# Jakarta EE

## Table des matières



© AFPA 2025

# Contexte

- Vous allez créer au fil de votre lecture un projet Jakarta EE.
- Ce projet sera une gestion des clients et des prospects d'une entreprise.
  - Dans ce document, vous allez créer un site qui pourra lister, créer, modifier ou supprimer les clients et les prospects
  - Dans le document sur la persistance et l'authentification
    - ✓ Vous ferez une authentification par mot de passe crypté/haché
    - ✓ Vous sécuriserez votre site à l'aide de tokens CSRF
- Vous utiliserez Maven et Cargo.
- Vous pouvez utiliser l'IDE qui vous convient le mieux. Celui auquel je ferai référence est Intellij Ultimate

❖ Ce symbole vous dira quel travail à faire

 Celui-ci les liens pour en savoir plus ou des informations à connaître

➔ Enfin celui-là vous indiquera les ressources à lire et/ou à utiliser

© AFPA 2025

## 1. Présentation générale

- Jakarta EE de la fondation Eclipse remplace Java EE 8 d'Oracle. Il est entièrement compatible avec les spécifications de Java EE 8 et inclut les mêmes interfaces de programmation applicative (API)

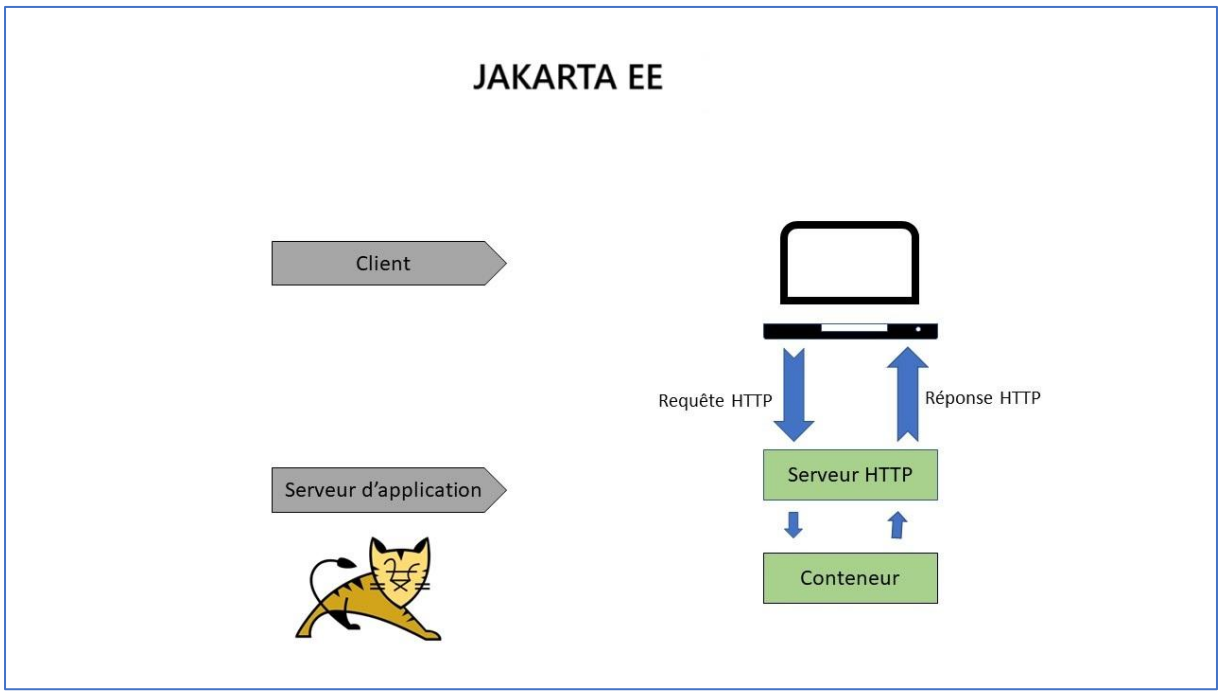
- Jakarta EE est une spécification (ensemble explicite d'exigences à satisfaire par un matériau, produit ou service) pour la plate-forme Java et il est destiné aux entreprises.

- Il inclut Java SE (Java platform Standard Edition)

➔ https://fr.wikipedia.org/wiki/Jakarta_EE

Vous trouverez dans la ressource ci-dessus les interfaces de programmation de Jakarta EE

Nous en utiliserons certaines : Servlet, JavaServer Pages (JSP), Java Standard Tag Library (JSTL), JDBC, JPA.
Jakarta EE s'exécute sur un serveur d'applications. Nous prendrons Tomcat.




© AFPA 2025

Le client demande une page web à l'aide d'une URL et l'envoie au serveur via une **Requête** HTTP.

Le serveur, après traitement, va renvoyer une page HTML via une **Réponse** HTTP.

Le serveur d'application Tomcat contient, entre autres choses, un serveur Web Apache (appelé aussi serveur HTTP) et un conteneur de servlets.

Le serveur Web reçoit la requête HTTP, la découpe, l'analyse et l'envoie au conteneur où se trouve Jakarta EE

Documentation de Jakarta EE :

https://jakarta.ee/learn/#documentation

**De nombreuses documentations trouvées sur le Web sont pour des versions antérieures à Jakarta EE, notamment les documentations docs.oracle.com**

Néanmoins, certaines documentations feront référence à docs.oracle.com quand cela sera nécessaire ou judicieux.

