
<table class="table table-striped">
    <thead>
    <tr>
        <th>Numero</th>
        <th>Nom Utilisateur</th>
      <th>Direction</th>
      <th>role</th>
    </tr>
    </thead>
    <tbody>
      <tr *ngFor="let userr1 of userrs">
        <td>{{userr1.numUser}}</td>
        <td>{{userr1.userName}}</td>
        <td>{{getEntiteLib(userr1.numService)}}</td>
        <td>{{userr1.role}}</td>
        <td><button class="btn btn-danger"  (click)="deleteUserr(userr1)">Delete </button></td>
                <td><button class="btn btn-success" (click)="editUserr(userr1)">Update</button></td>

              </tr>
    </tbody>
      </table>
</div>
```

Etude de cas 1: Création de la vue Userr1 (2)

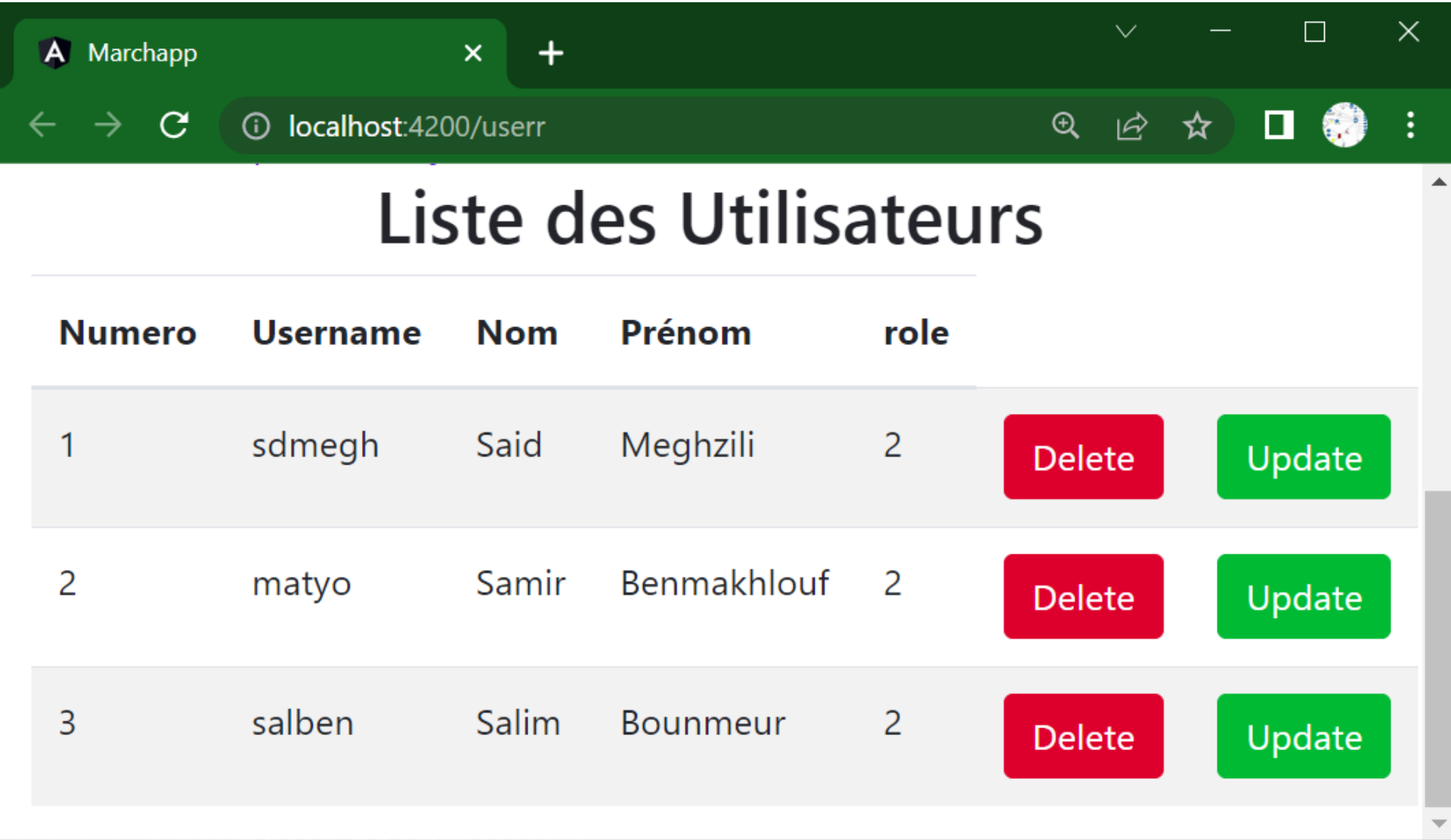
-Modifier un utilisateur

userr1.component.html

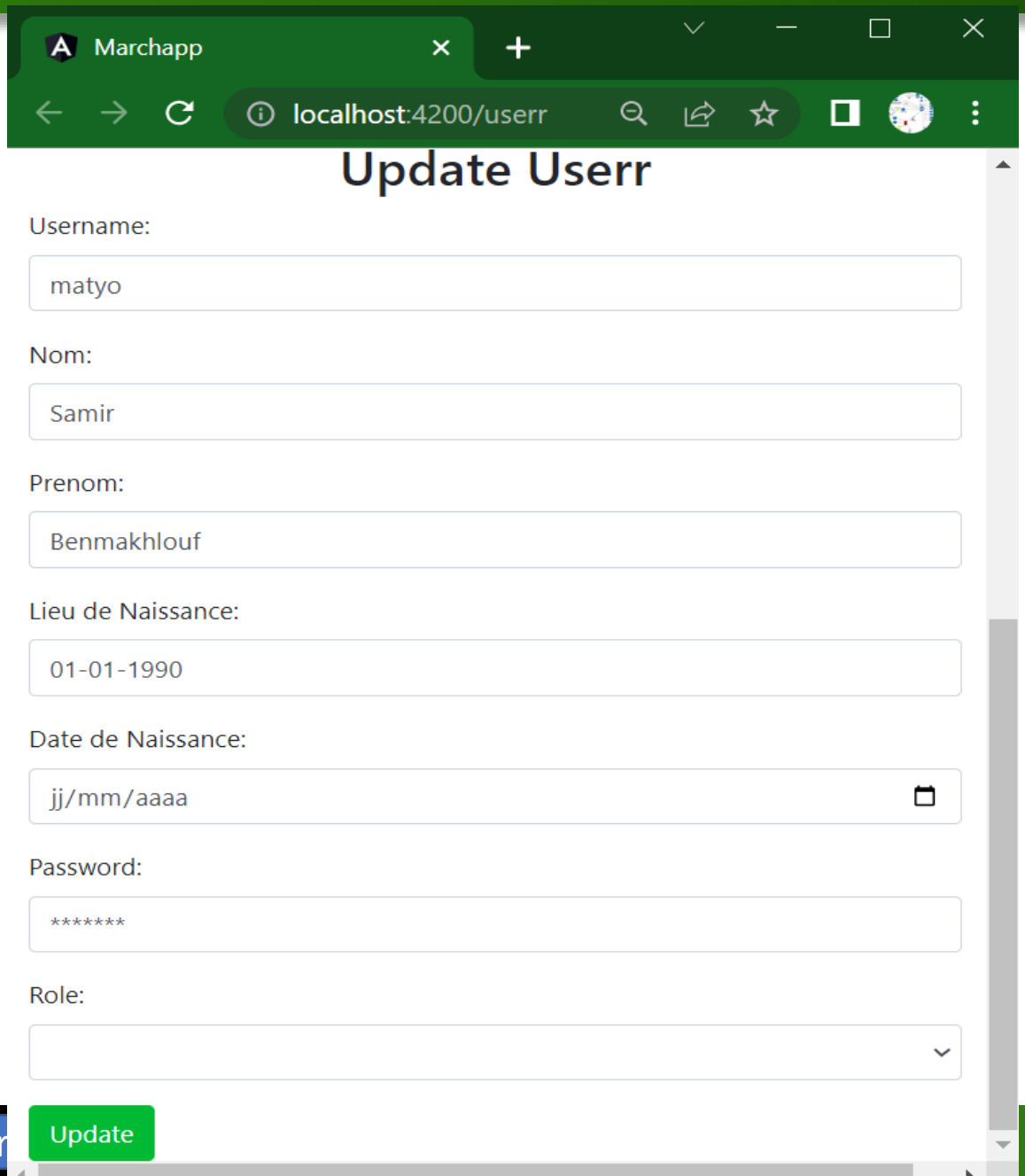
src > app > userr1 > userr1.component.html > div.col-md-6 > form > div.form-group > select#role.form-control

