BUT INFORMATIQUE · 2E ANNÉE · R3.01 DÉVELOPPEMENT WEB

TP3
Les prix Nobel en MVC

Modèle, Vue, Contrôleur : les concepts clés des 8 exercices

Correction du TP · Support de projection et de révision



Ex. 1 → 8

Le même site qu'au TP2, rangé autrement

1

Comprendre l'architecture

Ex. 1

- Le routeur `index.php`
- Contrôleur et actions
- `render()` et `extract()`
- Le modèle fourni

`index.php` · `Controller.php`
`Controller_home.php`

2

Afficher des données

Ex. 2 → 3

- Recette d'une nouvelle page
- Erreurs : `action_error()`
- Écrire une vue : `<?=`, `e()`, `??`

`Controller_list.php`
`view_last` · `view_informations`

3

Modifier les données

Ex. 4 → 6

- Formulaire + action de traitement
- Ajout, suppression, modification
- La vue `message` réutilisée

`Controller_set.php`
`view_form_add` · `view_form_update`

4

Naviguer dans les données

Ex. 7 → 8

- Vues partielles
- Pagination
- Recherche en GET
- `http_build_query()`

`view_list_nobel` · `view_pages_links`
`Controller_search.php`

→ **Fil rouge.** Le site fait exactement la même chose qu'à la fin du TP2. Ce qui change, c'est **l'organisation du code**. Chaque dossier `exercice1` → `exercice8` contient l'application complète à la fin de l'exercice.

MVC : un rôle par fichier

Au TP2, chaque page (`last25.php`, `add.php`…) faisait tout : lire l'URL, vérifier, appeler le Model, écrire le HTML. En MVC, on sépare ces rôles.

M — Modèle
`Models/Model.php`
Accède à la base. C'est le **seul** fichier qui écrit du SQL.

C — Contrôleur
`Controllers/Controller_*.php`
Lit la requête, vérifie les paramètres, interroge le modèle, **choisit la vue**.

V — Vue
`Views/view_*.php`
Du HTML qui **affiche** les variables reçues. Ni SQL, ni vérification.

Page du TP2	En MVC : contrôleur · action
`home.php`	`home` · `home`
`last25.php`	`list` · `last`
`informations.php?id=19`	`list` · `informations`
`list_all.php?page=3`	`list` · `pagination`

Page du TP2	En MVC : contrôleur · action
`form_add.php` → `add.php`	`set` · `form_add` → `add`
`remove.php?id=19`	`set` · `remove`
`form_update.php` → `update.php`	`set` · `form_update` → `update`
`form_search.php` → `search.php`	`search` · `form` → `pagination`

À RETENIR Modèle = les données, Vue = l'affichage, Contrôleur = le chef d'orchestre qui relie les deux.

Le trajet d'une requête

index.php	?controller=list	&action=informations	&id=19
LE ROUTEUR	QUEL CONTRÔLEUR	QUELLE ACTION	UN PARAMÈTRE

1	**index.php**	vérifie que `list` est autorisé, puis `new Controller_list()`
2	**Controller.php**	le constructeur lit `action` et appelle `action_informations()`
3	**Controller_list.php**	l'action vérifie `id` et demande les données au modèle
4	**Models/Model.php**	`getNobelPrizeInformations(19)` exécute la requête SQL
5	**view_informations.php**	`render("informations", $infos)` affiche la vue : le HTML part au navigateur

ORGANISATION D'UN DOSSIER

```
exercice8/
├── index.php
├── Controllers/
│   ├── Controller.php
│   ├── Controller_home.php
│   ├── Controller_list.php
│   ├── Controller_set.php
│   └── Controller_search.php
├── Models/
│   └── Model.php
├── Views/
│   └── view_*.php
└── Utils/
    └── functions.php
```

À RETENIR Toutes les pages passent par `index.php` : c'est l'URL qui choisit le contrôleur et l'action.

