


TP application gestion d'équipes

L'objectif de ce TP est de développer une application en `fullstack` qui permet de gérer des équipes de Football 

Une partie backend/api en Spring boot permettra de gérer les équipes avec une base de données `in memory H2` et d'exposer des `endPoints` en REST.
Spring data JPA et Hibernate seront utilisés pour la partie accès aux données

Une partie frontend permettra de gérer les équipes via le framework Angular 11.
NodeJs, TypeScript, RxJs, Material... seront utilisés avec Angular

L'application va permettre d'afficher une liste d'équipes

D'afficher le détail d'une équipe

D'ajouter une nouvelle équipe en base de données via un formulaire Web

De supprimer une équipe existante

Et en dernière partie d'afficher et ajouter une liste de joueurs pour une équipe

Même si nous apprécions particulièrement les architectures `hexagonal` et le `DDD` : `Domain Driven Design`, le choix pour ce TP a été de ne pas aborder ces concepts et s'est porté sur une architecture en couche classique avec une séparation `controller`, `service` et `repository` (pourra faire l'objet d'un prochain cours ou TP).



Partie backend/API

Installer l'open JDK 15 selon l'OS sur lequel vous travaillez (Windows, Linux, Mac)

<https://jdk.java.net/15/>

Certains tuto expliquent comment faire selon l'OS :

- Windows : <https://java.tutorials24x7.com/blog/how-to-install-openjdk-15-on-windows-10>
- Linux/Ubuntu : <https://techoral.com/blog/java/install-openjdk-15-ubuntu.html>

Les variables d'environnement pour Java ne sont pas obligatoires pour ce TP l'application Java Spring sera lancée par l'IDE IntelliJ Idea.



Créer la structure du projet Spring Boot

Aller sur le lien suivant : <https://start.spring.io/>

Dans cette interface Spring Initializr nous allons pouvoir initialiser une structure de projet avec les dépendances souhaitées.

Spring Boot est initialisé avec un outil de build Maven ou Gradle, dans le cadre de ce TP c'est Maven qui a été choisi.

Maven reste à ce jour l'outil de build le plus utilisé de l'écosystème Java.

Spring peut également être initialisé avec plusieurs langages, Java, Kotlin et Groovy.

Dans le cadre de ce TP c'est Java qui a été choisi avec la version 15.

Project

☒ Maven Project ☐ Gradle Project

Language

☒ Java ☐ Kotlin ☐ Groovy

Spring Boot

☐ 2.5.0 (SNAPSHOT) ☐ 2.4.2 (SNAPSHOT) ☒ 2.4.1 ☐ 2.3.8 (SNAPSHOT) ☐ 2.3.7

Project Metadata

Group

com.example

Artifact

demo

Name

demo

Description

Demo project for Spring Boot

Package name

com.example.demo

Packaging

☒ Jar ☐ War

Java

☒ 15 ☐ 11 ☐ 8

Dependencies

ADD DEPENDENCIES... CTRL + B

No dependency selected

Dans la partie de droite, plusieurs dépendances peuvent être ajoutées au projet avec un champ d'autocomplétion. Les dépendances suivantes doivent être ajoutées :

- Spring web
- Spring data JPA
- H2

Dependencies

ADD DEPENDENCIES... CTRL + B

Spring Web

WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data JPA

SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

H2 Database

SQL

Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

Dans le bloc Project Metadata, mettez les configurations suivantes :

- Group : fr.sdv.cnit.university
- Artifact : tp-teams-handling-api
- Name : tp-teams-handling-api
- Description : TP for the SDV that handles the api part

- Package name : fr.sdv.cnit.university.api
- Packaging : jar
- Java : 15

Cliquez ensuite sur le bouton "generate".

