spring MVC
-
(jsp , thymeleaf, REST)

Table des matières

I - Architectures web / spring (vue d'ensemble)

1. Spring web overview

Plein de variantes sont envisageables et on peut quelquefois les faire coexister .

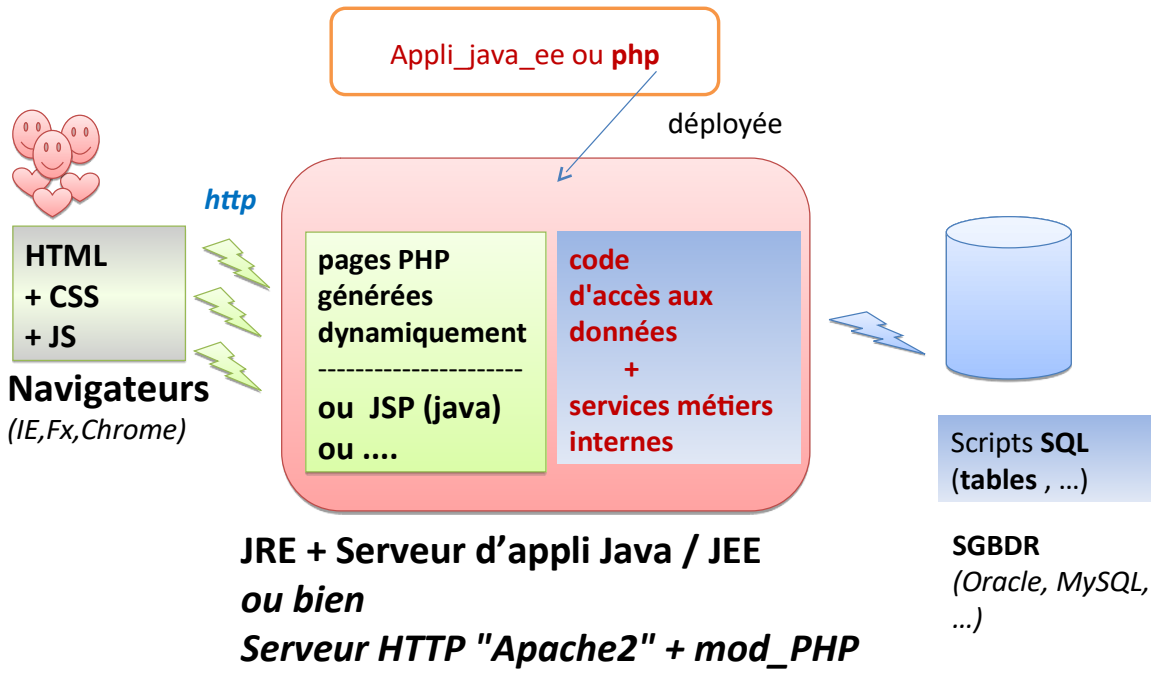
Types de parties web	*Caractéristiques importantes*
Partie web ne comportant que des **WEB-SERVICES REST**	**@RestController** et sécurisation avec des **tokens** (souvent JWT) quelquefois issus d'un serveur OAuth2/Oidc . Avec souvent une **documentation swagger/openApidoc**.
Partie web ne comportant que des **pages HTML générées dynamiquement** (via ".jsp" ou thymeleaf ou autres)	À sécuriser avec des **cookies** et la protection **CRSF** . Généralement Codé via : - **"Spring + JSF"** ou bien -**"Spring + Struts 2"** ou bien - **"Spring-MVC"** (avec jsp ou bien thymeleaf) ou bien (rarement) - simples **Servlet**(s) plus pages **JSP**
Double partie web (API REST + pages HTML générées dynamiquement)	Prévoir plusieurs points d'entrée et deux types de sécurisation (à faire coexister) !

Dans presque tous les cas il y aura une page d'accueil simple (faisant office de menu) matérialisée sous forme de fichier **index.html** (souvent placée dans **src/main/resources/static**) .

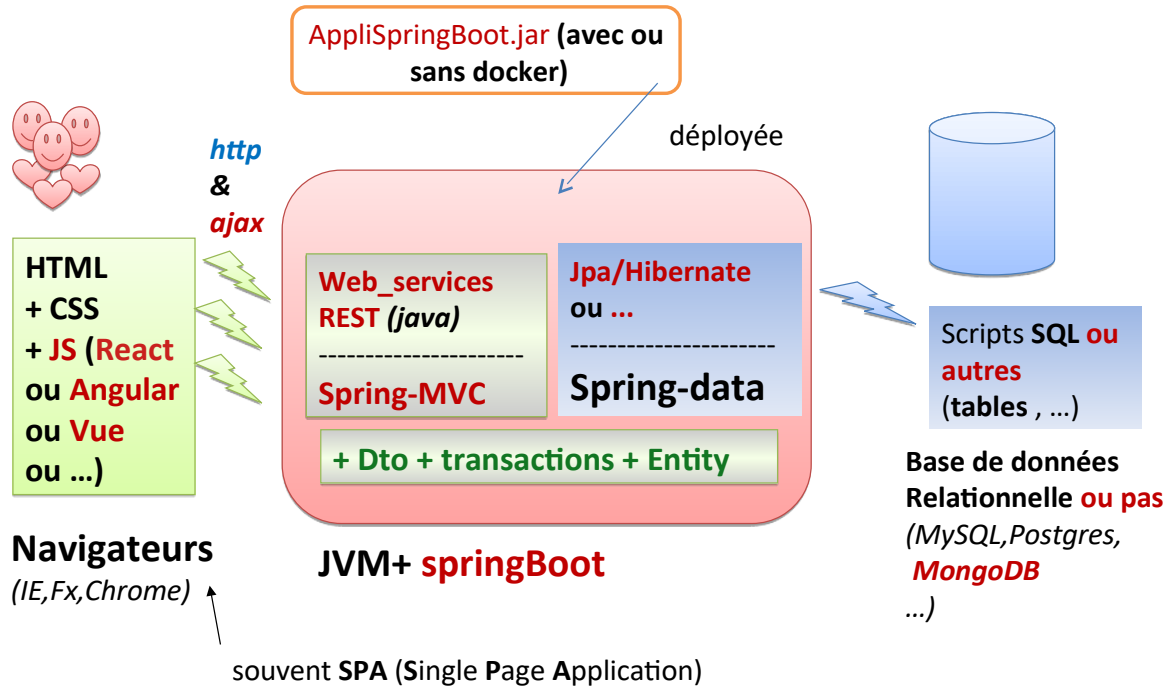
NB :

- Si l'on utilise un framework web additionnel "pas spring" tel que "Struts2" ou "JSF" alors le lien s'effectue via "spring-web" et d'autres compléments facultatifs.
- Le framework "Spring-MVC" peut aussi bien servir à coder une "api REST" qu'à générer dynamiquement des pages HTML (via ".jsp" ou thymeleaf")
- L'application complète spring (avec sa ou ses parties "web") pourra soit fonctionner dans un serveur d'application JEE (ex : tomcat) après un déploiement de ".war" ou bien fonctionner de manière autonome si elle est basée sur springBoot avec un déploiement de ".jar" (placé ou pas dans un conteneur "docker") .
- Depuis "spring >=5" , il est quelquefois possible de coder la partie web en s'appuyant sur des technologies très asynchrones (ceci permet d'un coté d'obtenir un petit gain en performance mais d'un autre coté ça rend le code beaucoup plus complexe à écrire et maintenir d'autant plus que l'on manque encore de recul sur la pérennité des technos asynchrones pas encore standardisées à l'échelle du langage java)

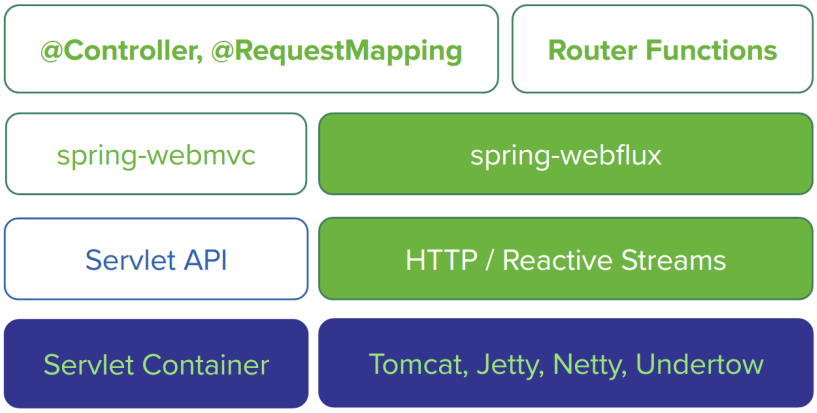
Ancienne architecture web

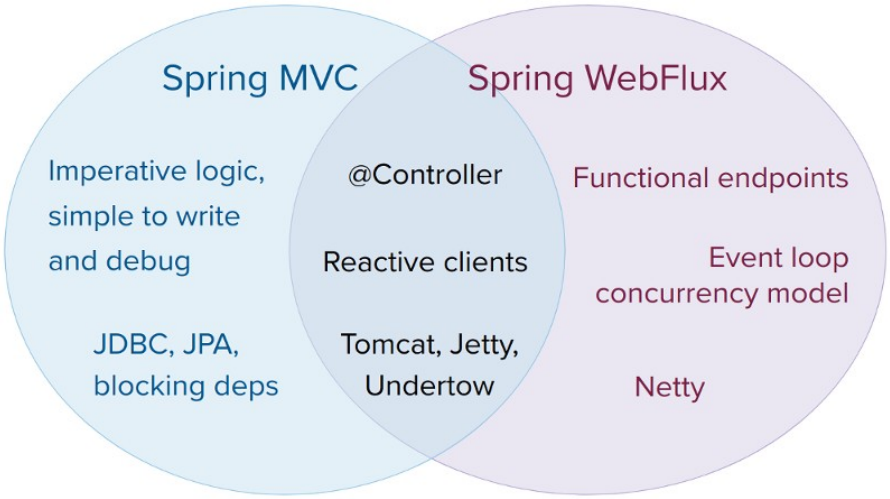


Frontend + Backend



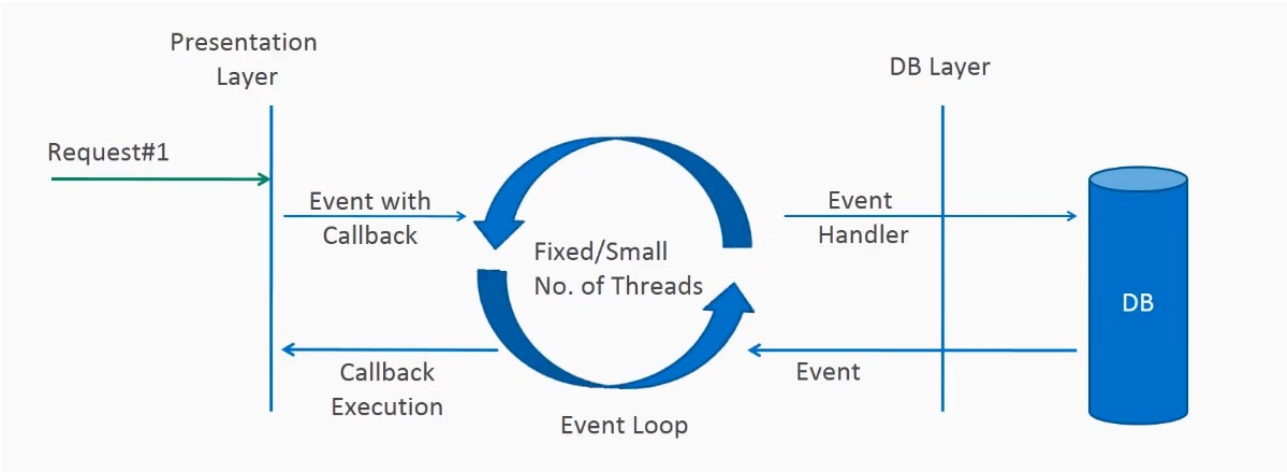
2. Vue d'ensemble sur Spring asynchrone





Framework	Fonctionnalités	Documentation et exemples
Netty	Framework asynchrone non bloquant , peut être utilisé en mode serveur (à la place de tomcat)	https://www.baeldung.com/netty
Reactor	Framework de programmation réactive en java (un peu comme RxJava et RxJs)	https://projectreactor.io/ https://spring.io/reactive https://www.baeldung.com/reactor-core
WebFlux	Une sorte de version asynchrone de Spring-mvc (pouvant être utilisé pour coder ou invoquer des api REST)	https://docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html https://www.baeldung.com/spring-webflux

Attention: ***Nouveautés de version 5 (technologies très récentes, pas encore "classique/mature").***



En gros , Spring web-flux reprend les mêmes principes de fonctionnement que nodeJs .

II - Spring-web (avec serveur, génération HTML)

1. Lien avec un serveur JEE et Spring web

Configuration nécessaire pour qu'une application Spring (packagée comme un ".war") puisse bien démarrer au sein d'un serveur Jakarta-EE (ex : tomcat10) :

Type d'application Spring	Point d'entrée en mode "web" au sein d'un serveur JEE (après déploiement du ".war")
Spring-framework (sans SpringBoot) et avec veille config ***mySpringConf.xml*** et **WEB-INF/web.xmml**	<context-param> <param-name>**contextConfigLocation**</param-name> <param-value>classpath:/**mySpringConf.xml**</param-value> </context-param> .... <listener> <listener-class>org.springframework.web.context.**ContextLoaderListener**</listener-class> </listener>
Spring-framework (sans SpringBoot) et avec **config java**	Classe java implémentant l'interface **WebApplicationInitializer** et comportant une méthode **onStartup**() au sein de laquelle on enregistre la configuration Spring dans un ***ContextLoaderListener*** associé au ***servletContext*** . *Ou bien variante simplifiée en utilisant la classe abstraite **AbstractContextLoaderInitializer** .*
SpringBoot moderne	Faire hériter la classe principale de **SpringBootServletInitializer** et coder .**configure**() avec builder.**source**(...)

2. Injection de Spring au sein d'un framework WEB

2.1. WebApplicationContext (configuration xml)

Cette ancienne configuration (au format XML) est placée dans un document complémentaire à cette version récente du support de cours

2.2. WebApplicationContext (configuration java-config)

```
class MyWebApplicationInitializer implements WebApplicationInitializer {
 public void onStartup (.. servletContext )...{
   WebApplicationContext context = new AnnotationConfigWebApplicationContext ();
 context.register (MyWebRootAppConfig.class );
 servletContext .addListener (new    ContextLoaderListener (context ));
 //... }
```

```
}
```

MyWebRootAppConfig.class peut par exemple correspondre à la classe de configuration Spring principale (elle même potentiellement reliée à d'autres sous-configurations via @Import).

Variante simplifiée via AbstractContextLoaderInitializer

```
public class AnnotationsBasedApplicationInitializer
  extends AbstractContextLoaderInitializer {

    @Override
    protected WebApplicationContext createRootApplicationContext() {
        AnnotationConfigWebApplicationContext rootContext
          = new AnnotationConfigWebApplicationContext();
        rootContext.register(MyWebRootAppConfig.class);
        return rootContext;
    }
}
```

URL pour approfondir si bseoin le sujet :
https://www.baeldung.com/spring-web-contexts

2.3. WebApplicationContext (accès et utilisation)

Au sein d'un servlet ou bien d'un élément annexe on peut instancier des Beans via Spring :

```
application = .... getServletContext(); // application prédéfini au sein d'une page JSP
WebApplicationContext ctx =
                    WebApplicationContextUtils.getWebApplicationContext(application);
IXxx  bean = (IXxx) ctx.getBean(....);
....
request.setAttribute("nomBean",bean); // on stocke le bean au sein d'un scope (session,request,...)
rd.forward(request,response); // redirection vers page JSP
```

NB : Spring-web propose en plus des configurations complémentaires spécifiques pour bien intégrer la plupart des frameworks java-web (Struts, JSF , ...)

3. Packaging d'une application SpringBoot pour un déploiement vers un serveur JEE

Par défaut, une application moderne SpringBoot est packagée en tant que ".jar" et fonctionne de manière autonome sans serveur (en embarquant un conteneur web imbriqué tel que tomcat ou jetty).

Il est tout de même possible de configurer une application SpringBoot de manière à ce que l'on puisse effectuer un déploiement web classique dans un serveur d'application JEE.

Il faut pour cela :

- modifier le **packaging** de l'application dans pom.xml (de "jar" vers "**war**")
  <packaging>war</packaging>
- faire en sorte que la classe principale de l'application hérite de **SpringBootServletInitializer**
- déclencher **builder.sources**(MySpringBootApplication .class); dans une redéfinition de **configure**() de manière à préciser le point d'entrée de la configuration de l'application.

Exemple :

```
package tp.appliSpring;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.support.SpringBootServletInitializer;

@SpringBootApplication
public class MySpringBootApplication  extends SpringBootServletInitializer {

    @Override
   protected SpringApplicationBuilder configure(SpringApplicationBuilder builder) {
      /* builder.profiles("dev"); */ //setting profiles here
      // or with system properties of the server (ex: tomcat)
       return builder.sources(MySpringBootApplication.class);
   }

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

4. Présentation du framework "Spring MVC"

"Spring Web MVC" est une partie optionnelle du framework spring servant à gérer la logique du

design pattern "MVC" dans le cadre d'une intégration "spring" .

A l'origine (vers les années 2005-2012) , "Spring MVC" était à voir comme un petit framework java/web (pour le coté serveur) qui se posait comme une alternative à Struts2 ou JSF2 .

Plus récemment , "Spring MVC" (souvent intégré dans SpringBoot) est énormément utilisé pour développer des Web-Services REST et est quelquefois encore un peu utilisé pour générer des pages HTML (via des vues ".jsp" ou bien des vues ".html" de Thymeleaf) .

4.1. configuration maven pour spring-mvc

Dépendances maven nécessaires (en intégration moderne "spring-boot"):

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

et (si vues de type "**.jsp**")

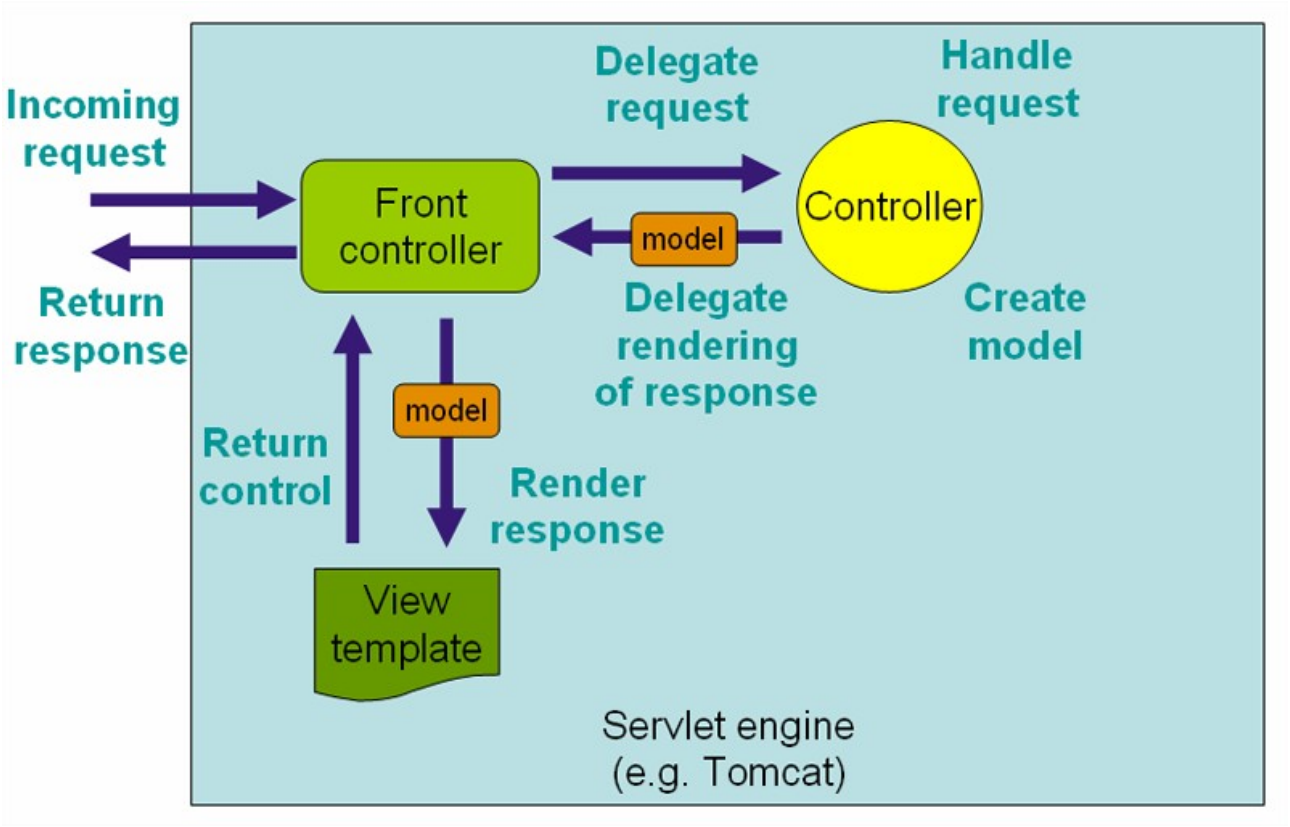
```
      <dependency>
            <groupId>org.apache.tomcat.embed</groupId>
            <artifactId>tomcat-embed-jasper</artifactId>
            <scope>provided</scope>
      </dependency>
      <dependency>
            <groupId>jakarta.servlet.jsp.jstl</groupId>
            <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
      </dependency>
      <dependency>
            <groupId>org.glassfish.web</groupId>
            <artifactId>jakarta.servlet.jsp.jstl</artifactId>
      </dependency>
  <!-- ou ancien équivalent spring5/springBoot2/jee -->
  <!-- <dependency>
                  <groupId>javax.servlet</groupId>
                  <artifactId>jstl</artifactId>
      </dependency> -->
```

ou bien (avec **Thymeleaf**)

```
            <dependency>
                  <groupId>org.springframework.boot</groupId>
                  <artifactId>spring-boot-starter-thymeleaf</artifactId>
            </dependency>
        <dependency>
             <groupId>nz.net.ultraq.thymeleaf</groupId>
             <artifactId>thymeleaf-layout-dialect</artifactId>
         </dependency>
```

5. Mécanismes fondamentaux de "spring mvc"

5.1. Principe de fonctionnement de SpringMvc :



NB :

- Le **contrôleur** est une instance d'une classe java préfixée par **@Controller** (composant spring de type contrôleur web) et de @Scope("singleton") par défaut .

  Ce contrôleur a la responsabilité de préparer des données (souvent récupérées en base et quelquefois à partir de critères de recherches)
- Le **model** est une table d'association (nomAttribut, valeurAttribut) (par défaut en scope=request) permettant de passer des objets de valeurs à afficher au niveau de la vue.
- La **vue** est responsable d'effectuer un rendu (souvent "html + css + js") à partir des valeurs du modèle.
  La **vue** est souvent une **page JSP** ou bien un **template "Thymeleaf"** .
- Ce framework peut fonctionner en mode simplifié (sans vue) avec une génération automatique de réponse au format **JSON** ou **XML** dans le cadre de Web Services REST.

5.2. Exemple Spring-Mvc simple en version ".jsp":

src/main/resources/**application.properties**

```
server.servlet.context-path=/myMvcSpringBootApp
server.port=8080
#spring.mvc.view.prefix=/WEB-INF/view/
spring.mvc.view.prefix=/jsp/
spring.mvc.view.suffix=.jsp
```

Avec cette configuration , un **return "xy"** d'un contrôleur déclenchera l'affichage de la page **/jsp/xy.jsp** et selon la structure du projet , le répertoire /***jsp*** sera placé dans **src/main/resources/META-INF/resources** ou **ailleurs** .
(NB : si projet sans springBoot , dans src/main/webapp).

Exemple élémentaire :

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HelloWorldController {

   @RequestMapping("/helloWorld")
   public String helloWorld(Model model) {
      model.addAttribute("message", "Hello World!");
      return "showMessage";
   }

}
```

Au niveau de **/jsp/**showMessage**.jsp**, l'affichage de message pourra être effectué via **${message}**.

```
<html>
<head><title>showMessage</title></head>
<body>
   <p>message=<b>${message}</b></p>
</body>
</html>
```

III - Api REST via spring-Mvc et OpenApiDoc

1. Web services "REST" pour application Spring

Pour développer des Web Services "REST" au sein d'une application Spring , il y a deux possibilités distinctes (à choisir) :

- s'appuyer sur l'API standard **JAX-RS** et choisir une de ses implémentations (**CXF3** ou **Jersey** ou …)
- s'appuyer sur le framework "**Spring web mvc**" et utiliser **@RestController** .

La version "JAX-RS standard" nécessite pas mal de librairies (jax-rs, jersey ou cxf , jackson et tout un tas de dépendances indirectes) .

La version spécifique spring nécessite un peu moins de librairies (spring-web , spring-mvc , jackson) et s'intègre mieux dans un écosystème spring (spring-security , ...) .

Exemples de dépendances "maven" sans spring-boot :

```
        <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-webmvc</artifactId>
                        <version>5.2.3.RELEASE</version>
                        <scope>compile</scope>
                </dependency>

                <dependency>
                        <groupId>com.fasterxml.jackson.core</groupId>
                        <artifactId>jackson-databind</artifactId>
                        <version>2.10.2</version> <!-- to produces json -->
                </dependency>
...
```

Dépendances "maven" indirecte (avec spring-boot) :

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

Dans ***application.properties*** :

server.servlet.context-path=/***webappXy*** *ou ...*

server.port=***8181*** ou ***8080*** ou ...

