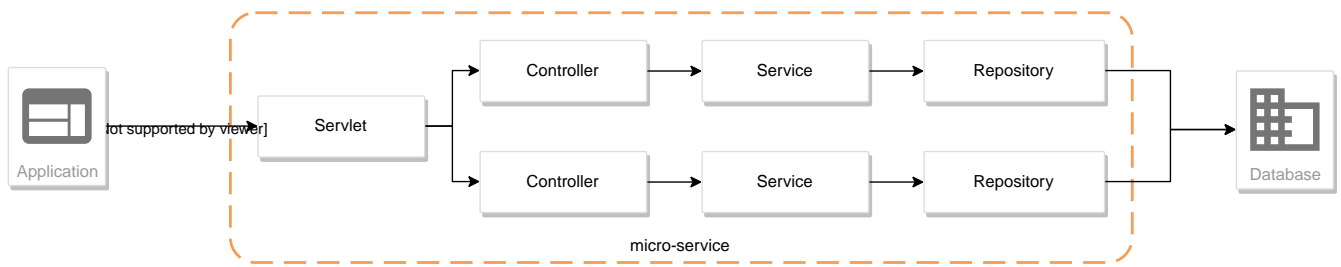


ALOM - TP 2 - Handcraft

Table of Contents

1. Présentation et objectifs	2
2. La première Servlet et la structure projet	3
2.1. Initialisation du projet	3
2.1.1. Création de l'arborescence projet	3
2.2. Ecriture de la première servlet	3
2.3. Installation de Tomcat	5
2.3.1. Configuration pour IntelliJ IDEA	5
2.4. Démarrer notre première Servlet	8
3. Passer votre servlet en mode "annotations" <code>javax.servlet-api 3.0</code>	8
3.1. Le code	8
3.2. Le packaging	9
4. La servlet dynamique	12
4.1. Les annotations	12
4.2. Notre premier controller	13
4.3. L'analyse dynamique du code	13
4.3.1. JUnit et Maven	13
4.3.2. Le test unitaire	14
4.3.3. La DispatcherServlet (code à trous)	17
4.4. Le routage des requêtes (code à trous)	19
4.4.1. Les tests unitaires du routage	19
5. Le micro-service PokemonType	21
5.1. La structure	22
5.2. La classe PokemonType	22
5.3. Le PokemonTypeRepository	25
5.3.1. jackson-databind	25
5.3.2. Le jeu de données du repository	26
5.3.3. Les tests unitaires du repository	27
5.3.4. Le PokemonTypeRepository	28
5.4. Le PokemonTypeController	29
5.4.1. Les tests unitaires du PokemonTypeController	29
5.4.2. Le PokemonTypeController (code à trous)	31
5.5. Modifications de la DispatcherServlet	32

1. Présentation et objectifs

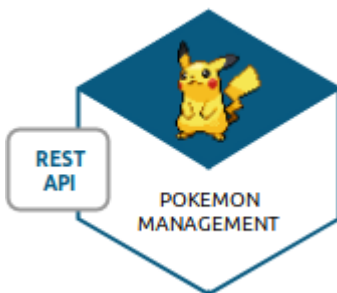


Le but est de créer une architecture "à la microservice".

Dans cette architecture, chaque composant a son rôle précis :

- la servlet reçoit les requêtes HTTP, et les envoie au bon controller (rôle de point d'entrée de l'application)
- le contrôleur implémente une méthode Java par route HTTP, récupère les paramètres, et appelle le service (rôle de routage)
- le service implémente le métier de notre micro-service
- le repository représente les accès aux données (avec potentiellement une base de données)

Et pour s'amuser un peu, nous allons réaliser un micro-service qui nous renvoie des données sur les Pokemons !



On retrouve en général le même découpage dans les micro-services NodeJS avec express :



- La déclaration de l'application (express)
- La déclaration des routeurs (express.Router)
- L'implémentation du code métier et les accès à une base de données

Nous allons donc développer un micro-service, qui exposera un canal de communication REST/JSON.

Pour ce faire, nous allons :

- Créer des annotations Java pour représenter nos objects
- Créer une servlet, qui se configurera dynamiquement pour router les requêtes au bon contrôleur

- Implémenter un petit service

2. La première Servlet et la structure projet

Pour commencer, créons une première servlet.

2.1. Initialisation du projet

2.1.1. Création de l'arborescence projet

Initialisez un repository GitLab avec ce lien : <https://gitlab-classrooms.cleverapps.io/assignments/a480d724-79a4-4f8b-8773-cc1678542477/accept>

Ajoutez-y les répertoires de sources java et de test :

```
$ mkdir -p src/main/java
$ mkdir -p src/test/java
```

Initialiser un fichier pom.xml à la racine du projet

```
1 <project>
2   <modelVersion>4.0.0</modelVersion>
3   <groupId>com.miage.alom.tp</groupId>
4   <artifactId>handcrafting</artifactId>
5   <version>0.1.0</version>
6   <packaging>war</packaging> ①
7
8   <properties>
9     <maven.compiler.source>21</maven.compiler.source> ②
10    <maven.compiler.target>21</maven.compiler.target> ③
11  </properties>
12
13  <dependencies>
14  </dependencies>
15
16 </project>
```

① On va fabriquer un war

② On indique à maven quelle version de Java utiliser pour les sources !

③ On indique à maven quelle version de JVM on cible !

2.2. Ecriture de la première servlet

Pour écrire notre première servlet, nous avons besoin de la dépendance `jakarta.servlet-api`. Cette dépendance aura le scope `provided` puisque:

- nous en avons besoin à la compilation
- à l'exécution, c'est **Tomcat** qui portera la librairie

Ajouter la dépendance suivante dans votre `pom.xml`

```
1 <dependency>
2   <groupId>jakarta.servlet</groupId>
3   <artifactId>jakarta.servlet-api</artifactId>
4   <version>6.0.0</version>
5   <scope>provided</scope> ①
6 </dependency>
```

① On précise bien un scope *provided* à Maven

Écrire une première servlet :

`src/main/java/FirstServlet.java`

```
1 import jakarta.servlet.ServletConfig;
2 import jakarta.servlet.ServletException;
3 import jakarta.servlet.http.HttpServlet;
4 import jakarta.servlet.http.HttpServletRequest;
5 import jakarta.servlet.http.HttpServletResponse;
6 import java.io.IOException;
7
8 public class FirstServlet extends HttpServlet {
9
10     @Override
11     protected void doGet(HttpServletRequest req, HttpServletResponse resp)
12         throws ServletException, IOException {
13         var writer = resp.getWriter();
14         writer.println("Hello !"); ①
15     }
16
17     @Override
18     public void init(ServletConfig config) throws ServletException {
19         super.init(config);
20
21         System.out.println("Initialisation de la servlet"); ②
22     }
23 }
```

① On dit bonjour !

② On affiche un log au démarrage

Écrire un fichier `web.xml` pour déclarer la servlet :

`src/main/webapp/WEB-INF/web.xml`

```
1 <?xml version="1.0" encoding="UTF-8"?>
```

```

2
3 <web-app>
4
5     <display-name>handcraft</display-name> ①
6
7     <servlet>
8         <servlet-name>dispatcherServlet</servlet-name> ②
9         <servlet-class>FirstServlet</servlet-class>
10        <load-on-startup>1</load-on-startup> ④
11    </servlet>
12
13    <servlet-mapping>
14        <servlet-name>dispatcherServlet</servlet-name>
15        <url-pattern>/*</url-pattern> ③
16    </servlet-mapping>
17
18 </web-app>

```

① Notre application

② Notre servlet

③ On écoute l'ensemble des URLs !

④ *load-on-startup* permet de préciser qu'on souhaite démarrer la servlet immédiatement (sans attendre la première requête)

2.3. Installation de Tomcat

Nous avons besoin de Tomcat pour exécuter notre Servlet !

