Haymoz Alexandre | Classe 300232
Module 184
Mr. Clerc

Rapport personnel

Développer des applications WEB

Version 1 du 09.12.2022

E — M — F Ecole des Métiers
Berufsfachschule
Fribourg
Informatique

Module du 23.03.2023 au
05.05.2023

TABLE DES MATIERES

1 Introduction

1.1 Objectifs

- Connaître des solutions possibles afin de délimiter les fonctionnalités côté client comme serveur en se basant sur les directives.
- Connaître les éléments d'un concept de réalisation pour des applications Web.
- Connaître les avantages de la séparation entre la présentation et la logique du programme ainsi que leur mise en œuvre dans une application Web.
- Connaître des éléments de formulaires ainsi que des fonctions pour vérifier les saisies de l´utilisateur, et comment ceux-ci mis en œuvre conformément aux directives.
- Connaître des techniques courantes pour la réalisation du suivi de session.
- Connaître les risques sur la sécurité et des solutions possibles pour protéger une application Web.
- Connaître des possibilités d'administration de lots de données.
- Connaître des possibilités pour garantir l'authentification et les autorisations dans une application Web.
- Connaître des langages de programmation possibles pour des applications Web.
- Connaître des solutions possibles pour la mise en œuvre d'architectures appropriées pour des applications Web.
- Connaître des possibilités de structuration de code source ainsi que leur mise en œuvre en tenant compte des directives de codification.
- Connaît des procédures de tests, et leur contribution, pour garantir la qualité d'applications Web.
- Connaître des procédures de tests permettant de vérifier les solutions possibles contre les Cross-Site-Scripting, Script-Injection et Session-Hijacking.

2 Tests technologiques

2.1 Tizoo déploiement sur Tomcat

La seule chose notable à démarquer et que les fichiers .war (fichier JSP compilé) peuvent de déployer de manière automatique avec Tomcat. Les conditions pour que les fichiers se déploient automatiquement et que leur nom commence par java et que la configuration Tomcat permet le déploiement automatique.

2.2 Préparation Tomcat locale

Apache Tomcat est un conteneur de servlets Java open source de longue date (depuis plus 1999 !) qui met en œuvre les principales spécifications de Java Enterprise (aujourd'hui Jakarta EE), notamment les spécifications **Jakarta** Servlet, **Jakarta** Server Pages et **Jakarta** WebSocket. Tomcat est aussi un serveur d'application (conteneur). Un serveur d'applications est un programme sur un ordinateur qui gère toutes les opérations d'application entre les utilisateurs et les applications commerciales ou les bases de données d'une organisation.

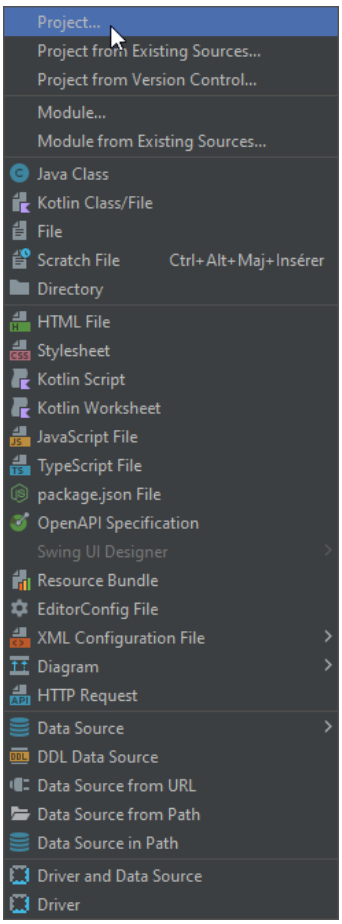
Jakarta EE est un ensemble de composants logiciels, d'API, permettant de développer des applications Java spécifiques aux services web. Ces composants sont souvent appelés spécifications.

Tomcat n'est pas le seul serveur Web Java, il y a par exemple Jetty, Apache TomEE, Oracle WebLogic, WebSphere, WildFly, Apache Geronimo et GlassFish

