Formation Architecture JAVA EE

El Hadji Gaye

Auteur	El Hadji Gaye
Pour	Formations
Date	16/01/2024
Objet	Architecture JAVA EE Jakarta EE.

I) Glossaire

- **API** : signifie Application Programming Interface. Ce qui veut dire que c'est un ensemble de bibliothèques et librairies dédié pour implémenter une fonctionnalité donnée.
- **Servlet** : une servlet est une classe Java qui permet de créer dynamiquement des données au sein d'un serveur HTTP. Ces données sont le plus généralement présentées au format HTML, mais elles peuvent également l'être au format XML ou tout autre format destiné aux navigateurs web. Les servlets utilisent l'API Java Servlet (package **javax.servlet**). Une servlet s'exécute dynamiquement sur le serveur web et permet l'extension des fonctions de ce dernier, typiquement : accès à des bases de données, transactions d'e-commerce, etc. Une servlet peut être chargée automatiquement lors du démarrage du serveur web ou lors de la première requête du client. Une fois chargées, les servlets restent actives dans l'attente d'autres requêtes du client.
- **Bean** : le « **Bean** » (ou haricot en français) est une technologie de composants logiciels écrits en langage Java. Les **Beans** sont utilisés pour encapsuler plusieurs objets dans un seul objet. Le « **Bean** » regroupe alors tous les attributs des objets encapsulés. Ainsi, il représente une entité plus globale que les objets encapsulés de manière à répondre à un besoin métier.
- **JSP** : Java Server Pages.
- **JSF** : Java Server Faces.
- **EJB** : Entreprise Java Bean.
- **EAR** : Un EAR (pour Enterprise Application Archive) est un format de fichier utilisé par Java EE pour empaqueter (en) un ou plusieurs modules dans une seule archive, de façon à pouvoir déployer ces modules sur un serveur d'applications en une seule opération, et de façon cohérente.
- **War** : WAR (Web Application Archive) est un fichier JAR utilisé pour contenir un ensemble de Java Server Pages, servlets, classes Java, fichiers XML, et des pages web statiques (HTML, JavaScript…), le tout constituant une application web. Cette archive est utilisée pour déployer une application web sur un serveur d'applications.
- **JSTL** : JSTL (Java Server page Standard Tag Library). C'est un ensemble de tags personnalisés et développés qui propose des fonctionnalités souvent rencontrées dans les JSP.

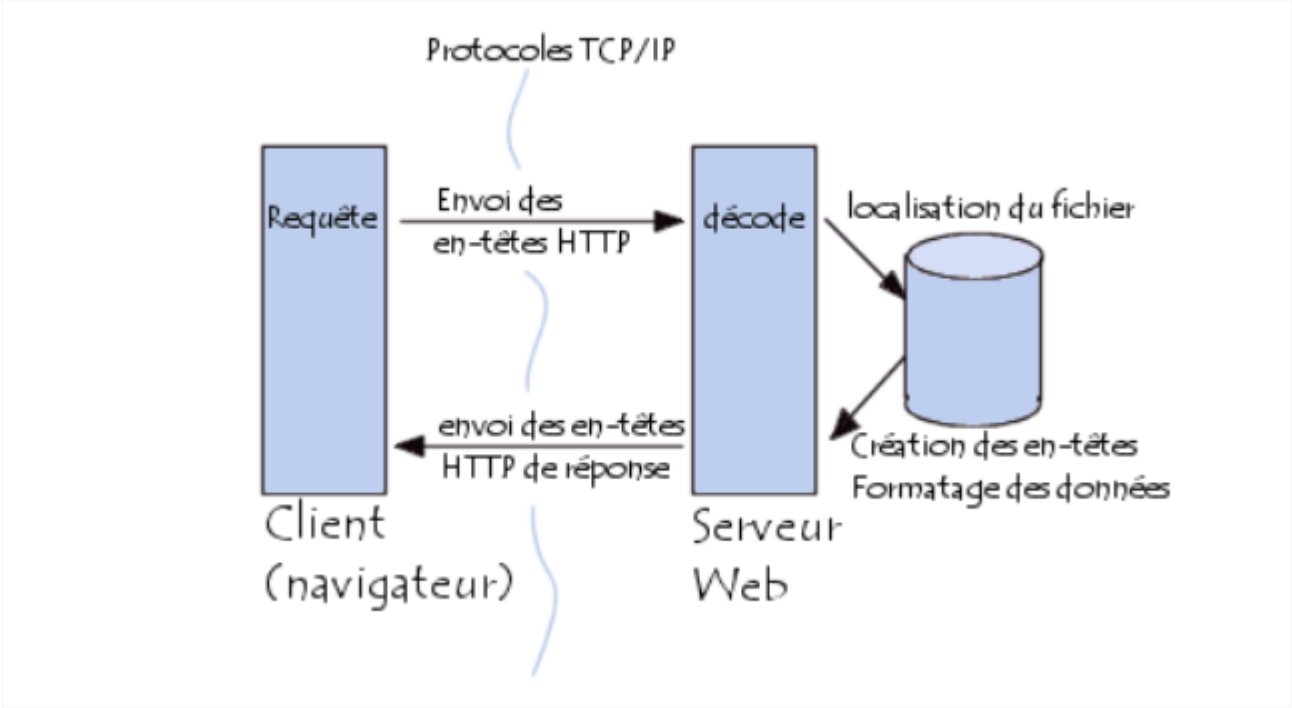
II) Introduction

JAVA EE est l'acronyme de Java Entreprise Edition. Cette édition est dédiée à la réalisation d'applications pour les entreprises. J2EE est basée sur J2SE (Java 2 Standard Edition) qui contient les API de base de Java. J2EE désigne aussi les technologies Java utilisées pour créer des applications d'entreprise avec le JDK 1.4 (ou antérieur).
Depuis la version du JDK 1.5, J2EE est renommée Java EE (Enterprise Edition).
Les termes Java EE 5, Java EE 6 et Java EE 7 ont alors été utilisés pour désigner l'ensemble des technologies qui concourent à créer une application d'entreprise avec la plate-forme Java.