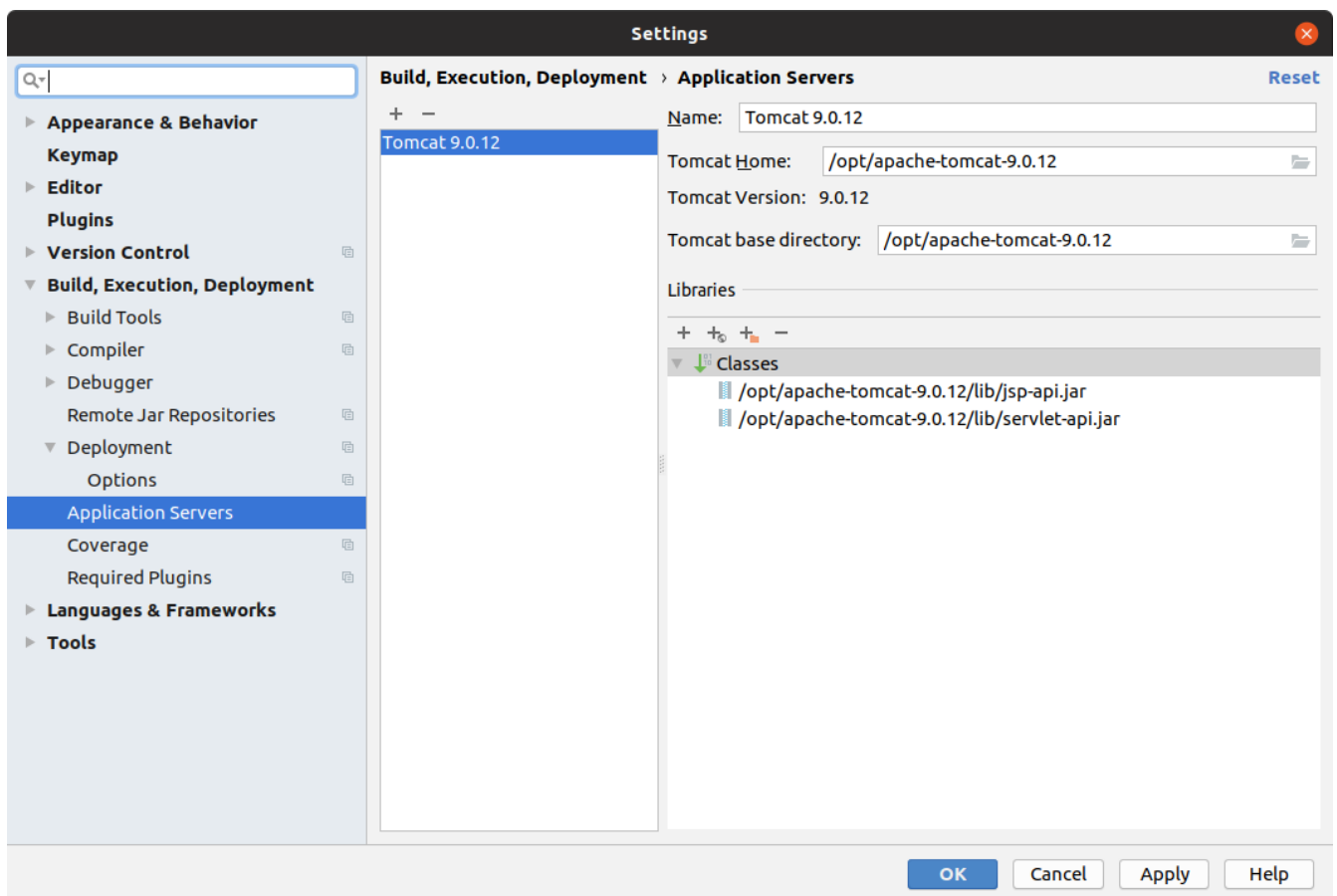
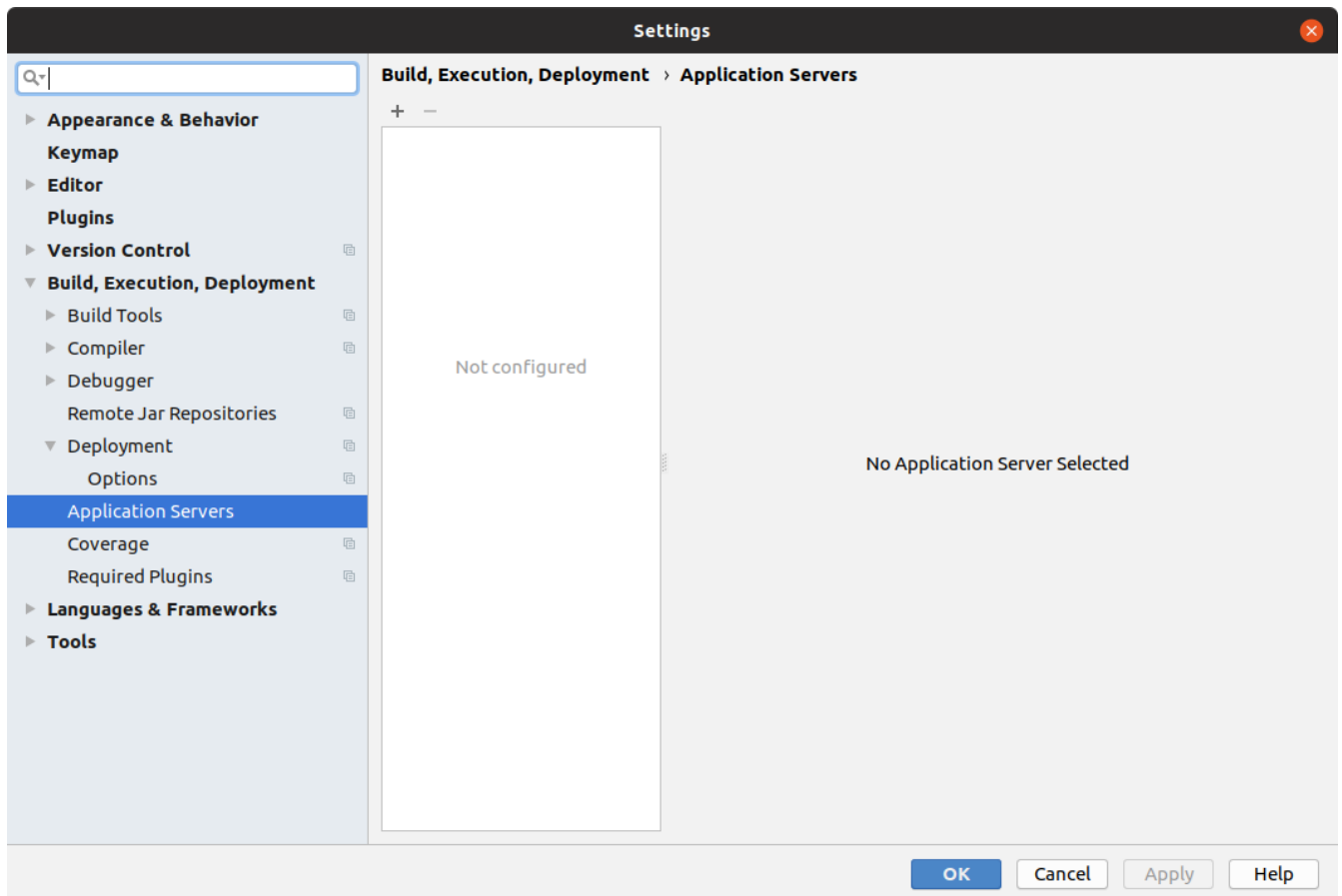
Télécharger tomcat depuis la page officielle : <https://tomcat.apache.org/download-10.cgi>



