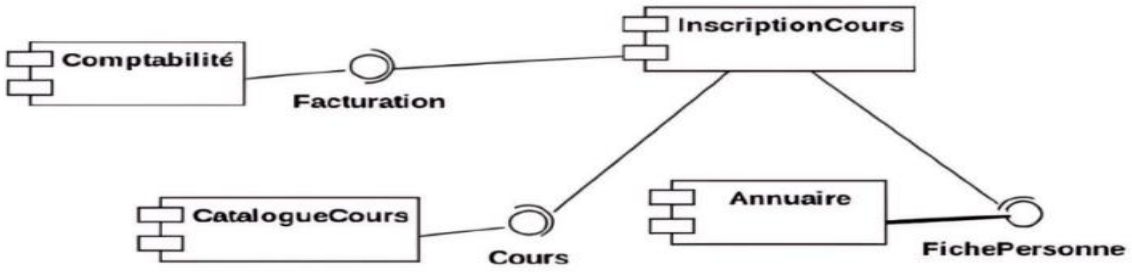
Compte_Rendu de TD3 Architecture

Enseignante: Ghazala Hcini Date : 05-12-2025



Exercice 1 – Analyse du diagramme de composants

1. Qu'est-ce qu'il représente le diagramme 1

Le diagramme 1 est un diagramme de composants UML 2 qui représente l'architecture logicielle statique d'un système d'inscription à des cours universitaires. Il illustre les **composants logiciels**, leurs **interfaces fournies**, leurs **interfaces requises**, ainsi que les **relations d'assemblage** entre eux.

2. Commenter les différents éléments de ce diagramme

***Composants**
Ce sont des modules logiciels autonomes qui encapsulent des fonctionnalités. Chaque composant est représenté par un rectangle stéréotypé <<component>>.

***interface**:

- **Interfaces fournies (Provided Interface)**
  Le « lollipop » noir (cerle) = interface fournie (service offert)
  Une interface fournie indique **les services que le composant met à disposition**.
- **Interfaces requises (Required Interface)**
  • Le « socket» blanc (semi_cerle) = interface requise (dépendance)
  Elles indiquent **les services dont le composant dépend** pour fonctionner.

***Connecteurs :**ce sont les liens entre interfaces fournies et requises**.**

Composant	Responsabilité principale	**Interfaces**

Compatibilité	Gérer la facturation et les paiements des inscriptions	Facturations
CatalogueCours	Fournir la liste des cours disponibles	Cours
Annuaire	Fournir les informations sur les personnes (étudiants , profs)	FichePersonne(four
InscriptionCours	Coeur métier : gérer l'inscription d'un étudiant à un cours	Facturation,Cours,FichePersonne

3. Quelles sont les interfaces fournies et requises de différents composants ?

Composant	Interfaces fournies	Interfaces requises
Compatibilité	-	facturation
Catalogue_Cours	Cours	-
Annuaire	FichePersonne	-
Inscription_Cours	Facturation	Cours , FichePersonne

→ InscriptionCours est donc le composant assembleur qui orchestre les autres.

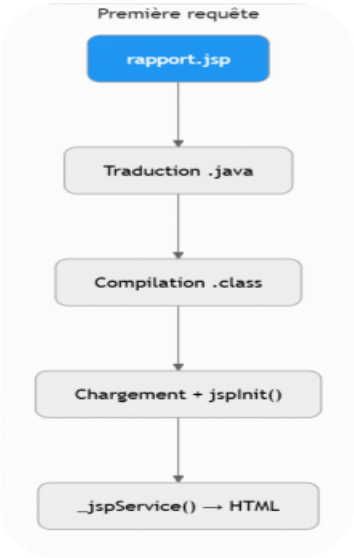
Exercice 2 – Cycle de vie des JSP dans un conteneur Java EE

1. Mécanisme de Transformation (première requête vers rapport.jsp)

Processus détaillé lors de la toute première requête HTTP :
Le client envoie une requête GET http://…/rapport.jsp
Le conteneur web (Tomcat, GlassFish, WildFly…) reçoit la requête.
Rôle du conteneur :
Verifier si la JSP a déjà été traduite et compilée.
Le conteneur constate qu'il n'existe pas encore de servlet associée à cette JSP dans son cache.
• Il lit le fichier rapport.jsp
•**Traduction**: Il le **traduit** en un fichier source Java : une classe servlet qui étend HttpJspBase (indirectement javax.servlet.http.HttpServlet)

•**Compilation**: Il **compile** cette servlet en bytecode (.class)

• Il charge la classe en mémoire (ClassLoader du conteneur)
• Il instancie la servlet (appel du constructeur)
• Il appelle la méthode _jspInit() (équivalent de init() des servlets)
• Il invoque la méthode _jspService() pour traiter la requête
• La réponse HTML est renvoyée au client



Résultat de la transformation :
Un fichier **.java** et un fichier **.class** sont générés dans le répertoire de travail du conteneur (ex : work/Catalina/localhost/app/org/apache/jsp/rapport_jsp.java et rapport_jsp.class).
La servlet générée s'appelle généralement **rapport_jsp** et étend org.apache.jasper.runtime.HttpJspBase.
La servlet auto-générée est ensuite prête à être exécutée et répond aux requêtes via service() puis doGet() / doPost().

