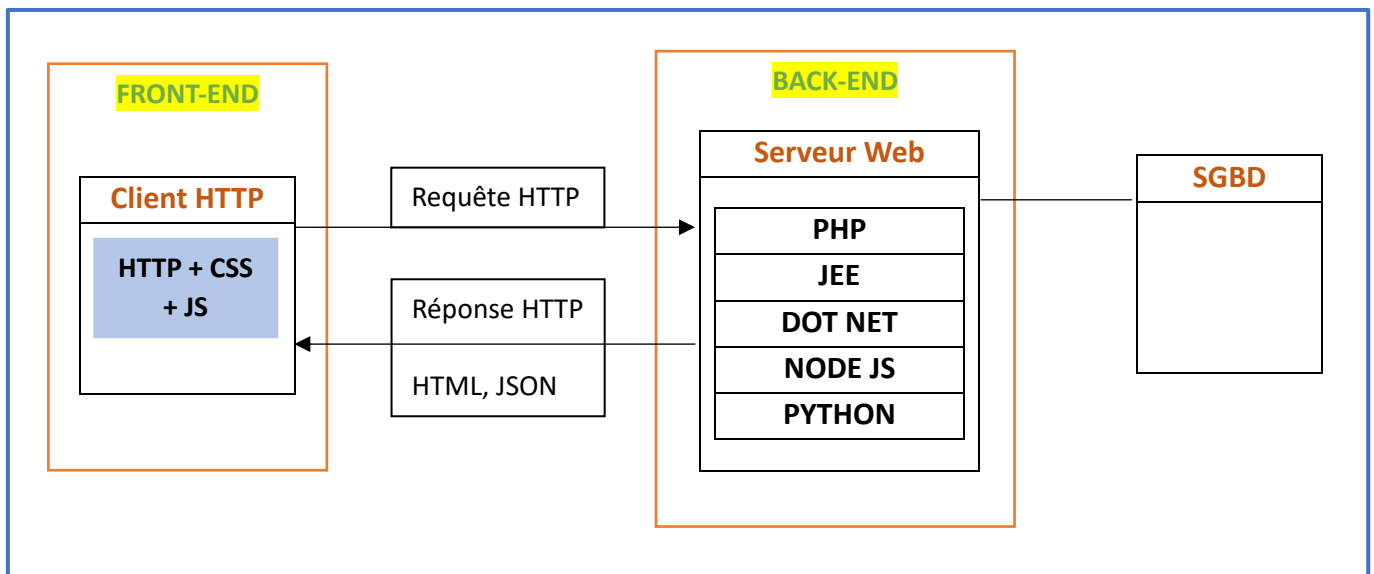


## JEE – Exigences Logicielle

### Architecture web :

- Un client web (Browser) communique avec le serveur web (Apache) en utilisant le protocole HTTP.
- Une app web se compose de deux parties :
  - **La partie Backend (Logic)** : S'occupe des traitements effectués coté serveur.
  - **La partie Frontend (UI)** : S'occupe de la présentation coté client.
- La communication entre la partie front et back se fait via le **protocole HTTP**



### Les exigences d'un projet informatique :

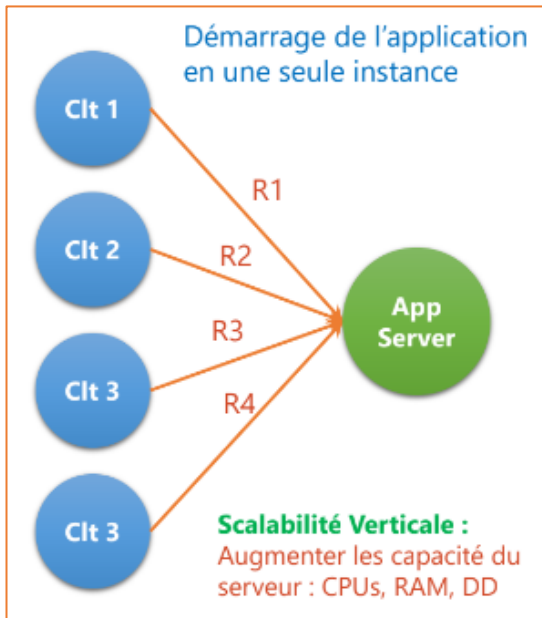
#### ➤ Exigences fonctionnelles :