© AFPA 2025

## 2. Tomcat

C'est un serveur d'application comportant un serveur http Apache

https://fr.wikipedia.org/wiki/Apache_Tomcat

Spécifications Tomcat

https://tomcat.apache.org/whichversion.html

Installation de Tomcat :

https://www.liquidweb.com/kb/installing-tomcat-9-on-windows/

La démarche est la même pour Tomcat 10 en utilisant le lien :
https://tomcat.apache.org/download-10.cgi

Adresse URL du gestionnaire d'applications WEB Tomcat

http://localhost:numerodeport/manager

En cas de problème de connexion, modifier le fichier conf se trouvant généralement au chemin suivant :

C:\Program Files\Apache Software Foundation\Tomcat XX.X\conf

Dans le fichier : tomcat-users.xml

```
<tomcat-users …>
        <user username="tomcat" password="tomcat" roles="manager-gui,admin-gui,manager-script"/>
        <role rolename="manager-gui"/>
  <role rolename="admin-gui"/>
</tomcat-users>
```

© AFPA 2025

Les infos sont visibles dans les fichiers logs de Tomcat se trouvant au même niveau que le fichier conf :

C:\Program Files\Apache Software Foundation\Tomcat 10.x\logs

Vous devrez passer par un Logger pour voir vos anomalies :

private static final Logger LOGGER = Logger.getLogger(**NomClass**.class.getName());

LOGGER.severe message) ;

Ils seront dans le log catalina du dossier logs

Les erreurs d'exécution seront dans le log localhost

## 3. Création d'un programme

➔ Installation de Maven si vous ne l'avez pas : http://maven.apache.org/install.html

❖ Créez votre application Jakarta EE avec Intellij Ultimate :

- ✓ File => new projet
- ✓ jakartaEE
- ✓ java
- ✓ Maven
- ✓ Template web application
- ✓ Choisir le bon JDK
- ✓ Next
- ✓ Spécifications : Web Profile
- ✓ Implementations : Hibernate et Hibernate Validator
- ✓ Create
- ✓ Load maven project

© AFPA 2025

❖ Regardez l'arborescence de votre projet avec l'explorateur de dossiers et dans votre IDE

- Un dossier webapp a été créé. Il contient les fichiers de l'application web qui sont publics. Il se trouve à la racine de l'application lors de l'exécution. Un fichier index.jsp y a été généré. C'est une JSP (Java Server Page) que vous verrez chapitre 10.
- Regardez le fichier index.jsp avec votre IDE, c'est un simple fichier contenant du HTML.
- Le sous dossier WEB-INF, quant à lui, masque au visiteur ce qu'il contient. Un fichier web.xml a été créé. C'est un descripteur de déploiement. Il fournit aux composants de l'application les informations pour sa configuration et son déploiement.

❖ Configuration du fichier web.xml se trouvant dans le package WEB-INF

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd"
        version="5.0">
```

❖ Modifiez le pom.xml avec la version de votre JDK

```
<maven.compiler.target>21</maven.compiler.target>

<maven.compiler.source>21</maven.compiler.source>
```

Les dépendances de JUNIT 5 (jupiter) sont présentes.

❖ En ligne de commande, taper : mvn package
Dans le dossier target, un fichier .war a été crée

❖ Aller dans le manager de tomcat : http://localhost:8080/manager/html
Dans la partie « Fichier WAR à déployer », choisissez votre fichier .war et cliquer sur deployer.

❖ Dans la partie « Applications », cliquer sur le chemin de votre application.

© AFPA 2025

## 4. Cargo

Cargo est un Wrapper (programme « enveloppe » permettant d'appeler un ou des autres programmes) qui permet de manipuler divers types de conteneurs d'application comme jakarta EE.

Via Maven, vous pourrez configurer, démarrer et déployer l'application dans Tomcat

https://codehaus-cargo.github.io/cargo/Tomcat+10.x.html

❖ Code à rajouter à votre fichier pom.xml

```
<plugin>
        <groupId>org.codehaus.cargo</groupId>
        <artifactId>cargo-maven3-plugin</artifactId>
        <version>1.10.3</version>
        <configuration>
          <container>
              <containerId>tomcat10x</containerId>
              <type>remote</type>
          </container>
          <configuration>
            <type>runtime</type>
            <properties>
              <cargo.tomcat.context.reloadable>true</cargo.tomcat.context.reloadable>
              <cargo.remote.username>user</cargo.remote.username>
              <cargo.remote.password>mdp</cargo.remote.password>
              <cargo.servlet.port>8080</cargo.servlet.port>
            </properties>
          </configuration>
        </configuration>
      </plugin>
```

© AFPA 2025

Rq : Vous pouvez mettre les valeurs des properties suivantes. Cela devra être fait pour toute livraison.

- dans le fichier ~/.m2/setting.xml

```
<settings>
        <activeProfiles>
                <activeProfile>local</activeProfile>
        </activeProfiles>
</settings>
<profiles>
        <profile>
          <id>local</id>
          <properties>
         <catalina.manager.url>http://localhost:8080/manager</catalina.manager.url>
             <catalina.admin.username>user</catalina.admin.username>
             <catalina.admin.password>mdp</catalina.admin.password>
          </properties>
        </profile>
</profiles>
```

- Dans le fichier pom.xml