```
26  <div class="col-md-6"  *ngIf="x">  <h2 class="text-center">Update Userr</h2> <form>
27      <div class="form-group">
28        <label for="userName">Username:</label>
29        <input  type="text "[(ngModel)]="user.userName" name="user.userName"   class="form-control">
30      </div>
31      <div class="form-group">
32      <label for="nom">Nom:</label>
33      <input  [(ngModel)]="user.nom"  name="user.nom" class="form-control" id="name">
34    </div>
35    <div class="form-group">
36      <label for="prnom">Prenom:</label>
37      <input  [(ngModel)]="user.prnom"  name="user.prnom" class="form-control" id="prnom">
38    </div>
39    <div class="form-group">
40      <label for="lieuDeNaissance">Lieu de Naissance:</label>
41      <input  [(ngModel)]="user.lieuDeNaissance"  name="user.lieuDeNaissance" class="form-control" >
42    </div>
43    <div class="form-group">
44      <label for="dateDeNaissance">Date de Naissance:</label>
45      <input type="date" [(ngModel)]="user.dateDeNaissance"  name="user.dateDeNaissance" class="form-control">
46    </div>
47    <div class="form-group">
48      <label for="motPasse">Password:</label>
49      <input [(ngModel)]="user.motPasse"  name="user.motPasse" class="form-control" id="motPasse">
50    </div>
51    <div class="form-group">
52      <label for="role">Role:</label>
53      <select  class="form-control"  name="user.role" [(ngModel)]="user.role" id="role">
54        <option disabled>Sélectionner Role</option>
55        <option *ngFor="let r of RoleList"   [value]="r.id">{{r.libelle}}</option>
56      </select>
57    </div> <button class="btn btn-success" (click)="updateUserr()">Update</button> </form> </div>
```

Etude de cas 1: Création de la vue Userr1 (3)



Etude de cas 1: Modification d'un utilisateur



Etude de cas 1: Création du contrôleur add-User

add-userr.component.ts ●

src > app > add-userr > TS add-userr.component.ts > AddUserrComponent

```ts
import { Component, OnInit } from '@angular/core';
import { HttpClientService, Userr1, Entite  } from '../service/http-client.service';
@Component({
  selector: 'app-add-userr',
  templateUrl: './add-userr.component.html',
  styleUrls: ['./add-userr.component.css']
})
export class AddUserrComponent implements OnInit {

  RoleList: any[];
  user: Userr1 =new Userr1(null, null, null, null, null, null, null, null,null);
  constructor(private httpClientService: HttpClientService) { }

  ngOnInit(): void {

    this.httpClientService.getRoles().subscribe(courrierras => this.RoleList = courrierras);

  }

  createUserr(): void {
    this.httpClientService.createUserr(this.user)
        .subscribe( data => {
          alert("User created successfully.");  }); };}
```

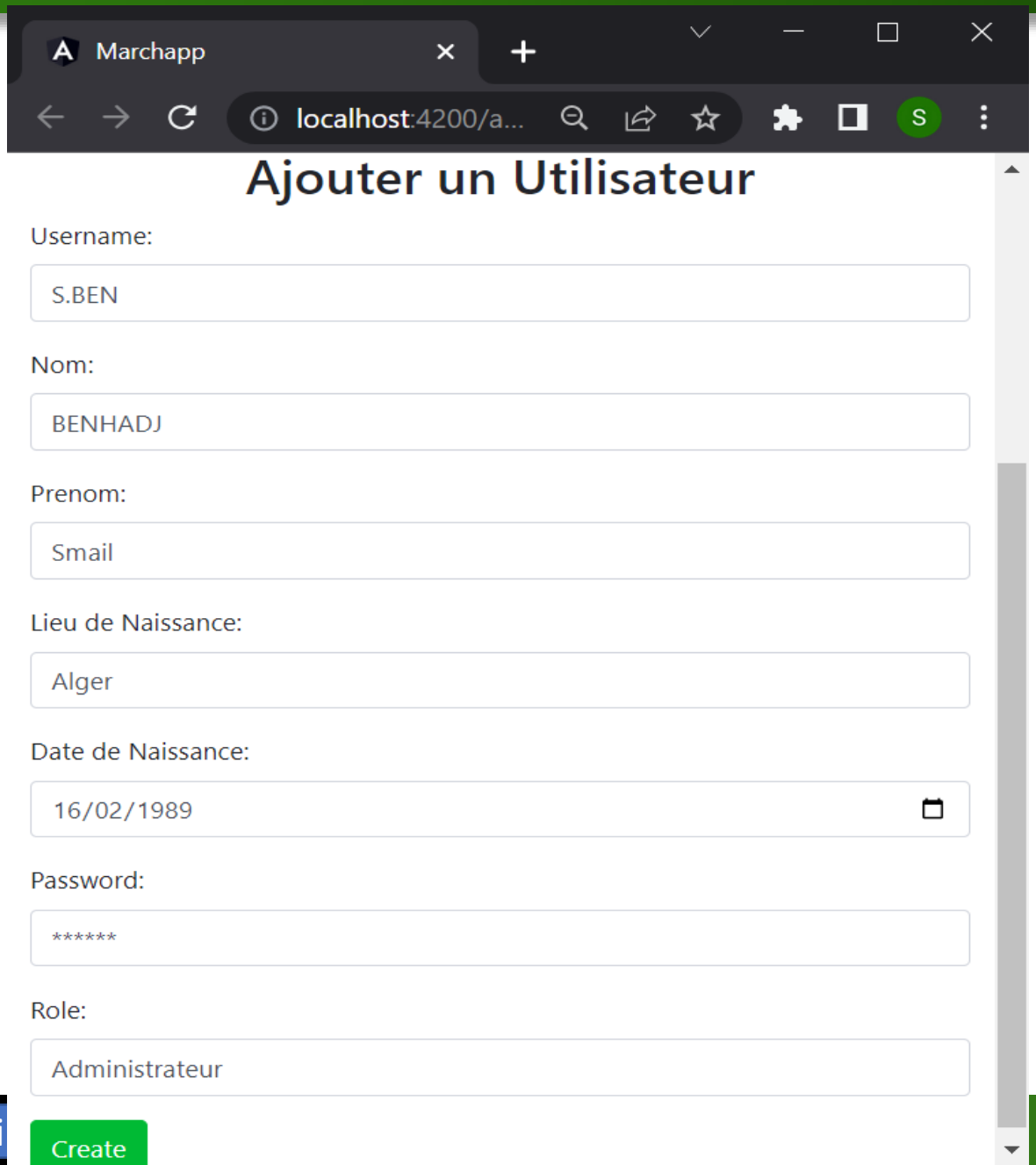
Etude de cas 1: Création de la vue add-User (1)

add-userr.component.html ●

src > app > add-userr > add-userr.component.html > div.col-md-6

```
<div class="col-md-6">     <h2 class="text-center">Ajouter un Utilisateur</h2>
    <form>
      <div class="form-group">
        <label for="userName">Username:</label>
        <input type="name" [(ngModel)]="user.userName"  name="name" class="form-control" id="username">
      </div>
      <div class="form-group">
      <label for="nom">Nom:</label>
      <input type="name" [(ngModel)]="user.nom"  name="name" class="form-control" id="name">
    </div>
    <div class="form-group">
      <label for="prnom">Prenom:</label>
      <input type="name" [(ngModel)]="user.prnom"  name="name" class="form-control" id="prnom">
    </div>
    <div class="form-group">
      <label for="lieuDeNaissance">Lieu de Naissance:</label>
      <input type="name" [(ngModel)]="user.lieuDeNaissance"  name="name" class="form-control" id="lieuDeNaissance">
    </div>
    <div class="form-group">
      <label for="dateDeNaissance">Date de Naissance:</label>
      <input type="date" [(ngModel)]="user.dateDeNaissance"  name="name" class="form-control" id="dateDeNaissance">
    </div>
    <div class="form-group">
      <label for="motPasse">Password:</label>
      <input type="name" [(ngModel)]="user.motPasse"  name="name" class="form-control" id="motPasse">
    </div>
    <div class="form-group">
      <label for="role">Role:</label>
      <input type="name" [(ngModel)]="user.role"  name="name" class="form-control" id="role">
    </div>
      <button class="btn btn-success" (click)="createUserr()">Create</button> </form> </div>
```

Etude de cas 1: Création de la vue add-User (2)



Etude de cas 1: Création du service HttpclientService

Permet la connexion avec l'application back-end

```
export class HttpClientService {
  private url = 'http://Www…….';

  constructor( private httpClient:HttpClient) { }

getUserrs()
{
  return this.httpClient.get<Userr1[]>(url);
}

public deleteUserr(employee) {
  return this.httpClient.delete<Userr1>(url + "/"+ employee.numUser);
}

public createUserr(employee) {
  return this.httpClient.post<Userr1>(url, employee);

}
public updateUserr(id,employee) {
return this.httpClient.put<Userr1>(url+ "/"+id,employee);
```

Voila comment inclure Bootstrap 5 dans les applications Angular 15 à l'aide de la bibliothèque ng-bootstrap:

npm install bootstrap

npm i @ng-bootstrap/ng-bootstrap --legacy-peer-deps

npm uninstall @ng-bootstrap/ng-bootstrap

ng add @ng-bootstrap/ng-bootstrap

Framework Spring Boot

Framework Spring

- *Application open source (Libre)*
- *Plugins et prise en charge de la plupart des technologies Web*
- *De nombreuses API utilitaires*
- *Extrêmement populaire*
- *Prise en charge de Java, Groovy e*
- *Bon pour le cloud et les microservices*

- **Problème: Framework puissant mais lourd à configurer**

Spring Boot ?

- *Spring Boot est un **module** de Spring Framework. **Il nous permet de créer une application autonome avec des configurations minimales ou nulles.***
- ***Il est préférable de l'utiliser si nous voulons développer une simple application basée sur Spring ou des services REST.***
  - *Applications Spring autonomes*
  - *Serveur Web intégré*
  - *Accélérer le développement d'applications Spring*
  - *Configurer automatiquement Spring dans la mesure du possible*
  - *Pas de code généré*
  - *Déploiement facile*
  - *Prêt pour la production*

Utilisation de Spring Boot

- **– Créer un projet Maven qui hérite d'un parent**
  - • spring-boot-starter-parent

- **– Insérer les dépendances et les plugins nécessaires dans le pom.xml**
- **– Lier le code à une classe principale**
  - • public static void main

- **– Lancer l'application:**
- **• Avec un goal Maven spécifique : mvn spring-boot:run**
- **• Comme une application Java standard : java -jar …**
- **• Générer un fichier war et le déployer sur votre serveur**

Exemple Pom.xml

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.0.BUILD-SNAPSHOT</version>
    <relativePath/>
</parent>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
```

