



Une entreprise ordinaire à l'agilité extraordinaire

# **FORMATION JAVA SPRING ANGULAR**

## **API REST & SPRING**

[mohamed.el-babili@fms-ea.com](mailto:mohamed.el-babili@fms-ea.com)

33+ 628 111 476

Version : 2.0

DMAJ : 26/06/23

## Module : DEV-SPRI-002

### Api Rest & Spring

DEV-SPRI-002	Api Rest avec Spring	<b>01 Connaitre les web et micro services</b>
		<b>02 Exploiter une Api en ligne à l'aide d'un client Rest</b>
		<b>03 Mise en œuvre d'une api Rest à l'aide de Spring</b>
		<b>04 Réalisation d'une appli cliente avec Angular exploitant une Api Rest</b>
		05 Mise en œuvre de la sécurité d'une Api en utilisant Jwt

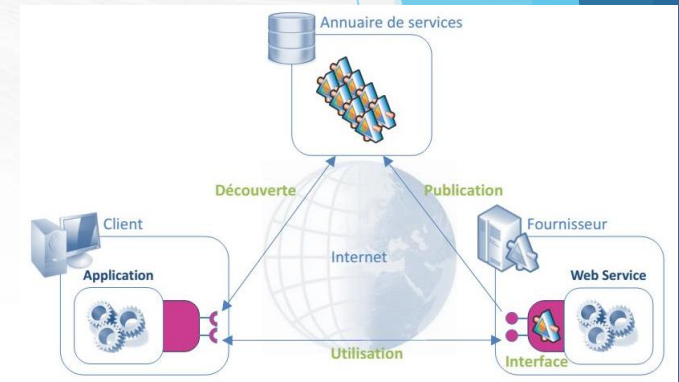
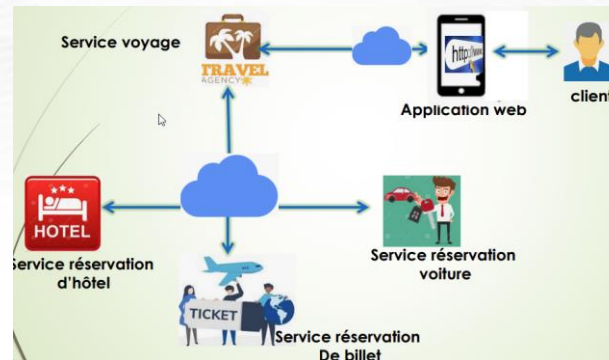
# SOMMAIRE

- ▶ Api Rest & web service
- ▶ Soap & Rest
- ▶ Micro service
- ▶ Web Service Vs Micro service
- ▶ IntelliJ
- ▶ Mise en œuvre d'une Api Rest avec Spring + Bdd H2
- ▶ Tester votre Api avec un client Rest
- ▶ ResponseEntity
- ▶ Exploiter votre Api avec un client Angular
- ▶ Http & Outils de dev
- ▶ Trames Http
- ▶ @RepositoryRestResource
- ▶ Bdd réelle & JpaBuddy
- ▶ Tester une Api
- ▶ Exception & Api
- ▶ Next : Jwt
- ▶ Ressources

# Api Rest & Web service

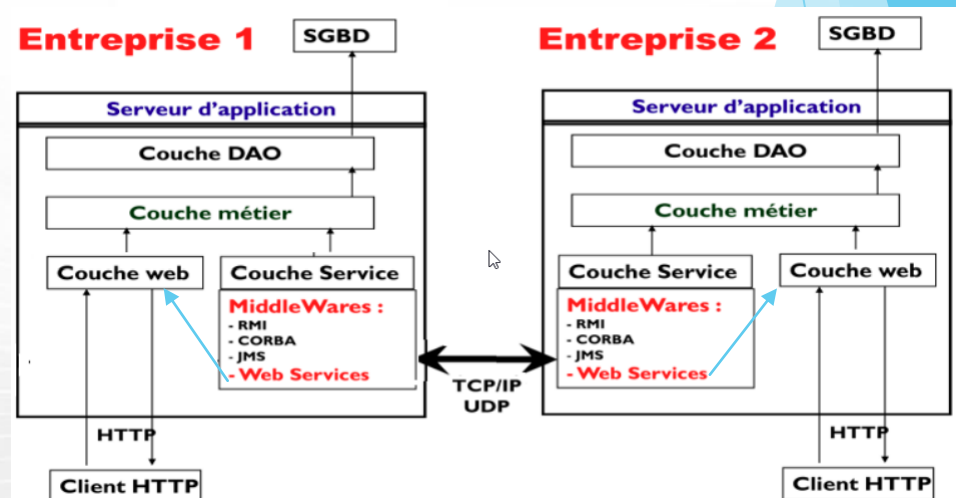
- ▶ **Api Rest** : une application qui communique avec d'autres applications, Rest est la norme utilisée pour l'application
- ▶ L'api propose des **Web services** permettant à une application sur une machine d'avoir accès par ex aux données météorologiques sur une autre machine à l'autre bout du pays, ou encore lorsque vous souhaitez réserver 1 semaine de vacances, l'agence de voyage peut solliciter 3 services Web :

- ▶ **Billet d'avion**
- ▶ **Hôtel**
- ▶ **Voiture de location**



On parle ici d'applications distribuées avec les technologies aujourd'hui dépassées tels que : Rmi, Dcom et Corba car limitées dans la diversité des plates formes, leur lenteur pour traverser les firewalls et l'utilisation du couplage fort.

A contrario, les services web imposent des moyens de communication plus succinct avec un débit plus faible, un client léger (navigateur) et l'utilisation du protocole http comme moyen d'échanges.





# SOAP & REST

Si les services web sont des méthodes de communication entre applications (et/ou appareils électroniques) exclusivement via le web(http), il existe 2 manières ou solutions :

- REST : **Representational State Transfer**
- SOAP : **Simple Object Access Protocol**

**Soap** est un protocole standard pour l'échange de message Xml. Il repose sur différents protocoles de transport comme http, Sntp... Il impose des règles intégrées qui augmentent certes la sécurité mais aussi la complexité et les couts.



**Rest** est un ensemble architectural adapté aux applications légères utilisant le protocole http. Elle est plus simple à comprendre et à mettre en œuvre.