Le routeur : `index.php`

EXERCICE8/INDEX.PHP

```php
require_once "Utils/functions.php";
require_once "Models/Model.php";
require_once "Controllers/Controller.php";

$controllers = ['home', 'list', 'set', 'search'];   // liste blanche
$controller_default = "home";

if (isset($_GET['controller']) and in_array($_GET['controller'], $controllers)) {
    $nom_controller = $_GET['controller'];
} else {
    $nom_controller = $controller_default;
}

$nom_classe = 'Controller_' . $nom_controller;          // Controller_list
$nom_fichier = 'Controllers/' . $nom_classe . '.php';

if (is_readable($nom_fichier)) {
    require_once $nom_fichier;
    new $nom_classe();      // le constructeur lance l'action
} else {
    die("Error 404: not found!");
}
```

- **Point d'entrée unique** : toutes les URL du site commencent par `index.php?…`
- **Liste blanche** : seuls les contrôleurs de `$controllers` sont acceptés. Toute autre valeur donne le contrôleur par défaut, `home`.
- **Classe dynamique** : `new $nom_classe()` crée un objet de la classe dont le nom est dans la variable.
- **Convention** : `list` → classe `Controller_list` → fichier `Controllers/Controller_list.php`.
- `is_readable()` : le fichier existe et peut être lu. Sinon, « Error 404 ».

À RETENIR Ne jamais construire un nom de fichier à partir de l'URL sans le comparer d'abord à une liste de valeurs autorisées.

Contrôleurs et actions : héritage d'une classe abstraite

CONTROLLERS/CONTROLLER.PHP (EXTRAIT)

```php
abstract class Controller
{
    public function __construct()
    {
        if (isset($_GET['action'])
                and method_exists($this, "action_" . $_GET['action'])) {
            $action = "action_" . $_GET['action'];
            $this->$action();          // ex. action_informations()
        } else {
            $this->action_default();
        }
    }
    abstract public function action_default();   // imposée aux filles
}
```

CONTROLLERS/CONTROLLER_HOME.PHP

```php
class Controller_home extends Controller
{
    public function action_default() { $this->action_home(); }
    public function action_home()
    {
        $data = ['nb_nobels' => Model::getModel()->getNbNobelPrizes()];
        $this->render("home", $data);
    }
}
```

- `abstract class` : on ne peut pas l'instancier. Elle sert de base commune à tous les contrôleurs (`extends Controller`).
- **Méthode abstraite** : chaque contrôleur **doit** définir `action_default()`.
- `method_exists()` : une action inconnue (`?action=xyz`) donne l'action par défaut.
- `$this->$action()` : le nom de la méthode est dans une variable.
- Préfixe `action_` : depuis l'URL, on ne peut appeler que les méthodes prévues pour ça.
- `protected` : `render()` et `action_error()` servent aux classes filles, pas à l'extérieur.

À RETENIR `?action=last` appelle la méthode `action_last()` du contrôleur ; sans action, c'est `action_default()`.

`render()` : du contrôleur à la vue

CONTROLLERS/CONTROLLER.PHP : RENDER()

```php
protected function render($vue, $data = [])
{
    extract($data);                        // chaque clé devient une variable
    $file_name = "Views/view_" . $vue . '.php';
    if (file_exists($file_name)) {
        include $file_name;                // la vue voit ces variables
    } else {
        $this->action_error("La vue n'existe pas !");
    }
    die();                                 // fin du script
}
```

VIEWS/VIEW_HOME.PHP

```php
<?php require "view_begin.php"; ?>

<h1> List of the nobel prizes </h1>
<p> It references
    <strong><?= e($nb_nobels) ?></strong>
    nobel prizes. </p>

<?php require "view_end.php"; ?>
```

`$data = ['nb_nobels' => 839]` → `render("home", $data)` → `extract() : $nb_nobels = 839` → `Views/view_home.php`

- `render("home")` cherche le fichier `Views/view_home.php`.
- `die()` : rien ne s'exécute après `render()`.
- La vue est incluse **dans** `render()` : elle ne voit que `$data`, pas les variables de l'action.
- `action_error($msg)` appelle `render("message", …)` : elle arrête donc aussi le script.

À RETENIR Le contrôleur prépare un tableau `$data` ; dans la vue, `$data['nb_nobels']` devient `$nb_nobels`.