Auto configuration

```
@SpringBootApplication
public class DemoApplication {
public static void main(String[] args) {
    SpringApplication.run(DemoApplication.class, args); } }
```

- L'annotation ***@SpringBootApplication*** déclenche la configuration automatique de l'infrastructure Spring
- Au démarrage de l'application, Spring Boot :
  - Scanne toutes classes de ***@Configuration***
    - Classes de configuration spécifiques à l'application
    - Classes de Spring Boot **su**ffixées par ***AutoConfiguration***
  - Utilise les **JAR** présents dans le **classpath** pour prendre des décisions

Auto configuration (2)

@SpringBootApplication : Rassemble les annotations

– @ComponentScan : déclenche la recherche de composants dans les sous-packages

– @Configuration : spécifie une classe de configuration

– @EnableAutoConfiguration : Identifie la classe principale et démarre la détection de configuration

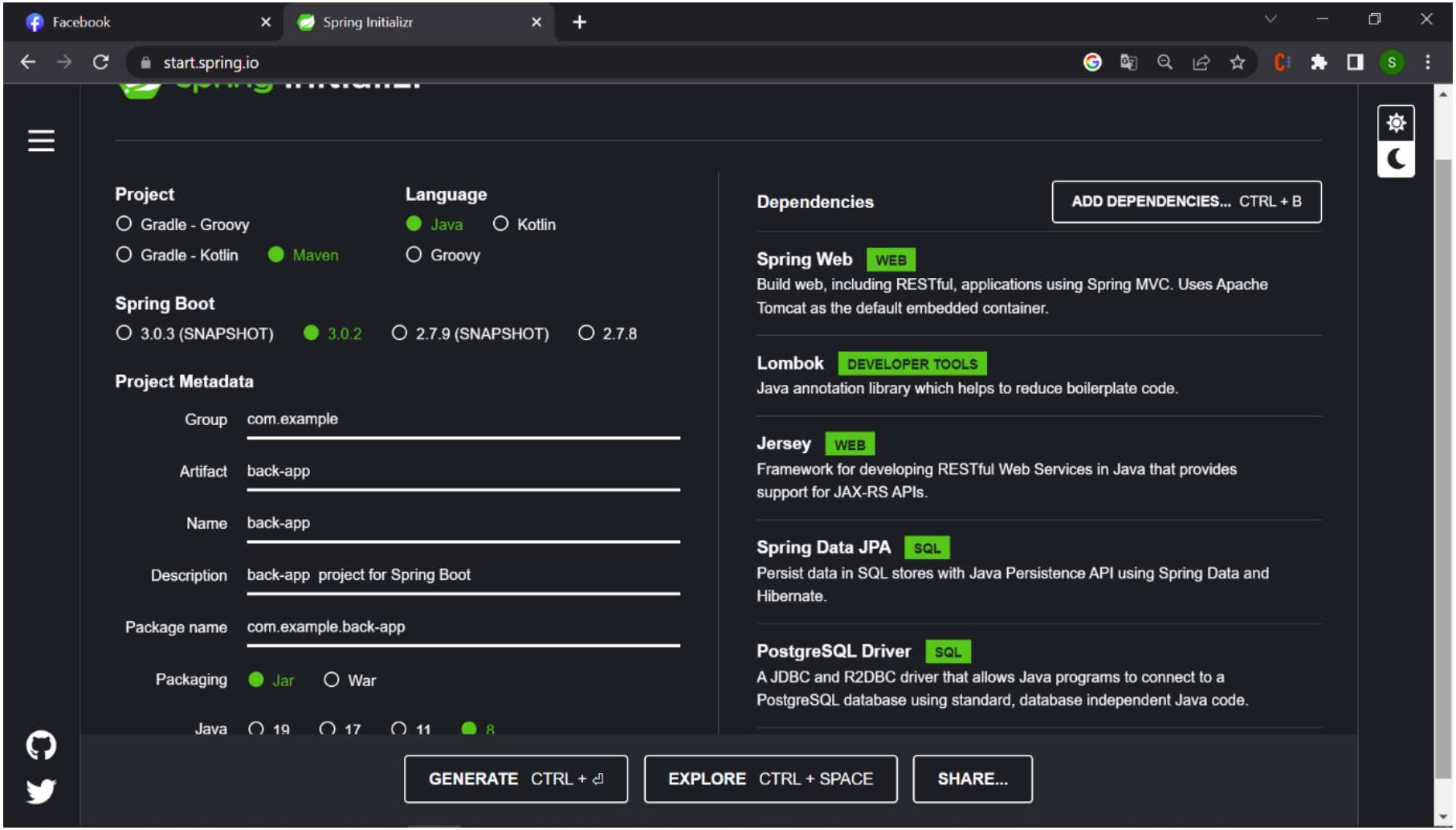
- **À utiliser :**

– Pour « économiser » les annotations dans un projet standard

Création d’un projet (1)

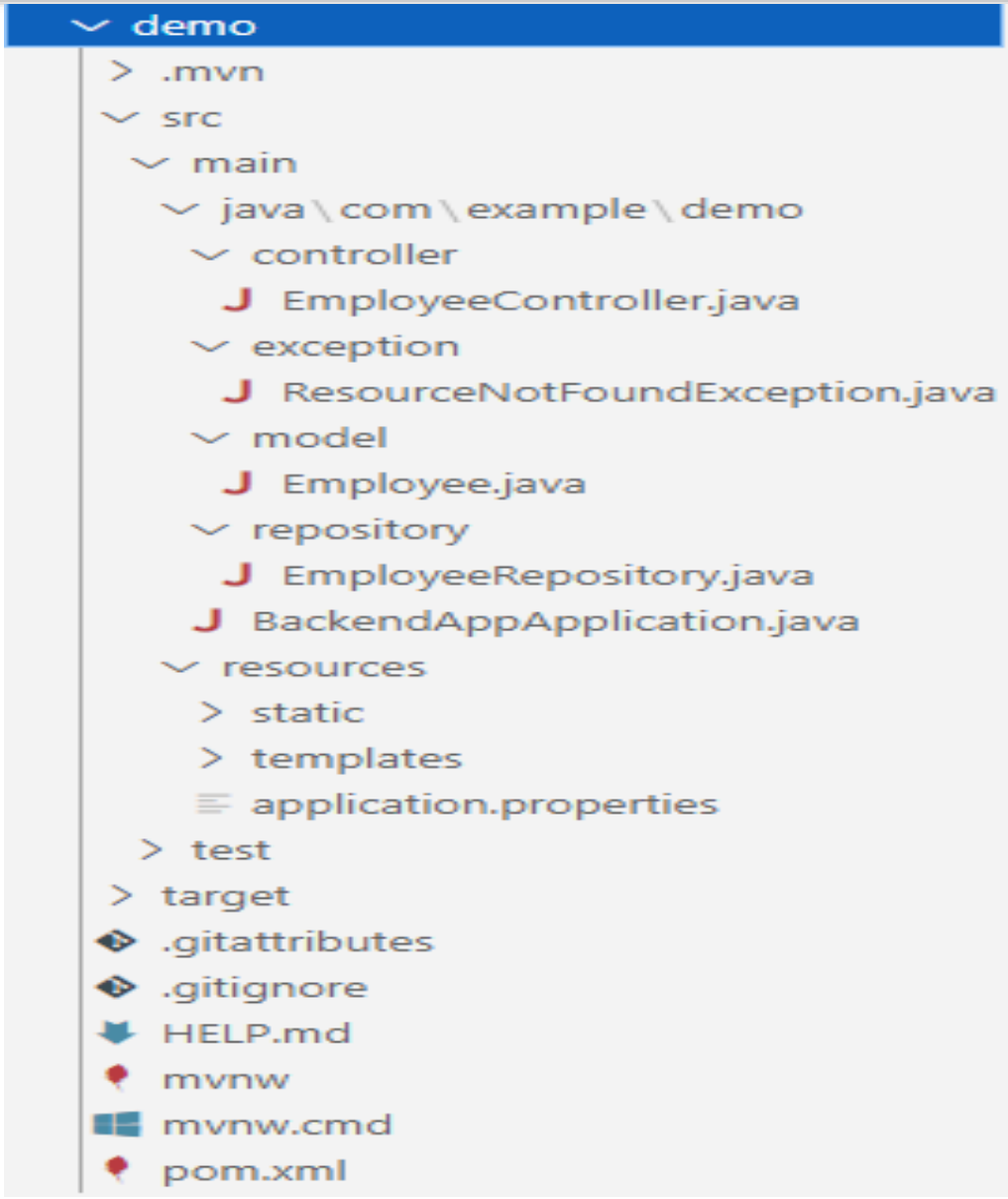
- **• Création d’un projet vide Spring Initializr : https://start.spring.io/**

Création d'un projet (1)



Etude de cas : Backend spring Boot
Gestion des Employees (suite)

Structure du Projet (1)



Structure du Projet (2)

– ***Employee:*** *classe de modèle de données des* ***employees*** *correspond aux* ***employees*** *d'entité et de table.*