Nb : Une application qui respecte pleinement l'architecture Rest est dite RESTfull

# Micro service

Une architecture à base de micro services est un style architectural visant les applications Web moderne ou la fonctionnalité est divisée en fragments plus petits. En effet, les micro services sont une évolution du style d'architecture orientée services ou SOA (Web Service) dite monolithique.

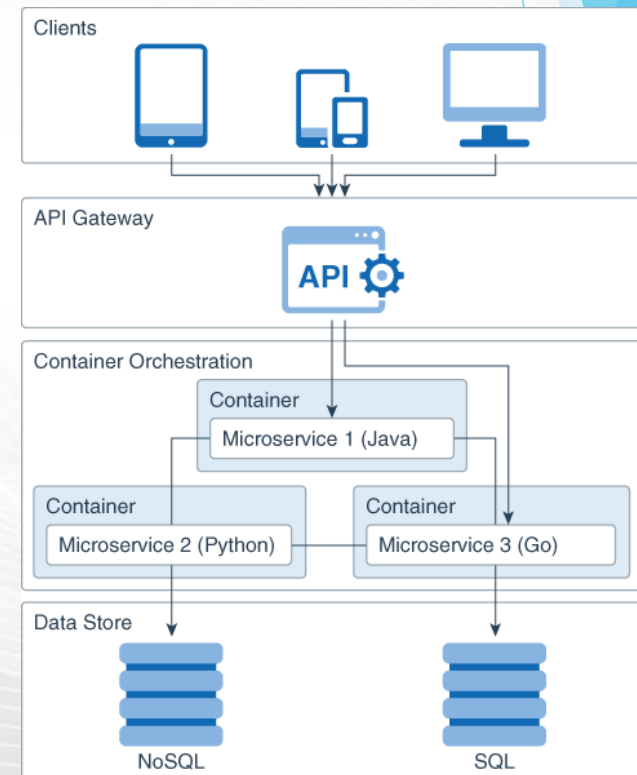
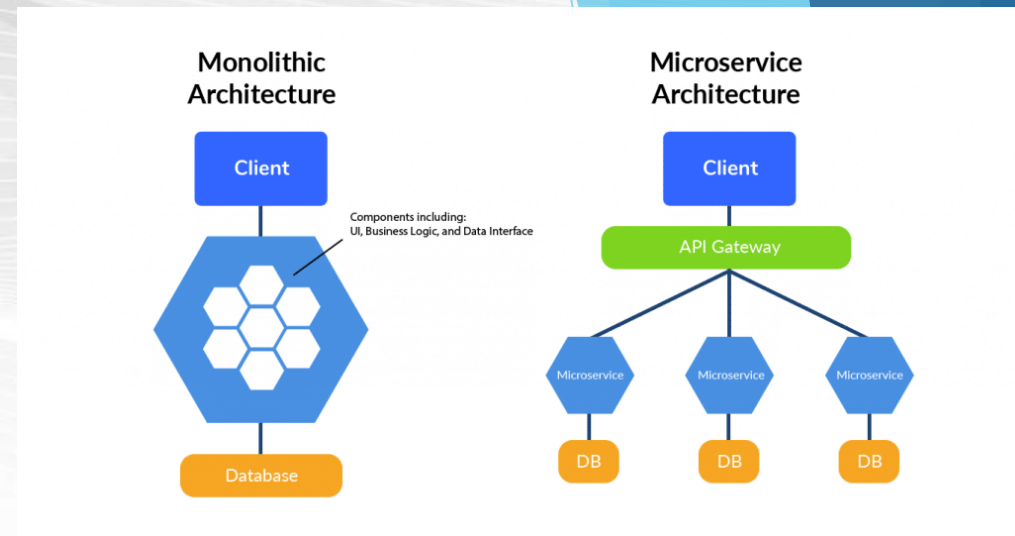
Les micro services sont bien plus facile à développer, à intégrer et à maintenir car les fonctionnalités individuelles sont traités séparément.

Ils sont utiles pour les grandes entreprises car les équipes travaillent sur des éléments distincts sans avoir besoin d'une orchestration complexe.

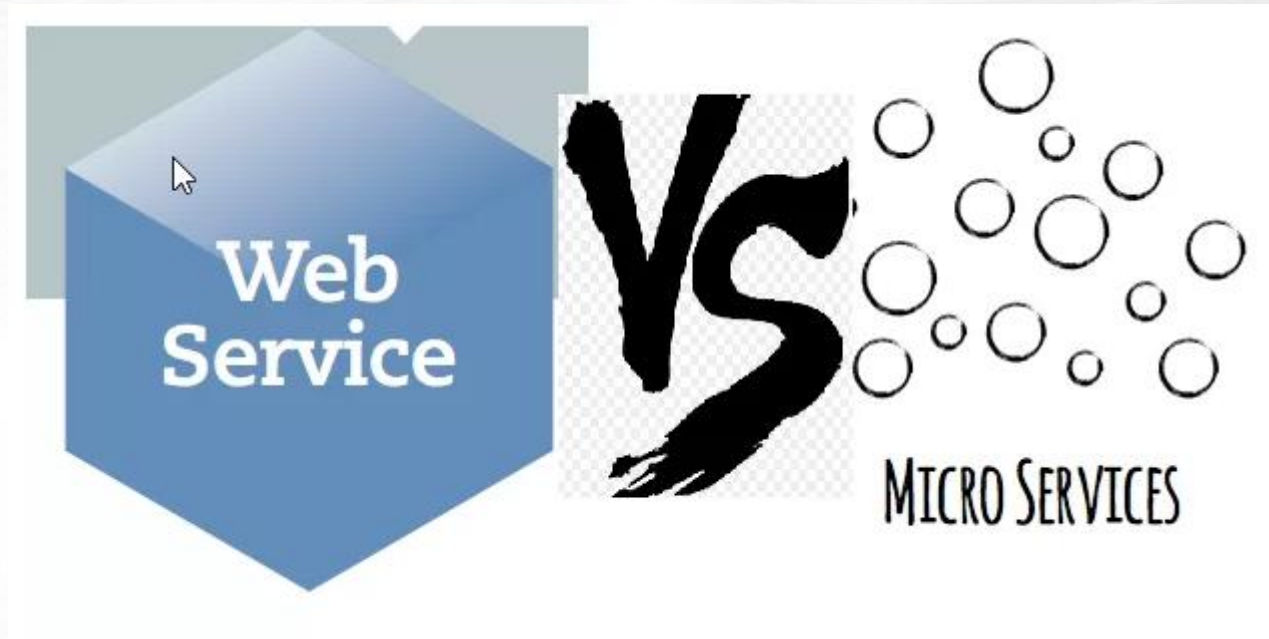
En revanche, si vous divisez une application en plusieurs parties, elles auront besoin de communiquer efficacement, c'est ce qui relie les micro services aux API : **de nombreux micro services utilisent des Api pour communiquer entre eux.**