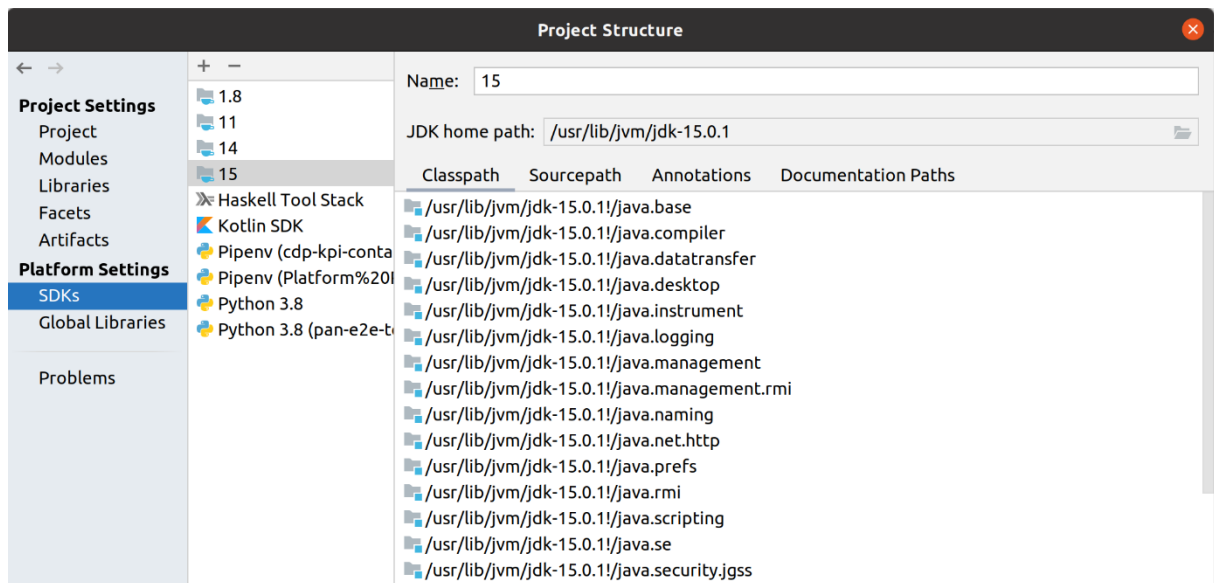


Lancer l'IDE

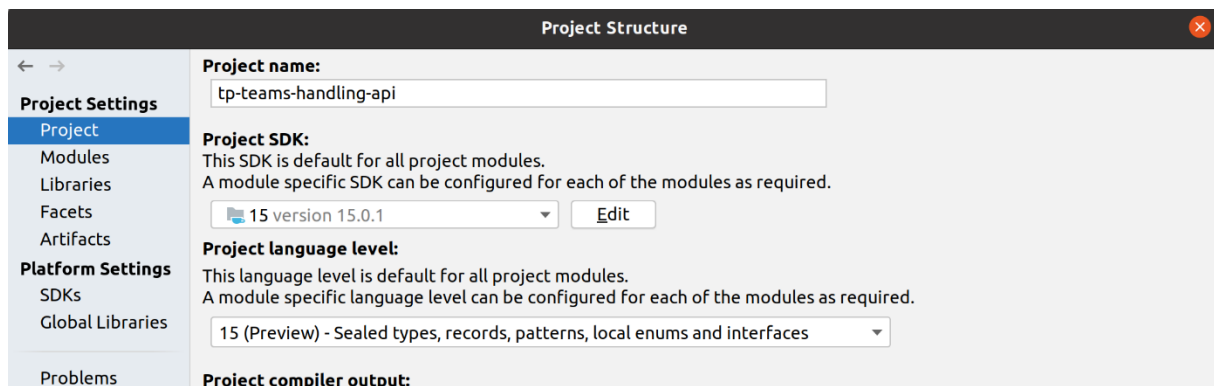
Dans le cadre de ce TP nous préconisons l'utilisons de l'IDE **IntelliJ Idea**. Cet IDE est gratuit pour les étudiants et il est surement le plus populaire dans l'écosystème Java.

Ce qui fait sa force est qu'il propose également des supports très complets dans beaucoup de langages, dont ceux utilisés dans la partie front (Angular, ES6, Typescript, CSS, SCSS...)

- Lancer l'IDE
- Utiliser l'option "Create project from existing source"
- Dans la fenêtre qui s'ouvre, sélectionnez le fichier `pom.xml` du projet Spring boot généré précédemment. Ce fichier pom est à la racine du projet.
- IntelliJ va initialiser le projet et grâce au fichier `pom.xml`, va automatiquement détecter qu'il s'agit d'un projet Maven. L'IDE propose également beaucoup de supports pour la partie Spring.
- Sélectionner ensuite l'option `File/Project Structure`
- Sélectionner `Platform settings/SDK` à gauche
- Cliquer ensuite sur le bouton +
- Un explorateur de fichier va s'ouvrir, aller vers le chemin contenant le répertoire d'install du JDK (celui qui a été dézippé suite au téléchargement)
- Si cela a bien fonctionné, vous verrez un 15 dans la liste comme sur l'image ci dessous.



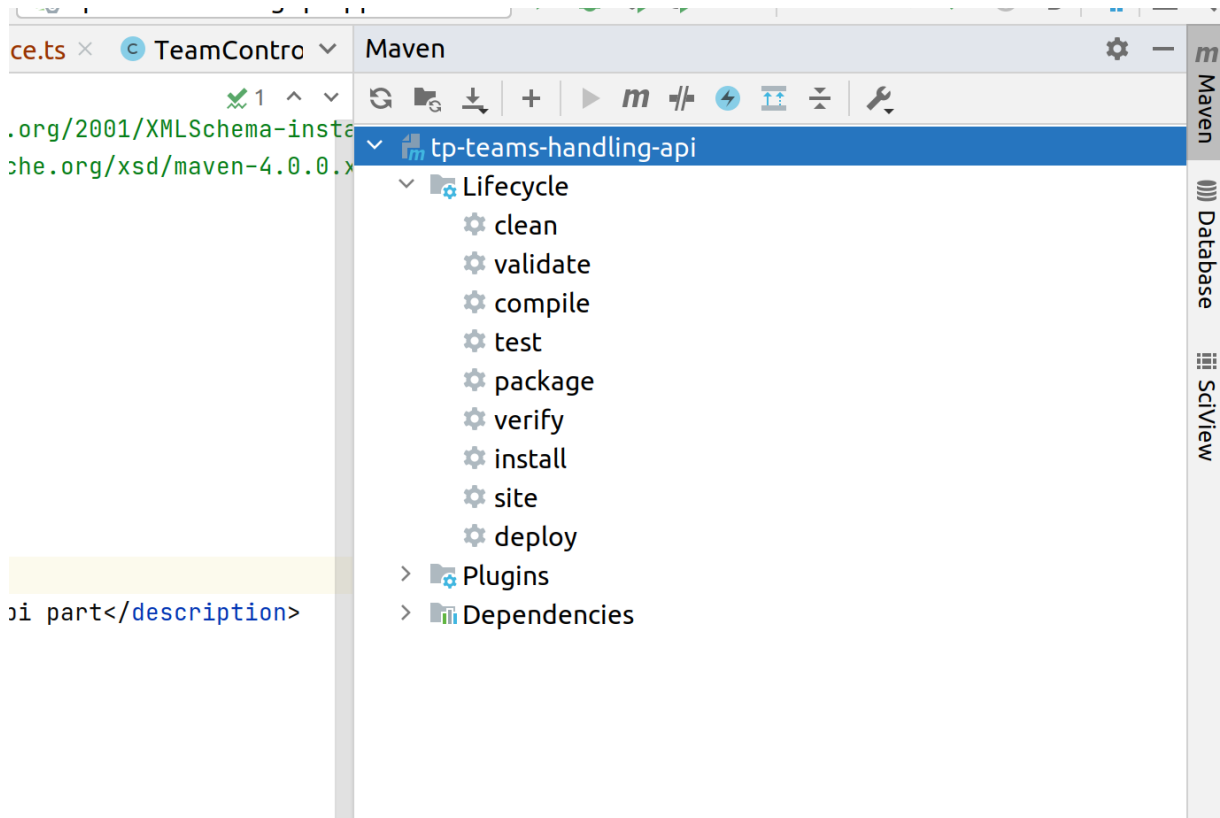
- Aller ensuite dans Project settings/Project
- Choisir le SDK 15 ajouté précédemment grâce à la liste déroulante proposée
- Dans project language level, sélectionnez Java 15