III) Rappels sur le protocole HTTP.

Le protocole HTTP (HyperText Transfer Protocol) est le protocole le plus utilisé sur Internet depuis 1990.
Le but ce protocole est de permettre un transfert de fichiers (essentiellement au format HTML) localisés grâce à une chaîne de caractères appelée URL entre un navigateur (le client) et un serveur Web.

La communication entre le navigateur et le serveur se fait en deux temps :



- Le navigateur effectue une **requête HTTP**
- Le serveur traite la requête puis envoie une **réponse HTTP**

Une requête HTTP est traitée à travers plusieurs méthodes :

GET : c'est la méthode la plus courante pour demander une ressource. Une requête GET est sans effet sur la ressource, il est possible de répéter la requête sans effet.

POST : cette méthode est utilisée pour transmettre des données en vue d'un traitement de ressource (le plus souvent depuis un formulaire HTML). L'URI fourni est l'URI d'une ressource à laquelle s'appliqueront les données envoyées. Le résultat peut être la création de nouvelles ressources ou la modification de ressources existantes.

HEAD : cette méthode ne demande que des informations sur la ressource, sans demander la ressource elle-même.

OPTIONS : cette méthode permet d'obtenir les options de communication d'une ressource ou du serveur en général.

CONNECT : cette méthode permet d'utiliser un proxy comme un tunnel de communication.

TRACE : cette méthode demande au serveur de retourner ce qu'il a reçu dans le but de tester et d'effectuer un diagnostic sur la connexion.

PUT: cette méthode permet de remplacer ou d'ajouter une ressource sur le serveur. L'URI fourni est celui de la ressource en question.

PATCH: cette méthode permet, contrairement à PUT, de faire une modification **partielle** d'une ressource.

DELETE: cette méthode permet de supprimer une ressource du serveur.

IV) Architecture Java Web et Java EE.

Java EE est la version "entreprise" de Java, elle a pour but de faciliter le développement d'applications distribuées.
Mais en fait, Java EE est avant tout une norme.

C'est un ensemble de standard décrivant des services techniques comme, par exemple, comment accéder à un annuaire, à une base de données, à des documents...

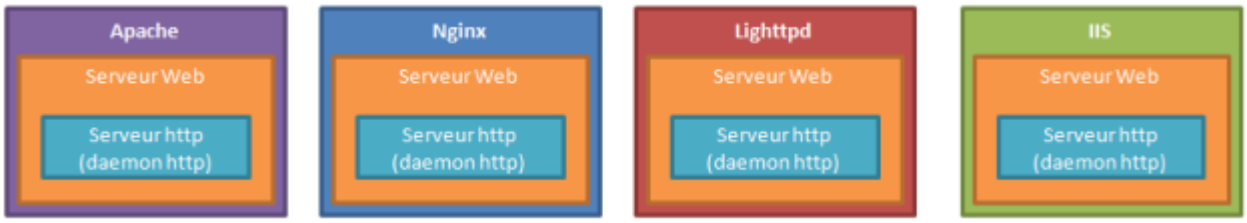
Important : Java EE définit ce qui doit être fournit mais ne dit pas comment cela doit être fournit.

Exemple de services :

- JNDI (Java Naming and Directory Interface) est une API d'accès aux services de nommage et aux annuaires d'entreprises tels que DNS, NIS, LDAP…
- JTA (Java Transaction API) est une API définissant des interfaces standard avec un gestionnaire de transactions.

1. *Serveur Web Http*

Pour commencer, il nous faut tout d'abord expliquer un certain nombre de choses :
Qu'est-ce c'est un serveur HTTP ?
Un serveur HTTP, c'est un serveur qui gère exclusivement des requêtes HTTP. Il a pour rôle d'intercepter les requêtes HTTP, sur un port qui est par défaut 80, pour les traiter et générer ensuite des réponses HTTP. Tous les serveurs web embarquent un daemon HTTP (httpd) ou équivalent qui s'occupe de cette fonctionnalité.



Exemple de serveurs web : Apache, Nginx, Lighttpd, IIS…

Quelles sont les fonctionnalités d'un serveur Web ?
A part sa fonction basique qui est d'intercepter et de répondre en Http, un serveur web peut avoir d'autres fonctionnalités telles que :

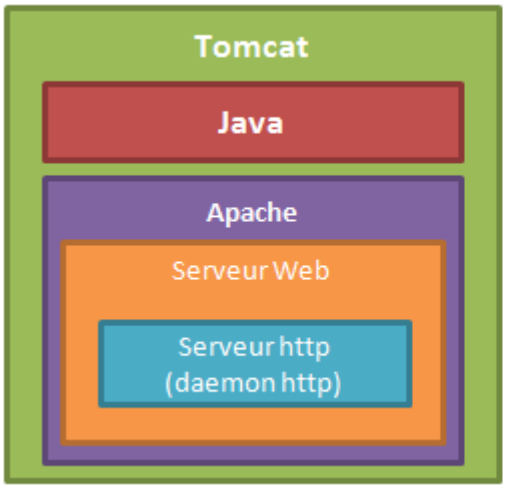
- La gestion de la sécurité, comme les fonctionnalités de restriction des accès par domaine, par utilisateur, par groupe ou par adresse IP,
- La gestion du contenu, comme la redirection des requêtes http, la personnalisation des messages d'erreurs, ou la gestion des timeout ...

