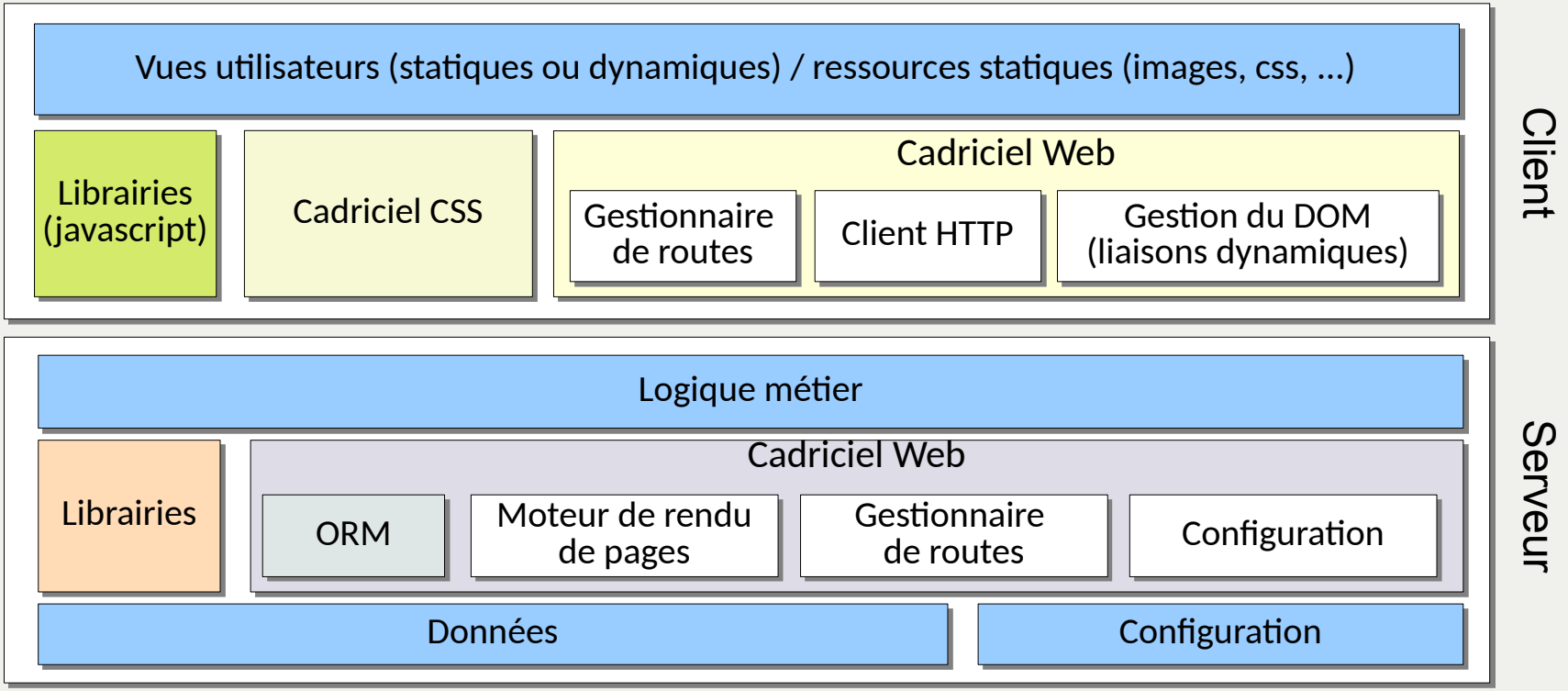
Architecture d'une application Web

Nicolas Le Sommer
Nicolas.Le-Sommer@univ-ubs.fr
Université Bretagne Sud, IUT de Vannes, Département Informatique

