Explications liées au programme DemoTest6

Rest WebService

@Data **@AllArgsConstructor** **@NoArgsConstructor** **public class Info {** **private String message;** **private Date date;** **}**	Classe qui sera utilisée pour passer des objets au Web service ou en provenance du Web Service

@Data **public class ListInfo implements Serializable {** **private List<Info> list;** **public ListInfo()** **{list=new ArrayList<>();}** **}**	Classe qui encapsule une ArrayList<Info> pour permettre d'échanger une ArrayList<Info> avec le Web Service Impossible, coté client, de récupérer dans du code les différents objets Info de la liste ! → on doit encapsuler cette liste dans une classe

<dependency> **<groupId>com.fasterxml.jackson.core</groupId>** **<artifactId>jackson-databind</artifactId>** **</dependency>**	Dès qu'on a cette dépendance → Spring Boot utilise d'office Jackson pour produire et consommer du Json

@RestController **@CrossOrigin(origins="*")** **@RequestMapping("/api")** **public class ServiceRest {**	Un Web Rest Service est annoté par **@RestController** **@CrossOrigin(origins="*")** indique que le client peut provenir de n'importe quel domaine internet **@RequestMapping("/api")** → Url relatif du Web Service
@GetMapping("/hello") public String hello(){return "Hello les MA1 Info";}	→ **/api/hello** appelle cette méthode qui renvoie une String
@GetMapping("/hello2") public String hello2(**@RequestParam(defaultValue="Inconnu")** String **name**) {return "Hello2 "+name;}	→ Utilisation d'un paramètre de requête → **/api/hello2 ?name=Severs**
@GetMapping("/hello3/{name}") public String hello3**(@PathVariable String name**) {return "Hello3 "+name;}	→ **/api/helo3/Severs** // Principe de la réécriture d'URL --> le paramètre est concaténé à l'URL

```
@PostMapping("/hello4")
  public String hello4(String name)
  { return "Hello4 "+name; }
```

➔ Ici on reçoit un paramètre posté par un formulaire (page index.html par ex.)

```
<body>
 <form  action="/api/hello4"
          method="post">
  <input type="text" name="name">
  <button type="submit">
        Envoyer
   </button>
 </form>
 </body>
```

```
@GetMapping("/**")
  public String helloError()
  {return "URL invalide !!!!";}
```

➔ Cette méthode sera appelée en cas d'URL invalide

```
@GetMapping("/hello5")
 public ResponseEntity<String>
          hello5( @RequestParam String name)
{if(name.equals("Severs")) return
 new  ResponseEntity<>(null, HttpStatus.FORBIDDEN);
  else return new
   ResponseEntity<>("Hello5"+name, HttpStatus.OK); }
```

On peut appliquer des conditions sur la valeur des paramètres reçus. Ici le paramètre Severs génère une erreur

ResponseEntity est un objet qui encapsule la réponse de type String ici et un statut

```
@GetMapping("/hello6")
public ResponseEntity<String>
        hello6(@RequestParam(required=true,
                defaultValue="-1") String name)

 {if(name.equals("-1")) return new
 ResponseEntity<>(null, HttpStatus.BAD_REQUEST);
  else return new
  ResponseEntity<>("Hello6 "+name, HttpStatus.OK); }
```

➔ Ici, on prévoit en cas d'oubli du paramètre dans la requête une valeur par défaut (-1) qui génère un statut BAD_REQUEST

```
@GetMapping("/clientjson")
public String clientJson()
 { RestTemplate rst=new RestTemplate();
    Info
info=rst.getForObject("http://localhost:8080/apijson",
Info.class);
// Trt qcq sur l'objet .... par ex stockage dans la BD !

     return info.toString();
   }

}
```

// On crée ici un client Rest d'un WebService accessible via un Get avec l'URL **http://localhost:8080/apijson** et qui renvoie un objet Info au format Json

On utilise un objet **RestTemplate** sur lequel on applique la méthode **getForObject**

WebService qui produit des objets envoyés au format JSON

@RestController **@CrossOrigin(origins="*")** **@RequestMapping("/apijson")** public class JsonServiceRest {	
@GetMapping public Info getInfo(){return info();}	On utilise deux méthodes qui renvoie le même objet Info, l'une via un GET, l'autre via un POST.
@PostMapping public Info postInfo(){return info();}	A partir de POSTMAN ou CURL on peut utiliser les deux. A partir du Client RestTemplate, on doit appliquer getForObject → GET
private Info info(){ return new Info("Evénement dangereux type 4", new Date());}	

@GetMapping("/listinfo")
public ListInfo getListInfo()
{ // On devrait récupérer une liste d'Info dans la BD
List<Info> list=new ArrayList<>();
list.add(new Info("Item 1", new Date()));
list.add(new Info("Item 2", new Date()));
list.add(new Info("Item 3", new Date()));
ListInfo listInfo=new ListInfo();
listInfo.setList(list);
return listInfo;
}
 	
@GetMapping("/tabinfo")
public Info[] getTabInfo()
{ // On devrait récupérer une liste d'Info dans la BD
Info[] tab=new Info[3];
tab[0]=new Info("Item 1", new Date());
tab[1]=new Info("Item 2", new Date());
tab[2]=new Info("Item 3", new Date());
return tab;
}
	Cette méthode renvoie un objet de la classe ListeInfo qui encapsule une ArrayList<Info>
	
	Permet de contourner le problème que RestTemplate ne sait pas typer des objets Info lorsqu'elle reçoit une List>Info>
	
	Une autre solution est de renvoyer un tableau Info[]

@GetMapping("/listinfo2")
// public List<Info> getListInfo() --> problème coté
// client pour récupérer les différents objets Info
public List<Info> getListInfo2()
{ List<Info> list=new ArrayList<>();
list.add(new Info("Item 1", new Date()));
list.add(new Info("Item 2", new Date()));
list.add(new Info("Item 3", new Date()));
return list;
}
	Ici le Web Service renvoie bien une List<Info> → ok cela fonctionne bien par exemple si on teste le web Service avec Postman ou curl
	
	Mais pas avec **RestTemplate**

@PostMapping("/postinfo")
public void postInfo(**@RequestBody Info inf**)
{ // En principe le web service va stocker dans la BD
// l'objet ....
System.out.println(inf.toString()+ "objet posté par le client");
 	
}
 	
@PostMapping("/postlistinfo")
public void postListInfo(**@RequestBody ListInfo listInf**)
{
// En principe le web service va stocker dans la BD
// les différents objets de la liste
for(Info inf:listInf.getList())
System.out.println(inf.toString()+
"objet posté par le client dans une liste");
}
}
	Ce Web Service reçoit un objet Info posté par un client (requête POST). La méthode reçoit comme paramètre un objet Info qui provient du corps de la requête (@RequestBody)
	
	Ce Web Service reçoit un objet ListInfo posté par un client (requête POST). La méthode reçoit comme paramètre un objet Info qui provient du corps de la requête (@RequestBody)
	
	On extrait les différents objets Info encapsulés dans ListInfo