```
<cargo.remote.username>${catalina.admin.username}</cargo.remote.username>
<cargo.remote.password>${catalina.admin.password}</cargo.remote.password>
<cargo.tomcat.manager.url>${catalina.manager.url</cargo.tomcat.manager.url>
```

- Pour déployer ou retirer l'application dans Tomcat

mvn cargo:deploy

mvn cargo:undeploy

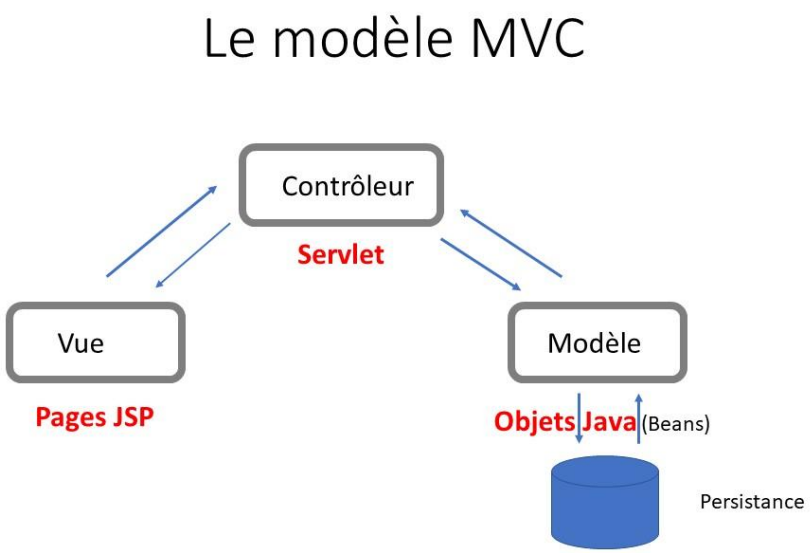
mvn cargo:redeploy <= une fois que vous avez lancé une première fois Cargo, faites

redeploy qui fait ces 3 phases dans cet ordre : Redeploying, Undeploying, Deploying

 Parfois, le « redeploy » ne fonctionne pas correctement. Passer par le Ajoutez un des 2 plugins dans votre fichier POM.XML

© AFPA 2025

## 4. Le design pattern MVC




Le modèle MVC est le pattern d'architecture préconisé par Jakarta EE et en règle générale pour les applications Web. Nous l'utiliserons dans ce projet.

Dans le modèle **MVC de JakartaEE** :

La partie **Modèle** est constitué de :

- Les Beans : classe Java contenant les objets « métiers » de l'application
- Les classes Java destinées à faire le lien avec la persistance de l'application (bases de données, fichiers, API)

La partie **Contrôleur** est constituée

- **D'une ou plusieurs servlets** (voir paragraphe suivant) qui sont des classes Java qui vont communiquer avec le web via des requêtes http
- De classes Java permettant de préparer la Vue.

La partie **Vue** est constituées

- D'une ou plusieurs Java Server Page (JSP) générant les pages Web contenant le code HTML destiné aux navigateurs.
  **Les JSP sont des servlets** (vous le verrez dans le paragraphe dédié aux JSP).

© AFPA 2025

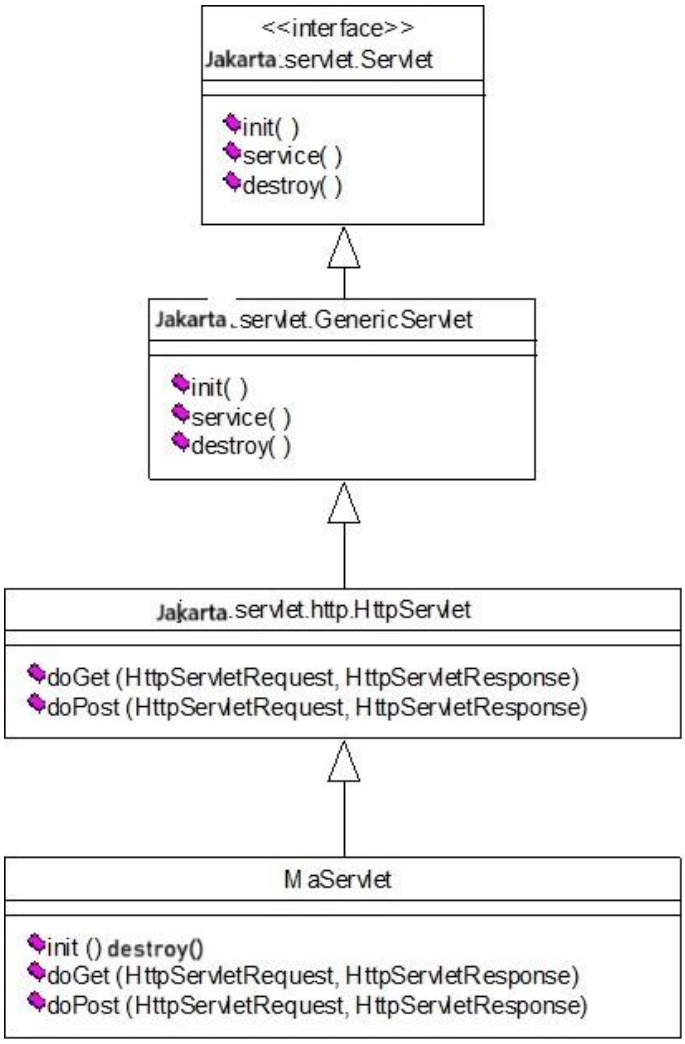
## 5. La Servlet

C'est un objet d'une classe Java exécuté dans le conteneur du serveur d'application et qui va communiquer avec le Web via des requêtes HTTP.

Les objets servlet sont « multi-threadés » c'est-à-dire qu'il n'y aura qu'une seule instance de la servlet, même si plusieurs clients y accèdent en même temps.

Une servlet hérite de la classe HttpServlet qui se trouve dans l'API jakarta.servlet (jakarta.servlet.http.HttpServlet).

Elle reçoit une requête du client (via les méthodes doGet et doPost) et renvoie une page de données, généralement au format HTML.




© AFPA 2025

➢ Le cycle de vie de la servlet : 5 étapes

1. Chargement de la servlet en mémoire (load) au démarrage de l'application
2. Création d'une instance de la classe
3. Appel de la méthode init()
4. Appel de la méthode service()
5. Appel de la méthode destroy()

La servlet a 2 méthodes, doGet et doPost ayant pour paramètres Request (retour) et Response (envoi) de la requête http.

Rappel :

- La méthode get permet un passage de paramètre dans l'url (URL?nomParamètre=valeurParamètre) ou une validation de formulaire avec cette méthode
- La méthode post est utilisée lors de la validation de formulaire avec cette méthode

- Les étapes 1,2,3 ne sont exécutées que quand l'application démarre ou quand la servlet est montée en mémoire

- L'étape 4 est exécutée quand la servlet est appelée par une requête http

La méthode service() héritée de HttpServlet appelle l'une ou l'autre de ces méthodes en fonction du type de la requête http.

- L'étape 5 est exécutée quand le container de la servlet décharge la servlet (unload)

❖ HelloServlet.java

Il y a la méthode init() et la méthode destroy().
La méthode doGet est appelée.
La réponse est de type text et génère du code HTML

© AFPA 2025

❖ Vous allez créer une méthode processRequest qui sera appelée par la méthode doGet() et la méthode doPost()

```
protected void processRequest(HttpServletRequest request,
HttpServletResponse response) {
    try {
        request.getRequestDispatcher("index.jsp")
                .forward(request, response);
    } catch (ServletException | IOException e) {
        LOGGER.severe("pb forward" + e.getMessage());
    }
}