Les **micro services** sont un style architectural, ou la fonctionnalité est répartie entre de petits services web

alors que les **Api** sont les **cadres** à travers lesquels les développeurs peuvent interagir avec une application



# Web Service Vs Micro service



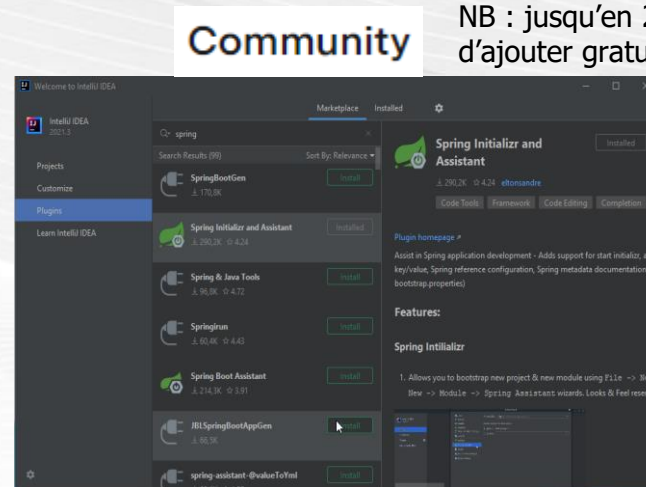
# IntelliJ

	IntelliJ IDEA Ultimate	IntelliJ IDEA Community Edition ⓘ
Java, Kotlin, Groovy, Scala	✓	✓
Maven, Gradle, sbt	✓	✓
Git, GitHub, SVN, Mercurial, Perforce	✓	✓
Débogueur	✓	✓
Docker	✓	✓
Outils de profilage ⓘ	✓	
Spring, Jakarta EE, Java EE, Micronaut, Quarkus, Helidon, etc. ⓘ	✓	
Client HTTP	✓	
JavaScript, TypeScript, HTML, CSS, Node.js, Angular, React, Vue.js	✓	
Outils de base de données, SQL	✓	
Développement à distance (Bêta)	✓	
Développement collaboratif	✓	<input checked="" type="checkbox"/>



# Mise en œuvre d'une Api Rest avec Spring

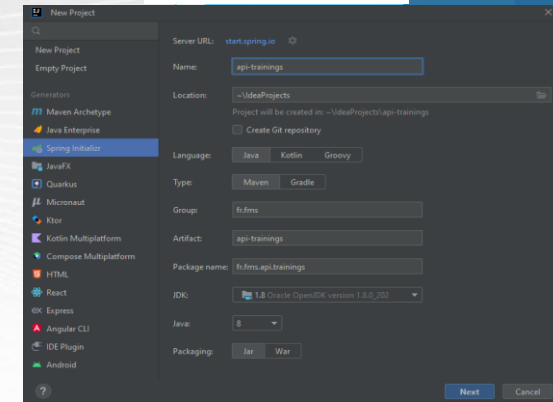
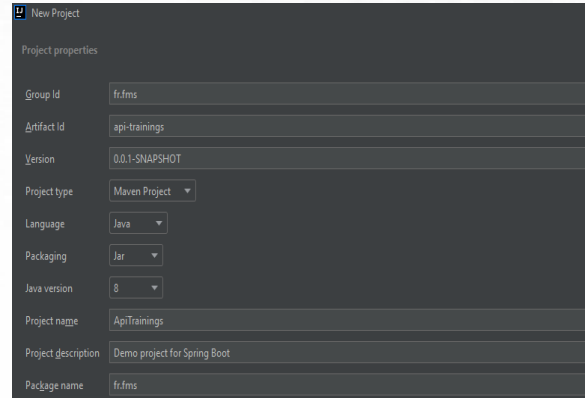
1/ Au démarrage d'IntelliJ, vous devez installer un plug in pour utiliser Spring (ou via settings)



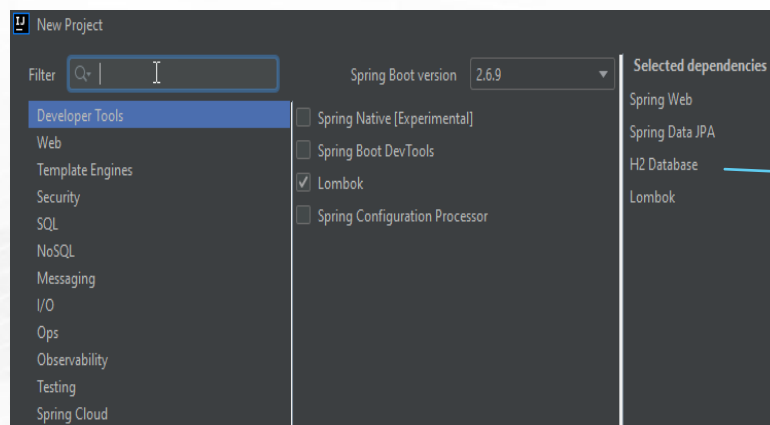
NB : jusqu'en 2022, la version community permet d'ajouter gratuitement l'assistant Spring Initializr

Ultimate

2/ New Project / Spring Initializr / Java 1.8 :