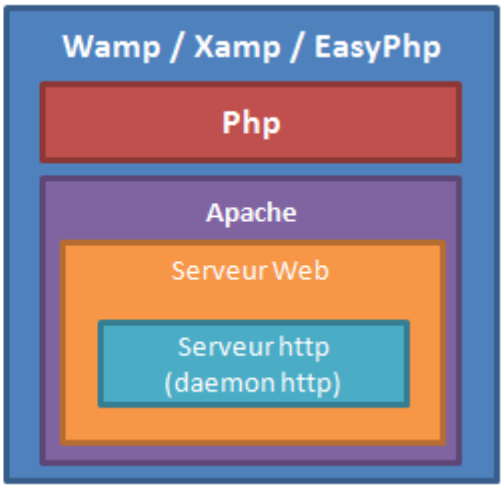
Dans le serveur apache, par exemple, ces fonctionnalités sont implémentées sous forme de modules (mod_alias, mod_authn_core, mod_proxy…).

2. *Conteneur web ou conteneur à Servlets*

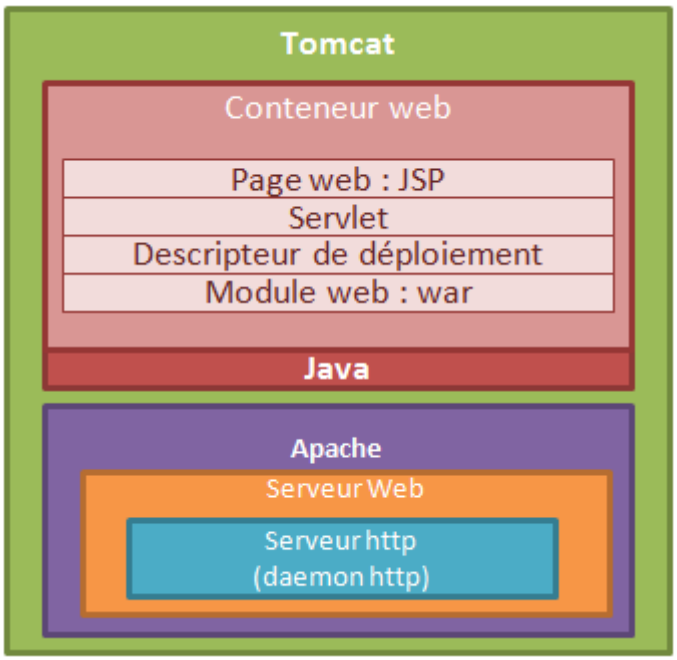
Lorsqu'on étend notre serveur web il devient un conteneur Web. Cette extension va permettre d'avoir la possibilité d'exécuter des programmes écrits avec des langages de programmation (Java, Php, C# ou autres) dans le serveur web.

Par exemple, le serveur Tomcat n'est autre qu'un serveur Apache couplé avec un moteur web java et, les serveurs tel-qu'EasyPhp, Wamp ou Xamp ne sont que des serveurs Apaches couplés avec un moteur web Php.





Maintenant, on va se concentrer sur le serveur Tomcat qui est un conteneur web Java (et pas un serveur d'application JEE car il n'a pas de conteneur d'EJB), pour analyser son architecture.



Le conteneur Web Tomcat est composé d'un moteur JSP, un moteur servlet et d'un descripteur de déploiement pour les modules web de type **war**. Ces moteurs sont en réalité des API qui sont implémentés dans le serveur Tomcat, et qui permettent de faire déployer seulement des applications web Java de type **war**.

Les applications Java de type EAR ne peuvent pas être déployées dans Tomcat parce que, tout simplement, le serveur manque les API nécessaires et conformes aux spécifications pour l'implémentation des serveurs d'application Java JEE.

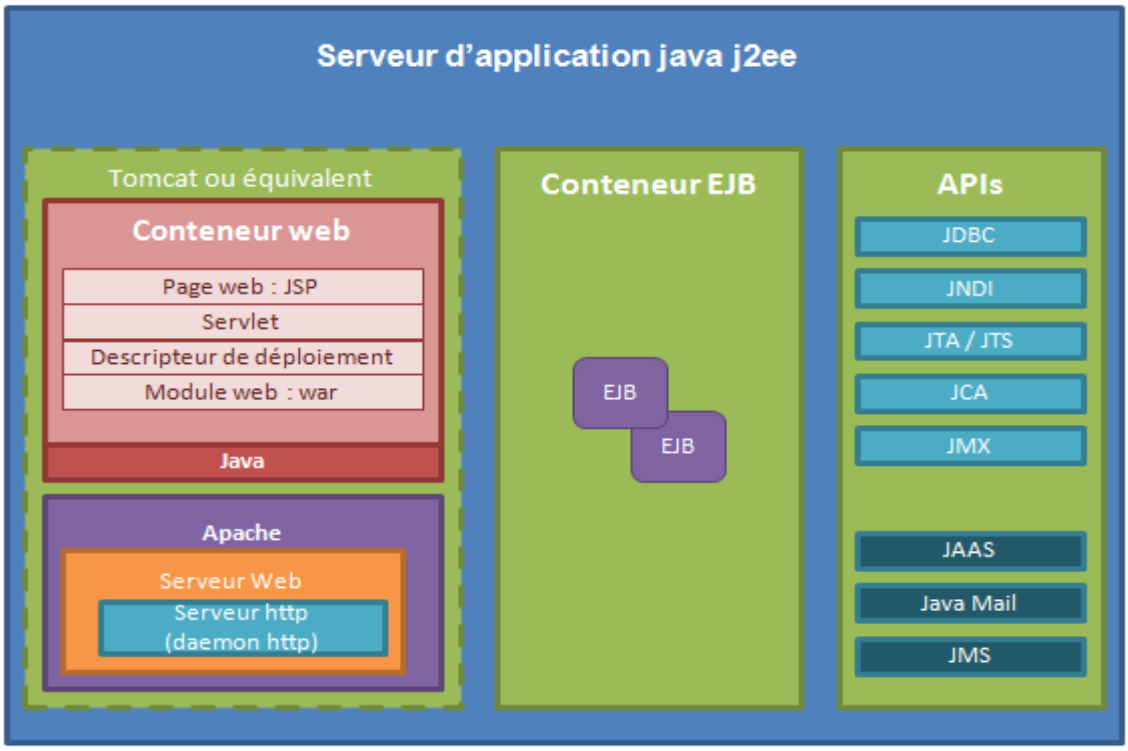
Par exemple, si vous utilisez la bibliothèque JPA dans votre application web et vous le déployez sur un serveur Tomcat, vous serez obligés d'embarquer les jars JPA dans le répertoire lib de l'application, alors que si vous déployez sur un serveur d'application comme JBOSS, vous n'aurez besoin d'aucun **jar** additionnel.