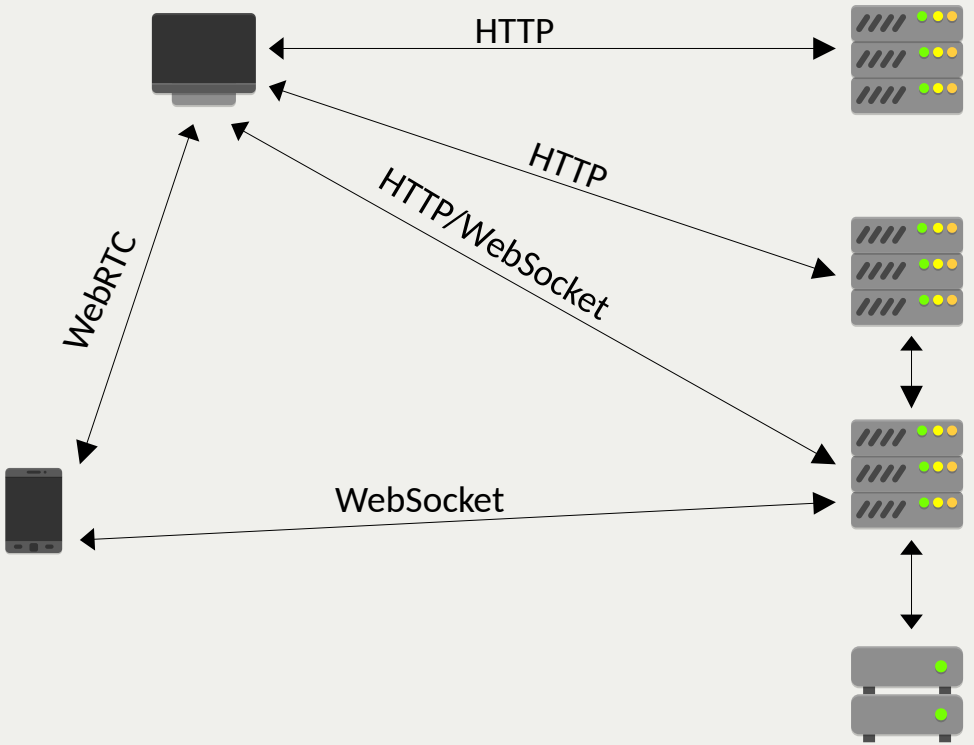
Plan du cours

- Architecture et fonctionnement général d'une application Web et architectures n-tiers
- Exécution de code côté client et côté serveur
- Applications Web avec et sans API REST d'accès aux ressources
- Les patrons Singleton et *Data Access Object* (DA0)

Vue d'ensemble d'une application Web



Environnement d'exécution d'une application Web



CDN (*Content Delivery Network*) hébergeant des cadriels et librairies utilisées par la partie cliente de l'application

Serveur Web étendu hébergeant des services Web utilisés par les parties cliente et serveur de l'application

Serveur Web étendu hébergeant l'application

Serveur de base de données assurant la persistance des données de l'application

Application Web et Architecture *n*-tiers

- Une application Web est une application distribuée reposant sur une architecture *n*-tiers
  - le tiers fait référence non pas à une entité physique mais logique
- Architecture 3-tiers = architecture composée de 3 entités logiques :
  - Tiers de présentation (client léger),
  - Tiers de traitement (serveur étendu),
  - Tiers de persistance des données (e.g. serveur de base de données)

Architecture 3-tiers : tiers de présentation

- Le tiers de présentation est un navigateur Web
- Le tiers de présentation
  - affiche les interfaces utilisateurs (i.e. pages Web et contenus associés)
  - réagit aux actions de l'utilisateur
  - émet des requêtes vers le(s) serveur(s) et traite les réponses
  - effectue (peu) de traitements

Architecture 3-tiers : tiers de traitement

- Le tiers de traitement est un serveur Web étendu
- Le tiers de traitement se caractérise notamment par sa capacité
  - à répondre à plusieurs clients simultanément
    - Serveur « multi-threadé »
  - à traiter des requêtes et à émettre des réponses HTTP
  - à traiter éventuellement des appels de méthodes à distance
    - Protocoles SOAP et XMLRPC
- Le tiers de traitement
  - effectue la majorité des calculs
  - peut adresser des requêtes vers d'autres serveurs additionnels (e.g., vers des SGBD).

Architecture 3-tiers : tiers de données

- Le tiers de données permet de
  - manipuler les données sous la forme objets
  - persister les objets dans une base de données, dans des fichiers, ...
- L'accès au tiers de données peut s'effectuer à des travers des bibliothèques logicielles de type *Object Relational Mapping* (ORM)
  - Exemples :
    - Java : JPA (Java Persistence API), Hibernate, Java Data Object
    - PHP: Doctrine
- Différents motifs d'accès aux données peuvent être utilisés pour mettre en œuvre les ORMs
  - Active Record, Data Access Object, ...