- Satisfaire les besoins fonctionnels (métiers) demandés par l'entreprise.
- Utiliser UML pour formaliser le besoin de l'app (Use Case Diagram)

#### ➤ Exigences Technique :

##### - Performances :

- Temps de réponse
- Problème de montée en charge : **Scalabilité Verticale et Horizontale**
- Equilibrage de charge et Tolérance aux pannes ( Réplication de l'app dans plusieurs instances)

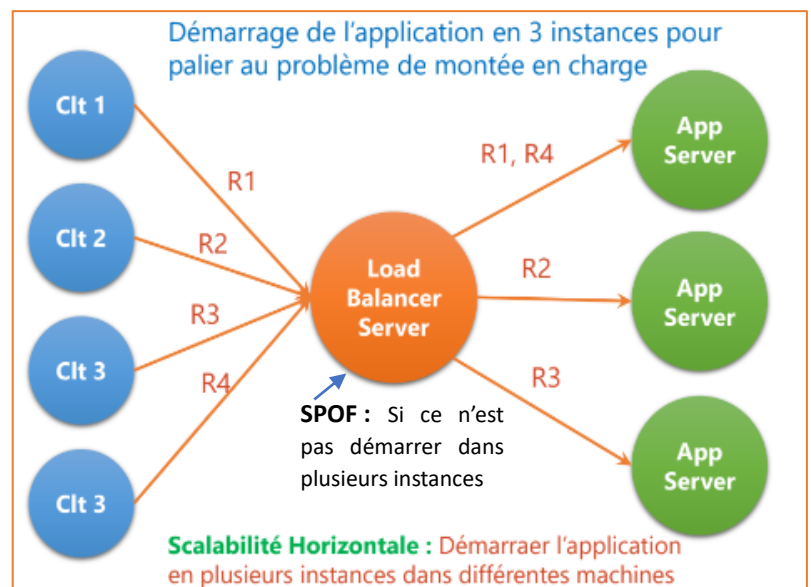


- Toutes les requêtes vont utiliser les mêmes ressources : CPU, RAM
- Beaucoup de tâches vont utiliser par exemple le même CPU → **CPU saturé**
- Beaucoup d'objets s'écrivent dans la RAM → **RAM saturé**

Quant l'utilisateur envoie la requête, il attend beaucoup de temps pour avoir la réponse → **application trop lente (problème de montée en charge)**

- **Scalabilité Verticale :** rester dans la même machine (une seule machine) quand le nombre d'utilisateurs augmente, on va augmenter les capacités du serveur : CPUs, RAM...

- **Architecture distribuée :** Démarrer la même app dans plusieurs machines (instances)
- **Load Balancer :** va répartir la charge sur plusieurs instances, augmenter ou diminuer le nombre d'instances pour palier au problème de montée en charge. (Exemple de l'algorithme : Round Robin)
- **Load Balancer = Gateway = Proxy**
- **Scalabilité Horizontale :** Démarrer l'app dans différentes machines



- Un point de défaillance unique (« **single point of failure** » ou **SPOF** en anglais) correspond à une vulnérabilité d'un système matérialisée par un seul et unique composant. Si le composant fait défaut, le système dans sa totalité échoue

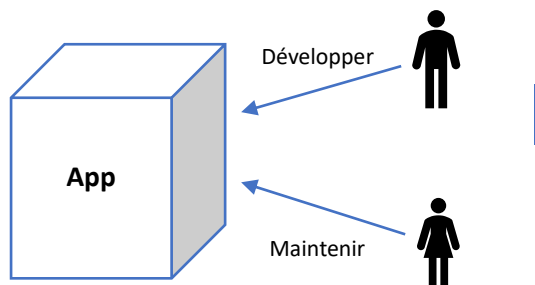
- **Maintenance : l'app doit être facile à maintenir**


