



MODULE JAVA

Java - JEE



1.

Introduction



Introduction

- ▶ **Java SE** : Standard Edition → packages de base (java.lang, java.io, java.math, java.util, java.sql, ...)
- ▶ **Java EE** : Entreprise Edition → extension de la plateforme standard
 - ▶ **Objectif** : Faciliter le développement d'application web déployées et exécutées sur un **serveur d'application**

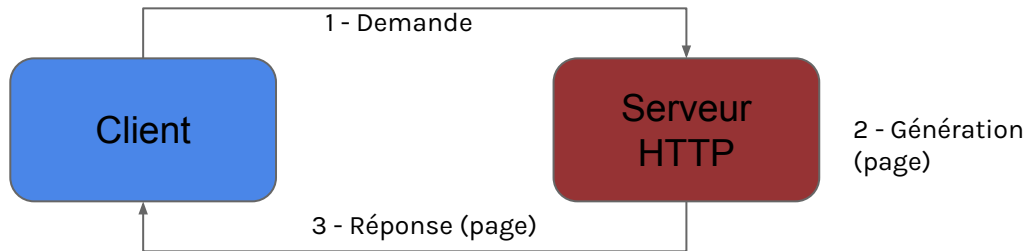


Introduction

- ▶ **Java EE** : Exécutable avec une JRE mais nécessite en complément des librairies
- ▶ Depuis 2018, on parle de **Jakarta EE**
 - ▷ Oracle a confié la maintenance à Apache

Introduction

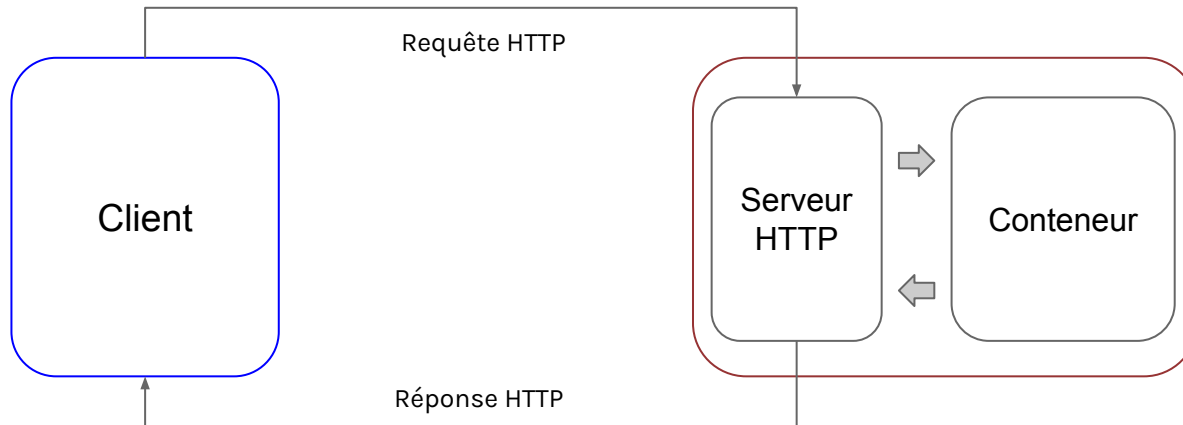
- **Echanges HTTP** : Entre un client et un serveur
 - ▷ Client → Navigateur
 - ▷ Serveur → Serveur HTTP



- Pas suffisant pour JEE car le serveur doit pouvoir effectuer d'autres tâches en dehors de l'affichage de pages !

Introduction

- **Echanges HTTP pour une application WEB** : Entre un client et un serveur
 - ▷ Client → Navigateur
 - ▷ Serveur → **Serveur d'application**





1.2.

ENVIRONNEMENT DE TRAVAIL



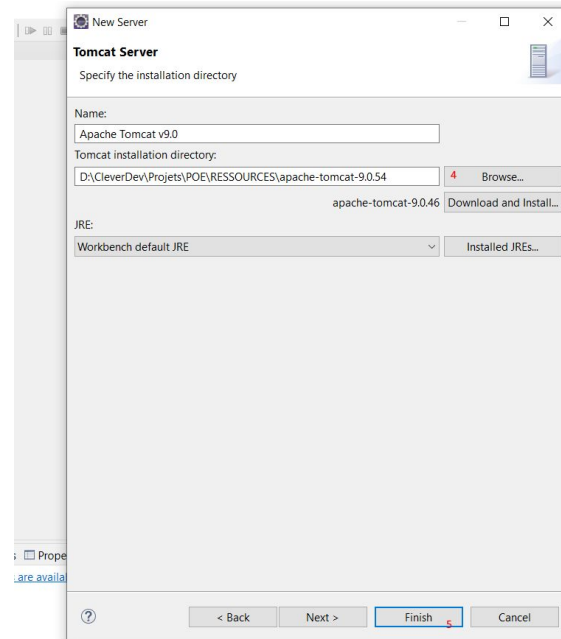
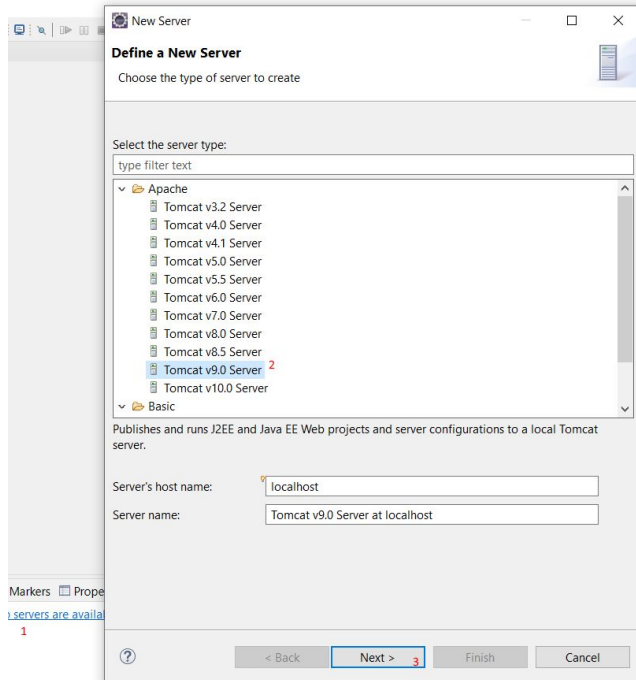
Environnement de développement