– ***EmployeeRepository*** *: est une interface qui étend* ***JpaRepository*** *pour les méthodes* ***CRUD*** *et les méthodes de recherche personnalisées. Il sera câblé automatiquement dans* ***EmployeeController****.*

– ***EmployeeController*** *est un* ***RestController*** *qui a des méthodes de mappage de requêtes pour les requêtes* ***RESTful*** *telles que :* ***getAllEmployees, createEmployee, updateEmployee, deleteEmployee, findById…***

– ***application.properties :*** *Configuration pour Spring* ***Datasource****,* ***JPA*** *&* ***Hibernate****.*

– ***pom.xml*** *contient des dépendances pour Spring Boot et la base de données.*

Configuration du dépendance fichier pom.xml

*back-app_/pom.xml

```
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
            <version>3.0.2</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jersey</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
```

Configuration : Datasource, JPA, Hibernate (1)

fichier application.properties

application.properties

```
1 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
2 spring.jpa.hibernate.ddl-auto= update
3 spring.jpa.hibernate.show-sql=true
4 spring.datasource.url=jdbc:postgresql://localhost:5432/user
5 spring.datasource.username=postgres
6 spring.datasource.password=1234
```

Configuration : Datasource, JPA, Hibernate (2)

- **spring.datasource.url** : URL de la base de données.
- **spring.datasource.username** et **spring.datasource.password** sont identiques à celles de votre installation de base de données.

- **Spring Boot** utilise Hibernate pour l'implémentation de **JPA**, nous configurons **Dialect** pour la base de données

- **spring.jpa.hibernate.ddl-auto : (create, update ou validate)**
  - Est utilisé pour l'**initialisation** de la base de données (**create**).
  - Nous définissons la valeur « **update »** pour **mettre à jour** la base de données correspondant au modèle de données défini.
  - Pour la **production**, cette propriété doit être validée (**valide**).

Modèle de données (1)

Employee.java

demo > src > main > java > com > example > demo > model > Employee.java > Employee

```java
package com.example.demo.model;

import jakarta.persistence.*;

@Entity
@Table(name = "employees")
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Column(name = "email_id")
    private String emailId;

```

Modèle de données (2)

@Entity: indique que la classe est une classe Java persistante.

@Table : fournit la table qui mappe cette entité**.**

@Id : concerne la clé primaire**.**

@GeneratedValue: est utilisée pour définir la stratégie de génération de la clé primaire**.**

GenerationType.AUTO : signifie champ d'incrémentation automatique**.**

@Column est utilisée pour définir la colonne dans la base de données qui mappe le champ annoté.

Repository Interface

EmployeeRepository.java

demo > src > main > java > com > example > demo > repository > EmployeeRepository.java > Employe

```java
package com.example.demo.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.example.demo.model.Employee;

@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long>{}
```

- *Permet l'interaction avec la Base de données*

nous pouvons utiliser les méthodes de **JpaRepository : save(), findOne(), findById(), findAll(), count(), delete(), deleteById()…** sans implémenter ces méthodes**.**

Nous pouvons définir également des méthodes de recherche personnalisées :– **findByName(string nom) : renvoie tous les utilisateurs dont la valeur = « nom».**

– findByNameContaining(string x) : renvoie tous les users dont le nom contient l'entrée x.

Spring Rest APIs Controller (1)

- *Le contrôleur fournit des API pour **créer**, **récupérer**, **mettre à jour**, **supprimer** et **trouver** des utilisateurs.*

– ***@CrossOrigin :*** *sert à configurer les origines autorisées.*

– ***@RestController :****est utilisée pour définir un contrôleur et pour indiquer que la valeur de retour des méthodes doit être liée au corps de la réponse Web.*

– ***@RequestMapping("/employees")*** *déclare que toutes les URL des API dans le contrôleur commenceront par : /employees.*

– *Nous utilisons **@Autowired** pour injecter le bean **EmloyeeRepository** dans la variable locale.*

Spring Rest APIs Controller (2)

EmployeeController.java

demo > src > main > java > com > example > demo > controller > EmployeeController.java

```
23  @CrossOrigin(origins = "http://localhost:4200")
24  @RestController
25  @RequestMapping("/api/v1/")
26  public class EmployeeController {
27
28      @Autowired
29      private EmployeeRepository employeeRepository;
30
31      // get all employees
32      @GetMapping("/employees")
33      public List<Employee> getAllEmployees(){
34          return employeeRepository.findAll();
35      }
36
37      // create employee rest api
38      @PostMapping("/employees")
39      public Employee createEmployee(@RequestBody Employee employee) {
40          return employeeRepository.save(employee);
41      }
```

Persistance des données:

JPA/Hibernate
Spring Data

Framework Hibernate

1 Introduction:

2 Jakarta Persistence API

2 Object-Relational Mapping (ORM)

3 Entity Bean

4 Persistance des champs

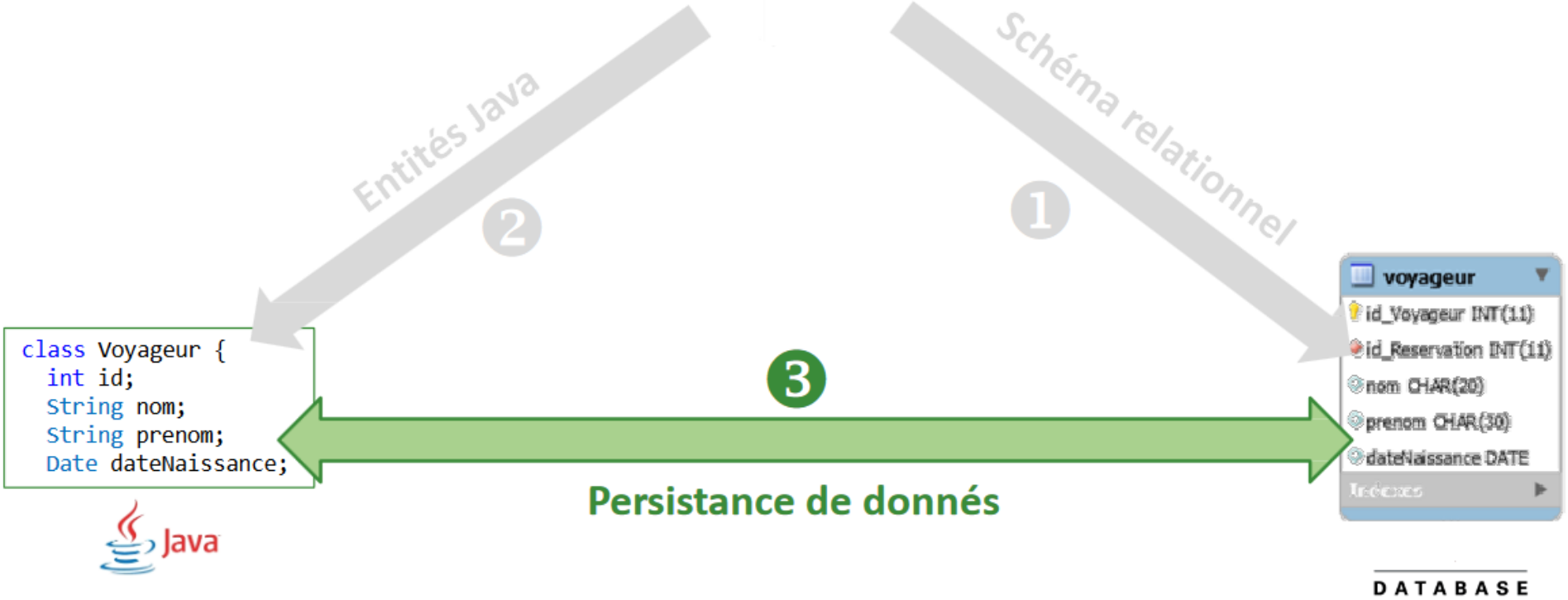
5 Hibernate DDL

6 Associations

7 Langage JPQL

Introduction : persistance de données?

- **Mapping Objet/Relationnelle**



Jakarta Persistence API (1)

- L'API Java Persistence (**JPA**) est une approche possible de l'**ORM (Object-Relational Mapping)**.