@Override
protected void doGet(HttpServletRequest request, HttpServletResponse
response)
    {
    processRequest(request, response);
}
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse
response)
    {
    processRequest(request, response);
}
```

La méthode getRequestDispatcher de l'Objet HttpServletRequest redirige la requête HTTP en lui indiquant la servlet à traiter. Une JSP, comme vous le verrez dans le chapitre consacré aux JSP, est une servlet.
Notez l'exception ServletException levée lors d'une erreur de traitement de la servlet

https://jakarta.ee/specifications/servlet/4.0/apidocs/javax/servlet/servletexception

➢ Vous remarquerez l'**annotation** @WebServlet qui est une métadonnée demandant comme paramètre l'URL souhaitée de la première page de l'application. Vous pouvez mettre urlPatterns = {"/accueil"} par exemple

© AFPA 2025

Vous pouvez aussi lui passer en paramètre (name=) le nom que vous voulez donner à votre Servlet si vous en voulez un différent de celui de la classe.

Beaucoup de ressources sur le web utilisent encore le fichier web.xml pour déclarer les servlets. Nous utiliserons les annotations.

Dans le fichier web.xml (qui fournit des informations de configuration et de déploiement pour les composants d'une application Web), il faut par contre déclarer l'adresse URL de la première servlet appelée par l'application, ici l'adresse URL (value) que vous avez dans votre Servlet (sans le /)

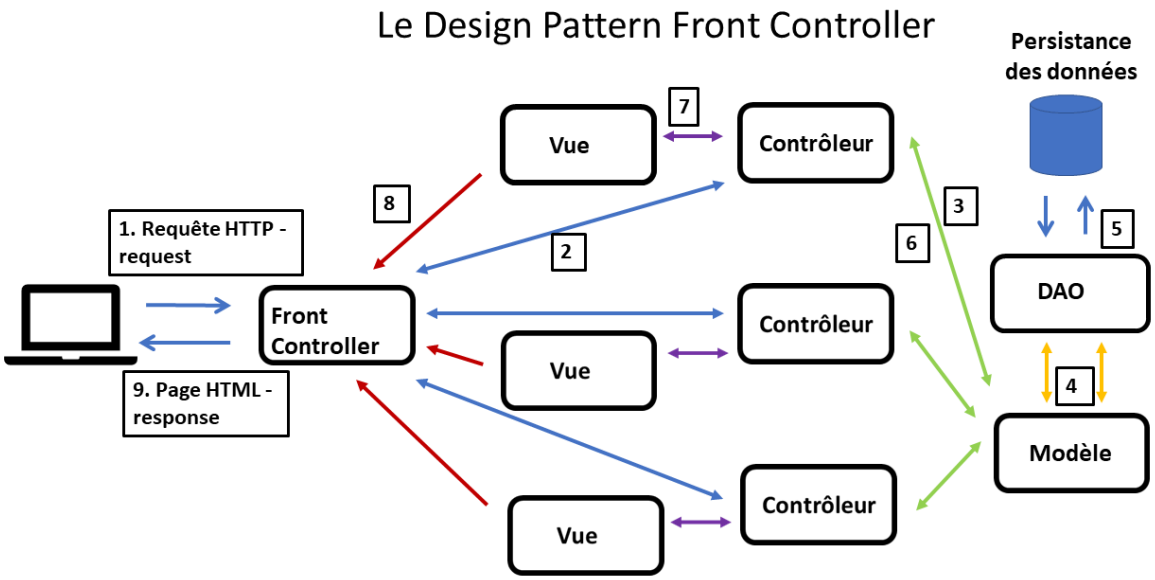
❖ Modifiez votre fichier web.xml :

```
<welcome-file-list>
        <welcome-file> AdresseUrl </welcome-file>
</welcome-file-list>
```

❖ Refaite un package et redéployez votre application (mvn package cargo:redeploy ou mvn package cargo:run) et vérifiez l'affichage de votre application avec l'adresse URL du fichier web.xml (/AdresseUrl).
Vous verrez apparaître le contenu de la JSP index.jsp

© AFPA 2025

## 6. Le Pattern Front Controller




https://en.wikipedia.org/wiki/Front_controller

Le pattern d'architecture Front Controller est utilisé par de nombreuses applications Web.

Son principe est que les requêtes HTTP arrivent et sont dispatchées au même endroit, par une servlet dédiée à cette fonction, **le Front Controller** et **qui sera la seule servlet de l'application**, donc la seule classe Java implémentant l'interface Servlet, et la seule à posséder les méthodes doGet et doPost.

Le Front Controller est souvent appelé routeur.

On utilisera dans ce cours le Design Pattern Command pour l'implémenter

Chaque vue sera reliée à un contrôleur qui implémentera une interface ICommand

**Un contrôleur, avec le Design Pattern Front Controller, n'est pas une servlet.** Seule la classe FrontController sera une servlet. Les contrôleurs sont de simples classes java qui ont comme paramètres HttpServletRequest et HttpServletResponse provenant de la requête http.

© AFPA 2025

Il est donc judicieux de ne pas mettre les contrôleurs dans le même package que le Front Controller. Leur fonctionnalité n'est pas du tout la même.

- Créez un package nommé « controllers » devant contenir les contrôleurs et qui se trouvera sous src/main/java.

- Créez une JSP erreur.jsp au même endroit que la JSP index.jsp

- Créez une interface ICommand dans ce package

```
public interface ICommand
{
   public String execute(HttpServletRequest request, HttpServletResponse response)
         throws Exception;
 }
```

Chaque contrôleur aura donc une méthode execute(HttpServletRequest request, HttpServletResponse response) qui retournera au Front Controller la JSP à dispatcher.

- Créez et Codez un contrôleur PageAccueilController

