



Applications WEB

Applications Web

- + Le modèle MVC
- + Persistance
- + Frameworks, Services
- + PHP, Symfony
- + J2EE – Spring
- + Angular, React, VueJS

Applications WEB

- + Souvent les applications sont décomposées en 2 parties :
 - + Front End
 - La partie visible, celle qui se passe sur le navigateur
 - + HTML, CSS, Javascript, Ajax
 - + Back End
 - Ce qui se passe au niveau du serveur
 - + Application Web
 - + Modèle MVC
 - + Persistance
 - + Base de données
- + Full Stack = celui qui sait faire les 2.

Quelques éléments de sécurité

- + Opération sensible => identification
- + Gestion Login / Mot de passe
- + Chiffrement : md5, SHA256 ou +
- + Identifiants de session
- + Éléments sensibles (paiement, ...) = HTTPS
- + Injections SQL, XSS (cross-scripting)

Quelques éléments de présentation

- + Pensez aux bibliothèques
 - + Jquery (un classique)
 - + Bootstrap (présentation)
 - + Datatable (présentation des tableaux)
 - + Materialize
 - + ...

Quelques éléments de présentation

- + Pensez aux autres
 - + Application Static / Fluide / Adaptative / Responsive
Comment vous adaptez-vous à l'écran ?
 - + Static = données fixes
 - + Fluide = données en pourcentage, ...
 - + Adaptative = Static, mais dépendant de la taille de l'écran
 - + Responsive = la présentation change en fonction de l'écran
 - + Adaptation aux personnes ayant un handicap
 - + <https://www.w3.org/Translations/WCAG20-fr/>
 - + <http://www.aphvbsl.org/site-accessible>

Modele MVC



Modele MVC

+ Architecture Client / Serveur

+ Les rôles :

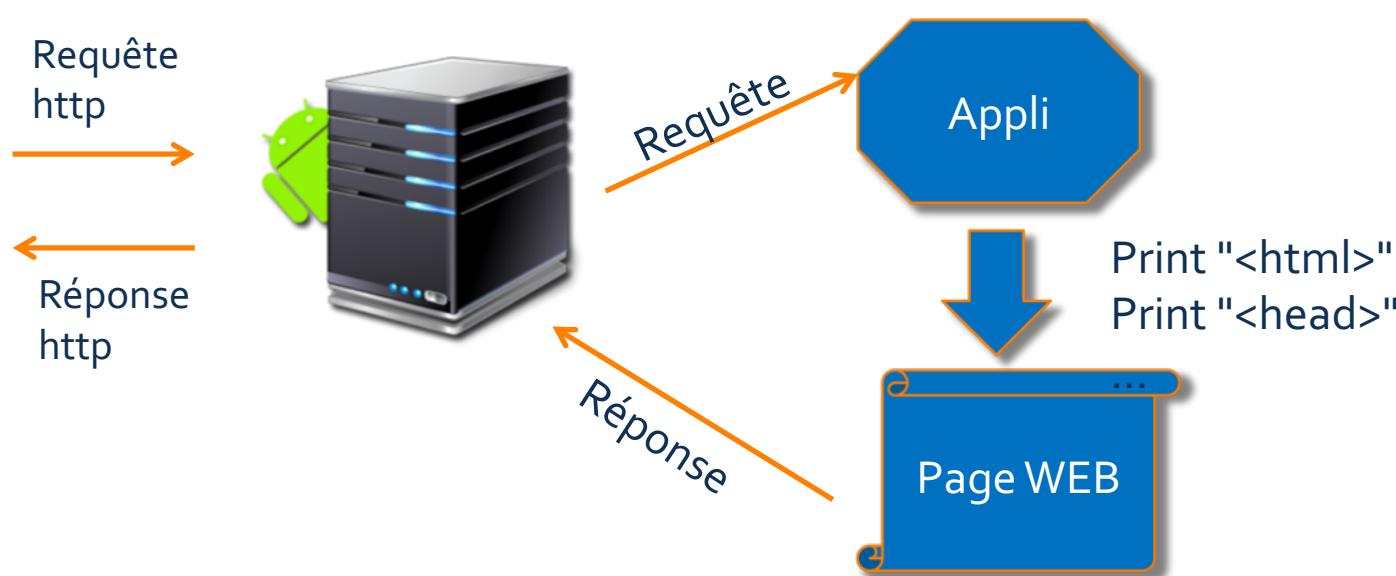
- + Un serveur sur lequel tournent des applications dédiées
Cohérence des données, gros traitements, ...
- + Des clients sur lesquels tournent des applications spécifiques
IHM, traitements locaux
- + Middleware : assure le dialogue entre les deux composants

+ Remarques

- + A chaque installation de poste, il faut installer le logiciel spécifique
- + A chaque mise à jour du logiciel spécifique, il faut mettre à jour tous les postes
- + 2 applications à développer (le serveur et le client)

Modèle MVC

+ Application web classique

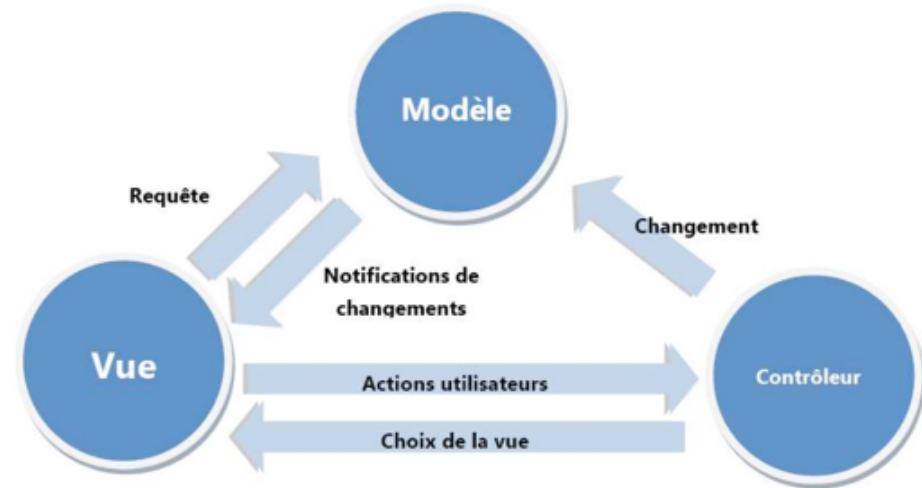


Modele MVC

- + Mais...
 - + Changement de design => changer l'application ?
 - + Changement de modèle de données
=> changement de l'affichage ?
 - + ...
- + Problématique
 - + Comment dissocier données et affichage de manière à travailler en modules indépendants
- + Objectif
 - + Faciliter le développement
 - + Faciliter les mises à jour