- Une app doit évoluer dans le temps
- L'app doit être **fermée à la modification** et **ouverte à l'extension**

On ne touche pas le code source au moment de la maintenance

Au lieu de modifier le code source, on va ajouter des extensions (classes), alors l'app évolue sans modifier le code source

➤ **Comment ?**



- Utiliser le **couplage faible** au lieu de **couplage fort**
- **Dépendre d'interface et non plus de l'implémentation** 

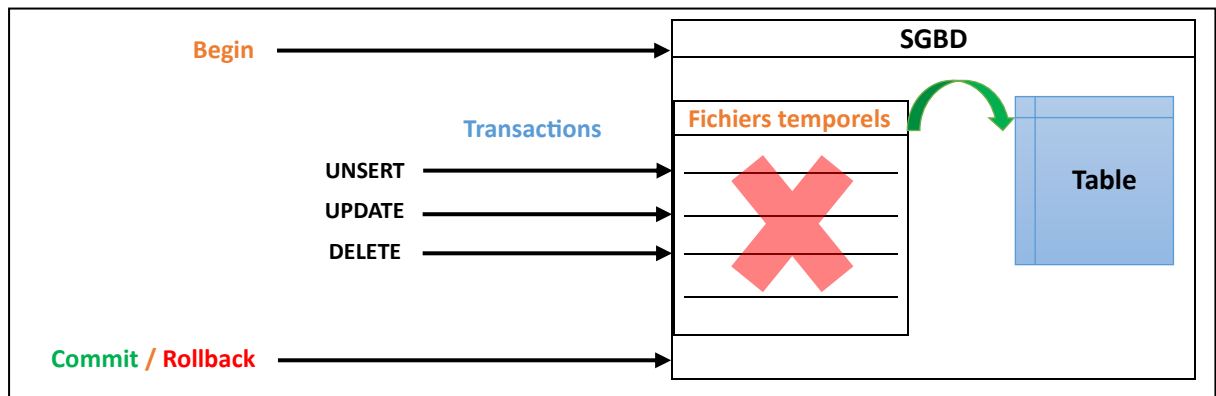
Couplage Fort = App ouverte à la modification	Couplage Faible = App fermée à la modification
<p>⇒ La Classe A est dépend de la classe B</p>	
<p>Après 6 mois de développement de cette app : Au lieu de BD utiliser un web service</p>	
	<p><b>Config.txt</b></p> <pre>- B (Injection des dépendances)</pre> <p>⇒ Le couplage faible : La classe A est prête à utiliser n'importe quelle classe qui implémente l'interface IB ici il va utiliser la classe B qu'on a injecté dans le fichier de configuration.</p>
<p>⇒ Dépendre d'une implémentation (classe)</p>	<p>⇒ Dépendre d'une interface</p>

➤ **Comment savoir qu'on va utiliser B ou B2 ? : [\( Injection des dépendances \) 1 :36](#)**

➤ **Design pattern Adapter utiliser pour ajouter une fonction dans l'interface IB si le métier de l'app a été modifiée.**

- Sécurité : utiliser le Framework Spring Security

- Persistences des données, Gestion des **Transactions**



- ❖ **Commit** : Transférer les données vers les vraies tables si toutes les opérations sont exécutées comme il faut
- ❖ **Rollback** : Annuler la transaction, toutes les données dans les fichiers temporels va être rejeté

- Versions : Web, Mobile, Desktop (logique présentation différente mais le même logique Métier) ➔ **Il faut toujours séparer la logique métier de la logique présentation** ⚠

➤ **Exigences Financière :**

Le cout du logiciel doit respecter les contraintes budgétaires ➔ Utiliser la Baguette magique de l'ingénierie logicielle : **Inversion de Contrôle**