```
public class PageAccueilController implements ICommand {

   public String execute(HttpServletRequest request, HttpServletResponse response)
   throws Exception
   {
        return  "index.jsp " ;
   }
}
```

© AFPA 2025

❖ Codez le Front Controller

- Créez un package nommé servlet, routeur ou frontController qui se trouvera sous src/main/java.

Vous pouvez prendre la servlet déjà crée ou bien en recréer une autre avec un nom plus explicite. Si c'est le cas, penser à modifier votre fichier web.xml. Dans cette servlet :

- Créez un attribut HashMap() ;

  Exemple : private Map commands=new HashMap();

- Créez une méthode public void init() dans ce Front Controller

- Ajoutez des enregistrements clés/valeurs à votre hashMap.

  - La clé correspondant à la valeur du paramètre reçu de l'URL
  - La valeur sera l'instanciation de l'objet d'une classe contrôleur

  Exemple : commands.put("page1", new PageAccueilController()) ;

- Dans votre méthode processRequest, codez un try/catch approprié et y inclure

  String cmd=request.getParameter("cmd"); // cmd correspond au nom du paramètre passé avec l'url

  ICommand com=(ICommand)commands.get(cmd); // on récupère l'objet de la classe du contrôleur voulu

  urlSuite=com.execute(request, response); // on récupère dans urlSuite la JSP à dispatcher en exécutant le contrôleur appelé par l'URL

  - Dans le catch, mettez « erreur.jsp » dans urlSuite
  - Dans le finally du try/catch, lui-même dans un try/catch :

    request.getRequestDispatcher(urlSuite).forward(request, response);

❖ Testez votre application en ajoutant à votre url ?cmd=page1

- Rajoutez un enregistrement à votre hashMap en remplaçant « page1 » par null et retester votre application avec juste l'url sans paramètre.

Vous remarquerez qu'avec un Front Controller, la maintenance est largement simplifiée, ainsi que les évolutions futures. En ajoutant ou en enlevant des enregistrements à la hashMap, l'application évolue très simplement.

© AFPA 2025

❖ Structure de votre application

- Créez 4 contrôleurs : liste, création, modification et suppression des clients
- Créez les 4 JSP correspondantes avec des liens (href) pointant sur ces différentes JSP (vous pouvez créer une barre de navigation, comme la Navbar de Bootstrap par exemple)
- Idem pour les prospects
- Modifiez votre Front Controller en conséquence
- Testez votre application
- Ajouter dans votre fichier web.xml la gestion des erreurs 500 et 404

```
<error-page>
        <error-code>500</error-code>
        <location>/WEB-INF/JSP/erreur.jsp</location>
    </error-page>
    <error-page>
        <error-code>404</error-code>
        <location>/WEB-INF/JSP/erreur.jsp</location>
    </error-page>
```

© AFPA 2025

## 7. Checkstyle

Un linter est un outil d'analyse de code qui permet de détecter les erreurs et les problèmes de syntaxe.

https://checkstyle.sourceforge.io/

❖ Dépendance et pluging à rajouter dans votre fichier pom.xml

```
<dependency>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-checkstyle-plugin</artifactId>
        <version>3.1.2</version>
</dependency>


<plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-checkstyle-plugin</artifactId>
            <version>3.1.2</version>
            <dependencies>
               <dependency>
                  <groupId>com.puppycrawl.tools</groupId>
                  <artifactId>checkstyle</artifactId>
                  <version>9.3</version>
               </dependency>
            </dependencies>
</plugin>
```

❖ Pour lancer le linter :

mvn checkstyle:checkstyle

Le résultat est visible en lançant le fichier checkstyle.html se trouvant dans le dossier target/site

© AFPA 2025

## 8. La partie Modèle (JavaBean)

### a) Création du JavaBean

Vous allez créer un JavaBean

➔ https://fr.wikipedia.org/wiki/JavaBeans

Dans un premier temps, notre modèle comprendra 1 classe JavaBean

- Créez un package « models » (src/main/java/models)
- Créez une classe Client et copier le code de votre projet Swing

### b) La validation des données : l'API Bean Validation et le Bean Validation provider

L'API Bean Validation standardise la définition, la déclaration et la validation des contraintes sur les données du ou des beans

La validation des données se fera grâce aux annotations et grâce à un validateur composé d'une classe de type Validator fournie par l'API.

Le package de l'API Bean Validation est : javax.validation

L'implémentation de référence est celle proposée par Hibernate

- Vous aurez donc besoin de 2 dépendances Maven

  https://mvnrepository.com/artifact/org.hibernate/hibernate-validator

  Prendre la version 7.0.5.Final

  https://mvnrepository.com/artifact/org.hibernate.validator/hibernate-validator

  Prendre la version 7.0.5.Final

© AFPA 2025

### c) Les annotations des attributs des Beans

➔ https://jakarta.ee/specifications/bean-validation/3.0/apidocs/jakarta/validation/constraints/package-summary

Un exemple : https://www.jmdoudoux.fr/java/dej/chap-validation_donnees.htm

Vous pouvez voir dans la ressource ci-dessus les annotations des contraintes possibles.

- Si l'annotation est mise sur une classe ou une interface, c'est l'instance de la classe qui est testée par le ConstraintValidator
- Si elle est mise sur un attribut, c'est la valeur de l'attribut qui est testée
- Si elle est mise sur le getter de l'attribut, c'est la valeur du retour du getter qui est testée

Rq :

- Ne pas mettre une annotation en même temps sur l'attribut et sur le getter d'un attribut
- En cas d'héritage, les contraintes sont héritées de la classe mère. Elles peuvent être redéfinies dans la classe fille
- L'annotation @valid permet de valider un attribut qui serait un objet ou une collection d'objets. Si la validation de l'un des objets dépendants est invalide, toute la validation du bean échoue. De même, si la validation de l'un des objets de la collection est invalide, toute la validation du bean échoue.

❖ Ajouter à votre Bean Client les annotations pour valider vos attributs (non null, valeur mini et maxi, …) et enlever les tests de vos setters (throw)

### d) La validation des contraintes

Vous allez instancier un objet de l'interface Validator qui retourne les erreurs de validation dans une collection Set d'objets de type ConstraintViolation

➔ https://jakarta.ee/specifications/bean-validation/3.0/apidocs/jakarta/validation/validator

© AFPA 2025

L'instanciation de l'objet de la classe Validator s'effectue avec la fabrique de la classe ValidatorFactory et sa méthode getValidator() :