- ▶ Eclipse
 - ▷ Eclipse.org
 - ▷ Eclipse IDE for Java EE Developers
 - ▷ Config « encoding » : utf8
 - ▷ <https://www.eclipse.org/downloads/>

- ▶ Apache Tomcat
 - ▷ tomcat.apache.org
 - ▷ <http://localhost:8080/>
 - ▷ <https://tomcat.apache.org/download-90.cgi>

Environnement de développement

Apache Tomcat





Environnement de développement

Premier projet Java EE / Eclipse

□ Eclipse

- « Dynamic Web Project »
- « New Runtime »
 - Tomcat 8 + spécifier le chemin d'accès
- Organisation projet :
 - Java Ressources / src → (sources java)
 - WebContent → (html, css, png, jsp)
 - WebContent/WEB-INF → (config)
 - WebContent/WEB-INF/lib → (bibliothèque externes)
 - *Remarque : WEB-INF est un répertoire « protégé » non accessible en http.*

□ Première page statique

- index.html (dans WebContent)
- Run as... « Run On Server » (TomCat ne doit pas être déjà activé)



Exemple 1 : Première Servlet

- Classe dérivant de HttpServlet
 - dans src / de préférence dans un package (`com.formation.servlet.First.java`)

- Fichier web.xml (dans WEB-INF)
 - `<web-app>`
 - `<servlet>` // description de la servlet
 - `<servlet-name>` → Nom
 - `<servlet-class>` → Classe java correspondante
 - `<servlet-mapping>` // association avec une URL
 - `<servlet-name>` → Nom
 - `<url-pattern>` → Adresse (URL)

Exemple 1 : com.cleverdev.servlet.First.java

```
package com.formation.servlet;
// imports...

@WebServlet("/First")

public class First extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public First() {
        super();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        response.setContentType("text/html");
        response.setCharacterEncoding("UTF-8");
        java.io.PrintWriter out = response.getWriter();
        out.println("Bonjour Servlet !");
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException { doGet(request,
        response);
    }
}
```



Exemple 1 : WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"
  version="3.0">

  <servlet>
    <servlet-name>First</servlet-name>
    <servlet-class>com.formation.servlet.First</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>First</servlet-name>
    <url-pattern>/bonjour</url-pattern>
  </servlet-mapping>

</web-app>
```

L'URL <http://localhost:8080/monprojet/bonjour> sera traitée par la Servlet **com.formation.servlet.First**

Exemple 1 : com.cleverdev.servlet.First.java

```
package com.formation.servlet;
// imports...

@WebServlet("/First")

public class First extends HttpServlet {

    // ...


    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        response.setContentType("text/html");
        response.setCharacterEncoding("UTF-8");
        java.io.PrintWriter out = response.getWriter();

        out.println("<!DOCTYPE html>"); out.println("<html>"); out.println("<head>");
        out.println("<meta charset=\"utf-8\" />"); out.println("<title>First servlet</title>");
        out.println("</head>"); out.println("<body>");
        out.println("<p>Bonjour Servlet en HTML !</p>"); out.println("</body>");
        out.println("</html>");
    }

    // ...
}
```

Création d'une page HTML "valide"



Même si la servlet peut tout gérer en java, ce n'est pas cette méthode que l'on utilisera et la partie "présentation" sera confiée à une Vue (utilisant le langage JSP).

Exemple 1 : WEB-INF/bonjour.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Première vue JSP</title>
</head>
<body>

    <p>Bonjour JSP en HTML !</p>

</body>
</html>
```

```
package com.formation.servlet;
// ...

public class First extends HttpServlet {
    // ...
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        this.getRequestDispatcher("/WEB-INF/bonjour.jsp").forward(request, response);
    }
    // ...
}
```



2.

Le modèle MVC

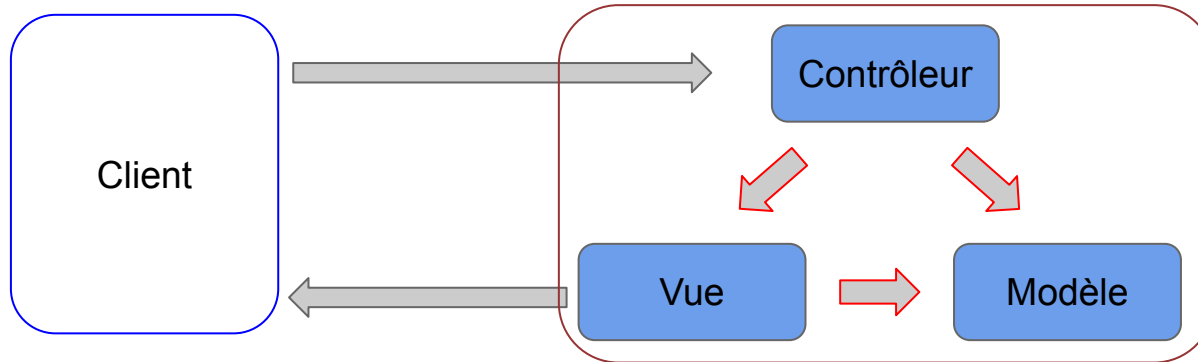


Le modèle **MVC**

- ▶ **MVC** = Modèle - Vue - Contrôleur (Model - View - Controller)
- ▶ **Design Pattern** : découper l'application en couches pour séparer les responsabilités
 - ▷ Traitement, stockage et mise à jour des données : **couche "Modèle"**
 - ▷ Interaction utilisateur et présentation : **couche "Vue"**
 - ▷ Contrôles des actions et des données : **couche "Contrôleur"**