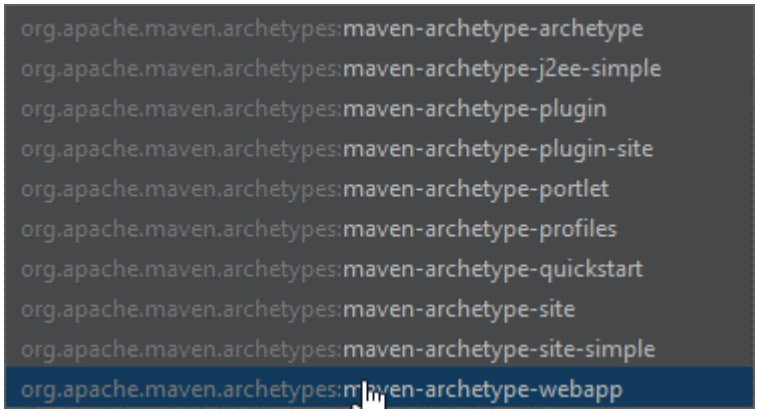
2.3 HTML & JSP

Afin de pouvoir configurer l'IDE pour un projet JSP sous Intellij IDEA et Maven, il faut suivre les étapes suivantes :

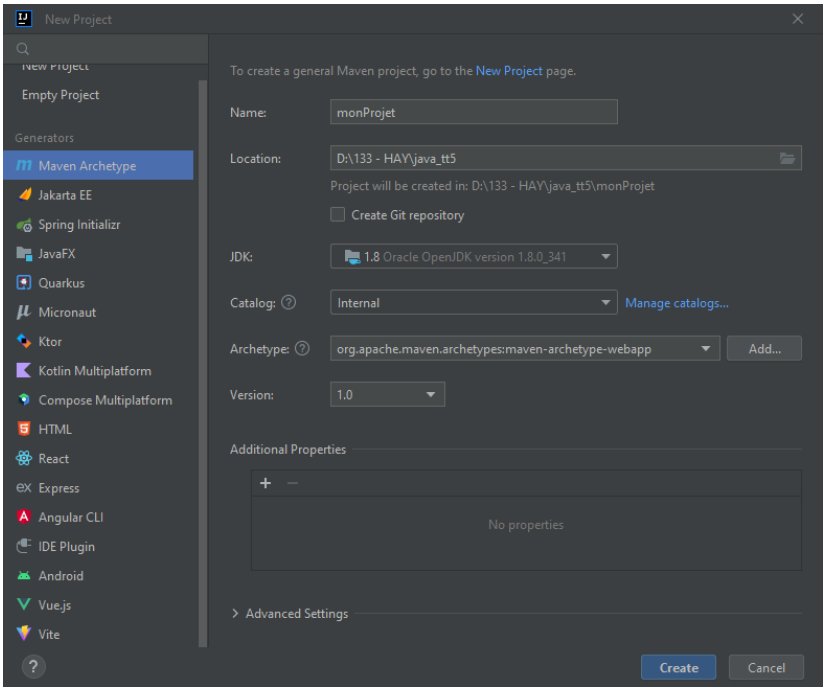
Il faut d'abord créer un nouveau projet :



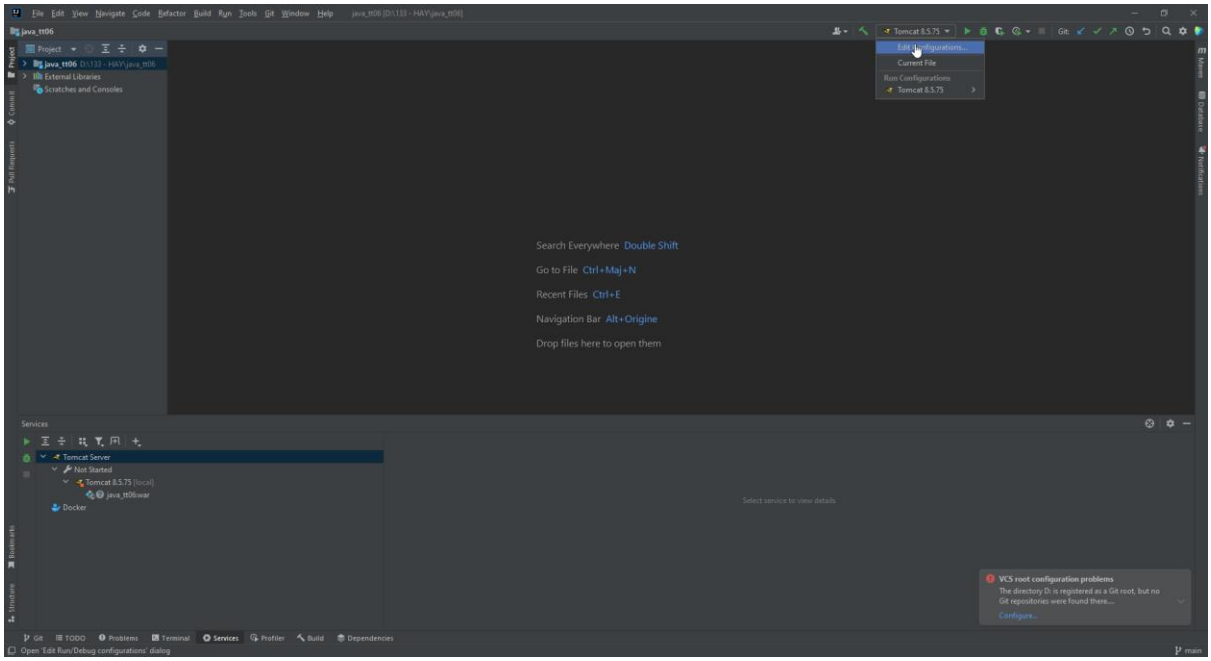
Un archétype est l'architecture que le projet Maven va prendre, on choisi ici l'archétype d'une application Web afin de bénéficier d'un design similaire à celui de NetBeans.