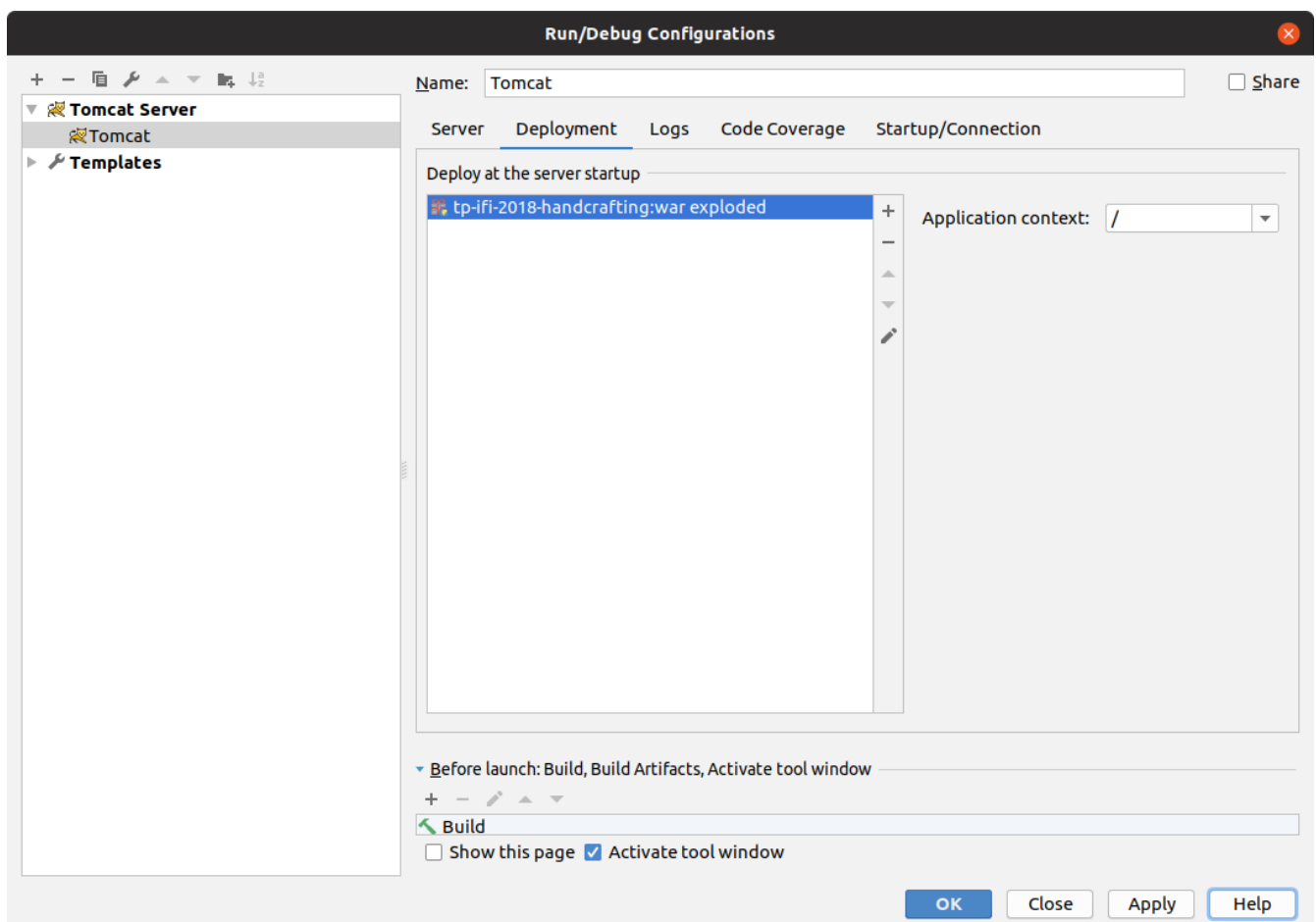
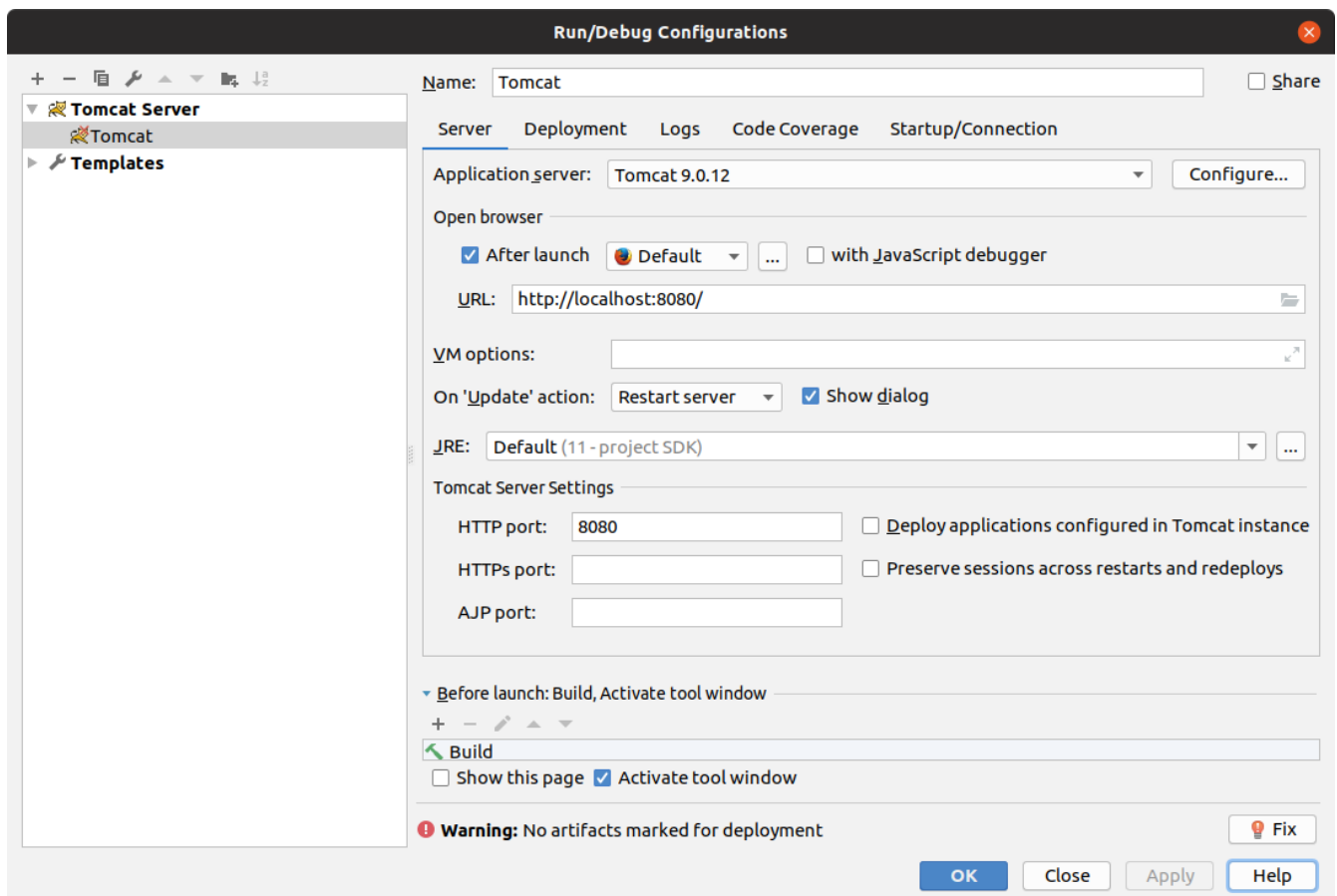
Prenez bien la 'Binary Distribution', sous la section 'Core'. Si vous prenez la source vous devrez compiler Tomcat vous-même ! Sous Linux, privilégiez le format `.tar.gz`, qui conserve les bons droits sur les fichiers.

2.3.1. Configuration pour IntelliJ IDEA

Ajouter le serveur Tomcat à IntelliJ



Créer une configuration d'exécution utilisant le Tomcat



2.4. Démarrer notre première Servlet

Démarrez votre serveur Tomcat, avec votre servlet, et allez constater le résultat !