- **JPA** est une abstraction de la couche **JDBC,** les classes et les annotations **JPA** sont dans le package **javax.persistance (jakarta.persistence).**

- **JPA 1.0** a été introduite dans la spécification **JAVA EE 5,** la spécification **Jakarta EE 9 (eclipse)** supporte la version **JPA 3.0**

Jakarta Persistence API (2)

- **Composants:**

  – **ORM:** mécanisme de mapping **relationnel objet**
  – **L'API Entity Manager** qui gère les opérations **CRUD**
  – **JPQL :** langage de **requête orienté objet.**
  – **JTA (Jakarta Transaction API)** : Mécanisme de gestion des verrouillages et des transactions dans un environnement concurren

- Les **implémentations** populaires de JPA sont :
  - **Hibernate,**
  - **EclipseLink**
  - **Apache OpenJPA.**

Object-Relational Mapping

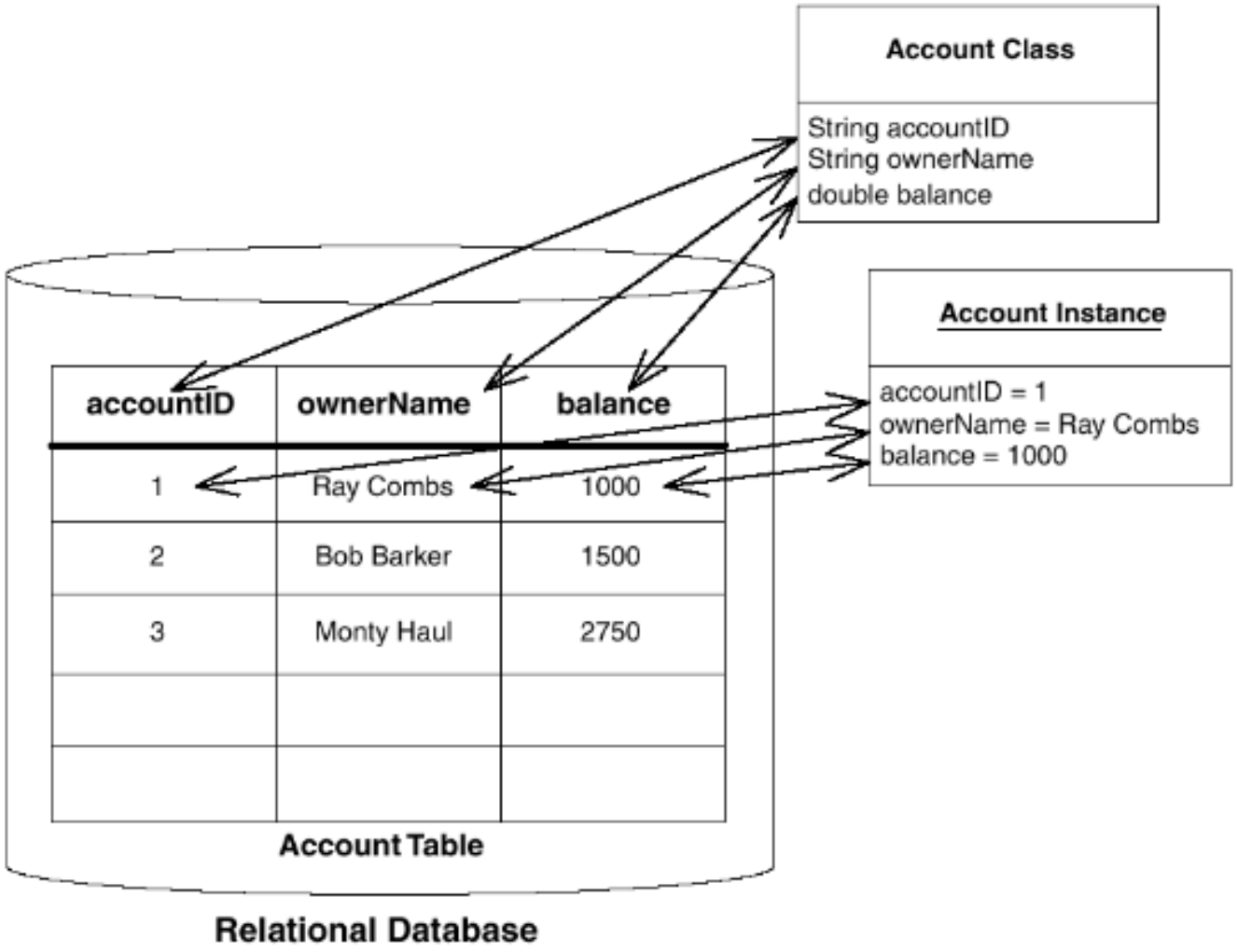
- Le processus de mappage des **objets Java** aux **tables** de base de données et vice versa est appelé « **Object-relationnal Mapping**" (**ORM**).

- On stocke l'**état** d'un **objet** dans une base de donnée.
  - Exemple :

    la *classe* **Personne** possède deux attributs `nom` et `prenom`, on associe cette classe à une **table** qui possède deux **colonnes** : **`nom`** et **`prenom`**.

- On décompose chaque **objet** en une suite de variables dont on ***stockera*** la valeur dans une ou plusieurs ***tables***.
- Permet des requêtes complexes.

Example compte bancaire



Qu'est-ce qu'un Entity Bean ? (1)

- Ce sont des **objets** qui savent **se mapper** dans une **base de donnée.**
- Ils utilisent un mécanisme de **persistance**
- Ils servent **à représenter** sous forme **d'objets** des **données** situées dans une **base de donnée**
- Le plus souvent **un objet = une** ou **plusieurs ligne(s)** dans **une** ou **plusieurs table(s)**

Qu'est-ce qu'un Entity Bean ? (2)

- Toutes les **classes d'entité** doivent :
  - définir une **clé primaire**,
  - avoir un **constructeur non arg** et/ou ne pas être autorisées à être **finales**.
  - Les **clés** peuvent être un **seul champ** ou une **combinaison** de champs.
- **JPA** permet de **générer** automatiquement la **clé primaire @Id** dans la base de données via l'annotation **@GeneratedValue**.
- Par défaut, le nom de la **table** correspond au nom de la **classe**. Vous pouvez changer cela avec l'ajout à l'annotation **@Table(name="NEWTABLENAME")**.
- **@Column**: pour personnaliser les attributs (par défaut le nom d'un champ dans la table est identique au nom de l'attribut)
- **@Transient** champ ne sera pas enregistré dans la **BD**

Exemple

```java
package spring.cours.jpa.model;

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Utilisateur {
@Id
private int id;
private String nom;

public int getId() {
return id;
}

public void setId(int id) {
this.id = id;
}

public String getNom() {
return nom;
}

public void setNom(String nom) {
this.nom = nom;
}

}
```

Propriétés d'une colonne

- @Column permet de déterminer les propriétés d'une colonne dans la table associée.
- Principales propriétés de l'annotation column :

Propriété	Valeur par défaut
name	Nom de l'attribut
nullable	True
Length (String)	255
unique	False

```
@Entity
@Table(name="users")
public class Utilisateur {
@Id
private int id;
@Column(nullable = false, name = "username", length = 60)
private String nom;
@Column(unique = true)
private String email;
// get, set
}
```

Hibernate DDL : configuration

La propriété **spring.jpa.hibernate.ddl-auto** définit le mode de génération de la base de données par Hibernate, valeurs:

– **create**

– **create-drop**: à chaque démarrage la base de données est créée puis supprimée.

– **update** : mise du schéma de la base de données (si c'est possibe)

– v**alidate** : valeur par défaut (à utiliser en production).

Mapping des Associations

La spécification **JPA** supporte trois types d'association :

@One to one

@Many to one / @One to Many

@Many to Many

Les trois types d'association peuvent être **unidirectionnelles** (navigables dans un sens) ou **bidirectionnelles** (navigables dans les deux sens), la navigabilité impacte l'utilisation des classes dans le programme et non pas les tables générées dans la base de données.

Association: @ManyToOne

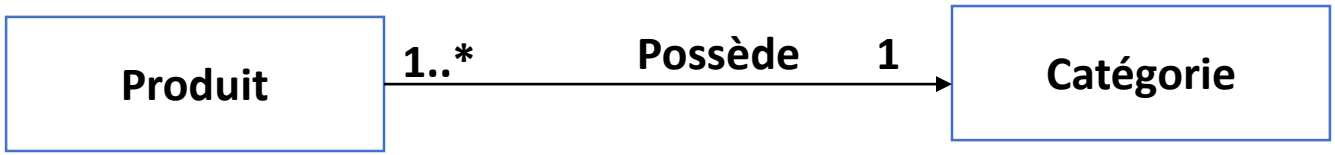
L'association **ManyToOne** est la plus utilisée dans la pratique, dans la classe **Produit** on ajoutera un attribut de type **Categorie**.

Categorie

```
@Entity

public class Categorie {

            @Id

            @GeneratedValue(strategy =
GenerationType.IDENTITY)

            private long catId;

            private String libelle;

…
```

Produit

```
@Entity

public class Produit {

@Id
@GeneratedValue(strategy=GenerationType
.IDENTITY)

private long reference;

private String designation;

private double prix;

@ManyToOne

private Categorie categorie;
```



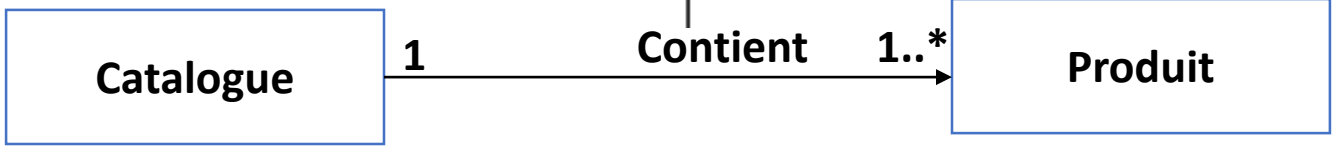
Navigabilité : Catalogue → Produit

pour généré une ***clé étrangère*** dans la table **Produit**, il faut ajouter l'annotation **JoinColumn** pour définir le nom de la clé étrangère dans la table produit :

Catalogue

```
@Entity
public class Catalogue {
            @Id
            @GeneratedValue(strategy =
GenerationType.IDENTITY)
            private long id;
            private String titre;
            @OneToMany
            private List<Produit>
produits=new ArrayList<>();
…}
```

```
@OneToMany
@JoinColumn(name="catalogue_id")
private List<Produit> produits=new ArrayList<>();
```



Association: @ManytoMany

- la navigabilité **Produit → Magasin**

@ManyToMany(mappedBy = "produits")

private List<Magasin> magasins=new ArrayList<>();