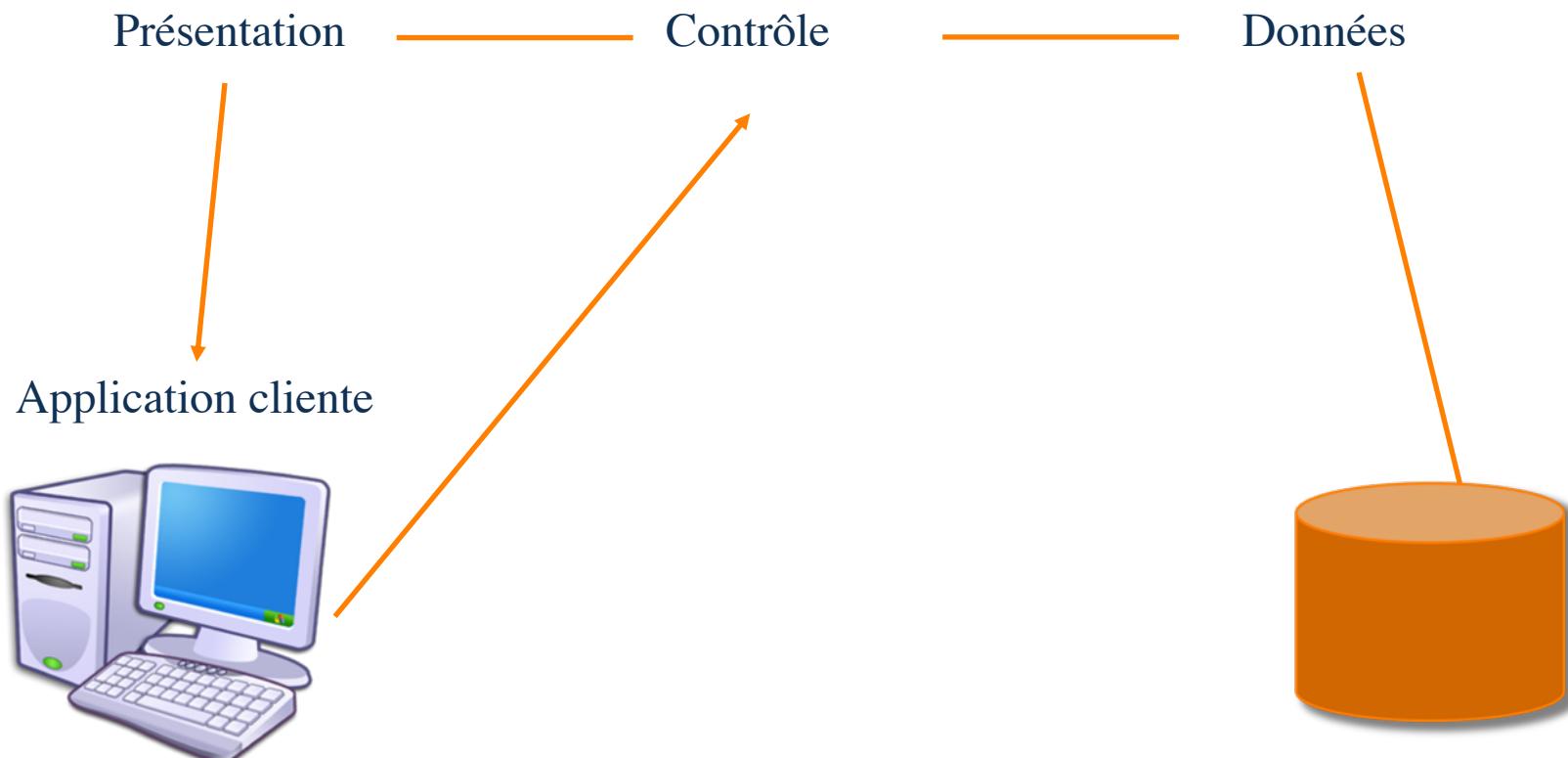
Modele MVC

- + Structuration de l'application en plusieurs parties
 - + IHM
 - + Traitement
 - + Gestion des données