Votre application est disponible à l'URL <http://localhost:8080>

3. Passer votre servlet en mode "annotations" **servlet-api 3.0**

3.1. Le code

Depuis la version 3.0 de **servlet-api**, les servlets supportent les annotations Java.

Plus besoin de **web.xml**!

Supprimer le fichier **web.xml**, et le répertoire **src/main/webapp**.

Modifier la servlet pour ajouter une annotation java :

src/main/java/FirstServlet.java

```
1 @WebServlet(urlPatterns = "/*", ① ②
2   loadOnStartup = 1) ③
3 public class FirstServlet extends HttpServlet {
4
5   @Override
6   protected void doGet(HttpServletRequest req, HttpServletResponse resp)
7     throws ServletException, IOException {
8     PrintWriter writer = resp.getWriter();
9     writer.println("Hello !");
10  }
11
12  @Override
13  public void init(ServletConfig config) throws ServletException {
14    super.init(config);
15
16    System.out.println("Initialisation de la servlet"); ②
17  }
18 }
```

- ① On déclare la servlet avec une annotation java !
- ② On déclare les URL d'écoute
- ③ et on déclare souhaiter démarrer la servlet sans attendre de première requête

3.2. Le packaging

Par défaut, Maven ne connaît pas les servlets 3.0. Il s'attend donc à trouver un fichier `web.xml` dans le répertoire `src/main/webapp/WEB-INF`.

Si on lance un `mvn package` après avoir supprimé le `web.xml` et le répertoire `webapp`, on obtient l'erreur suivante :

mvn package

```
$> mvn clean package
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.miage.alom.tp.handcrafting >-----
[INFO] Building handcrafting 0.1.0
[INFO] -----[ war ]-----
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ handcrafting ---
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ handcrafting ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e.
build is platform dependent!
[INFO] skip non existing resourceDirectory /home/jwittouck/workspaces/alom/alom-2020-
2021/tp/02-handcrafting/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ handcrafting ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is
platform dependent!
[INFO] Compiling 1 source file to /home/jwittouck/workspaces/alom/alom-2020-
2021/tp/02-handcrafting/target/classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @
handcrafting ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e.
build is platform dependent!
[INFO] skip non existing resourceDirectory /home/jwittouck/workspaces/alom/alom-2020-
2021/tp/02-handcrafting/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ handcrafting
---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ handcrafting ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-war-plugin:2.2:war (default-war) @ handcrafting ---
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by com.thoughtworks.xstream.core.util.Fields
(file:/home/jwittouck/.m2/repository/com/thoughtworks/xstream/xstream/1.3.1/xstream-
1.3.1.jar) to field java.util.Properties.defaults
```

```

WARNING: Please consider reporting this to the maintainers of
com.thoughtworks.xstream.core.util.Fields
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective
access operations
WARNING: All illegal access operations will be denied in a future release
[INFO] Packaging webapp
[INFO] Assembling webapp [handcrafting] in [/home/jwittouck/workspaces/alom/alom-2020-
2021/tp/02-handcrafting/target/handcrafting-0.1.0]
[INFO] Processing war project
[INFO] Webapp assembled in [23 msecs]
[INFO] Building war: /home/jwittouck/workspaces/alom/alom-2020-2021/tp/02-
handcrafting/target/handcrafting-0.1.0.war
[INFO] -----
[INFO] BUILD FAILURE
[INFO] -----
[INFO] Total time: 1.635 s
[INFO] Finished at: 2019-01-11T14:55:59+01:00
[INFO] -----
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-war-plugin:2.2:war
(default-war) on project handcrafting: Error assembling WAR: webxml attribute is
required (or pre-existing WEB-INF/web.xml if executing in update mode) -> [Help 1] ①
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the
following articles:
[ERROR] [Help 1]
http://cwiki.apache.org/confluence/display/MAVEN/MojoExecutionException

```

① Maven n'est pas content, et veut un fichier `web.xml` !

Pour corriger ce comportement, il faut utiliser une version récente du plugin maven `war`. Pour ce faire, ajouter dans votre `pom.xml` le bloc suivant (en dessous de votre bloc `dependencies`)

pom.xml

```

1 <build>
2   <pluginManagement>
3     <plugins>
4       <plugin>
5         <artifactId>maven-war-plugin</artifactId>
6         <version>3.4.0</version> ①
7       </plugin>
8     </plugins>
9   </pluginManagement>
10 </build>

```

① La version 3.4.0 du maven-war-plugin ne nécessite pas de fichier `web.xml` par défaut, comme précisé dans la [documentation](#)

On relance un `mvn package` pour valider la configuration

mvn package

```
$> mvn clean package
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.miage.alom.tp:w01-servlet >-----
[INFO] Building w01-servlet 0.1.0
[INFO] -----[ war ]-----
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ w01-servlet ---
[INFO] Deleting /home/jwittouck/workspaces/univ-lille/alom-
2024/exercices/corrections/w01-servlet/target
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ w01-servlet ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e.
build is platform dependent!
[INFO] skip non existing resourceDirectory /home/jwittouck/workspaces/univ-lille/alom-
2024/exercices/corrections/w01-servlet/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ w01-servlet ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is
platform dependent!
[INFO] Compiling 1 source file to /home/jwittouck/workspaces/univ-lille/alom-
2024/exercices/corrections/w01-servlet/target/classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ w01-
servlet ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e.
build is platform dependent!
[INFO] skip non existing resourceDirectory /home/jwittouck/workspaces/univ-lille/alom-
2024/exercices/corrections/w01-servlet/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ w01-servlet
---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ w01-servlet ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-war-plugin:3.4.0:war (default-war) @ w01-servlet ---
[INFO] Packaging webapp
[INFO] Assembling webapp [w01-servlet] in [/home/jwittouck/workspaces/univ-lille/alom-
2024/exercices/corrections/w01-servlet/target/w01-servlet-0.1.0]
[INFO] Processing war project
[INFO] Building war: /home/jwittouck/workspaces/univ-lille/alom-
2024/exercices/corrections/w01-servlet/target/w01-servlet-0.1.0.war
[INFO] -----
[INFO] BUILD SUCCESS ①
```

```
[INFO] -----  
[INFO] Total time: 0.915 s  
[INFO] Finished at: 2024-09-15T14:55:58+02:00  
[INFO] -----
```

① Maven est content !



Validez que votre servlet fonctionne toujours en la démarrant et en allant voir <http://localhost:8080>

4. La servlet dynamique

4.1. Les annotations

Nous allons utiliser des annotations Java customisées pour créer notre couche de routage. Ces annotations seront analysées par la servlet, avec l'aide des api `java.lang.reflect`, afin de configurer le routage des requêtes HTTP vers le bon controller.

Pour la couche Controller, nous allons créer 2 annotations :

- `@ServletController` : afin de marquer une classe comme étant un controller dans notre architecture
- `@ServletRequestMapping` : afin de marquer une méthode de controller comme devant recevoir des requêtes HTTP

Créer les annotations suivantes dans votre projet :



Positionnez votre code dans un package Java ! Par exemple dans `com.miage.alom.servlet`.

L'annotation @ServletController

```
1 @Retention(RetentionPolicy.RUNTIME) ①  
2 public @interface ServletController {  
3 }
```

① On met une rétention au *runtime*, puisque nous allons utiliser l'annotation à l'exécution

L'annotation ServletRequestMapping

```
1 @Retention(RetentionPolicy.RUNTIME) ①  
2 public @interface ServletRequestMapping {  
3     // uri à écouter  
4     String uri(); ②  
5 }
```

① On a encore une rétention au *runtime*

- ② Notre annotation utilise un paramètre `uri`, permettant de déclarer quelle URI sera écoutée (comme ce qu'on peut faire avec une servlet)

4.2. Notre premier controller

Un controller simple qui dit bonjour

```
1 @ServletController ①
2 public class HelloController {
3
4     @RequestMapping(uri="/hello") ②
5     public String sayHello(){
6         return "Hello World !";
7     }
8
9     @RequestMapping(uri="/bye")
10    public String sayGoodBye(){
11        return "Goodbye !";
12    }
13
14    @RequestMapping(uri="/bom")
15    public String explode(){
16        throw new RuntimeException("Explosion !"); ③
17    }
18
19 }
```

- ① Nous utilisons ici notre annotation
- ② La méthode `sayHello` écoute à l'URI `/hello` et renvoie une chaîne de caractères
- ③ La méthode `explode` lève une exception !