Ce qui permettra d'accéder aux magasins à partir d'un produit
p.getMagasins()

Langage JPQL

JPQL (Java Persistence Query Language) est un langage d'interrogation orienté objet similaire au langage SQL,
Structure d'une requête

Exemple de requêtes:

- SELECT li From Livre li
- SELECT li From Livre li Where li.titre='UML'
- SELECT li.titre, li.Editeur from livre li
- SELECT c.Adresse.pays from client c

Syntaxe d'une requête:

- SELECT
- FROM
- [WHERE]
- [ORDER BY]
- [GROUP BY]
- [HAVING]

Sélection selon une condition (JPA 2.0)

- SELECT CASE l.editeur WHEN 'Eyrolles"
- THEN l.prix * 0.5
- ELSE l.prix * 0.8
- END
- FROM Livre l

SQL	JPQL
SELECT * FROM produit WHERE prix > 100 ORDER BY designation;	**SELECT p** **FROM** Produit p **WHERE p.prix** > 100 **ORDER BY** p.designation

Plan : Microservices

1 Evolution des Applications

2 Concepts de Microservices

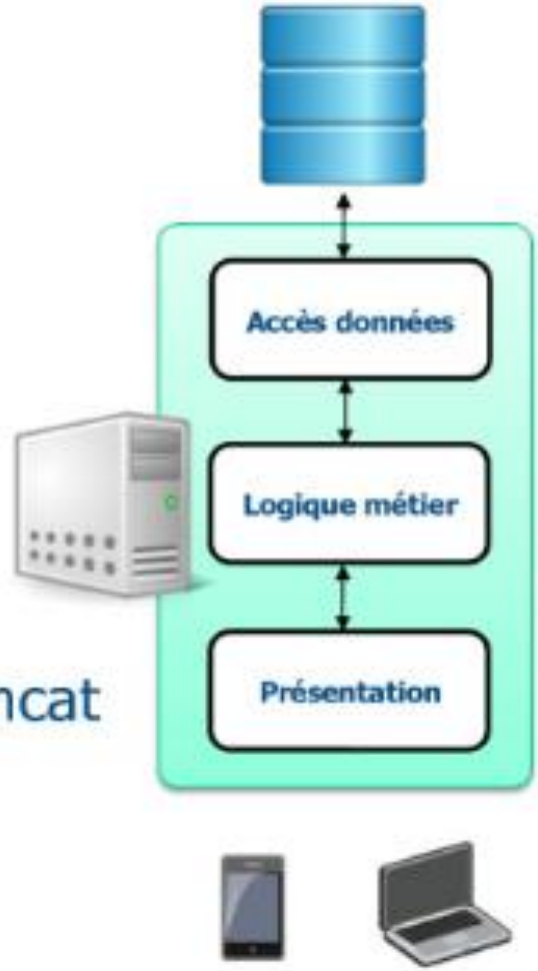
3 Développer avec les microservices

4 Eudes de cas : Système de gestion de projets & E-Commerce

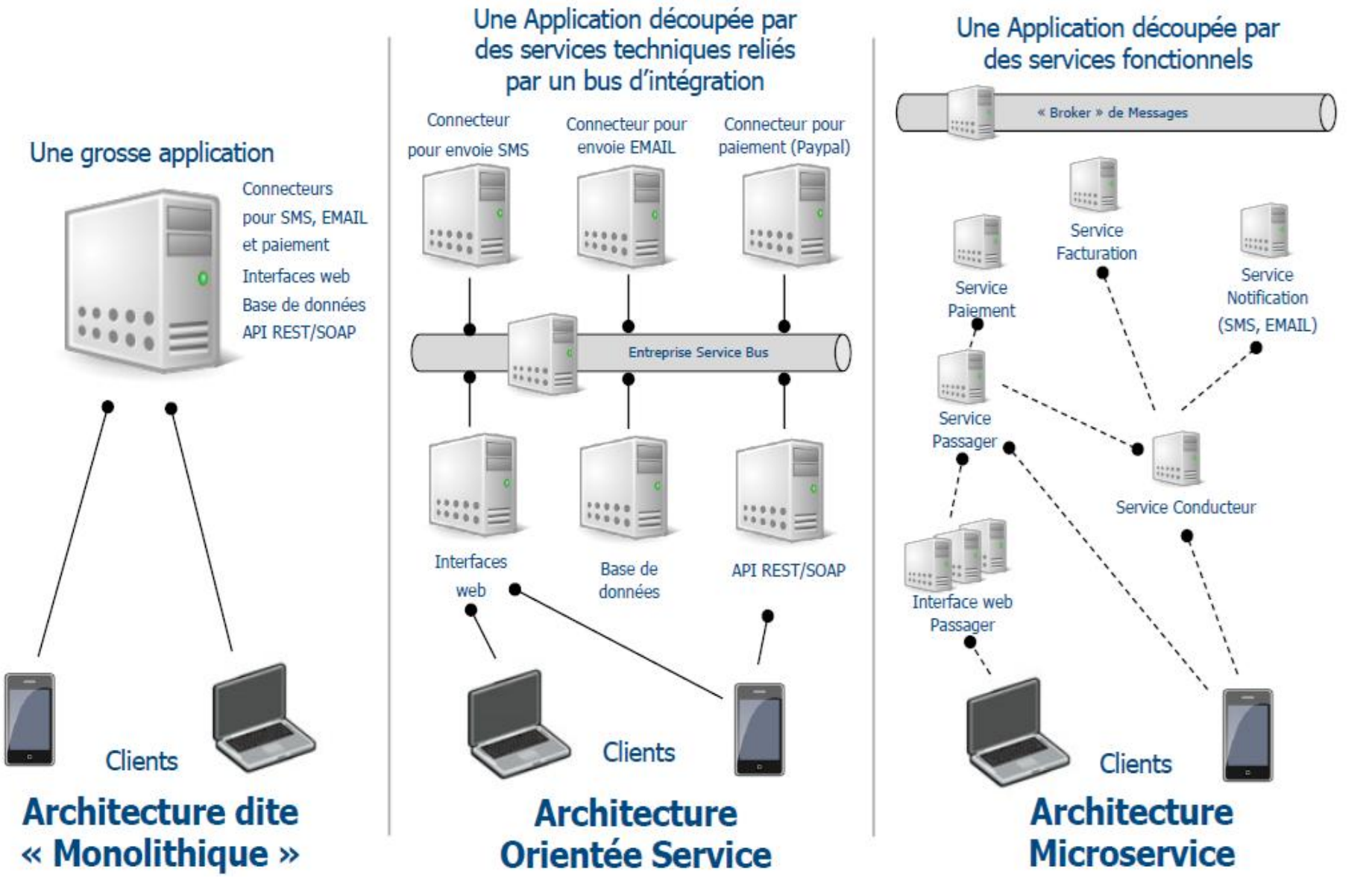
5 Avantages et Challenges

Caractéristiques des Applications Monolithiques:

- Un gros code contenant toutes les fonctionnalités et les différentes couches logicielles
- Une seule grosse compilation et un seul livrable (un gros fichier WAR)
- Une seule pile logicielle (Linux, JVM, Tomcat et bibliothèques tierces)



Evolution des applications (2)

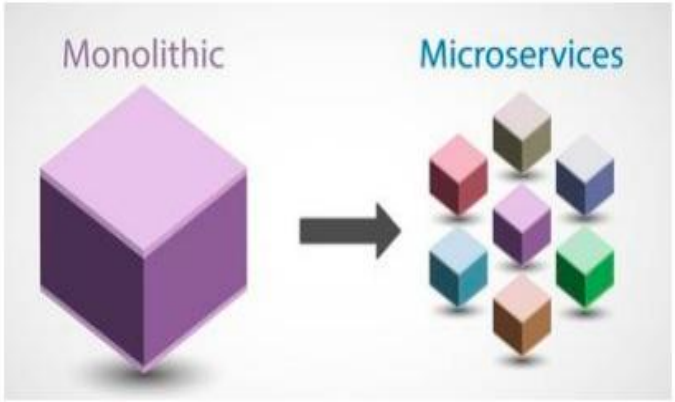


Evolution des applications (3): Historique

	Année 80 - 90	Années 2000 - 2010	Année 2010 à nos jours
Méthode de développement	Waterfull (cycle en V)	Agile	DevSecOps / SRE
Architecture applicative	Client / serveur	N-Tiers SOA / Webservices Monolithe	Micro services Cloud Native 12 Factor App
Packaging et déploiement	Clients lourds Serveurs physiques	Serveurs Virtuels (VM) Serveurs physiques	Conteneurs Serverless
Infrastructure	Datacenter Terminaux des utilisateurs	Datacenters privés IaaS	PaaS Kubernetes Services managés des cloud providers

Concepts de Microservices (1)

Microservices ?