Exécution côté client *vs* côté serveur

Exécution de code côté client *vs* côté serveur

- Le code exécuté côté client est
  - défini ou référencé (i.e. lien hypertexte vers un fichier contenant le code) par une balise `script` HTML.
  - transmis par un serveur (ou un CDN) pour exécution
  - exécuté suite à un évènement
    - utilisateur
      - clic de souris, entrée clavier, …
    - du navigateur
      - après le chargement d'une page, redimensionnement d'une page, …

Exécution de code côté client *vs* côté serveur

- Le code exécuté côté serveur
  - est contenu dans des fichiers ayant une extension spécifique (e.g. « .php »)
    - Le contenu du fichier peut être un mélange de code HTML et de code à interpréter/exécuter par le serveur (php, python, javascript, ...)
  - est interprété/exécuté par le serveur Web afin de générer la réponse à la requête demandée par un client HTTP.
  - n'apparaît jamais dans les pages transmises aux clients ; c'est le résultat de l'exécution qui est transmis au client

Exemple de code exécuté côté client

Ouvrir + test.html /tmp Enregistrer

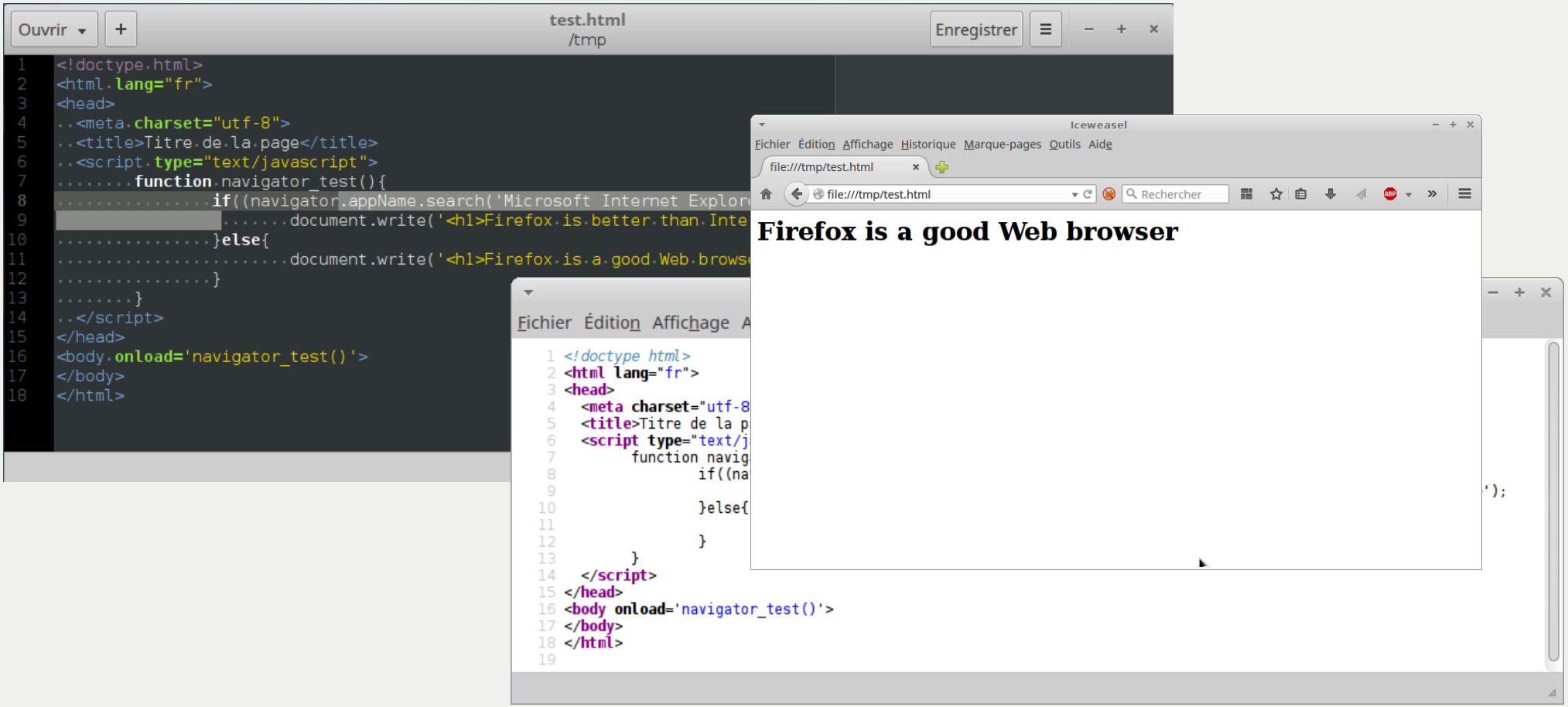
```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Titre de la page</title>
  <script type="text/javascript">
    function navigator_test(){
      if((navigator.appName.search('Microsoft Internet Explorer') == 1)){
        document.write('<h1>Firefox is better than Internet Explorer. Please use Firefox</h1>');
      }else{
        document.write('<h1>Firefox is a good Web browser.</h1>');
      }
    }
  </script>
</head>
<body onload='navigator_test()'>
</body>
</html>
```