4.3. L'analyse dynamique du code

Notre servlet, que l'on nommera `DispatcherServlet` va analyser le code de notre controller, pour être capable de router les requêtes HTTP, et récupérer les résultats

Supprimez votre servlet précédente, elle ne nous sera plus utile pour la suite.

Pour réaliser notre servlet, nous allons travailler en TDD (test-driven-development).

J'ai implémenté pour vous les tests, il ne reste plus qu'à les faire passer !

4.3.1. JUnit et Maven

Pour utiliser les tests unitaires, il faut rajouter JUnit en dépendance maven.

Ajoutez les dépendances suivant dans votre pom.xml

pom.xml

```
1 <dependency>
2   <groupId>org.junit.jupiter</groupId>
3   <artifactId>junit-jupiter-api</artifactId> ①
4   <version>5.10.0</version>
5   <scope>test</scope>
6 </dependency>
7 <dependency>
8   <groupId>org.junit.jupiter</groupId>
9   <artifactId>junit-jupiter-engine</artifactId> ②
10  <version>5.10.0</version>
11  <scope>test</scope>
12 </dependency>
13 <dependency>
14   <groupId>org.mockito</groupId>
15   <artifactId>mockito-core</artifactId>
16   <version>5.5.0</version>
17   <scope>test</scope>
18 </dependency>
```

① L'API de JUnit 5

② Le moteur d'exécution

Il vous faut également surcharger la version du `maven-surefire-plugin` (qui est le plugin maven qui implémente la phase d'exécution des tests).

pom.xml

```
1 <pluginManagement>
2   <plugins>
3     <plugin>
4       <artifactId>maven-war-plugin</artifactId>
5       <version>3.4.0</version>
6     </plugin>
7     <plugin>
8       <artifactId>maven-surefire-plugin</artifactId>
9       <version>3.1.2</version> ①
10    </plugin>
11  </plugins>
12 </pluginManagement>
```

① On a besoin de la version 2.22.0 minimum pour JUnit 5 comme indiqué [dans la documentation junit](#)

4.3.2. Le test unitaire

Implémentez le test unitaire suivant :

```

1 package com.miage.alom.servlet;
2
3 import com.miage.alom.controller.HelloController;
4 import org.junit.jupiter.api.Test;
5
6 import java.util.Map;
7
8 import static org.junit.jupiter.api.Assertions.*;
9
10 class DispatcherServletTest { ①
11
12     @Test ②
13     void
14     registerController_throwsIllegalArgumentException_forNonControllerClasses() {
15
16         var servlet = new DispatcherServlet();
17
18         assertThrows(IllegalArgumentException.class,
19             () -> servlet.registerController(String.class));
20         assertThrows(IllegalArgumentException.class,
21             () -> servlet.registerController(SomeEmptyClass.class));
22     }
23
24     @Test
25     void registerController_doesNotRegisters_nonAnnotatedMethods() {
26         var servlet = new DispatcherServlet();
27
28         servlet.registerController(SomeControllerClassWithAMethod.class);
29
30         assertTrue(servlet.getMappings().isEmpty());
31     }
32
33     @Test
34     void registerController_doesNotRegisters_voidReturningMethods() {
35         var servlet = new DispatcherServlet();
36
37         servlet.registerController(SomeControllerClassWithAVoidMethod.class);
38
39         assertTrue(servlet.getMappings().isEmpty());
40     }
41
42     @Test ④
43     void registerController_shouldRegisterCorrectyMethods(){
44         var servlet = new DispatcherServlet();
45
46         servlet.registerController(SomeControllerClass.class);
47         servlet.registerController(SomeOtherControllerClass.class);
48
49         assertEquals("someGoodMethod",
50             servlet.getMappingForUri("/test").getName());

```

```

49     assertEquals("someOtherNiceMethod",
50                 servlet.getMappingForUri("/otherTest").getName());
51 }
52
53 @Test
54 void registerHelloController_shouldWorkCorrectly(){
55     var servlet = new DispatcherServlet();
56     servlet.registerController(HelloController.class);
57
58     assertEquals("sayHello", servlet.getMappingForUri("/hello").getName());
59     assertEquals("sayGoodBye", servlet.getMappingForUri("/bye").getName());
60     assertEquals("explode", servlet.getMappingForUri("/boum").getName());
61 }
62 }
63
64
65 class SomeEmptyClass{}
66
67 ③
68 @com.miage.alom.servlet.ServletController
69 class SomeControllerClassWithAMethod{
70     public String myMethod(){
71         return "test";
72     }
73 }
74
75 @com.miage.alom.servlet.ServletController
76 class SomeControllerClassWithAVoidMethod{
77     @com.miage.alom.servlet.RequestMapping(uri="/test")
78     public void myMethod(){}
79 }
80
81 @com.miage.alom.servlet.ServletController
82 class SomeControllerClass {
83     @com.miage.alom.servlet.RequestMapping(uri="/test")
84     public String someGoodMethod(){
85         return "Hello";
86     }
87
88     @com.miage.alom.servlet.RequestMapping(uri="/test-throwing")
89     public String someThrowingMethod(){
90         throw new RuntimeException("some exception message");
91     }
92
93     @com.miage.alom.servlet.RequestMapping(uri="/test-with-params")
94     public String someThrowingMethod(Map<String, String[]> params){
95         return params.get("id")[0];
96     }
97 }
98
99 @com.miage.alom.servlet.ServletController

```

```

100 class SomeOtherControllerClass {
101     @com.miage.alom.servlet.RequestMapping(uri="/otherTest")
102     public String someOtherNiceMethod(){
103         return "Hello again";
104     }
105 }

```

- ① Notre classe de test
- ② Nos tests sont annotés `@Test`
- ③ Quelques contrôleurs d'exemple pour valider le fonctionnement de votre implémentation
- ④ On teste l'enregistrement du `HelloController`

4.3.3. La DispatcherServlet (code à trous)

Implémentez la servlet suivante :

La DispatcherServlet

```

1 package com.miage.alom.servlet;
2
3 import com.miage.alom.controller>HelloController;
4
5 import jakarta.servlet.ServletConfig;
6 import jakarta.servlet.ServletException;
7 import jakarta.servlet.annotation.WebServlet;
8 import jakarta.servlet.http.HttpServlet;
9 import jakarta.servlet.http.HttpServletRequest;
10 import jakarta.servlet.http.HttpServletResponse;
11 import java.lang.reflect.Method;
12 import java.util.HashMap;
13 import java.util.Map;
14
15 @WebServlet(urlPatterns = "/*", loadOnStartup = 1)
16 public class DispatcherServlet extends HttpServlet {
17
18     private Map<String, Method> uriMappings = new HashMap<>(); ①
19
20     @Override
21     protected void doGet(HttpServletRequest req, HttpServletResponse resp) {
22         System.out.println("Getting request for " + req.getRequestURI());
23         // TODO ③
24     }
25
26     @Override
27     public void init(ServletConfig config) throws ServletException {
28         super.init(config);
29         // on enregistre notre controller au démarrage de la servlet
30         this.registerController>HelloController.class);
31     }

```

```

32
33  /**
34   * This methods checks the following rules :
35   * - The controllerClass is annotated with @ServletController
36   * Then all methods are scanned and processed by the registerMethod method
37   * @param controllerClass the controller to scan
38   */
39  protected void registerController(Class controllerClass){
40      System.out.println("Analysing class " + controllerClass.getName());
41      // TODO ②
42  }
43
44  /**
45   * This methods checks the following rules :
46   * - The method is annotated with @RequestMapping
47   * - The @RequestMapping annotation has a URI
48   * - The method does not return void
49   * If these rules are followed, the method and its URI are added to the
    uriMapping map.
50   * @param method the method to scan
51   */
52  protected void registerMethod(Method method) {
53      System.out.println("Registering method " + method.getName());
54      // TODO ②
55  }
56
57  protected Map<String, Method> getMappings(){
58      return this.uriMappings;
59  }
60
61  protected Method getMappingForUri(String uri){
62      return this.uriMappings.get(uri);
63  }
64 }

```

- ① Cette `Map` va contenir l'association entre une URI et la méthode Java qui l'écoute (annotée `@RequestMapping`)
- ② C'est là qu'il faut coder !
- ③ Cette méthode sera implémentée dans la partie 4.4

Il faut maintenant implémenter les méthodes `registerController` et `registerMethod` pour faire passer les tests unitaires.