Le modèle **MVC**

- ▶ **MVC** = Modèle - Vue - Contrôleur (Model - View - Controller)





Le modèle **MVC**

- ▶ **MVC** = Modèle - Vue - Contrôleur (Model - View - Controller)
- ▶ Exemple de mise en place du **MVC** sans framework :
 - ▶ **“Modèle”** : Traitements et données en **Java** (“beans” et “DAO”)
 - ▶ **“Vue”** : Pages **JSP** (Java Server Pages)
 - ▶ **“Contrôleur”** : **Servlets** Java

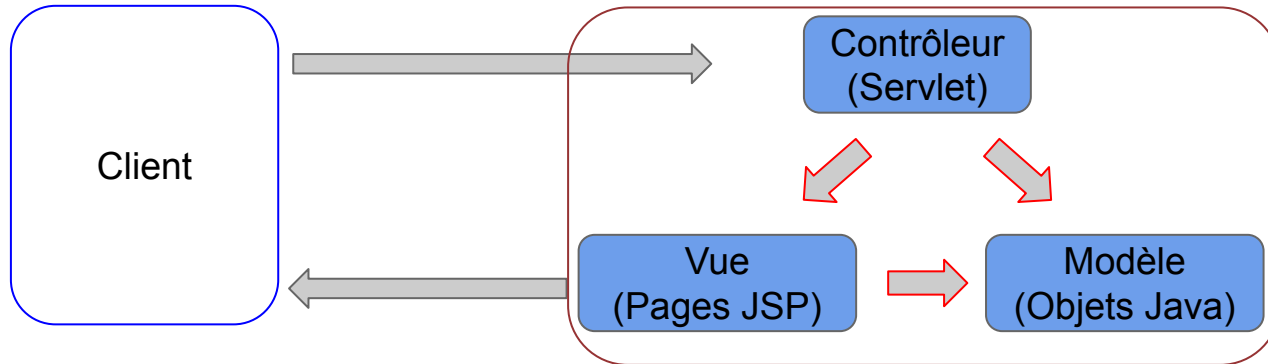


Le modèle **MVC**

- ▶ **MVC** = Modèle - Vue - Contrôleur (Model - View - Controller)
- ▶ **Servlet** : objet permettant d'intercepter les requêtes faites par un client et de gérer la réponse
- ▶
 - ▷ Méthodes pour scruter les requêtes HTTP
 - ▷
 - ▷ Fait appel aux traitements de la couche “Modèle”
 - ▷
 - ▷ Demande à la couche “Vue” de retourner le résultat au client

Le modèle **MVC**

- Exemple de mise en place du MVC sans framework :





3.

Servlet Java



Servlet Java

- ▶ Echanges HTTP : verbes GET et POST
- ▶ GET : pour récupérer une ressource via son URI
 - ▷ Possibilité de positionner des paramètres
 - ▷ Lorsque que le serveur reçoit une requête HTTP GET, il retourne la ressource demandée



Servlet Java

- ▶ Echanges HTTP : verbes GET et POST
- ▶ **POST** : pour soumettre des données aux serveurs
 - ▷ Pour modifier la ressource (création, modification, suppression)



Servlet Java

- ▶ Mécanisme sur le serveur d'application
 - ▷ Réception de la requête HTTP par le serveur HTTP
 - ▷ Transmission au conteneur de servlets
 - ▷ Transformation de la requête : objet `HttpServletRequest`
 - Contient la requête et donne accès aux informations de celle-ci (header et body)
 - ▷ Initialisation de la réponse : objet `HttpServletResponse`
 - Permet de personnaliser les informations de la réponse (header et body)



Servlet Java

- ▶ Servlet ?
 - ▷ Une servlet = une classe Java
 - ▷ Doit permettre le traitement de requêtes et la personnalisation de réponses
 - ▷ En synthèse : doit permettre de recevoir une requête HTTP envoyée par un client, et de retourner une réponse HTTP à ce client



Servlet Java

- ▶ Servlet

- ▷ Package **javax.servlet**

- ▷ Interface mère Servlet

- ▷ Package **javax.servlet.http**

- ▷ Classe HttpServlet → Classe abstraite

- ▷ Faire hériter nos classes "Servlet" de cette classe !



Servlet Java

► Servlet

- ▷ Redéfinition de méthodes dans nos classes Servlet :
 - ▷ doGet() → gestion des requêtes HTTP GET
 - ▷ doPost() → gestion des requêtes HTTP POST
- ▷ Utilisation de la bonne méthode ? rôle de la méthode **service()**
 - Exécution automatique
 - Lecture de l'objet HttpServletRequest et distribution de la requête HTTP à la bonne méthode doXXX() en fonction du type



Servlet Java

► Servlet

▷ Déclaration des Servlets

- ▷ Pour que l'application les connaisse et les expose !
- ▷ Nécessite un lien entre la **servlet** et une **URL**
- ▷ Permet de **diriger** la requête HTTP vers la bonne Servlet
- ▷ Déclaration dans un fichier : **web.xml**



Servlet Java

- ▶ Servlet
 - ▷ Déclaration des Servlets
 - ▷ Fichier **web.xml** positionné dans un répertoire **WEB-INF** sous `src/main/webapp`



Servlet Java

- ▶ Servlet

- ▷ Définition d'une Servlet

```
<servlet>  
    <servlet-name>UserServlet</servlet-name>  
    <servlet-class>fr.m2i.crm.servlet.UserServlet</servlet-class>  
</servlet>
```



Servlet Java

► Servlet

▷ Définition d'une Servlet - options

```
<description>Description de la Servlet</description>  
<init-param>  
    <param-name>auteur</param-name>  
    <param-value>Marc</param-value>  
</init-param>  
<load-on-startup>1</load-on-startup>
```




Servlet Java

- ▶ Servlet

- ▷ Mapping d'une Servlet

```
<servlet-mapping>  
    <servlet-name>UserServlet</servlet-name>  
    <url-pattern>/userServlet</url-pattern>  
</servlet-mapping>
```



Servlet Java

► Servlet

▷ Cycle de vie d'une Servlet

- ▷ Lors de la première sollicitation d'une Servlet ou lors du démarrage de l'application web, le conteneur de Servlets crée une instance de cette Servlet
- ▷ Elle est conservée en mémoire pendant toute la durée de vie de l'application
- ▷ La même instance de la Servlet est utilisée pour chaque requête entrante dont l'URL correspond au pattern d'URL défini pour la Servlet



4.