Le modèle fourni et `credentials.php`

TP3/CREDENTIALS.PHP (HORS DES DOSSIERS PUBLIÉS)

```php
$dsn   = 'pgsql:host=aquabdd;dbname=etudiants';
$login = 'numero_etudiant';
$mdp   = 'code_INE';
```

MODELS/MODEL.PHP : LE CONSTRUCTEUR

```php
private function __construct()
{
    // définit $dsn, $login et $mdp
    include __DIR__ . "/../../credentials.php";
    $this->bd = new PDO($dsn, $login, $mdp);
    $this->bd->setAttribute(PDO::ATTR_ERRMODE,
                            PDO::ERRMODE_EXCEPTION);
}
```

- Un mot de passe ne doit **jamais** être dans un fichier accessible par une URL.
- Même **singleton** qu'au TP2, en camelCase : `Model::getModel()`.

Méthode du modèle	Appelée par (ex.)
`getNbNobelPrizes()`	home (1), list/pagination (7)
`getLast()`	list/last (2)
`getNobelPrizeInformations($id)`	list/informations (3), set/form_update (6)
`getCategories()`	set/form_add (4), search/form (8)
`addNobelPrize($infos)`	set/add (4)
`isInDataBase($id)`, `removeNobelPrize($id)`	set/remove (5)
`updateNobelPrize($infos)`	set/update (6)
`getNobelPrizesWithLimit(…)`	list/pagination (7)
`findNobelPrizes(…)`, `nbFindNobelPrizes(…)`	search/pagination (8)

Deux corrections dans le modèle de l'archive : `ORDER BY year DESC, id` (ordre stable pour la pagination) et `==` remplacé par `=` (qui n'existe pas en SQL).

À RETENIR Le modèle est fourni : on ne le réécrit pas, on l'appelle depuis les actions des contrôleurs.

Ajouter une page : la recette en 4 étapes

1 Déclarer le nouveau contrôleur dans `index.php`

```php
$controllers = ['home', 'list'];
```

2 Écrire l'action : elle prépare `$data`

CONTROLLERS/CONTROLLER_LIST.PHP

```php
class Controller_list extends Controller
{
    public function action_default()
    {
        $this->action_last();
    }

    public function action_last()
    {
        $data = ['nobels' => Model::getModel()->getLast()];
        $this->render("last", $data);
    }
}
```

3 Écrire la vue : elle affiche `$nobels`

VIEWS/VIEW_LAST.PHP

```php
<?php
$title = "List of the last 25 Nobel prizes";
require "view_begin.php";
?>
<table>
    <?php foreach ($nobels as $nobel) { ?>
        <tr>
            <td><?= e($nobel['name']) ?></td>
            <td><?= e($nobel['year']) ?></td>
        </tr>
    <?php } ?>
</table>
<?php require "view_end.php"; ?>
```

4 Ajouter le lien dans le menu (`view_begin.php`)

```php
<a href="?controller=list&amp;action=last"> Last Nobel Prizes</a>
```

À RETENIR Une page = une action qui prépare les données + une vue qui les affiche. Le `last25.php` du TP2 est coupé en deux.

La page d'informations : s'arrêter tôt en cas d'erreur

CONTROLLERS/CONTROLLER_LIST.PHP

```php
public function action_informations()
{
    if (!isset($_GET['id']) || !is_strictly_positive_integer($_GET['id'])) {
        $this->action_error("There is no valid id in the url.");   // fin ici
    }

    $infos = Model::getModel()->getNobelPrizeInformations((int) $_GET['id']);
    if ($infos === false) {
        $this->action_error("There is no nobel prize with such id.");
    }

    $this->render("informations", $infos);   // $id, $year, $name…
}
```

VIEWS/VIEW_INFORMATIONS.PHP (EXTRAIT)

```php
<li><strong>Name:</strong> <?= e($name ?? '???') ?></li>
<li><strong>Birth date:</strong> <?= e($birthdate ?? '???') ?></li>
```

- **Pas de `else`** : `action_error()` → `render()` → `die()`. Le reste de l'action ne s'exécute pas.
- Une seule vue `message` pour toutes les erreurs : `action_error()` la remplit.
- `$infos` est passé tel quel : ses colonnes deviennent les variables de la vue.
- `??` : si la valeur est `null` (colonne vide en base), on affiche `???`.
- `is_strictly_positive_integer()` est reprise du TP2, dans `Utils/functions.php`.

À RETENIR Vérifier, et sortir avec `action_error()` dès qu'un test échoue : le cas normal reste en bas, sans `else` imbriqués.