Cette partie fait un usage intensif de l'api `java.lang.reflect`

Vous aurez surement besoin des méthodes

- `getAnnotation`
- `getDeclaredMethods`

- `getDeclaredAnnotation`
- `newInstance`
- etc...

4.4. Le routage des requêtes (code à trous)

Une fois les annotations analysées, le routage des requêtes se fait de la manière suivante :

1. Récupération de l'URI entrante (depuis l'objet `HttpServletRequest`)
2. Récupération de la méthode implémentant l'URI (issue de l'analyse du code)
 - Si aucune méthode n'est trouvée, renvoyer une erreur 404
3. Instanciation du controller
4. Récupération des paramètres (depuis l'objet `HttpServletRequest`)
5. Appel de la méthode (avec les paramètres ou non)
 - En cas d'exception, renvoyer une erreur 500 avec le message de l'exception
 - En cas de succès, récupérer le résultat de l'appel, et renvoyer le résultat converti en chaîne de caractères

Nous devons donc ici, implémenter la méthode `doGet` de notre `DispatcherServlet`.

4.4.1. Les tests unitaires du routage

Ajoutez les tests suivants dans le test unitaire de la `DispatcherServlet` :

Les tests unitaires du routage

```

1 @Test
2 void doGet_shouldReturn404_whenNotMethodIsFound() throws IOException {
3     var servlet = new DispatcherServlet();
4
5     var req = mock(HttpServletRequest.class);
6     var resp = mock(HttpServletResponse.class);
7     when(req.getRequestURI()).thenReturn("/test");
8
9     servlet.doGet(req, resp);
10
11     verify(resp).sendError(404, "no mapping found for request uri /test");
12 }
13
14 @Test
15 void doGet_shouldReturn500WithMessage_whenMethodThrowsException() throws
    IOException {
16     var servlet = new DispatcherServlet();
17
18     servlet.registerController(SomeControllerClass.class);
19

```

```

20     var req = mock(HttpServletRequest.class);
21     var resp = mock(HttpServletResponse.class);
22     when(req.getRequestURI()).thenReturn("/test-throwing");
23
24     servlet.doGet(req, resp);
25
26     verify(resp).sendError(500,
27         "exception when calling method someThrowingMethod : some exception
    message");
28 }
29
30 @Test
31 void doGet_shouldReturnAResult_whenMethodSucceeds() throws IOException {
32     var servlet = new DispatcherServlet();
33
34     servlet.registerController(SomeControllerClass.class);
35
36     var req = mock(HttpServletRequest.class);
37     var resp = mock(HttpServletResponse.class);
38     var printWriter = mock(PrintWriter.class);
39
40     when(resp.getWriter()).thenReturn(printWriter);
41     when(req.getRequestURI()).thenReturn("/test");
42
43     servlet.doGet(req, resp);
44
45     verify(printWriter).print((Object)"Hello");
46 }
47
48 @Test
49 void doGet_shouldReturnAResult_whenMethodWithParametersSucceeds() throws
    IOException {
50     var servlet = new DispatcherServlet();
51
52     servlet.registerController(SomeControllerClass.class);
53
54     var req = mock(HttpServletRequest.class);
55     var resp = mock(HttpServletResponse.class);
56     var printWriter = mock(PrintWriter.class);
57
58     when(req.getRequestURI()).thenReturn("/test-with-params");
59     when(req.getParameterMap()).thenReturn(Map.of("id", new String[]{"12"}));
60     when(resp.getWriter()).thenReturn(printWriter);
61
62     servlet.doGet(req, resp);
63
64     verify(printWriter).print((Object)"12");
65 }
66
67 @Test
68 void doGet_shouldReturnAResult_forHelloController() throws IOException {

```

```

69     var servlet = new DispatcherServlet();
70     servlet.registerController(HelloController.class);
71
72     var req = mock(HttpServletRequest.class);
73     var resp = mock(HttpServletResponse.class);
74     var printWriter = mock(PrintWriter.class);
75
76     when(req.getRequestURI()).thenReturn("/hello");
77     when(resp.getWriter()).thenReturn(printWriter);
78
79     servlet.doGet(req, resp);
80
81     verify(printWriter).print((Object)"Hello World !");
82 }

```

Ces tests unitaires valident que les méthodes sont correctement appelées et que les erreurs sont renvoyées.

Vous devrez probablement ajouter l'import java suivant

```
import static org.mockito.Mockito.*;
```

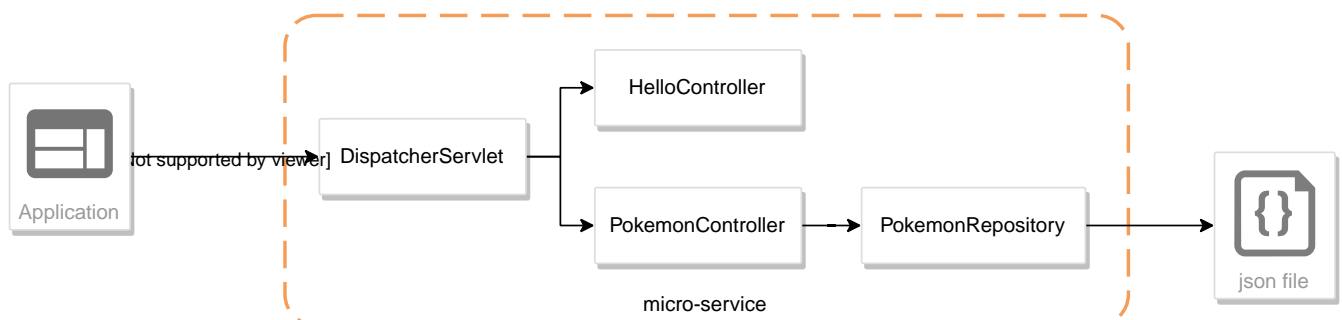


Une fois tous les tests au vert 😊, vous pouvez démarrer votre projet et requêter via votre navigateur web :

- <http://localhost:8080/hello>
- <http://localhost:8080/bye>
- <http://localhost:8080/boum>

5. Le micro-service PokemonType

Pour la suite de ce TP, nous allons développer un micro-service pokemon-type, qui s'appuiera sur notre DispatcherServlet. Ce micro-service a pour but de gérer les données de référence des pokémons, à savoir les 151 types de pokemon existants.



Le micro-service sera composé de 3 niveaux:

1. La DispatcherServlet

2. Le `PokemonController`, qui va exposer une route dédiée
3. Le `PokemonRepository`, qui va consommer un fichier JSON

Pour avoir quelques données à disposition, nous utiliserons les données de l'API <https://pokeapi.co>

5.1. La structure

Nous allons donner une structure à notre micro-service. Cette structure prendra la forme de packages Java.

