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 , ...) .

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. 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)

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="/rest/devise" , headers="Accept=application/json")
public class DeviseJsonRestCtrl {

@Autowired //ou @Inject
private GestionDevises gestionDevises; //internal business service or DAO

//RECHERCHE UNIQUE selon RESOURCE-ID:
//URL de déclenchement: .../webappXy/rest/devise/EUR
@RequestMapping(value="/{codeDevise}" , method=RequestMethod.GET)
 public Devise getDeviseByName(@PathVariable("codeDevise") String  codeDevise) {
        return gestionDevises.getDeviseByPk(codeDevise);
}

//RECHERCHE MULTIPLE :
//URL de déclenchement: webappXy/rest/devise
                        //ou webappXy/rest/devise?name=euro
@RequestMapping(value="" , method=RequestMethod.GET)
public List<Devise> getDevisesByCriteria(@RequestParam(value="name",required=false)
                                        String nomMonnaie) {
        if(nomMonnaie==null)
                return gestionDevises.getListeDevises();
        else{
                List<Devise> listeDev= new ArrayList<Devise>();
                Devise devise = gestionDevises.getDeviseByName(nomMonnaie);
                if(devise!=null) listeDev.add(devise);                    return listeDev;
           }
}
}
```

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/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 .

Prise en charge des modes "PUT" , "POST" , "DELETE" :

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="/rest/devises" , headers="Accept=application/json")
public class DeviseJsonRestCtrl {
...
 @RequestMapping(value="" , method=RequestMethod.PUT )
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);
```

```
                gestionDevises.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="/rest/devise" , headers="Accept=application/json")
public class DeviseJsonRestCtrl {
...
 @RequestMapping(value="" , method=RequestMethod.PUT )
Devise updateDevise(@RequestBody  Devise devise) {
        System.out.println("devise to update:" + devise);
        gestionDevises.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) est assez souvent 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:

```
@RequestMapping(value="/{codeDev}" , method=RequestMethod.GET)
 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/rest/devise/EUR
@RequestMapping(value="/{codeDev}",method=RequestMethod.DELETE)
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 ...ClasseExceptionPréfixéePar_@ResponseStatus plus simple et plus efficace (vu dans le paragraphe ci-après)**

Eventuelles variations (simplifications):

@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**)

2.1. 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) .

2.2. @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"
}
```

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;
}
```

2.3. Exemples d'appels en js/ajax

js/ajax-util.js

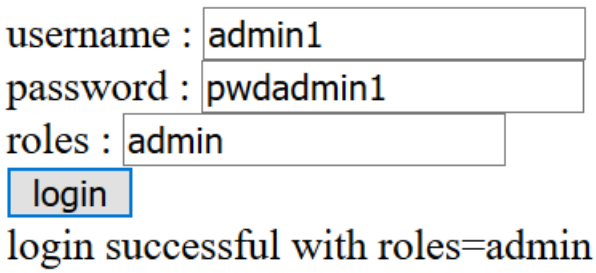
```
//fonction utilitaire pour preparer xhr en vu d'effectuer juste apres un appel ajax en mode Get ou post ou ...
function initXhrWithCallback(callback,errCallback){
	var xhr = new XMLHttpRequest();
	xhr.onreadystatechange = function() {
		if (xhr.readyState == 4){
			if (xhr.status == 200 || xhr.status == 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,"./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,"./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,"./rest/devise-api/private/role_admin/devise" ,
				JSON.stringify(nouvelleDevise));
	});//end of btnGetDevises.addEventListener/"click"
} //end of window.onload
```

2.4. 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);
```

```
        }
    }
}
```

2.5. 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();
}
*/
```

2.6. Test d'un "RestController" via @WebMvcTest et MockMvc

Pour tester le comportement d'un composant "RestController" de Spring-Mvc sans avoir à démarrer un serveur complet tel que Tomcat (ou un équivalent) , on peut utiliser la classe **MockMvc** et l'annotation **@WebMvcTest** 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 .

```
@RunWith(SpringRunner.class)
@WebMvcTest(DeviseJsonRestCtrl.class)
public class DeviseJsonRestCtrlIntegrationTest {

    @Autowired
    private MockMvc mvc;

    @Test
    public void testXyz(){
    mvc.perform(get("/rest/devise?name=euro")
                .contentType(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$", hasSize(1) ))
                .andExpect(jsonPath("$[0].name", is("euro") ));
    }
}
```

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