3. *Serveur d'application Java EE*

Si vous avez bien suivi, il faut étendre encore plus le serveur Tomcat pour qu'il devienne un vrai serveur d'application Java JEE.

L'extension nécessaire est composée de deux parties essentielles :

a) **Un conteneur EJB** qui encapsule les traitements des Entreprise JavaBeans.
b) **Un conteneur à Servlet.**
c) Un ensemble de services répartis en :
   A. Des services d'infrastructures
      i. **JDBC** (Java DataBase Connectivity) API d'accès aux bases de données relationnelles.
      ii. **JNDI** (Java Naming and Directory Interface) API d'accès aux services de nommage et aux annuaires d'entreprises.
      iii. **JTA/JTS** (Java Transaction API/Java Transaction Services) API pour la gestion de transactions.
      iv. **JCA** (JEE Connector Architecture) API de connexion au système d'information de l'entreprise comme les ERP.
      v. **JMX** (Java Management Extension) API permettant de développer des applications web de supervision d'applications
   B. Des services de communication:
      i. **JAAS** (Java Authentication and Authorization Service) API de gestion de l'authentification.
      ii. **JavaMail API** pour la gestion de courrier électronique.
      iii. **JMS** (Java Message Service) API de communication asynchrone entre application.



4. *Compaisons des serveurs d'applications JEE*

a. La spécification Jakarta EE (anciennement Java EE)

La spécification Jakarta/Java EE 8 est définie par le JCP : https://jcp.org/en/jsr/detail?id=366. Elle définit les services et les API proposés par un serveur d'applications Java EE. Il s'agit bien d'une spécification et non d'un logiciel. Elle est composée d'un grand nombre de sous-spécifications : **Servlet, JSP, EL, JSF, EBJ, JPA**, ...

b. Les serveurs d'applications compatibles Jakarta/Java EE

Un certain nombre de logiciels implémentent ces spécifications. Parmi les principaux serveurs d'applications compatibles Java EE, on retrouve : **IBM WebSphere, RedHat WildFly (JBoss), Oracle/Eclipse GlassFish, Oracle WebLogic, Apache Geronimo**, ...

c. Comparatif entre les différents serveurs d'applications

Le tableau suivant présente les versions des principales APIs Java EE pour quelques serveurs d'applications.

Nom du serveur	Version du serveur	Développeur	SE	EE	Servlet	JSP	EL	JSF	EJB	JPA	JMS	JAX-WS	JAX-RS
Tomcat	8.0	Fondation Apache	7.0-10.0	7.0	3.1	2.3	3.0						
Tomcat	9.0	Fondation Apache	8.0-10.0	8.0	4.0	2.4	3.1						
Jetty	9.4	Eclipse	8.0	7.0	3.1	2.3	3.0						
Glassfish	4.0	Oracle	7.0-10.0	7.0	3.1	2.3	3.0	2.2	3.2	2.2	2.0	2.2	2.0
Glassfish	5.0	Oracle	8.0-10.0	8.0	4.0	2.3	3.0	2.3	3.2	2.2	2.0	Removed	2.1
WildFly (JBoss)	13	Redhat		7.0	3.1	2.3	3.0	2.2	3.2	2.2	2.0	2.2	2.0
WildFly (JBoss)	14	Redhat		8.0	4.0	2.3	3.0	2.3	3.2	2.2	2.0	Removed	2.1
WebSphere	8.5.5	IBM	6.0-8.0	6.0	3.0	2.2		2.0	3.1	2.0	1.1		
WebSphere	9.0	IBM	8.0	7.0	3.1	2.3		2.2	3.2	2.1	2.0	2.2	2.0