Installer et compiler le projet avec Maven

IntelliJ propose une version de **Maven** intégré embedded, donc dans le cadre de cet exercice, pas besoin d'installer Maven en global sur la machine.

L'IDE propose une fenêtre à droite avec tout le cycle de vie d'un projet Maven Lifecycle :



- Double cliquer sur `clean`, ceci videra le répertoire de travail `target` utilisé par Maven
- Double cliquer sur `install`, ceci va installer toutes les dépendances proposées dans le fichier `pom.xml`, compiler le projet Java et exécuter les tests existants dans le projet.

Si tout se passe bien, la console dans l'IDE affichera un SUCCESS :

```
[INFO]
[INFO]
[INFO] --- maven-jar-plugin:3.2.0:jar (default-jar) @ tp-teams-handling-api ---
[INFO] Building jar: /home/mazlum/my-projects/custom/tp-teams-handling/tp-teams-handling-api/t
[INFO]
[INFO] --- spring-boot-maven-plugin:2.4.1:repackage (repackage) @ tp-teams-handling-api ---
[INFO] Replacing main artifact with repackaged archive
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ tp-teams-handling-api ---
[INFO] Installing /home/mazlum/my-projects/custom/tp-teams-handling/tp-teams-handling-api/targ
.m2/repository/org/tosunsi/tp-teams-handling-api/0.0.1-SNAPSHOT/tp-teams-handling-api-0.0.1-
[INFO] Installing /home/mazlum/my-projects/custom/tp-teams-handling/tp-teams-handling-api/pom.
.1-SNAPSHOT/tp-teams-handling-api-0.0.1-SNAPSHOT.pom
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 9.783 s
[INFO] Finished at: 2021-01-11T23:23:33+01:00
[INFO] -----
```

Si ce n'est pas le cas, vérifier si le bloc suivant existe dans le fichier pom.xml, dans la section plugins :

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <configuration>
    <argLine>--enable-preview</argLine>
  </configuration>
</plugin>
```

Et y copier/coller ce bloc.

En effet Java 15 propose des features en cours de validation enable-preview et lors des tests unitaire, Maven utilise le plugin maven-surefire-plugin.

Parfois il y a besoin de configurer ça dans le plugin Maven surefire.

Félicitation, vous venez de compiler et installer pour la première fois votre projet

backend/api Spring Boot 🙌



Ajout du premier webservice REST

Dans cette partie nous allons ajouter la classe qui va contenir notre premier **webservice REST**.


Ce webservice va juste afficher "Hello World" sur une page web.

- Dans le package fr.sdv.cnit.university.api, ajouter un package controller
- Y ajouter une nouvelle class Java appelée TeamController
- Cette classe contient une annotation Spring @RestController pour indiquer qu'il s'agit d'un controller pour exposer du Rest
- Elle contient une méthode et le type de webservice, dans cet exemple un @Get est utilisé car nous souhaitons récupérer une resource
- L'URL associée à la resource est hello dans cet exemple

```
@RestController
public class TeamController {

    @GetMapping("/hello")
    public String getTeams() {
        return "Hello World";
    }
}
```

- Chaque projet Spring boot contient un main avec un serveur embarqué. Tomcat est utilisé par défaut pour du Spring Web (Jetty peut aussi être utilisé)
- Ouvrir la classe TpTeamsHandlingApiApplication à la racine du package api

- Lancer le main via IntelliJ (bouton lecture en vert sur la classe ou la méthode main)
- Si tout se passe bien, Spring n'affichera pas d'erreur dans la console
-  spring utilise par défaut le port 8080 dans son serveur embarqué, si ce port est occupé par un autre process sur votre machine, il va falloir soit arrêter le process en question, ou soit changer le port par défaut utilisé par Spring boot, via le fichier application.properties ou application.yml
- ouvrir un navigateur web et afficher sur la colonne l'URL suivante : `http://localhost:8080/hello`
- Normalement le text `hello` devrait s'afficher

Félicitations, vous venez de développer votre premier webservice Rest et d'y afficher le résultat.



Configuration de la base embarquée H2

Dans le répertoire `src/main/resource` ajouter 2 fichiers :

- **schema.sql** : qui va permettre de gérer la structure de notre modèle de donnée

```
CREATE TABLE team
(
    id IDENTITY NOT NULL PRIMARY KEY,
    name VARCHAR(200),
    slogan VARCHAR(500)
);
```

- **data.sql** : qui va permettre d'ingérer de la donnée dans la base