Écrire une vue : les règles

Écriture	Signification
`<?= $x ?>`	raccourci de `<?php echo $x; ?>`, toujours disponible
`e($x)`	`htmlspecialchars($x, ENT_QUOTES)` : à mettre autour de **tout** ce qui vient de la base ou de l'utilisateur
`$x ?? '???'`	valeur par défaut si `$x` est `null` ou n'existe pas
`&`	dans le HTML, `&` s'écrit `&`, même dans un `href` : `?controller=list&action=last`
`$title`	défini **avant** `require "view_begin.php"` : c'est le titre de l'onglet
`foreach … { ?>`	on alterne PHP et HTML, comme au TP2

Jamais dans une vue

- de requête SQL ni d'appel au modèle
- de `$_GET` ou `$_POST`
- de vérification des données

Pourquoi ?

Le contrôleur a déjà tout vérifié et préparé. Une vue peut alors être relue, modifiée ou réutilisée sans risque de casser la logique.

TOUT EN UNE LIGNE : VIEW_LIST_NOBEL.PHP

```
<td><a href="?controller=list&amp;action=informations&amp;id=<?= e($nobel['id']) ?>"><?= e($nobel['name']) ?></a></td>
```

À RETENIR Une vue ne fait qu'afficher les variables reçues, toujours échappées avec `e()`.

Ajouter un prix : deux actions dans `Controller_set`

set · form_add → view_form_add → POST → **set · add** → check_data() → addNobelPrize() → view_message

ACTION 1 : AFFICHER LE FORMULAIRE

```php
public function action_form_add()
{
    $data = [
        'categories' => Model::getModel()->getCategories(),
    ];
    $this->render("form_add", $data);
}
```

VIEWS/VIEW_FORM_ADD.PHP (EXTRAIT)

```php
<form action="?controller=set&amp;action=add" method="post">
  <input type="text" name="name"/>
  <input type="text" name="year"/>
  <?php foreach ($categories as $category) { ?>
    <input type="radio" name="category"
           value="<?= e($category) ?>"/>
  <?php } ?>
```

ACTION 2 : TRAITER LE FORMULAIRE (SIMPLIFIÉE)

```php
public function action_add()
{
    $infos = check_data();    // false si name, category ou year invalide
    if ($infos === false) {
        $message = "Error: name, category and year are required.";
    } else {
        try {
            $added = Model::getModel()->addNobelPrize($infos);
        } catch (PDOException $e) {   // catégorie inconnue
            $added = false;
        }
        $message = $added ? "The nobel prize has been added." : "Error: …";
    }
    $data = ['title' => "Add a Nobel Prize", 'message' => $message];
    $this->render("message", $data);
}
```

Les noms des champs (`name`, `year`…) sont les clés qu'attend `addNobelPrize()`. Le corrigé teste aussi `empty($_POST)`.

`get_text($tab, $key)` : texte sans espaces autour, ou `null`. Elle prend `$_POST` ou `$_GET` : on la réutilise à l'ex. 8.

La vue `message` sert pour tous les comptes rendus : ajout, suppression, modification, erreurs.

À RETENIR Un formulaire = deux actions du même contrôleur : l'une l'affiche, l'autre traite les données envoyées.

Supprimer et modifier un prix

Supprimer · ex. 5

ICÔNE DU TABLEAU (HTML PRODUIT)

```
<a href="?controller=set&amp;action=remove&amp;id=19">
```

ACTION_REMOVE() EN 4 TEMPS

1. id présent et valide ?
2. le prix existe ? `isInDataBase($id)`
3. suppression : `removeNobelPrize($id)`
4. compte rendu : `render("message", …)`

- Même schéma que l'ajout : chaque cas produit un `$message`.
- Pour aller plus loin : sur un vrai site, on supprime via un formulaire POST avec confirmation, pas avec un simple lien GET.