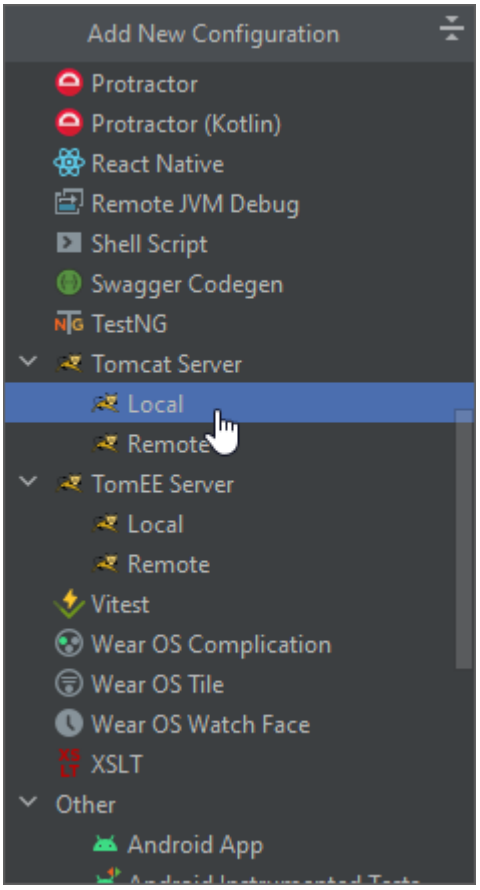
La vue globale devrait ressembler à ça :



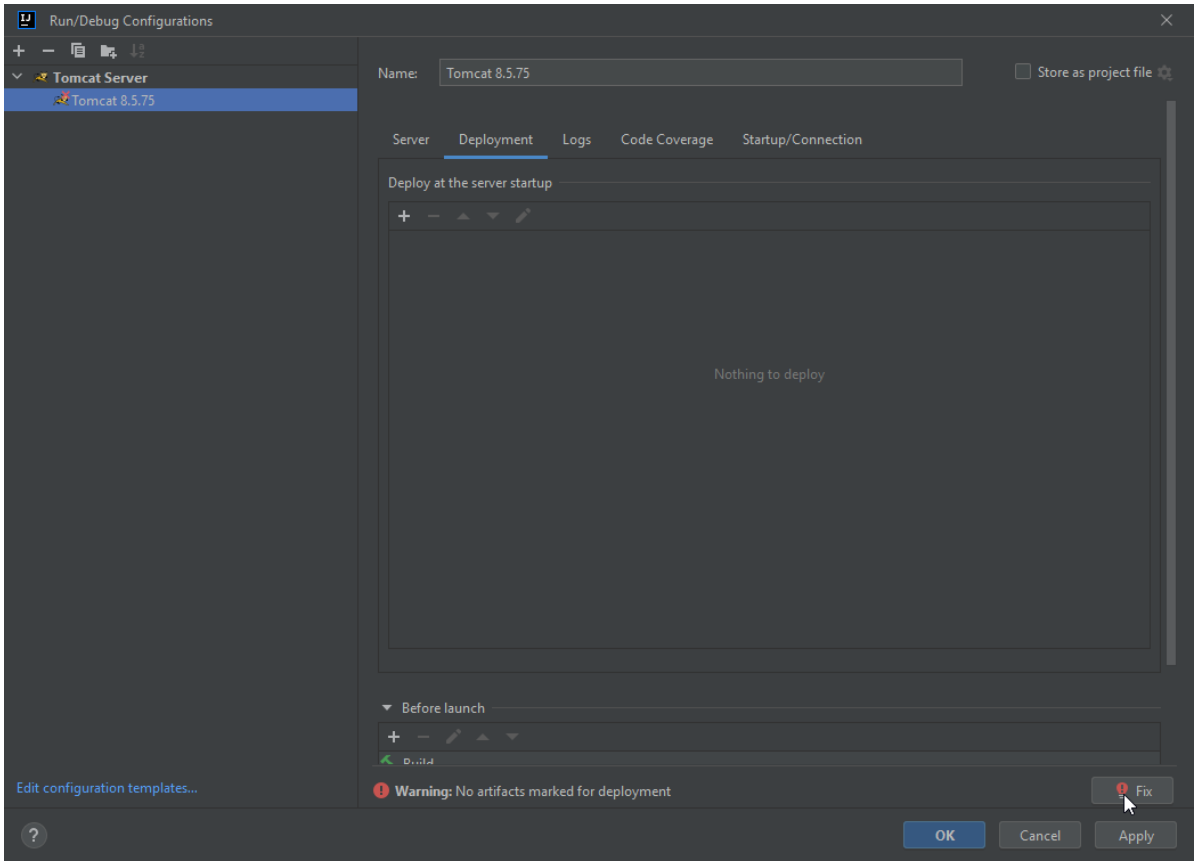
Il est nécessaire de lier le serveur Tomcat au projet afin de pouvoir afficher le fichier .war directement sans devoir à l'héberger dans le dossier /webapp du serveur Tomcat installé précédemment.