Servlet et Vue



Servlet et **Vue**

- ▶ Mise en place de la couche **vue** sans framework : **JSP**
 - ▷ JSP ?
 - ▷ Page qui contient des **balises** HTML et des **balises** JSP
 - ▷ Extension de fichier “.jsp”
 - ▷ Exécution **côté serveur** !



Servlet et **Vue**

- ▶ Mise en place de la couche **vue** sans framework : **JSP**
 - ▷ JSP : technologie de la plateforme Java EE
 - ▷ **Combine** les technologies HTML, XML, Servlet et JavaBeans pour créer des vues dynamiques



Servlet et **Vue**

- ▶ Mise en place de la couche **vue** sans framework : **JSP**
 - ▶ **Pourquoi ?**
 - ▷ **Simplifier** la partie présentation (trop lourd pour la Servlet)
 - ▷ **Séparation claire** entre le code de **contrôle** et la **présentation**, telle que recommandé par le pattern MVC



Servlet et **Vue**

- ▶ Cycle de vie d'une **JSP**
 - ▷ Une page JSP est traduite et compilée en Servlet (classe Java héritant de HttpServlet)
 - ▷ La Servlet ainsi générée est utilisée durant l'existence de l'application
 - ▷ Si la JSP est modifiée, la Servlet est générée et compilée de nouveau



Servlet et **Vue**

- ▶ Cycle de vie d'une **JSP**
 - ▶ **Astuce** : on peut trouver le code généré et compilé par le serveur d'application → avec Tomcat, répertoire “work”



Servlet et **Vue**

- ▶ Association **Servlet** / **JSP**
 - ▷ Servlets Java développées : **contrôleurs**
 - ▷ Servlets générées à partir de JSP : **vues**
 - ▷ **Association dans les Servlets Java développées pour indiquer la JSP en charge de la présentation**

```
this.getServletContext().getRequestDispatcher( "/ma_jsp.jsp").forward( request,  
response );
```



5.

Echange de données



Echange de **données**

- ▶ Données issues du serveur : **attributs**
 - ▷ Modification de la requête **par le serveur** pour y ajouter des attributs (méthode **setAttribute()** sur la requête)
 - ▷ Récupération possible dans la JSP
- ▶ Données issues du clients : **paramètres**
 - ▷ Récupération de paramètres **fournis par le client** (méthode **getParameter()** sur la requête)



6.

Technologie JSP

- ▶ Différentes manières de **coder** dans une JSP :
 - ▷ Code Java → **à éviter**
 - ▷ Balises pour la déclaration : `<%! %>` (déclaration de variables et méthodes)
 - ▷ Balises commentaires : `<%-- --%>`
 - ▷ Balises Scriptlet : `<% %>` (code Java)
 - ▷ Balise d'expression : `<%= %>` (affichage dans la présentation)
 - ▷ EL (“Expression Language”) → **à préférer**
 - ▷ `${ expression }`
 - ▷ Syntaxe simple

- ▶ Utilisation des **EL** :
 - ▷ Manipulation des beans Java :
 - ▷ `${monBean.attribut}`
 - ▷ “monBean” = nom du bean Java
 - ▷ “attribut” = attribut (propriété) du bean Java
 - ▷ → équivalent au code Java : `monBean.getAttribut()`



JSP

- ▶ Utilisation des **EL** :
 - ▷ Manipulation des beans Java :
 - ▷ `${monBean.attribut}`
 - ▷ Protection contre les valeurs "null" → pas d'affichage si "monBean" est "null" ou si la valeur de "attribut" est "null" (pas d'erreur)

- ▶ Utilisation des **EL** :
 - ▷ Manipulation des collections Java :
 - ▷ Exemple d'une liste Java :

```
List<String> customers = new ArrayList<>();  
customers.add(customer1);
```
 - ▷ En "EL", accès aux éléments de la liste : `${customers.get(i)}`
 - "i" représentant la position de l'élément (à partir de 0)



JSP

Exo 1 : Faire une servlet qui créer une liste de 10 fruits, et afficher la à l'aide d'un fichier JSP.

Exo 2 : Faire une servlet permettant de récupérer un paramètre isGood dans l'url (valeur : 0 ou 1).

Si isGood = 0, alors afficher “Ce n’est pas bon !”

Sinon afficher “C’est bon !”

Exemple requête : localhost:8080/xxxx/exo2?**isGood=0**



7.

JSTL

- ▶ **JSTL** : JSP Standard Tag Library
 - ▷ Ensemble de tags utilitaires
 - ▷ Afficher une expression et sécuriser les formulaires contre les failles XSS (affichage avec balise "c:out")
 - ▷ Gérer dynamiquement les différents liens et URL (redirection avec "balise c:url")
 - ▷ Import dans les pages (balise "c:import")
 - ▷ Condition à l'affichage du résultat de la validation (balise "c:if")
 - ▷ Parcours d'une collection (balise "c:foreach")
 - ▷ ...