```
INSERT INTO team (name, slogan) VALUES ('PSG', 'Revons plus grand');
INSERT INTO team (name, slogan) VALUES ('Real Madrid', 'Les galactiques');
INSERT INTO team (name, slogan) VALUES ('Barcelone', 'La Macia');
INSERT INTO team (name, slogan) VALUES ('Bayern', 'Les puissants en
Allemagne');
INSERT INTO team (name, slogan) VALUES ('Manchester United', 'Les red
devils');
```

- Ajouter ensuite de la configuration pour le modèle de données H2, dans le fichier `application.properties` proposé par défaut dans Spring boot :

```
spring.h2.console.enabled=true
```

Cette configuration permet de pouvoir afficher la console Web H2, pour y voir le modèle de données.

- Gérer les paramètres d'accès à la base de données :

```
spring.datasource.username=sa  
spring.datasource.url=jdbc:h2:mem:testdb  
spring.datasource.password=  
spring.datasource.dbname=testdb
```

Ces paramètres permettent de configurer les éléments nécessaires pour accéder à la base de données.

- Configuration pour l'ORM Hibernate :

```
spring.jpa.show-sql=true  
spring.jpa.hibernate.ddl-auto=validate
```

Le premier paramètre `spring.jpa.show-sql` permet d'afficher dans la console les requêtes SQL lancées par Hibernate.

Le second paramètre `spring.jpa.hibernate.ddl-auto` permet d'indiquer à Hibernate de valider que les types des objets `entity`, soient cohérents avec les types de colonnes des tables associées.

Ainsi le mapping objet relationnel est cohérent car une validation est effectuée.

Par défaut Hibernate utilise un mode `create`, qui permet de créer et modifier le model de données à partir des objets `entity`.

Or dans notre cas nous souhaitons déléguer cette tâche à H2.

- Relancer l'application Spring boot (la classe avec la méthode `main`) afin de prendre en compte les nouvelles configurations.
- Lancer la console H2 depuis un navigateur en tapant l'url suivante :

```
http://localhost:8080/h2-console
```

Saisir les paramètres d'accès à la base, ajoutés précédemment dans le fichier de configuration.

English ▼

[Preferences](#) [Tools](#) [Help](#)

Login

Saved Settings: Generic H2 (Embedded) ▼

Setting Name: Generic H2 (Embedded)

Save Remove

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:testdb

User Name: sa

Password:

Connect

Test Connection

Cliquer ensuite sur **Connect**, vous verrez ensuite la table `team` avec les données et vous pourrez y exécuter des requêtes SQL



Développement de la partie accès aux données

- A la racine du package `api`, ajouter un package `entity` et y ajouter une classe `TeamEntity`. Implémenter le contenu de cette classe avec le mapping objet relationnel. Cet objet doit être associé à la table `team`
- A la racine du package `api`, ajouter un package `repository` et y ajouter une interface `TeamRepository` qui étend l'interface `JpaRepository` proposée par Spring Data Jpa. Cette interface propose des méthodes par défaut pour des traitements autour de la donnée.
- A la racine du package `api`, ajouter un package `service` et y ajouter une classe `TeamService`
- Injecter l'interface `TeamRepository` dans la classe `TeamService`, grâce au support d'injection de dépendance proposé par Spring. Nous utiliserons

l'injection par constructeur, car c'est une pratique recommandée (mockable facilement et évite les dépendances cycliques)

- Dans la classe service, implémenter les méthodes suivantes :
 - Récupérer toutes les équipes en base de données
 - Récupérer une équipe en base à partir de son ID
 - Insérer une équipe en base
 - Modifier une équipe en base
 - Supprimer une équipe existante de la base de données à partir de son ID

Ne pas hésiter à s'aider des documentations sur internet, Spring data Jpa, Hibernate...



Implémenter des tests d'intégration pour la classe TeamService

- Dans le package src/test/java à la racine du package api, ajouter une classe TeamServiceTest
- Dans cette classe, faire un test "passant" (sans erreur) pour chacune des méthodes du service
- Pour la convention de nommage utilisées, les développeurs sont libres, ils peuvent par exemple utiliser une convention should ou une convention en given when then

```
@Test
public void shouldReturnTheExpectedTeam_whenGetTeamByExistingId(){
    ....
}

@Test
public void givenExistingId_whenGetTeamById_thenExpectedTeamInResult(){
    // Given.

    // When.

    // Then.
}
```

Pour les assertions et les tests, le framework JUnit en version 5 est utilisé. Un framework très populaire dans l'écosystème Java, appelé AssertJ est directement apporté par Spring Boot. Dans le cadre de cet exercice, nous souhaitons que les assertions se fasse avec **AssertJ**, en voici un exemple :

```
import static org.assertj.core.api.Assertions.assertThat;
```

```

@Test
public void given...(){
    // Given.

    // When.
    String result = ...