Source de : file:///tmp/test.html - Iceweasel

Fichier Édition Affichage Aide

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Titre de la page</title>
  <script type="text/javascript">
        function navigator_test(){
                if((navigator.appName.search('Microsoft Internet Explorer') == 1)){
                        document.write('<h1>Firefox is better than Internet Explorer. Please use Firefox</h1>');
                }else{
                        document.write('<h1>Firefox is a good Web browser </h1>');
                }
        }
  </script>
</head>
<body onload='navigator_test()'>
</body>
</html>
```

Exemple de code exécuté côté client



Exemple de code exécuté côté serveur

Ouvrir + test.php /var/www Enregistrer

```php
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Titre de la page</title>

</head>
<body>
    <?php
        if (strstr($_SERVER['HTTP_USER_AGENT'], 'Firefox')){
            echo ('<h1>Firefox is a good Web browser!');
        }else{
            echo ('<h1>Please use Firefox !!');
        }
    ?>
</body>
</html>
```

PHP Largeur des tabulations : 8 Lig 17, Col 8

Source de : http://localhost/test.php - Iceweasel

Fichier Édition Affichage Aide

```html
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Titre de la page</title>

</head>
<body>
        <h1>Firefox is a good Web browser!</body>
</html>
```

Titre de la page - Iceweasel

Titre de la page

localhost/test.php

Firefox is a good Web browser!

Applications Web avec et sans API REST d'accès aux ressources

Modèle architectural REST

- Les applications développées selon le modèle architectural REST (*REpresentational State Transfer*) doivent respecter 6 contraintes.
- Contrainte 1 : « client/serveur »
  - séparation des responsabilités entre le client et le serveur.
    - Client : présentation
    - Serveur : sauvegarde et accès des données
- Contrainte 2 : « serveur sans états »
  - chaque requête d'un client contient toute l'information nécessaire pour permettre au serveur de traiter la requête
- Contrainte 3 : « cache »
  - chaque réponse d'un serveur contient une information indiquant si la réponse peut ou non être mise en cache côté client ou côté serveur.