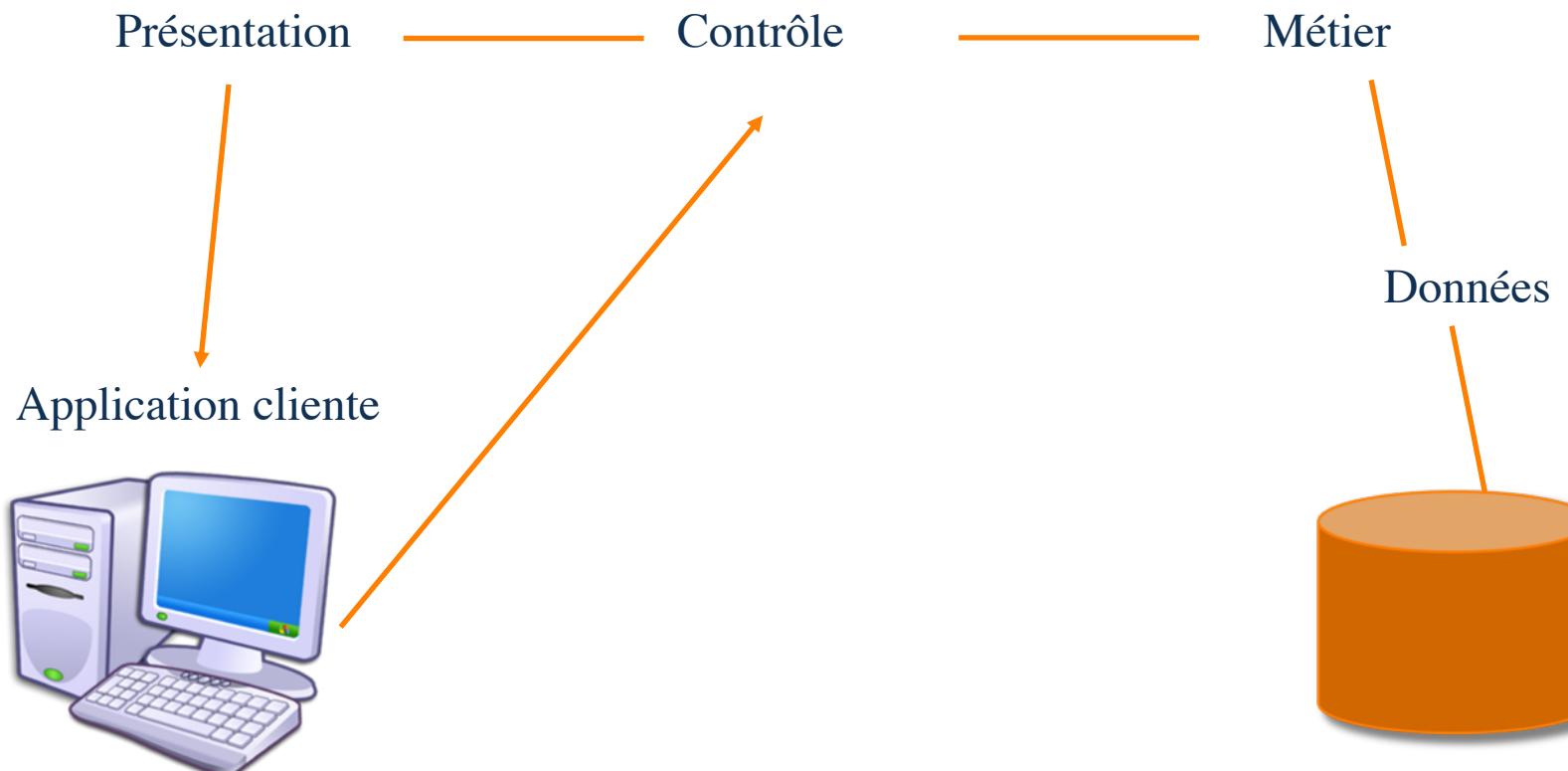
Vers le Multi-tiers

- + Structuration de l'application en plusieurs parties
Architectures 3-Tiers



Vers le Multi-tiers

+ Architecture 4-Tiers



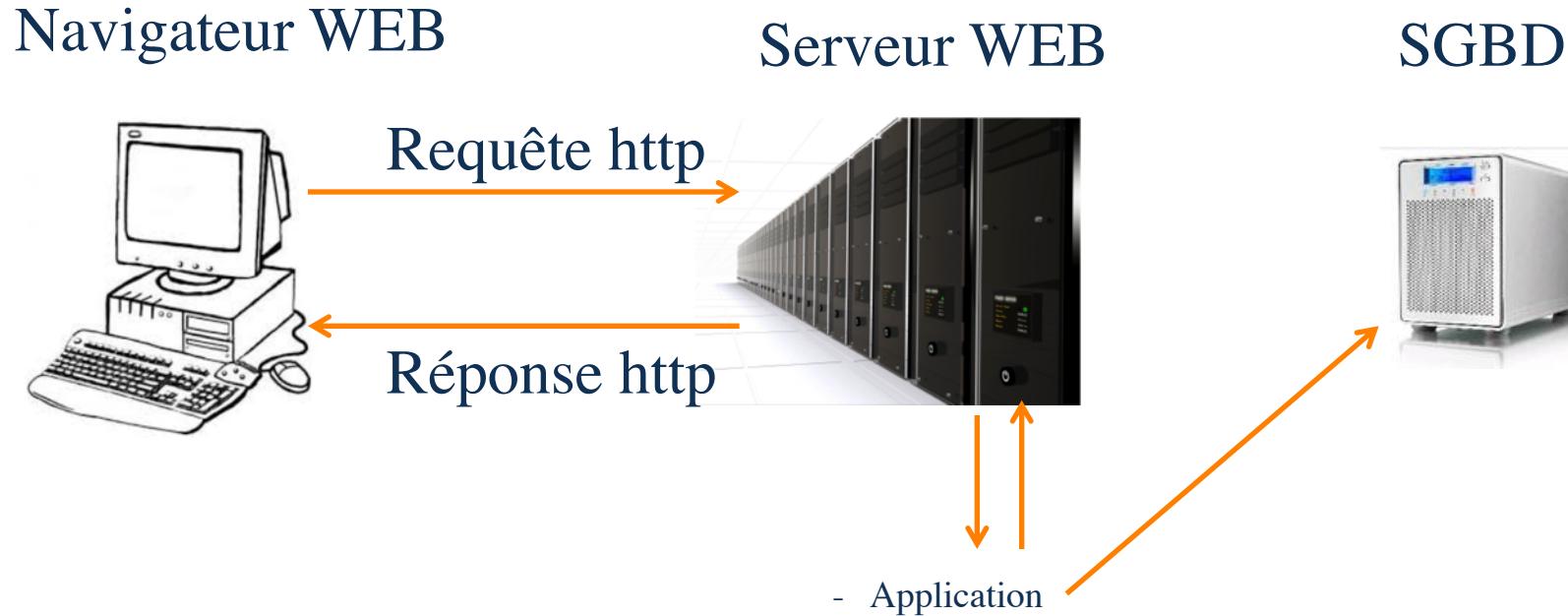
Le modele MVC

- + Pourquoi ?
 - + Chaque composant gère un aspect du problème
 - + Programmation plus souple, plus simple, plus facile à maintenir
 - + Implémenté dans quasiment tous les Frameworks

Notions de Persistance

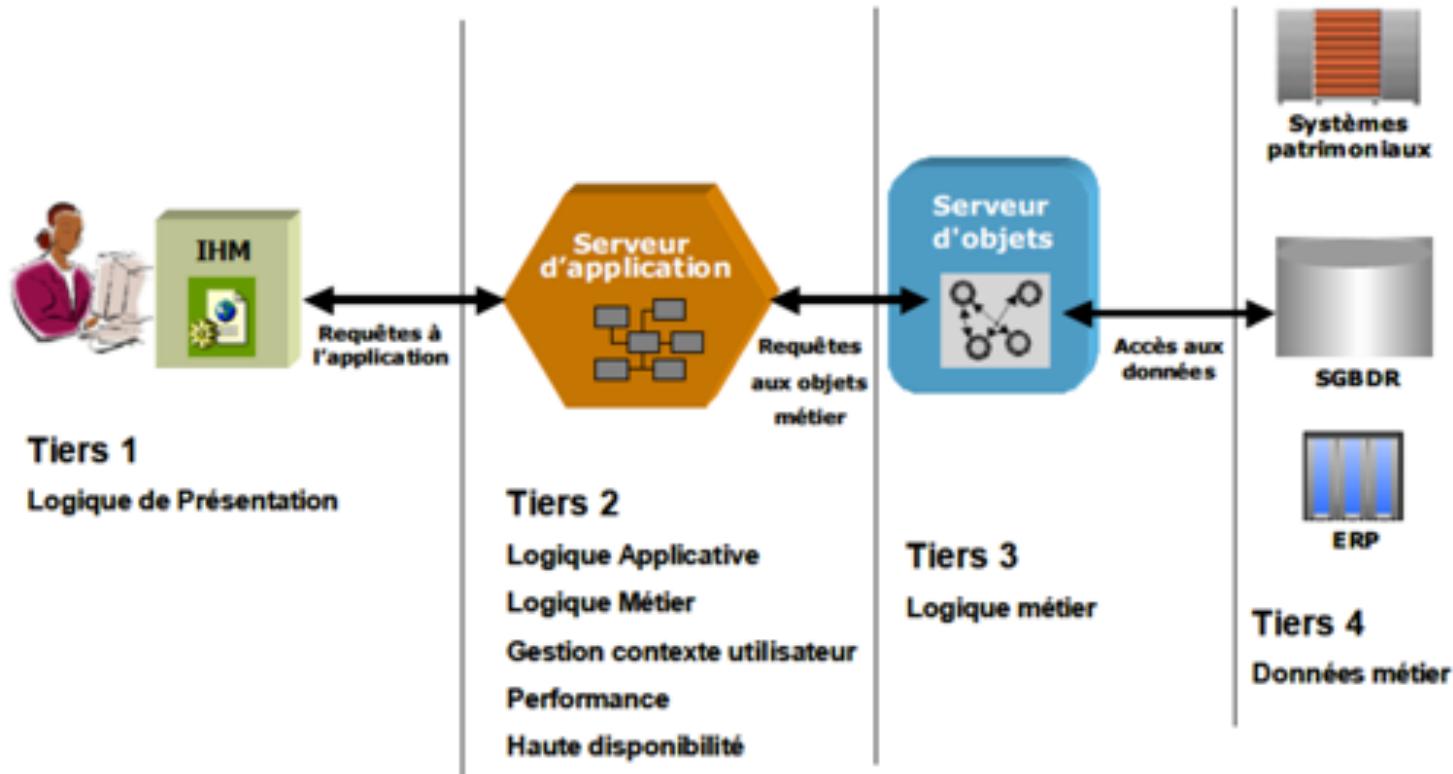


Persistante



Persistante

+ Architecture n-tiers



Persistiance

- + Problème : les objets métiers
 - + 1 objet métier ≠ une table dans une base de données
 - + 2 solutions
 - + Reconstruire l'objet métier à chaque fois qu'on le demande
 - + Mémoriser l'objet métier sur le serveur
 - + S'il n'existe pas : requêtes à la base de données
 - + S'il existe, on utilise l'objet mémorisé
 - + Problème : la gestion de l'objet

Persistiance

- + Mémoriser l'objet en local
 - + Bases de données objet
Fiabilité ?
 - + Sérialisation
Objet = flux linéaire enregistrable (binaire – xml)
 - + ORM : Object Relationnal Mapping
Utiliser une base de données relationnelle pour mémoriser les informations. L'ORM fait le lien entre les deux.

Persistance

- + La persistance c'est
 - + Un mécanisme de mémorisation
 - + Une logique de persistance
 - + Couche de persistance
 - + Moteur de persistance (API) : CRUD
 - + Outil de projection
 - Transformer les objets en mémoire pour les mémoriser physiquement

Persistance

- + Persistance transparente = Persistance orthogonale
 - + Découplage total entre l'application et l'infrastructure de persistance
 - + Nécessite :
 - + Orthogonalité vis-à-vis des types de données
 - + Tout objet doit pouvoir persister
 - + Tout objet doit pouvoir être temporaire, ou transitoire
 - + Transparence de l'utilisation des objets persistant et transitoires
 - + Persistance transitive : tout objet référencé à partir d'un objet persistant devient persistant
 - + Persistance par héritage : si une classe est persistante, toute classe en héritant devient persistante.