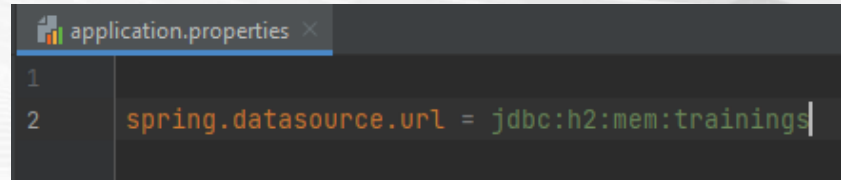
3/ Dépendances & version spring



Bas de données virtuelle

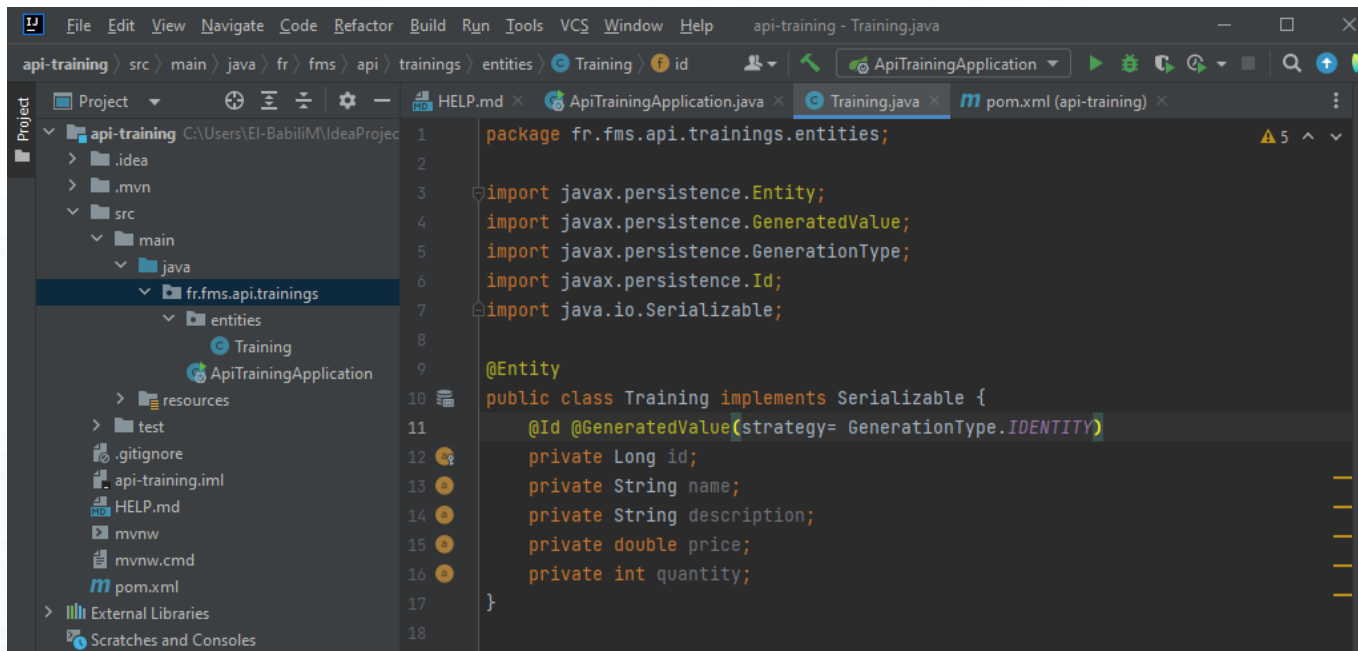
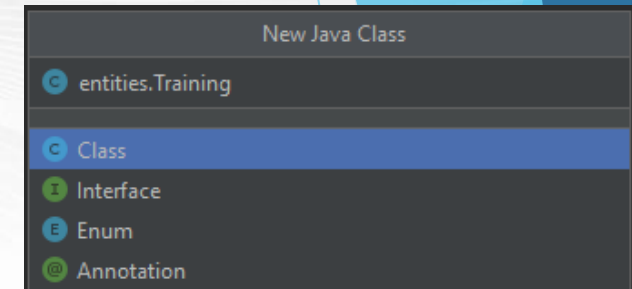
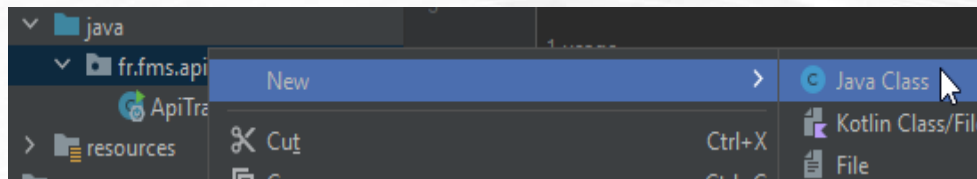
# Mise en œuvre d'une Api Rest avec Spring + Bdd H2

3/ Application.properties :



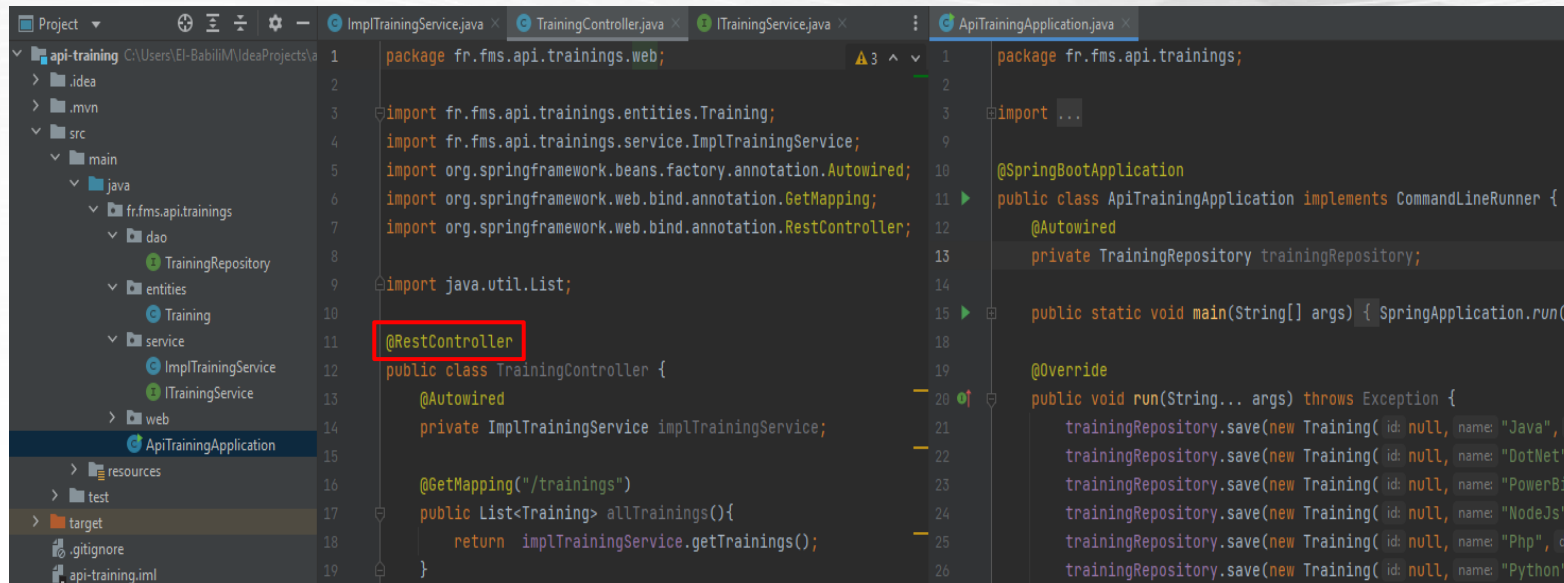
```
1  
2 spring.datasource.url = jdbc:h2:mem:trainings
```