Il faut cliquer en haut à droite afin d'ajouter une nouvelle configuration et choisir le Tomcat Server.



On doit ici choisir quelle artifact sera converti en .war, soit l'artifact du projet. On peut simplement cliquer sur Fix en bas à gauche te choisir le premier choix.



2.4 HTML appelant JSP

Un scriptlet peut contenir un nombre quelconque d'instructions en langage Java, de déclarations de variables ou de méthodes, ou d'expressions valables dans le langage de script de la page.

La syntaxe est :

```
<% code java %>
```

On peut y accéder via Tomcat comme on accède à une page HTML.

Un élément d'expression JSP contient une expression de langage de script qui est évaluée, convertie en chaîne et insérée à l'endroit où l'expression apparaît dans le fichier JSP.

```
<%= code java %>
```

Par exemple :

```
<html>
   <head><title>JSP Test</title></head>

   <body>
      <p>La date aujourd'hui est: <%= (new
java.util.Date()).toLocaleString()%></p>
   </body>
</html>
```

2.5 JSP consommation de WebService

Une requête HTTP peut être construite par n'importe quelle application dans n'importe quel langage, y compris Java, ASP, PHP, etc. Dans ce cas, on a dû créer une requête http qui appelé un service (prévisions météorologiques) d'une autre page (prevision-meteo.ch). Ici on a dû simplement concaténer le nom de la ville qu'un utilisateur entre dans l'URL d'une requête GET vers cette page, récupérer les informations et les affichées à l'utilisateur.

```
    <body>
        <h1>Contrôle du login par JSP</h1>
        <!-- Insertion de code java -->
        <%
            String ville = request.getParameter("ville");
        %>
        <p> Voici la météo de <%=ville%> </p> <br>
        <img src="https://www.prevision-meteo.ch/uploads/widget/<%=ville%>_0.png"
width="650" height="250"/>
    </body>
```

2.6 JSP Worker et JDBC

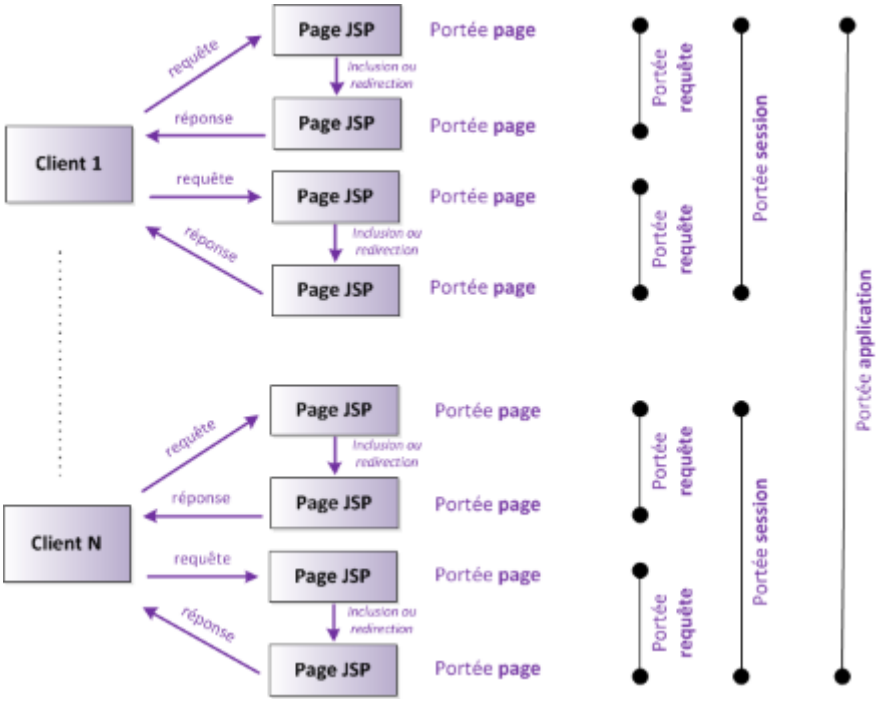
Une ressource JDBC fournit aux applications un moyen de se connecter à une base de données. L'accès transactionnel aux ressources JDBC est disponible à partir de servlets, de pages JSP et de beans d'entreprise. Les fonctions de mise en commun des connexions et de transactions distribuées sont destinées à être utilisées par les pilotes JDBC pour assurer la coordination avec un serveur d'application.