Persistiance

- + Le serveur d'Objets
 - + Gestion du cycle de vie des objets
création, recherche, manipulation, destruction des objets.
 - + Gestion de la persistance
 - + Gestion des transactions
Intégrité des données, accès concurrents
 - + Gestion de la sécurité
 - + Gestion de la montée en charge
 - + Gestion de la distribution, du clustering, de la reprise sur pannes et de la transparence de localisation des objets

Persistiance

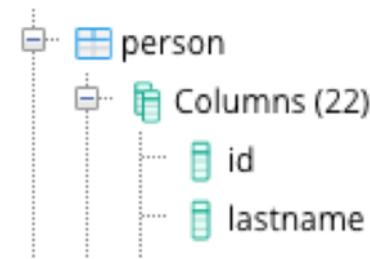
- + Les ORMs
 - + Plusieurs bibliothèques
 - + PHP : Doctrine, Propel
 - + Java : EJB, Hibernate, JPA
 - + ...
 - + 2 façons de fonctionner
 - + Application => structure de la base de données
 - + Base de données => classes de l'application

Persistance

+ Exemple : Doctrine sous Symfony

```
/**  
 * @ORM\HasLifecycleCallbacks()  
 * @ORM\Entity(repositoryClass="App\Repository\PersonRepository")  
 */  
class Person implements UserInterface  
{  
    /**  
     * @ORM\Id()  
     * @ORM\GeneratedValue()  
     * @ORM\Column(type="integer")  
     */  
    private $id;  
  
    /**  
     * @ORM\Column(type="string", length=255)  
     * @Assert\NotNull(message="Lastname field cannot be empty")  
     * @Assert\Length(max="255")  
     */  
    private $lastname;  
    ...  
}
```

```
/**  
 * @method Person|null find($id, $lockMode = null, $lockVersion = null)  
 * @method Person|null findOneBy(array $criteria, array $orderBy = null)  
 * @method Person[]    findAll()  
 * @method Person[]    findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null)  
 */  
class PersonRepository extends ServiceEntityRepository  
{  
    public function __construct(RegistryInterface $registry)  
    {  
        parent::__construct($registry, Person::class);  
    }  
  
    /**  
     * @return Person[] Returns an array of Person objects  
     */  
    public function findByEmail($value)
```



Persistance

+ Exemple : JPA sous Spring



```
@Entity
@Table(name = "personne")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Personne.findAll", query = "SELECT
        , @NamedQuery(name = "Personne.findByPersonneId", query =
        , @NamedQuery(name = "Personne.findByPersonneTitre",
        , @NamedQuery(name = "Personne.findByPersonneNomofficiel"
    )})
public class Personne implements Serializable {

    // ...
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "personne_id")
    private Integer personneId;
    @Column(name = "personne_titre")
    private String personneTitre;
    @Basic(optional = false)
    @Column(name = "personne_nomofficiel")
```

Persistiance

+ Fonctionnement

- + Faire persister un objet =
 - + enregistrer l'objet
 - + Insérer les éléments dans la base de données
- + Arrêter de faire persister =
 - + Supprimer l'objet
 - + Supprimer les éléments dans la base de données
- + Enregistrer un objet =
 - + mettre à jour l'objet
 - + Enregistrer les éléments dans les tables
- + Rechercher un objet =
 - + Find sur l'objet
- + Evaluation paresseuse (les requêtes sont faites quand on en a besoin)

Les Frameworks



Frameworks

- + Développer une application Web
 - + Pourquoi refaire tout à chaque fois ?
- + Utilisation d'environnement permettant de faciliter le travail
 - + Prise en charge des principales fonctionnalités
 - + Sécurité
 - + Gestion des accès à la base de données
 - + Modèle MVC
 - + ...

Frameworks

- + Développement -> Utilisation d'API Application Programming Interface
 - + Ensemble de fonctions permettant de mettre en œuvre les développements
 - + En général orienté vers des sujets précis

Frameworks

+ Framework

- + Ensemble d'outils permettant de mettre en œuvre une application
- + Avantage
 - + Fiabilité
 - + Prend en charge de nombreux aspects (optimisation, sécurité, ...)
 - + Evolue régulièrement
- + Inconvénient
 - + Attention aux compatibilités des mises à jour
 - + Parfois lourd

Services



Notions de services

- + Application = réponse à des sollicitations
- + Idée : développement sous forme de briques fonctionnelles indépendantes = service

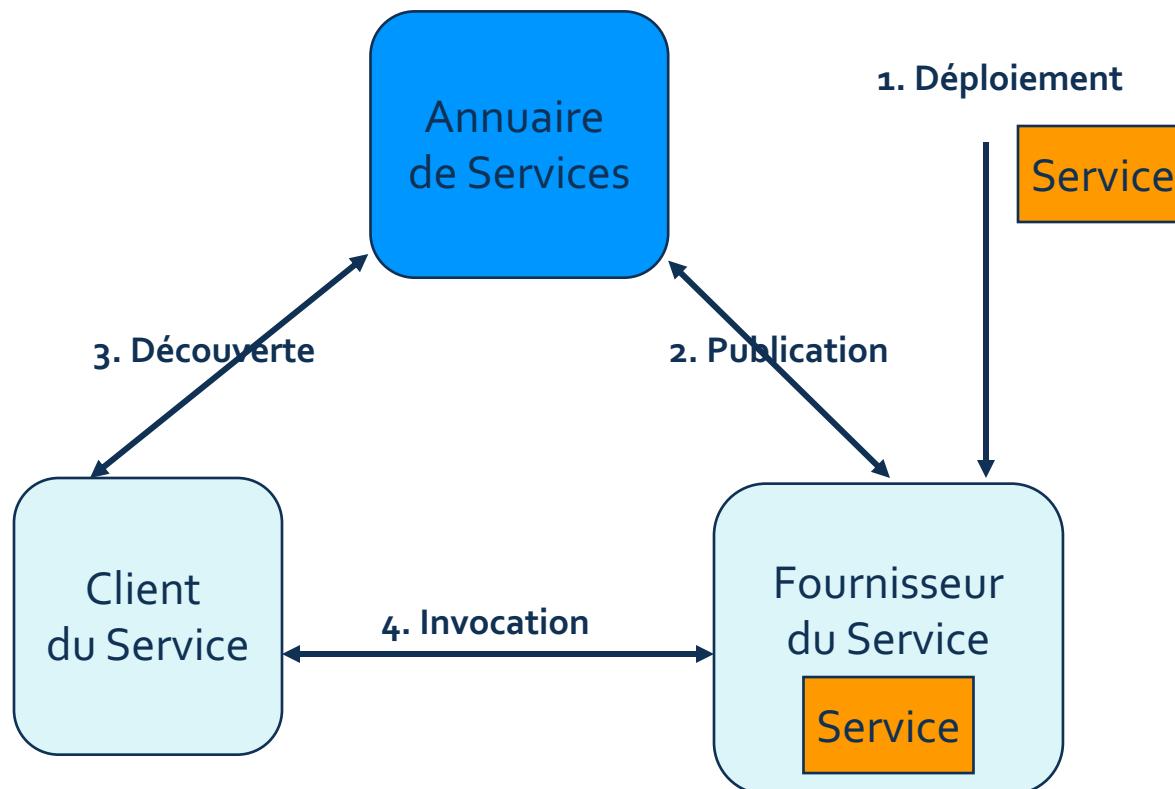
Service accessible via le web = web-service