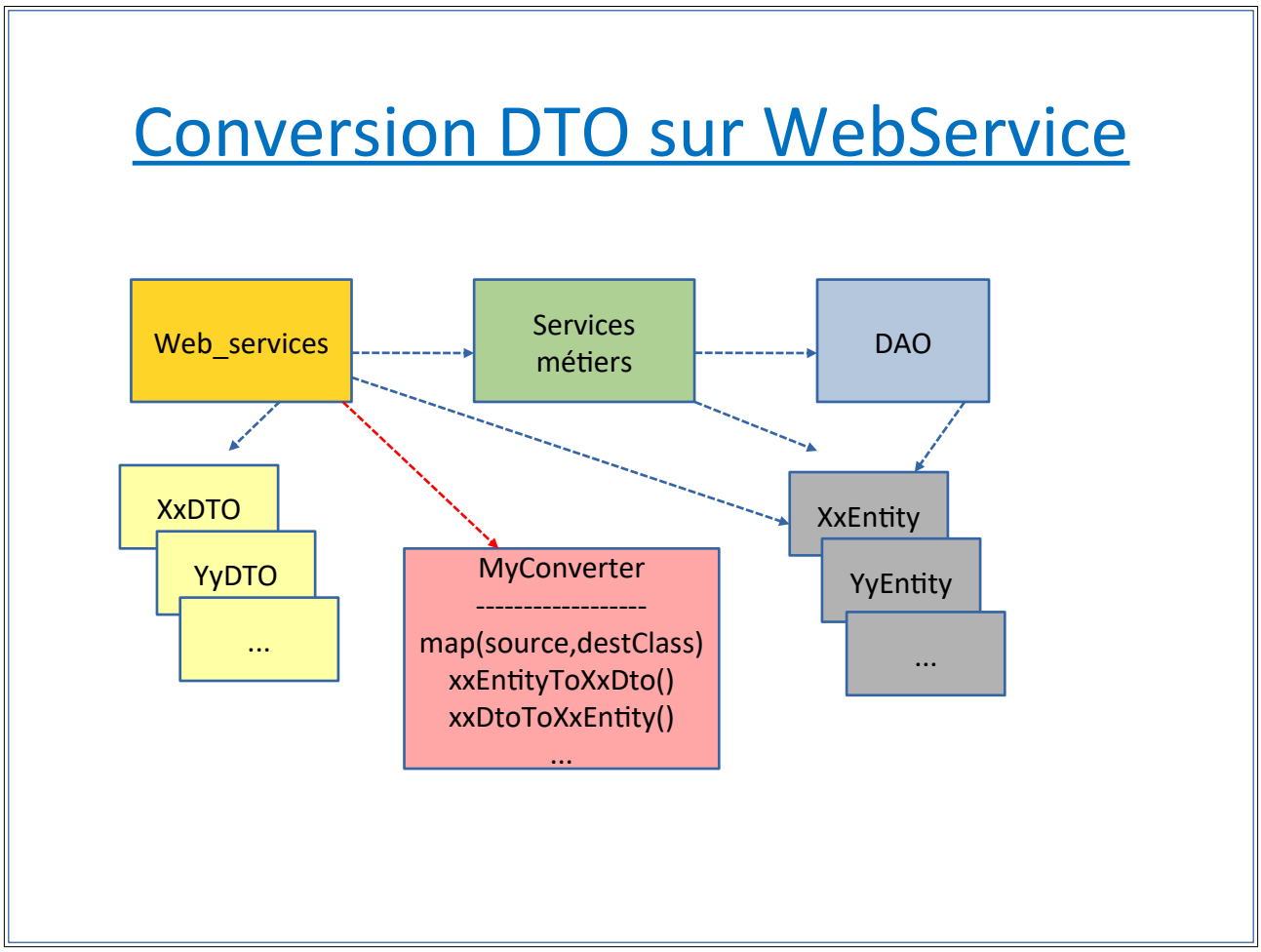
2. Variantes d'architectures

Un service web doit absolument retourner des structures de données stables (DTO = Data Transfert Object) idéalement indépendantes des structures des entités persistantes de manière à limiter les adhérences entre couches logicielles et garantir une bonne évolutivité .

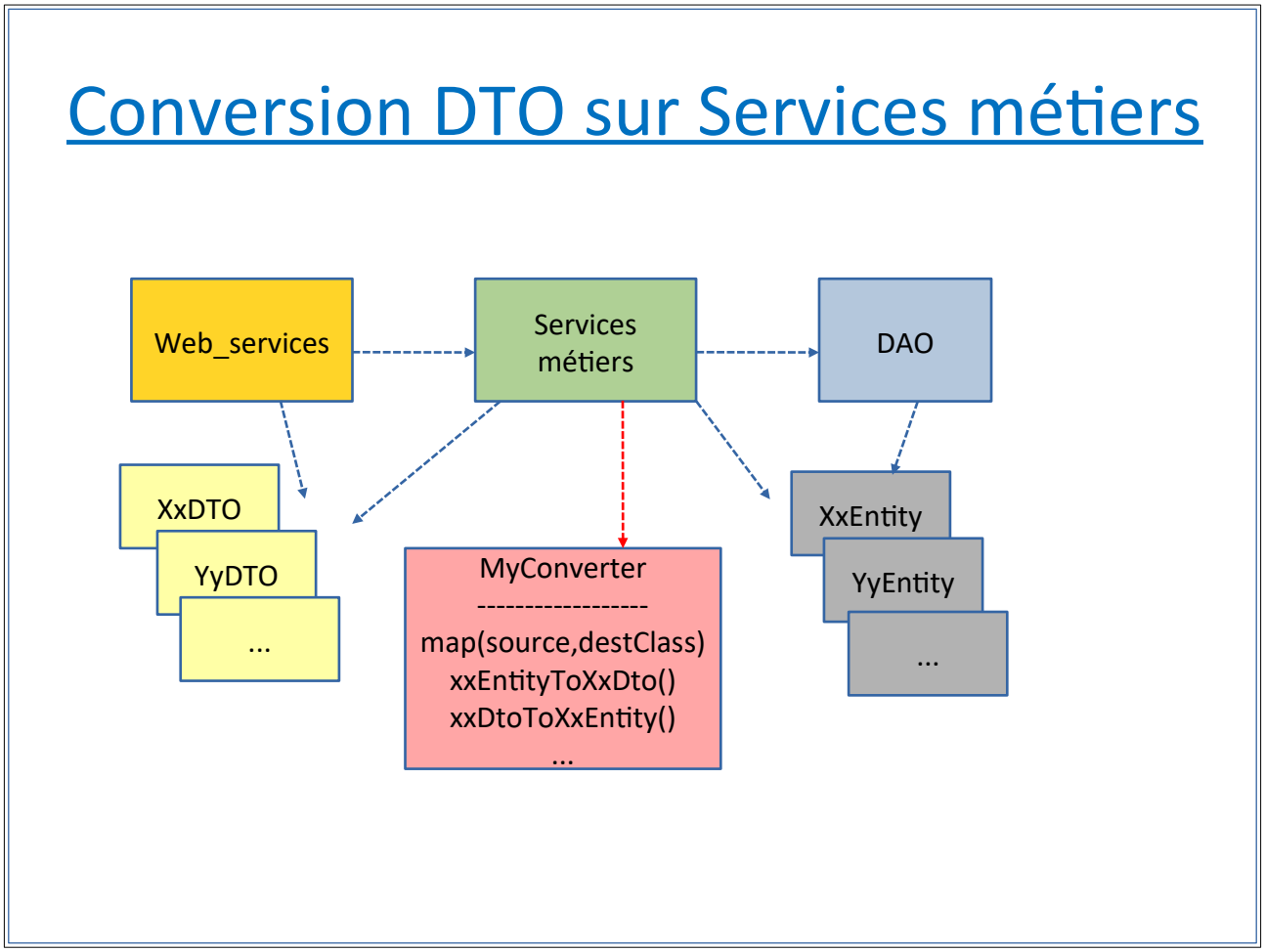
Le code des conversions entre DTO et entités persistantes s'effectue souvent au sein d'un composant utilitaire "Converter" ou "ModelMapper" .

Ce convertisseur peut être codé de des manières suivantes :
- entièrement manuellement (méthodes xxEntityToXxDto() , ….)
- en s'appuyant sur la très bonne technologie "**MapStruct**" (très performante) décrite en annexes
- en s'appuyant sur BeanUtils.copyProperties()
- en s'appuyant sur la technologie "modelMapper" (moins performante mais plus générique)
- sur une combinaison "maison" des technologies précédentes

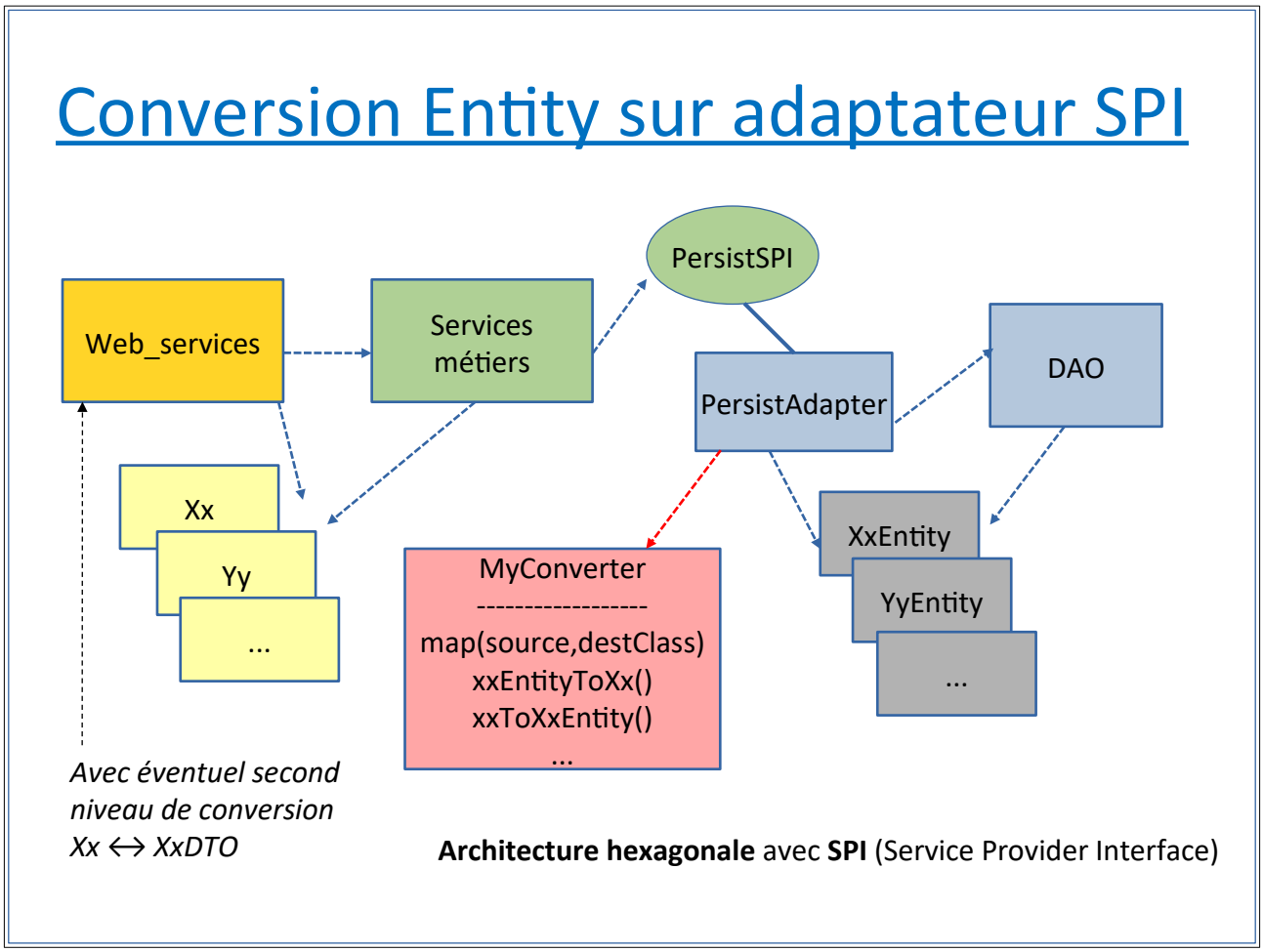
Le déclenchement des conversions peut être opéré/déclenché à différents niveaux :
- au niveau des web services
- au niveau des services internes
- au niveau des adaptateurs de persistance (dans le cadre d'une architecture hexagonale)
- éventuellement à plusieurs niveaux



pour cas "extrêmement simples" seulement.



Lorsque la conversion est effectuée en mode "n-tiers" au niveau des services métiers internes, alors :
- l'**interface du service** expose des **DTO** (ou autre) au niveau des paramètres d'entrée et au niveau des valeurs de retour des méthodes publiques
- le code interne du service métier (souvent préfixé par @Service) utilise des éléments **privés** de type **"DAO/repository"** et "**Entity**" avec des **conversions** à déclencher .



Le schéma ci-dessus correspond à une vision extrêmement simplifiée de l'architecture hexagonale (complexe et facultative).
Au sein de cette architecture , on a :
- en plein milieu la zone "domaine métier" (avec ses données métiers "Xx" , "Yy" et ses services métiers)
- en périphérie des zones "d'infrastructures" (persistance , échanges "kafka" , ….) qui doivent s'adapter aux interfaces entrantes et sortantes (SPI) de la zone centrale "domaine métier" .
Dans un tel cadre , les adaptateurs de persistance peuvent prendre en charge des conversions entre "Xx" et "XxEntity" et la partie "Api REST" peut soit réutiliser les classes "Xx" telles quelles si elles semblent très stables ou bien opérer un second niveau de conversion "Xx" vers "XxDTO" .

NB :

- Tous les schémas précédents sont eux-même sujets à de multiples variantes (design-patterns aux multiples implémentations possibles, …).
  Sources de variantes : "record" ou "lombok" , généricité/héritage , etc ...
- Comme souvent un bon compromis "simplicité/fonctionnalités" doit être déterminé en fonction de la taille et de la complexité de l'application (KISS : Keep It Simple Stupid , …)

3. WS REST via Spring MVC et @RestController

L'annotation fondamentale **@RestController** (héritant de @Controller et de @Component) déclare que la classe ....*RestCtrl* correspond à l'implémentation "spring-mvc" d'un composant de l'application de type "Contrôleur de Web Service REST" .

On a par défaut *@ResponseBody* avec *@RestController* et cela signifie que la valeur de retour d'une des méthodes publiques du contrôleur sera quasi directement renvoyée au client http (sans passer par une page JSP ni un autre type de vue) .

Cependant , Lorsque la valeur de retour sera un ***objet java , celui ci sera automatiquement transformé en JSON*** (ou autre) avant d'être retourné au client http (ex : code js / appel ajax)

3.1. Gestion des requêtes en lecture (mode GET)

Exemple :

DeviseJsonRestCtrl.java

```
package tp.app.zz.web.rest;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;
...
@RestController
@RequestMapping(value="/api-rest/devise" , headers="Accept=application/json")
public class DeviseJsonRestCtrl {

@Autowired //ou @Inject
private ServiceDevises serviceDevises; //internal business service or DAO

//RECHERCHE UNIQUE selon RESOURCE-ID:
//URL de déclenchement: .../webappXy/api-rest/devise/EUR
@GetMapping("/{codeDevise}" )
 public Devise getDeviseByCode(@PathVariable("codeDevise") String  codeDevise) {
        return serviceDevises.getDeviseByCode(codeDevise);
}

//RECHERCHE MULTIPLE :
//URL de déclenchement: webappXy/api-rest/devise
                        //ou webappXy/api-rest/devise?changeMini=1
@GetMapping("" )
public List<Devise> getDevisesByCriteria(
     @RequestParam(value="changeMini",required=false) Double changeMini) {
        if(changeMini==null)
                return serviceDevises.getAllDevises();
        else
                return serviceDevises.getDevisesByChangeMini(changeMini);
  }
}
```

NB :

@RequestParam avec required=false si paramètre facultatif en fin d'URL

Si l'ensemble de la classe java préfixée par @RestController comporte

```
@RequestMapping(value="...." , headers="Accept=application/json")
```

alors par défaut les valeurs en retour des méthodes publiques préfixées par **@RequestMapping** seront automatiquement converties au format **JSON** (en s'appuyant en interne sur la technologie *jackson-databind*) .

Techniquement possible mais très rare : retour direct d'une simple "String' (text/plain) :

```
//URL : webappXy/api-rest/devise/convert?amount=50&src=EUR&target=USD
@RequestMapping(value="/convert" , method=RequestMethod.GET ,
                        headers="Accept=text/plain")
//@ResponseBody  par défaut avec @RestController
 String convert(@RequestParam("amount") double  amount,
                @RequestParam("src") String  src ,
                @RequestParam("target") String  target) {
        double sommeConvertie=gestionDevises.convertir(amount, src, target);
        System.out.println("sommeConvertie="+sommeConvertie);
        return String.valueOf(sommeConvertie);
}
```

==> L'exemple ci-dessus est très déconseillé sur une api REST .
Un format de retour homogène (XML ou très souvent JSON) est en général attendu à la place .

3.2. modes "PUT" , "POST" , "DELETE" et ReponseEntity<T>

NB : il est techniquement possible de convertir explicitement une "Json String" en objet java via l'api "jackson" comme le montre l'exemple inutilement long suivant (**à ne pas reproduire , juste pour montrer certains mécanismes internes**):

```
...
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMethod;
import com.fasterxml.jackson.databind.DeserializationFeature;
import com.fasterxml.jackson.databind.ObjectMapper;

@RestController
@RequestMapping(value="/api-rest/devises" , headers="Accept=application/json")
public class DeviseJsonRestCtrl {
...
 @PutMapping("")
Devise updateDevise(@RequestBody String deviseAsJsonString) {
        Devise devise=null;
        try {
                ObjectMapper jacksonMapper = new ObjectMapper();
                jacksonMapper.configure(
                        DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES, false);
                devise = jacksonMapper.readValue(deviseAsJsonString,Devise.class);
                System.out.println("devise to update:" + devise);
                serviceDevises.updateDevise(devise);
                return devise;
```

```
        } catch (Exception e) {
                e.printStackTrace();
                return null;
        }
} ….}
```

Ceci dit , **Spring-Mvc est capable d'effectuer de lui même automatiquement cette conversion.**

L'écriture suivante (plus simple, à reproduire) assure les mêmes fonctionnalités :

```
@RestController
@RequestMapping(value="/api-rest/devise" , headers="Accept=application/json")
public class DeviseJsonRestCtrl {
...
 @PutMapping("" )
Devise updateDevise(@RequestBody  Devise devise) {
        System.out.println("devise to update:" + devise);
        serviceDevises.updateDevise(devise);
                return devise;
} ….
}
```

NB : dans tous les cas , il sera souvent nécessaire de contrôler le comportement des "sérialisations/dé-sérialisations java <--> json" en incorporant certaines annotations de "jackson" au sein des classes de données (dto / payload) à véhiculer.

A ce sujet , l'annotation **@JsonIgnore** (sémantiquement équivalent à @XmlTransient) peut quelquefois être utile pour limiter la profondeur des données échangées .

Apport important de la version 4 : **ResponseEntity<T>**

Depuis "Spring4" , une méthode d'un web-service REST peut éventuellement retourner une réponse de Type **ResponseEntity<T>** ce qui permet de **retourner d'un seul coup**:

- **un statut** (OK , NOT_FOUND , ...)
- **le corps de la réponse** : objet (ou liste) T convertie en json
- **un éventuel "header"** (ex: url avec id si auto_incr lors d'un POST)

Exemple:

```
@GetMapping("/{codeDev}" )
 ResponseEntity<Devise> getDeviseByName(@PathVariable("codeDev") String  codeDevise) {
        Devise dev = gestionDevises.getDeviseByPk(codeDevise);
        if(dev!=null)
            return new ResponseEntity<Devise>(dev, HttpStatus.OK);
        else
            return new ResponseEntity<Devise>(HttpStatus.NOT_FOUND);//404
}
```

ou bien

```
ResponseEntity< ?> getDeviseByName(....){
```

```
....
        else
    return new ResponseEntity<String> ("{ \"err\" : \"devise not found\"}",
                                            HttpStatus.NOT_FOUND) ;//404
}
```

Autre exemple (ici en mode **DELETE**) :

```
//url : http://localhost:8181/webappXy/api-rest/devise/EUR
@DeleteMapping("/{codeDev}")
public ResponseEntity< ?> deleteDeviseByCode(@PathVariable("codeDev")String codeDevise){
        try {
                deviseDao.deleteDeviseBycode(codeDevise);
                return new ResponseEntity< ?>(HttpStatus.OK);
        } catch (Exception e) {
                e.printStackTrace(); //ou logger.error(e) ;
                return new ResponseEntity< ?>(HttpStatus.NOT_FOUND);
                                        //ou HttpStatus.INTERNAL_SERVER_ERROR
                }
}
```

NB : Bien que très finement paramétrable , un **return new ResponseEntity<?>** **sera généralement moins bien** **qu'un** un simple **throw new ...ClasseException gérée par un ResponseEntityExceptionHandler plus simple et plus efficace (ce sera vu dans un paragraphe ultérieur)**

Eventuelles variations (équivalences):

@GetMapping(...) est équivalent à @RequestMapping(... , **method=RequestMethod.GET**)
@PostMapping(...) est équivalent à @RequestMapping(... , **method=RequestMethod.POST**)
@PutMapping(...) est équivalent à @RequestMapping(... , **method=RequestMethod.PUT**)
@DeleteMapping(...) équivalent à @RequestMapping(..., **method=RequestMethod.DELETE**)

3.3. Réponse et statut http par défaut en cas d'exception

Si une méthode d'un contrôleur REST remonte une exception java qui n'est pas rattrapée par un try/catch , la technologie Spring-Mvc retourne alors une réponse et un statut HTTP par défaut :

```
{ "timestamp" : 152....56,
"status" : 500 ,
"error" : "Internal Server Error",
"exception" : "java.lang.NullPointerException",
"message" : "......",
"path" : "/rest/devise/67573567" }
```

Le statut HTTP retourné par défaut dans l'entête de la réponse en cas d'exception est généralement **500** (INTERNAL_SERVER_ERROR) .

3.4. @ResponseStatus

Dans le cadre d'une remontée d'exception personnalisée il est possible de préciser le statut HTTP (pas systématiquement 500) qui sera remonté via l'annotation **@ResponseStatus()**

Exemple :

```
@ResponseStatus(HttpStatus.NOT_FOUND)  //404
public class MyEntityNotFoundException extends RuntimeException{
        public MyEntityNotFoundException() {
        }
        public MyEntityNotFoundException(String message) {
                super(message);
        }
        public MyEntityNotFoundException(Throwable cause) {
                super(cause);
        }
        public MyEntityNotFoundException(String message, Throwable cause) {
                super(message, cause);
        }
...
}
```

.../...

```
@RequestMapping(value="/{codeDevise}" , method=RequestMethod.DELETE)
	public void deleteDeviseByCode(@PathVariable("codeDevise")
					String codeDevise) throws MyEntityNotFoundException{
	try {
		deviseService.deleteByCode(codeDevise);
	} catch (Exception e) {
		logger.error(e.getMessage());
		throw new MyEntityNotFoundException(
			"echec suppresssion devise pour codeDevise="+codeDevise ,e);
		}
	}
```

Un appel HTTP avec une URL finissant (avec une erreur ici volontaire) par "/devise/**EURy**"

---> renvoie **404** et un **message d'erreur au format JSON/spring-Web-MCV HOMOGENE :**

```
{
   "timestamp": "2020-02-03T17:23:45.888+0000",
   "status": 404,
   "error": "Not Found",
   "message": "echec suppresssion devise pour codeDevise=EURy",
   "trace": "org.mycontrib.backend.exception.MyEntityNotFoundException:.....",
   "path": "/spring-boot-backend/rest/devise-api/private/role_admin/devise/EURy"
}
```

3.5. Validation des valeurs entrantes (@Valid)

Dans le cadre d'un échec de validation de la requête avec **@Valid** sur le paramètre d'entrée d'une méthode d'un contrôleur REST et avec des annotations de javax.validation (@Min , @Max , ...) sur la classe du "DTO" (ex : Devise) , le statut HTTP alors automatiquement remonté dans l'entête de la réponse HTTP est **400** (**Bad Request**) et le le corps de la réponse comporte tous les détails sur les éléments invalides .

```
public ResponseEntity<Void> ajouterDevise(@Valid @RequestBody Devise devise) {
....
}
```

```
public class Devise{
...
@Length(min=3, max=20, message = "Nom trop long ou trop court")
private String nom;
}
```

3.6. ResponseEntityExceptionHandler (très bien)

ApiError.java (DTO for custom error message)

```
package tp.appliSpring.dto;

import java.time.LocalDateTime;
import org.springframework.http.HttpStatus;
import com.fasterxml.jackson.annotation.JsonFormat;
import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

@Getter @Setter @ToString
public class ApiError {

  private HttpStatus status;
  @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "dd-MM-yyyy hh:mm:ss")
  private LocalDateTime timestamp;
  private String message;
  private String debugMessage;
  //private List<ApiSubError> subErrors;

  public ApiError() {
     timestamp = LocalDateTime.now();
  }

  public ApiError(HttpStatus status) {
     this();
     this.status = status;
  }

  public ApiError(HttpStatus status, Throwable ex) {
     this();
     this.status = status;
     this.message = "Unexpected error";
     this.debugMessage = ex.getLocalizedMessage();
  }

  public ApiError(HttpStatus status, String message, Throwable ex) {
     this();
     this.status = status;
     this.message = message;
     this.debugMessage = ex.getLocalizedMessage();
  }
}
```

RestResponseEntityExceptionHandler.java

```
package tp.appliSpring.web.rest;

import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatusCode;
import org.springframework.http.ResponseEntity;
import org.springframework.http.converter.HttpMessageNotReadableException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;
import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;
import tp.appliSpring.core.exception.ConflictException;
import tp.appliSpring.core.exception.NotFoundException;
import tp.appliSpring.dto.ApiError;

@ControllerAdvice
public class RestResponseEntityExceptionHandler
  extends ResponseEntityExceptionHandler {

        private ResponseEntity<Object> buildResponseEntity(ApiError apiError) {
            return new ResponseEntity<>(apiError, apiError.getStatus());
          }

        @Override
          protected ResponseEntity<Object>
        handleHttpMessageNotReadable(HttpMessageNotReadableException ex,
                HttpHeaders headers, HttpStatusCode status, WebRequest request) {
            String error = "Malformed JSON request";
            return buildResponseEntity(new ApiError(HttpStatus.BAD_REQUEST, error, ex));
          }

        @ExceptionHandler(NotFoundException.class)
          protected ResponseEntity<Object> handleEntityNotFound(
                NotFoundException ex) {
            return buildResponseEntity(new ApiError(HttpStatus.NOT_FOUND,ex));
          }
        @ExceptionHandler(ConflictException.class)
          protected ResponseEntity<Object> handleConflict(
                        ConflictException ex) {
            return buildResponseEntity(new ApiError(HttpStatus.CONFLICT,ex));
          }
}
```

Et grace à cela les exceptions java retournées par les services et contrôleurs REST :

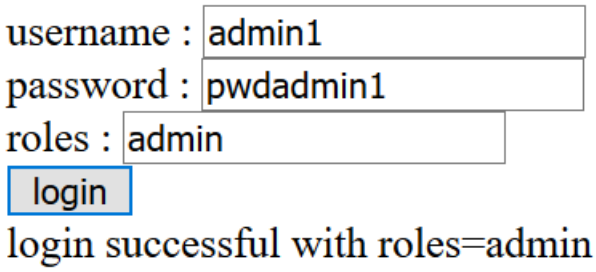
- n'ont plus besoin d'être décorées par @ResponseStatus → meilleurs séparation des couches
- seront automatiquement transformées en messages très personnalisés
  et accompagnés du bon statut HTTP .