    // Then.
    assertThat(result).isNotNull().isNotEmpty()
}

```

AssertJ permet de faire des assertions en mode `fluent` ainsi qu'une api très fournie pour la partie testing.



Appeler les services depuis la classe TeamController

- Modifier la classe `TeamController` et y ajouter les webservice suivants :
 - Récupérer la liste des équipes
 - Récupérer une équipe à partir de son ID
 - Créer une équipe
 - Modifier une équipe
 - Supprimer une équipe existante
- Aidez-vous de la doc de Spring pour Spring Web et Spring MVC Rest.
- Aidez-vous également de la documentation sur les bonnes pratiques de Rest, pour utiliser le bon verbe en fonction du webservice et de la bonne URL.
- Dans le cadre de cet exercice, nous ne souhaitons pas directement exposer l'objet `TeamEntity` aux clients de notre api, mais un objet `TeamDto`. Si notre api évolue, ceci nous permet de contrôler exactement ce que nous exposons.
- L'objet `TeamDto` peut être un pojo classique (avec getters/setters) ou une `Record` qui est un concept encore en preview dans Java 15. L'avantage des `Records` est que le `boilerplate code` (code standard) n'a pas besoin d'être écrit.



Tester unitairement les webservices Rest

Après avoir implémenté les webservices Rest dans la class `TeamController`, nous souhaitons les tester avec du code pour garantir leur bon fonctionnement.

Précédemment des tests d'intégration ont été faits dans les traitements métiers et d'accès à la donnée.

L'objectif ici est de tester les statuts de retour de chacun des webservices, sans avoir besoin de redescendre dans les classes service et d'accès aux données.

- Créer une classe `controller/TeamControllerTest` dans le package `test` à la racine de `api`.
- Nous souhaitons pour tester nos webservices, utiliser une librairie populaire dans l'écosystème Java, appelée `Rest Assured`. Cette librairie utilise une convention en `given/when/then` (comme proposé précédemment dans la partie `test` des services). `Spring boot` propose un support natif pour `Rest Assured`, à travers la class `RestAssuredMockMvc` :

Ajouter dans le `pom` :

```
<dependency>
  <groupId>io.rest-assured</groupId>
  <artifactId>spring-mock-mvc</artifactId>
  <scope>test</scope>
</dependency>
```

Pour vous guider un peu, voici une partie de code qui doit être ajoutée dans la classe `TeamControllerTest` :

```
@BeforeEach
public void initialiseRestAssuredMockMvcStandalone() {
    RestAssuredMockMvc.standaloneSetup(teamController);
}
```

- Faire un test pour chacun des webservices de la classe `TeamController` (`Get`, `Post`, `Put`, `Delete`), en vérifiant qu'il retourne bien un statut `200`.
- Utiliser l'api `Mockito` pour mocker la classe `TeamService` et les méthodes de cette classe.
- Utiliser ensuite `Rest assured` pour faire les assertions.

Pour la suite, nous allons partir dans le principe qu'une nouvelle règle métier a été proposée par l'équipe fonctionnelle du projet.

Lors de l'ajout d'une équipe en base de données, si le champ `slogan` de la classe `team` est nul ou vide, nous souhaitons que le traitement ne se fasse pas et qu'une exception soit levée.

Pour les clients de notre api, nous souhaitons renvoyer un statut `Rest 400` (`Bad Request`) dans ce cas.

- Ajouter la nouvelle règle dans la classe `TeamService`, si le champ `slogan` est nul ou vide, alors on `throw` une exception `TeamInvalidException`. Cette classe peut être ajoutée dans un package `exception` à la racine du package `api`.
- Dans la classe `TeamController`, nous souhaitons renvoyer un statut `400` si l'exception est levée depuis la classe `service`. Trouver un moyen de renvoyer un statut `400` dans la classe `controller` sans faire de `try catch`. Utilisez la doc de `Spring` pour voir quelles sont les moyens proposés pour le faire.
- Dans la classe `TeamControllerTest`, ajouter un nouveau test pour vérifier ce cas d'erreur. Pour nous faciliter la vie, nous allons utiliser `Mockito` pour

mocker la méthode `addTeam` de la classe `TeamService`, en simulant le cas où elle lève une `TeamInvalidException`. Utiliser ensuite `Rest Assured` pour tester le `Post` et faire l'assertion sur le statut 400.



Partie frontend

Après avoir durement travaillé sur la partie backend et vérifier que tout fonctionnait correctement, nous passons à la partie front.

Nous allons utiliser le framework Angular version 11.

- Installer NodeJs : <https://nodejs.org/en/download/> (suivre les instructions selon l'OS)
- Node est installé avec le gestionnaire de paquets NPM. Ce dernier va permettre d'installer des packages en global sur la machine mais aussi dans un projet Angular, par exemple.
- Pour vérifier que node et npm ont bien été installés, lancer les commandes suivantes depuis une invite de commande (shell pour Linux et Mac et cmd pour Windows) :

```
node --version
```

```
npm --version
```

Si les installations se sont bien passées, les versions s'afficheront dans l'invite de commande.

- Installer le package Angular cli en global sur votre machine. Ce package va permettre de créer un projet Angular avec une structure initiale et une application qui fonctionne. Pour vous aider, ne pas hésiter à aller sur la doc officielle de Angular cli : <https://angular.io/cli>

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

- Créer ensuite un nouveau projet appelé tp-teams-handling-app via la commande suivante :

```
ng new tp-teams-handling-app
```

Certaines questions seront posées par le cli, pour la partie style, choisir scss.

Ne pas hésiter à aller sur la doc d'Angular : <https://angular.io/guide/setup-local>

- Lancer ensuite l'application :

