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

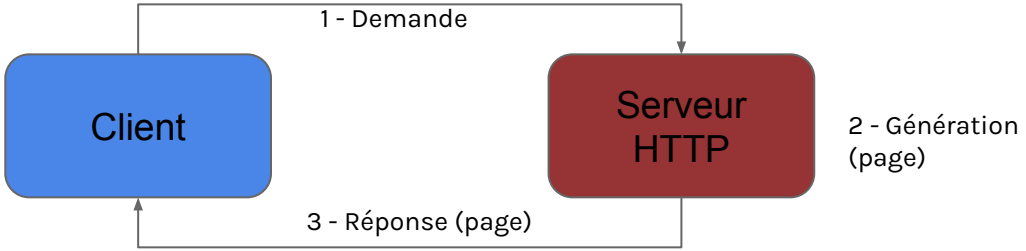
  - **Objectif** : Faciliter le développement d'application web déployées et exécutées sur un **serveur d'application**

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

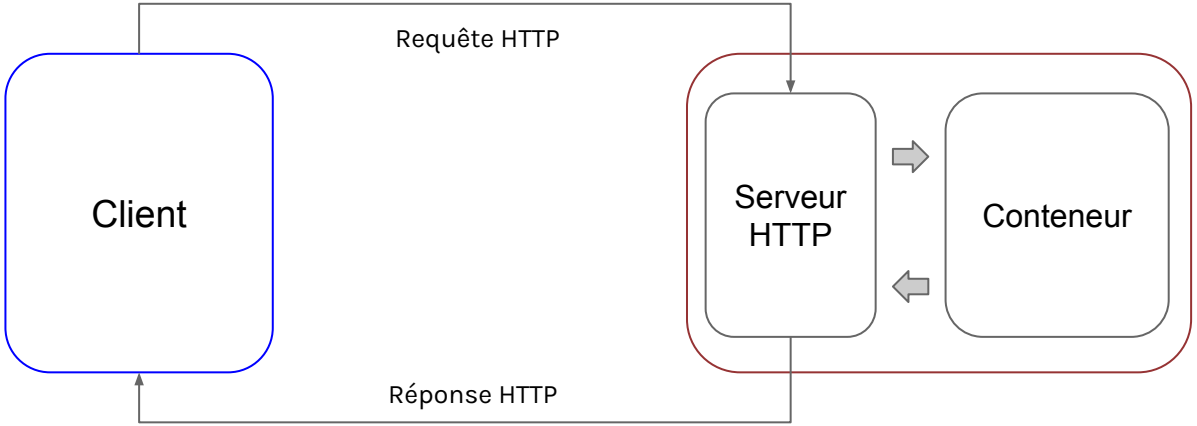
- Echanges HTTP : Entre un client et un serveur
  - Client → Navigateur
  - Serveur → Serveur HTTP



- 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**



2.

Le modèle MVC

Le modèle MVC

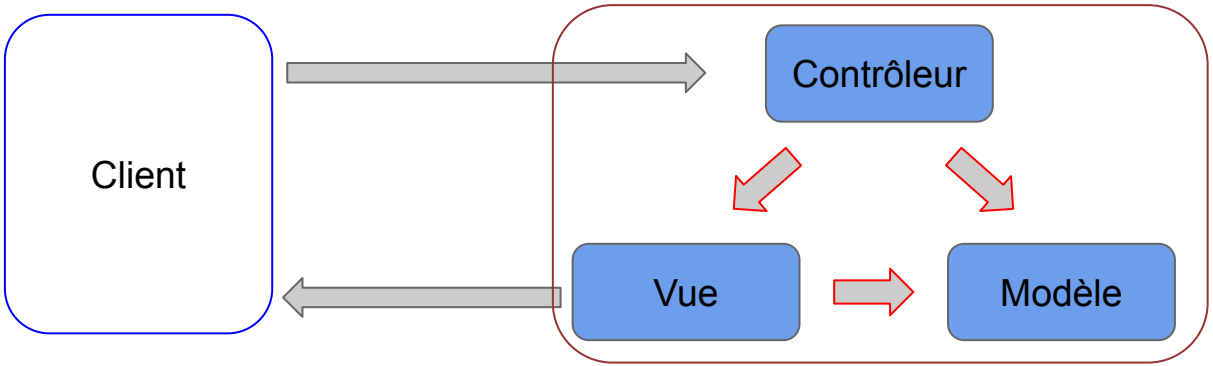
- MVC = Modèle - Vue - Contrôleur (Model - View - Controller)