- ▶ **JSTL** : JSP Standard Tag Library

- ▷ Pour utilisation :

- ▷ 1. Ajout de la librairie **jstl**

- ▷ Exemple sur projet Maven :

- ```
<dependency>
 <groupId>javax.servlet</groupId>
 <artifactId>jstl</artifactId>
 <version>1.2</version>
</dependency>
```

- ▷ 2. Déclaration **taglib** dans les fichiers JSP :

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



JSTL

- ▶ **JSTL** : JSP Standard Tag Library
 - ▷ Documentation : <https://javaee.github.io/jstl-api/>

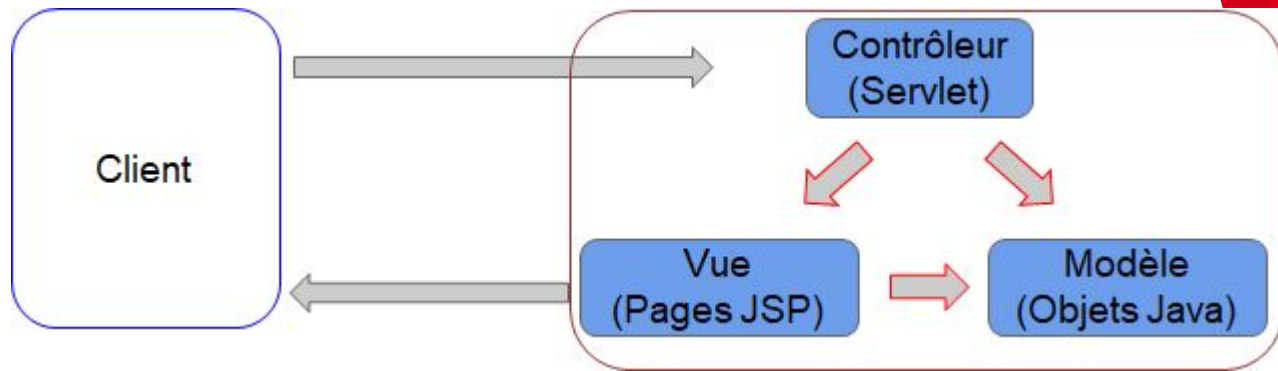


8.

Traitement de formulaire

Traitement de formulaire

- Rappel sur modèle MVC
 - La Servlet est un **contrôleur** chargé de diriger les requêtes vers les traitements correspondants
 - Ces **traitements** doivent être délégués à la couche « Modèle » du MVC





Traitement de formulaire

- ▶ Éléments nécessaires pour le traitement d'un formulaire
 - ▶ Un **objet** (bean Java) pour **stocker** les données validées du formulaire
 - ▶ Un **objet** « Métier » pour **recupérer et valider** les données du formulaire à fait partie de la couche « **Modèle** » du MVC
 - ▶ Chargé également d'indiquer les erreurs
 - ▶ Une **Servlet** capable de rediriger vers la vue formulaire en cas d'erreur pour afficher celle(s)-ci



9.

Gestion de session



Gestion de session

- ▶ Pourquoi ?
 - ▷ HTTP = protocole **sans état** (“stateless”)
 - ▷ Par défaut, **pas de lien** entre les requêtes d’un même client !
 - ▷ La session a pour but de **mémoriser** des informations pour un client donné (informations récupérables sur les requêtes faites par ce client)



Gestion de session

- ▶ Session en JEE
 - ▶ **Session** = espace mémoire alloué pour un client
 - ▶ **Fermeture** d'une session :
 - ▷ Au bout d'un certain temps d'inactivité
 - ▷ Sur déconnexion
 - ▷ ...
 - ▶ Objet Java **HttpSession** récupérable depuis la requête



Gestion de session

- ▶ Dans la **Servlet** (ou bean Java ayant accès la requête) :
 - ▷ Récupération de la session :
HttpSession session = request.getSession();
 - ▷ Récupération d'un attribut dans la session :
session.getAttribute("nomAttribut");
 - ▷ Positionnement d'un attribut dans la session :
session.setAttribute("nomAttribut", valeur);



Gestion de session

- ▶ Dans la **JSP** :
 - ▷ Récupération de la session dans une expression « EL » :
sessionScope.nomAttribut

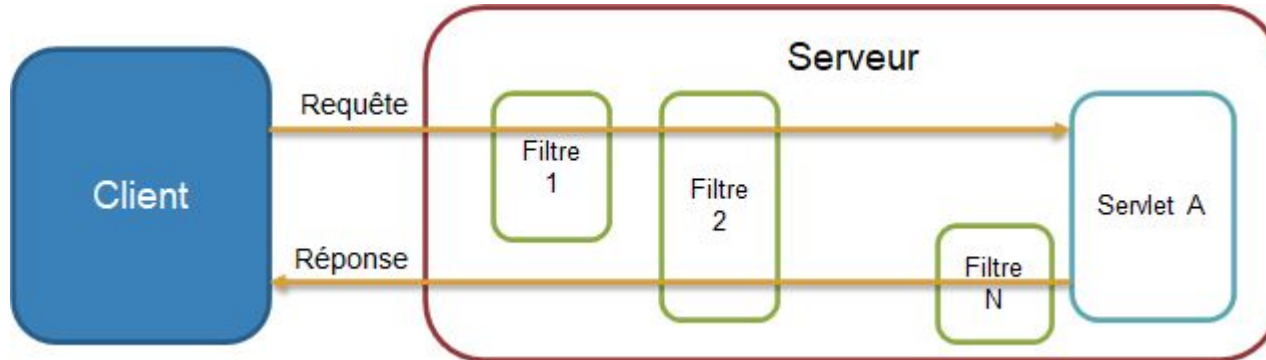


10.

Gestion des filtres et des listeners

► Filtre ? :

- Permettre de prétraiter ou post-traiter les données d'une requête
- Possibilité de chaîner des filtres





Filtre

- Mise en œuvre :

- Déclaration dans fichier **web.xml**

- Définition

```
<filter>  
    <filter-name>LoadingFilter</filter-name>  
    <filter-class>fr.m2i.crm.filter.LoadingFilter</filter-class>  
</filter>
```




Filtre

► Mise en œuvre :

► Déclaration dans fichier **web.xml**

► Mapping (URL)

```
<filter-mapping>  
  <filter-name>LoadingFilter</filter-name>  
  <url-pattern>/*</url-pattern>  
</filter-mapping>
```

Le filtre va s'appliquer sur les requêtes effectuées pour le pattern d'URL indiqué :

*/** à toutes les URL
/createCustomer à l'URL *createCustomer*



Filtre

► Mise en œuvre :

► Classe Java

- **Implémentation** de *javax.servlet.Filter*
- Méthode **init()** : exécutée à l'**initialisation** du filtre par le serveur d'application
- Méthode **doFilter()** : exécutée à chaque fois qu'une requête **arrive** pour l'URL du filtre
- Méthode **destroy()** : exécutée lors de la **destruction** de l'objet par le serveur d'application



Filtre

- Mise en œuvre :

- Classe Java

- Méthode `doFilter()` : la requête est **interceptée** par le listener !