1 **Architecture de développement logiciels**

2 **Application est décomposée en plusieurs petite services indépendantes et autonomes.**

3 **Offrent la possibilité aux développeurs de choisir et de combiner différentes technologies :**

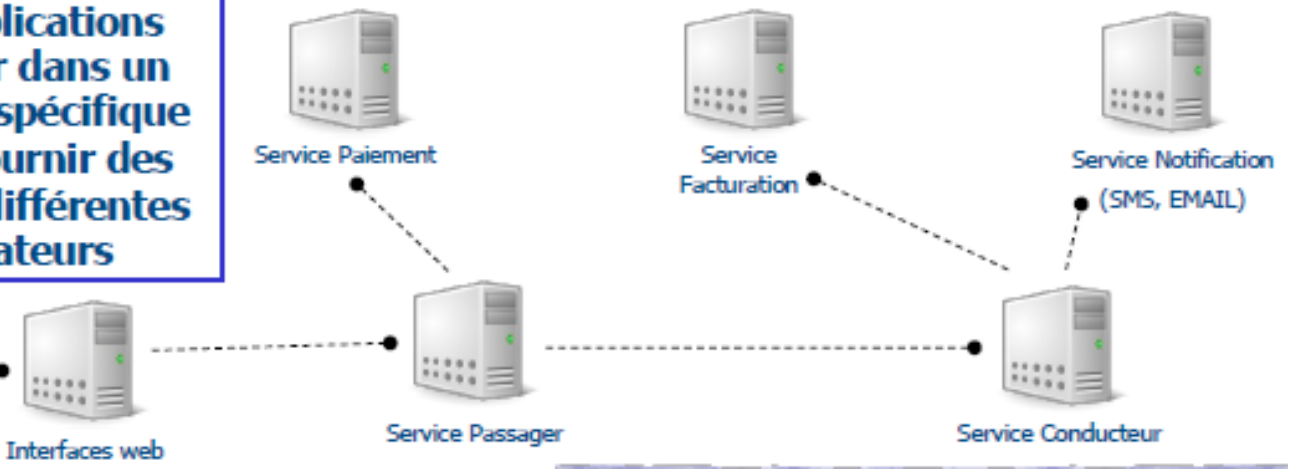
- **les langages de programmation,**
- **les bases de données ou**
- **les protocoles de communication entre les Microservices.**

Concepts Microservices (3)

Une seule fonctionnalité

- Un microservice doit réaliser une seule fonctionnalité de l'application globale
- Un microservice peut contenir toutes les couches logicielles (IHM, middleware et base de données)
- Un microservice possède un contexte d'exécution séparé des autres (exemple : machine virtuelle ou conteneur)

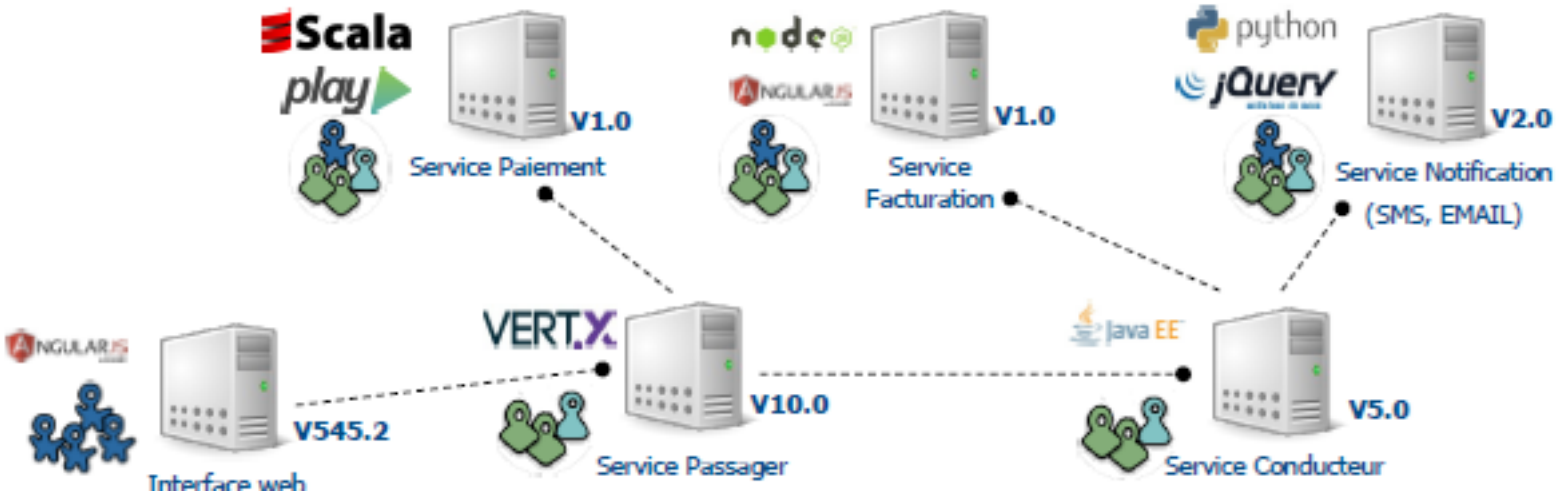
Pour les applications web, séparer dans un microservice spécifique permet de fournir des expériences différentes aux utilisateurs



Concepts de Microservices (4)

Déploiement ciblé

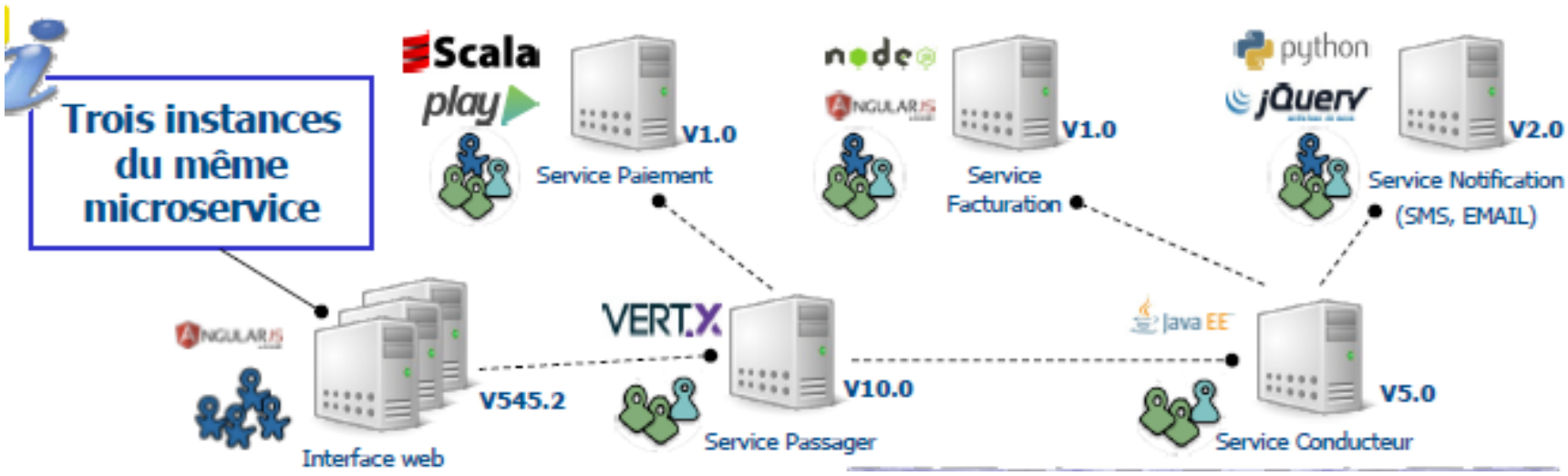
- Evolution d'une certaine partie sans tout redéployer
- Un seul livrable à partir d'un seul code source
- Moins de coordination entre équipe quand il y a un seul déploiement
  - Plus souvent
  - Moins de risque
  - Plus rapide



Concepts de Microservices (5)

Montée en charge / scalabilité

- Forte sollicitation sur un microservice ?
  - page web : Amazon en période de « Black Friday »
  - batch : compression de vidéos
- Comme chaque fonctionnalité est isolée possibilité de multiplier le nombre d'instances d'un microservice

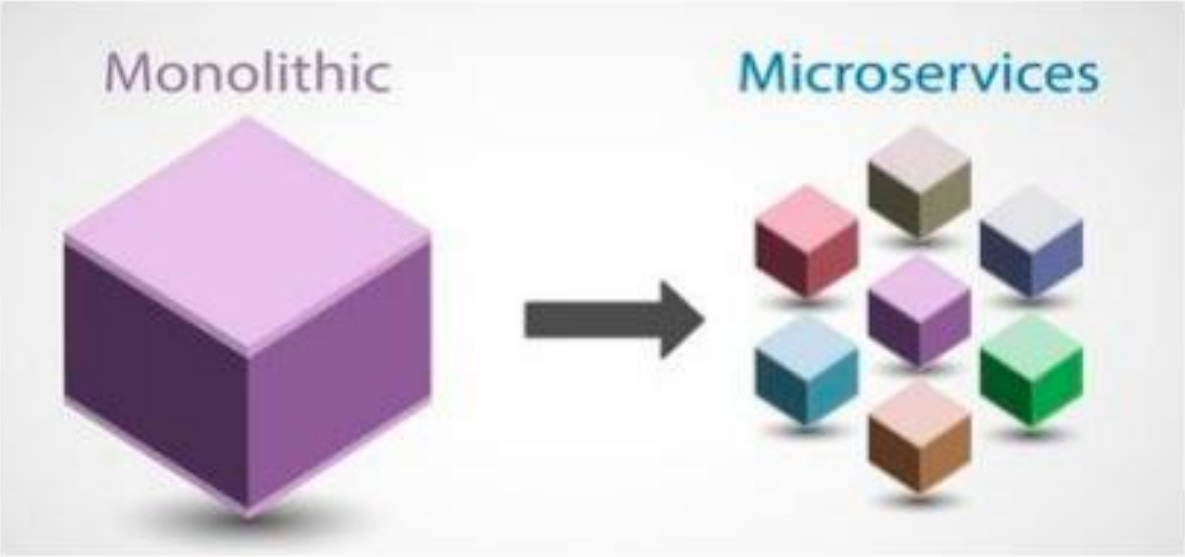


Développer avec les microservices (1)

Les entreprises ont commencé l'utilisation des Microservices à travers :

1 - La migration de leurs systèmes existants (*monolithique*) vers une architecture basée sur les Micro-services

2- Développement des nouvelles applications à base de Microservices



Développer avec les microservices (2)

Qui utilisent les microservices ?