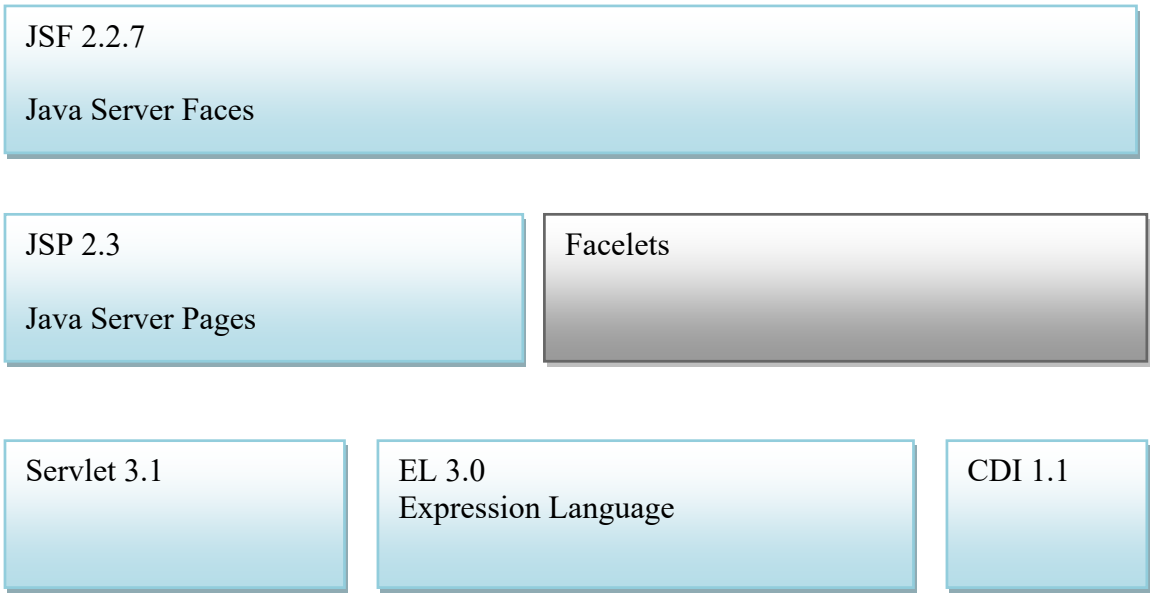
Note : attention, Tomcat et Jetty ne sont pas à proprement parler des serveurs d'applications, dans le sens où ils n'implémentent pas toute la spécification Jakarta/Java EE. Ce sont juste des serveurs HTTP.

d. Carte d'identité du serveur d'application GlassFish 4.1

Le serveur d'application que nous allons utiliser est **GlassFish 4.1**. **GlassFish 4.1** est sur **Java SE 7.0 et Java EE 7.0**. Il doit être installé sur un **JDK > 6**.

Nom du serveur	Version du serveur	Développeur	SE	EE	Servlet	JSP	EL	JSF	EBJ	JPA	JMS	JAX-WS	JAX-RS
Glassfish	4.0	Oracle	7.0-10.0	7.0	3.1	2.3	3.0	2.2	3.2	2.2	2.0	2.2	2.0

Le schéma ci-dessous donne plus de détails sur l'architecture d'un serveur **GlassFish 4.1**.



Vous pouvez vérifier ces informations dans la liste des librairies de glassFish sur :

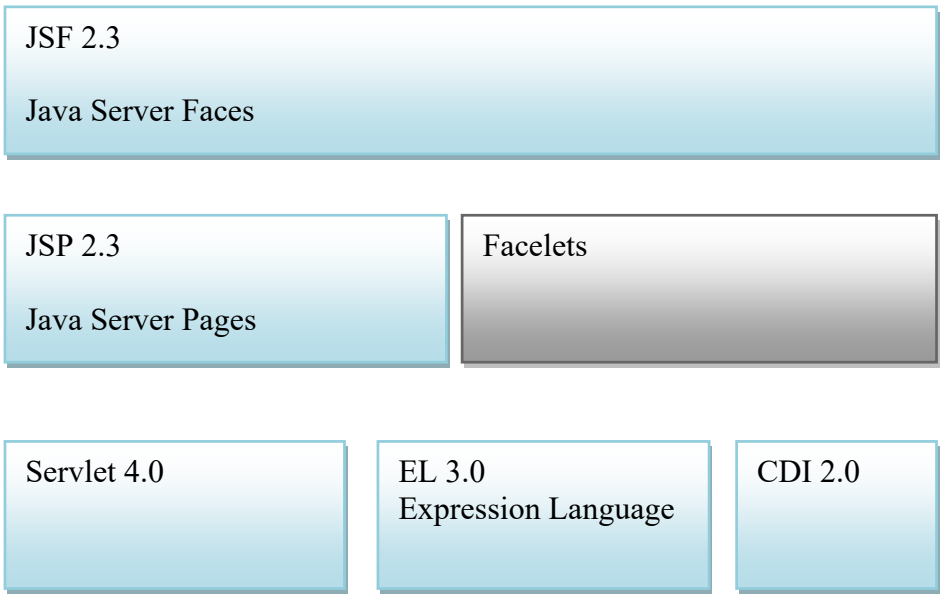
C:/Program Files/glassfish-4.1/glassfish/modules.

e. Carte d'identité du serveur d'application GlassFish 5.0

Le serveur d'application que nous allons utiliser est **GlassFish 5.0**. **GlassFish 5.0** est sur **Java SE 8.0** et **Java EE 8.0**. Il doit être installé sur un **JDK > 7**.

Nom du serveur	Version du serveur	Développeur	SE	EE	Servlet	JSP	EL	JSF	EBJ	JPA	JMS	JAX-WS	JAX-RS
Glassfish	5.0	Oracle	8.0	8.0	4.0	2.3	3.0	2.3	3.2	2.2	2.0	Removed	2.1

Le schéma ci-dessous donne plus de détails sur l'architecture d'un serveur **GlassFish 5.0**.