2.7 JSP appelle JSP avec Bean

Les beans en JSP sont appelées avec la syntaxe suivante :

```
<jsp:useBean id="beanInfo" scope="session" class="beans.BeanInfo"/>
```

L'id fait référence au nom du bean, ce nom est unique pour le scope de celui-ci. Un scope est la portée dû bean. Ici, le bean a un scope de session ce qui signifie que le bean interagit uniquement avec la session de l'utilisateur, mais est partagée par toutes les pages JSP. Un scope d'une page définit le bean uniquement pour la page, une autre page JSP n'y aura pas accès. Un scope application définit le bean pour l'entièreté de l'application, ce bean est partagé entre les sessions.



Pour importe une classe, la syntaxe suivante est utilisée :

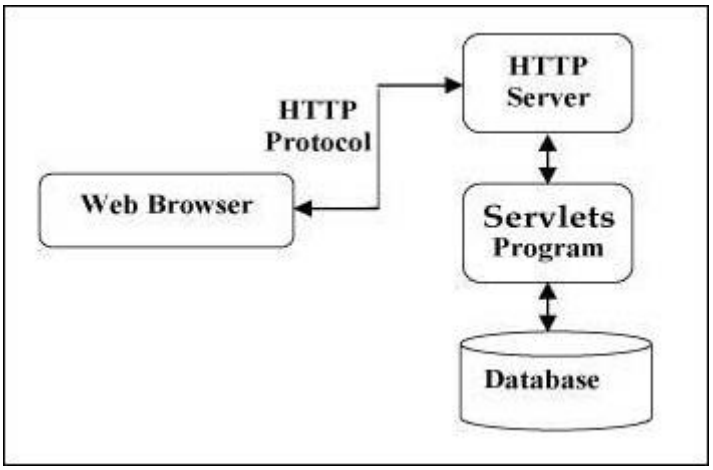
```
<%@ page import='fichier1.file'%>
```

La syntaxe pour rediriger un utilisateur vers une page JSP est la suivante :

```
<jsp:forward page='folder1.folder2.fileName.jsp'/>
```

2.8 Servlet simple

Les servlets sont des classes Java qui gèrent les requêtes HTTP et mettent en œuvre l'interface javax.servlet.Servlet. Les servlets communs sont généralement des servlets qui étendent javax.servlet.http.HttpServlet, une classe abstraite qui implémente l'interface Servlet et qui est spécialement conçue pour traiter les requêtes HTTP.



Une servlet peut traiter un ou plusieurs types de requêtes grâce à plusieurs autres méthodes

- doGet() : pour les requêtes http de type GET
- doPost : pour les requêtes http de type POST

- doHead() : pour les requêtes http de type HEAD
- doPut() : pour les requêtes http de type PUT
- doDelete() : pour les requêtes http de type DELETE
- doOptions() : pour les requêtes http de type OPTIONS
- doTrace() : pour les requêtes http de type TRACE

Une servlet peut aussi générer une réponse

- void sendError(int) : Envoie une erreur avec un code retour et message par défaut
- void sendError(int, String) : Envoie une erreur avec un code retour et un message
- void setContentType(String) : Héritée de ServletResponse, cette méthode permet de préciser le type MIME de la réponse
- void setContentLength(int) : Héritée de ServletResponse ,cette mtéhode permet de préciser la longueur de la réponse
- ServletOutputStream getOutputStream() : Héritée de ServletResponse, elle retourne un flux pour l'envoi de la réponse
- PrintWriter getWriter() : Héritée de ServletResponse, elle retourne un flux pour l'envoi de la réponse