NFA 017
MVC

Architecture Modèle/Vue/Contrôleur

Rappels sur les sessions

Utilité des sessions

- HTTP protocole sans état (requête sans mémoire)
- mécanisme des cookies
  - limité en taille (données transitent par le réseau)
  - peu sûr (données chez l'utilisateur)

Utilité des sessions

- sessions
  - données stockées sur le serveur
  - numéro de session en cookie

Technique

- Création : session_start() (au début de chaque page php)
- destruction : à la sortie du navigateur, ou au bout d'un certain temps (voir poly)

Utilisation

```
<?php
    session start ();
    if ( isset ($ SESSION[ ’compte ’ ]))
        $ SESSION[’compte’]++;
    else
        $ SESSION[’compte’]= 1;
?>
<html> <body>
<?php
    echo ”bonjour , c’est la visite numéro ”.
        $SESSION[’compte’];
?>
</body> </html>
```

Sessions et tableaux

- tableaux php manipulés par valeur :
  - $t= $_SESSION : $t est une *copie* de $_SESSION

- le mieux est de mettre des *objets* dans les sessions
- attention, les classes doivent être connues *avant* le session_start()
  - utiliser autoload ou require.

```php
<?php
class Panier {
  private $contenu=array();
  function ajouter($v) {
    $this->contenu[]= $v;
  }
}
session_start();

if (! isset($_SESSION['panier1'])) {
  $_SESSION['panier1']= new Panier();
 }

$u= $_SESSION['panier1'];
$u->ajouter('b');
?>
```

```php
<html>
<body>
<?php
        var_dump($t);
         echo "<p>";
         var_dump($u);
?>
</body>
</html>
```

problème GET/POST

- (sans rapport direct avec le cours mais utile à savoir)
- Requête de modification : typiquement en mode POST
- pb. rechargement de la page → ré-expédition de la requête

problème GET/POST

- Solution : POST = modification/GET = affichage
- Après la modification, le code php fait un affichage →nouvelle requête en mode GET...

```php
<?php
// Accès à la bd.
require("dataAccess.inc");
$produitCorrect=false;
if (isset($_POST['prix']) || isset($_POST['designation']) {
  // Exemple d'insertion d'un nouveau produit dans la base de données
  $prix= $_POST['prix'];
  $designation= $_POST['designation'];
  if ($designation != "" && is_numeric($prix) && doubleval($prix) > 0) {
    $produitCorrect= true;
  }
}

if ($produitCorrect) {
    $id= sauverProduit($prix, $designation);
    header("Location: " . "/afficherProduit.php?id=" . $id);
} else {
    require("formulaireProduit.php");
}
?>
```

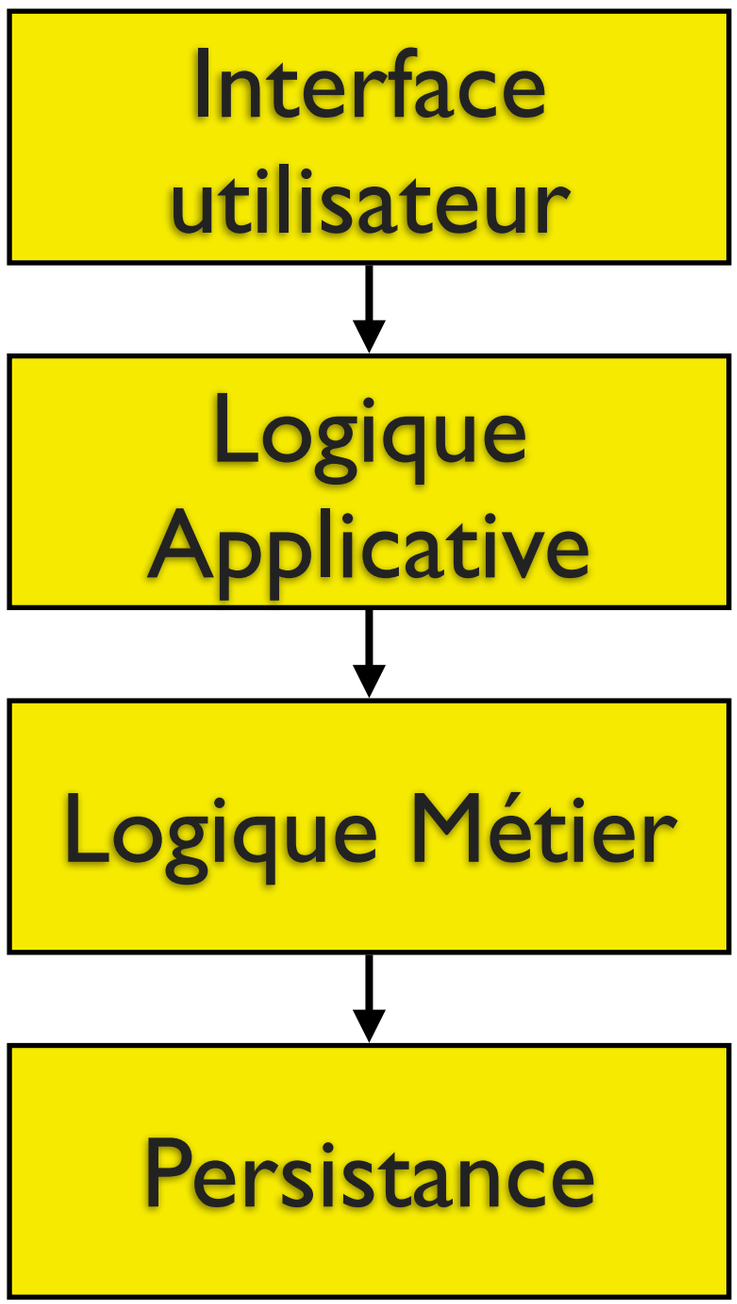
```
<html>
<body>
<form method='POST' action="creerProduit.php">
<p>designation <input type="text" name="designation"
   value="<?php echo htmlspecialchars($designation); ?>"/>
<p>prix <input type="text" name="prix"
   value="<?php echo htmlspecialchars($prix); ?>"/>
<p><input type="submit"/>
</form>
</body>
</html>
```