- Il faut donc propager cette requête en fin de méthode (chaînage)

- `chain.doFilter(request, res);`

« chain » est de type `FilterChain`, est fourni en paramètre de la méthode `doFilter()`

L'appel de `doFilter` sur l'objet « chain » déclenche le transfert vers le suivant dans la chaîne (filtre suivant ou Servlet cible)



Listener

► Listener ?

- Objet **notifié** lors d'une modification de son environnement
- Design Pattern « **Observateur** »
- Objectif en JEE : avertir l'application d'un **changement**
- Différentes interfaces de listener
 - Changement d'état du ServletContext
 - Changement d'état des attributs stockés dans le ServletContext
 - Entrée/Sortie d'une requête
 - Changement d'état des attributs stockés dans un HttpServletRequest
 -



Listener

► Mise en œuvre

► Déclaration dans fichier **web.xml**

► Définition

```
<listener>  
    <listener-class>fr.m2i.crm.config.MonListener</listener-class>  
</listener>
```



Listener

► Mise en œuvre

- Classe Java à exemple pour listener sur changement d'état du ServletContext

- **Implémentation** de *javax.servlet.ServletContextListener*

- Méthode **contextInitialized()** : exécutée au **lancement de l'application**, avant le chargement des Servlets et des filtres à on y positionne ce qu'on souhaite faire

- Méthode **contextDestroyed()** : exécutée à la destruction du contexte d'exécution



11.

Projet fil rouge CRM

Mise en place de JEE avec
Servlet et JSP



Projet Fil Rouge

► Etape 1 :

- Mise en place des premières **Servlets** pour traitement des requêtes **HTTP GET**, avec traitement de validation des données
- Mise en place des premières **Vues** (pages JSP)
 - ▷ Création d'un client (formulaire)
 - ▷ Affichage d'un client
 - ▷ Création d'une commande (formulaire)
 - ▷ Affichage d'une commande
- Récupération des **paramètres** de la requête issue du formulaire dans la Servlet pour **contrôle** et **affichage** d'un résultat



Projet Fil Rouge

► Etape 1 :

- Limites de cette étape 1 :
 - ▷ Comportement figé
 - ▷ Duplication de code
 - ▷ Pas d'affichage conditionnel



Projet Fil Rouge

► Etape 2 :

- Utilisation de la librairie **JSTL**
- **Factorisation** de code (formulaire de création d'un client)
- **Génération dynamique des pages Vues** (pages JSP) avec utilisation des balises JSTL
 - ▷ Ajout d'un menu pour accéder aux formulaires de création
 - ▷ Affichage conditionnel des résultats en fonction des valeurs des attributs présent dans la requête



Projet Fil Rouge

► Etape 2 :

- Limites de cette étape 2 :
 - ▷ L'accès direct aux pages JSP est possible (pas de masquage de la technologie utilisée)
 - ▷ Pas de normalisation HTTP (seules les requêtes GET sont traitées)
 - ▷ Les traitements de validation ne sont pas en dehors des Servlets



Projet Fil Rouge

► Etape 3 :

- Masquage des JSP : positionnement dans le répertoire **WEB-INF** pour qu'elles ne soient pas accessibles directement par un utilisateur
 - ▷ Modification des servlets pour que la méthode doGet gère l'affichage de la vue
- Envoi des formulaires avec le verbe http **POST**
 - ▷ Modification des fichiers JSP
 - ▷ Modification des servlets pour implémenter une méthode doPost()



Projet Fil Rouge

► Etape 3 :

- **Extraction** des traitements de validation dans des classes **métiers dédiées**
 - ▷ Appel depuis les servlets
 - ▷ Gestion des erreurs
- Traitement de la **redirection** depuis les Servlets
 - ▷ En fonction du résultat du traitement, redirection vers le formulaire (en cas d'erreur) ou vers la page d'affichage
 - ▷ Modification des pages JSP pour gérer l'affichage des erreurs et des valeurs saisies sur les formulaires ré-affichés



Projet Fil Rouge

► Etape 3 :

- Limites de cette étape 3 :
 - ▷ Pas de conservation et donc pas d'accès aux clients et commandes créées
 - ▷ Pas de gestion de session (pour conserver des informations d'une requête à l'autre)



Projet Fil Rouge

► Etape 4 :

- Gestion de la **session**
 - ▷ Enregistrement dans la session des clients et commandes créées
 - ▷ Affichage de la liste des clients et des commandes présents en session --> nouvelles Servlets et vues JSP
 - ▷ Suppression d'un client ou d'une commande de la session à nouvelles Servlets



Projet Fil Rouge

► Etape 4 :

- Limites de cette étape 4 :
 - ▷ Pas de gestion spécifique des exceptions de validation
 - ▷ Pas de lien avec la base de données
 - ▷ Pas de chargement des données
 - ▷ Pas de persistance des ajouts/suppressions



Projet Fil Rouge

► Etape 5 :

- **Spécialisation des exceptions** de validation de formulaires **POST**
 - ▷ Modification des classes métiers
- **Lien** avec la couche d'accès aux **données**
 - ▷ Persistance des données
 - ▷ Stockage de la DaoFactory au niveau du contexte applicatif (ServletContext) avec un **listener**
- **Chargement** des données en session au **démarrage** de l'application (**Filtre**)



MODULE JAVA

Java - Java Server Faces



1.