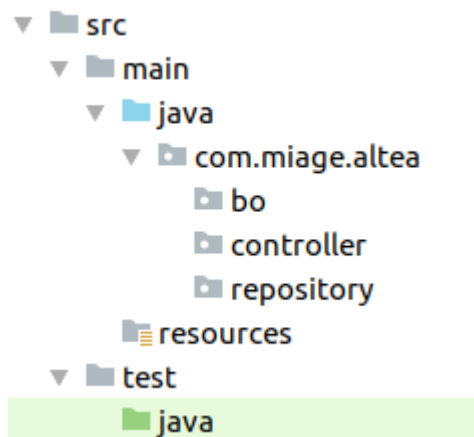


On retrouvera cette organisation de packages dans l'ensemble de nos TPs.

Créez les packages suivants :

- `com.miage.alom.bo`
- `com.miage.alom.controller`
- `com.miage.alom.repository`

Créez également le répertoire `src/main/resources`.



5.2. La classe `PokemonType`

Pour commencer, nous allons créer notre objet métier.

Pour implémenter notre objet, nous devons nous inspirer des champs que propose l'API <https://pokeapi.co>.

Par exemple, voici ce qu'on obtient en appelant l'API (un peu simplifiée) :

Electhor !

```
{
  "base_experience": 261,
  "height": 16,
  "id": 145,
  "moves": [],
```

```

    "name": "zapdos",
    "sprites": {
      "back_default":
"https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/back/145.png"
    ,
      "back_shiny":
"https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/back/shiny/145.png"
    ,
      "front_default":
"https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/145.png"
    ,
      "front_shiny":
"https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/shiny/145.png"
    }
  ,
  "stats": [
    {
      "base_stat": 100,
      "effort": 0,
      "stat": {
        "name": "speed",
        "url": "https://pokeapi.co/api/v2/stat/6/"
      }
    }
  ,
    {
      "base_stat": 90,
      "effort": 0,
      "stat": {
        "name": "special-defense",
        "url": "https://pokeapi.co/api/v2/stat/5/"
      }
    }
  ,
    {
      "base_stat": 125,
      "effort": 3,
      "stat": {
        "name": "special-attack",
        "url": "https://pokeapi.co/api/v2/stat/4/"
      }
    }
  ,
    {
      "base_stat": 85,
      "effort": 0,
      "stat": {
        "name": "defense",
        "url": "https://pokeapi.co/api/v2/stat/3/"
      }
    }
  ,
    {
      "base_stat": 90,
      "effort": 0,
      "stat": {

```

```

        "name": "attack",
        "url": "https://pokeapi.co/api/v2/stat/2/"
    },
    {
        "base_stat": 90,
        "effort": 0,
        "stat": {
            "name": "hp",
            "url": "https://pokeapi.co/api/v2/stat/1/"
        }
    }
],
"types": [
    {
        "slot": 2,
        "type": {
            "name": "flying",
            "url": "https://pokeapi.co/api/v2/type/3/"
        }
    },
    {
        "slot": 1,
        "type": {
            "name": "electric",
            "url": "https://pokeapi.co/api/v2/type/13/"
        }
    }
],
"weight": 526
}

```

Nous allons donc créer une classe Java qui reprend cette structure, mais en ne conservant que les champs qui nous intéressent.

com.miage.alom.bo.PokemonType.java

```

1 package com.miage.alom.bo;
2
3 public class PokemonType { ①
4
5     private int id;
6     private int baseExperience;
7     private int height;
8     private String name;
9     private Sprites sprites; ③
10    private Stats stats; ③
11    private int weight;
12
13    ②

```

```
14  
15 }
```

- ① On sélectionne les champs "id", "name", et "sprites"
- ② On a besoin des getters et setters par la suite (pour les générer, utilisez `Alt` + `Insert` sous IntelliJ)
- ③ Pour les objets imbriqués, on utilise d'autres classes

com.miage.alom.bo.Sprites.java

```
1 package com.miage.alom.bo;  
2  
3 public class Sprites {  
4  
5     private String back_default;  
6     private String front_default;  
7  
8 }
```

com.miage.alom.bo.Stats.java

```
1 package com.miage.alom.bo;  
2  
3 public class Stats {  
4  
5     private Integer speed;  
6     private Integer defense;  
7     private Integer attack;  
8     private Integer hp;  
9  
10 }
```

5.3. Le PokemonTypeRepository

Le repository est donc la classe qui va consommer notre fichier JSON et retourner notre Pokemon.

Le repository va utiliser l'API `jackson-databind` pour convertir le JSON en objet Java

5.3.1. jackson-databind

Ajouter la dépendance suivante à votre projet :

pom.xml

```
1 <dependency>  
2     <groupId>com.fasterxml.jackson.core</groupId>  
3     <artifactId>jackson-databind</artifactId>  
4     <version>2.15.2</version>
```

Ecrire un test unitaire pour apprendre à manipuler *jackson-databind* :

JacksonDatabindTest.java

```

1 class JacksonDatabindTest {
2
3     public static class Car { ①
4         public String color; ②
5         public String brand;
6     }
7
8     @Test
9     void testWriteJson() throws JsonProcessingException { ③
10         var objectMapper = new ObjectMapper();
11         var car = new Car();
12         car.color = "yellow";
13         car.brand = "renault";
14         var json = objectMapper.writeValueAsString(car);
15         assertEquals("{\"color\":\"yellow\",\"brand\":\"renault\"}", json);
16     }
17
18     @Test
19     void testReadJson() throws IOException { ④
20         var objectMapper = new ObjectMapper();
21         var json = "{\"color\":\"black\",\"brand\":\"opel\"}";
22         var car = objectMapper.readValue(json, Car.class);
23         assertEquals("black", car.color);
24         assertEquals("opel", car.brand);
25     }
26
27 }

```

- ① La classe qui représente nos données
- ② On positionne les champs en visibilité `public` pour ne pas avoir à écrire de getters/setters sur ce cas de test
- ③ L'écriture de JSON depuis notre objet
- ④ La lecture d'un JSON pour reconstruire un objet

Plus d'infos sur le [Github](#) de *jackson-databind*



Dans la DispatcherServlet, on peut utiliser *jackson-databind* pour transformer le résultat de nos appels de controllers en JSON !

5.3.2. Le jeu de données du repository

Récupérez le fichier *pokemons.json* et enregistrez-le dans le répertoire *src/main/resources* de votre

projet.

5.3.3. Les tests unitaires du repository

Comme pour la `DispatcherServlet`, nous allons travailler en TDD.

Voici la classe de tests unitaires à implémenter

com.miage.alom.repository.PokemonTypeRepositoryTest.java

```
1 package com.miage.alom.repository;
2
3 import org.junit.jupiter.api.Test;
4
5 import static org.junit.jupiter.api.Assertions.*;
6
7 class PokemonTypeRepositoryTest {
8
9     private PokemonTypeRepository repository = new PokemonTypeRepository();
10
11     @Test
12     void findPokemonById_with25_shouldReturnPikachu(){ ①
13         var pikachu = repository.findPokemonById(25);
14         assertNotNull(pikachu);
15         assertEquals("pikachu", pikachu.getName());
16         assertEquals(25, pikachu.getId());
17     }
18
19     @Test
20     void findPokemonById_with145_shouldReturnZapdos(){ ①
21         var zapdos = repository.findPokemonById(145);
22         assertNotNull(zapdos);
23         assertEquals("zapdos", zapdos.getName());
24         assertEquals(145, zapdos.getId());
25     }
26
27     @Test
28     void findPokemonByName_withEevee_shouldReturnEevee(){ ②
29         var eevee = repository.findPokemonByName("eevee");
30         assertNotNull(eevee);
31         assertEquals("eevee", eevee.getName());
32         assertEquals(133, eevee.getId());
33     }
34
35     @Test
36     void findPokemonByName_withMewTwo_shouldReturnMewTwo(){ ②
37         var mewtwo = repository.findPokemonByName("mewtwo");
38         assertNotNull(mewtwo);
39         assertEquals("mewtwo", mewtwo.getName());
40         assertEquals(150, mewtwo.getId());
41     }
42 }
```

```

42
43     @Test
44     void findAllPokemon_shouldReturn151Pokemons(){
45         var pokemons = repository.findAllPokemon();
46         assertNotNull(pokemons);
47         assertEquals(151, pokemons.size());
48     }
49
50 }