Notions de services

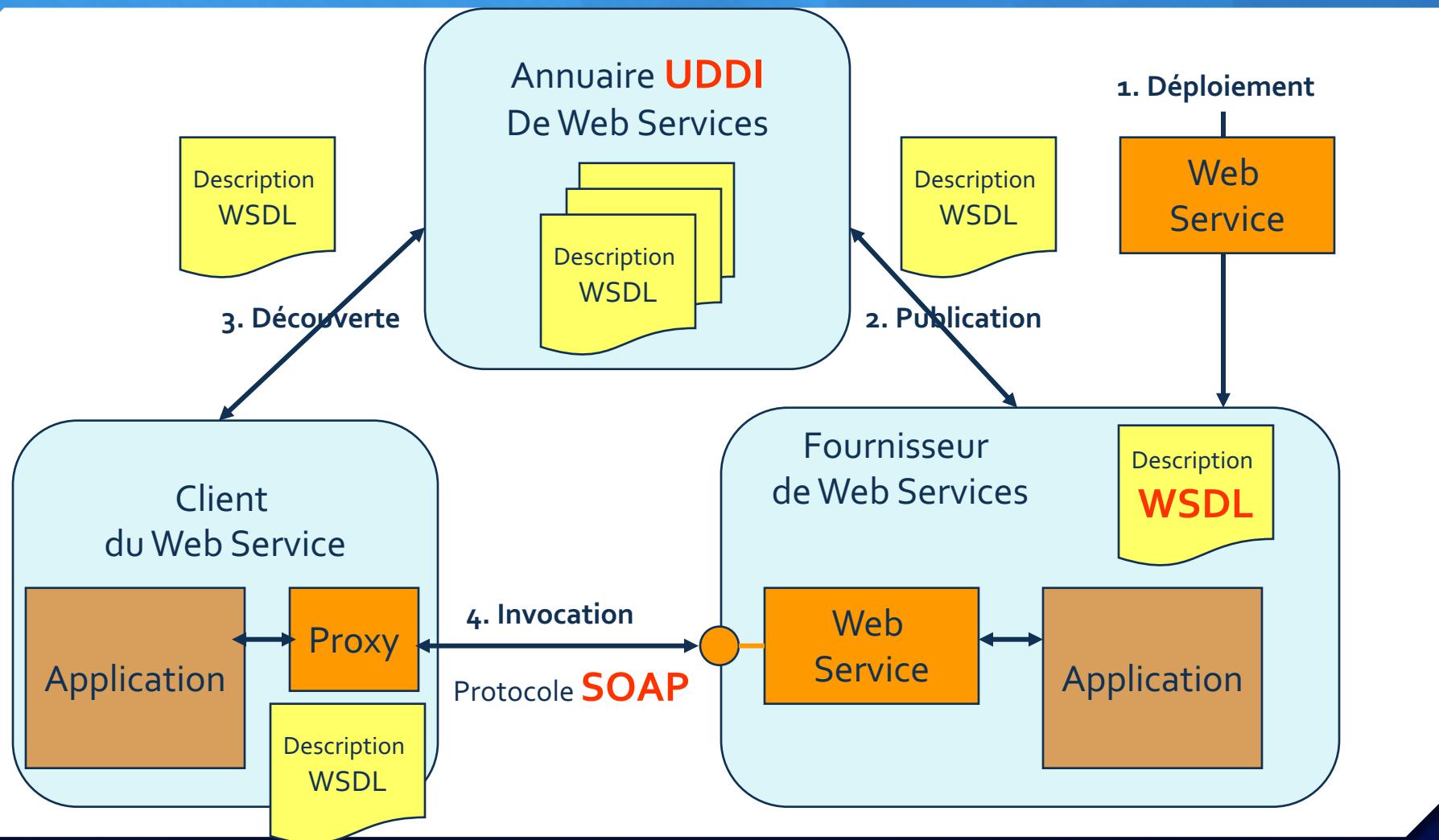
- + Service
 - + Composant autonome
 - + Faiblement couplé
 - + Synchrone ou asynchrone
 - + S'appuie généralement sur le web pour fonctionner
 - + Instance unique

Notions de Services

+ Web-Services



Notions de Services



Notions de Services

- + SOA = Service Oriented Architecture
Architecture Orientée Services
 - + Couplage faible
 - + Encapsulation des services
 - + Notion de contrat de service
 - + Réutilisable, Autonome,
 - + ...

Notions de Service

+ REST

REpresentational State Transfer

Basé sur HTTP

+ URI = identifiant

<http://monAdresse/Livres/87/Comments>

+ Utilisation des "verbes" HTTP

GET, POST, PUT, DELETE

+ Réponses HTTP = ressources

XML, HTML, CSV, JSON, ...

+ Lines = relations entre des ressources

+ Jeton d'identification en paramètre (GET)

PHP & Symfony



PHP

+ Fonctionnement

Navigateur WEB



Requête http

Serveur



Réponse http

SGBD



- Application



PHP

- + Syntaxe
 - + Script dans du code HTML
 - + Script à part
- + Variables \$maVariable
 - + Typage dynamique
 - + Typage explicite
- + Instructions de contrôle
 - + Tests (if, switch)
 - + Boucles (for, while, foreach)
- + Fonctions
- + Classes et interfaces

```
<?php  
    print "Hello world !!!";  
?>
```

PHP

- + Eléments de syntaxe
 - + Affectation \$variable = valeur;
 - + Opérateurs classiques : + - * /
 - + Comparateurs classiques : && || != == <= < >= > !=
 - + Comparateurs avec type === !==:
 - + Comparateur tie Fighter (compareTo) <=>
 - + Chaines de caractères "..." ou '...'
 - + Concaténation .
 - + Vecteurs. Indexation par des entiers ou des chaines de caractère. Dimension définie en fonction des circonstances.

PHP

+ Vecteurs

+ Déclaration explicite

```
$monTableau = array();  
$monTableau = array(1, 2, 3);  
$monTableau = [1, 2, 3];  
$montableau = array(1 => 1, "toto" => 2, 3 => 1);  
...
```

+ Utilisation implicite

```
$monTableau[1] = 1;
```

+ Utilisation

```
$x = monTableau[1];  
$x = $monTableau[2]["x"];
```

PHP

+ Structures de contrôle

```
if ($i == 0) {  
    $val = 1;  
} else if ($i==1) {  
    $val = 2;  
} else {  
    $val = 3;  
}
```

```
switch ($x) {  
    case "abc" :  
        $val = 1;  
        break;  
    case 1 :  
        $val = 2;  
        break;  
    default :  
        $val = 3;  
        break;  
}
```

PHP

+ Structures de contrôle

```
for ($i=0; $i < $n; $i++) {  
}
```

```
while ($i < $n) {  
}
```

```
do {  
} while ($i < $n);
```

```
foreach ($vecteur as $valeur) {  
}
```

```
foreach ($vecteur as $index => $valeur) {  
}
```

PHP

+ Fonctions

```
function maFonction(parametres) {  
    ...  
    return ...  
}
```

```
$x = maFonction(arguments);
```

```
function nomFonction(int $x, float $y, string $s, UnObjet $o) {  
    ...  
}
```

PHP

+ Classes

- + Attributs
private, protected, public
- + Méthodes
private, protected, public
- + Constructeurs
- + L'objet courant \$this->
- + Héritage
- + Polymorphisme
- + Interfaces

```
<?php
class maClasse {
    // Attributs
    private $attrib;

    // Méthodes
    public function maClasse ($value) {
        $this->attrib = $value;
    }
    public function faire_qqchose() {
        print("Possède : ".$this->attrib);
    }
}

$bar = new maClasse (1);
$bar->faire_qqchose ();

?>
```

PHP

+ Utilisation d'objets

```
<?php  
use unObjet;  
  
class myClass {  
    public function f(unObjet $o) {  
        ...  
    }  
    ...  
}  
?>
```

PHP

+ Interfaces

```
<?php
    interface myTemplate {
        public function setScreenName($name);
    }

    class myImplementation implements myTemplate {
        ...
        public function setScreenName($name) {
        }
    }
?>
```

PHP

+ Les échanges avec le navigateur



Requête http mode=GET
Requête http mode=POST

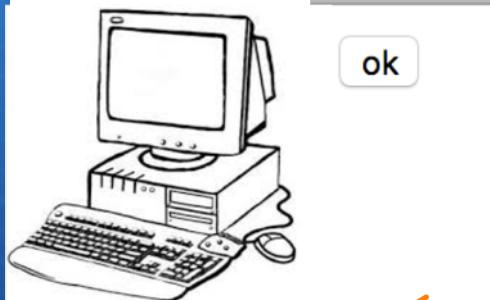


PHP `$_GET`
 `$_POST`

PHP

HTML

```
<form method="GET" name="test" action="test.php" >  
    <input type="submit" name="test" value="ok" />  
</form>
```



ok

http://... action.php?test=ok

Réponse http

```
<html>  
<head><title></title></head>  
<body>  
ok  
</body>  
</html>
```

```
$_GET = [  
    "test" => "ok"  
]
```