```
ValidatorFactory factory = Validation.buildDefaultValidatorFactory();
Validator validator = factory.getValidator();
```

Les erreurs sont ensuite ajoutées dans une collection typée de la classe de l'objet du Bean à valider :

```
Set<ConstraintViolation<Client>> violations = validator.validate(client);
```

Si la collection est vide, c'est que le Bean a été validé.

- Ajoutez dans vos contrôleurs de saisie et de modification des clients la validation des contraintes que vous avez ajouté dans votre Bean Client

- Testez la validation de votre Bean dans vos contrôleurs en instanciant des objets de la classe Client et en créant un log contenant la liste violations

- Vous pourrez par la suite afficher les erreurs dans vos JSP (chapitre 10)

© AFPA 2025

## 9. Echanges de données entre Servlets et JSP

L'interface HttpServletRequest :

➔ https://jakarta.ee/specifications/servlet/4.0/apidocs/javax/servlet/http/httpservletrequest

L'interface ServletRequest :

➔ https://jakarta.ee/specifications/platform/9/apidocs/jakarta/servlet/servletrequest

L'interface ServletRequest possède une méthode : getParameterMap() qui donne tous les paramètres de la requête.

Celle-ci a une méthode containsKey(" nomDeCLé ") pour savoir si un paramètre se trouve dans la requête
3 autres méthodes dont nous aurons besoin dans ce projet seront :

- setAttribute("nomDuParametre", valeurDuParametre) pour envoyer un attribut à la JSP
- getParameter("nomDuParametre") pour récupérer une valeur provenant d'une requête HTTP Get ou Post, comme une valeur provenant d'un formulaire
- getAttribute("nomDuParametre") pour récupérer une valeur provenant d'une requête HTTP Get ou Post, ou d'une variable de session (voir plus loin le chapitre session)

Remarque :

- Le retour de la méthode getAttribute est un objet
- Le retour de la méthode getParameter est un objet de la classe String. On ne pourra donc l'utiliser que si on est sûr de récupérer une chaîne de caractères

© AFPA 2025

Les différents scopes :

| Scope | Annotation | Duration |
|---|---|---|
| Request | @RequestScoped | A user's interaction with a web application in a single HTTP request. |
| Session | @SessionScoped | A user's interaction with a web application across multiple HTTP requests. |
| Application | @ApplicationScoped | Shared state across all users' interactions with a web application. |
| Dependent | @Dependent | The default scope if none is specified; it means that an object exists to serve exactly one client (bean) and has the same lifecycle as that client (bean). |
| Conversation | @ConversationScoped | A user's interaction with a servlet, including JavaServer Faces applications. The conversation scope exists within developer-controlled boundaries that extend it across multiple requests for long-running conversations. All long-running conversations are scoped to a particular HTTP servlet session and may not cross session boundaries. |

(cf : https://docs.oracle.com/javaee/7/tutorial/cdi-basic008.htm )

➔ https://jakarta.ee/specifications/platform/8/apidocs/javax/enterprise/context/package-summary

Au sein d'une requête, nous utiliserons le scope « Request » de l'interface HttpServletRequest.

© AFPA 2025

## 10. Les JSP (Java Server Pages)

Les JSP, bien qu'elles n'en aient pas l'apparence, sont des servlets qui génèrent des pages Web.
Elles contiennent du code HTML et du code java qui va générer du code HTML.

Elles doivent avoir la structure d'une page HTML

Les JSP ont 3 cycles de vie, jspInit(), _jspService() et jspDestroy()
Le contenu d'une JSP est compilé et interprété à l'exécution.

Les JSP ont un scope particulier qui est le scope « Page ».

Quand on produit un WAR (.war), les assets publics (JSP, images, fichiers CSS et JavaScript) doivent être placés dans le dossier webapp qui se trouve à l'emplacement \src\main. Ils seront accessibles depuis la racine de l'application à l'exécution.

Regarder l'arborescence de votre projet, votre dossier WEB-INF se trouve dans le dossier webapp

Le dossier WEB-INF masque au visiteur ce qu'il contient. Les JSP ne pouvant pas être appelées de manière individuelle, nous les placerons désormais dans un sous dossier JSP du dossier WEB-INF.

❖ Déplacez vos JSP et Modifiez le dispatcher du Front Controller en mettant ("WEB-INF/JSP/" + urlSuite). Vérifiez en relançant votre application.

### a) Les JSP Scriptlets

Une Scriptlet de JSP est du code se trouvant dans une JSP et elle est utilisée pour contenir un fragment de code. La syntaxe est :