```

- ① On valide la récupération d'un pokemon par son id
- ② et par son nom

5.3.4. Le PokemonTypeRepository

Et voici la classe du repository, à compléter !

com.miage.alom.repository.PokemonTypeRepository.java

```

1  package com.miage.alom.repository;
2
3  import com.fasterxml.jackson.core.type.TypeReference;
4  import com.fasterxml.jackson.databind.ObjectMapper;
5  import com.miage.alom.bo.PokemonType;
6
7  import java.io.IOException;
8  import java.util.Arrays;
9  import java.util.List;
10
11 public class PokemonTypeRepository {
12
13     private List<PokemonType> pokemons;
14
15     public PokemonTypeRepository() {
16         try {
17             var pokemonsStream = this.getClass().getResourceAsStream
18                 ("/pokemons.json"); ①
19
20             var objectMapper = new ObjectMapper(); ②
21             var pokemonsArray = objectMapper.readValue(pokemonsStream,
22                 PokemonType[].class);
23             this.pokemons = Arrays.asList(pokemonsArray);
24         } catch (IOException e) {
25             e.printStackTrace();
26         }
27     }
28
29     public PokemonType findPokemonById(int id) {
30         System.out.println("Loading Pokemon information for Pokemon id " + id);
31     }
32 }

```

```

30      // TODO ③
31  }
32
33  public PokemonType findPokemonByName(String name) {
34      System.out.println("Loading Pokemon information for Pokemon name " + name);
35
36      // TODO ③
37  }
38
39  public List<PokemonType> findAllPokemon() {
40      // TODO ③
41  }
42 }

```

- ① On charge le fichier json depuis le classpath (maven ajoute le répertoire `src/main/resources` au classpath java !)
- ② On utilise l'ObjectMapper de `jackson-databind` pour transformer les objets JSON en objets JAVA
- ③ On a un peu de code à compléter !

5.4. Le PokemonTypeController

Écrire un controller qui expose une route `"/pokemon"`. Cette route pourra être appelée avec des paramètres éventuels, `id` ou `name`.

Les requêtes devant être implémentées sont donc, par exemple :

- <http://localhost:8080/pokemon?id=25>
- <http://localhost:8080/pokemon?id=145>
- <http://localhost:8080/pokemon?name=pikachu>
- <http://localhost:8080/pokemon?name=zapdos>

5.4.1. Les tests unitaires du PokemonTypeController

Implémenter les tests unitaires suivants :

com.miage.alom.controller.PokemonTypeControllerTest.java

```

1  package com.miage.alom.controller;
2
3  import com.miage.alom.bo.PokemonType;
4  import com.miage.alom.repository.PokemonTypeRepository;
5  import org.junit.jupiter.api.BeforeEach;
6  import org.junit.jupiter.api.Test;
7  import org.mockito.InjectMocks;
8  import org.mockito.Mock;
9  import org.mockito.MockitoAnnotations;
10
11 import java.util.Map;

```

```

12
13 import static org.junit.jupiter.api.Assertions.*;
14 import static org.mockito.Mockito.*;
15
16 class PokemonTypeControllerTest {
17
18     @InjectMocks
19     PokemonTypeController controller;
20
21     @Mock
22     PokemonTypeRepository pokemonRepository;
23
24     @BeforeEach
25     void init(){
26         MockitoAnnotations.initMocks(this);
27     }
28
29     @Test
30     void getPokemon_shouldRequireAParameter(){
31         var exception = assertThrows(IllegalArgumentException.class,
32             () -> controller.getPokemon(null));
33         assertEquals("parameters should not be empty", exception.getMessage());
34     }
35
36     @Test
37     void getPokemon_shouldRequireAKnownParameter(){
38         var parameters = Map.of("test", new String[]{"25"});
39         var exception = assertThrows(IllegalArgumentException.class,
40             () -> controller.getPokemon(parameters));
41         assertEquals("unknown parameter", exception.getMessage());
42     }
43
44     @Test
45     void getPokemon_withAnIdParameter_shouldReturnAPokemon(){
46         var pikachu = new PokemonType();
47         pikachu.setId(25);
48         pikachu.setName("pikachu");
49         when(pokemonRepository.findPokemonById(25)).thenReturn(pikachu);
50
51         var parameters = Map.of("id", new String[]{"25"});
52         var pokemon = controller.getPokemon(parameters);
53         assertNotNull(pokemon);
54         assertEquals(25, pokemon.getId());
55         assertEquals("pikachu", pokemon.getName());
56
57         verify(pokemonRepository).findPokemonById(25);
58         verifyNoMoreInteractions(pokemonRepository);
59     }
60
61     @Test
62     void getPokemon_withANameParameter_shouldReturnAPokemon(){

```

```

63     var zapdos = new PokemonType();
64     zapdos.setId(145);
65     zapdos.setName("zapdos");
66     when(pokemonRepository.findPokemonByName("zapdos")).thenReturn(zapdos);
67
68     var parameters = Map.of("name", new String[]{"zapdos"});
69     var pokemon = controller.getPokemon(parameters);
70     assertNotNull(pokemon);
71     assertEquals(145, pokemon.getId());
72     assertEquals("zapdos", pokemon.getName());
73
74     verify(pokemonRepository).findPokemonByName("zapdos");
75     verifyNoMoreInteractions(pokemonRepository);
76 }
77
78 @Test
79 void pokemonTypeController_shouldBeAnnotated(){
80     var controllerAnnotation =
81         PokemonTypeController.class.getAnnotation(ServletController.class);
82     assertNotNull(controllerAnnotation);
83 }
84
85 @Test
86 void getPokemon_shouldBeAnnotated() throws NoSuchMethodException {
87     var getPokemonMethod =
88         PokemonTypeController.class.getDeclaredMethod("getPokemon",
89     Map.class);
89     var requestMappingAnnotation =
90         getPokemonMethod.getAnnotation(ServletRequestMapping.class);
91
92     assertNotNull(requestMappingAnnotation);
93     assertEquals("/pokemons", requestMappingAnnotation.uri());
94 }
95
96 }

```

5.4.2. Le PokemonTypeController (code à trous)

Implémenter le PokemonTypeController et compléter la méthode !

com.miage.alom.controller.PokemonTypeController.java

```

1 package com.miage.alom.controller;
2
3 import com.miage.alom.bo.PokemonType;
4 import com.miage.alom.repository.PokemonTypeRepository;
5
6 import java.util.Map;
7
8 public class PokemonTypeController {

```

```
9     private PokemonTypeRepository repository = new PokemonTypeRepository();
10
11     public PokemonType getPokemon(Map<String,String[]> parameters){
12         // TODO
13     }
14 }
```



Peut-être faut-il ajouter des annotations java sur le controller pour l'enregistrer auprès de la `DispatcherServlet`.

5.5. Modifications de la DispatcherServlet

Enfin, pour finaliser notre développement, nous devons :

1. Enregistrer notre `PokemonTypeController` dans la `DispatcherServlet` (en modifiant la méthode `init` de la `DispatcherServlet`)
2. Utiliser `jackson-databind` pour transformer les résultats de nos contrôleurs en JSON
3. Ne pas oublier de transmettre les paramètres reçus en requête au contrôleur !

Testez votre micro-service en consultant les urls suivantes :

- <http://localhost:8080/pokemon?id=25>
- <http://localhost:8080/pokemon?id=145>
- <http://localhost:8080/pokemon?name=pikachu>
- <http://localhost:8080/pokemon?name=zapdos>