Architecture MVC

Buts

- Application complexe
- Maintenance, facilité de développement séparé
- découper le code en parties
  - le plus indépendantes possibles
  - accomplissant une tâche chacune
- Séparation du code en couches

Architecture typique



visualisation et contrôle

La flèche signifie « utilise »

Définitions

- **Modèle** données sur lesquelles travaille le programme
- **Interface utilisateur** partie du programme qui affiche les données du modèle, et récupère les informations fournies par l'utilisateur
- **Logique applicative** traitements liés à une application particulière.
- **Logique métier/Modèle** représentation du domaine sur lequel travaille le programme. Exemple : les commandes, les produits à commander... en OO, fait bcp de traitements
- **Persistance** copie/sauvegarde de/vers la BD.

(Sans) Séparation modèle/vue

```
<html> <head>
<title> sans séparation</title>
</head> <body>
<p> liste des articles : </p>
<?php
$db= new PDO(”mysql:host=localhost;dbname=publi”, ’xxxx’, ’xxxx’);
$st = $db–>query(”SELECT * FROM article”);
while ($obj= $st–>fetchObject ()) : ?>
    <h3><em><?php echo htmlspecialchars($obj–>id article ); ?></em>
<?php echo htmlspecialchars($obj–>titre ); ?></h3>
<p> <?php echo htmlspecialchars($obj–>texte ); ?>
</p> <?php   endwhile ; ?>
</body> </html>
```

Séparation

- Le code PHP va remplir un tableau ($VUE)
- La vue va *afficher* les données *déjà calculées* qui sont dans $VUE

```php
<?php
$db= new PDO
("mysql:host=localhost;dbname=publi1",
    'publi1', 'publi1');
$st = $db->query("SELECT * FROM article");
$VUE= array();

while ($obj= $st->fetchObject()) {
  $VUE['articles'][]= array(
   'id_article' => htmlspecialchars(
        $obj->id_article),
   'titre' =>  htmlspecialchars($obj->titre),
   'texte' => htmlspecialchars($obj->texte)
  );
}
?>
<html>
<head><title>exemple avec séparation</title>
</head>
<body>
<p> liste des articles : </p>
<?php foreach ($VUE['articles'] as $art): ?>

<h3><em><?php echo $art['id_article']; ?></
em>  <?php echo $art['titre']; ?></h3>
  <p> <?php echo $art['texte']; ?>
  </p>
<?php endforeach; ?>
</body>
</html>
```