```
<%
        Fragment de code
%>
```

Les Scriptlets servent entre autres choses à insérer des directives dans la servlet. Les directives n'apparaissent pas dans la page HTML.

© AFPA 2025

Celle ci-dessous indique que la JSP est encodée en UTF-8

<%@page contentType="text/html" pageEncoding="UTF-8"%>

❖ Ajoutez la directive d'encodage ci-dessus dans vos JSP

Les directives sont aussi utilisées pour coder des inclusions qui vont permettre d'insérer dans une JSP d'autres pages JSP ou HTML comme par exemple des entêtes comme ci-dessous ou des pieds de page.

<%@ include file="Header.jsp" %>

❖ Créez un header commun à vos JSP, comme une Navbar Bootstrap et l'inclure dans toutes vos JSP. Il faudra aussi peut être créer un footer si vous avez à inclure des fichiers JavaScript

Les sriptlets peuvent aussi contenir des blocs de code Java compris entre les balises <% et %>

⚠ C'est une mauvaise pratique. Du code java inclus dans une scriptlet rend une JSP difficilement lisible et très peu maintenable

Nous utiliserons les Expression Language (EL) pour insérer du code java (voir chapitre ci-dessous)

### b) Les Expressions Languages (EL)

➔ https://jakarta.ee/specifications/expression-language/4.0/jakarta-expression-language-spec-4.0

Les Expressions Languages (EL) permettent le traitement sur des données au sein d'une JSP, comme des calculs, ou la lecture des données provenant d'un contrôleur (avec la méthode setAttribute de l'interface HttpServletRequest)

Syntaxe :

${ expression } l'expression est évaluée avant l'envoi de la JSP

Exemple :

© AFPA 2025

${ empty name ? "" : name }  si name est vide ( ?) , on n'affiche rien sinon (:) on affiche la valeur de name

Les EL permettent également d'afficher des tableaux, Map, Array et d'accéder à certains éléments de ceux-ci {nomTableau[indice]}

❖ Dans un de vos contrôleurs, envoyez à la JSP un attribut avec la méthode setAttribute de l'interface HttpServletRequest, contenant une variable à laquelle vous aurez préalablement donné une valeur et affichez-le. Essayez également de faire des tests sur cet attribut dans la JSP :

Dans le contrôleur :
request.setAttribute("variable","Hello" ));

Dans la JSP :
${variable}

Cela nous servira quand nous allons récupérer des données d'une base de données ou bien quand nous voudrons la mettre à jour

### c) JSTL (JavaServer Standard Tag Library)

C'est une bibliothèque contenant des tags et qui propose des fonctionnalités pour développer des JSP.

La **JSTL** permet de développer des pages JSP en utilisant des balises XML, donc avec une syntaxe proche des langages utilisés par les web designers. Cela leur permet de concevoir des pages dynamiques complexes sans connaissances du langage Java.

Sa syntaxe est facile à lire et à réutiliser, et permet ainsi aux développeurs de coder des JSP maintenables.

Elle comporte 5 sous bibliothèques

- Core (variables, conditions boucles)
- Format (format les données, internationalisation d'un site)
- XML (manipule des données XML)
- SQL (permet d'écrire des requêtes SQL) ⚠ mauvaise pratique
- Function (traiter des chaines de caractères)

© AFPA 2025

La sous bibliothèque que nous aborderons sera la Core

❖ Ajoutez ces dépendances dans votre fichier pom.xml
https://mvnrepository.com/artifact/jakarta.servlet.jsp.jstl/jakarta.servlet.jsp.jstl-api

Prendre la version 3.0.0

https://mvnrepository.com/artifact/org.glassfish.web/jakarta.servlet.jsp.jstl

Prendre la version 3.0.0

Avantages de la JSTL :

- Echappe les caractères spéciaux XML et HTML => évite la faille XSS
- Les balises HTML ne sont pas interprétées
- Les variables EL sont protégées également de la faille XSS
- On a des variables à l'intérieur de la JSP

Inclusion de la bibliothèque JSTL dans les JSP :

<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

La JSTL sera chargé dans la JSP avec le préfixe c (pour Core)

Pour éviter d'avoir à rajouter à toutes les JSP la directive d'encoding et l'inclusion de la JSTL, vous pouvez créer un fichier taglibs.jsp et donner l'inclusion dans le fichier web.xml pour l'inclure dans toutes les JSP

❖ Créez un fichier taglibs.jsp

```
<%@ page pageEncoding="UTF-8" %>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@page contentType="text/html"%>
```

❖ Modifiez le fichier web.xml :

```
<jsp-config>
    <jsp-property-group>
        <url-pattern>*.jsp</url-pattern>
        <include-prelude>/WEB-INF/JSP/taglibs.jsp</include-prelude>
    </jsp-property-group>
</jsp-config>
```

© AFPA 2025

➔ https://jakarta.ee/specifications/tags/1.2/tagdocs/c/tld-summary.html

**Exemples d'utilisation**

<c:set var="variable" value="val" scope="page" />

Ou <c:set var="variable" scope="page">val</c :set >

<c:out value= "${ variable } " >valeur par défaut</c:out>

<c:remove var='variable' scope="page" />

Modifier un bean :

<c:set target="${ personne }" property="prenom" value="val" />

<c:out value="${ auteur.prenom }" />

**Les conditions**

```
<c:if test="${ 50 > 10 }" var="variable" scope= 'session'>
   C'est vrai !
</c:if>
```

Le résultat du test sera enregistré dans la variable 'variable' par défaut de portée de page

Le else n'existe pas, donc utiliser :

```
<c:choose>
   <c:when test="${ variable }">Du texte</c:when>
   <c:when test="${ autreVariable }">Du texte</c:when>
   <c:when test="${ encoreUneAutreVariable }">Du texte</c:when>
   <c:otherwise></c:otherwise>
</c:choose>
```

**Les boucles**

```
<c:forEach var="i" begin="0" end="10" step="2">
   <p>Un message n°<c:out value="${ i }" /> !</p>
</c:forEach>

<c:forEach var="personne" begin="0" items="${personnes}" varStatus= "status">
        <c:out value="${ status.count } » />
 </c:forEach> // on peut avoir aussi end="valeur"
```

© AFPA 2025

La variable status contient le statut de la boucle
Attributs : count, index (commence à 0), current (élément parcouru), first (premier élément), last

**Découpage d'une chaine de caractères**

```
<c: forTokens var= "variable" items="chaine/ de/ caractère" delims= "/">
        <p>${variable}</p>
</c:forTokens>
```

**Affichage des erreurs de saisie (validation du bean)**

- Dans le contrôleur :
  ```
  for (ConstraintViolation<Personne> violation : violations) {
                  message = message + violation.getMessage() + "<br> ";
  }
  ```

- Dans la JSP :