3.7. Exemples d'appels en js/ajax

js/ajax-util.js

```
//fonction utilitaire pour preparer xhr en vu d'effectuer juste apres un appel ajax en mode Get ou post ou ...
function initXhrWithCallback(callback,errCallback){
	var xhr = new XMLHttpRequest();
	xhr.onreadystatechange = function() {
		if (xhr.readyState == 4){
			if (xhr.status == 200 || xhr.status == 0) {
				callback(xhr.responseText,xhr);
			}
			else {
				errCallback(xhr);
			}
		}
	};
	return xhr;
}

function xhrStatusToErrorMessage(xhr){
	var errMsg = "ajax error";//by default
	var detailsMsg=""; //by default
	console.log("xhr.status="+xhr.status);
	if(xhr.responseText!=null)
		detailsMsg = xhr.responseText;
	switch(xhr.status){
		case 400 :
			errMsg = "Server understood the request, but request content was invalid."; break;
		case 401 :
			errMsg = "Unauthorized access (401)"; break;
		case 403 :
			errMsg = "Forbidden resource can't be accessed (403)"; break;
		case 404 :
			errMsg = "resource not found (404)"; break;
		case 500 :
			errMsg = "Internal server error (500)"; break;
		case 503 :
			errMsg = "Service unavailable (503)"; break;
	}
	return errMsg+" "+detailsMsg;
}
```

```
function makeAjaxGetRequest(xhr,url) {
	xhr.open("GET", url, true);
	xhr.send(null);
}

function makeAjaxPostRequest(xhr,url,jsonData) {
	xhr.open("POST", url, true);
	xhr.setRequestHeader("Content-Type", "application/json");
	//pour re-vehiculer (si necessaire) un jeton d'authentification (jwt ou pas):
	var authToken = sessionStorage.getItem("authToken");
	if(authToken != null ){
		xhr.setRequestHeader('Authorization',"Bearer "+ authToken);
	}
	xhr.send(jsonData);
}
```



login.html

```
<html>
<head> <title>login</title><script src="js/ajax-util.js"></script>  <script src="js/login.js"></script>
</head>
<body>
  <h3> login (ws security) </h3>
  username : <input id="txtUsername" type='text' value="admin1"/><br/>
  password : <input id="txtPassword" type='text' value="pwdadmin1"/><br/>
  roles : <input id="txtRoles" type='text' value="admin"/><br/>
  <input type='button' value="login" id="btnLogin"/> <br/>
  <span id="spanMsg"></span> <br/>
 <hr/> <a href="index.html">retour vers index.html</a>
</body>
</html>
```

js/login.js

```
window.onload=function(){
	var spanMsg = document.querySelector('#spanMsg');
	var btnLogin=document.querySelector('#btnLogin');
	btnLogin.addEventListener("click" , function (){
		var auth = { username : null, password : null , roles : null } ;
		auth.username = document.querySelector('#txtUsername').value;
		auth.password = document.querySelector('#txtPassword').value;
		auth.roles = document.querySelector('#txtRoles').value;

		var cbLogin = function(data,xhr){
		  console.log(data); //data as json string;
		  var authResponse = JSON.parse(data);
		  if(authResponse.status){
			spanMsg.innerHTML=authResponse.message + " with roles=" + authResponse.roles;
			//localStorage.setItem("authToken",authResponse.token);
			sessionStorage.setItem("authToken",authResponse.token);
		  }else{
			spanMsg.innerHTML=authResponse.message ;
		  }
		}//end of cbLogin

		var cbError = function(xhr){
			spanMsg.innerHTML= xhrStatusToErrorMessage(xhr)  ;
		}

		var xhr = initXhrWithCallback(cbLogin,cbError);
		makeAjaxPostRequest(xhr,"./api-rest/login-api/public/auth" , JSON.stringify(auth));

	});//end of btnLogin.addEventListener/click
}//end of window.onload
```

recherche devises selon taux mini (public)

changeMini : 1
getDevises

- Euro , 1
- Dollar , 1.1243
- Yen , 121.6477

ajout de monnaie (after logging as ADMIN)

codeMonnaie: ms (ex: EUR,USD,...)
nommonnaie: monnaieSinge (ex: euro,dollar,...)
tauxChange: 1.23456 (ex: 1, 0.85 , 1.5, ...)
sauvegarder devise
{"code":"ms","name":"monnaieSinge","change":1.23456}

appel_ajax.html

```
<html>
<head>
        <script src="js/ajax-util.js"></script>     <script src="js/appelAjax.js"></script>
        <meta charset="UTF-8"> <title>appel_ajax</title>
</head>
<body>
   <h3>recherche devises selon taux mini (public)</h3>
   changeMini : <input type="text" id="txtChangeMini" value="1"/> <br/>
                <input type="button" value="getDevises" id="btnGetDevises" /> <br/>
        <div id="divRes"></div>

   <h3> ajout de monnaie (after logging as ADMIN)</h3>
   codeMonnaie: <input type="text" id="txtCode" value="ms" /> (ex: EUR,USD,...)<br/>
   nommonnaie: <input type="text" id="txtName" value="monnaieSinge" /> (ex: euro,dollar,...)<br/>
   tauxChange: <input type="text" id="txtChange" value="1.23456" /> (ex: 1, 0.85 , 1.5, ... )<br/>
   <input type="button" id="btnPostDevise" value="sauvegarder devise" /> <br/>
   <div id="divMessage"></div>
   <hr/>
   <a href="index.html">retour index.html</a>
</body>
</html>
```

js/appelAjx.js

```
window.onload=function(){
	var inputChangeMini = document.querySelector("#txtChangeMini");
	var btnGetDevises = document.querySelector("#btnGetDevises");
	var btnPostDevise = document.querySelector("#btnPostDevise");
	var divRes = document.querySelector("#divRes");
	var divMessage = document.querySelector("#divMessage");
	var cbError = function(xhr){
		divMessage.innerHTML= xhrStatusToErrorMessage(xhr)  ;
	}
	btnGetDevises.addEventListener("click" , function (){
		var changeMini = inputChangeMini.value;
		var cbAffDevises=function(texteReponse,xhr){
			//divRes.innerHTML = texteReponse;
			var listeDeviseJs = JSON.parse(texteReponse /* au format json string */)
			var htmlListeDevises = "<ul>" ;
			for(i=0; i<listeDeviseJs.length ; i++){
				htmlListeDevises = htmlListeDevises + "<li>" + listeDeviseJs[i].name + " , "
					+ listeDeviseJs[i].change + "</li>";
			}
			htmlListeDevises = htmlListeDevises + "</ul>";
			divRes.innerHTML= htmlListeDevises;
		}
		var xhr = initXhrWithCallback(cbAffDevises , cbError);
		makeAjaxGetRequest(xhr,"./api-rest/devise-api/public/devise?changeMini="+changeMini );
	});//end of btnGetDevises.addEventListener/"click"

	btnPostDevise.addEventListener("click" , function (){
		var nouvelleDevise =  {    code : null,    name : null, change : null       };
		nouvelleDevise.code = document.querySelector("#txtCode").value;
		nouvelleDevise.name = document.querySelector("#txtName").value;
		nouvelleDevise.change = document.querySelector("#txtChange").value;
		var cbGererResultatPostDevise = function (texteReponse,xhr){
			divMessage.innerHTML= texteReponse;
		}
		var xhr = initXhrWithCallback(cbGererResultatPostDevise, cbError);
		makeAjaxPostRequest(xhr,"./api-rest/devise-api/private/role_admin/devise" ,
				JSON.stringify(nouvelleDevise));
	});//end of btnGetDevises.addEventListener/"click"
} //end of window.onload
```

3.8. Invocation java de service REST via RestTemplate de Spring

Utile pour une **délégation de service** ou bien pour un **test d'intégration** (automatisable via maven et intégration continue).

```
.....
import org.junit.Assert;
import org.junit.BeforeClass;
import org.junit.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.web.client.RestTemplate;

/ * cette classe à un nom qui commence ou se termine par IT (et par par Test)
 * car c'est un Test d'Integration qui ne fonctionne que lorsque toute l'application
 * est entièrement démarrée (avec EmbeddedTomcat ou équivalent) .*/
public class PersonWsRestIT {

	private static Logger logger = LoggerFactory.getLogger(PersonWsRestIT.class);

	private static RestTemplate restTemplate; //objet technique de Spring pour test WS REST

	//pas de @Autowired ni de @RunWith
	//car ce test EXTERNE est censé tester le WebService sans connaître sa structure interne
	// (test BOITE_NOIRE)
	@BeforeClass
	public static void init(){
		restTemplate = new RestTemplate();
	}

	@Test
	public void testGetSpectacleById(){
		final String BASE_URL =
			"http://localhost:8888/spring-boot-spectacle-ws/spectacle-api/public";
		final String uri = BASE_URL + "/spectacle/1";
		String resultAsJsonString = restTemplate.getForObject(uri, String.class);
		logger.info("json string of spectacle 1 via rest: " + resultAsJsonString);
```

```
            Spectacle s1 = restTemplate.getForObject(uri, Spectacle.class);
            logger.info("spectacle 1 via rest: " + s1);
            Assert.assertTrue(s1.getId()==1L);
    }

    @Test
    public void testListeComptesDuClient(){
       final String villeDepart = "Paris";
       final String dateDepart = "2018-09-20";
       final String uri = "http://localhost:8080/flight_web/mvc/rest/vols/byCriteria"
                  +"?villeDepart=" + villeDepart + "&dateDepart=" + dateDepart;
       String resultAsJsonString = restTemplate.getForObject(uri, String.class);
       logger.info("json listeVols via rest: " + resultAsJsonString);
       Vol[] tabVols = restTemplate.getForObject(uri,Vol[].class);
       logger.info("java listeComptes via rest: "  +tabVols.toString());
       Assert.assertNotNull(tabVols);   Assert.assertTrue(tabVols.length>=0);
       for(Vol cpt : tabVols){
            System.out.println("\t" + cpt.toString());
       }
    }

    @Test
    public void testVirement(){
            final String uri =
                  "http://localhost:8080/tpSpringWeb/mvc/rest/compte/virement";
              //post/envoi:
              OrdreVirement ordreVirement = new OrdreVirement();
              ordreVirement.setMontant(50.0);
              ordreVirement.setNumCptDeb(1L);
              ordreVirement.setNumCptCred(2L);
              OrdreVirement savedOrdreVirement =
                    restTemplate.postForObject(uri, ordreVirement, OrdreVirement.class);
              logger.info("savedOrdreVirement via rest: " + savedOrdreVirement.toString());
              Assert.assertTrue(savedOrdreVirement.getOk().equals(true));
    }

}
```

Exemple 2 (délégation de service) :

```
...
import java.nio.charset.Charset;
import java.util.Base64;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.util.LinkedMultiValueMap;
import org.springframework.util.MultiValueMap;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@RestController
@RequestMapping(value="/myapi/auth" ,    headers="Accept=application/json")
public class LoginDelegateCtrl {

        private static Logger logger = LoggerFactory.getLogger(LoginDelegateCtrl.class);

        private static final String ACCESS_TOKEN_URL =
                                        "http://localhost:8081/basic-oauth-server/oauth/token";

        private static RestTemplate restTemplate = new RestTemplate();

        HttpHeaders createBasicHttpAuthHeaders(String username, String password){
                HttpHeaders headers = new HttpHeaders();
                headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
                String auth = username + ":" + password;
```

```
            byte[] encodedAuth = Base64.getEncoder().encode(
                                        auth.getBytes(Charset.forName("US-ASCII")) );
            String authHeader = "Basic " + new String( encodedAuth );
            headers.add("Authorization", authHeader);
            return headers;
            }

    @PostMapping("/login")
     public ResponseEntity<?> authenticateUser(@RequestBody AuthRequest loginRequest) {
    logger.debug("/login , loginRequest:"+loginRequest);
    String authResponse="{}";
    try{
    MultiValueMap<String, String> params= new LinkedMultiValueMap<String,
String>();
    params.add("username", loginRequest.getUsername());
    params.add("password", loginRequest.getPassword());
    params.add("grant_type", "password");
    //ResponseEntity<String> tokenResponse =
    //                restTemplate.postForEntity(ACCESS_TOKEN_URL,params, String.class);
    // si pas besoin de spécifier headers spécifique .

    HttpHeaders headers = createBasicHttpAuthHeaders("fooClientIdPassword","secret");
    HttpEntity<MultiValueMap<String, String>> entityReq =
                new HttpEntity<MultiValueMap<String, String>>(params, headers);

    ResponseEntity<String> tokenResponse=
                restTemplate.exchange(ACCESS_TOKEN_URL,
                                        HttpMethod.POST,
                                        entityReq,
                                        String.class);
    authResponse=tokenResponse.getBody();
    logger.debug("/login authResponse:" + authResponse.toString());
    return ResponseEntity.ok(authResponse);
    }
    catch (Exception e) {
      logger.debug("echec authentification:" + e.getMessage()); //for log
     return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
                                    .body(authResponse);
```

```
        }
    }
}
```

3.9. Appel moderne/asynchrone de WS-REST avec WebClient

```
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-webflux</artifactId>
            <!-- pour appel de WS-REST externes , WebClient mieux que RestTemplate -->
        </dependency>
```

RestClientApp.java

```
package tp.appliSpring.client;

import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.web.reactive.function.client.WebClient;
import reactor.core.publisher.Mono;
import tp.appliSpring.dto.Currency;   import tp.appliSpring.dto.LoginRequest;
import tp.appliSpring.dto.LoginResponse;

public class RestClientApp {

public static String token="?";

public static void main(String[] args) {
                postLoginForToken();
                posterNouvelleDevise();
        }

private static void postLoginForToken() {
        WebClient.Builder builder = WebClient.builder();
        String baseUrl="http://localhost:8080/appliSpring/api-bank";
        WebClient webClient = builder
           .baseUrl(baseUrl)
           .defaultHeader(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE)
           .build();


        LoginRequest loginRequest = new LoginRequest("admin1","pwd1");

        //envoyer cela via un appel en POST
        Mono<LoginResponse> reactiveStream = webClient.post().uri("/public/login")
                .body(Mono.just(loginRequest), LoginRequest.class)
                .retrieve()
                .bodyToMono(LoginResponse.class)
                .onErrorReturn(new LoginResponse("admin1",false,"login failed",null));
        LoginResponse loginResponse = reactiveStream.block();
```

```
        System.out.println("loginResponse=" + loginResponse.toString());
        if(loginResponse.getOk())
                token = loginResponse.getToken();
}

private static void posterNouvelleDevise() {
        WebClient.Builder builder = WebClient.builder();
        String baseUrl="http://localhost:8080/appliSpring/api-bank";
        WebClient webClient = builder
          .baseUrl(baseUrl)
          .defaultHeader(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE)
          .defaultHeader(HttpHeaders.AUTHORIZATION, "Bearer " + token)
          .build();

        //créer une instance du DTO Currency
        //avec les valeurs
        //{ "code" : "DDK" , "name" : "couronne danoise" , "rate" : 7.77 }

        Currency currencyDDK = new Currency("DDK","couronne danoise" , 7.77);

        //envoyer cela via un appel en POST
        Mono<Currency> reactiveStream = webClient.post().uri("/devise")
                .body(Mono.just(currencyDDK), Currency.class)
                .retrieve()
                .bodyToMono(Currency.class)
                .onErrorReturn(new Currency("?","not saved !!",0.0));

        Currency savedCurrency = reactiveStream.block();

        System.out.println("savedCurrency=" + savedCurrency.toString());
        }
}
```

Variantes pour appel(s) en mode GET :

```
private String tempApiKey="26ca93ee7fc19cbe0a423aaa27cab235";
private String fixerApiUrl="http://data.fixer.io/api/latest"
      +"?access_key="+tempApiKey; //apiKey may be passed in header with other api

/*
Mono<String> reactiveStream = webClient.get()
.retrieve()
.bodyToMono(new ParameterizedTypeReference<String>() {});
String result = reactiveStream.block();
System.out.println("result="+result);
*/
//type de réponse brute attendue:
/*
{"success":true,"timestamp":1635959583,"base":"EUR","date":"2021-11-03",
"rates":{"AED":4.254663,"AFN":105.467869,..., "EUR":1 , ...}}
*/
Mono<FixerIoResponse> reactiveStream = webClient.get() //.uri("/suiteUrlQuiVaBien")
                                .retrieve()
                                .bodyToMono(new ParameterizedTypeReference<FixerIoResponse>() {});
FixerIoResponse fixerIoResponse = reactiveStream.block();
```

```
/*
ResponseEntity<FixerIoResponse> fixerIoResponseEntity=
                webClient.get().retrieve()
                .toEntity(FixerIoResponse.class).block();

FixerIoResponse fixerIoResponse = null;
if(fixerIoResponseEntity.getStatusCode()==HttpStatus.OK) {
    fixerIoResponse=fixerIoResponseEntity.getBody();
}
*/
```

3.10. Test d'un "RestController" via MockMvc

Pour tester le comportement d'un composant "RestController" de Spring-Mvc sans avoir à préalablement démarrer l'application complète, on peut utiliser la classe **MockMvc** et l'annotation **@WebMvcTest** ou bien **@AutoConfigureMockMvc** qui sont spécialement prévues pour faire fonctionner le code d'un web service rest de spring-mvc en recréant un contexte local ayant à peu près de même comportement que celui d'un conteneur web mais sans accès réseau/http .

Deux Grandes Variantes :

- via **@WebMvcTest** : **test unitaire** avec mock de service interne
- via **@SpringBootTest** et **@AutoConfigureMockMvc : test d'intégration avec réels services**

3.11. Test unitaire de contrôleur Rest

```
package tp.appliSpring.rest;

import static org.hamcrest.Matchers.hasSize;
import static org.hamcrest.Matchers.is;
import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import java.util.ArrayList;          import java.util.List;
import org.junit.jupiter.api.BeforeEach;     import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;      import org.mockito.Mockito;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.http.MediaType;
import org.springframework.test.context.junit.jupiter.SpringExtension;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.MvcResult;
import tp.appliSpring.entity.Compte;     import tp.appliSpring.service.CompteService;

@ExtendWith(SpringExtension.class) //si junit5/jupiter
@WebMvcTest(CompteRestCtrl.class)
//NB: @WebMvcTest without security and without service layer , service must be mocked !!!
public class TestCompteRestCtrlWithServiceMock {
```

```
	@Autowired
	private MockMvc mvc;


	@MockBean
	private CompteService compteService; //not real implementation but mock to configure .

	@BeforeEach
	public void reInitMock() {
		//vérification que le service injecté est bien un mock
		assertTrue(Mockito.mockingDetails(compteService).isMock());
		//reinitialisation du mock(de scope=Singleton par defaut) sur aspects stub et spy
		Mockito.reset(compteService);
	}


	@Test //à lancer sans le profile withSecurity
	public void testComptesDuClient1WithMockOfCompteService(){

	//préparation du mock (qui sera utilisé en arrière plan du contrôleur rest à tester):
	List<Compte> comptes = new ArrayList<>();
	comptes.add(new Compte(1L,"compteA",40.0));
	comptes.add(new Compte(2L,"compteB",90.0));
	Mockito.when(compteService.comptesDuClient(1)).thenReturn(comptes);

	try {
		MvcResult mvcResult =
		mvc.perform(get("/api-bank/compte?numClient=1")
		.contentType(MediaType.APPLICATION_JSON))
		.andExpect(status().isOk())
		.andExpect(jsonPath("$", hasSize(2) ))
		.andExpect(jsonPath("$[0].label", is("compteA") ))
		.andExpect(jsonPath("$[1].solde", is(90.0) ))
		.andReturn();
		System.out.println(">>>>>>>>>> jsonResult="
				+mvcResult.getResponse().getContentAsString());
	} catch (Exception e) {
		System.err.println(e.getMessage());
	}
	}
}
```

NB : Spring5 propose une variante @WebFluxTest et WebTestClient pour WebFlux .

3.12. Test d'intégration de contrôleur Rest avec réels services

```
package tp.appliSpring.rest;
import static org.hamcrest.Matchers.hasSize;
import static org.hamcrest.Matchers.is;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.MediaType;
import org.springframework.test.context.ActiveProfiles;
import org.springframework.test.context.junit.jupiter.SpringExtension;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.MvcResult;

@ExtendWith(SpringExtension.class) //si junit5/jupiter
@SpringBootTest //with all layers
@AutoConfigureMockMvc //to test controller with reals spring services implementations
@ActiveProfiles({"embbededDb","init"}) //init profile for ...init.ReinitDefaultDataSet
public class TestCompteRestCtrlWithRealService {

	@Autowired
	private MockMvc mvc;


	@Test //à lancer sans le profile withSecurity
	public void testComptesDuClient1WithRealService(){
		try {
			MvcResult mvcResult =
			mvc.perform(get("/api-bank/compte?numClient=1")
			.contentType(MediaType.APPLICATION_JSON))
			.andExpect(status().isOk())
			.andExpect(jsonPath("$", hasSize(2) ))
			.andExpect(jsonPath("$[0].label", is("compteA") ))
			.andReturn();
			//à adapter selon jeux de données de init.ReInitDefaultDataset
			System.out.println(">>>>>>>>>> jsonResult="+
					mvcResult.getResponse().getContentAsString());
		} catch (Exception e) {
			System.err.println(e.getMessage());
			//e.printStackTrace();
		}
	}
}
```

4. Config swagger3 / openapi-doc pour spring

Ancienne version à ne pas ajouter dans pom.xml

```
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger2</artifactId>
  <version>2.9.2</version>
</dependency>

 <dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger-ui</artifactId>
  <version>2.9.2</version>
</dependency>
```

Version plus récente à ajouter dans pom.xml

```
<!-- springdoc-openapi-ui for spring5/springBoot2 ,
     springdoc-openapi-starter-webmvc-ui for spring6/springBoot3 -->
<dependency>
        <groupId>org.springdoc</groupId>
        <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
        <version>2.3.0</version>
</dependency>
```

en plus de

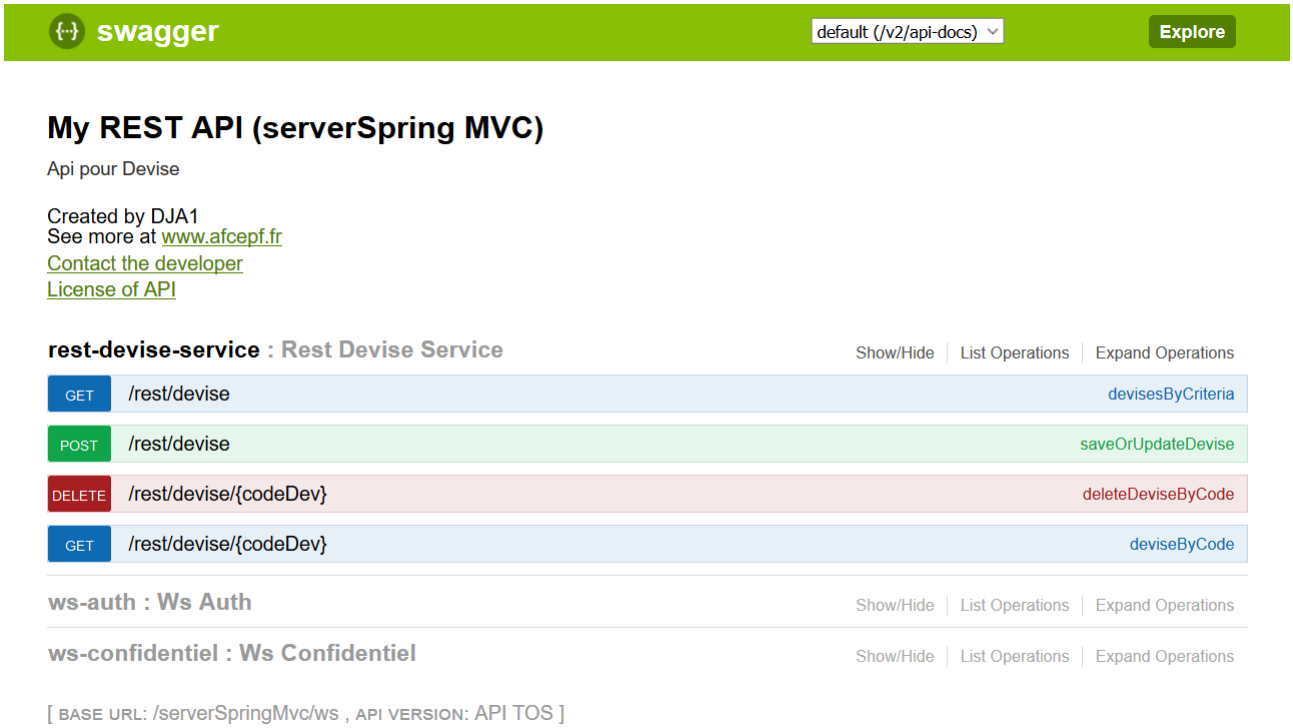
```
 <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
 </dependency>
```

Configuration explicite à idéalement placer dans application.properties :

```
springdoc.swagger-ui.path=/doc-swagger.html
```

dans **index.html** (ou ailleurs) :

```
<a href="./doc-swagger.html">documentation Api REST générée dynamiquement par swagger3/openapi</a>
```


swagger
default (/v2/api-docs)
Explore
My REST API (serverSpring MVC)
Api pour Devise
Created by DJA1
See more at www.afcepf.fr
Contact the developer
License of API
rest-devise-service : Rest Devise Service
Show/Hide
List Operations
Expand Operations
GET /rest/devise devisesByCriteria
POST /rest/devise saveOrUpdateDevise
DELETE /rest/devise/{codeDev} deleteDeviseByCode
GET /rest/devise/{codeDev} deviseByCode
ws-auth : Ws Auth
Show/Hide
List Operations
Expand Operations
ws-confidentiel : Ws Confidentiel
Show/Hide
List Operations
Expand Operations
[BASE URL: /serverSpringMvc/ws , API VERSION: API TOS]

NB : Selon le contexte applicatif , il faudra peut être paramétrer la sécurité de façon à pouvoir accéder à la documentation "swagger" générée :

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter
        //....
private static final String[] SWAGGER_AUTH_WHITELIST = {
        "/swagger-resources/**",  "/swagger-ui.html",   "/v2/api-docs",     "/webjars/**"
 };

protected void configure(HttpSecurity http) throws Exception {
        // configuration  partielle à compléter:
        http.authorizeRequests()
             .antMatchers("/rest/devise-api/public/**").permitAll()
             .antMatchers(SWAGGER_AUTH_WHITELIST).permitAll()
            .anyRequest().authenticated();
   }
}
```

Configuration de l'api via "annotations OpenApi/ swagger 3" :

Attention : les anciennes annotations de l'époque "swagger2" (~~@ApiModelProperty~~ , ~~@ApiOperation~~ , ~~@ApiParam~~) ont été changées lors de la standardisation swagger3/OpenApi (@Schema , @Operation, @Parameter , ...)