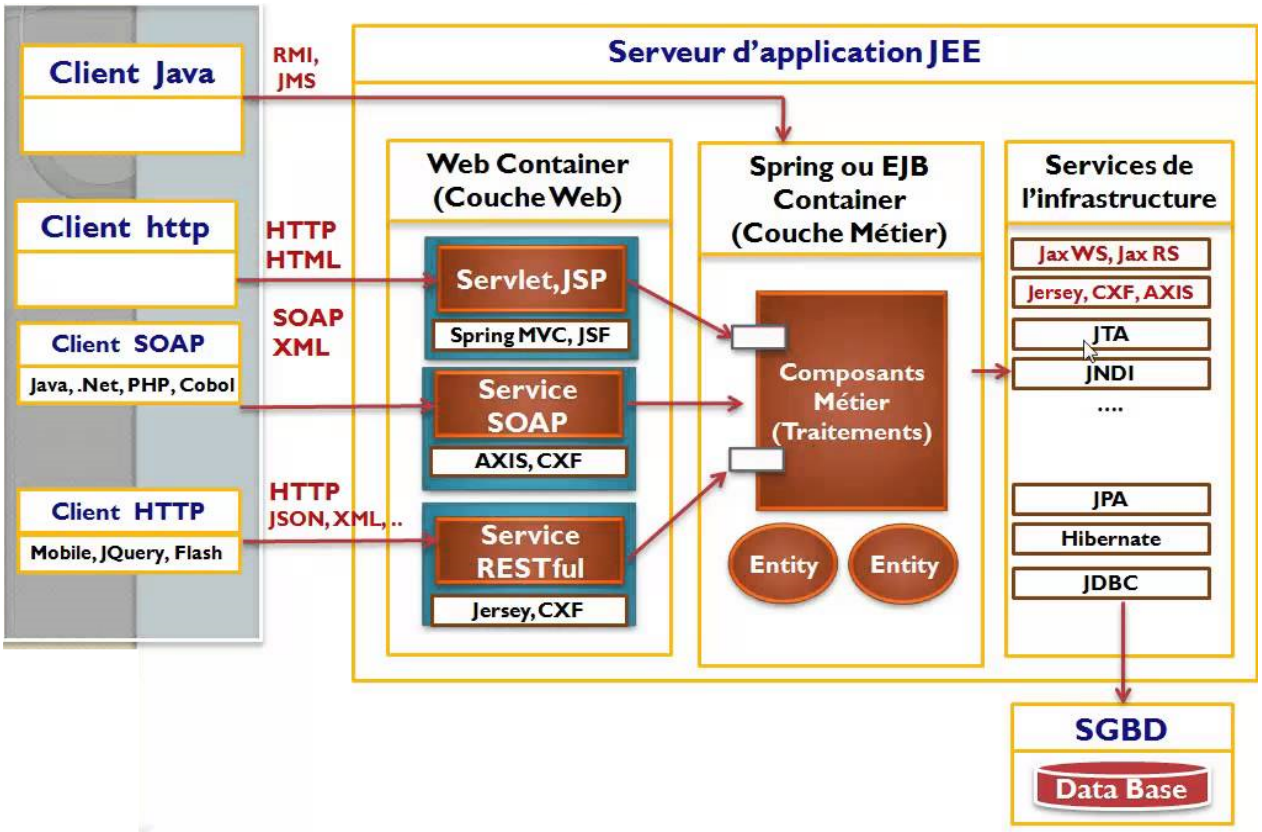
f. Les différentes versions de Tomcat

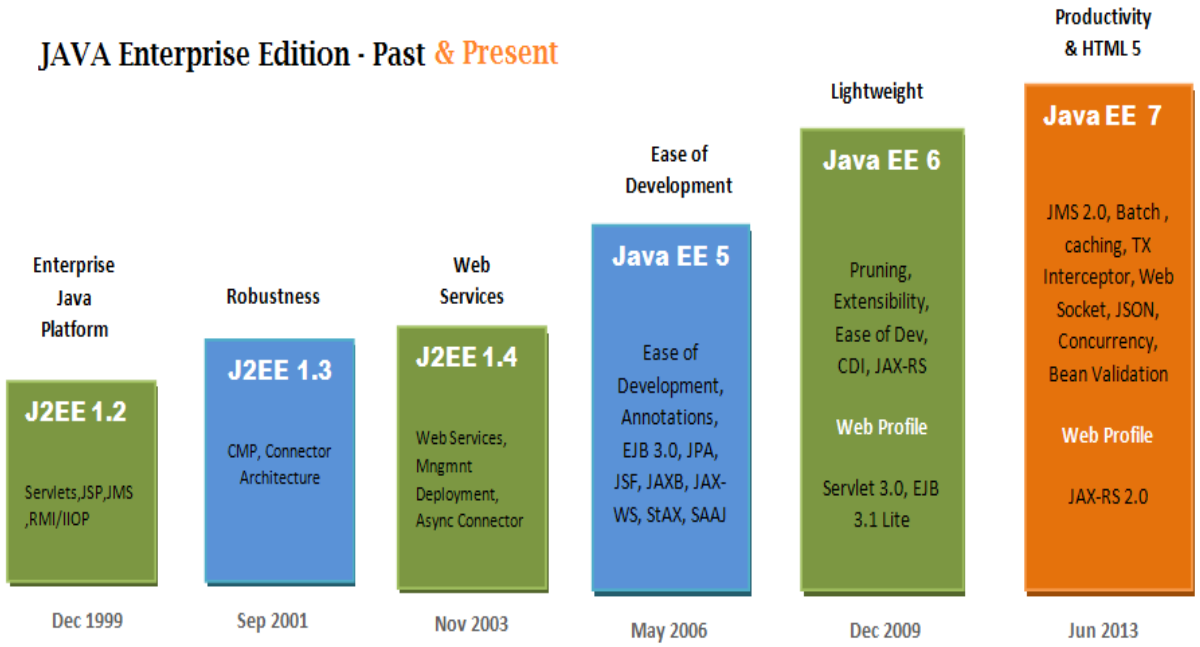
Le tableau ci-dessous donne les différentes versions de Tomcat avec leurs spécificités.