```
cd tp-teams-handling-app  
ng serve --open
```

Le ng serve va lancer un serveur Node et lancer l'application Angular.

L'option --open permet d'ouvrir un navigateur automatiquement.

Vous verrez ensuite à l'adresse suivante <http://localhost:4200/>, une page affichant des éléments.



Quelque explications sur certains fichiers de configurations

- Les dépendances installées par npm sont déclarées dans le fichier `package.json` et lors de l'installation un fichier `package-lock.json` est généré et permet de garantir d'utiliser les bonnes versions
- Le fichier `angular.json` contient de la configuration pour Angular
- Le fichier `tsconfig.json` contient la config pour Typescript
- Le fichier `tslint.json` contient des règles de linting et de bonnes pratiques de code pour le code typescript



Quelque explications sur la philosophie d'Angular

Angular est un framework orienté composant, le root component de l'application est le `app.component`.

Ensuite d'autres composants peuvent être insérés dans `app.component` et ainsi de suite. Au final l'application Angular est un arbre de composant.

Les développeurs chez Google ont souhaité utiliser le standard Web Component et aujourd'hui, les autres frameworks du marché comme React et VueJs sont également orientés composant.

Angular propose également la notion de module et le component `app.component` est mis par défaut dans le module parent de l'application appelé `app.module`.

Par la suite d'autres modules pourront être créés et des composants seront rangés dans ces modules.

De nouveaux nous conseillons la doc d'Angular pour avoir plus de précisions sur les modules, composants, services, routing ou autre.

Autre aspect aussi à préciser est qu'Angular est une `single page application`, ce qui veut dire que l'application web ne contient qu'une seule page avec des `views` à l'intérieur.

Un système de `routing` permet d'aller d'une view à une autre.

Angular utilise le langage Typescript et le framework RxJs par défaut. Le premier apporte un typage et une compilation par rapport à Javascript et l'autre permet en utilisant le paradigme `Reactive programming`, par exemple, d'appeler les webservice depuis la partie api, en asynchrone et non bloquant. Ce paradigme peut permettre de récupérer de la donnée avec une notion temporelle (les éléments arrivent au fur et à mesure), et permet également de souscrire facilement à des événements (clic ou autre).

- Typescript par défaut utilise des imports avec des chemins relatifs, il est possible de changer ce comportement en utilisant des `path alias`, ce qui permet d'avoir des `sexier imports` (chemins plus élégants).
- Ouvrir le fichier de config typescript `tsconfig.json` et appliquer les modifications suivantes :

```
"compilerOptions": {
  "baseUrl": "./",
  ....

  "paths": {
    "@team-handling/*": [
      "src/app/*"
    ],
    "@team-handling-env/*": [
      "src/environments/*"
    ]
  }
}
```



Créer le module teams

- Dans le cadre de ce TP, nous souhaitons créer un module appelé `teams`. Nous allons utiliser `Angular cli` pour générer toutes nos ressources Angular. Sur votre shell, aller dans le répertoire `src/app` du projet front et taper la commande :

```
ng generate module teams
```

Le fichier `ts` pour le module est créé sous le chemin suivant `teams/teams.module.ts`



Créer un composant qui va afficher la liste des équipes

- Créer un composant dans le module `teams` avec `Angular cli`. Une des bonnes pratiques des frameworks orientés composants est de mettre les composants qui gèrent la `view` (ou un conteneur de page) dans un répertoire `containers` et les composants réutilisables dans un répertoire `components`. Il s'agit de l'architecture `smart and dumb components`



un composant dans Angular :

- Contient une partie `model` qui est gérée dans le fichier `ts`
- Une `vue` qui peut être mise dans le composant `ts` ou dans un fichier `Html` à part

- Une partie style encapsulée par défaut seulement pour le composant en cours. Le style peut directement être mis dans le ts, ou dans un fichier css ou scss séparé
- Angular cli par défaut utilise des fichiers séparés pour la vue et la partie style

Exemple :

```
RouterRouterTeamListPageComponent (smart)TeamListPageComponent (smart)TeamList
PageComponent Page containerTeamListPageComponent Page containerDumb Component
sDumb ComponentsUrl teamssmart a des interactions avec extérieur (router, serv
ice)Peut contenirdumb n'a pas d' interactions avec extérieur (router, service)
```

Les smart components ont des interactions avec l'extérieur, comme le service (pour interroger des api) et le router.

Les dumb components sont des composants de présentation.

Ils vont gérer une partie de la page en cours et pourront souvent être réutilisables, car ils ont moins d'intelligence que les smart.

Ils ont pour vocation d'être partagés car ce sont des composants dits purs car ils n'ont pas d'interactions avec l'extérieur.

- Créer un répertoire `containers` dans le répertoire `teams`. Ensuite se positionner dans le répertoire `containers` et via le cli, créer un smart component appelé `TeamListPageComponent` :

```
ng generate component team-list-page --module teams
```

L'option `--module` permet de déclarer le component dans le module souhaité, dans notre cas `teams`. Dans le code d'exemple ci-dessous, nous voyons que le composant est bien déclaré dans l'attribut `declarations` du module.

```
@NgModule({
  declarations: [
    TeamsListPageComponent
  ],
  imports: [
    CommonModule,
    ...
  ]
})
export class TeamsModule {
}
```



Initialiser la partie router

- Pour accéder à la page/view teams=>TeamsListPageComponent, comme montré dans le diagramme précédent, il faut utiliser le router fourni par défaut dans Angular et le positionner dans le fichier `app.component.html`
- Dans le fichier `app-routing.module.ts`, ajouter une nouvelle route pour pointer sur le path `teams` et le composant associé `TeamListPageComponent`. Aidez vous de la doc si besoin.
- Dans le fichier `app.component.html`, appeler le composant router via son selector (dans Angular chaque composant a un selector, ainsi le composant peut être appelé par un autre)



Créer le service teams

- Générer un service Angular `team.service` dans le module `teams`. Aidez-vous de la doc de cli pour le générer



Il est intéressant de préciser qu'Angular utilise 1'IOC au même titre que Spring.

Le composant précédent est déclaré dans le module, il est connu du cycle de vie d'Angular.

Le service team est une ressource à part mais elle contient une annotation `@injectable`.