```
package org.mycontrib.backend.dto;
import io.swagger.v3.oas.annotations.media.Schema;
@Schema(description = "DTO (Result of conversion)")
public class ResConv {
        @Schema(description = "amount to convert", defaultValue = "100")
        private Double amount;

        @Schema( description = "source currency code", defaultValue = "EUR")
        private String source;
        ...
}
```

et dans une classe de @RestController :

```
...
import io.swagger.v3.oas.annotations.media.Operation;
import io.swagger.v3.oas.annotations.media.Parameter;
...
@RestController
@RequestMapping(value="/rest/devise-api/public" , headers="Accept=application/json")
public class PublicDeviseRestCtrl {
...
@RequestMapping(value="/convert" , method=RequestMethod.GET)
@Operation(summary= "convert amount from source to target currency",
    description = "exemple: convert?source=EUR&target=USD&amount=100")
        public ResConv convertir(
                    @RequestParam("amount")
                    @Parameter(description = "amount to convert", ... = "100")
                    Double montant,
                    @RequestParam("source")
                    @Parameter(description = "source currency code", … = "EUR")
                    String source,
                    @RequestParam("target")
                    @Parameter(description = "target currency code", ... = "USD")
                    String cible) {
                            Double res = convertisseur.convertir(montant, source, cible);
                            return new ResConv(montant, source, cible,res);
        }
….
```

Configuration (à peaufiner) pour intégrer Swagger 3 dans une application JEE (avec jax-rs) :

dans pom.xml

```
...
<properties>
      <swagger.version>2.0.10</swagger.version>
 </properties>
…
 <dependency>
         <groupId>io.swagger.core.v3</groupId>
         <artifactId>swagger-jaxrs2</artifactId>          <version>${swagger.version}</version>
      </dependency>

      <dependency>
         <groupId>io.swagger.core.v3</groupId>
         <artifactId>swagger-jaxrs2-servlet-initializer</artifactId>          <version>${swagger.version}</version>
      </dependency>

      <dependency>
         <groupId>io.swagger.core.v3</groupId>
         <artifactId>swagger-annotations</artifactId>          <version>${swagger.version}</version>
      </dependency>

      <!--
      <dependency>
                  <groupId>org.webjars</groupId>
                  <artifactId>swagger-ui</artifactId>
                  <version>3.20.8</version>
          </dependency>
          -->
          <!-- ou bien copy du contenu de  /dist de https://github.com/swagger-api/swagger-ui
            vers src/main/webapp/swagger-ui -->
…
 <build>
           <finalName>webapp_rest</finalName>
           <plugins>
                    <!-- plugin for api data generation -->
         <plugin>
           <groupId>io.swagger.core.v3</groupId>
           <artifactId>swagger-maven-plugin</artifactId>
           <version>${swagger.version}</version>
           <configuration>
              <outputFileName>my-api</outputFileName>
              <!-- <outputPath>${project.build.directory}/swagger-ui</outputPath> -->
              <outputPath>${project.basedir}/src/main/webapp</outputPath>
              <outputFormat>JSON</outputFormat>
              <configurationFilePath>${project.basedir}/src/main/resources/openapi.json
               </configurationFilePath>
           </configuration>
           <executions>
              <execution>
                 <phase>compile</phase>
                 <goals>
                    <goal>resolve</goal>
                 </goals>
              </execution>
           </executions>
         </plugin>
      </plugins>
 </build>
```

```
…
```

src/main/resources/openapi.json

```
{
 "resourcePackages": [
  "tp.web.rest"
 ],
 "ignoredRoutes": [],
 "prettyPrint": true,
 "cacheTTL": 0,
 "openAPI": {
        "servers":[
                { "url" : "http://localhost:8080/webapp_rest/rest",
                  "description" : "dev mode server"
                 }
        ],
  "info": {
   "version": 1.0,
   "title": "webapp_rest my-api",
   "description": "JAX-RS API docs",
   "license": {
    "name": "Apache 2.0",
    "url": "http://www.apache.org/licenses/LICENSE-2.0.html"
   }
  }
 }
}
```

via un **mvn package** , ça construit automatiquement

le fichier src/main/webapp/**my-api.json** décrivant la structure des web-services trouvés dans le package tp.web.rest .

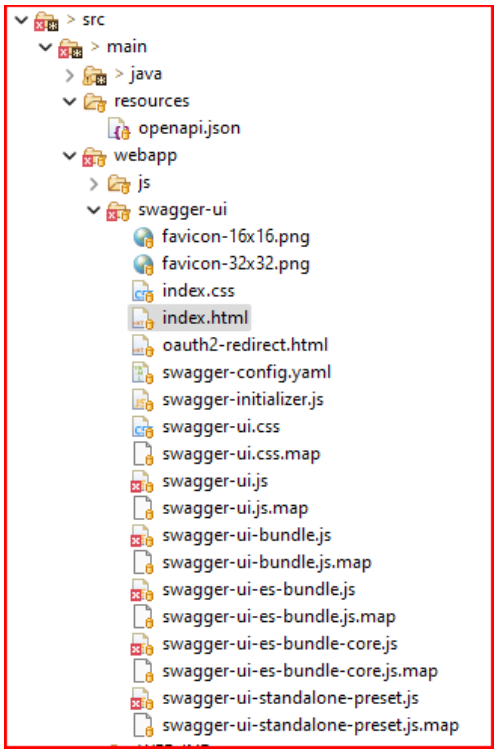
Dans **index.html**

```
  <a href="./my-api.json" target="_blank">my-api.json (swagger/openapidoc)</a><br/>
  <a href="./swagger-ui/index.html" target="_blank">swagger-ui index</a><br/>
```

Dans **swagger-ui/index.html**

```
<!-- HTML for static distribution bundle build -->
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="UTF-8">
  <title>Swagger UI</title>
  <link rel="stylesheet" type="text/css" href="./swagger-ui.css" />
  <link rel="stylesheet" type="text/css" href="index.css" />
  <link rel="icon" type="image/png" href="./favicon-32x32.png" sizes="32x32" />
  <link rel="icon" type="image/png" href="./favicon-16x16.png" sizes="16x16" />
 </head>

 <body>
```

```
  <div id="swagger-ui"></div>
  <script src="./swagger-ui-bundle.js" charset="UTF-8"> </script>
  <script src="./swagger-ui-standalone-preset.js" charset="UTF-8"> </script>
  <script src="./swagger-initializer.js" charset="UTF-8"> </script>

  <script>
  window.onload = function() {
    // Begin Swagger UI call region
    const ui = SwaggerUIBundle({
      url: "http://localhost:8080/webapp_rest/my-api.json",
      dom_id: '#swagger-ui',
      deepLinking: true,
      presets: [
        SwaggerUIBundle.presets.apis,
        SwaggerUIStandalonePreset
      ],
      plugins: [
        SwaggerUIBundle.plugins.DownloadUrl
      ],
      layout: "StandaloneLayout"
    });
    // End Swagger UI call region

    window.ui = ui;
  };
 </script>
 </body>
</html>
```



IV - Spring Security (l'essentiel)

1. Extension Spring-security (généralités)

L'extension **Spring-security** permet de simplifier le paramétrage de la **sécurité JEE** dans le cadre d'une application JEE/Web basée sur Spring.

1.1. Principales fonctionnalités de spring-security

Spring-security (fonctionnalités)

- **Configurer les zones web protégées**
  (URL publiques et URL nécessitants authentification)
- **Configurer le mode d'authentification** (HttpBasic ou BearerToken , formulaire de login, …)
- **Configurer** un accès à un "**realm**" (***liste d'utilisateurs*** pouvant s'authentifier *(via username/password)* et ayant des *rôles* et/ou des *permissions/privilèges*) (variantes : InMemory, JDBC, OAuth2/OIDC , UserDetailsService spécifique)
- ***Intégration Spring*** et ***compatibilité JavaWeb (WebFilter,...)***

Autres caractéristiques de spring-security :

- syntaxe xml ou java simplifiée (plus compacte et plus lisible que le standard "web.xml")
- possibilité de configurer via l'annotation *@PreAuthorize("hasRole('role1'))*les méthodes des composants "spring" qui seront ou pas accessibles selon le rôle de l'utilisateur authentifié.
- cryptage des mots de passe via bcrypt, ...

1.2. Principaux besoins types (spring-security)

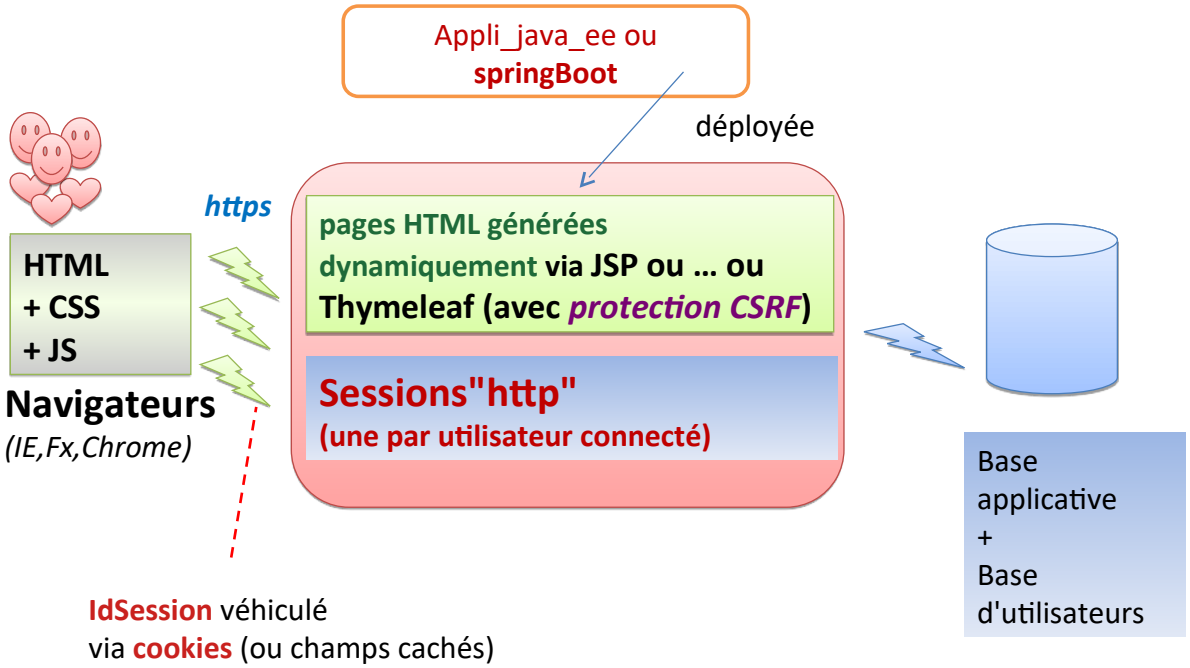
Spring-security (besoins classiques)

- ***Partie d'appli Web*** (avec ***@Controller*** et ***pages JSP*** ou bien **Thymeleaf**) avec sécurité JavaEE classique (*id de HttpSession* véhiculé par *cookie* , authentification Basic Http et formulaire de login)

- ***Partie Api REST*** (avec ***@RestController***) et avec ***BearerToken*** (ex : ***JWT***) gérée par le backend springBoot en mode **standalone**

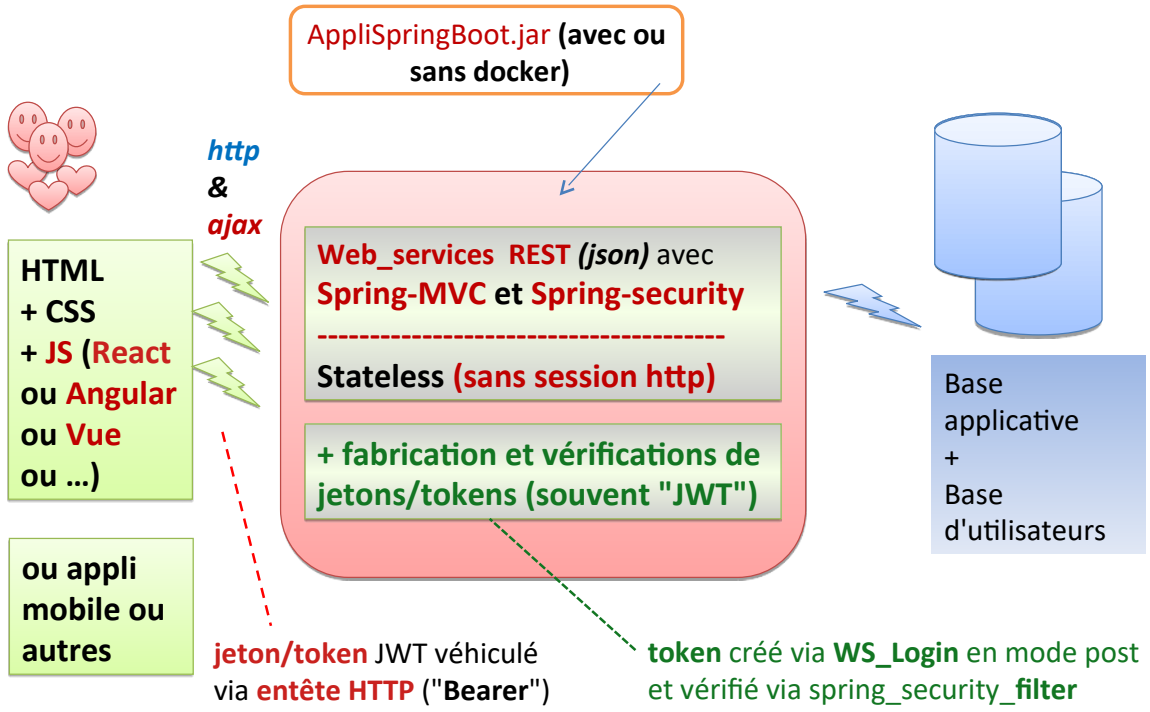
- ***Api REST en mode "ResourceServer"*** où ***l'authentification est déléguée*** via *OAuth2/OIDC* à un "***AuthorizationServer***" (ex : KeyCloak , Cognito , Azure-Directory, Okta , ...)

Sécurité en mode "jee/dhtml"

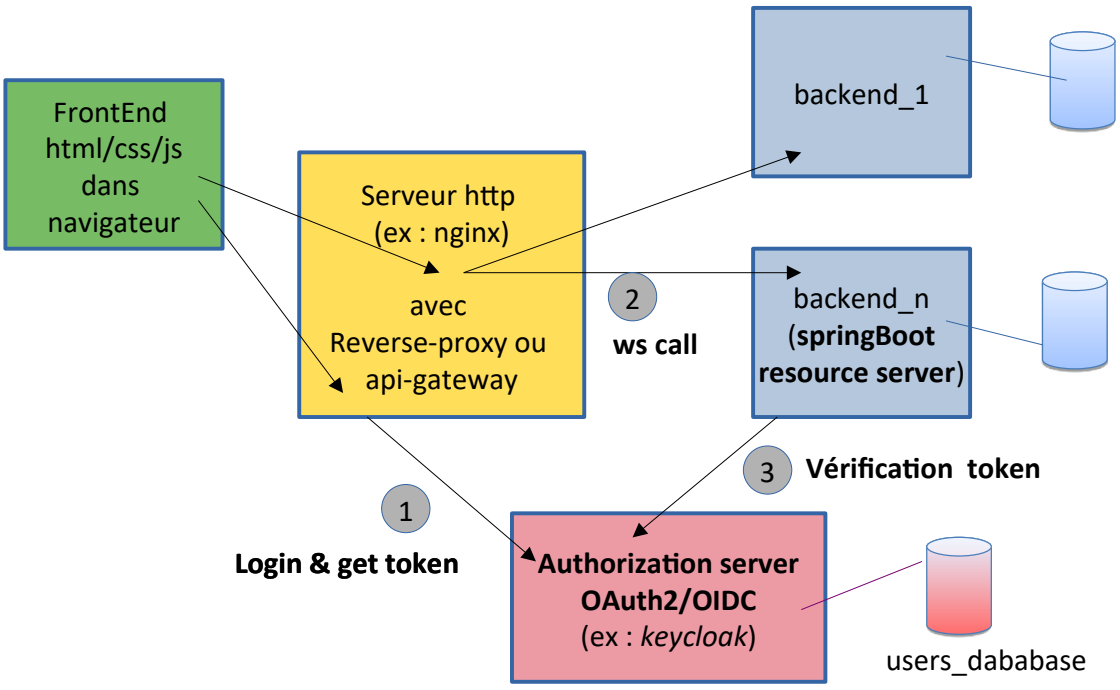


....

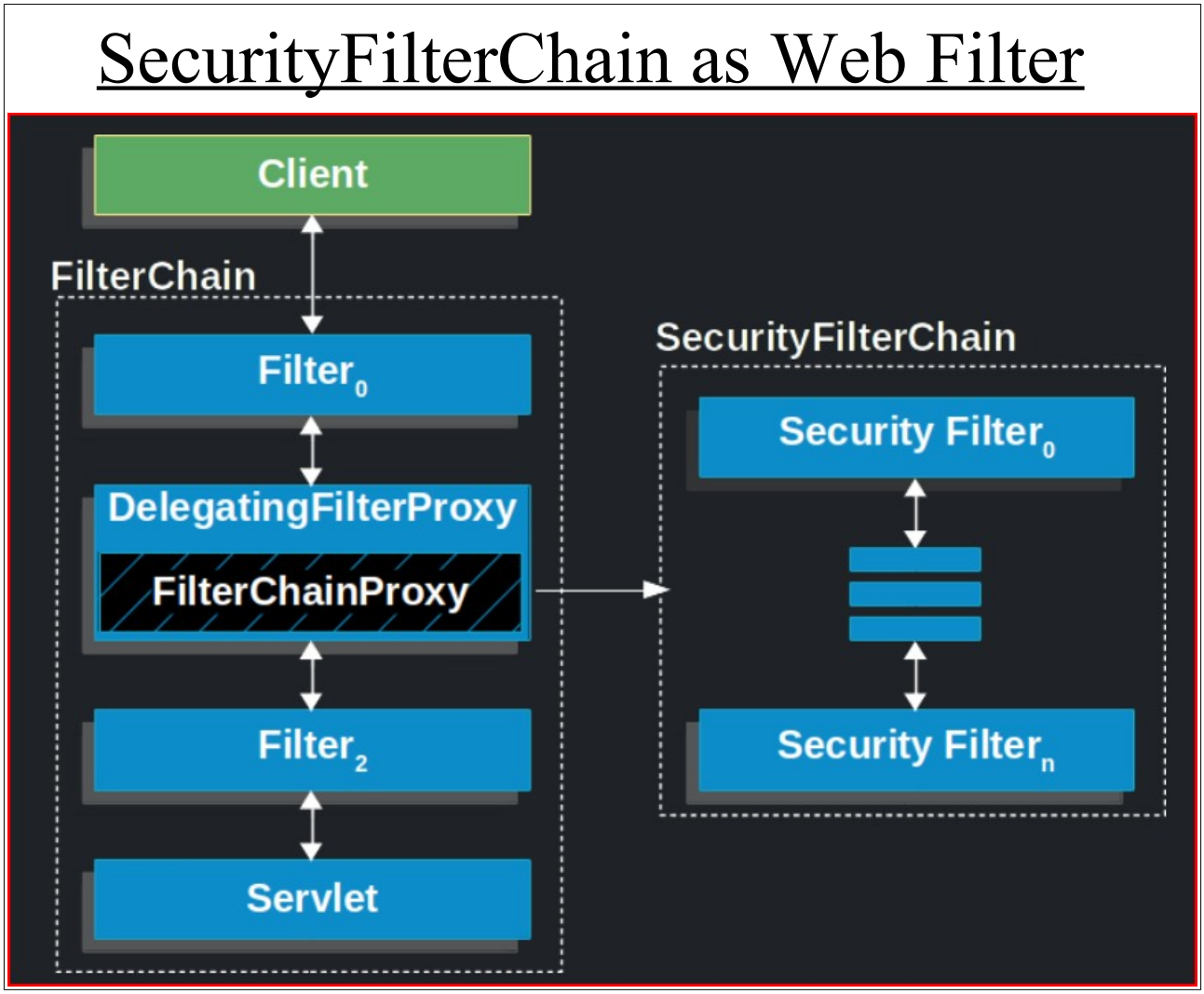
Sécurité en mode "standalone_jwt"



Sécurité en mode "oauth2/oidc"



1.3. Filtre web et SecurityFilterChain



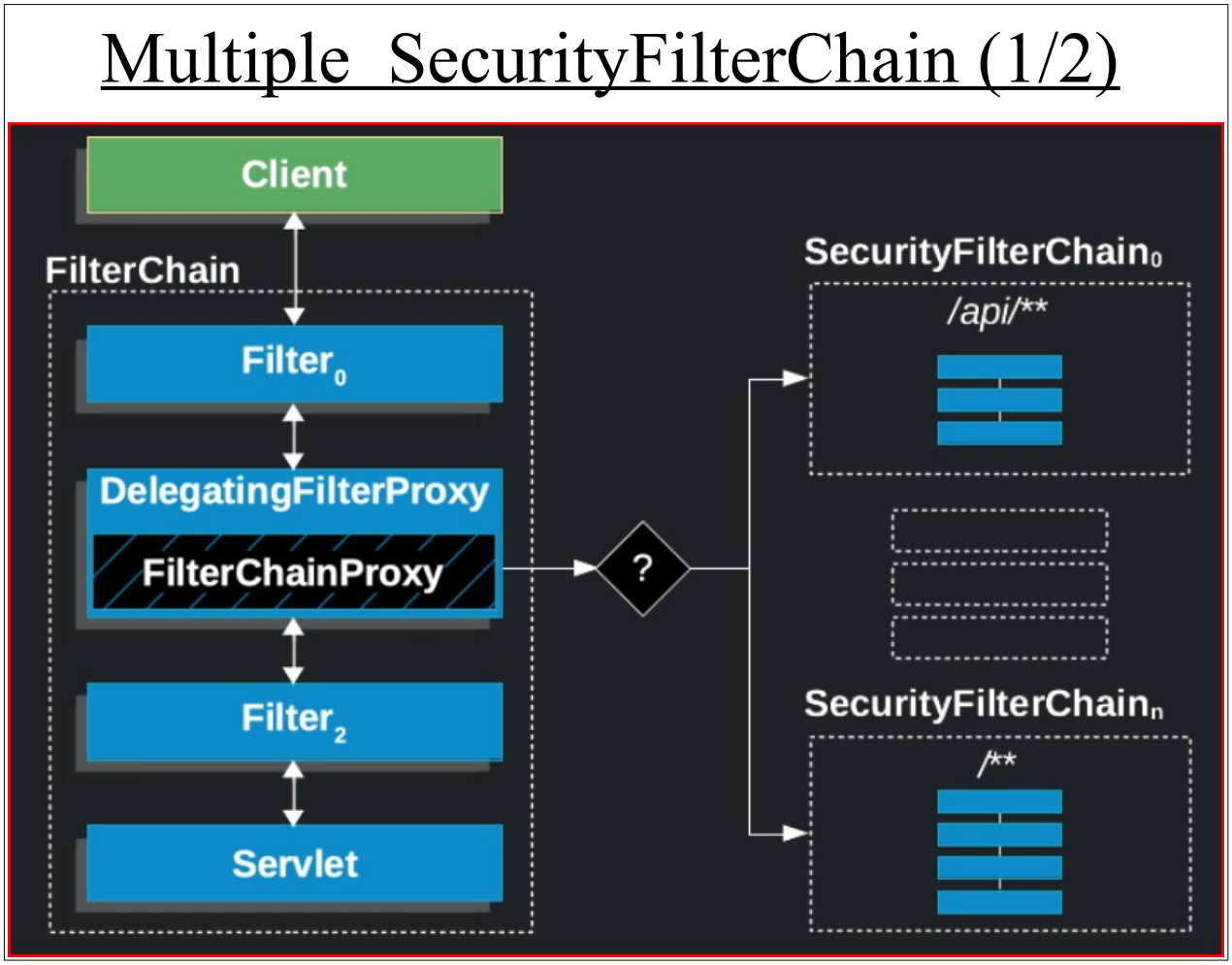
En interne les principales technologies "web" de spring sont basées sur des Servlets (Http).
Exemple : SpringMvc avec DispatcherServlet .
Avant qu'une requête Http soit traitée par Spring-Mvc et le DispatcherServlet , on peut configurer des filtres web (respectant l'interface normalisée javax.servlet.Filter) qui vont intercepter cette requête de manière à effectuer des pré-traitements (et d'éventuels post-traitements).

Au sein d'une application Spring ou SpringBoot, le composant prédéfini "DelegatingFilterProxy / FilterChainProxy" va automatiquement intercepter une requête Http et déclencher une chaîne ordonnée de "SecurityFilter" .

La "SecurityFilterChain" peut être unique dans le cas d'une application bien précise (ex1 : AppliWeb uniquement basée sur JSP/thymeleaf , ex2 : Api-rest en mode micro-service)

Une application Spring/SpringBoot peut cependant comporter plusieurs parties complémentaires et il est alors possible de configurer plusieurs "SecurityFilterChain" .

1.4. Multiple SecurityFilterChain



NB: quand une requête arrive, le FilterChainProxy de Spring-security va utiliser le premier SecurityFilterChain correpondant à l'url de la requête et va ignorer les autres
(point clef: la correspondance se fait via httpSecurity.antMatcher() sans s)

Il est donc important qu'une partie de l'URL (plutôt au début) puisse faire office d'aiguillage non ambigü vers une SecurityFilterChain ou une autre.

Exemple de convention d'URL :

/rest/api-xyz/...
ou
/site/...
ou
**

Multiple SecurityFilterChain (2/2)

```
@Configuration
public class MySecurityConfig {

    @Bean  @Order(1)
    protected SecurityFilterChain restApiFilterChain(
                    HttpSecurity http)   throws Exception {
     http.securityMatcher("/rest/**")
          .authorizeHttpRequests(...)...build();
    }

    @Bean  @Order(2)
    protected SecurityFilterChain siteFilterChain(
                    HttpSecurity http)   throws Exception {
     http.securityMatcher("/site/**")
          .authorizeHttpRequests(...)...build();
    }