Servlet API	JSP API	EL Spec	WebSocket Spec	JASPIC Spec	Apache Tomcat Version	Latest Released Version	Supported Java Versions
4.0	2.3	3.0	1.1	1.1	9.0.x	9.0.17	Java EE 8
3.1	2.3	3.0	1.1	1.1	8.5.x	8.5.39	Java EE 7
3.1	2.3	3.0	1.1	N/A	8.0.x	8.0.53	Java EE 7
3.0	2.2	2.2	1.1	N/A	7.0.x	7.0.93	Java EE 6
2.5	2.1	2.1	N/A	N/A	6.0.x	6.0.53	Java EE 5
2.4	2.0	N/A	N/A	N/A	5.5.x	5.5.36	Java EE 1.4
2.3	1.2	N/A	N/A	N/A	4.1.x	4.1.40	Java EE 1.3
2.2	1.1	N/A	N/A	N/A	3.3.x	3.3.2	Java EE 1.1

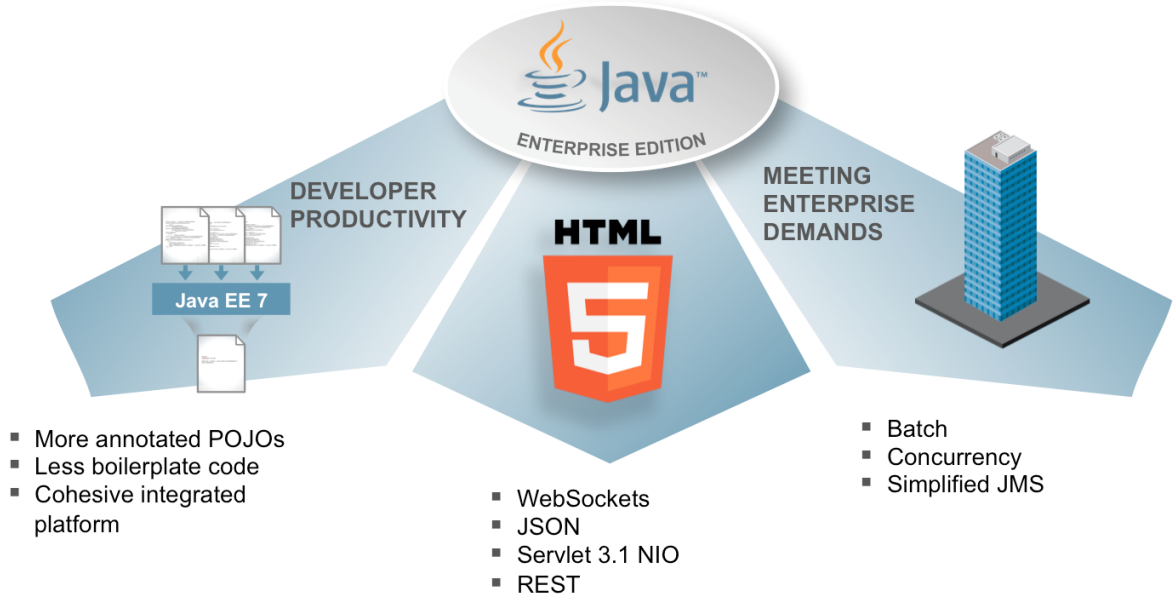
5. *Plus loins dans l'architecture Java EE*

Pour aller plus loin dans l'architecture Java EE nous pouvons ajouté :