Frameworks MVC



Frameworks MVC

- ▶ **Objectif** : simplifier la mise en oeuvre du modèle MVC sur son application
- ▶ Rappel MVC :
 - ▷ **Modèle de conception** pour organiser le code source d'une application Web
 - ▷ Découpage clair des **responsabilités**
 - ▷ Découpage simplifiée du développement des fonctionnalités
 - ▷ Meilleure maintenabilité



Frameworks **MVC**

- ▶ **Framework MVC** = surcouche aux Servlets
- ▶ Inversion de **contrôle** :
 - ▷ Le framework prend la main sur le contrôle de l'application



Frameworks **MVC**

► **Avantages**

- **Simplification** du flot d'exécution
 - ▷ Moins de code
 - ▷ Prise en compte native de tâches génériques
 - ▷ Organisation du code à respecter



Frameworks MVC

- ▶ Différents type de frameworks MVC
 - ▷ Basé sur les **requêtes**
 - ▷ Basé sur les **composants**



Frameworks MVC

- ▶ **Différents type de frameworks MVC**
 - ▶ **Basé sur les requêtes**
 - ▶ Le plus simple à comprendre
 - ▶ Agit sur le cycle de vie d'une requête (actions)



Frameworks MVC

► Différents type de frameworks MVC

► Basé sur les requêtes

- ▷ Prend en entrée une requête et retourne une réponse
 - Flot d'exécution linéaire
 - Orienté "actions"
- ▷ Utilisation d'une **Servlet unique** pour la redirection
 - Délègue les actions et traitements à la couche "Modèle"



Frameworks MVC

- ▶ **Différents type de frameworks MVC**
 - ▶ **Basé sur les requêtes**
 - ▶ Exemples de frameworks :
 - Struts
 - Stripes



Frameworks MVC

- ▶ Différents type de frameworks MVC
 - ▶ Basé sur les composants
 - ▶ Découpage du code en composants
 - ▶ Masquage des chemins des requêtes



Frameworks MVC

- ▶ **Différents type de frameworks MVC**
 - ▶ **Basé sur les composants**
 - ▶ Utilisation d'une **Servlet unique**
 - ▶ Développement des actions métiers **uniquement**
 - ▶ Création des vues avec un langage **spécifique**



Frameworks MVC

- ▶ **Différents type de frameworks MVC**
 - ▷ **Basé sur les composants**
 - ▷ Exemples de frameworks :
 - JSF (Java Server Faces) → adopté par le standard JEE
 - Wicket



Frameworks MVC

► Différents type de frameworks MVC

► Comparatif

- Basé sur les **requêtes**
 - Plus de code à produire
 - Plus de contrôle sur le cycle de vie des requêtes
- Basé sur les **composants**
 - Moins de code à produire
 - Moins de possibilité de contrôle sur le cycle de vie des requêtes et le rendu



2.

Java Server Faces



Framework JSF

- ▶ **Framework MVC basé sur les composants**
 - ▶ **Spécification** construite sur l'**API Servlet**
 - ▶ Fournit des **composants** sous forme de bibliothèques de balises
 - ▶ Utilisables dans les fichiers **JSP** ou d'autres technologies de restitution (par exemple : **Facelet** → **utilisée par défaut**)
 - ▶ Permet de créer des **templates**



Framework JSF

- ▶ **Template ?**
 - ▶ **Découpage** des pages/vues en plusieurs composants
 - ▶ **Assemblage** de ces composants pour obtenir une vue finale



Framework JSF

- ▶ **Principes de JSF**
 - ▶ **Contrôleur unique** (Servlet)
 - ▷ Gère le cycle de vie des requêtes
 - ▷ Nommée la “FacesServlet”
 - ▷ → Pattern “Front Controller”
 - ▶ **Fonctionnalités natives**
 - ▷ Récupération des paramètres des requêtes
 - ▷ Conversions
 - ▷ Validations
 - ▷ Génération des réponses
 - ▷ ...



Framework JSF

- ▶ **Principes de JSF**
 - ▶ **On a besoin de :**
 - ▷ Une **vue** : page JSP ou page “Facelet”
 - ▷ Un **modèle** : bean Java
 - ▶ **JSF** s’occupe de **lier** la vue avec le modèle
 - ▶ La **FacesServlet** gère **le traitement des requêtes** qui arrivent sur l’application
 - ▷ Plus besoin de coder les Servlets !!!



Framework JSF

- ▶ **Organisation du code sur une application avec JSF**
 - ▶ **Vues :**
 - ▷ Pages JSP (utilisation **dépréciée**) ou pages “**Facelet**” (XHTML)
 - ▶ **Modèles :**
 - ▷ Beans **Java**
 - ▶ **Contrôleurs :**
 - ▷ Servlet unique “**FacesServlet**”
 - ▷ Bean Java “**managed-bean**”



Framework JSF

► Organisation du code sur une application avec JSF

► Configuration

► Fichier “web.xml”

- Déclaration de la **Servlet unique** (FacesServlet)

```
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
```



Framework JSF

► Organisation du code sur une application avec JSF

► Configuration

► Fichier “**web.xml**”

- Mapping des requêtes vers la Servlet unique

```
<servlet-mapping>  
    <servlet-name>Faces Servlet</servlet-name>  
    <url-pattern>*.xhtml</url-pattern>  
</servlet-mapping>
```

→ toutes les requêtes concernant des fichiers **xhtml**



3.