4/ Entités Jpa :



# Mise en œuvre d'une Api Rest avec Spring

## 5/ Couches Dao + Business + Web :



```
package fr.fms.api.trainings.web;

import fr.fms.api.trainings.entities.Training;
import fr.fms.api.trainings.service.ImplTrainingService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import java.util.List;

@RestController
public class TrainingController {

    @Autowired
    private ImplTrainingService implTrainingService;

    @GetMapping("/trainings")
    public List<Training> allTrainings(){
        return implTrainingService.getTrainings();
    }
}
```



```
localhost:8080/trainings

// 20220714125809
// http://localhost:8080/trainings

[
  {
    "id": 1,
    "name": "Java",
    "description": "Java Standard Edition 8 sur 5 jours",
    "price": 3500.0,
    "quantity": 1
  },
  {
    "id": 2,
    "name": "DotNet",
    "description": "DotNet & entityframework en 5 jours",
    "price": 2750.0,
    "quantity": 1
  },
  {
    "id": 3,
    "name": "PowerBi",
    "description": "Business Intelligence 5 jours",
    "price": 3000.0,
    "quantity": 1
  },
  {
    "id": 4,
    "name": "NodeJs",
    "description": "Prise en main de NodeJs/Express 2 jours",
    "price": 1400.0,
    "quantity": 1
  },
  {
    "id": 5,
    "name": "Php",
    "description": "Initiation au Dev/Web avec hp 4 jours",
    "price": 1300.0,
    "quantity": 1
  }
]
```

6/ Lancer l'appli puis un navigateur :  
(pour obtenir ce rendu dans Chrome,  
installer json viewer)

# Tester votre Api avec un client Rest

```
@RestController
@RequestMapping("/api")
public class TrainingController {
    @Autowired
    private ImplTrainingService implTrainingService;

    @GetMapping("/trainings")
    public List<Training> allTrainings(){
        return implTrainingService.getTrainings();
    }

    @PostMapping("/trainings")
    public Training saveTraining(@RequestBody Training t){
        return implTrainingService.saveTraining(t);
    }

    @DeleteMapping("/trainings/{id}")
    public void deleteTraining(@PathVariable("id") Long id){
        implTrainingService.deleteTraining(id);
    }

    @GetMapping("/trainings/{id}")
    public ResponseEntity<Training> getTrainingById(@PathVariable("id") Long id){
        Optional<Training> training = implTrainingService.readTraining(id);
        if(training.isPresent()) {
            return new ResponseEntity<>(training.get(), HttpStatus.OK);
        }
        return null;
    }
}
```

The screenshot shows a REST client interface with the following details:

- Method:** POST
- Request URL:** http://localhost:8080/api/trainings
- Parameters:** (empty)
- Body:** application/json, Raw input view, containing the JSON: 

```
{
  "name": "PowerFull",
  "description": "IA 10 jours",
  "price": 3000,
  "quantity": 1
}
```
- Response:** 200 OK, 8.60 ms, with a JSON body: 

```
{
  "id": 8,
  "name": "PowerFull",
  "description": "IA 10 jours",
  "price": 3000,
  "quantity": 1
}
```

Red arrows indicate the mapping from the Java code to the REST client: one from the `@PostMapping("/trainings")` annotation to the Method and URL fields, and another from the `HttpStatus.OK` constant to the response status.

Est-ce que ce code  
ici est correcte ?



# ResponseEntity

POST ▼ http://localhost:8080/trainings

HEADERS BODY AUTHORIZATION **1** ACTION

Raw input ▼

```
1  {
2    "name": "C++",
3    "description": "Bjarne Stroustrup",
4    "price": 3250,
5    "quantity": 1,
6    "photo": "unknown.png",
7    "category": {
8      "id": 1,
9      "name": "Langage de programmation"
10   }
11 }
```

⋮ Response x

204 no content

Si insertion ok, création d'une URI (Uniform Resource Identifier) contenant l'id de la nouvelle formation transmis à l'objet renvoyé avec un code 201

<https://github.com/advanced-rest-client/arc-electron/releases>

Pour windows (App Desktop) installer arc-setup.exe

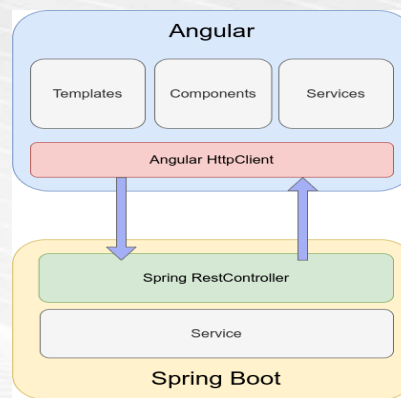
**ResponseEntity** : Classe qui hérite de `HttpEntity` permettant de définir le code http retourné par nos soins, voir la javadoc de spring [ici](#)

```
@PostMapping("/trainings")
public ResponseEntity<Training> saveTraining(@RequestBody Training t){
    Training training = trainingService.saveTraining(t);
    if(Objects.isNull(training)) {
        return ResponseEntity.noContent().build();
    }
    URI location = ServletUriComponentsBuilder
        .fromCurrentRequest()
        .path("/{id}")
        .buildAndExpand(training.getId())
        .toUri();
    return ResponseEntity.created(location).build();
}
```

Construit le header et ajoute le code

201

# Tester votre Api avec un client Angular



Les 2 applications sont autonomes en soi, juste que l'appli angular sollicite dorénavant l'api rest spring pour récupérer les données