1. Performance et Cycle de Vie (requêtes suivantes)

Stratégie du conteneur : ne re-traduire et re-compiler la JSP que si elle a changé.
• **Critère de vérification :** À chaque requête, le conteneur compare le timestamp (date de dernière modification) du fichier .jsp avec celui du fichier .class de la servlet générée. Si le .jsp est plus récent → re-traduction + recompilation. Sinon → la servlet déjà chargée en mémoire est réutilisée.
• **Conséquence du cycle / Principal avantage :** La coûteuse phase de traduction/compilation n'est effectuée qu'une seule fois (ou uniquement lors des modifications).
Toutes les requêtes suivantes exécutent directement la méthode _jspService() de la servlet déjà chargée en mémoire → gain de performance très important (de l'ordre de 50 à 100 fois plus rapide) Cycle de vie optimisé,.

1. Maintenance et Évolution (modification du fichier JSP)

• **Déclencheur de Recompilation :** Le conteneur détecte automatiquement la modification grâce à la comparaison des timestamps (voir question précédente).
Dès qu'un client effectue une nouvelle requête sur rapport.jsp après la modification du fichier sur le disque, le conteneur :
• Supprime l'ancienne classe servlet
• Relance tout le cycle de traduction → nouveau fichier .java
• Recompilation → nouveau .class
• Recharge la nouvelle version de la servlet Ce mécanisme est totalement transparent pour le développeur et permet un rechargement à chaud (hot deploy) des JSP.

Exercice 3 – Application de gestion de projets et tâches en MVC

---

1. Besoins fonctionnels principaux

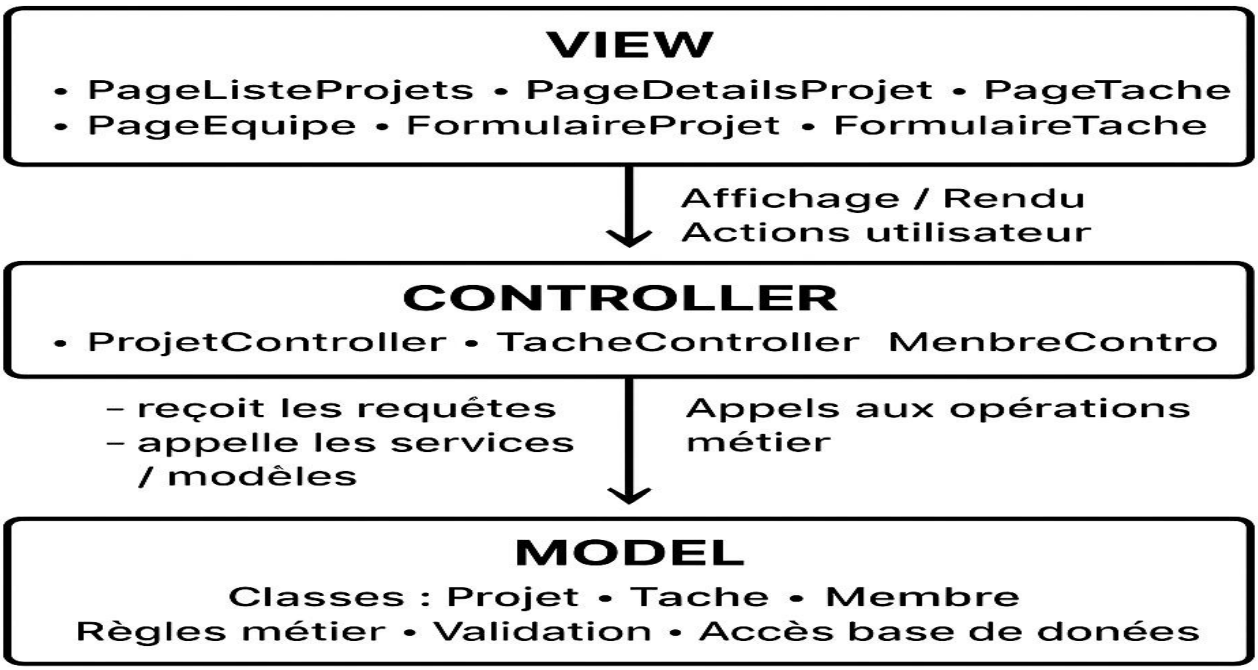
• Gestion des projets (créer, modifier, supprimer, archiver)
• Gestion des tâches (to-do list) associées à un projet (créer, marquer comme terminée, assigner, deadlines, priorités)
• Gestion des équipes/membres (utilisateurs, rôles : admin, chef de projet, membre)
• Assignation de tâches à des membres
• Suivi d'avancement (pourcentage, burndown, notifications)
• Authentification et gestion des droits

2. Technologies utilisées

Selon le type d'application souhaité :

Architecture MVC (Model–View–Controller)	**Web (recommandé)**	**Mobile**
Front-end	HTML, CSS, JS, Angular ou React ,JSP/Servlet pour Java.	Android (Java/Kotlin) ios(swift)ou Flutter,react native
Back-end	Java Spring Boot, Node.js ou PHP Laravel	API backend REST pour communication
Base de données	MySQL / PostgreSQL/SQLite, MongoDB, selon besoin.	

3. Modéliser les différents composants suivant l'architecture MVC



Etudiante: Yakine Benali Date : 05-12-2025