- 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

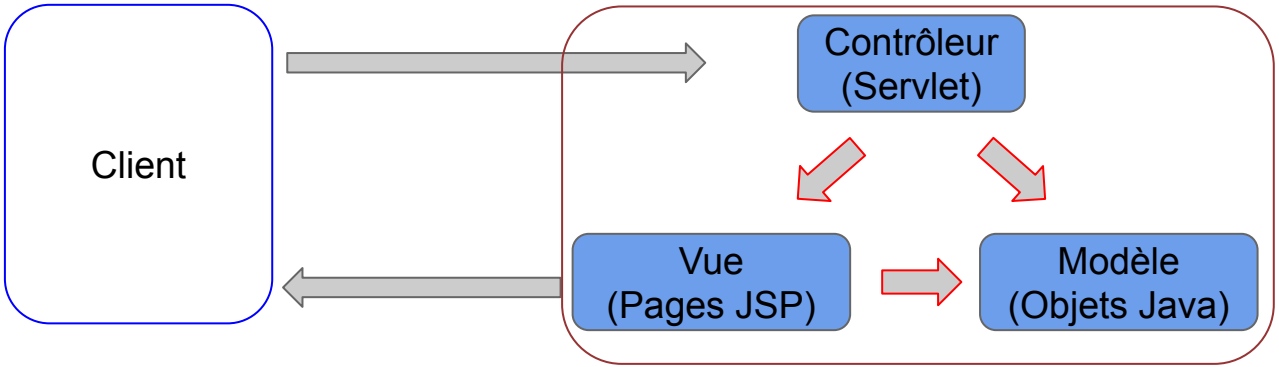
Le modèle MVC

- 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 le 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

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

JSP

- 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)

JSP

- 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)

7.

JSTL

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

- 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

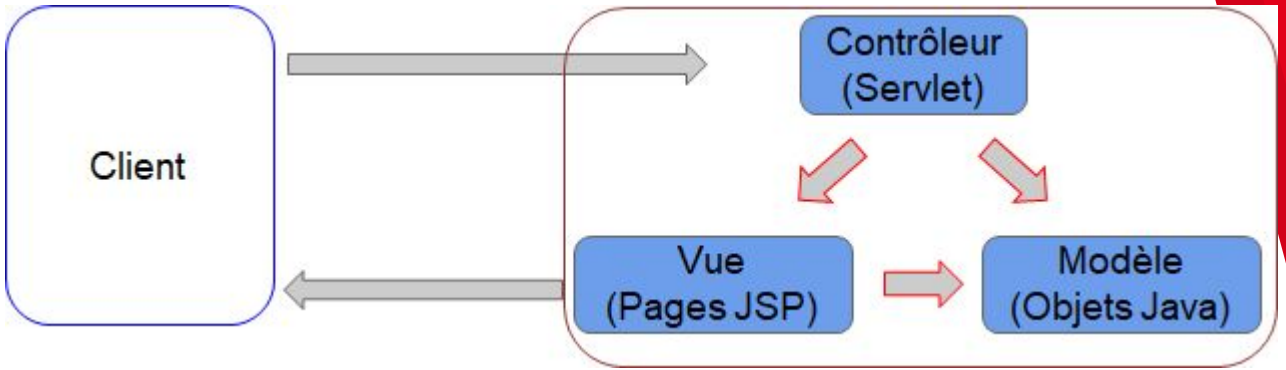
- JSTL : JSP Standard Tag Library
  - Documentation : https://javaee.github.io/jstl-api/

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

- Eléments nécessaires pour le traitement d'un formulaire
  - Un objet (bean Java) pour stocker les données validées du formulaire
  - Un objet « Métier » pour récupérer 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