Nous pouvons reprendre l'appli Angular existante, vérifier si le service dédié à la communication est cohérent :

```
export const environment = {
  production: false,
  host : "http://localhost:8080/api"
};
```

```
export class ApiService {

  constructor(private http:HttpClient) { }

  public getTrainings() {
    return this.http.get<Training[]>(environment.host+"/trainings");
  }

  public postTraining(training : any){
    return this.http.post<Training>(environment.host+"/trainings" , training);
  }

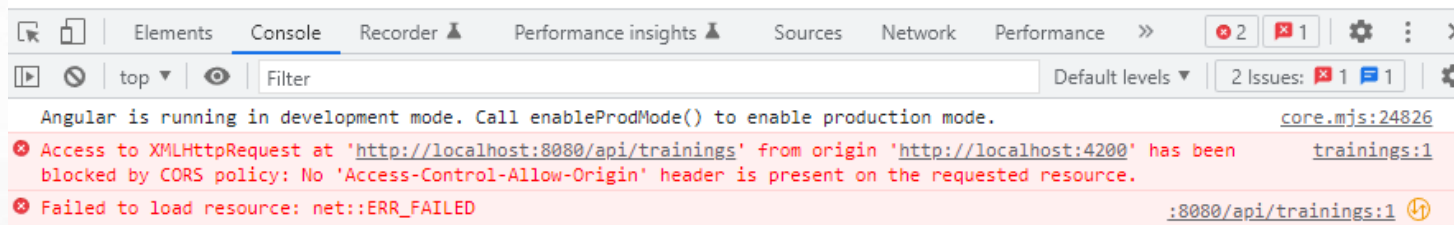
  public delTraining(training: Training) {
    return this.http.delete(environment.host+"/trainings/" + training.id);
  }

  public getTraining(id : number) {
    return this.http.get<Training>(environment.host+"/trainings/"+id);
  }
}
```

Pour tester, il faudra d'abord lancer l'Api avec IntelliJ

Puis le client Angular avec Vsc

Si ça ne marche pas alors que tous les indicateurs sur nos Ide sont bon !









Il faut juste indiquer à notre Controller qu'on accepte les requêtes qui viennent d'autre domaine

```
@CrossOrigin("*")
//@CrossOrigin("http://localhost:4200/")
@RestController
@RequestMapping("/api")
public class TrainingController {
```

# Nous avons migré notre Api de Json server vers Spring Rest Api

localhost:4200/trainings

TRAININGS Liste des Formations Contenu du Panier [3] Connexion

ID	NOM	DESCRIPTION	PRIX	QUANTITE	PANIER
1	Java	Java Standard Edition 8 sur 5 jours	3500	<input type="text" value="1"/>	 Ajouter
2	DotNet	DotNet & entityframework en 5 jours	2750	<input type="text" value="1"/>	 Ajouter
3	PowerBi	Business Intelligence 5 jours	3000	<input type="text" value="1"/>	 Ajouter
4	NodeJs	Prise en main de NodeJs/Express 2 jours	1400	<input type="text" value="1"/>	 Ajouter
5	Php	Initiation au Dev/Web avec hp 4 jours	1300	<input type="text" value="1"/>	 Ajouter
6	Python	Formation Python/Django 10 jours	1500	<input type="text" value="1"/>	 Ajouter

Elements Console Recorder Performance insights Sources Network Performance Memory Application Security Lighthouse Redux Angular 1

top Filter Default levels 1 Issue: 1

[webpack-dev-server] Live Reloading enabled. [index.js:551](#)

Angular is running in development mode. Call enableProdMode() to enable production mode. [core.mjs:24826](#)

# Http & Outils de dev : F12

Afin d'observer le contenu des requêtes http, tous les navigateurs offrent des outils de dev qu'il faut connaître.



Chrome DevTools

TrainingsFrontApp

localhost:4200/trainings

TRAININGS Liste des Formations Contenu du Panier [3]

ID	NOM	DESCRIPTION	PRIX	QUANTITE	PANIER
1	Java	Java Standard Edition 8 sur 5 jours	3500	1	Ajouter
2	DotNet	DotNet & entityframework en 5 jours	2750	1	Ajouter

Elements Console Recorder Performance insights Sources Network Performance Memory Application Security Lighthouse Redux Angular

Filter Invert Hide data URLs All Fetch/XHR JS CSS Img Media Font Doc WS Wasm Manifest Other Has blocked cookies Blocked Requests 3rd-party requests

100 ms 200 ms 300 ms 400 ms 500 ms 600 ms 700 ms 800 ms 900 ms 1000 ms 1100 ms 1200 ms

Name

- trainings
- styles.css
- runtime.js
- polyfills.js
- vendor.js
- main.js
- styles.js
- backend\_bundle.js
- trainings
- favicon.ico
- ws
- fontawesome-webfont.woff2?...

Headers Preview Response Initiator Timing

General

Request URL: http://localhost:8080/api/trainings

Request Method: GET

Status Code: 200

Remote Address: [::1]:8080

Referrer Policy: strict-origin-when-cross-origin

Response Headers View source

Access-Control-Allow-Origin: http://localhost:4200

Connection: keep-alive

Content-Type: application/json

Date: Fri, 15 Jul 2022 09:17:10 GMT

Keep-Alive: timeout=60

Transfer-Encoding: chunked

Vary: Origin

Vary: Access-Control-Request-Method

Vary: Access-Control-Request-Headers

Request Headers

Accept: application/json, text/plain, \*/\*

Accept-Encoding: gzip, deflate, br

Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7

Connection: keep-alive

Host: localhost:8080

Origin: http://localhost:4200

Referer: http://localhost:4200/

- 200 : succès de la requête ;
- 301 et 302 : redirection, respectivement permanente et temporaire ;
- 401 : utilisateur non authentifié ;
- 403 : accès refusé ;
- 404 : ressource non trouvée ;
- 500, 502 et 503 : erreurs serveur ;
- 504 : le serveur n'a pas répondu.

- **Origin** : l'URL de laquelle est produite la requête
- **Content-Type** : Indique le type de média (text/html ou text/JSON) de la réponse envoyée au client par le serveur, cela aidera le client à traiter correctement le corps de la réponse.
- **Cache-Control** : Il s'agit de la politique de cache définie par le serveur pour cette réponse, une réponse mise en cache peut être stockée par le client et réutilisée jusqu'à l'heure définie par l'en-tête Cache-Control.
- **Authorization** : porte les informations d'identification contenant les informations d'authentification du client pour la ressource demandée.