Modifier · ex. 6

ACTION_FORM_UPDATE() (EXTRAIT)

```php
$infos = $model->getNobelPrizeInformations((int) $_GET['id']);
if ($infos === false) {
    $this->action_error("There is no nobel prize with such id.");
}
$infos['categories'] = $model->getCategories();   // on ajoute
$this->render("form_update", $infos);  // $id, $name… et $categories
```

VIEWS/VIEW_FORM_UPDATE.PHP (EXTRAIT)

```php
<input type="hidden" name="id" value="<?= e($id) ?>"/>
<?php foreach ($categories as $cat) { ?>
  <input type="radio" name="category" value="<?= e($cat) ?>"
         <?= $cat === $category ? 'checked' : '' ?>/>
<?php } ?>
```

- **Piège d'`extract()`** : `$category` est déjà la catégorie du prix. La boucle utilise donc `$cat`, sinon elle l'écraserait.
- `action_update` réutilise `check_data()` et ajoute `$_POST['id']`.

À RETENIR Pour pré-remplir un formulaire, l'action réunit dans un seul tableau les infos du prix et la liste des catégories.

Pagination et vues partielles

CONTROLLERS/CONTROLLER_LIST.PHP

```php
public function action_pagination()
{
    $nb_per_page = 25;
    $start = 1;                  // ?start=3 → page 3
    if (isset($_GET['start']) && is_strictly_positive_integer($_GET['start'])) {
        $start = (int) $_GET['start'];
    }
    $model = Model::getModel();
    $nb_pages = (int) ceil($model->getNbNobelPrizes() / $nb_per_page);
    if ($start > $nb_pages) {
        $this->action_error("This page does not exist.");
    }
    $offset = ($start - 1) * $nb_per_page;
    $data = [
        'nobels'   => $model->getNobelPrizesWithLimit($offset, $nb_per_page),
        'start'    => $start,
        'nb_pages' => $nb_pages,
        'link'     => '?controller=list&action=pagination',   // sans start
    ];
    $this->render("pagination", $data);
}
```

VIEWS/VIEW_PAGINATION.PHP

```php
<?php require "view_begin.php"; ?>
<h1> List of all the Nobel prizes </h1>
<?php
require "view_list_nobel.php";   // le tableau
require "view_pages_links.php";  // 1 2 3 …
require "view_end.php";
?>
```

- **Vue partielle** : un morceau de vue inclus par plusieurs vues. Elle voit les mêmes variables.
- `view_list_nobel.php` : le tableau, pour `last`, `pagination` et `search`.
- `view_pages_links.php` attend `$start`, `$nb_pages` et `$link` : elle resservira pour la recherche.
- Même calcul qu'au TP2 : 839 prix → 34 pages.

À RETENIR Les vues partielles évitent de recopier le tableau et les liens de pages dans chaque vue.

Recherche : un formulaire envoyé en GET

VIEWS/VIEW_FORM_SEARCH.PHP (EXTRAIT)

```php
<form action="index.php" method="get">
    <input type="hidden" name="controller" value="search"/>
    <input type="hidden" name="action" value="pagination"/>

    <input type="text" name="name"/>
    <select name="sign">
        <option value=">=">&gt;= (from this year on)</option>
        <option value="<=">&lt;= (up to this year)</option>
        <option value="==">== (this year only)</option>
    </select>
    <input type="text" name="year"/>

    <?php foreach ($categories as $category) { ?>
        <input type="checkbox" name="categories[]"
               value="<?= e($category) ?>"/>
    <?php } ?>
</form>
```

Pourquoi deux champs cachés ?

En GET, le navigateur **remplace** tout ce qui suit le `?` de l'attribut `action` par les champs du formulaire. Écrire `action="?controller=search"` ne suffirait donc pas : on transmet `controller` et `action` comme des champs.

URL OBTENUE (DÉCODÉE)

```
index.php?controller=search&action=pagination
  &name=Curie&sign=>=&year=1900
  &categories[]=chemistry&categories[]=physics
```

- `name="categories[]"` : `$_GET['categories']` est un **tableau**.
- Les critères sont dans l'URL : on peut copier le lien, le mettre en favori, revenir en arrière.

À RETENIR Une recherche ne modifie rien : on l'envoie en GET, et ses critères apparaissent dans l'URL.