  - Si une réponse peut être mise en cache, elle peut être réutilisée pour des requêtes ultérieures équivalentes.

Modèle architectural REST

- Contrainte 4 : « interface uniforme »
  - Identification unique des ressources
  - Manipulation des ressources par leur représentation
    - Les représentations des ressources permettent de les créer, les modifier ou les supprimer
  - Permet de découpler le client du serveur
- Contrainte 5 : « système en couche »
  - Le client ne doit pas savoir s'il est connecté directement au serveur
  - Des serveurs intermédiaires peuvent être ajoutés pour des raisons de performances et de passage à l'échelle
- Contrainte 6 (optionnelle): « code à la demande »
  - Les serveurs peuvent étendre une fonctionnalité en transférant du code au client (e.g., Javascript)

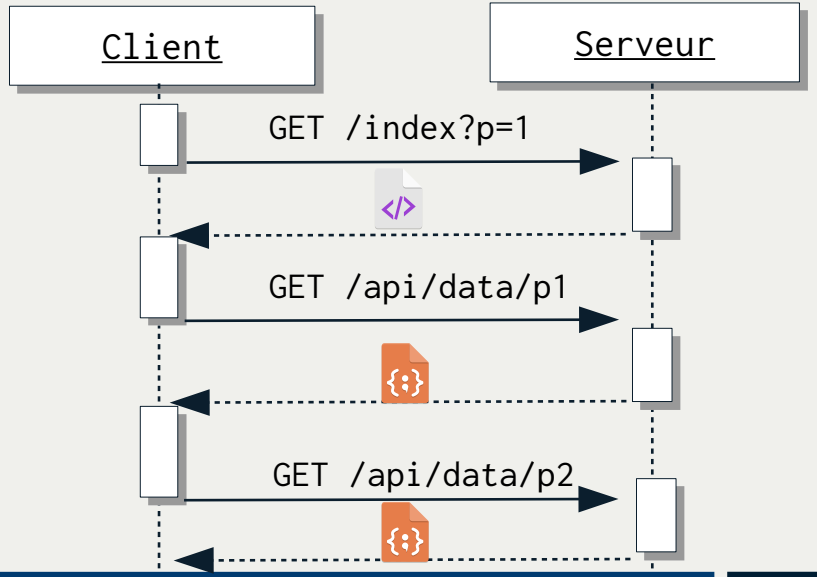
REST, HTTP et Opération « CRUD »

- Dans le modèle REST, les ressources sont identifiées par des URI
- On utilise les méthodes HTTP pour effectuer les opérations élémentaires de création, de consultation, de mise-à-jour et de suppression

Méthode HTTP	Opération CRUD
POST	Create
GET	Read
PUT	Update
DELETE	Delete

Avantage de l'approche

- Découplage complet entre le client et le serveur
- Le serveur retourne
  - du contenu statique (pages HTML, images, code CSS, ...)
  - des données (souvent au format JSON)
  - du code Javascript pour mettre en forme les données et communiquer avec l'application Web via son API REST
- Pas d'état à sauvegarder côté serveur
- Passage à l'échelle plus facile



Exemple d'API REST

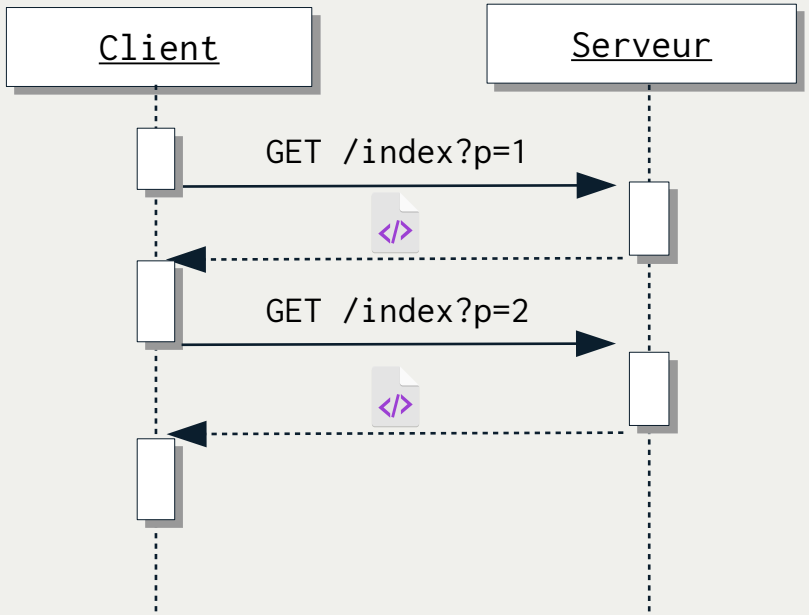
Méthode HTTP	URI	Action
GET	/employees	Retourne la liste des employés
POST	/employees	Ajoute un employé à la liste
PUT	/employees	Méthode non autorisée
PATCH	/employees	Méthode non autorisée
DELETE	/employees	Supprime la liste des employés
PUT	*/employees*/employeeID	Modifie les informations de l'employé identifié par employeeID ou le créé
PATCH	*/employees*/employeeID	Modifie les informations de l'employé identifié par employeeID
POST	*/employees*/employeeID	Méthode non autorisée
GET	*/employees*/employeeID	Retourne les informations de l'employé identifié par employeeID
DELETE	*/employees*/employeeID	Supprime l'employé identifié par employeeID

Exemple d'application REST