NB : La réponse contient les éléments de la requête

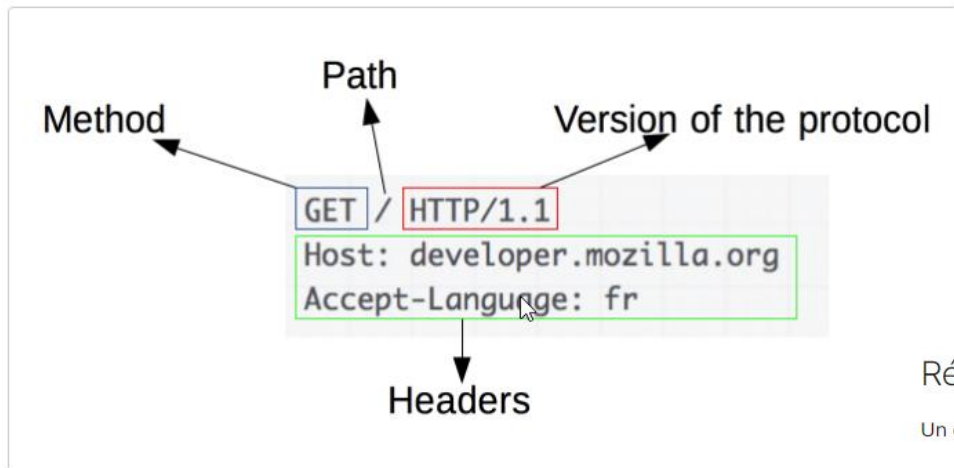
- **1xx codes: Informational**
- **2xx codes: Succès**
- **3xx codes: Redirection**
- **4xx codes: Client Error**
- **5xx codes: Server Error**



# Trames Http

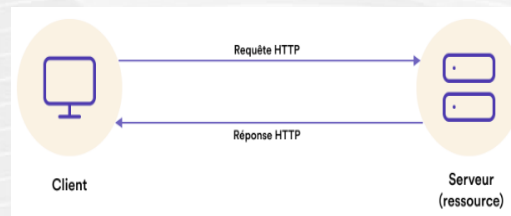
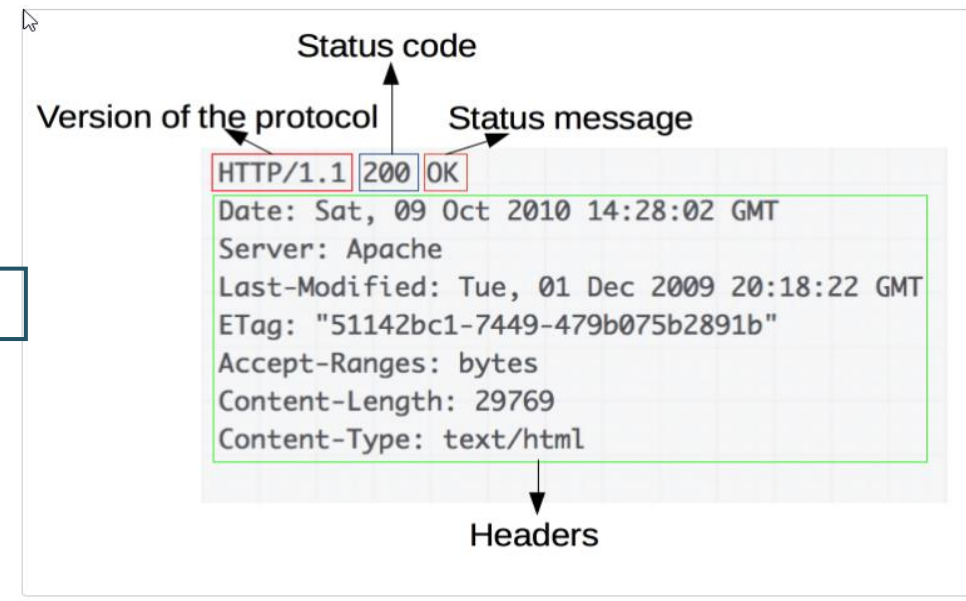
## Requêtes

Un exemple de requête HTTP :



## Réponses

Un exemple de réponse :



# @RepositoryRestResource

Voilà une autre manière de récupérer les données, en utilisant cette annotation, on transforme notre interface Jpa ressource Rest offrant l'accès aux données « brutes »  
Commençons par désactiver notre unique Controller :

```
//@CrossOrigin("*")
//@CrossOrigin("http://localhost:4200/")
//@RestController
//@RequestMapping("/api")
public class TrainingController {
```



Request URL: http://localhost:8080/api/trainings  
Request Method: GET  
Status Code: 404

Ajoutons l'annotation @RepositoryRestResource dans notre interface Jpa

```
@CrossOrigin("*")
@RepositoryRestResource
public interface TrainingRepository extends JpaRepository<Training, Long> {
    public List<Training> findAll();
}
```

Il faut au préalable ajouter la dépendance :

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
```



Résultat

```
{
  "_embedded": {
    "trainings": [Array[6]]
    -0: {
      "name": "Java",
      "description": "Java Standard Edition 8 sur 5 jours",
      "price": 3500,
      "quantity": 1,
      "_links": {
        "self": {
          "href": "http://localhost:8080/trainings/1"
        },
        "training": {
          "href": "http://localhost:8080/trainings/1"
        }
      }
    },
    -1: {
      "name": "DotNet",
      "description": "DotNet & entityframework en 5 jours",
      "price": 2750,
      "quantity": 1,
      "_links": {
        "self": {
          "href": "http://localhost:8080/trainings/2"
        },
        "training": {
          "href": "http://localhost:8080/trainings/2"
        }
      }
    },
    -2: {
      "name": "PowerBi",
      "description": "Business Intelligence 5 jours",
      "price": 3000,
      "quantity": 1,
      "_links": {
        "self": {
          "href": "http://localhost:8080/trainings/3"
        },
        "training": {
          "href": "http://localhost:8080/trainings/3"
        }
      }
    },
    -3: {
      "name": "NodeJs",
      "description": "Prise en main de NodeJs/Express 2 jours",
      "price": 1400,
      "quantity": 1,
      "_links": {
        "self": {
          "href": "http://localhost:8080/trainings/4"
        }
      }
    }
  ]
}
```

@RestController annotation renvoie une ressource Rest à partir d'un contrôleur, alors que @RepositoryRestResource expose une interface Jpa comme une ressource Rest.

# Bdd réelle & JpaBuddy

Commençons par les dépendances + fichier de config