Action.php

```
<html>  
<head><title></title></head>  
<body>  
<?php  
    print $_GET["test"];  
?>  
</body>  
</html>
```

PHP

- + Accès aux bases de données
 - + Plusieurs modes
 - + Accès spécifiques dépendant du type de bases de données
Obsolète depuis PHP7
 - + Via des drivers génériques (PDO)
 - + Principe :
 - + Connexion à la base => ressource
 - + Requêtes
 - + Fermeture de la connexion

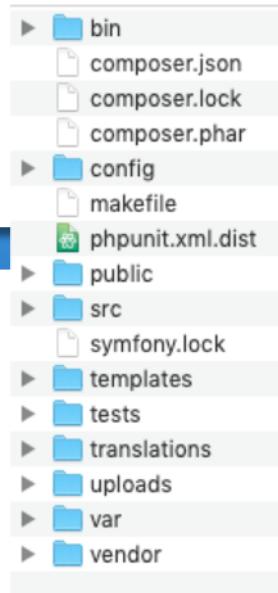
PHP

```
$dsn = "pgsql:dbname=testdb;host=127.0.0.1";
$user = "monloginBD";
$password = "monPwdBD";
try {
    $dbh = new PDO($dsn, $user, $password);
    $sql = 'SELECT nom, couleur, calories
            FROM fruit WHERE calories < :calories AND couleur = :couleur';
    $stmt = $dbh->prepare($sql, array(PDO::ATTR_CURSOR, PDO::CURSOR_SCROLL));
    $stmt->execute(array(':calories' => 150, ':couleur' => 'red'));
    $nb = $stmt>rowCount();
    while ($row = $stmt->fetch(PDO::FETCH_ASSOC, PDO::FETCH_ORI_NEXT)) {
        $data = $row["nom"] . "\t" . $row["couleur"] . "\t" . $row["calories"] . "\n";
        print $data;
    }
} catch (PDOException $e) {
    echo 'Connexion échouée : ' . $e->getMessage();
}
```

De PHP à Symfony

- + Symfony
 - + Framework PHP
 - + Un des plus utilisés (en France)
 - + Version 4 (évitez les versions 1 et 2)
 - + Modèle MVC
 - + Inclus un environnement de production et un environnement de développement
 - + En général couplé aux ORM Doctrine ou Propel
 - + Utilise TWIG (aspect Vue)
 - + Peut s'exécuter via apache ou seul (embarque son propre serveur web)
- + Mise en œuvre : utilisez **composer**

Symfony



- + Structuration forte du projet
 - + src = le code source
 - + templates = les vues
 - + test = les outils de test
 - + public = les éléments d'affichage (html, images, css, js, ...)
 - + config = les éléments de configuration
- + Les éléments du MVC
 - + Contrôleurs
 - + Entités, Repository
 - + Templates (twig)

Symfony

- + Gestion des objets (entités) via un ORM (en général Doctrine)
 - + Déclaration des entités dans des fichiers PHP
 - + Déclaration des méthodes de gestion dans les Repository
 - + Gestion de l'historique dans les Migration
 - + Lien avec la base de données via l'ORM

De JavaEE à Spring



JavaEE

+ Objectif : application web en JAVA

Navigateur WEB



Requête http

Serveur WEB



Réponse http

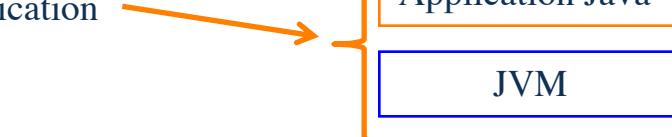
SGBD



- Application

Application Java

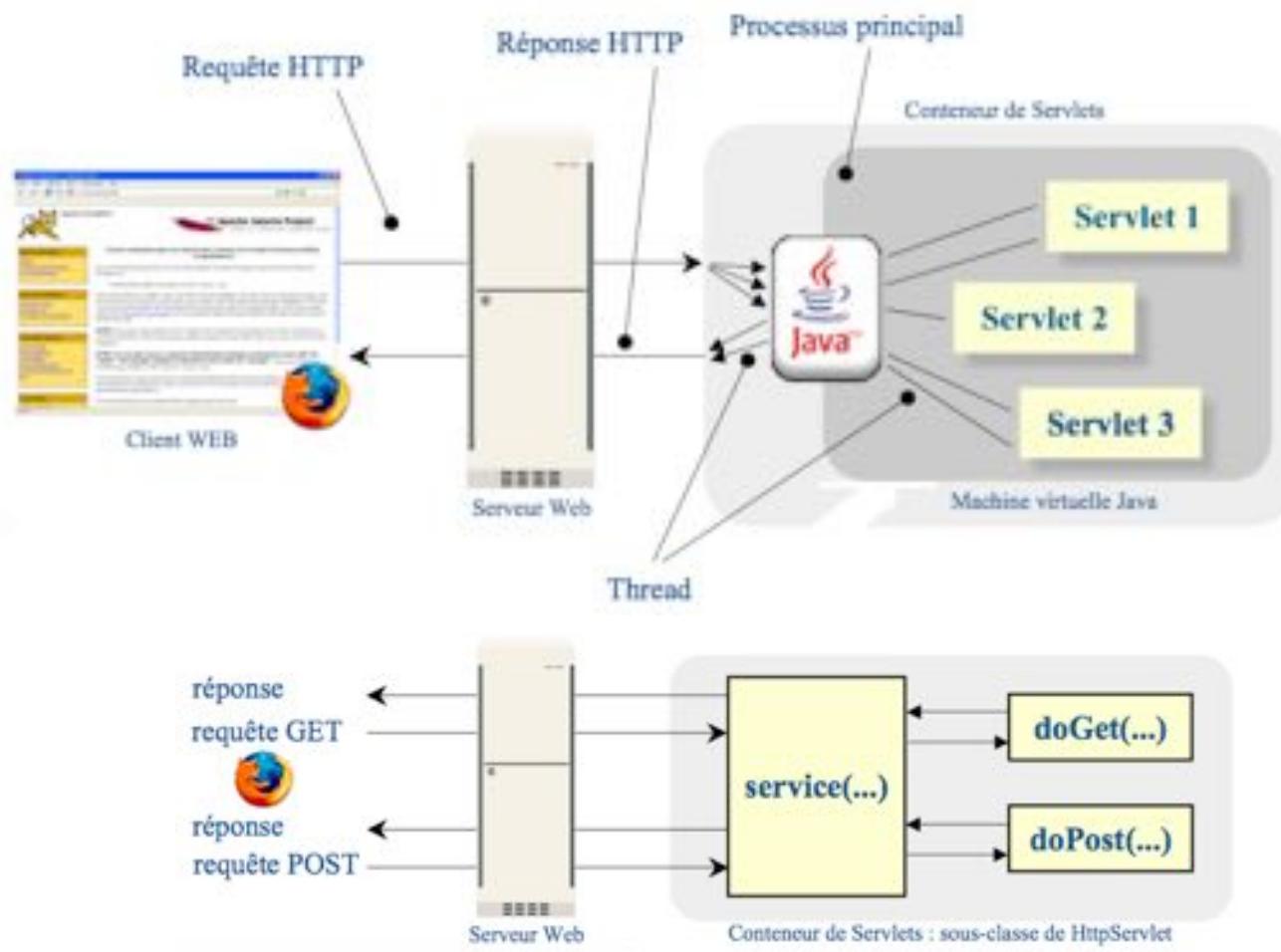
JVM



JavaEE

- + Java Enterprise Edition ou JavaEE
J2EE = ancien nom, mais encore utilisé
 - + Ensemble d'outils JAVA (API) permettant de répondre à des requêtes de la part d'un navigateur web
 - + Servlets
 - + JSP (Java Server Pages)
 - + ...
- et évidemment d'échanger entre les éléments.