```
@Injectable({  
  providedIn: 'root'  
})
```

Ainsi dans Angular, ce service est injectable dans les composants.

Pour injecter le service dans le composant, il suffit de le passer dans le constructeur.

Dans la partie backend et Spring nous avons choisi l'injection par constructeur car c'est la pratique recommandée.

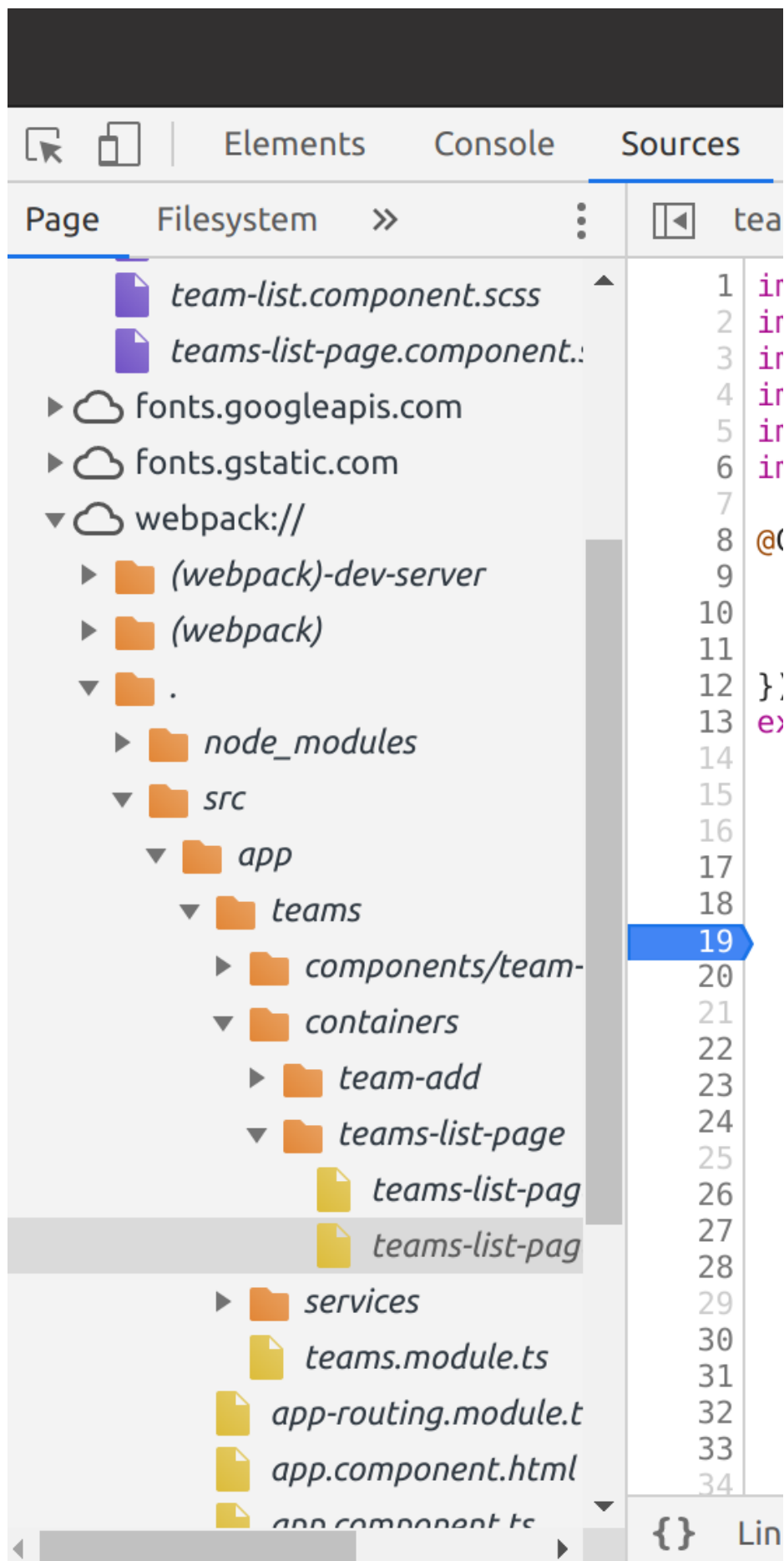
Dans la partie Angular, c'est nativement utilisé comme ça.

- Dans le module `teams`, ajouter un répertoire `models` et y crée une interface Typescript qui représente une équipe `team.ts`
- Ajouter une méthode dans la classe service qui permet de récupérer toutes les équipes, en interrogeant l'api Rest, on peut l'appeler `getTeams`. Cette méthode retournera un `Observable` de `List de Team`

- Injecter le service dans le composant et appeler la méthode `getTeams`
- Afficher la liste d'équipes dans le fichier Html du composant. Pour un premier affichage, pas besoin de mettre des styles CSS
- Si le routeur a correctement été modifié, en tapant l'url suivante : <http://localhost:4200/teams>, vous devriez voir la liste d'équipes s'afficher
- Normalement une erreur de ce type sera affichée `Access-Control-Allow-origin`. Lorsqu'un domaine A tente d'accéder à un domaine B, domaine B peut renvoyer cette erreur. Il faut dans ce cas configurer dans domaine B le fait que certaines origines peuvent y accéder. Dans notre exemple le domaine B est exposé par le serveur côté api, il faut donc ajouter une configuration pour permettre au front d'y accéder. Ajouter cette configuration côté backend/api.

Nous recommandons l'utilisation du navigateur chrome, car il propose un `devTools` intégré très complet.

On peut y voir le code HTML et Typescript par exemple et aussi debugger le code Typescript, en mettant des `breakPoints`



- Pour lancer le `devTools` dans chrome taper sur F12
- Pour debugger le code, sélectionner l'onglet `Sources` en haut à droite et dans la fenêtre `Filesystem` à gauche aller sur `webpack://` et ensuite `./src/app.....` vous y verrez tous les fichiers `Typescript` et pourrez y mettre des `breakpoints`, très utile pour debugger un problème
- Vous verrez aussi les erreurs dans l'onglet `Console`
- Si la liste des équipes s'affichent sans problème, améliorer l'affichage avec la librairie `Angular material`
- La plupart des librairies open source sont installées via `npm` dans le fichier `package.json`, mais pour certaines librairies appartenant à l'écosystème `Angular`, il est recommandé de les installer avec `Angular cli` pour assurer une compatibilité avec la version d'`Angular` utilisée sur le projet
- Tapper la commande suivante :