```
<!--      <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <scope>runtime</scope>
    </dependency>-->
    <dependency>
      <groupId>org.mariadb.jdbc</groupId>
      <artifactId>mariadb-java-client</artifactId>
      <scope>runtime</scope>
    </dependency>
```

```
#Database
spring.datasource.url = jdbc:mariadb://localhost:3306/training?createDatabaseIfNotExist=true
spring.datasource.username = root
spring.datasource.password = fms2022
spring.datasource.driver-class-name = org.mariadb.jdbc.Driver

#Jpa-Hibernate
spring.jpa.show-sql= false
spring.jpa.hibernate.ddl-auto= create
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MariaDB53Dialect

#Server.port=8081
```

Une fois que tout est opérationnel, penchons nous sur JpaBuddy



Il s'agit d'un plugin dans IntelliJ permettant d'exploiter efficacement Hibernate, EclipseLink, Spring data Jpa, Flyway & Liquibase, lombok, MapStruct... et toutes techno propre à Java et Kotlin.

- Il rend plus simple le démarrage d'appli usant de Jpa et tout ce qui a trait
- Il booste la productivité des dévs qq soit le niveau
- Assure que le code généré suit les bonnes pratiques sans jamais planter

Il permet rapidement et facilement de :

- créer des entités Jpa et les objets liés aux données
- gérer la migration d'une base de données
- générer des entités à partir de tables existantes dans une base

JpaBuddy répond parfaitement à notre besoin d'automatiser des aspects techniques afin de nous concentrer sur les aspects métier, [voir plus ici](#)

# Tester une Api

## Test unitaire

```
@WebMvcTest(controllers = TrainingController.class) //déclenche le test sur le controller cible
public class TrainingControllerTest {

    @Autowired
    private MockMvc mockMvc; //va servir à appeler la méthode perform

    @MockBean //indispensable pour appeler le service
    private ImplTrainingService implTrainingService; //la méthode get("/api/trainings") va appeler :
                                                    //allTrainings(){ return implTrainingService.getTrainings(); }

    @Test
    public void testGetTrainings() throws Exception {
        mockMvc.perform(get( uriTemplate: "/api/trainings")) //requete en get sur le controller avec cette url
                .andExpect(status().isOk()); //nous attendons une réponse http 200
    }
}
```

## Test d'intégration

```
//Les annotations ci-dessous chargent le contexte spring permettant de réaliser des requetes sur un controller
@SpringBootTest //Injecte le service nécessaire ici, plus besoin de l'annotation MockBean
@AutoConfigureMockMvc
class ApiTrainingApplicationTests {

    @Autowired
    private MockMvc mockMvc;

    @Test
    void testGetTrainingsAndTestName() throws Exception {
        mockMvc.perform(get( uriTemplate: "/api/trainings"))
                .andExpect(status().isOk()) //vérif si status 200 ok
                .andExpect(jsonPath( expression: "$[0].name",is( value: "Java")));
        //vérifie si la réponse contient une valeur précise :
        // $ pointe sur la racine de la structure JSON.
        // [0] indique qu'on veut vérifier le premier élément de la liste.
        // name désigne l'attribut qu'on veut consulter.
    }
}
```



# Exceptions & Api

Que se passe-t-il si vous sollicitez votre api sur un id inexistant : en l'état actuel, rien !

Il existe [plusieurs approches usant des exceptions](#) en fonctions des évolutions de Spring, nous allons présenter ici la combinaison : `@ControllerAdvice` & `@ExceptionHandler`

L'idée est de centraliser toutes les exceptions vers un gestionnaire unique

```
@GetMapping("/trainings/{id}")
public Training getTrainingById(@PathVariable("id") Long id){
    return implTrainingService.readTraining(id)
        .orElseThrow( () -> new RecordNotFoundException("Id de Formation " + id + " n'existe pas") );
}
```

```
@ResponseStatus(HttpStatus.NOT_FOUND)
public class RecordNotFoundException extends RuntimeException {
    private static final long serialVersionUID = 1L;
    public RecordNotFoundException(String message) {
        super(message);
    }
}
```

```
@Data
public class ErrorResponse {
    private String message;
    private List<String> details;
    public ErrorResponse(String message, List<String> details) {
        super();
        this.message = message;
        this.details = details;
    }
}
```

```
@ControllerAdvice
public class TrainExceptionHandler extends ResponseEntityExceptionHandler
{
    private String INCORRECT_REQUEST = "INCORRECT_REQUEST";
    private String BAD_REQUEST = "BAD_REQUEST";

    @ExceptionHandler({RecordNotFoundException.class})
    public final ResponseEntity<ErrorResponse> handleUserNotFoundExpection
        (RecordNotFoundException ex, WebRequest request)
    {
        List<String> details = new ArrayList<>();
        details.add(ex.getLocalizedMessage());
        ErrorResponse error = new ErrorResponse(INCORRECT_REQUEST, details);
        return new ResponseEntity<>(error, HttpStatus.NOT_FOUND);
    }

    @ExceptionHandler({MissingHeaderInfoException.class})
    public final ResponseEntity<ErrorResponse> handleInvalidTraceIdExpection
        (MissingHeaderInfoException ex, WebRequest request) {
        List<String> details = new ArrayList<>();
        details.add(ex.getLocalizedMessage());
        ErrorResponse error = new ErrorResponse(BAD_REQUEST, details);
        return new ResponseEntity<>(error, HttpStatus.BAD_REQUEST);
    }
}
```

```
localhost:8080/api/trainings/20

// 20220717180832
// http://localhost:8080/api/trainings/20

{
  "message": "INCORRECT_REQUEST",
  "details": [
    "Id de Formation 20 n'existe pas"
  ]
}
```

# Next : Json Web Token

Sécurité basée sur les cookies et les sessions :  
**statefull**

Les données de la session sont enregistrés côté serveur



Sécurité basée sur les tokens (Jwt) :  
**stateless**

Les données de la session sont enregistrés dans un jeton envoyé au client



Sécurité basée sur les blockchains :  
Décentralisation + consensus + crypto



# Ressources

<https://openclassrooms.com/fr/courses/4668056-construisez-des-microservices>

<https://openclassrooms.com/fr/courses/4668056-construisez-des-microservices/7652911-documentez-votre-microservice-avec-swagger-2>