JavaEE



JavaEE

```
import javax.servlet.*;
import javax.servlet.http.*;

import java.io.*;

public class BonjourMonde extends HttpServlet {
    public static final long serialVersionUID = 1;

    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>Test Servlet</title></head>");
        out.println("<body>");
        out.println("<p>Hello World!</p>");
        out.println("</body></html>");
    }
}
```

```
protected void doPost(HttpServletRequest request,
                      HttpServletResponse response) throws ServletException, IOException {
    ...
}
```

Objet de la requête

Réponse à la requête

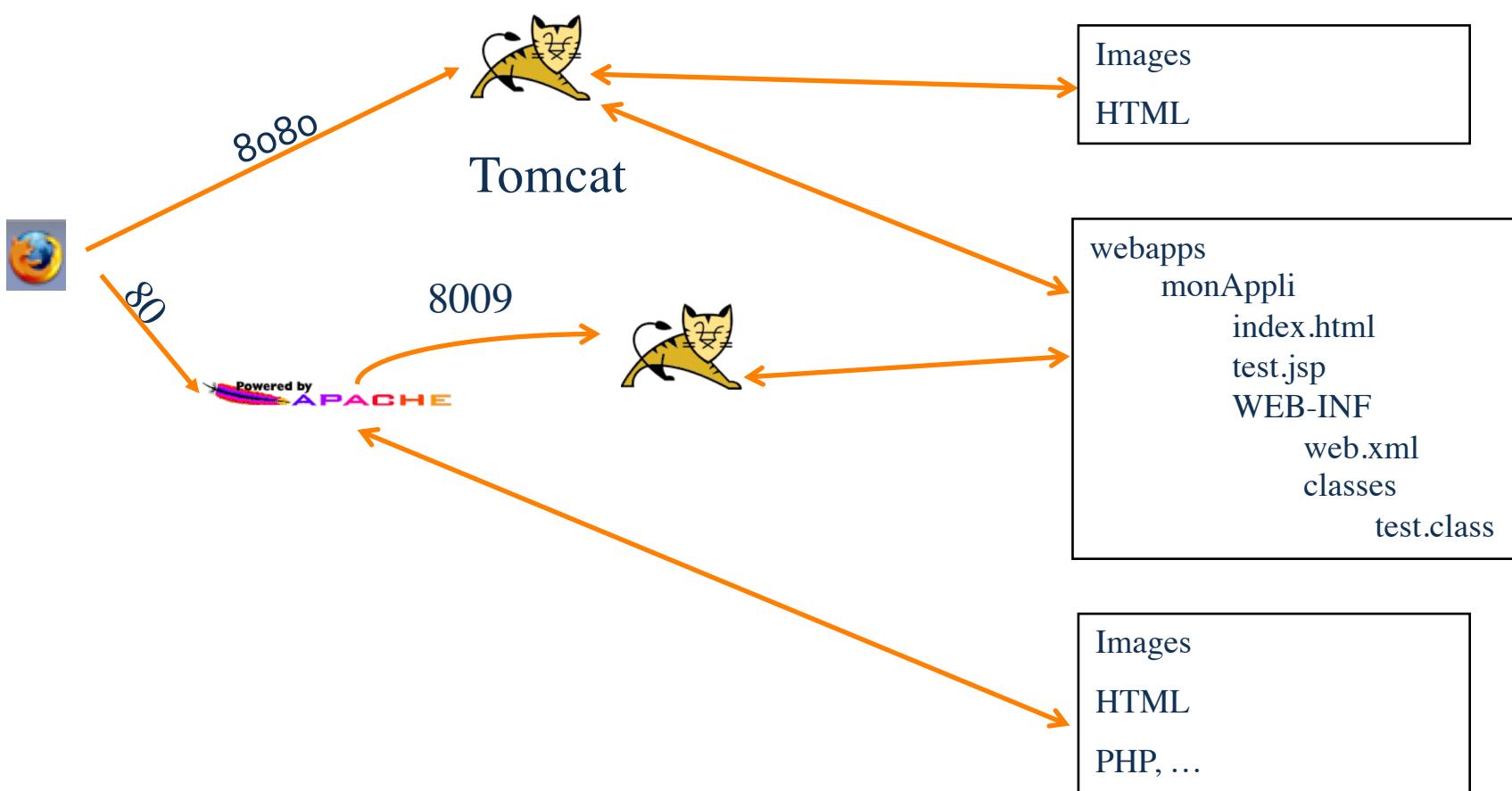
JavaEE

- + HttpServletRequest
 - + String getParameter(String name) : retourne la valeur d'un paramètre
 - + String[] getParameterNames() : retourne le nom de tous les paramètres
 - + String getRemoteHost() : retourne l'IP du client
 - + Object getAttribute(String name) : retourne la valeur de l'attribut
 - + void setAttribute(String name, Object value) : ajoute un attribut
 - + ...
- + HttpServletResponse
 - + void setStatus(int) : définit le code de retour de la réponse
 - + void.setContentType(String) : définit le type de contenu MIME
 - + ServletOutputStream getOutputStream() : flot pour envoyer des données binaires au client
 - + PrintWriter getWriter() : flot de sortie HTML
 - + void.sendRedirect(String) : redirige le navigateur

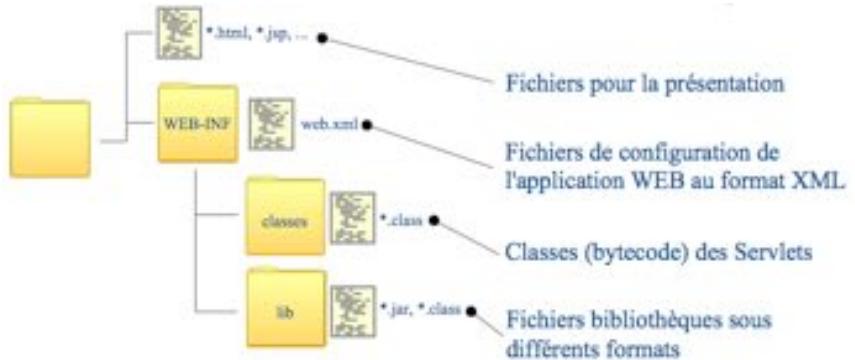
JavaEE

- + Les servlets sont persistantes
 - + Entre chaque requête du client les Servlets persistent sous forme d'instances d'objet
 - + Au moment où le code d'une Servlet est chargé, le serveur ne crée qu'une seule instance de classe
 - + L'instance (unique) traite chaque requête effectuée sur la Servlet
- + Avantages :
 - + L'empreinte mémoire reste petite
 - + Le surcoût en temps lié à la création d'un nouvel objet pour la Servlet est éliminé
 - + La persistance est possible c'est-à-dire la possibilité de conserver l'état de l'objet à chaque requête

JavaEE



JavaEE



```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <servlet>
        <servlet-name>HelloWorld</servlet-name>
        <servlet-class> HelloWorld</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloWorld</servlet-name>
        <url-pattern>/servlet/HelloWorld</url-pattern>
    </servlet-mapping>
</web-app>
```

JavaEE

+ Utiliser les annotations

```
@WebServlet(name = "NewServlet", urlPatterns = {"/NewServlet"})
public class NewServlet extends HttpServlet {
    ...
}
```

JavaEE

+ JSP - Java Server Pages

Inclure du code JAVA dans les pages HTML

```
<html>
<head>
<title>Essai de JSP</title>
</head>
<body>
<% if (request.getParameter("nom") == null) {
    out.println("Bonjour monde !");
}
else {
    out.println("Bonjour " + request.getParameter("nom") + " !");
}
%>
</body>
</html>
```

JavaEE

+ Directives JSP

Permettent de définir certains aspects structurels de la servlet

+ l'importation d'un package JAVA, y compris défini par l'utilisateur

```
<%@page import="monpackage.*" %>
<%@page import="java.lang.String" %>
<%@page import="monpackage1.maclasse,monPackage2.maclasse2" %>
```