  ```
  <c:if test="${!empty message}" >
          <div id="message">
                  ${message}
          </div>
  </c:if>
  ```

❖ Affichez les clients

✓ Option 1 :
- Créez une ArrayList dans votre contrôleur dédié à afficher la liste des clients
- Instanciez 2-3 clients et ajoutez-les à votre ArrayList

✓ Option 2 :
- Allez dans la ressource « Java EE 8 persistance et authentification » et mettez en place la Data Source pour pouvoir utiliser vos classes DAO
- Accédez à la méthode findAll et créez une ArrayList

- Envoyez cette ArrayList en paramètre à votre JSP
- Affichez la liste des clients si celle-ci n'est pas vide ainsi que le nombre de clients, sinon, afficher un message indiquant que la liste est vide
  (Testez avec une liste vide)

© AFPA 2025

### d) Les formulaires

Rappels :

- La méthode doPost ou doGet est appelée au submit du formulaire.
- Action= L'URI du programme qui traitera les informations soumises par le formulaire
- Les valeurs saisies dans le formulaire sont récupérées grâce à l'attribut « name »

❖ Créez un client

- Créez, dans la JSP de création, un formulaire de saisie et récupérez les valeurs dans le contrôleur grâce à la méthode getParameter.
- Instanciez un objet de la classe Client avec ces valeurs.
- Ajoutez le client dans l'ArrayList (option 1) ou la BDD (option2)

❖ Modifiez un client

- Copiez dans le contrôleur de modification des clients l'ArrayList créée précédemment (option1) ou accédez à la méthode findAll et créez une ArrayList (option 2)
- Envoyez cette ArrayList en paramètre à votre JSP
- Dans la JSP, la récupérer dans un select.
- Récupérez dans le contrôleur le client sélectionné
- Renvoyez à la JSP les valeurs de le client à modifier (setAttribute)
- Récupérez les valeurs dans le contrôleur grâce à request.getParameter
- Modifiez le client dans l'ArrayList (option 1) ou la BDD (option2)

❖ Supprimez un client

- Copiez dans le contrôleur de suppression des clients l'ArrayList créée précédemment (option1) ou accédez à la méthode findAll et créez une ArrayList (option 2)
- Envoyez cette ArrayList en paramètre à votre JSP
- Dans la JSP, la récupérer dans un select.
- Récupérez dans le contrôleur le client sélectionné
- Renvoyez à la JSP les valeurs du client pour confirmation (setAttribute)
- Au retour de formulaire, supprimez le client dans l'ArrayList option 1) ou la BDD (option2)

© AFPA 2025

**Conseils :**

- Vous saurez à quelle étape vous vous trouvez dans votre contrôleur en testant quel ou quels paramètres vous recevez grâce à la méthode getParameterMap().containsKey.
- De même pour l'affichage dans la JSP, vous devrez tester quelle ou quelles variables vous recevez.

❖ Vous vous êtes aperçu que les 2 JSP de création et de modification comportent de nombreuses lignes en commun.
Vous pouvez supprimer la JSP de création et modifier votre contrôleur d'ajout pour qu'il « pointe » vers la JSP de modification. Ajoutez à la JSP de modification un libellé pour indiquer à l'utilisateur s'il est en création ou en modification, information donnée par le contrôleur correspondant.

## 11. Les sessions

Une session utilisateur permet de suivre un visiteur tout au long de sa navigation sur le site web.

La session est définie pour l'adresse IP de l'ordinateur et pour le navigateur.

Un artéfact se trouve sur le navigateur (Cookies). Il s'appelle jSESSIONID et il est envoyé via la requête HTTP au serveur qui reconnaîtra alors l'utilisateur.

On doit enregistrer des informations de l'utilisateur dans des variables de session car la norme HTTP ne permet pas de conserver des informations d'une page à l'autre.

La session utilisateur se termine quand il ferme l'onglet du navigateur ou quand la session est inactive depuis trop longtemps

❖ Paramétrez votre fichier web.xml avec un temps de suppression d'une session au bout de 30 minutes.

```
<session-config>
  <session-timeout>
    30
  </session-timeout>
</session-config>
```

© AFPA 2025

➢ Dans les JSP, on peut directement récupérer les variables de session : l'objet implicite de EL est : sessionScope

sessionScope.nomVariable

➢ Dans les contrôleurs

- Pour initier une session :

  HttpSession session = request.getSession();

- Pour créer une variable de session et lui attribuer une valeur :

  session.setAttribute("nomDeLaVariable", valeurDeLaVariable)

- Pour récupérer une valeur d'une variable de session

  session.getAttribute("nomDeLaVariable ")

- Pour supprimer une session :

  session.invalidate() :

❖ Ajoutez un compteur de pages lues dans votre projet avec une variable de session.

- Créer la variable de session dans le Front Controller

```
if (session.getAttribute("compteurPage") == null) {
        session.setAttribute("compteurPage", 0);
}
```

- Incrémenter ce compteur à chaque appel de page

© AFPA 2025

## 12. Les cookies

Les cookies sont des données stockées sur le navigateur du client.

Cookie cookie = new Cookie("variable", variable);

cookie.setMaxAge(60*60*24);  // durée de la persistance du cookie sur le navigateur du client, en secondes

response.addCookie(cookie);

- Pour voir les cookies présentes sur le navigateur pour une application :

Exemple pour Chrome : Outils de développement => Application =>cookies

- Pour récupérer les cookies :

Cookie[] cookies = request.getCookies() ;

Rq : On n'a pas accès aux cookies d'autres sites

```
If (cookies != null) {
        for (Cookie cookie : cookies) {
                If (cookie.getName().equals('variable')) {
                        Todo  cookie.getValue() ;
                }
        }
}
```

❖ Créez un cookie et vérifier sur le navigateur qu'il a bien été enregistré. L'afficher dans une autre page de votre application.

© AFPA 2025

## 13. NPM

Se mettre au niveau du dossier **webapp** pour faire le NPM init (tout ce qui est public).

Dans la balise <head> ou dans header.jsp

```
<link href="<c:url value="/node_modules/bootstrap/dist/css/bootstrap.min.css"/>"
    rel="stylesheet" type="text/css"/>
```

A la fin de la balise <body> ou dans footer.jsp

```
<script src='<c:url value="/node_modules/bootstrap/dist/js/bootstrap.bundle.js"
/>'></script>
```

© AFPA 2025

# Java EE 8

**DATE DE MISE A JOUR**

03/2025

**© AFPA 2025**

**Reproduction interdite**

Article L 122-4 du code de la propriété intellectuelle

« Toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droits ou ayants cause est illicite. Il en est de même pour la traduction, l'adaptation ou la transformation, l'arrangement ou la reproduction par un art ou un procédé quelconque ».

**Agence nationale pour la Formation Professionnelle des Adultes**
3 rue Franklin – 93100 Montreuil

www.afpa.fr