Valider les critères et les garder dans les liens

CONTROLLER_SEARCH::ACTION_PAGINATION() (EXTRAIT)

```php
$filters = [];
if (get_text($_GET, 'name') !== null) {
    $filters['name'] = get_text($_GET, 'name');
}
if (isset($_GET['year'], $_GET['sign'])
        && is_strictly_positive_integer($_GET['year'])
        && in_array($_GET['sign'], ['<=', '>=', '=='], true)) {   // liste blanche
    $filters['year'] = (int) $_GET['year'];
    $filters['sign'] = $_GET['sign'];
}
if (isset($_GET['categories']) && is_array($_GET['categories'])) {
    $chosen = array_filter($_GET['categories'], 'is_string');
    $categories = array_values(array_intersect($model->getCategories(), $chosen));
    if (count($categories) > 0) {
        $filters['categories'] = $categories;
    }
}
$parameters = ['controller' => 'search', 'action' => 'pagination'] + $filters;
$data = [
    'nobels'     => $model->findNobelPrizes($filters, $offset, 25),
    'nb_results' => $model->nbFindNobelPrizes($filters),
    'link'       => '?' . http_build_query($parameters),   // …
];
```

	TP2	TP3
Critères	`$_SESSION` (serveur)	dans l'URL
Formulaire	POST	GET
Lien « page 2 »	`?page=2`	critères + `&start=2`
Partageable	non	oui

- `http_build_query()` assemble et encode les paramètres : `['name' => 'Curie']` → `name=Curie`.
- `+` réunit deux tableaux : `controller`, `action`, puis les filtres.
- `array_intersect()` ne garde que les catégories qui existent ; `array_values()` les renumérote 0, 1… pour les marqueurs `:c0`, `:c1`.
- Le modèle utilise `LIKE`, sensible à la casse : `Curie` ≠ `curie`.

À RETENIR On ne transmet au modèle que des critères vérifiés, et on les recopie dans les liens de pages avec `http_build_query()`.

Pour réviser

Huit pièges fréquents en MVC

1 ✗ Nouveau contrôleur absent de `$controllers` : on retombe sur l'accueil
✓ l'ajouter à la liste d'`index.php`

2 ✗ `controller_list.php` : « Error 404: not found! »
✓ `Controllers/Controller_list.php` (sous Linux, les majuscules comptent)

3 ✗ `render("last")` sans `Views/view_last.php` : « La vue n'existe pas ! »
✓ respecter la convention `view_` + nom

4 ✗ Méthode nommée `last()` : `?action=last` appelle `action_default()`
✓ préfixe obligatoire : `action_last()`

5 ✗ Du code après `render()` ou `action_error()` : il ne s'exécute jamais
✓ tout préparer **avant** : `render()` se termine par `die()`

6 ✗ Une boucle qui écrase une variable de la vue (`$category`)
✓ choisir un autre nom : `$cat`

7 ✗ Formulaire GET sans champs cachés : il arrive sur l'accueil
✓ `<input type="hidden" name="controller" …>`

8 ✗ SQL, `$_GET` ou vérifications dans une vue
✓ tout préparer dans l'action ; la vue affiche avec `e()`

À RETENIR `controller=xxx` → `Controller_xxx` · `action=yyy` → `action_yyy()` · `render("zzz")` → `view_zzz.php`

Pour réviser

Mémo de révision

CONVENTIONS

`?controller=list`	→ `Controller_list.php`
`&action=last`	→ `action_last()`
`(pas d'action)`	→ `action_default()`
`render("last", $data)`	→ `view_last.php`
`$data['nobels']`	→ `$nobels (vue)`
`action_error($msg)`	→ `view_message.php`

PLAN DU SITE

Contrôleur	Actions	Vues
`home`	home	home
`list`	last informations pagination	last informations pagination
`set`	form_add, add remove form_update, update	form_add form_update message
`search`	form pagination	form_search search

PHP UTILISÉ

`abstract / extends`	classe de base
`protected`	réservé aux filles
`new $classe()`	classe dynamique
`$this->$m()`	méthode dynamique
`method_exists()`	la méthode existe ?
`in_array()`	liste blanche
`extract()`	clés → variables
`<?= … ?>`	afficher
`$x ?? 'y'`	valeur par défaut
`http_build_query()`	construire une URL

▶ S'entraîner : arrêter le TP2 (mêmes ports), puis dans `TP3` : `docker compose up -d` et `localhost:8080/exercice8/`. Pour partir de l'archive : décompresser `MVC.zip` dans `TP3` → `localhost:8080/MVC/`.