    @Bean  @Order(3)
    protected SecurityFilterChain othersFilterChain(
                    HttpSecurity http)   throws Exception {
     http.securityMatcher("/**") // "/**" in last order !!!
          .authorizeHttpRequests(...)...build();
    }
}
```

NB: 3 securityChain avec ordre important à respecter
- @Order(1) pour les URL commencant par /rest (ex: /rest/api-xxx , /rest/api-yyy)
- @Order(2) pour une éventuelle partie /site/ basée sur @Controller + JSP ou Thymeleaf
- @Order(3) pour le reste (autres URLs , pages static ou pas "spring")

NB : une instance de SecurityFilterChain peut éventuellement être associée à un "AuthenticationManager" spécifique ou bien ne pas l'être et dans le cas un AuthenticationManager global/principal sera utilisé par défaut .

1.5. Vue d'ensemble sur les phases de Spring-security

1. Une des premières phases exécutées par un filtre de sécurité consiste à **extraire certaines informations d'authentification de la requête Http** (ex : **username/password** en mode "basic" ou bien **jeton** (jwt ou autre) en mode "bearer").

2. Une seconde phase consiste à déclencher **authManager.authenticate(authentication_to_check)** de manière à comparer les informations d'authentification à verifier avec une liste d'utilisateurs valide (à récupérer quelquepart : LDAP , JDBC , InMemory , OAuth2/OIDC, …)

3. Les informations sur l'authentification réussie sont stockée dans un point central ***SecurityContextHolder.getContext()*** au format **Authentication** (interface avec variantes)

4. Certaines configurations "xml" ou "java" ou "via annotations" précises permettront d'accepter ou refuser un traitement demandé en fonction des informations d'authentification réussies stockées préalablement dans le contexte de sécurité.
   (ex : `@PreAuthorize("hasRole('ADMIN')")`
   ou bien `@PreAuthorize("hasAuthority('SCOPE_resource.delete')")`)

1.6. Comportement de l'authentification (spring-security)

L'interface fondamentale "***AuthenticationManager***" comporte la méthode fondamentale ***authenticate***() dont le comportment est ci-après expliqué :

Authentication authenticate(Authentication authentication) throws AuthenticationException;

avant appel : authentication avec getPrincipal() retournant souvent username (String)
getCredential() retournant password à tester ou autre .

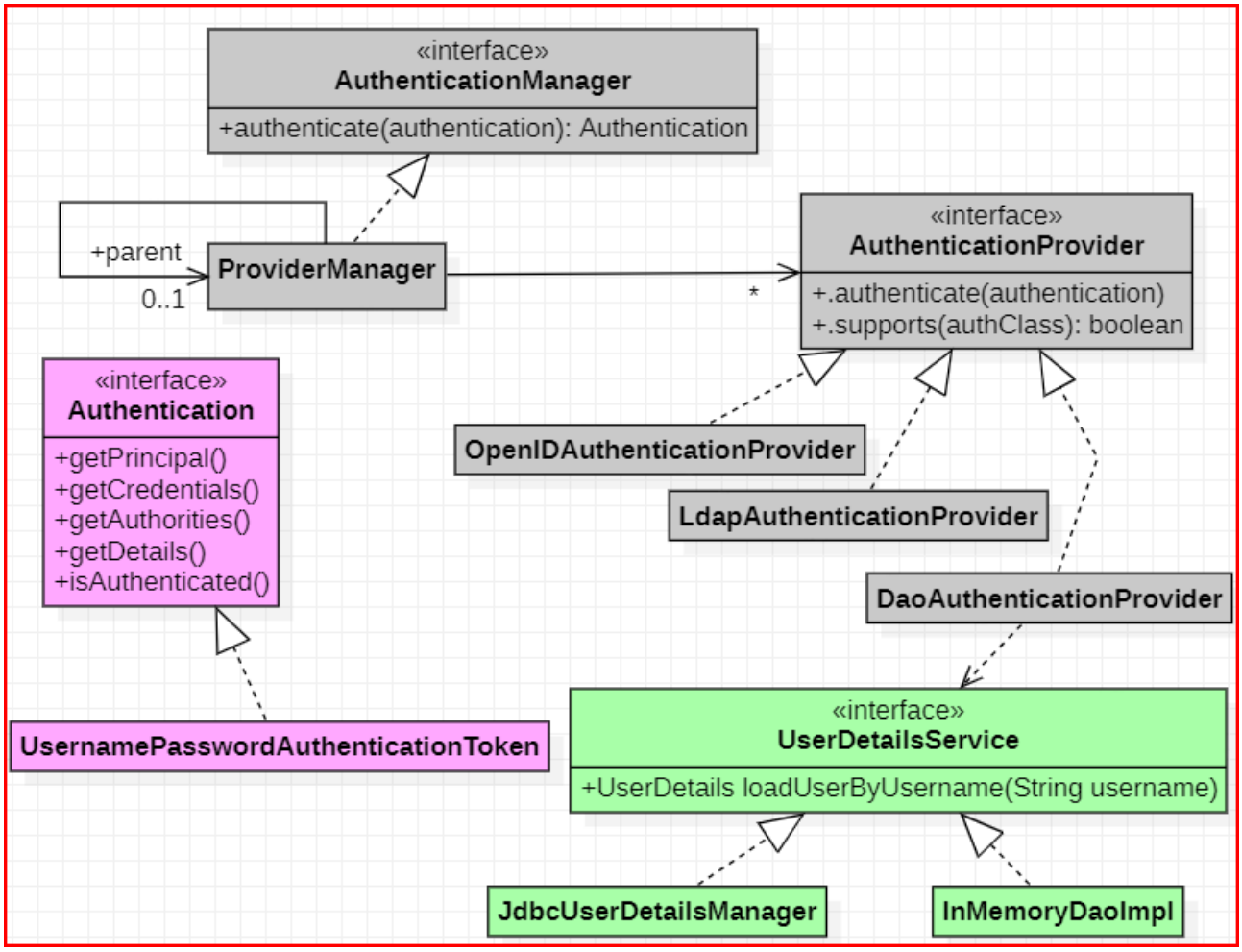
après appel : authentication avec getPrincipal() retournant UserDetails si ok
ou bien AuthenticationException sinon

Si l'authentification échoue --> AuthenticationException --> retour status HTTP 401 (Unauthorized) ou bien redirection vers formulaire de login (en fonction du contexte)

Si l'authentification est réussie -->

- la méthode authenticate() retourne un objet (implémentant l'interface "Authentication") bien complet (comportant "Roles utilisateurs" , ...) .
- L'objet "Authentication" est alors automatiquement stocké dans le "SecurityContextHolder / SecurityContext" (lié au Thread courant prenant en charge la requête Http) par spring-security .

1.7. Mécanismes d'authentification (spring-security)



NB1: La classe "ProviderManager" implémente l'interface AuthenticationManager
en itèrant sur une liste de AuthenticationProvider enregistrés de façon
à trouver le premier AuthenticationProvider capable de gérer l'authentification.

NB2 : A priori , Le "ProviderManager" principal (lié à l'implémentation de AuthenticationManager)
est potentiellement relié à un ProviderManager parent (qui n'est utilisé que si l'authentification
réalisée par le "AuthenticationManager/ ProviderManager" échoue).
Ce lien s'effectue via AuthenticationManagerBuilder.parentAuthenticationManager()

La classe DaoAuthenticationProvider correspond à une implémentation importante de
AuthenticationProvider qui s'appuie en interne sur UserDetailsService (Jdbc ou InMemory ou
spécifique)

1.8. Vue d'ensemble sur configuration concrète de la sécurité

Historique important :

- vers 2010 , configuration de spring-security au format xml (via un fichier spring-security.xml)
  et balises de types <security-http> , <security:intercept-url , permitAll , denyAll , …. /> , <security:authentication-manager> <security:authentication-provider> <security:user-service> <security:user name="user1" password="pwd1" authorities="ROLE_USER" /> .

- Vers 2015-2020 , configuration souvent "java" de la sécurité via

```
@Configuration
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    …. protected void configure(final HttpSecurity http) throws Exception {…}
}
```

- **Depuis 2022 et Spring 5.7** WebSecurityConfigurerAdapter est devenu "deprecated/obsolete" et il est conseillé d'utiliser

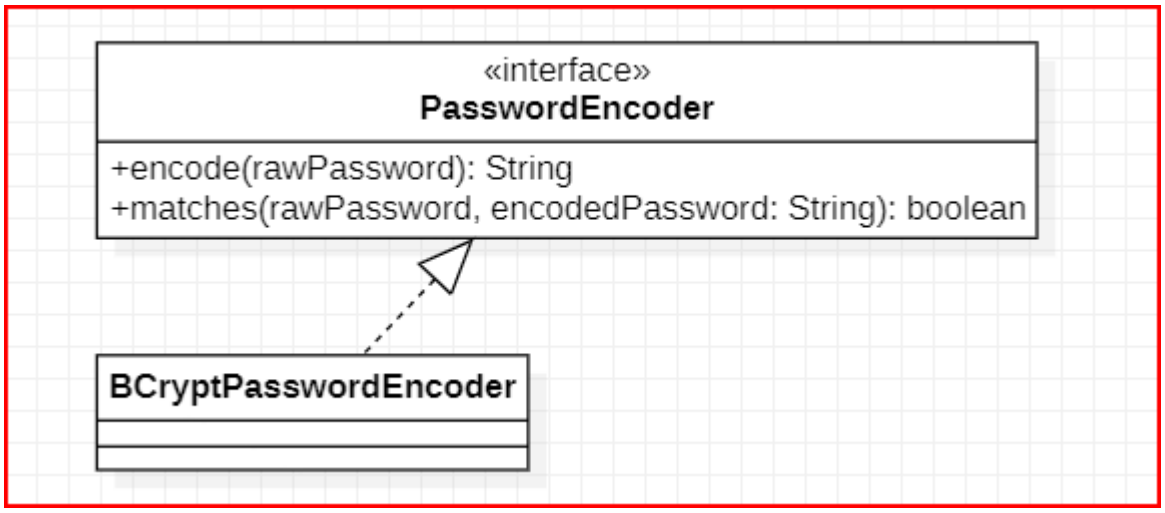
```
@Configuration
public class MySecurityConfig /* without inheritance */ {
 … protected void SecurityFilterChain
 myFilterChain(HttpSecurity http) throws Exception {…}
}
```

- **Depuis 2023 et Spring 6 , SpringSecurity a encore évolué :**
  Plus de ~~.and().~~ mais que des lambda-expressions imbriquées plus claires (mieux délimitées) **.**
  Quelques autres changement:
  ~~antMatchers()~~ remplacé par **requestMatchers(...)**
  ~~@EnableGlobalMethodSecurity(prePostEnabled = true)~~ remplacé apr
  @EnableMethodSecurity() avec prePostEnabled = true par défaut
  ...

Dans tous les cas , **HttpSecurity http** , correspond à un point centralisé de la configuration de spring-security qu'il faut :

- soit analyser (si d'origine XML)
- soit définir et construire dans le cas d'une configuration "java" (@Configuration)

1.9. Encodage classique des mots de passe via BCrypt



NB : L'algorithme de cryptage "BCrypt" a été spécialement mis au moins pour encoder des mots de passes avant de les stocker en base. A partir d'un encodage "bcrypt" il est quasi impossible de déterminer le mot de passe d'origine qui a été crypté .

NB : Via BCrypt , si on encodage plusieurs fois "**pwd1**" , ça va donner des encodages différents

(ex : "**$2a$10$wdysBwvK8l5t5zJsuKdcu.wMJJum8f3BA5/X6muaNpVoLx4rj1tKm**"
ou "**$2a$10$OBiZKdISPSio6LI7Mh9eRubBVIQ8q0NzCoSIcDIm9L4MvBzwmbfmq**")

Ceci dit via la méthode **.matches()** les 2 encodages seront tous les 2 considérés comme corrects pour tester le mot de passe "pwd1" .

```
....
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
@Configuration
public class MySecurity {
   @Bean
   public PasswordEncoder passwordEncoder() {
      return new BCryptPasswordEncoder();
     //or new BCryptPasswordEncoder(int strength) with strength between 4 and 31
   }
}
```

1.10. Prise en compte d'une authentification vérifiée

Une fois l'authentification effectuée et stockée dans le contexte "SecurityContextHolder", on peut alors très facilement accéder aux infos "utilisateur" vérifiées via des instructions de ce type :

```
Object principal = SecurityContextHolder.getContext().getAuthentication().getPrincipal();
if (principal instanceof UserDetails) {
  String username = ((UserDetails)principal).getUsername();
}
```

L'objet "Authentication" comporte une méthodes **getAuthorities()** retournant un paquet d'éléments de type "GrantedAuthority" dont "SimpleGrantedAuthority" est l'implémentation la plus classique.

"SimpleGrantedAuthority" comporte un nom de rôle (ex "ROLE_ADMIN" ou "ROLE_USER" , ...)

Lorsqu'un peu plus tard , un accès à une partie de l'application sera tenté (page jsp , méthode appelée sur un contrôleur , ...) les mécanismes de la partie "contrôle d'accès" de spring-security pour alors assez facilement autoriser ou refuser les actions en comparant les rôles mémorisés dans l'objet "Authentication" du contexte avec certaines configurations du genre :

```
@PreAuthorize("hasRole('ADMIN')")
```

2. Configuration des "Realms" (spring-security)

2.1. AuthenticationManagerBuilder

L'objet technique ***AuthenticationManagerBuilder*** sert à construire un objet implémentant l'interface ***AuthenticationManager*** qui servira lui même à authentifier l'utilisateur .

Selon le contexte de l'application, cet objet fondamental peut être récupéré de l'une des façons suivantes :

- par injection de dépendances (si déjà préparé/défini ailleurs)
- **par instanciation directe**
- par récupération dans la partie "***sharedObject***" de ***HttpSecurity***

Exemples (à adapter au contexte) :

```
public static AuthenticationManagerBuilder newAuthenticationManagerBuilder() {
        final ObjectPostProcessor<Object> objectPostProcessor =
                new ObjectPostProcessor<Object>() {
                        @Override
                        public <O extends Object> O postProcess(final O object) {
                         return object;
                        }
                };
        AuthenticationManagerBuilder authMgrBuilder  =
                        new AuthenticationManagerBuilder(objectPostProcessor);
        return authMgrBuilder;
        }
```

```
public static AuthenticationManagerBuilder
            authenticationManagerBuilderFromHttpSecurity(HttpSecurity httpSecurity) {
        AuthenticationManagerBuilder authenticationManagerBuilder = httpSecurity
                                .getSharedObject(AuthenticationManagerBuilder.class);
        authenticationManagerBuilder.parentAuthenticationManager(null) ;
         return authenticationManagerBuilder;
        }
```

NB : Une fois créé ou récupéré , cet objet "AuthenticationManagerBuilder" sera la base souvent indispensable du paramétrage d'un "realm" (liste d'utilisateurs autorisés à utiliser l'application).

2.2. Délégation d'authentification (OAuth2/Oidc)

```
import org.springframework.security.config.Customizer;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
...
public HttpSecurity configureEndOfSecurityChain(HttpSecurity http)throws Exception{
        return http.
                sessionManagement(sM →
                        sM.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .oauth2ResourceServer((oauth2) -> oauth2.jwt(Customizer.withDefaults()));
        }
```

et

dans application.properties

```
spring.security.oauth2.resourceserver.jwt.issuer-uri=https://www.d-defrance.fr/keycloak/realms/sandboxrealm
```

et

```
@PreAuthorize("hasAuthority('SCOPE_resource.write')") ou autre
```

avec dans *pom.xml*

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
</dependency>
```

2.3. Realm temporaire "InMemory"

```
authenticationManagerBuilder.inMemoryAuthentication()
    .withUser("user1").password(passwordEncoder.encode("pwd1")).roles("USER").and()
    .withUser("admin1").password(passwordEncoder.encode("pwd1")).roles("ADMIN").and()
    .withUser("user2").password(passwordEncoder.encode("pwd2")).roles("USER").and()
    .withUser("admin2").password(passwordEncoder.encode("pwd2")).roles("ADMIN");
```

2.4. Authentification jdbc ("realm" en base de données)

La configuration ci-après permet de configurer **spring-security** pour qu'il accède à une **liste de comptes "utilisateurs" dans une base de données relationnelle** (ex : H2 ou Mysql ou ...) .

Cette base de données sera éventuellement différente de celle utilisée par l'aspect fonctionnel de l'application .

Au sein de l'exemple suivant , la méthode ***initRealmDataSource()*** paramètre un objet DataSource vers une base h2 spécifique à l'authentification ***(jdbc:h2:~/realmdb).***

L'instruction

```
JdbcUserDetailsManagerConfigurer jdbcUserDetailsManagerConfigurer =
                                        auth.jdbcAuthentication().dataSource(realmDataSource);
```

permet d'initialiser AuthenticationManagerBuilder en mode jdbc en précisant le DataSource et donc la base de données à utiliser .

L'instruction jdbcUserDetailsManagerConfigurer.**withDefaultSchema**(); *(à ne lancer que si les tables "**users**" et "**authorities**" n'existent pas encore dans la base de données)* permet de créer les tables nécessaires *(avec noms et structures par défaut)* dans la base de données.

Par défaut , la table **users(username, password)** comporte les mots de passe (souvent cryptés)

et la table **authorities(username, authority)** comporte la liste des rôles de chaque utilisateur

JdbcAppDbGlobalUserDetailsConfig.java *à adapter au contexte*

```
package org.mycontrib.generic.security.config;
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.ResultSet;
import javax.sql.DataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Profile;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.authentication.configurers.provisioning.JdbcUserDetailsManagerConfigurer;
import org.springframework.security.config.annotation.authentication.configurers.provisioning.UserDetailsManagerConfigurer;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Configuration
//@Profile("appDbSecurity") //with jdbc
public class JdbcAppDbGlobalUserDetailsConfig  {
   @Autowired
   private BCryptPasswordEncoder passwordEncoder;

   private static DataSource realmDataSource;

   private static void initRealmDataSource() {
         DriverManagerDataSource driverManagerDataSource = new DriverManagerDataSource();
         driverManagerDataSource.setDriverClassName("org.h2.Driver");
         driverManagerDataSource.setUrl("jdbc:h2:~/realmdb");
         driverManagerDataSource.setUsername("sa");
         driverManagerDataSource.setPassword("");
         realmDataSource = driverManagerDataSource;
   }

   private boolean isRealmSchemaInitialized() {
         int nbExistingTablesOfRealmSchema = 0;
         try {
                        Connection cn = realmDataSource.getConnection();
                        DatabaseMetaData meta = cn.getMetaData();
                        String tabOfTableType[] = {"TABLE"};
                        ResultSet rs = meta.getTables(null,null,"%",tabOfTableType);
                        while(rs.next()){
```

```
                                        String existingTableName = rs.getString(3);
                                        if(existingTableName.equalsIgnoreCase("users")
                                           || existingTableName.equalsIgnoreCase("authorities")) {
                                                nbExistingTablesOfRealmSchema++;
                                        }
                                }
                                rs.close();
                                cn.close();
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
        return (nbExistingTablesOfRealmSchema>=2);
 }

 @Autowired
 public void globalUserDetails(final AuthenticationManagerBuilder auth) throws Exception {
        initRealmDataSource();
        JdbcUserDetailsManagerConfigurer jdbcUserDetailsManagerConfigurer =
                                        auth.jdbcAuthentication().dataSource(realmDataSource);
        if(isRealmSchemaInitialized()) {
                /*
                 jdbcUserDetailsManagerConfigurer
                .usersByUsernameQuery("select username,password, enabled from users where username=?")
                .authoritiesByUsernameQuery("select username, authority from authorities where username=?");
                //by default
                */
                // or .authoritiesByUsernameQuery("select username, role from user_roles where username=?")
                //if custom schema
        }else {
                //creating default schema and default tables "users" , "authorities"
                jdbcUserDetailsManagerConfigurer.withDefaultSchema();
                //insert default users:
                configureDefaultUsers(jdbcUserDetailsManagerConfigurer);
        }
 }

void configureDefaultUsers(UserDetailsManagerConfigurer udmc){
        udmc
          .withUser("user1").password(passwordEncoder.encode("pwduser1")).roles("USER").and()
           .withUser("admin1").password(passwordEncoder.encode("pwdadmin1")).roles("ADMIN","USER").and()
           .withUser("publisher1").password(passwordEncoder.encode("pwdpublisher1")).roles("PUBLISHER","USER").and()
           .withUser("user2").password(passwordEncoder.encode("pwduser2")).roles("USER").and()
           .withUser("admin2").password(passwordEncoder.encode("pwdadmin2")).roles("ADMIN").and()
           .withUser("publisher2").password(passwordEncoder.encode("pwdpublisher2")).roles("PUBLISHER");
}
}
```

2.5. Authentification "personnalisée" en implémentant l'interface UserDetailsService

Si l'on souhaite coder un accès spécifique à la liste des comptes utilisateurs (ex : via JPA ou autres), on peut implémenter l'interface **UserDetailsService** .

L'interface ***UserDetailsService*** comporte cette unique méthode :

```
UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;
```

Cette méthode est censée remonter les données d'un compte utilisateur depuis un certain endroit (base de données , mongoDB , ...) .

Ces infos "utilisateur" doivent être une implémentation de l'**interface "UserDetails**" (classe "User" par exemple). L'objet "User" (ou un équivalent implémentant "UserDetails") est censée comporter le bon mot de passe.

Les mécanismes internes de Spring-security ("AuthenticationProvider" , ...) vont alors pouvoir comparer le bon mot de passe avec celui renseigné par l'utilisateur qui souhaite s'authentifier.

Dans certains cas la comparaison passe par une implémentation de "PasswordEncoder" (ex : "BCryptPasswordEncoder") lorsque les mots de passe sont cryptés dans la base de données.

Exemple :

```
package ….;

import ...;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.password.PasswordEncoder;

@Profile("withSecurity")
@Service
public class MyUserDetailsService implements UserDetailsService {
  Logger logger = LoggerFactory.getLogger(MyUserDetailsService.class);
  @Autowired   private PasswordEncoder passwordEncoder;
  @Autowired  private ServiceCustomer serviceCustomer;

  @Override
  public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        UserDetails userDetails=null;
        logger.debug("MyUserDetailsService.loadUserByUsername() called with username="+username);

        List<GrantedAuthority> authorities = new ArrayList<GrantedAuthority>();
        String password=null;
        if(username.equals("james_Bond")) {
              password=passwordEncoder.encode("007");//simulation password ici
              authorities.add(new SimpleGrantedAuthority("ROLE_AGENTSECRET"));
```

```
                userDetails = new User(username, password, authorities);
        }
        else {
        //NB le username considéré comme potentiellement
        //égal à firstname_lastname
            try {
                String firstname = username.split("_")[0];
                String lastname = username.split("_")[1];

                List<Customer> customers =
                    serviceCustomer.rechercherCustomerSelonPrenomEtNom(firstname,lastname);
                if(!customers.isEmpty()) {
                        Customer firstCustomer = customers.get(0);
                        authorities.add(new SimpleGrantedAuthority("ROLE_CUSTOMER"));
                                                //ou "ROLE_USER" ou "ROLE_ADMIN"
                        password=firstCustomer.getPassword();// déjà stocké en base en mode crypté
                        //password=passwordEncoder.encode(firstCustomer.getPassword());
                        //si pas stocké en base en mode crypté (PAS BIEN !!!)
                        userDetails = new User(username, password, authorities);
                }
                } catch (Exception e) {
                        //e.printStackTrace();
                }
        }
     if(userDetails==null) {
        //NB: il est important de remonter UsernameNotFoundException (mais pas null , ni une autre exception)
        //si l'on souhaite qu'en cas d'échec avec cet AuthenticationManager
        //un éventuel AuthenticationManager parent soit utilisé en plan B
        throw new UsernameNotFoundException(username + " not found");
        }

    return userDetails;
    //NB: en retournant userDetails = new User(username, password, authorities);
    //on retourne comme information une association entre usernameRecherché et
    //(bonMotDePasseCrypté + liste des rôles)
    //Le bonMotDePasseCrypté servira simplement à effectuer une comparaison avec le mot
    //de passe qui sera saisi ultérieurement par l'utilisateur
     //(via l'aide de passwordEncoder.matches())
   }

}
```

3. Configuration des zones(url) à protéger

3.1. Généralités sur la configuration de HttpSecurity

Ancienne façon de faire (devenue obsolète/deprecated depuis la version 5.7 de spring-security) :

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfig extends ~~WebSecurityConfigurerAdapter~~ {
  ...
   @Override
   protected void ~~configure~~(HttpSecurity http) throws Exception {
            http.authorizeRequests()
                     .antMatchers("/", "/favicon.ico", "/**/*.png","/**/*.gif", "/**/*.svg",
                     "/**/*.jpg", "/**/*.css","/**/*.map","/**/*.js").permitAll()
                     .antMatchers("/to-welcome").permitAll()
                     .antMatchers("/session-end").permitAll()
                    .antMatchers("/xyz").permitAll()
                     .anyRequest().authenticated()
                   .and().formLogin().permitAll()
                  .and().csrf();
  }
...
}
```

Nouvelle façon conseillée depuis la version 5.7 de spring-security *(ici en syntaxe v6)*:

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WithSecurityMainFilterChainConfig {
…
        @Bean
        @Order(99)
        protected SecurityFilterChain myFilterChain(HttpSecurity http)
                        throws Exception {
                http.authorizeHttpRequests(
                    //exemple très permissif ici à grandement adapter !!!!
                     auth -> auth.requestMatchers("/**/*.*").permitAll() )
                .cors( Customizer.withDefaults() )
                .headers(headers ->
                        headers.frameOptions( frameOptions-> frameOptions.sameOrigin()) )
                .csrf(csrf->csrf().disable());
                return http.build();
              }
}
```

NB : la méthode *myFilterChain()* pourra éventuellement appeler des sous fonctions pour paramétrer http (de type HttpSecurity) de façon flexible et modulaire avant de déclencher http.build() .

3.2. Configuration type pour un projet de type Thymeleaf ou JSP

```
package .....;
....
{
   public HttpSecurity configureHttpSecurityV1(HttpSecurity http) throws Exception {

        return   http.authorizeHttpRequests( auth ->
                        auth.requestMatchers("/",
                         "/favicon.ico",
                        "/**/*.png",
                        "/**/*.gif",
                        "/**/*.svg",
                        "/**/*.jpg",
                        "/**/*.css",
                         "/**/*.map",
                         "/**/*.js").permitAll()
                        .requestMatchers("/to-welcome").permitAll()
                        .requestMatchers("/session-end").permitAll()
                        .requestMatchers("/xyz").permitAll()
                         .anyRequest().authenticated() )
                    .formLogin( formLogin -> formLogin.permitAll() )
                   /*.formLogin(  formLogin -> formLogin.loginPage("/login")
                                                        .failureUrl("/login-error")
                                                         .permitAll()*/
                    .csrf(Customizer.withDefaults());
   }
}
```

3.3. Champ caché " _csrf " de spring-mvc utile pour pages/vues "java/jsp" mais inutile pour Api-REST avec tokens .

NB: Ce champ caché correspond au "*Synchronizer Token Pattern*" (que l'on retrouve dans les frameworks web concurrents "Stuts" ou "JSF") : le coté serveur compare la valeur d'un jeton aléatoire stockée en session http avec celle stockée dans un champ caché et refuse de gérer la requête "re-postée" si la comparaison n'est pas réussie.

D'autre part , le terme ***CSRF*** (signifiant "*Cross Site Request Forgery*" correspond à un éventuel problème de sécurité : un site "malveillant" (utilisé en parallèle au sein d'un navigateur) déclenche automatiquement (via javascript ou autre) des requêtes non voulues (ex : virement monétaire) en utilisant le contexte d'un site à priori de confiance (mais pas assez protégé) .

Avec <form> (au lieu de <form:form> de SpringMvc / jsp) , il faut insérer nous même le champ suivant au sein du formulaire d'une page ".jsp" :

<input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>

<form:form …> de SpringMvc / jsp ou bien l'équivalent thymeleaf gère (génère) automatiquement le champ caché **_csrf** attendu par **spring-security** . *Exemple* : <input type="hidden" name="_csrf" value="8df91b84-74c1-4013-bd44-ede7b00779a2" />) .

3.4. Configuration type pour un projet de type "Api REST"

```
package .....;
....
{
    public HttpSecurity configureHttpSecurityV2(HttpSecurity http) throws Exception {
          return http.authorizeHttpRequests( auth ->
                auth.requestMatchers("/", "/favicon.ico", "/**/*.png", "/**/*.gif", "/**/*.svg",
                   "/**/*.jpg", "/**/*.html", "/**/*.css", "/**/*.js").permitAll()
                    .requestMatchers(HttpMethod.POST,"/auth/**").permitAll()
                   .requestMatchers("/xyz-api/public/**").permitAll()
                   .requestMatchers("/xyz-api/private/**").authenticated() )
                .cors( Customizer.withDefaults())
                 //enable CORS (avec @CrossOrigin sur class @RestController)
                .csrf( csrf -> csrf.disable() )
                // If the user is not authenticated, returns 401
                .exceptionHandling(eh ->
                            eh.authenticationEntryPoint(getRestAuthenticationEntryPoint() )
                // This is a stateless application, disable sessions
                .sessionManagement(sM ->
                          sM.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                // Custom filter for authenticating users using tokens
                .addFilterBefore(jwtAuthenticationFilter,
                                  UsernamePasswordAuthenticationFilter.class);
   }

  private AuthenticationEntryPoint getRestAuthenticationEntryPoint() {
       return new HttpStatusEntryPoint(HttpStatus.UNAUTHORIZED);
    }
}
```

...