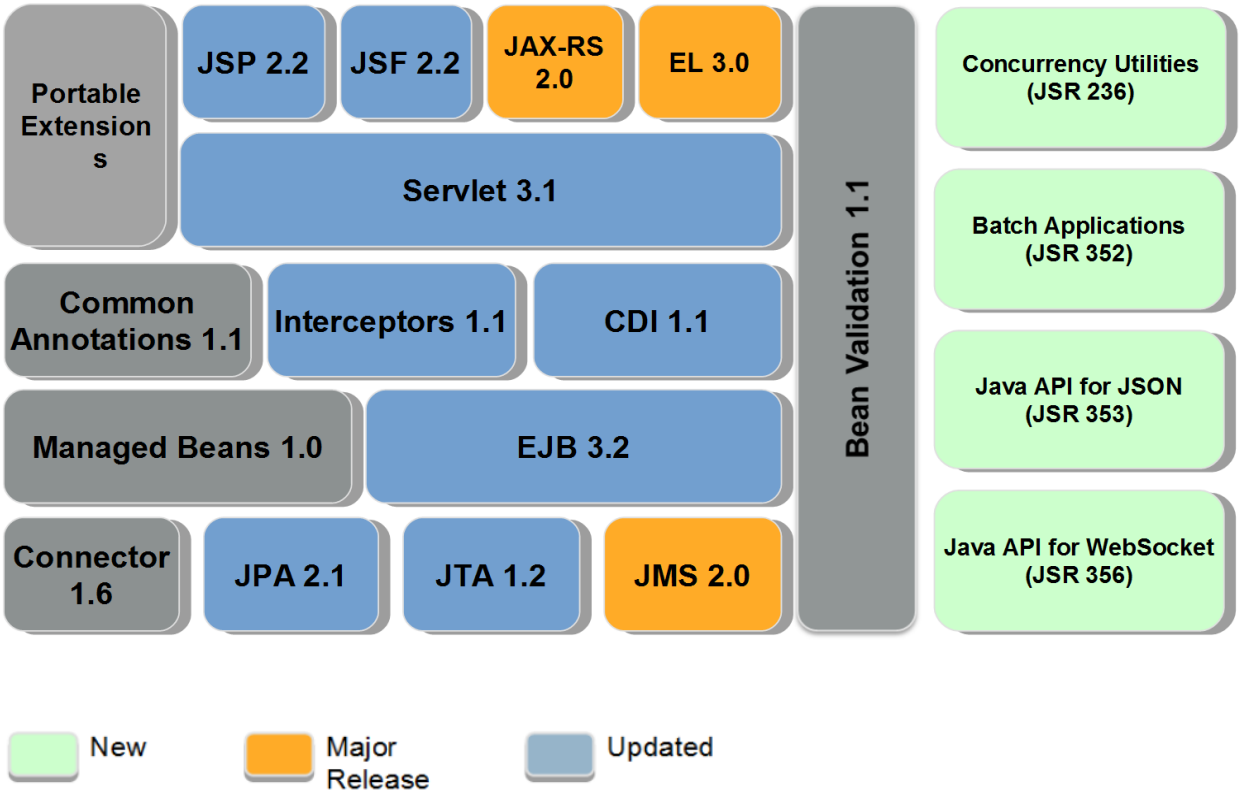




Pour Java EE 7 :

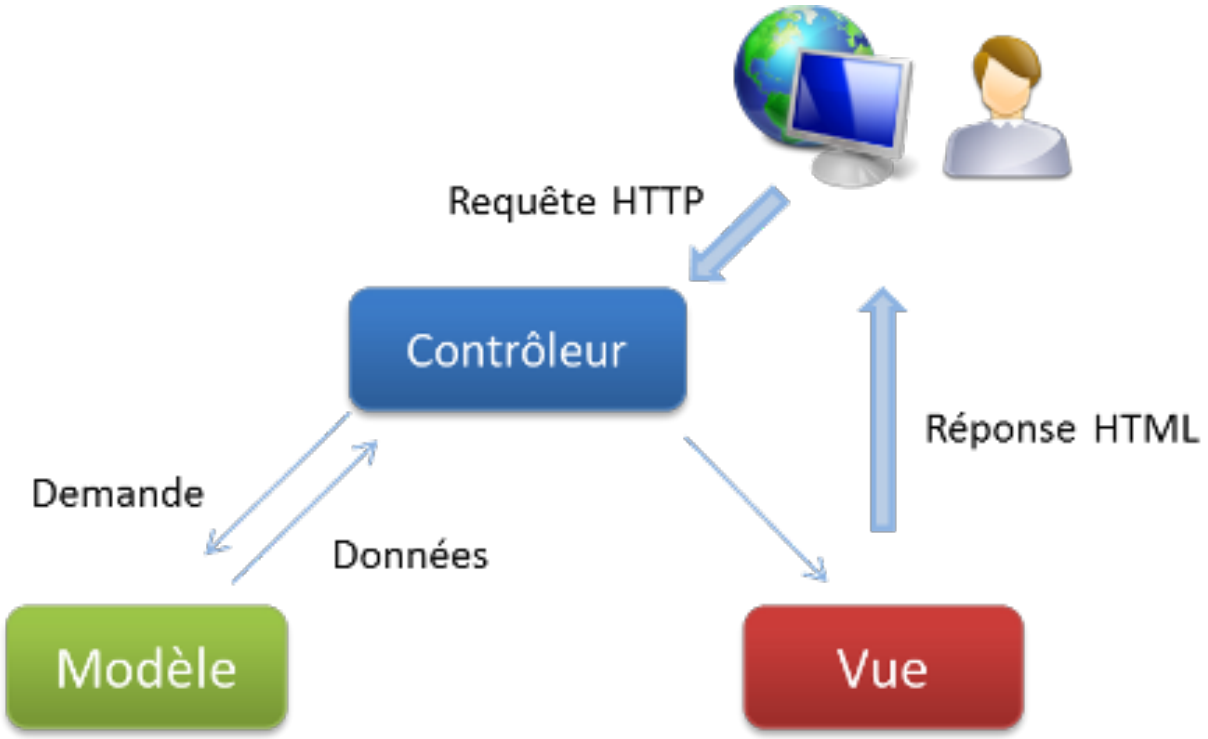


JEE 7 API :



V) Architecture MVC.

Model-View-Controller : désigne la séparation des données, du traitement sur ces données et la manière de les restituer.



1. *Architecture MVC 2*

L'architecture MVC2 consiste à introduction d'un *front controller* qui traite toutes les demandes et les renvoie au bon traitement.

Il est quand même bien de préciser certaines choses :

- MVC2 n'est pas le successeur de MVC.
- MVC2 est plus complexe que MVC.
- MVC2 sépare la logique de la présentation contrairement à MVC.
- MVC2 est plus flexible que MVC.
- MVC2 est plus adapté pour de grosses applications.
- MVC correspond bien à de petites applications.

Voici ci-dessous quelques Framework MVC.

MVC2 Orienté requête

Stripes	spring	Struts 2

MVC2 Orienté composant

JavaServer™ Faces JSF	APACHE WICKET	tapestry

2. *Architecture MVC 2 de Spring MVC*

L'architecture MVC2 de **Spring MVC** peut être schématisée de la façon suivante :