```
ng add @angular/material
```

Vous verrez dans le fichier `package.json` que la librairie a bien été installée.

- Utiliser des `mat card` avec `Material` pour afficher la liste des équipes
- `Angular material` étant très modulaire, pour chaque composants utilisés, il faut importer le module `Material` correspondant dans notre module actuelle
- Vous pouvez également utiliser des `flex box` pour améliorer l'affichage. N'hésitez pas à vous aider d'internet pour ça.

Aidez vous de la doc d'`Angular Material` qui est très complète.



Ajouter un composant qui va afficher la détail d'une équipe

- Dans la liste des équipes, lorsque l'on clic sur une ligne d'équipe, nous souhaitons afficher une page/container avec le détail d'une équipe. Y Afficher tous les attributs d'une équipe (name et slogan)
- Appuyez vous sur la partie précédente pour initialiser les composants correctement
- Proposer un bouton précédent pour faire un retour arrière vers la page de liste des équipes



Ajouter un composant qui va permettre de créer une nouvelle équipe

- Dans la page de liste des équipes, en bas de page proposer un bouton ajouter ou une icône avec un +
- Lors du clic sur ce bouton, une redirection sera faite vers une nouvelle page affichant un formulaire de création d'une équipe
- Proposer 2 champs : nom et slogan de type `input text`. Libre à vous d'utiliser Material pour ça ou du natif HTML
- Proposer un Validator sur les 2 champs qui sont obligatoires. Dans Angular 2 types de Validator peuvent être utilisés, directement via une directive dans le HTML ou depuis le composant. Pour cet exercice l'objectif est d'utiliser la méthode de validation depuis le composant
- Utiliser le module `ReactiveFormModule` proposé par Angular et valider les champs dans le composant
- Le message d'erreur utilisé sera `champ obligatoire` et devra s'afficher en rouge sur le champ concerné par l'erreur
- Le formulaire contiendra un bouton Ajouter qui permettra de soumettre le formulaire et d'ajouter l'équipe
- Un bouton Annuler permettra de rediriger vers la page de liste des équipes



Ajouter un composant qui va permettre de modifier une nouvelle équipe

- Dans la page de liste des équipes, sur chaque ligne proposer un bouton edit ou une icône avec un crayon
- Lors du clic sur cet icône, proposer une page et un composant permettant de modifier une équipe existante
- Les champs `name` et `slogan` seront initialisés avec leur valeurs par défaut
- Un bouton Annuler permettra d'être redirigé vers la page listant les équipes, en annulant l'opération en cours
- Un bouton Modifier permettra de procéder à la modification et redirigera également vers la page de liste des équipes
- Comme pour le composant d'ajout, un Validator dans le composant permettra de vérifier que les champs sont bien renseignés



Supprimer une équipe existante

- Dans le composant qui liste les équipes, sur chaque ligne, proposer un bouton supprimer ou une icône x

- Lors du clic sur cette icone, supprimer l'équipe concernée par la ligne en cours



Modification en fullStack avec ajout d'une liste de joueurs par équipe



Backend

- Dans la partie backend ajouter une liste de joueurs par équipe
- Un joueur `player` a les propriétés suivantes (name, number, position)
- Modifier le script H2
- Faire la mapping Hibernate pour un joueur `PlayerEntity`
- Gérer le mapping entre la classe `TeamEntity` et `PlayerEntity`
- Modifier le reste en vous inspirant des parties précédentes
- Modifier les tests d'intégration de la classe `TeamService` en tenant compte de la liste des joueurs
- Voir s'il y a des impacts dans les classe `TestController` et `TeamControllerTest`



Frontend

- Modifier la page de détail d'une équipe en y affichant la liste des joueurs
- Modifier la page d'ajout d'une nouvelle équipe, en proposant d'ajouter une liste de joueurs. Par exemple un petit bouton + permettra d'ajouter une nouvelle ligne de joueur avec 3 champs obligatoires : name, number, position en input text. Le validator doit aussi vérifier qu'au moins un joueur a été ajouté. Cette ligne sera seulement ajoutée en mémoire dans le composant. Lors du clic sur le bouton ajouter là seulement l'équipe et les joueurs seront ajoutés en base de données
- Modifier la page d'édition d'une équipe en donnant la possibilité d'éditer les joueurs en cours, d'en supprimer, ou d'en ajouter (opérations en mémoire avant le clic sur `Modifier` pour sauvegarder en base). Un validator sera utilisé dans le composant avec les même règles que pour la partie création



Bravo, vous êtes arrivés au bout du TP !!