ANNEXES

V - Annexe – Spring-MVC (JSP et Thymeleaf)

1. Spring-MVC avec pages JSP

1.1. Dépendances maven (SpringMvc + JSP)

Dépendances maven nécessaires (en intégration moderne "spring-boot"):

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

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

et (si vues de type "**.jsp**")

```
      <dependency>
            <groupId>org.apache.tomcat.embed</groupId>
            <artifactId>tomcat-embed-jasper</artifactId>
            <scope>provided</scope>
      </dependency>
      <dependency>
            <groupId>jakarta.servlet.jsp.jstl</groupId>
            <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
      </dependency>
      <dependency>
            <groupId>org.glassfish.web</groupId>
            <artifactId>jakarta.servlet.jsp.jstl</artifactId>
      </dependency>
      <!-- ou ancien équivalent spring5/springBoot2/jee -->
      <!--  <dependency>
                  <groupId>javax.servlet</groupId>
                  <artifactId>jstl</artifactId>
      </dependency> -->
```

1.2. Configuration en version ".jsp":

src/main/resources/**application.properties**

```
server.servlet.context-path=/myMvcSpringBootApp

server.port=8080

#spring.mvc.view.prefix=/WEB-INF/view/

spring.mvc.view.prefix=/jsp/

spring.mvc.view.suffix=.jsp
```

Avec cette configuration , un **return "xy"** d'un contrôleur déclenchera l'affichage de la page **/jsp/xy.jsp** et selon la structure du projet , le répertoire /***jsp*** sera placé dans **src/main/resources/META-INF/resources** ou **ailleurs** .

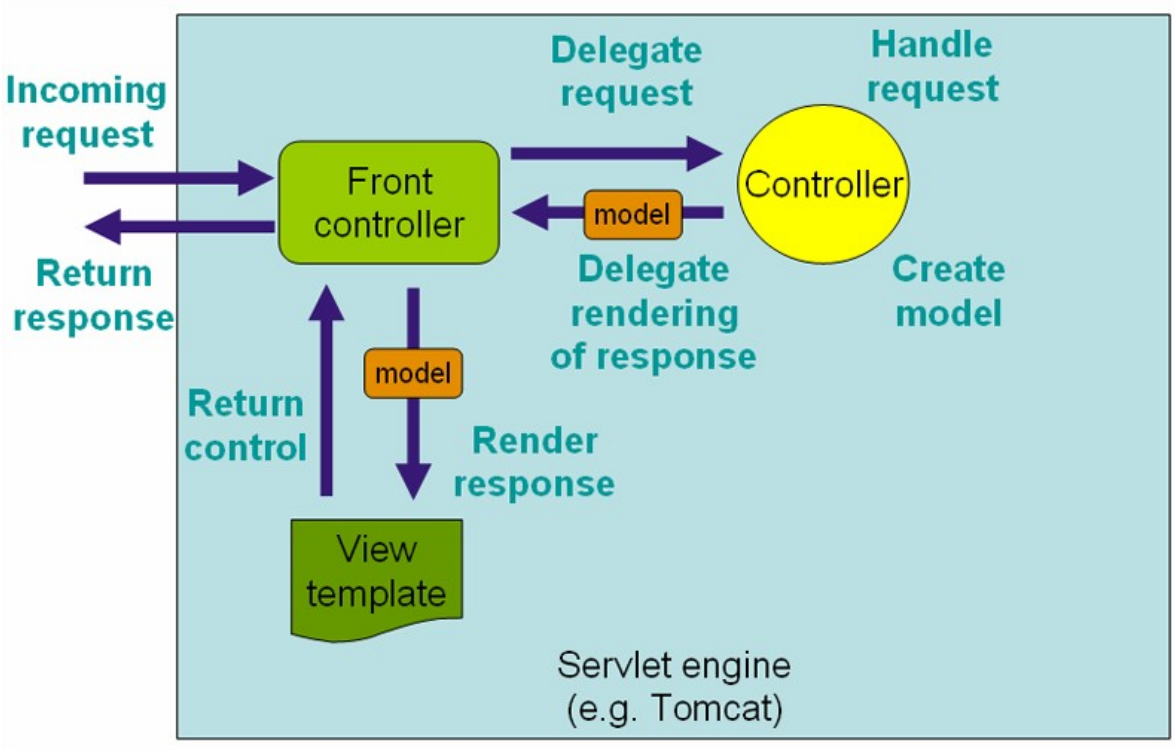
1.3. Exemple élémentaire (SpringMvc + JSP):

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HelloWorldController {

   @RequestMapping("/helloWorld")
   public String helloWorld(Model model) {
      model.addAttribute("message", "Hello World!");
      return "showMessage";
   }
}
```

Au niveau de **/jsp/**showMessage**.jsp**, l'affichage de message pourra être effectué via **${message}**.

```
<html><head><title>showMessage</title></head>
<body>
   <p>message=<b>${message}</b></p>
</body></html>
```

Rappel du principe de fonctionnement de SpringMvc :



2. éléments essentiels de Spring web MVC

2.1. éventuelle génération directe de la réponse HTTP

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller //but not "@Component" for spring web controller
@RequestMapping("/app")
public class WelcomeCtrl {

        @RequestMapping("/hello")
          @ResponseBody //si @ResponseBody , génération directe de la réponse ,
                        // sinon viewResolver (.jsp ou .html thymeleaf)
          String say_hello() {
             return "Hello World!";
          }
}
```

2.2. @RequestParam (accès aux paramètres HTTP)

conversion.jsp

```
… <form action="doConversion" method="GET_ou_POST">
          source: <select name="source" >
                <c:forEach var="d" items="${allDevises}">
                     <option  value="${d.monnaie}">${d.monnaie}</option>
                </c:forEach>
               </select>  <br/>
          cible: <select name="cible" > ... </select> <br/>
          montant: <input name="montant"  value="${montant}"/> <br/>
          <input  type="submit" value="convertir" /> <br/>
        </form>
        sommeConvertie=<b>${sommeConvertie}</b>   ...
```

```
@RequestMapping("/doConversion")
public String doConversion(Model model,@RequestParam(name="montant")double montant,
                                       @RequestParam(name="source")String monnaieSrc,
                                       @RequestParam(name="cible")String monnaieDest) {
….
   model.addAttribute("sommeConvertie",
                      gestionDevises.convertir(montant, monnaieSrc, monnaieDest));
   return "conversion";
}
```

2.3. @ModelAttribute

Pour spécifier un attribut du modèle on peut appeler ***model.addAttribute("attrName", attrVal);*** au sein d'une méthode préfixée par @RequestMapping .

Une autre solution consiste à coder une méthode addXyModelAttribute() préfixée par @ModelAttribute("attrName").

Exemple :

```
@ModelAttribute("conv")
        public ConversionForm addConvAttributeInModel() {
                return new ConversionForm();
        }
```

Le framework "spring mvc" va alors appeler automatiquement (*) toutes les méthodes préfixées par ***@ModelAttribute*** pour initialiser certains attributs du modèle avant de déclencher les méthodes préfixées par *@RequestMapping*.

L'appel n'est effectué que pour initialiser la valeur d'un attribut n'existant pas encore (pas d'écrasement des valeurs en session ni des valeurs saisies via <form:form …./>)

Une méthode préfixée par ***@ModelAttribute*** peut éventuellement avoir un paramètre préfixé par ***@RequestParam(name="numCli",required=true_or_false)*** mais elle n'a pas le droit de retourner une valeur "null" pour un attribut du modèle .

Variante syntaxique (en void et avec model) pour de multiples initialisations :

```
@ModelAttribute
        public void addAttributesInModel(Model model){
  model.addAttribute("xx", new Cxx());
  model.addAttribute("yy", new Cyy());
}
```

Autre Exemple :

```
@Controller //but not "@Component" for spring web controller
//@Scope(value="singleton")//by default
@RequestMapping("/devises")
public class DeviseListCtrl {

        @Autowired //ou @Inject
        private GestionDevises gestionDevises;

        private List<Devise> listeDevises = null; //cache
```

```
        @PostConstruct
        private void loadListeDevises(){
                if(listeDevises==null)
                        listeDevises=gestionDevises.getListeDevises();
        }

    @ModelAttribute("allDevises")
        public List<Devise> addAllDevisesAttributeInModel() {
            return listeDevises;
        }

         @RequestMapping("/liste")
           public String toDeviseList(Model model) {
              //model.addAttribute("allDevises", listeDevises);
              return "deviseList";
           }
}
```

deviseList.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>liste des devises</title>
</head>
<body>
      <h3>liste des devises (spring web mvc)</h3>
      <table border="1" >
      <tr><th>code</th><th>devise</th><th>change</th></tr>
            <c:forEach var="d" items="${allDevises}">
                  <tr><td>${d.codeDevise}</td><td>${d.monnaie}</td>
                        <td>${d.DChange}</td></tr>
            </c:forEach>
      </table>
      <hr/>
      <a href="../app/to_welcome">retour page accueil</a> <br/>
</body>
</html>
```

<u>Accès à un attribut pour effectuer une mise à jour</u>:

```
@RequestMapping("/info")
   public String toInfosClient(Model model) {
      //mise à jour du telephone du client 0L (pour le fun / la syntaxe):
                Client cli = (Client) model.asMap().get("customer");
                if(cli!=null && cli.getNumero()==0L)
                        cli.setTelephone("0102030405");
      return "infosClient";
   }
```

2.4. @SessionAttributes

```
@Controller
//@Scope(value="singleton")//by default
@RequestMapping("/client")
@SessionAttributes( value={"customer"} )
   //noms des "modelAttributes" qui sont EN PLUS récupérés/stockés
   // en SESSION HTTP au niveau de la page de rendu
   // --> visibles en requestScope ET en sessionScope
public class ClientCtrl {


   //NB: @SessionAttributes et @ModelAttribute sont gérés avant @RequestMapping


        @ModelAttribute("customer") //NB: cette méthode n'est pas appelée/déclenchée
                        //si "customer" est déjà présent en session (et par copie) dans le modèle
        public Client addCustomerAttributeInModel() {
                        return new Client(0L,null,null) ;
        }
}
```

Mettre fin à une session http:

```
@RequestMapping("/endSession")
   public String endSession(Model model,HttpSession session, SessionStatus sessionStatus) {
     /* if(model.containsAttribute("customer"))
                model.asMap().remove("customer"); */
      session.invalidate();
      sessionStatus.setComplete();
      return "infosClient";
   }
```

2.5. tags pour formulaires JSP (form:form , form:input , ...)

Spring-mvc offre une bibliothèque de tags permettant de simplifier la structuration d'une page JSP comportant un formulaire (à saisir , à valider , ….).

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
```

Ces nouvelles balises préfixées par *form:* s'utilisent quasiment de la même façon que les balises standards HTML (path="nomPropJava" à la place de name="nomParamHttp").

La principale valeur ajoutée des balises préfixées par *form:* consiste dans les liaisons automatiques entre certaines propriétés d'un objet java et les champs d'un formulaire.

Les balises <form:input ...> , <form:select ….> doivent être imbriquées dans <form:form >.

La balise principale d'un formulaire <***form:form*** *action="actionXY"* ***modelAttribute****="beanName" method="POST" > ... <form:form>* … comporte un attribut clef **modelAttribute** qui doit correspondre à un nom de "modelAttribute" lui même associé à un **objet java comportant toutes les données du formulaire à soumettre**.

Autrement dit , form:form ne fonctionne correctement que si la classe du sous-contrôleur est structurée avec au moins un "@ModelAttribute" (existant dès le départ , pas "null") dont le type correspond à une classe souvent spécifique au formulaire (ex : "UserForm" , "OrderForm" , ….) .

Exemple:

```
public class ConversionForm {
	private Double montant;
	private String monnaieSrc;
	private String monnaieDest;

	public ConversionForm(){
		monnaieSrc="dollar";
		monnaieDest="dollar"; //par défaut (dans formulaire avant saisies)
	}
	 //+ get/set
}
```

```
@Controller
//@Scope(value="singleton")//by default
@RequestMapping("/devises")
public class DeviseListCtrlV2 {
…
   //pour modelAttribute="conv" de  form:form
   @ModelAttribute("conv")
	public ConversionForm addConvAttributeInModel() {
		return new ConversionForm();
	}
…
}
```

L'attribut path="..." des sous balises <form:input ...> , <form:select ….> font alors référence aux propriétés de l'objet java (en lecture/écriture , get/set) .

NB: <form:form …> gère (génère) automatiquement le champ caché **_csrf** attendu par **spring-security** . *Exemple* : <input type="hidden" name="_csrf" value="8df91b84-74c1-4013-bd44-ede7b00779a2" />) . Ce champ caché correspond au "*Synchronizer Token Pattern*" (que l'on retrouve dans les frameworks web concurrents "Stuts" ou "JSF") : le coté serveur compare la valeur d'un jeton aléatoire stockée en session http avec celle stockée dans un champ caché et refuse de gérer la requête "re-postée" si la comparaison n'est pas réussie.

D'autre part , le terme ***CSRF*** (signifiant "*Cross Site Request Forgery*" correspond à un éventuel problème de sécurité : un site "malveillant" (utilisé en parallèle au sein d'un navigateur) déclenche automatiquement (via javascript ou autre) des requêtes non voulues (ex : virement monétaire) en utilisant le contexte d'un site à priori de confiance (mais pas assez protégé) .

Avec <form> (au lieu de <form:form>) , il faut insérer nous même le champ suivant au sein du formulaire d'une page ".jsp" :

<input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>

conversionV2.jsp

```
<form:form action="doConversion"  modelAttribute="conv" method="POST">
  source: <form:select path="monnaieSrc" >
                <form:options items="${allDevises}" itemLabel="monnaie" itemValue="monnaie"/>
          </form:select>  <br/>
  cible: <form:select path="monnaieDest" >
                <form:options items="${allDevises}" itemLabel="monnaie" itemValue="monnaie"/>
        </form:select> <br/>
   montant: <form:input path="montant" />
              <form:errors path="montant" cssClass="error"/><br/>
   <input  type="submit" value="convertir" /> <br/>
</form:form>
sommeConvertie=<b>${sommeConvertie}</b>
```

conversion de devises

source: dollar
cible: euro
montant: 45.0
convertir
sommeConvertie=**37.5**

Finalement , au sein du contrôleur , la méthode déclenchée par le formulaire peut s'écrire de la façon suivante:

```
@RequestMapping("/doConversion")
public String doConversion(Model model,@ModelAttribute("conv")  ConversionForm conv ) {

model.addAttribute("sommeConvertie",
                        gestionDevises.convertir(conv.getMontant(),
                                                 conv.getMonnaieSrc(), conv.getMonnaieDest()));

  return "conversionV2";
}
```

2.6. validation lors de la soumission d'un formulaire

Rappel: la classe de l'objet utilisé en tant que "modelAttribute" au niveau d'un formulaire peut comporter des annotations @Min , @Max , @Size , @NotEmpty , … de l'api normalisée javax.validation .

Exemples :

```
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;

public class ConversionForm {

        @Min(value=0)
        @Max(value=999999)
        private Double montant;
    …
}
```

```
import javax.validation.constraints.Size;
import org.hibernate.validator.constraints.Email;
import org.hibernate.validator.constraints.NotEmpty;

public class Client {
        private Long numero;    private String nom;       private String prenom;

        @NotEmpty(message = "Please enter your address.")
        @Size(min = 4, max = 128, message = "Your address must between 4 and 128 characters")
        private String adresse;
        private String telephone;

        @NotEmpty
        @Email
        private String email;
…
}
```

Il suffit en suite d'ajouter **@Valid** au niveau du paramètre de la méthode associée à la soumission du formulaire pour que spring-mvc tienne compte des contraintes de validation.

D'autre part, le paramètre (facultatif mais conseillé) de type "**BindingResult**" permet de gérer finement les cas d'erreur de validation :

```
@RequestMapping("/doConversion")
public String doConversion(Model model,
                                @ModelAttribute("conv") @Valid ConversionForm conv ,
                                BindingResult bindingResult) {
        if (bindingResult.hasErrors()) {
                // form validation error
                System.out.println("form validation error: " + bindingResult.toString());
        } else {
```

```
                // form input is ok*/
        model.addAttribute("sommeConvertie", gestionDevises.convertir(conv.getMontant(),
                                                conv.getMonnaieSrc(), conv.getMonnaieDest()));
        }
return "conversionV2";
}
```

conversion de devises

source: dollar
cible: livre
montant: -5.0 **doit être plus grand que 0**
convertir
sommeConvertie=

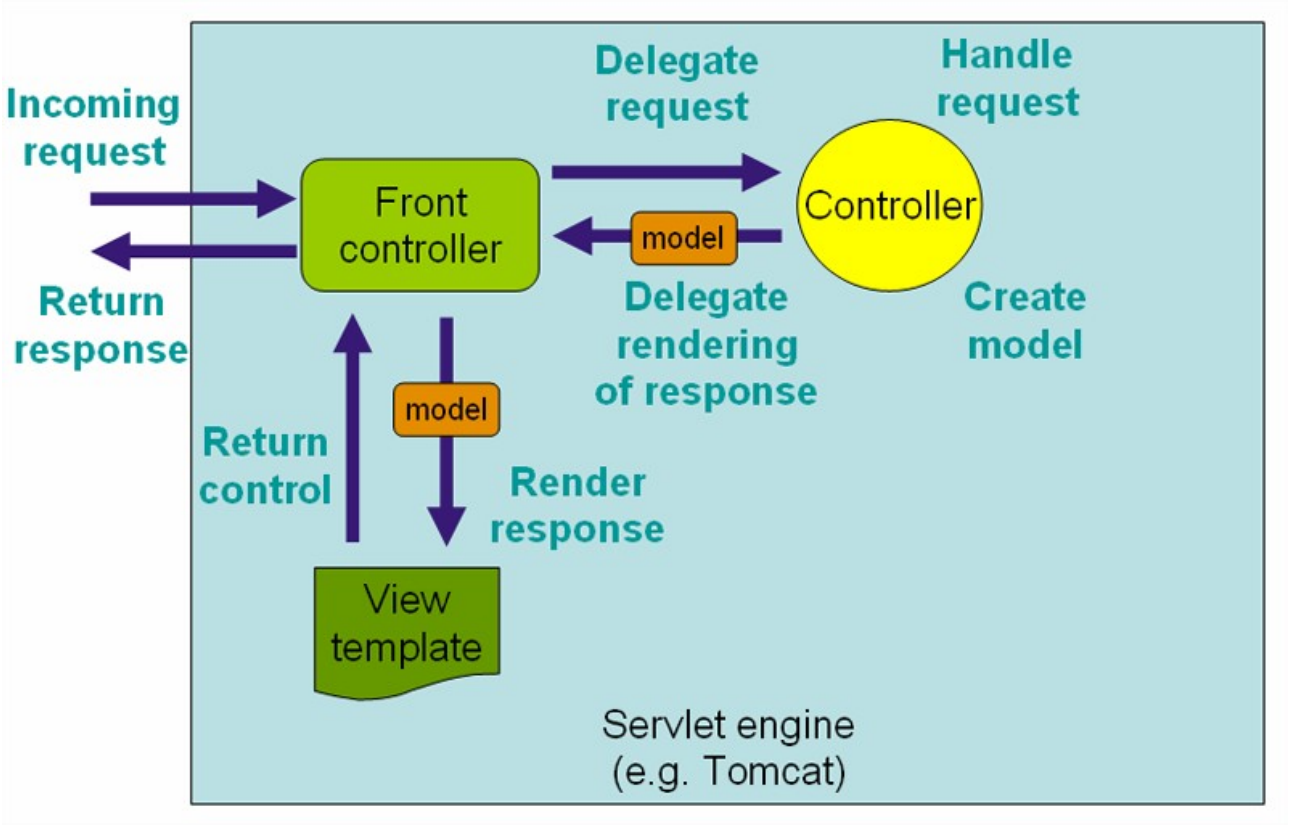
---

retour page accueil

numero: 0
nom: Therieur
prenom: alex
adresse: ici **Your address must between 4 and 128 characters**
telephone:
email: alex-therieur **Adresse email mal formée**
update

3. Spring-Mvc avec Thymeleaf

3.1. Vues en version Thymeleaf



Les vues peuvent être en version "**thymeleaf**" plutôt que "~~JSP~~"

3.2. Spring-mvc avec Thymeleaf

La technologie "**Thymeleaf**" est une alternative intéressante vis à vis des pages JSP et qui offre les avantages suivants :

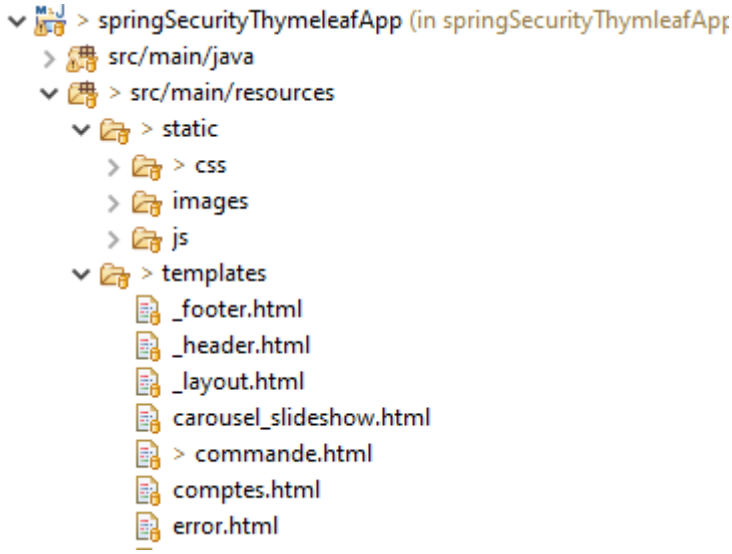
- **syntaxe plus développée** (plus concise , plus expressive , plus sophistiquée)
- **meilleures possibilités/fonctionnalités pour la mise en page** (héritage de layout , ...)
- technologie assez souvent utilisée avec SpringMvc et SpringBoot

Rappel des dépendances maven nécessaires :

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

        <dependency> < !-- pour @Max , ... @Valid -->
             <groupId>org.springframework.boot</groupId>
             <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>
       <dependency>
              <groupId>org.springframework.boot</groupId>
              <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>

    <dependency>
         <groupId>nz.net.ultraq.thymeleaf</groupId>
         <artifactId>thymeleaf-layout-dialect</artifactId>
     </dependency>

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

      <dependency>
        <groupId>org.thymeleaf.extras</groupId>
        <artifactId>thymeleaf-extras-springsecurity5</artifactId>
     </dependency>
  -->
```

Sans configuration spécifique dans application.properties le répertoire prévu pour accueillir les templates de **thymeleaf** est **src/main/resources/templates** .

Sachant que les fichiers annexes **".css" , ".js"** , ... sont à ranger dans **src/main/resources/static**.



Il n'y a pas de différence notable dans l'écriture des contrôleurs (JSP ou Thymeleaf : peu importe).

3.3. "Hello world" avec Spring-Mvc et Thymeleaf

src/main/resources/static/**index.html**

```
<a href="site/app/hello-world-th">hello-world-th (via Spring Mvc et thymeleaf)</a> <br/>
```

AppCtrl.java

```
package tp.appliSpring.web.controller;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/site/app")
public class AppCtrl {

    @RequestMapping("/hello-world-th")
    public String helloWorld(Model model) {
            model.addAttribute("message", "Hello World!");
            return "showMessage"; //aiguiller sur la vue "showMessage"
    }
}
```

src/main/resources/templates/showMessage.html

```
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <title>showMessage</title>
</head>
<body>
   message: <span th:utext="${message}" ></span>
</body>
</html>
```

Résultat :

```
message: Hello World!
```

3.4. Templates thymeleaf avec layout

Voici quelques exemples de "vues/templates" basés sur la technologie "Thymeleaf" :

_header.html

```
<header xmlns:th="http://www.thymeleaf.org"  th:fragment="_header">
 <div style="width:100%;font-size:36px;line-height:48px;background-color:navy;color:white">
          My SpringMVC Thymeleaf Application</div> </header>
```

_footer.html

```
<footer xmlns:th="http://www.thymeleaf.org"   th:fragment="_footer">
 <div style="background-color:navy;width:100%;color:white">
Mon pied de page ... <a th:href="@{/to-welcome}" style="color:yellow" >welcome</a>
</div> </footer>
```

NB :

- **th:href="@{/to-welcome}" au sens th:href="@{*controller_requestMapping*}"**
- les ***sous fichiers*** _header.html et _footer.html seront **inclus** dans _layout.html via **th:replace**="..."

Le fichier *_layout.html* suivant correspond à un template/**modèle commun/générique de mise en page** . La plupart des pages ordinaires de l'application reprendront (par héritage) la structure de _layout.html .

Le contenu des zones identifiées par **layout:fragment**="***nomLogiqueFragment***" pourront si besoin est redéfinies/remplacées au sein des futures pages basées sur ce template :

_layout.html

```
<html xmlns:th="http://www.thymeleaf.org"
        xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout" >
  <head>
     <meta charset="UTF-8" />
     <title layout:fragment="title" th:utext="${title}"></title>
     <link rel="stylesheet" type="text/css" th:href="@{/css/bootstrap.min.css}"/>
     <link rel="stylesheet" type="text/css" th:href="@{/css/styles.css}"/>
  </head>
  <body>
    <div class="container-fluid">
     <div th:replace="_header"></div>
     <div layout:fragment="content">
       default content from _layout.html (to override)
     </div>
     <div th:replace="_footer"></div>
    </div><!-- end of bootstrap css container-fluid -->
  </body>
</html>
```

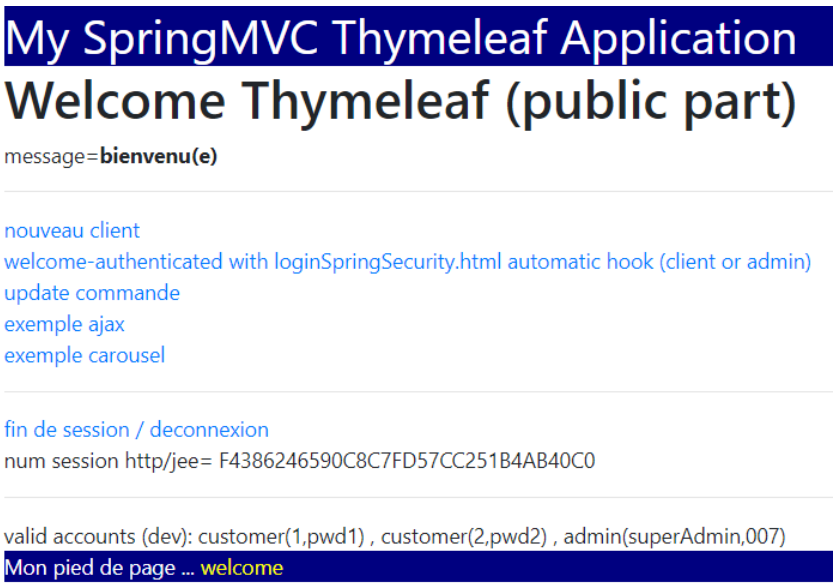
Le fichier *welcome.html* suivant est basé sur le modèle générique ***_layout.html*** via le lien d'héritage / de composition **layout:decorate="~{_layout}" .**

Au sein de *welcome.html* , tout le contenu imbriqué entre début et fin de la balise marquée via **layout:fragment="*content*"** va automatiquement remplacer le texte *default content from _layout.html (to override)* qui était encadré par la même nom logique de fragment au sein de ***_layout.html*** .

Le rendu globalement fabriqué par Thymeleaf sera ainsi une page HTML complète ayant comme structure celle de _layout.html (et donc avec _header et _footer par défaut) et dont le fragment "content" aura été redéfini avec un contenu spécifique à welcome.html .

welcome.html