Séparation physique des deux parties

```
<?php
$db= new PDO("mysql:host=localhost;dbname=publi1", 'publi1', 'publi1');
$st = $db->query("SELECT * FROM article");
$VUE= array();

while ($obj= $st->fetchObject()) {
  $VUE['articles'][]= array(
   'id_article' => htmlspecialchars($obj->id_article),
    'titre' =>  htmlspecialchars($obj->titre),
    'texte' => htmlspecialchars($obj->texte)
  );
}

require('pageListeArticle.php');
?>
```

Le contrôle

Séparation physique des deux parties

```
<html>
<head><title>exemple avec séparation</title></head>
<body>

<p> liste des articles : </p>
<?php foreach ($VUE['articles'] as $art): ?>

<h3><em><?php echo $art['id_article']; ?></em>  <?php echo $art['titre']; ?></h3>
  <p> <?php echo $art['texte']; ?>
  </p>
<?php endforeach; ?>
</body>
</html>
```

La vue

Avantage : test facilité

```
<?php
$VUE['articles'][]= array('id_article'=> 1, 'titre' => 'premier',
'texte' => "un texte");
$VUE['articles'][]= array('id_article'=> 2, 'titre' => 'premier',
'texte' => "un texte");
$VUE['articles'][]= array('id_article'=> 3, 'titre' => 'premier',
'texte' => "un texte");

require('pageListeArticle.php');
?>
```

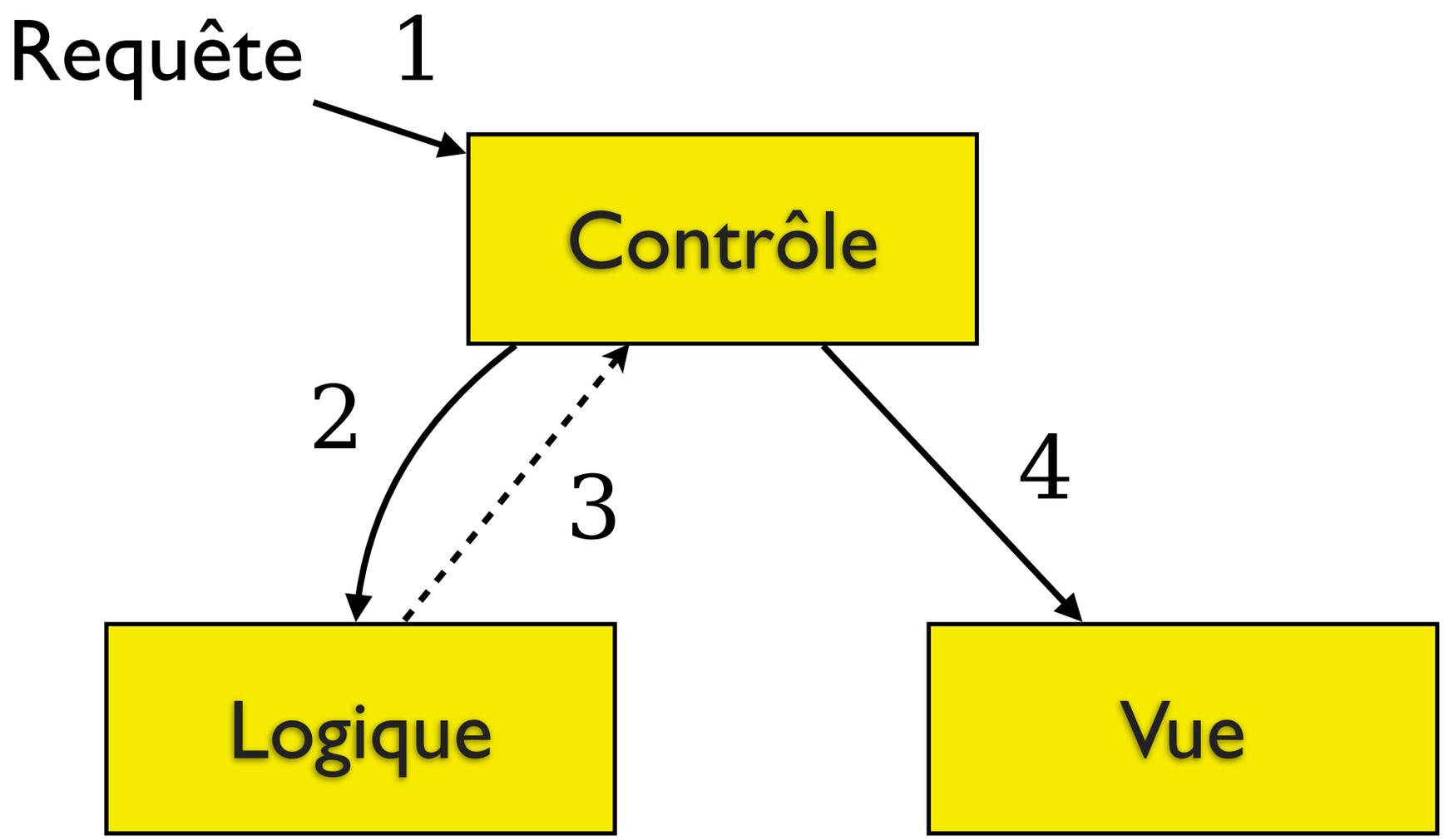
Choix d'affichage

```
<?php
try {
  $db= new PDO("mysql:host=localhost;dbname=publi1", 'publi1', 'publi1');
  $st = $db->query("SELECT * FROM article");
  $VUE= array();
  while ($obj= $st->fetchObject()) {
    $VUE['articles'][]= array(
                        'id_article' => htmlspecialchars($obj->id_article),
                        'titre' =>  htmlspecialchars($obj->titre),
                        'texte' => htmlspecialchars($obj->texte));
  }
  $ok= true;
} catch (PDOException $e) {
  $ok= false;
  $VUE['message']= htmlspecialchars("Désolé, erreur de connexion à la base");
}
if ($ok)
  require('pageListeArticle.php');  // affichage normal
else
  require('erreur.php'); // affichage erreur...
?>
```