Facelets



Facelets

- ▶ **Facelet : techno de référence pour les vues avec JSF**
 - ▶ **Pages XHTML**
 - ▷ Format **XML**
 - ▷ Extensions possibles :
 - .jsf
 - .xhtml
 - .faces



Facelets

- ▶ **Facelet : techno de référence pour les vues avec JSF**

- ▶ **Pages XHTML**

- ▶ **Structure**

```
<!DOCTYPE html>  
<html xmlns="http://www.w3.org/1999/xhtml">  
  ...  
</html>
```



Facelets

► Facelet : techno de référence pour les vues avec JSF

► Pages XHTML

► Utilisation de bibliothèques et de balises

- Inclusion via **namespace** (attribut xmlns)

```
<!DOCTYPE html>
```

```
<html xmlns="http://www.w3.org/1999/xhtml"
```

```
    xmlns:f="http://java.sun.com/jsf/core"
```

```
    xmlns:h="http://java.sun.com/jsf/html"
```

```
    xmlns:ui="http://java.sun.com/jsf/facelets">
```

```
...
```

```
</html>
```



Facelets

- ▶ **Facelet : techno de référence pour les vues avec JSF**
 - ▶ **Pages XHTML**
 - ▶ **Exemple de balise pour afficher un libellé** (bibliothèque html de JSF)

```
<h:outputLabel for="lastname">Nom : <span class="requis">*</span>  
</h:outputLabel>
```



Facelets

- ▶ **Facelet : techno de référence pour les vues avec JSF**

- ▶ **Utilisation possible des “EL”**

- ▶ **Utilisation du “#”**

`#{customer.lastname}`



4.

Processus



Processus

- ▶ **Différentes étapes pour le traitement d'une requête**
 - ▶ **1. Restauration de la vue**
 - ▶ La requête entrante est dirigée vers la Servlet unique (FacesServlet)
 - ▶ La **FacesServlet** détermine le nom de la page demandée et vérifie si une vue existe pour cette page
 - ▶ La **FaceServlet** cherche les composants utilisés par cette vue
 - Création ou réutilisation si un composant est déjà créé
 - ▶ La **FaceServlet** sauvegarde la vue dans un objet de type **FacesContext**



Processus

- ▶ **Différentes étapes pour le traitement d'une requête**
 - ▶ **2. Application des valeurs contenues dans la requête**
 - ▶ Les paramètres de la requête sont **recupérés** et **convertis** par rapport au type
 - Si erreur lors de la conversion, une erreur est stockée dans l'objet de type **FacesContext**



Processus

- ▶ **Différentes étapes pour le traitement d'une requête**
 - ▶ **3. Validation des données**
 - ▶ Exécution des règles de validation mises en oeuvre par les développeurs
 - Si la validation d'un champ échoue, une erreur est stockée dans l'objet de type **FacesContext**



Processus

► Différentes étapes pour le traitement d'une requête

► 4. Mise à jour des valeurs

► Positionnement des valeurs dans les beans

- Valeurs valides au sens **format**



Processus

► Différentes étapes pour le traitement d'une requête

► 5. Appel aux actions

- Appel par JSF au code métier (partie "Modèle" du MVC)
 - Classes "beans" (managed beans)
 - Utilisation des données par l'application



Processus

► Différentes étapes pour le traitement d'une requête

► 6. Envoi de la réponse

- Affichage de la vue à l'utilisateur
 - Génération de la page HTML



5.

Managed Beans



Managed Bean

- ▶ **Bean Java géré par JSF**
 - ▶ **Pour faire le lien entre l'IHM et le code métier de l'application**
 - ▶ Contient des accesseurs (lien avec les données de la page)
 - ▶ Contient des méthodes de validation (pour les formulaires)
 - ▶ Contient des méthodes d'action (lien avec le code métier de l'application)

6.

Utilisation d'une implémentation



Implémentation JSF

- ▶ **Différentes implémentations existantes**

- ▶ **Eclipse Mojarra**

- ▶ <https://projects.eclipse.org/projects/ee4j.mojarra>

- ▶ **Apache MyFaces**

- ▶ <http://myfaces.apache.org/#/>



Implémentation JSF

► Utilisation de Apache MyFaces

▷ Import Maven

▷ Implémentation

```
<dependency>  
  <groupId>org.apache.myfaces.core</groupId>  
  <artifactId>myfaces-impl</artifactId>  
  <version>2.3.7</version>  
</dependency>
```




Implémentation JSF

► Utilisation de Apache MyFaces

▷ Import Maven

- ▷ Utilisation de CDI pour l'injection de dépendances (beans)

```
<dependency>  
  <groupId>javax.enterprise</groupId>  
  <artifactId>cdi-api</artifactId>  
  <version>2.0</version>  
  <scope>provided</scope>  
</dependency>
```



Implémentation JSF

► Utilisation de Apache MyFaces

▷ Import Maven

- ▷ Utilitaire JAXB-API (transformation Java <-> XML)

```
<dependency>  
  <groupId>javax.xml.bind</groupId>  
  <artifactId>jaxb-api</artifactId>  
  <version>2.3.1</version>  
</dependency>
```



7.

Projet fil rouge

CRM

Mise en place de JSF



Projet Fil Rouge

- ▶ **Mise en place du projet**
 - ▷ pom.xml
- ▶ **Mise en place de la gestion des clients**
 - ▷ Formulaire de création d'un nouveau client
 - ▷ Page d'affichage de la liste des clients
- ▶ **Mise en place de la gestion des commandes**
 - ▷ Formulaire de création d'une nouvelle commande
 - ▷ Page d'affichage de la liste des commandes



Projet Fil Rouge

- ▶ Mise en place de la gestion des clients
 - ▷ Formulaire de création d'un nouveau client
 - ▷ Fichier createCustomer.xhtml
 - ▷ Bean CustomerBean.java
 - Méthode de création d'un client
 - Méthode de validation personnalisée pour le téléphone
 - ▷ Page d'affichage de la liste des clients
 - ▷ Fichier listCustomers.xhtml
 - ▷ Bean CustomerBean.java
 - Méthode de récupération des clients
 - Méthode de suppression d'un client



Projet Fil Rouge

- ▶ Mise en place de la gestion des commandes
 - ▷ Formulaire de création d'une nouvelle commande
 - ▷ Fichier createOrder.xhtml
 - ▷ Bean OrderBean.java
 - Méthode de création d'une commande
 - Méthode(s) de validation personnalisée(s) ?
 - ▷ Page d'affichage de la liste des commandes
 - ▷ Fichier listOrders.xhtml
 - ▷ Bean OrderBean.java
 - Méthode de récupération des commandes
 - Méthode de suppression d'une commande