```
<div xmlns:th="http://www.thymeleaf.org"
    xmlns:sec="http://www.thymeleaf.org/extras/spring-security"
    xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
    layout:decorate="~{_layout}"    layout:fragment="content">
  <h1>Welcome Thymeleaf (public part)</h1>
  <p>message=<b><span th:utext="${message}"></span></b></p>
  <hr/> ...  /div>
```



Exemple de formulaire ultra simple (élémentaire) avec thymeleaf :

```
…
<form th:action="@{/site/compte/verifLogin}" method="POST">
    numClient : <input name="numClient" type="text" /> <br/>
   <input  type="submit" value="identification client banque" /> <br/>
   <input type="hidden"   th:name="${_csrf.parameterName}"      th:value="${_csrf.token}"/>
</form> ...
```

en liaison avec

```
@Controller
@RequestMapping("/site/compte")
public class CompteController {
        @RequestMapping("/verifLogin")
         public String verifLogin(Model model,
                                  @RequestParam(name="numClient")  Long numClient) { … } }
```

```
NB:  (@RequestParam(name="numClient", required = false)  Long numClient) {
  if( numClient==null) { ….} else {….}
}
```

```
NB2: monCalcul(Model model, @RequestParam(name="val",defaultValue = "0") double val)
peut être pratique pour récupérer (la première fois) une valeur par défaut si l'on est pas encore passé
par un petit formulaire de saisie
```

Exemple de formulaire simple avec thymeleaf :

```
<div xmlns:th="http://www.thymeleaf.org"
     xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
     layout:decorate="~{_layout}"      layout:fragment="content">
   <h3>nouveau (client banque)</h3>
        <hr/>
        <form th:action="@{/nouveauClient}"
            th:object="${client}" method="POST">
           numero : <input th:field="*{numero}" type="text" /> <br/>
           password : <input th:field="*{password}" type="text" /> <br/>
           nom : <input th:field="*{nom}" type="text" /> <br/>
           prenom : <input th:field="*{prenom}" type="text" /> <br/>
           <input  type="submit" value="enregistrer nouveau client" /> <br/>
        </form>
</div>
```

pour déclencher :

```
@RequestMapping(value="/nouveauClient" )
   public String nouveauClient(Model model,
                @ModelAttribute("client") Client client) {
        //client avec propriétés .getNom() , .getPrenom() , ...
   }
```

Rappels sur la gestion d'une session Spring-Mvc :

Pour spécifier un attribut du modèle on peut appeler ***model.addAttribute("attrName", attrVal);*** au sein d'une méthode préfixée par @RequestMapping .

Une autre solution consiste à coder une méthode addXyModelAttribute() préfixée par @ModelAttribute("attrName").

Exemple :

```
@ModelAttribute("client")
        public Client addClientAttributeInModel() {
                return new Client();
        }
```

Le framework "spring mvc" va alors appeler automatiquement (*) toutes les méthodes préfixées par

@ModelAttribute pour initialiser certains attributs du modèle avant de déclencher les méthodes préfixées par *@RequestMapping*.

L'appel n'est effectué que pour initialiser la valeur d'un attribut n'existant pas encore (pas d'écrasement des valeurs en session ni des valeurs saisies via <form:form …./>)

D'autre part, sur une classe de type @Controller on peut placer l'annotation

@SessionAttributes(value={"client" , "caddy" , "…" })

de manière à préciser les noms des "modelAttributes" qui sont EN PLUS récupérés/stockés en SESSION HTTP au niveau de la page de rendu --> visibles en requestScope ET en sessionScope

NB: au sein d'un template "thymeleaf" , on peut accéder aux attributs stockés en session via la syntaxe **${#ctx.session.attributeName}**

Exemple :

<input type="text" name="numClient" **th:value="${#ctx.session.numClient}"** />

ou bien ${#ctx.session.client.num} selon les structures de données choisies

Logout avec Spring-Mvc (jsp ou thymeleaf) :

```
@RequestMapping("/logout")
        public String userLogout(Model model,
                                 HttpSession httpSession,
                                 SessionStatus sessionStatus) {
                httpSession.invalidate();
                sessionStatus.setComplete();
                model.addAttribute("message", "session terminée");
                model.addAttribute("title","welcome");
                return "welcome";
        }
```

Exemple partiel de formulaire complexe avec thymeleaf :

```
<div xmlns:th="http://www.thymeleaf.org"
     xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
     layout:decorate="~{_layout}"      layout:fragment="content">
   <script>        function onDelete(idToDelete,bToDelete){ /* .... */    }    </script>
   <h3>commande</h3>
          <hr/>
          <form th:action="@{/update-commande}"    th:object="${cmdeF}" method="POST">
             numero : <label th:text="*{cmde.numero}" ></label>
                    <input th:field="*{cmde.numero}" type="hidden" /><br/>
             sDate : <input th:field="*{cmde.sDate}" type="text" /> <br/>
             id (client) : <label th:text="*{cmde.client.id}" ></label>
                       <input th:field="*{cmde.client.id}" type="hidden" /><br/>
             nom (client) : <input th:field="*{cmde.client.nom}" type="text"
                              th:class="*{cmde.client.nom == 'Bon' } ? 'enEvidence' : '' " /> <br/>
             prenom (client): <input th:field="*{cmde.client.prenom}" type="text" /> <br/>
             <div th:each="p, rowStat  : *{cmde.produits}">
       <hr/>
            ref (produit) : <label th:text="*{cmde.produits[__${rowStat.index}__].ref}" ></label>
                         <input th:field="*{cmde.produits[__${rowStat.index}__].ref}" type="hidden" /><br/>
           label (produit) : <input type="text" th:field="*{cmde.produits[__${rowStat.index}__].label}" />
           prix (produit) : <input type="text" th:field="*{cmde.produits[__${rowStat.index}__].prix}" />
           : <input type="checkbox" value="delete"
              th:onclick=" 'onDelete(' + *{cmde.produits[__${rowStat.index}__].ref} + ',this.checked)' " /> delete

                    </div>
             <hr/>
              nb new product to add : <input th:field="*{prodActions.nbNew}" type="text" /> <br/>
              id of product(s) to delete :<input th:field="*{prodActions.idsToDelete}"
                                                  class="RedCssClass" type="text" /> <br/>
             <input  type="submit" value="update commande" /> <br/>
          </form>
</div>
```

numero : 1

sDate : 2021-05-27

id (client) : 1

nom (client) : Bon

prenom (client): Jean

ref (produit) : 1

label (produit) : produit1 prix (produit) : 5.5 : ☑ delete

ref (produit) : 2

label (produit) : produit2 prix (produit) : 6.6 : ☑ delete

ref (produit) : 3

label (produit) : produit3 prix (produit) : 7.7 : ☐ delete

nb new product to add : 0

id of product(s) to delete : 1 2

update commande

3.5. Principales syntaxes pour thymeleaf

Affichage des erreurs de validations (si **@Valid** du coté contrôleur) :

```
<form th:action="@{/site/….}"   th:object="${….}">
   <label>montant:</label>
   <input type="text" name="nom" th:field="*{montant}" />
        <span th:if="${#fields.hasErrors('montant')}"
            th:errorclass="error" th:errors="*{montant}" >
        </span>
...
```

Saisie de dates (LocalDate coté java) :

```
<input type="text" name="dateDebut" th:field="*{dateDebut}" />
(ex: 2024-12-25 ou 25/12/2024)<br/>
```

Case à cocher en liaison avec un booléen :

```
<label class="simpleAlign">célibataire:</label>
<input type="checkbox"  name="celibataire" th:checked="*{celibataire}" />
```

Boutons radios exclusifs :

```
<input type="radio" name="situation" th:value="CELIBATAIRE" th:field="*{situation}" />
        <span th:utext="CELIBATAIRE"></span>    
<input type="radio" name="situation" th:value="EN_COUPLE" th:field="*{situation}" />
        <span th:utext="EN_COUPLE"></span>
```

Liste déroulante :

```
<select th:field="*{numProduit}" >
    <option th:each ="p : ${listeProduits}" th:value="${p.numero}"
          th:utext="${p.numero} + ' - ' + ${p.label} + ' : ' + ${p.prix}"></option>
</select> <br/>
```

Eventuelle boucle sur valeurs possibles d'une énumération :

```
<…. th:each="j : ${T(tp.appliSpringMvc.web.model.Jour).values()}">
        <... th:value="${j}" ...>
</...>
```

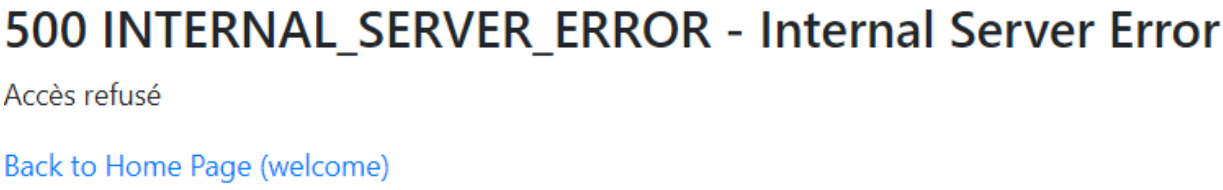
3.6. Sécurité avec SpringMvc et thymeleaf

Si thymeleaf est utilisé conjointement avec **spring-security** alors les éléments suivants peuvent être utiles :

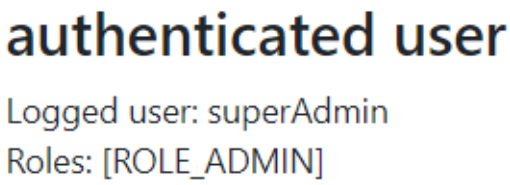
```
@ControllerAdvice
public class ErrorCtrlAdvice {
   private static Logger logger = LoggerFactory.getLogger(ErrorController.class);

   @ExceptionHandler(Throwable.class)
   @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
   public String exception(final Throwable throwable, final Model model) {
      logger.error("Exception during execution of SpringSecurity application", throwable);
      String errorMessage = (throwable != null ? throwable.getMessage() : "Unknown error");
      model.addAttribute("errorMessage", errorMessage);
      return "error";
   }
}
```

error.html

```
<!DOCTYPE HTML>
<div xmlns:th="http://www.thymeleaf.org"
     xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
     layout:decorate="~{_layout}"      layout:fragment="content">


  <div th:with="httpStatus=$
{T(org.springframework.http.HttpStatus).valueOf(#response.status)}">
       <h3 th:text="|${httpStatus} - ${httpStatus.reasonPhrase}|">404</h3>
       <p th:utext="${errorMessage}">Error</p>
       <a href="welcome.html" th:href="@{/to-welcome}">Back to Home Page (welcome)</a>
   </div>
</div>
```



```
<!-- necessite thymeleaf-extras-springsecurity5_ou_6 dans pom.xml -->
<div xmlns:th="http://www.thymeleaf.org"
     xmlns:sec="http://www.thymeleaf.org/extras/spring-security"
     xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
     layout:decorate="~{_layout}"     layout:fragment="content">
   <h1>Welcome Thymeleaf (for Authenticated user)</h1>
   <p>message=<b><span th:utext="${message}"></span></b></p>
   <hr/>
   <div sec:authorize="isAuthenticated()">
      <h3>authenticated user </h3>
      Logged user: <span sec:authentication="name">Unknown</span> <br/>
      Roles: <span sec:authentication="principal.authorities">[]</span> <br/>
   </div>
   <hr/>
....
</div>
```



Quelques liens hypertextes pour approfondir "thymeleaf" :

- https://www.thymeleaf.org/documentation.html
- https://www.thymeleaf.org/doc/tutorials/3.0/thymeleafspring.html
- https://gayerie.dev/docs/spring/spring/thymeleaf.html

VI - Annexe – Selon versions (ajustements)

1. Migration de Spring 5 vers Spring 6

1.1. Changement de dépendances (pom.xml)

Souvent indirectement via Spring-boot .

Dépendances directes et explicites pour Spring6 sans SpringBoot *:*

```
<properties>
        <java.version>17</java.version>
        <spring.version>6.1.2</spring.version>
        <junit.jupiter.version>5.10.1</junit.jupiter.version>
</properties>
…
<dependency>
        <groupId>jakarta.inject</groupId>
        <artifactId>jakarta.inject-api</artifactId>
        <version>2.0.1</version>
</dependency> <!-- pour que Spring puise interpreter @Inject comme @Autowired -->
<dependency>
        <groupId>jakarta.servlet</groupId>
        <artifactId>jakarta.servlet-api</artifactId>
        <version>6.0.0</version>
        <scope>provided</scope> <!-- provided by tomcat after deploying .war -->
</dependency>
<dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>6.4.1.Final</version>
</dependency> <!-- et indirectement jpa en version jakarta -->
<dependency>
        <groupId>jakarta.annotation</groupId>
        <artifactId>jakarta.annotation-api</artifactId> <!-- @PostConstruct -->
        <version>2.1.1</version>
</dependency>
<dependency>
        <groupId>jakarta.servlet.jsp.jstl</groupId>
        <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
        <version>3.0.0</version>
</dependency>
<dependency>
        <groupId>org.glassfish.web</groupId>
        <artifactId>jakarta.servlet.jsp.jstl</artifactId>
        <version>3.0.0</version>
</dependency>
<!-- springdoc-openapi-ui for spring5/springBoot2 ,
     springdoc-openapi-starter-webmvc-ui for spring6/springBoot3 -->
<dependency>
        <groupId>org.springdoc</groupId>
        <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
```

```
        <version>2.3.0</version>
</dependency>
```

1.2. Changement de packages

Spring5 et JEE <=8	*Spring 6 et JEE >=9*
javax.persistence.*	**jakarta.persistence.***
javax.servlet.http	**jakarta.servlet.http**
javax.annotation	jakarta.annotation
...	

Exemples (spring 6) :

```
import jakarta.annotation.PostConstruct;
import jakarta.inject.Inject;
import jakarta.persistence.EntityManager;
import jakarta.persistence.PersistenceContext;
import jakarta.persistence.*
```

Entête de l'éventuel META-INF/persistence.xml en version 3.0 :

```
<persistence xmlns="https://jakarta.ee/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd"
        version="3.0">
```

taglib de jstl en version jarkarta :

```
<%@ taglib prefix="c" uri="jakarta.tags.core"%>
```

1.3. Continuité vis à vis des évolutions de SpringSecurity

Du coté de SpringSecurity , Encore des changements entre Spring 5 et Spring 6

peu après les changements de la version 5.7 de Spring .

Changements récents pour SpringSecurity associé à Spring6/SpringBoot3 :

```
//@EnableGlobalMethodSecurity(prePostEnabled = true) for Spring5
@EnableMethodSecurity //with default prePostEnabled = true
```

```
//Spring5/SpringBoot2 : http.authorizeRequests().antMatchers("....")....
//Spring6/SpringBoot3 : http.authorizeHttpRequests( auth -> auth.requestMatchers("....")....);
```

plus de .and(). mais des lambdas imbriquées

1.4. Changements dans les paramétrages de l'auto-configuration

Dans un sous projet de type ***xyz-configure*** , le paramétrage de l'auto-configuration à changé entre spring 5 et spring 6.

Avec Spring5/SpringBoot2:

ancien fichier ***META-INF/spring.factories***

comportant

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration=org.mycontrib.mysecurity.chain.autoconfigure.MySecurityAutoConfiguration
```

Avec Spring6/SpringBoot3:

Nouveau fichier
META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports

comportant

```
org.mycontrib.mysecurity.chain.autoconfigure.MySecurityAutoConfiguration
```

1.5. Nouvelles possibilités de java 17

- **Pattern matching** : `if (obj instanceof String s) { … }`
- **`switch expression`** (avec lambda et mot clef yield)
- **`record`** possibles au niveau des DTOs *(avec restrictions!!!)*
- ***`Text blocs`*** *(entre """ et """" , pratiques pour tests unitaires)*

1.6. *Nouvelles possibilités de Spring 6*

- **GraalVM Native Images (voir annexe détaillée)**
- ***Spring Observability (https://www.baeldung.com/spring-boot-3-observability)***

VII - Annexe – Web Services REST (concepts)

1. Deux grands types de WS (REST et SOAP)

2 grands types de services WEB: **SOAP/XML** et **REST/HTTP**

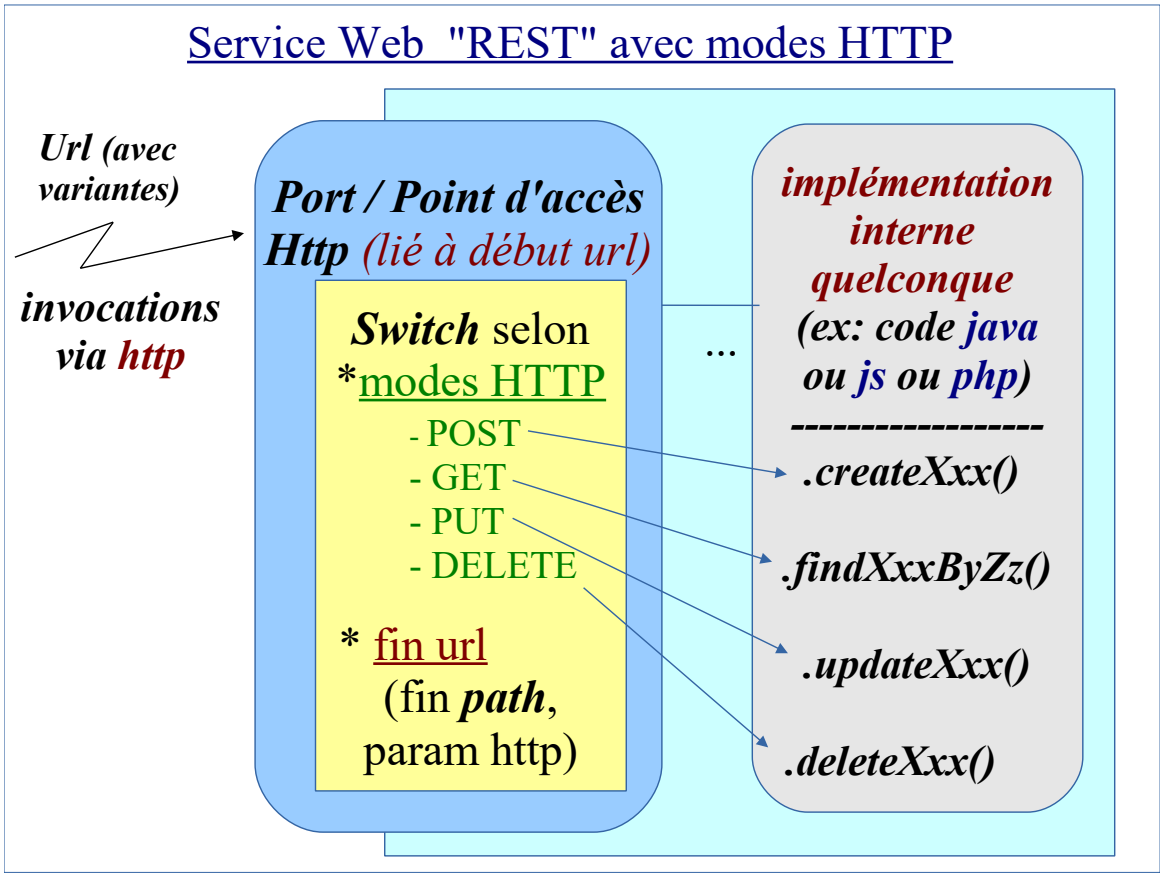
WS- (SOAP / XML)*

- "**Payload**" systématiquement en **XML** *(sauf pièces attachées / HTTP)*
- **Enveloppe SOAP** en XML *(header facultatif pour extensions)*
- **Protocole de transport au choix (HTTP, JMS, ...)**
- Sémantique quelconque (*appels méthodes*) , **description WSDL**
- **Plutôt** orienté Middleware SOA **(arrière plan)**

REST (HTTP)

- **"Payload" au choix (XML , HTML , JSON, ...)**
- Pas d'enveloppe imposée
- **Protocole de transport = toujours HTTP.**
- Sémantique "**CRUD**" (***modes http* PUT,GET,POST,DELETE)**
- **Plutôt** orienté IHM Web/Web2 **(avant plan)**

1.1. Caractéristiques clefs des web-services "REST" / "HTTP"



Points clefs des Web services "REST"

Retournant des données dans un format quelconque ("XML" , "JSON" et éventuellement "txt" ou "html") les web-services "REST" offrent des résultats qui nécessitent généralement peu de re-traitements pour être mis en forme au sein d'une IHM web.

Le format "au cas par cas" des données retournées par les services REST permet peu d'automatisme(s) sur les niveaux intermédiaires.

Souvent associés au format "JSON" les web-services "REST" conviennent parfaitement à des appels (ou implémentations) au sein du langage javascript .

La relative simplicité des URLs d'invocation des services "REST" permet des appels plus immédiats (*un simple href="..." suffit en mode GET pour les recherches de données*) .

La compacité/simplicité des messages "JSON" souvent associés à "REST" permet d'obtenir d'assez bonnes performances .

2. Web Services "R.E.S.T."

REST = style d'architecture (conventions)

REST est l'acronyme de **Representational State Transfert**.
C'est un **style d'architecture** qui a été décrit par ***Roy Thomas Fielding*** dans sa thèse *«Architectural Styles and the Design of Network-based Software Architectures».*

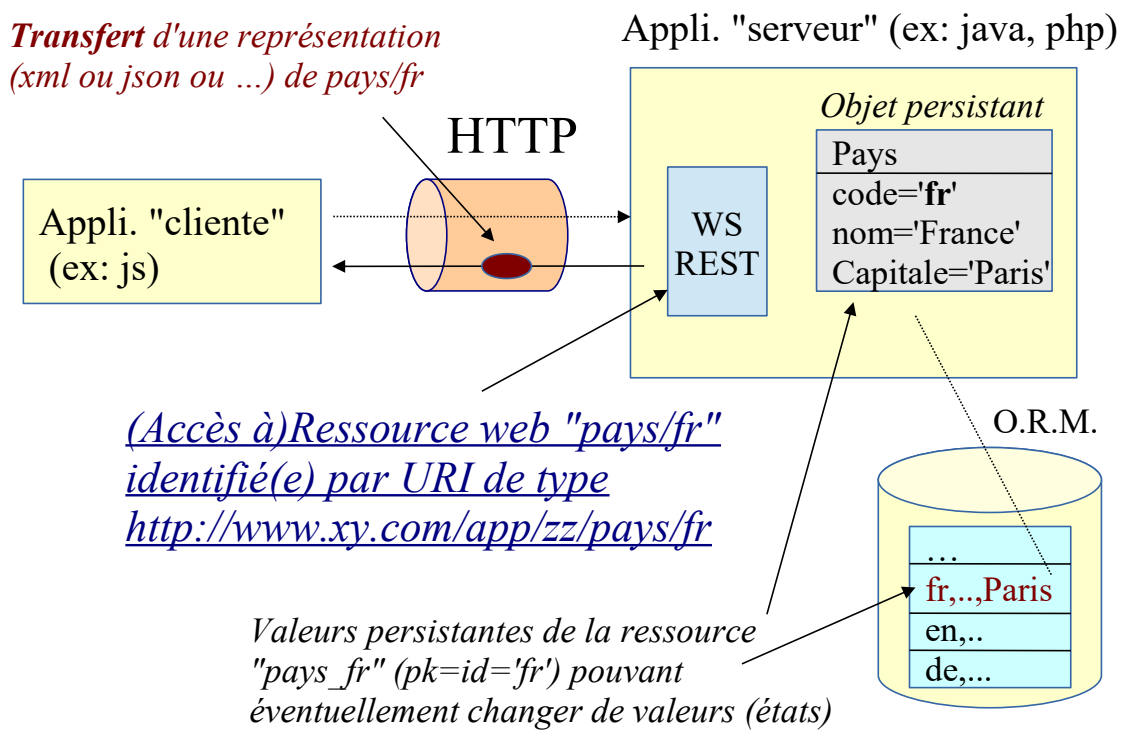
L'information de base, dans une architecture REST, est appelée **ressource**.
Toute information (à sémantique stable) qui peut être nommée est une ressource: un article , une photo, une personne , un service ou n'importe quel concept.

Une ressource est identifiée par un **identificateur de ressource**. Sur le web ces identificateurs sont les **URI** (Uniform Resource Identifier).

NB: dans la plupart des cas, une ressource REST correspond indirectement à un enregistrement en base (avec la *clef primaire* comme partie finale de l'uri "identifiant").

Les composants de l'architecture REST manipulent ces ressources en **transférant à travers le réseau** (via HTTP) des **représentations de ces ressources**.
Sur le web, on trouve aujourd'hui le plus souvent des représentations au format **HTML , XML ou JSON**.

REST : transferts de représentations de ressources



REST et principaux formats (xml,json)

Une invocation d'URL de service REST peut être accompagnée de données (en entrée ou en sortie) pouvant prendre des formats quelconques :
text/plain , text/html , application/xml , application/json , …

Dans le cas d'une lecture/recherche d'informations , le format du résultat retourné pourra (selon les cas) être :

- **imposé (en dur) par le code du service REST** .

- **au choix (xml , json) et précisé par une partie de l'url**

- **au choix (xml , json) et précisé par le champ "Accept :" de l'entête HTTP de la requête**. (exemple: Accept: application/json) .

Dans tous les cas, la réponse HTTP devra avoir son format précisé via le champ habituel ***Content-Type:*** *application/json* de l'entête.