Architecture MVC complète

- le contrôle (php) reçoit la requête, appelle la logique applicative, et transmet le résultat à la vue pour affichage
- la logique gère les données internes du programme
- la vue visualise les résultats

Architecture MVC



Exemple simple
Saisie et consultation de notes

- Deux types d'utilisateurs (à connecter) : étudiants et administration
- Les étudiants sont inscrits dans une ou plusieurs matières
- l'administrateur peut saisir les notes des étudiants dans une matière donnée
- les notes sont entre 0 et 20
- un étudiant peut consulter ses notes

Pages

- Connexion (obligatoire)
- visualisation (seule page possible pour un étudiant)
- choix d'une matière (admin)
- édition des notes (formulaire)
- visualisations notes (admin)

«façade» du modèle

- Modèle : prévoir les opérations à réaliser
  - vérifier la connexion
  - récupérer les notes d'un étudiant
  - récupérer la liste des matières
  - valider une série de notes dans une matière
  - récupérer toutes les notes dans une matière

```
class NotesFacade {
  function connecter($login, $passwd) {...}
  function getUtilisateur() {...}
  function getNotes($idEtud) {...}
  function getMatieres() {...}
  function valider($notes) {...}
  function getNotes($idMatiere) {...}
}
```

Contrôle

- fichier(s) php effectivement appelé(s) :
  - connecter.php
  - listerNotesEtudiant.php
  - listerMatieres.php
  - editerNotes.php
  - listerNotesMatiere.php

listerNotesEtudiant.php

- Récupère l'objet NotesFacade en session (s'il existe)
- si non, ou si l'utilisateur n'est pas le bon : connecter.php
- si l'utilisateur est un étudiant :
  - appeler la méthode qui récupère ses notes
  - les ajouter à la vue
  - afficher la vue

listerNotesEtudiant.php

- Récupère l'objet NotesFacade en session (s'il existe)
- si non, ou si l'utilisateur n'est pas le bon : connecter.php

Code réutilisable

- si l'utilisateur est un étudiant :
  - appeler la méthode qui récupère ses notes
  - les ajouter à la vue
  - afficher la vue

editerNotesMatieres.php

- vérifie les droits de l'utilisateur
- reçoit (POST) id de matière et une liste de notes déjà données (éventuellement vide)
- si les notes sont correctes (méthode du modèle ?) alors les sauver ; affichage suivant : la liste des notes (non éditable)
- si les notes sont vides ou incorrectes :

DAO

- Couche de Persistance
- Isole l'accès aux BD

- Généralement, une classe dans la DAO par classe du modèle Etudiant.php/ EtudiantDAO.php
- Appelée par le modèle (PAS par les vues)

«Front Controller»

- Utilisation d'un seul code php pour le contrôle de *toute* l'application
- Reçoit les requêtes, les dirige vers le «bon» code
- peut centraliser les opérations à faire dans tous les cas (conditions d'accès, etc...)

Exemple : un petit framework pour un forum