```
<!doctype html>
<html lang="fr">
<head> <meta charset="utf-8"> </header>
 <body>
    <table border=1 id="tab"></table>
    <script>
      var tab = document.getElementById('tab');
      var xhr = new XMLHttpRequest();
      xhr.open("GET", "http://localhost:8000/employees");
      xhr.onreadystatechange = function () {
        if (this.readyState == 4) {
           let thead = "<tr><th>Nom</th><th>Prénom</th></tr>";
          let jsonResult = JSON.parse(this.responseText);
          let tbody="";
          for(let i=0; i < jsonResult.length; i++){
            tbody+= "<tr><td>"+jsonResult[i].lastname+"</td><td>"+jsonResult[i].firstname+"</td></tr>";
          }
          tab.innerHTML= thead+tbody;
        }
      };
      xhr.send(null);
    </script>
 </body>
</html>
```

Application Web avec interface générée côté serveur

- Pas de découplage et de formalisation des interactions entre la partie cliente et la partie serveur
- Le serveur retourne au(x) client(s) des interfaces utilisateurs contenant les données
  - Pas de mise-à-jour dynamique côté client du DOM de la page reçue par le serveur pour afficher des données
- Cette approche
  - permet d'exécuter le traitement côté serveur
  - allège la charge du client
  - est adaptée aux terminaux ayant peu de ressources



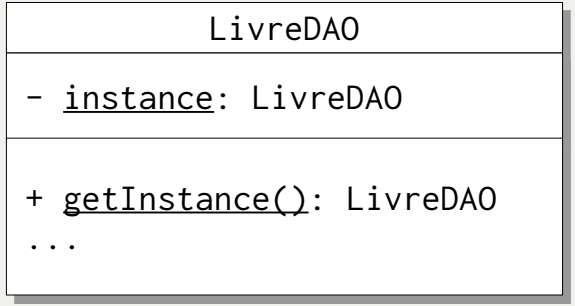
Exemple

```
<!doctype html>
<html lang="fr">
<head>...</header>
 <body>
 <table border="1">
 <tr><th>Nom</th><th>Prénom</th></tr>
<?php
$user='admin'; $pass='admin_password';
try {
      $connection = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
      $sqlQuery = 'SELECT firstname, lastname FROM Employee ORDER BY lastname';
      $result = $connection->query($sqlQuery)->fetchAll();
      foreach ($result as $row) {
         echo "<tr><td>".$row['firstname']."</td><td>".$row['lastname']."</td></tr>" ;
    }
} catch (PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br/>";
    die();
}
?>
 </table>
 </body>
</html>
```

Les patrons Singleton et *Data Access Object* (DA0)

Le patron de conception Singleton

- Singleton : patron de conception permettant de créer une unique instance d'une classe
- Patron utilisé pour développer le modèle d'accès aux données « DataAccessObject »




```php
<?php
class LivreDAO {
  private static $instance;

  // le constructeur est privé afin que l'on ne puisse
  // pas créer d'instances via le mot clé new
  private function __construct() { }

  // méthode statique permettant de retourner une unique
  // instance de la classe
  public final static function getInstance() {
    if(!isset(self::$instance)) {
      self::$instance= new LivreDAO();
    }
    return self::$instance;
  }
}
?>
```

Le patron d'accès aux données *Data Access Object* (DAO) : présentation

- Le patron de conception DAO associe à chaque classe modélisant un type de données (i.e. une table) une classe de gestion et d'accès aux données.
- La classe de gestion est définie comme un singleton et met en œuvre
  - des méthodes de suppression et de modification retournant un booléen indiquant le succès ou l'échec de l'opération
  - une méthode d'insertion retournant l'identifiant de la nouvelle ligne de la table (utile en cas d'identifiant auto-incrémenté par le SGBD),
  - plusieurs méthodes permettant d'obtenir un objet en fonction de différents critères de recherche.

Le patron d'accès aux données *Data Access Object* (DAO) : exemple