+ Inclure un autre fichier (jsp par exemple)

```
<%@ include file="relativeURL" %>
<jsp:include page="..." />
```

```
<%!
    String Chaine = "bonjour";
    int Numero = 10;
    public void jspInit() {
        // instructions;
    }
%>
```

Struts - Spring - Springboot

- + Struts
 - + Ancien framework J2EE
- + Spring
 - + Framework JAVA, au dessus de JavaEE
 - + Modele MVC
 - + Peut embarquer d'autres modèles
 - + JSP pour les vues. Utilisez si possible JSTL
- + Springboot
 - + Construit à partir de Spring
 - + Embarque son propre serveur de servlets

SPRING

- + Framework MVC au dessus de J2EE
 - + Utilisation massive des annotations (@...)
 - + Implementation du modèle MVC
 - + @Controller : contrôleur
 - + @Entity, @Repository : les objets et leur gestion
 - + JSP et l'objet ModelAndView pour les vues
 - + Utilisation de l'ORM JPA pour la liaison avec la base de données
- + Comporte de nombreux utilitaires



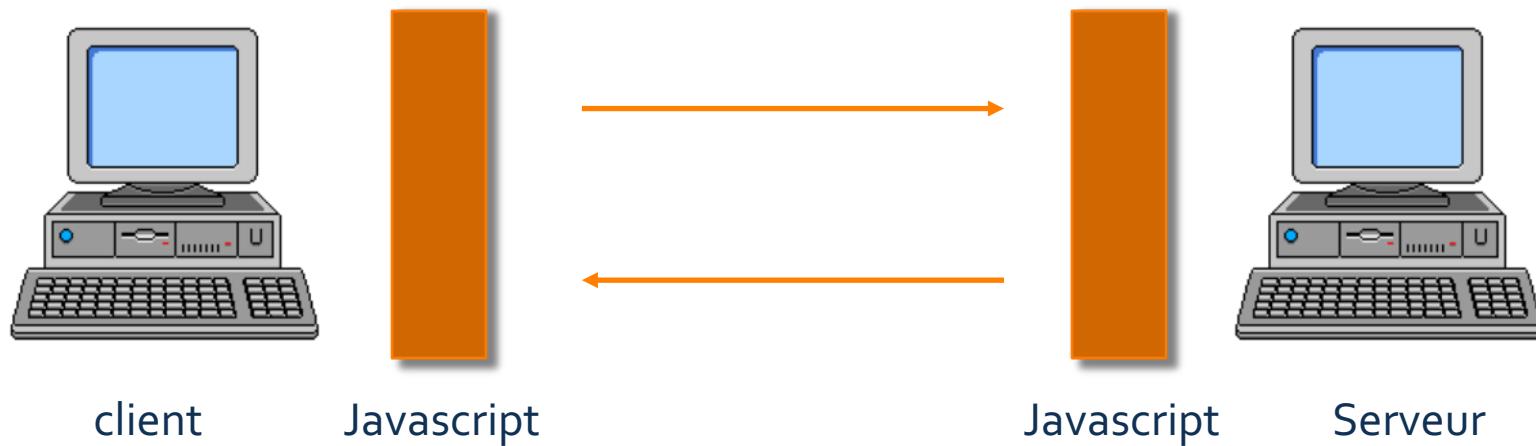
Angular, React, VueJS

Angular, React, VueJS

- + Framework Javascript / Typescript
- + Créeé par
 - + Angular : Google (Licence MIT)
 - + Angular JS : Javascript
 - + Angular 2, 3... 8 : Typescript
 - + React : Facebook (Licence MIT)
 - + VueJS : crééé à partir d'Angular (Licence MIT)

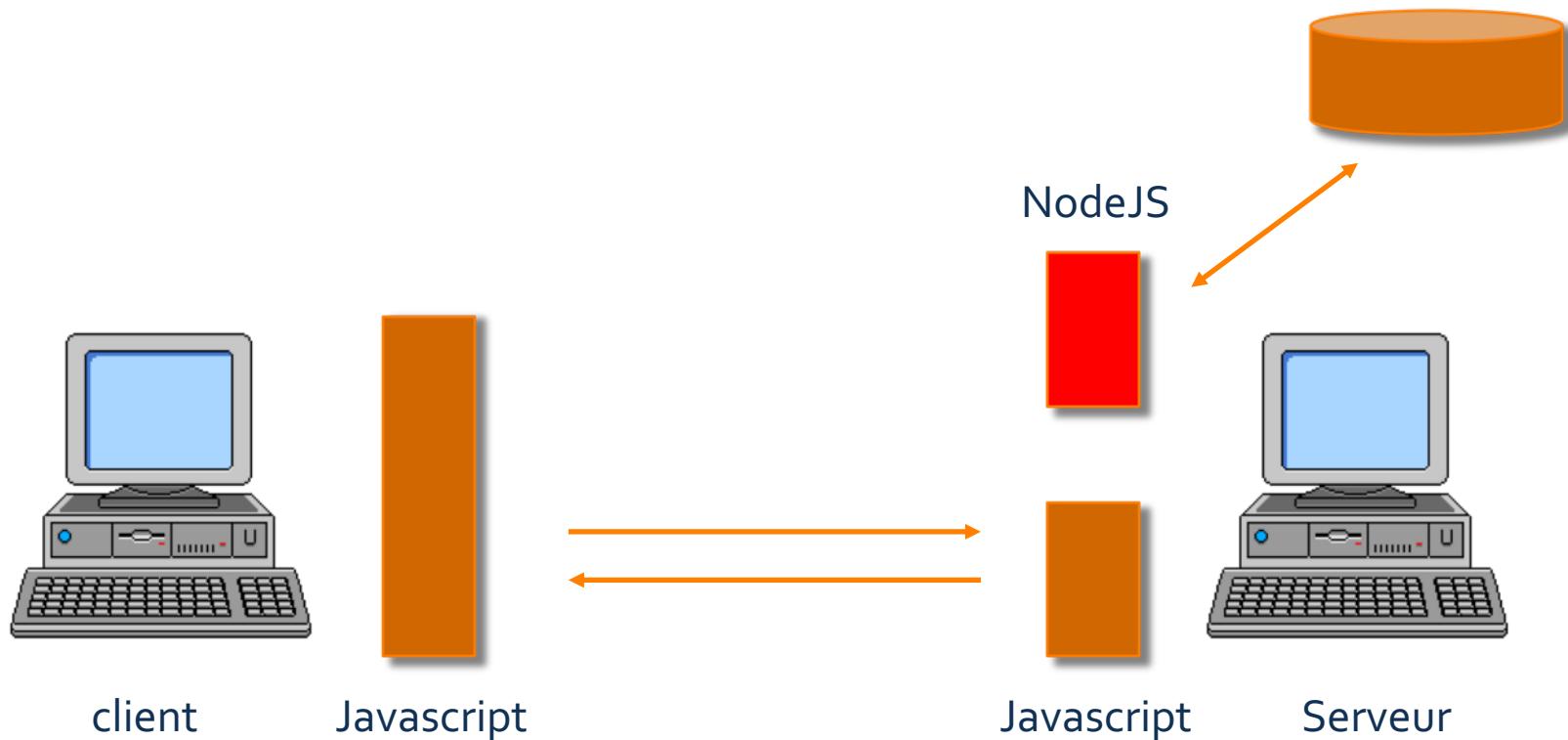
Angular, React, VueJS

+ Fonctionnement général



Angular, React, VueJS

- + On ajoute nodeJS, un SGBD, ...



Angular, React, VueJS

- + Le Framework
 - + Compile le code
 - + Pour le client
 - + Pour le serveur
 - + Assure le lien entre le client et le serveur (AJAX)
 - + Gère les informations sur le client
 - + Fonctionnement asynchrone
 - + Gestion des événements
 - + Gestion des modifications
 - + Gestion des propriétés (lien avec le DOM)
 - + Gestion des composants (interne)