- Uber : *https://eng.uber.com/service-oriented-architecture*

- Netflix : *https://netflixtechblog.com*

- Amazon : *https://aws.amazon.com/fr/microservices*

- Sound Cloud : *https://developers.soundcloud.com/blog/building-products-at-soundcloud-part-2-breaking-the-monolith*

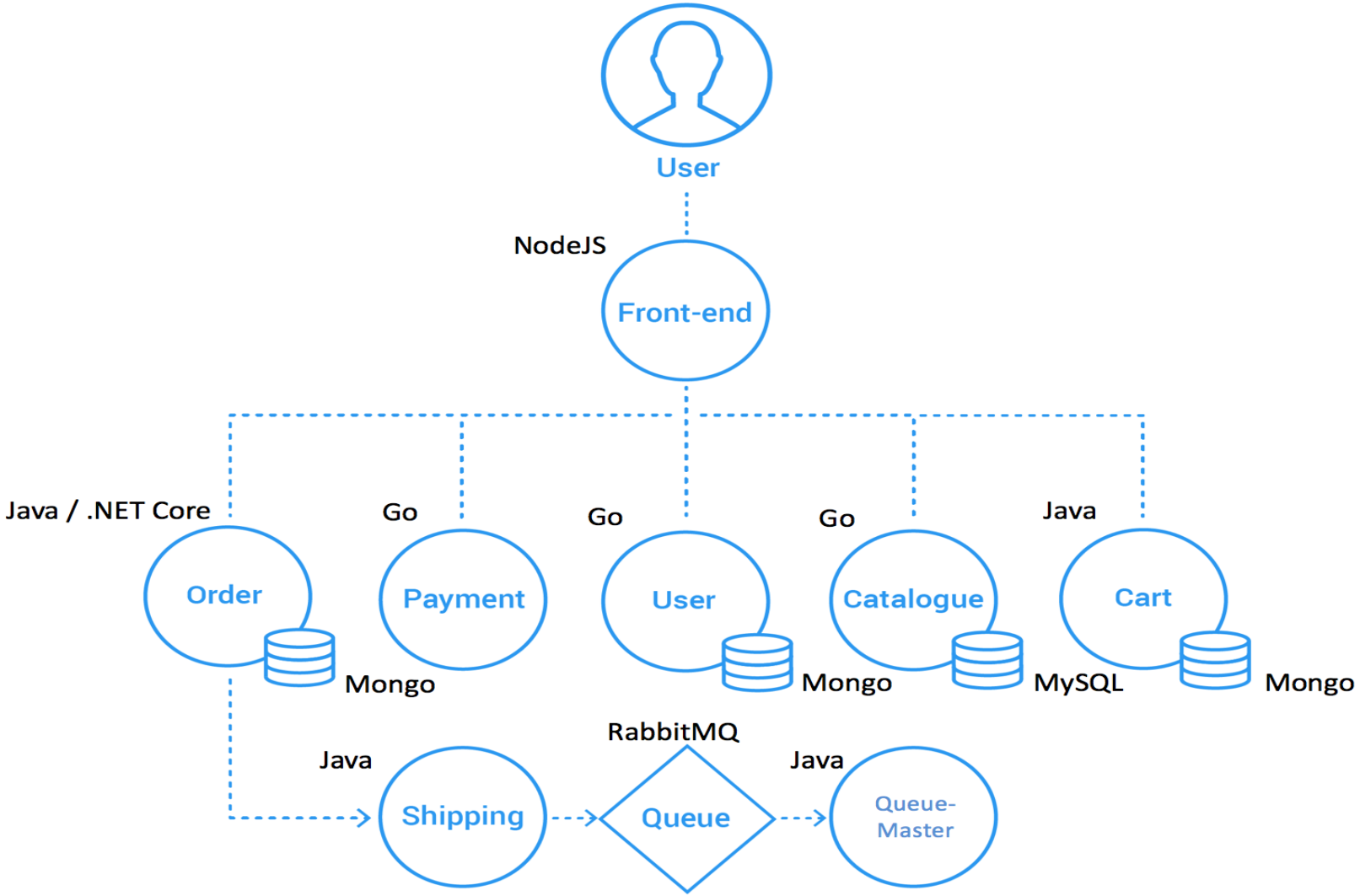
- Le prochain Eclipse Che IDE

- Offres d'emploi (tendances à la hausse)
  - *https://www.linkedin.com/jobs/search?keywords=Micro+Services&location=France*

Développer avec les microservices (3)

- Savoir **Isoler** un microservice
- Savoir **Coder** le contenu du microservice
- Savoir faire **Communiquer** des microservices
- Savoir **Composer** les microservices
- Savoir **Répartir** les charges

Eude de cas : E-commerce (Sock Shop App)



Eude de cas : Système de gestion de projets avec Microservices

- Les **projets** de développement :
  - Construction d'infrastructures,
  - Ponts,
  - Routes,
  - Tunnels,
  - Etc…
- Ces Projets contribuent à :
  - l'amélioration de la qualité de vie des populations
  - Et au développement des pays.
- Ces projets couvrent plusieurs **secteurs** :
  - Défense
  - Transport,
  - Agriculture,
  - Santé,
  - Education,
  - Etc…

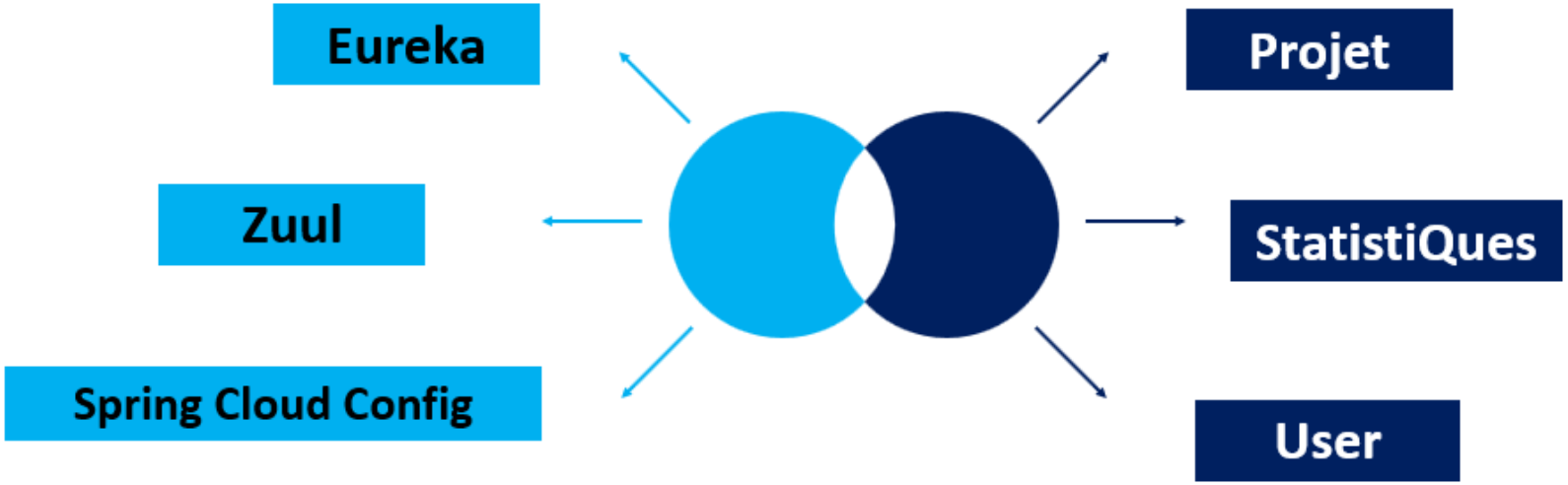
Eude de cas : Système de gestion de projets avec Microservices

Dans ce système, un utilisateur peut être :

- **Un administrateur** peut
  - S'authentifier,
  - Gérer des utilisateurs,
  - Gérer des projets,
  - Afficher des statistiques
  - Rechercher des projets.

- **Un opérateur** peut
  - S'authentifier,
  - Gérer ses projets,
  - Rechercher des projets.

Notre système est composé de **6 microservices:**



Avantages des Microservices

01 Réduction des efforts de maintenance

02 Disponibilité de services

03 Gestion optimisée de la Scalabilité

Challenges des Microservices

- **Environnements complexes et très distribués,**
- **Limite des méthodes de monitoring classiques**
- **Montée en compétence du métier, des développeurs et des opérations**
- **Couts de transformation**
- **Changement de culture d'entreprise: On passe au Cloud computing**

Bibliographie

1- Veebirakenduste loomine - Spring Kristjan Talvar, Nortal AS

2- **BOULAHIA Redha, Programmation web avancée et mobile (11812417) - MAILLOT Claudia (11507893)**

3- site web: https://angular.io/guide/file-structure

4- https://fr.slideshare.net/adieng/introduction-angular-250364526

5- https://koor.fr/Java/TutorialJEE/jee_jpa_introduction.wp

ηττπσ://σλιδεπλαψερ.χομ/σλιδε/17612235/#γοογλε_ϖιγνεττε