Format JSON (JSON = *JavaScript Object Notation*)

Les 2 principales caractéristiques de JS0N sont :

- Le principe de clé / valeur (map)
- L'organisation des données sous forme de tableau

```
[
  {
    "nom": "article a",
    "prix": 3.05,
    "disponible": false,
    "descriptif": "article1"
  },
  {
    "nom": "article b",
    "prix": 13.05,
    "disponible": true,
    "descriptif": null
  }
]
```

Les types de données valables sont :

- tableau
- objet
- chaîne de caractères
- valeur numérique (entier, double)
- booléen (true/false)
- null

une liste d'articles

une personne

```
{
  "nom": "xxxx",
  "prenom": "yyyy",
  "age": 25
}
```

REST et méthodes HTTP (verbes)

Les **méthodes HTTP** sont utilisées pour indiquer la **sémantique des actions demandées** :
• **GET** : **lecture/recherche** d'information
• **POST** : **envoi** d'information
• **PUT** : **mise à jour** d'information
• **DELETE** : **suppression** d'information

Par exemple, pour récupérer la liste des adhérents d'un club, on peut effectuer une requête de type **GET** vers la ressource **http://monsite.com/adherents**

Pour obtenir que les adhérents ayant plus de 20 ans, la requête devient http://monsite.com/adherents?ageMinimum=20

Pour supprimer numéro 4, on peut employer une requête de type **DELETE** telle que **http://monsite.com/adherents/4**

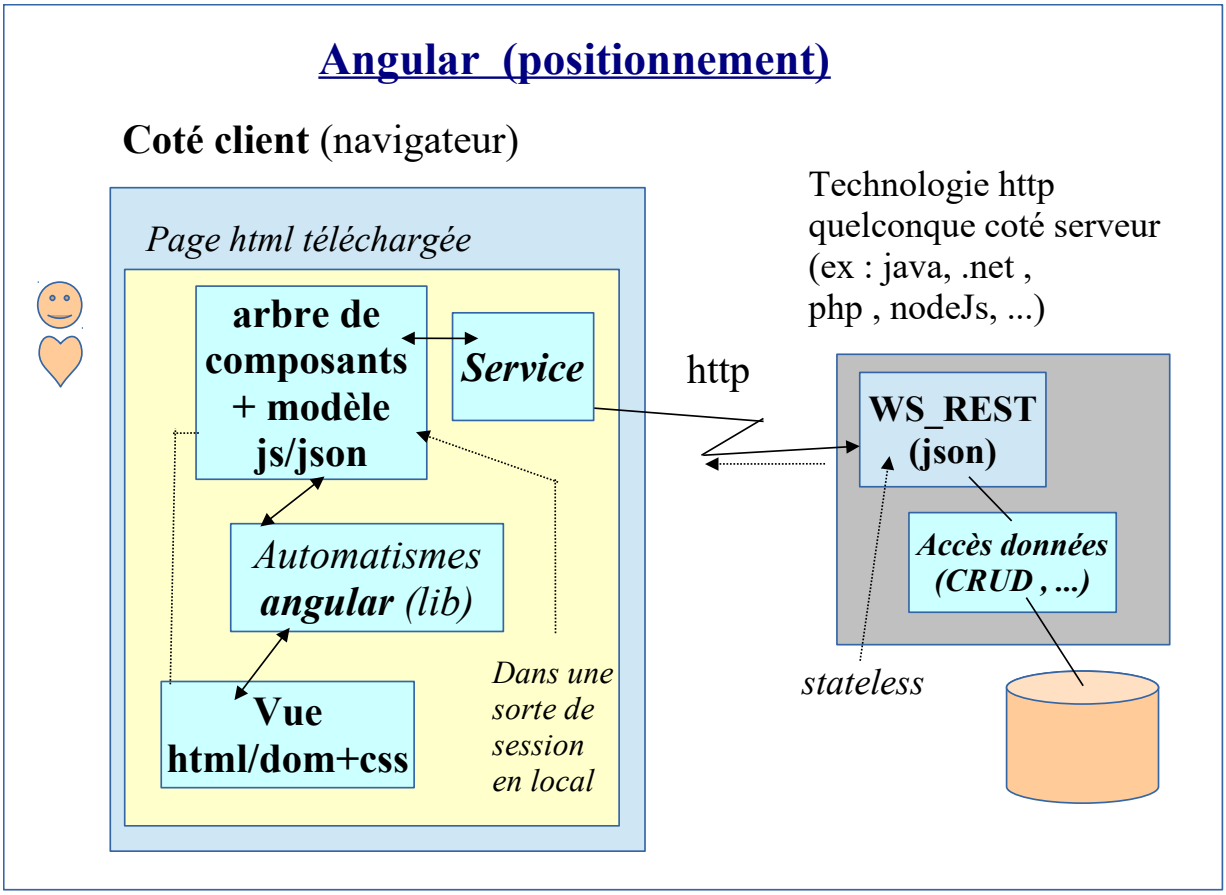
Pour envoyer des informations, on utilise **POST** ou **PUT** en passant les informations dans le corps (invisible) du message HTTP avec comme URL celle de la ressource web que l'on veut créer ou mettre à jour.

Exemple concret de service REST : "Elevation API"

L'entreprise "***Google***" fourni gratuitement certains services WEB de type REST.
"***Elevation API***" est un service REST de Google qui renvoie l'altitude d'un point de la planète selon ses coordonnées (latitude ,longitude) .
La documentation complète se trouve au bout de l'URL suivante :
https://developers.google.com/maps/documentation/elevation/?hl=fr
Sachant que les coordonnées du Mont blanc sont :
Lat/Lon : 45.8325 N / 6.86417 E *(GPS : 32T 334120 5077656)*
Les invocations suivantes (du service web rest "api/elevation")
http://maps.googleapis.com/maps/api/elevation/json?locations=45.8325,6.86417
http://maps.googleapis.com/maps/api/elevation/xml?locations=45.8325,6.86417
donne les résultats suivants "json" ou "xml":

```
{  "results" : [
     {
       "elevation" : 4766.466796875,
       "location" : {
         "lat" : 45.8325,
         "lng" : 6.86417
       },
       "resolution" : 152.7032318115234
     }
   ],   "status" : "OK"
}
```

```
?xml version="1.0" encoding="UTF-8"?>
<ElevationResponse>
 <status>OK</status>
 <result>
  <location>
   <lat>45.8325000</lat>
   <lng>6.8641700</lng>
  </location>
  <elevation>4766.4667969</elevation>
  <resolution>152.7032318</resolution>
 </result>
</ElevationResponse>
```



Conventions sur URL / Path des ressources REST

Type requêtes	HTTP *Method*	*URL ressource(s) distante(s)*	*Request body*	*Réponse JSON*
Recherche multiple	**GET**	**…/product** **…/product?crit1=v1&crit2=v2**	vide	Liste/tableau d'objets
Recherche par id	**GET**	**…/product/idRes** *(avec idRes=1,...)*	vide	Objet JSON
Ajout (seul)	**POST**	**…/product**	Objet JSON	Objet JSON avec id quelquefois calculé (incr)
Mise à jour (seule)	**PUT**	**…/product/idRes** **ou** **…/product**	Objet JSON avec **.id**	Objet JSON mis à jour
SaveOr Update	*POST*	*.../product*	*Objet JSON*	*Objet JSON ajouté (auto_incr id) ou modifié*
suppression	**DELETE**	**…/product/idRes**	vide	Statut et message
Autres	**...**	**…/product-action/opXy/....**	...	...

2.1. Statuts HTTP (code d'erreur ou ...)

Catégories de code/statut HTTP :

1xx	Information (rare)
2xx (ex : 200)	Succès
3xx	Redirection
4xx	Erreur du client
5xx (ex : 500)	Erreur du serveur

Principaux codes/statuts en cas de succès ou de redirection:

200 , OK	Requête traitée avec succès. La réponse selon méthode de requête utilisée
201 , Created	Requête traitée avec succès et création d'un document.
204 , No Content	Requête traitée avec succès mais pas d'information à renvoyer.
301 , *Moved Permanently*	Document déplacé de façon permanente
304 , *Not Modified*	Document non modifié depuis la dernière requête

Principaux codes d'erreurs :

400 , Bad Request	La syntaxe de la requête est erronée (ex : invalid argument)
401 , Unauthorized	Une authentification est nécessaire pour accéder à la ressource.
403, Forbidden	authentification effectuée mais manque de droits d'accès (selon rôles, ...)
404 , Not Found	Ressource non trouvée.
409 , Conflict	La requête ne peut être traitée en l'état actuel.
...	*liste complète sur https://fr.wikipedia.org/wiki/Liste_des_codes_HTTP*
500 , Internal Server Error	Erreur interne (vague) du serveur (ex ; bug , exception , ...) .
501, Not Implemented	Fonctionnalité réclamée non supportée par le serveur
503 , Service Unavailable	Service temporairement indisponible ou en maintenance.

2.2. Variantes classiques

Réponses plus ou moins détaillées (simple "http status" ou bien "message json")

Lorsqu'un serveur répond à une requête en mode POST , il peut soit :

*- retourner le "https status" **201/CREATED** et une réponse JSON comportant toute l'entité sauvegardée coté serveur avec souvent l'id (clef primaire) automatiquement généré ou incrémenté*
*{ **"id" : "a345b6788c335d56"** , "name" : "toto" , ... }*

*- se contenter de renvoyer le "http status" **201/CREATED** avec aucun message de réponse mais avec le champ **Location: /type_entite/idxy** comportant au moins l'id de la resource enregistrée au sein de l'entête HTTP de la réponse .*
*L'application cliente pourra alors effectuer un second appel en mode GET avec une fin d'URL en **/type_entite/idxy** si elle souhaite récupérer tous les détails de l'entité sauvegardée .*

- combiner les 2 styles de réponses (champ Location ET réponse JSON)

Lorsqu'un serveur répond à une requête en mode DELETE , il peut soit :

- se contenter de renvoyer le "http status" **204/NO_CONTENT** et aucun message

- retourner le "https status" **200/0K** et une réponse JSON de type
{ "message" : "resource of id ... successfully deleted" }

Lorsqu'un serveur répond à une requête en mode PUT , il peut soit :

- se contenter de renvoyer le "http status" **204/NO_CONTENT** et aucun message

*- retourner le "https status" **200/0K** et une réponse JSON comportant toutes les valeurs de l'entité mise à jour du coté serveur , exemple:*
{ "id" : "a345b6788c335d56" , "name" : "titi" , ... }

*On peut éventuellement envisager que le serveur réponde **par défaut** aux modes PUT et DELETE par un simple **204/NO_CONTENT** et qu'il réponde par **200/OK + un message JSON** si le paramètre http optionnel **?v=true** ou **?verbose=true** est présent en fin de l'URL de la requête .*

Identifiant de la resource à modifier en mode PUT placé en fin d'URL ou bien dans le corps de la requête HTTP, ou bien les deux.

Lorsqu'un serveur reçoit une requête de mise à jour en mode PUT , l'id de l'entity peut soit être précisée en fin d'URL , soit être précisée dans les données json de la partie body et si l'information est renseignée des 2 façons elle ne doit pas être incohérente .

Le serveur peut éventuellement faire l'effort de récupérer l'id de l'une ou des deux façons envisageables et peut renvoyer ***400/BAD_REQUEST*** si l'id de l'entité à mettre à jour n'est pas renseigné ou bien incohérent.

2.3. Safe and idempotent REST API

Une Api "Rest" désigne un ensemble de Web-services liés à un certain domaine fonctionnel (ex : gestion des stocks ou facturation ou ...)

Un appel "HTTP" vers une api-rest est dit "***safe***" s'il n'engendre pas de modifications du coté des ressources du serveur (**"safe" = "readonly"**) .

En mathématique , une fonction est dite "idempotente" si plusieurs appels successifs avec les mêmes paramètres retournent toujours le même résultat.

Au niveau d'une **api-rest** , une **invocation HTTP** (***ex : GET , PUT ou DELETE***) est dite "**idempotente**" si **plusieurs appels successifs avec les mêmes paramètres engendrent un même "état résultat" au niveau du serveur** .

Mais la réponse HTTP peut cependant varier .

Exemple : premier appel à "delete xyz/567" --> return "200/OK" ou "204/NO_CONTENT"

et second appel à "delete xyz/567"--> return 404 / notFound

mais dans les 2 cas , la ressource de type "xyz" et d'id=567 est censée ne plus exister .

Le DELETE est donc généralement considéré comme idempotent .

	safe	idempotent
GET *(et HEAD,OPTIONS)*	y	y
PUT	n	y
DELETE	n	y
POST	n	n

Intérêt de l'impotence comportementale du coté serveur :

Une application cliente doit souvent passer par des intermédiaires pour véhiculer une requête HTTP jusqu'au serveur . Certains mécanismes intermédiaires considèrent "internet / http" comme pas fiable à 100 % et vont quelquefois effectuer plusieurs retransmissions d'une requête si la première tentative échoue . il vaut mieux donc que le serveur se comporte de manière idempotente dans un maximum de cas .

Bien que le vocabulaire "~~idempotence~~" ne soit pas du tout approprié , **il est tout de même conseillé de retourner des réponses HTTP dans un format assez homogène vers le client** pour que celui-ci soit simple à programmer (pas trop de if ... else ...)

Dans tous les cas , bien documenter "comportements & réponses" d'une apit rest .

3. Test de W.S. REST via Postman

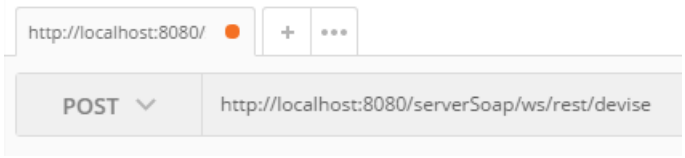
L'application "**postman**" (téléchargeable depuis l'url *https://www.postman.com/downloads/*)

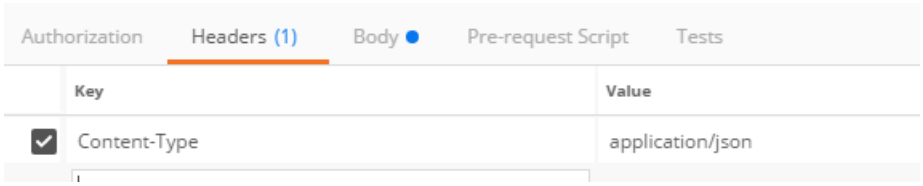
existe depuis longtemps et est souvent considérée comme l'application de référence pour tester les web services "REST" .

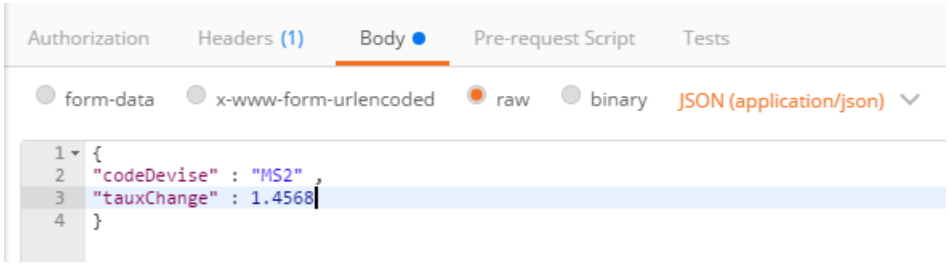
NB1 : après le premier lancement , il n'est pas obligatoire de s'enregistrer (créer un compte) pour utiliser l'application , on peut cliquer sur un lien à peine visible plus bas que la boite de dialogue nous invitant à nous enregistrer et l'on peut d'une manière générale fermer toutes les "popups" et créer un nouvel onglet de requête pour paramétrer et lancer un test.

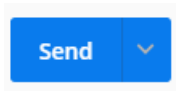
NB2 : A une certaine époque , "postman" pouvait s'utiliser en tant que plugin pour le navigateur "chrome" . Ce plugin est maintenant "deprecated" (plus maintenu) .

3.1. paramétrages "postman" pour une requête en mode "post"









3.2. Exemple de réponses précises reçues et affichées par "postman"

POST , http://localhost:8282/login-api/public/auth , Content-Type : application/json ,

request body : **{ "username": "admin1" , "password" : "pwdadmin1" , "roles" : "admin,user" }**

==> **200 ok**

et responseBody :


```
{
   "username": "admin1",
   "status": true,
   "message": "successful login",
   "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWJqZWN0IjoiYWRtaW4xIiwicm9sZXMiOiJh
ZG1pbix1c2VyIiwiaWF0IjoxNTk2Nzk2MTI0LCJleHAiOjE1OTY4MDMzMjQsImlzcyI6Imh0dH
A6Ly93d3cubXljb21wYW55In0.wBQHJPN20VE7tzrF8vk3Cq9FltiQQOf2RETDMSB19ho",
   "roles": "admin,user"
}
```


Si requête comportant **{ "username": "admin1" , "password" : "abcdef" , "roles" : "admin,user" }**

alors réponse de ce type :


```
{
   "username": "admin1",
   "status": false,
   "message": "login failed (wrong password)",
   "token": null,
   "roles": null
}
```


DELETE , http://localhost:8282/devise-api/private/role_admin/devise/m3875

==> **200 ok**

et responseBody :


```
{
   "action": "devise with code=m3875 was deleted"
}
```


ou bien (suite à un second appel successif) :

==> **404 not found**


```
{
   "errorCode": "404",
   "message": "deleteOne exception with id=m3875"
}
```

4. Test de W.S. REST via curl

curl (*command line url*) est un programme utilitaire (d'origine linux) permettant de déclencher des requêtes HTTP via une simple ligne de commande.

Via certaines options , curl peut effectuer des appels en mode "GET" , "POST" , "DELETE" ou "PUT".

Ceci peut être très pratique pour tester rapidement un web service REST via quelques lignes de commandes placées dans un script réutilisable (.bat, .sh , ...) .

lancer_curl.bat

```
cd /d "%~dp0"
REM instructions qui vont bien
set URL=http://localhost:8081/my-api/info/1
curl  %URL%
pause
```

curl fonctionne en **mode GET par défaut** si pas de -d *(pas de data)*

```
curl  %URL%
REM "verbose" (-v)  très pratique pour connaître les détails de la communication réseau
curl -v %URL%
```

```
curl -o out.json %URL%
```

pour stocker la réponse dans un fichier texte (ici out.json)

curl fonctionne en mode **POST** par défaut avec data (**-d** ...)

curl fonctionne en mode PUT si **-X PUT** et mode DELETE si **-X DELETE**

appel au format par défaut (`application/x-www-form-urlencoded`)

si pas d'option -H "Content-Type: ...." au niveau de la requête
alors par défaut logique champ/paramètre de formulaire en mode POST avec

-d paramName1=valeur1 -d paramName2=valeur2 ...

Exemple :

set URL=clientIdPassword:secret:@localhost:8081/basic-oauth-server/oauth/token

set PWD=d8dfc382-e012-491a-8d03-ca6ad9d81083

curl %URL% -d grant_type=password -d username=user -d password=%PWD%

Requête au format "**application/json"** **:**

NB : en version windows , curl ne gère pas bien les simples quotes et il faut préfixer les " internes par des \

```
curl %LOGIN_URL% -H "Content-Type: application/json"
                        -d "{\"username\":\"member1\" , \"password\": \"pwd1\"}"
```

il vaut mieux donc utiliser un fichier pour les données en entrée :

```
curl %LOGIN_URL% -H "Content-Type: application/json" -d @member1-login-request.json
```

avec

member1-login-request.json

```
{
"username":"member1",
"password":  "pwd1"
}
```

Authentification avec curl :

curl **--user myUsername:myPassword** ... permet une "***BASIC HTTP AUTHENTICATION***"

ou bien

curl **-H "Authorization: Bearer b1094abc.._ou_autre_jeton"** permet une demande d'autorisation en mode "Bearer / au porteur de jeton" (jeton à préalablement récupérer via login ou autre)

5. Api Key

Un web service hébergé par une entreprise et rendu accessible sur internet a un certain coût de fonctionnement (courant électrique , serveurs , ...) .

Pour limiter des abus (ex : appel en boucle) ou bien pour obtenir un paiement en contre partie d'une bonne qualité de service , un web service public est souvent invocable que si l'on renseigne une "api_key" (au niveau de l'URL ou bien au niveau de l'entête la requête HTTP).

Une "**api_key**" est très souvent de type "**uuid/guid**" .

Critères d'une api_key :

- lié à un abonnement (gratuit ou payant) , ex : compte utilisateur / compte d'entreprise
- ne doit idéalement pas être diffusé (à garder secret)
- souvent lié à un compteur d'invocations (limite selon prix d'abonnement)
- doit pouvoir être administré (régénéré si perdu/volé , ...)
  et les modifications doivent pouvoir être immédiatement ou rapidement prises en compte.

Exemple :

Le site **https://fixer.io** héberge un web service REST permettant de récupérer les taux de change (valeurs de *"USD"* , *"GBP"* , *"JPY"* , ... vis à vis de "*EUR*" par défaut).

Début 2018, ce web service était directement invocable sans "api_key" .

Courant 2018, ce web service est maintenant invocable qu'avec une "**api_key**" **liée à un compte utilisateur** *"gratuit" ou bien "payant"* selon le mode d'abonnement (options, fréquence d'invocation, ...).

URL d'appel sans "api_key" : **http://data.fixer.io/api/latest**

Réponse :

```
{
"success":false,
"error":{"code":101,"type":"missing_access_key",
        "info":"You have not supplied an API Access Key. [Required format:
                access_key=YOUR_ACCESS_KEY]"
    }
}
```

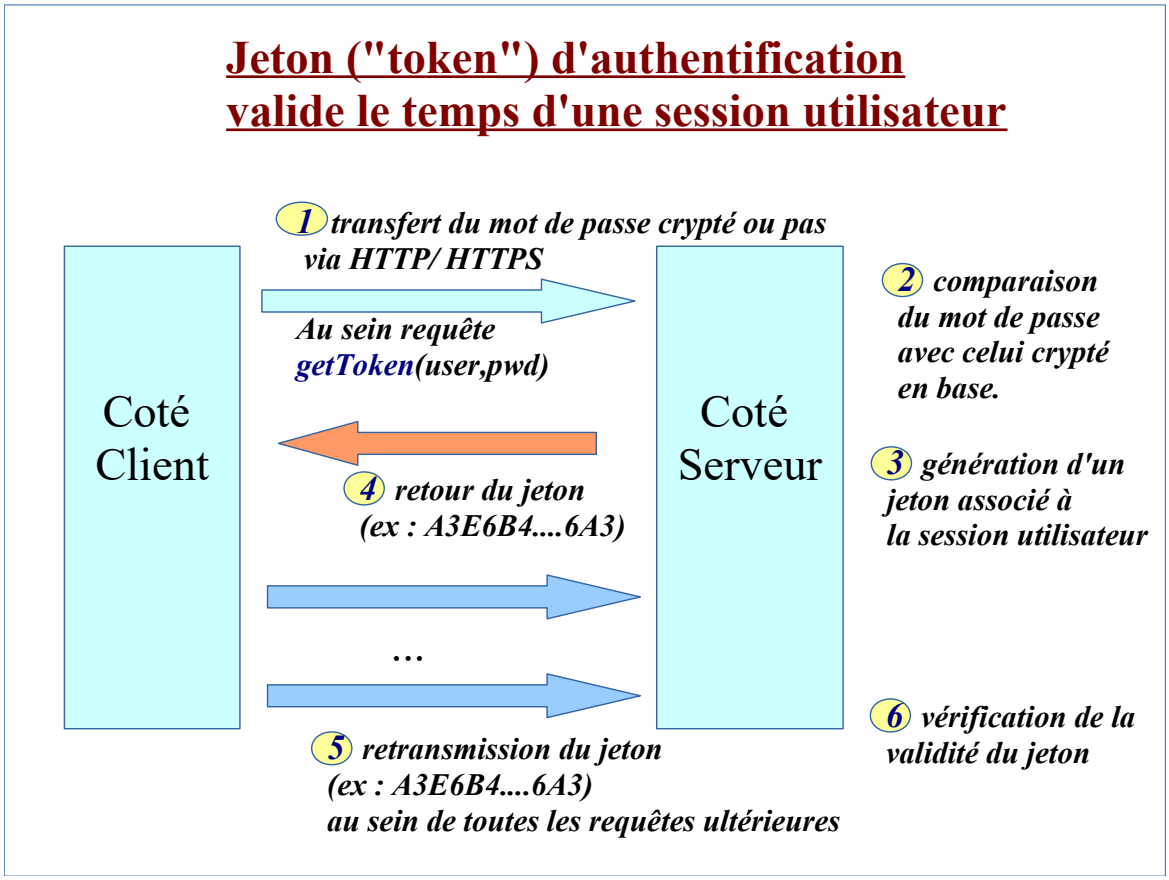
URL d'invocation avec api_key valide :

http://data.fixer.io/api/latest?access_key=*26ca93ee7.....aaa27cab235*

```
{
"success":true, "timestamp":1538984646, "base":"EUR", "date":"2018-10-08",
"rates":
{"AED":4.224369,...,"DKK":7.460075,"DOP":57.311592,"DZD":136.091172,"EGP":20.596249,
"ERN":17.250477,"ETB":31.695652,"EUR":1,"FJD":2.46956,"FKP":0.88584,"GBP":0.879667,.
...,"JPY":130.858498,....,"USD":1.15005,...,"ZWL":370.724343}
}
```


6. Token d'authentification

6.1. Tokens : notions et principes



Plusieurs sortes de jetons/tokens

Il existe plusieurs sortes de jetons (normalisés ou pas).

Dans le cas le plus simple, un **jeton est généré aléatoirement** (ex : **uuid** ou …) et **sa validation consiste essentiellement à vérifier son existence** en tentant de le récupérer quelque part *(en mémoire ou en base)* et éventuellement à vérifier une date et heure d'expiration.

JWT (Json Web Token) est un **format particulier de jeton** qui **comporte 3 parties** (une entête technique , un paquet d'informations en clair (ex : username , email , expiration, …) au format JSON et une signature qui ne peut être vérifiée qu'avec la clef secrète de l'émetteur du jeton.

6.2. Bearer Token (au porteur) / normalisé HTTP

Bearer token (jeton au porteur) et transmission

Le champ ***Authorization:*** normalisé d'une entête d'une requête HTTP peut comporter une valeur de type ***Basic …*** ou bien ***Bearer …***

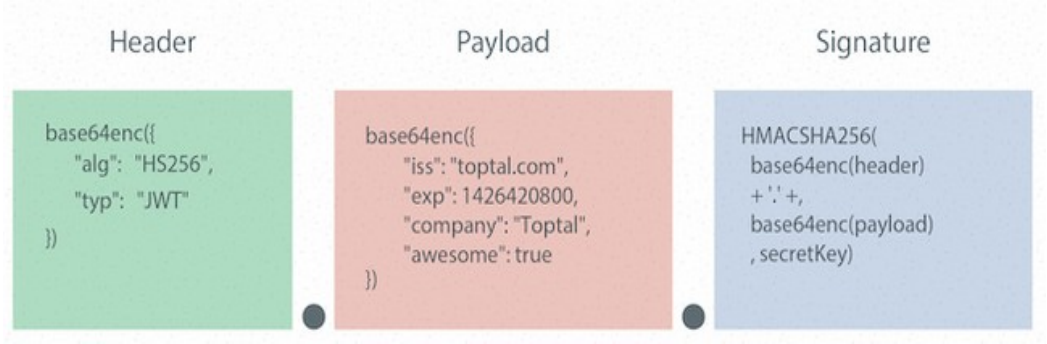
Le terme anglais "**Bearer**" signifiant "au **porteur**" en français indique que la simple possession d'un jeton valide par une application cliente devrait normalement , après transmission HTTP, permettre au serveur d'autoriser le traitement d'une requête (après vérification de l'existence du jeton véhiculé parmi l'ensemble de ceux préalablement générés et pas encore expirés).

NB: Les "bearer token" sont utilisés par le protocole "O2Auth" mais peuvent également être utilisés de façon simple sans "O2Auth" dans le cadre d'une authentification "sans tierce partie" pour API REST.

NB2 : un "bearer token" peut éventuellement être au format "JWT" mais ne l'est pas toujours (voir rarement) en fonction du contexte.

6.3. JWT (Json Web Token)

Structure jeton "JWT / Json Web Token"



Exemple:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJ0b3B0YWwuY29tIiwiZXhwIjoxNDI2NDIwODAwLCJodHRwOi8vdG9wdGFsLmNvbS9qd3RfY2xhaW1zL2lzX2FkbWluIjp0cnVlLCJjb21wYW55IjoiVG9wdGFsIiwiYXdlc29tZSI6dHJ1ZX0.yRQYnWzskCZUxPwaQupWkiUzKELZ49eM7oWxAQK_ZXw

NB : "iss" signifie "issuer" (émetteur) , "iat" : issue at time
"exp" correspond à "date/heure expiration" . Le reste du "payload" est libre (au cas par cas) (ex : "company" et/ou "email" , ...)