Développement WEB PHP - Séance 5

BUT Informatique parcours DACS

Table of contents:

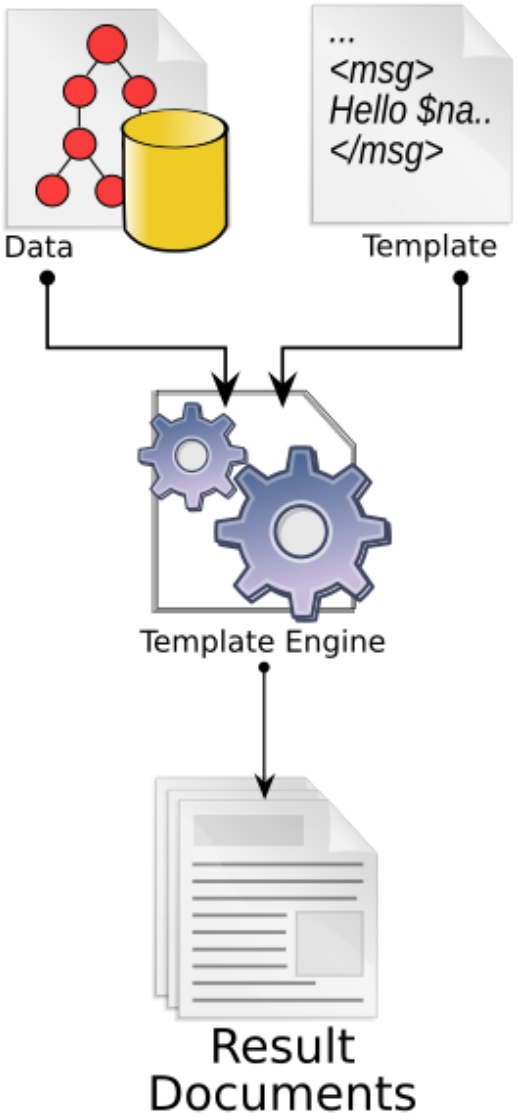
📜 | Qu'est-ce qu'un moteur de templating

> **INFO**
>
> Cette partie n'est pas à apprendre en détails. Cependant, **la compréhension des différents termes évoqués est nécessaire**.

Définition

Un **moteur de template** est un logiciel conçu pour combiner des **modèles** (template) avec un **modèle de données** afin de produire des documents finaux. Ces documents finaux sont souvent des pages Web, mais ils peuvent aussi être d'autres types de documents, tels que des fichiers PDF, des courriels ou des documents XML.

Dans le contexte du développement web, un moteur de template est généralement utilisé pour générer dynamiquement des pages HTML. **Le processus de génération de pages se déroule majoritairement côté serveur**, bien qu'il puisse aussi se produire côté client avec des frameworks JavaScript modernes (Svelte, React, Vue JS, ...).



Les moteurs de modèles incluent généralement des fonctionnalités communes à la plupart des langages de programmation de haut niveau, avec un accent sur les fonctionnalités de

traitement du texte brut.

Ces fonctionnalités incluent :

- variables
- évaluation conditionnelle et boucles
- remplacement de texte
- inclusion (ou transclusion) de fichier
- fonctions

Exemples de moteurs PHP existants

Twig

Twig est un moteur de templates pour le langage de programmation PHP, utilisé par défaut par le framework Symfony. Il a été inspiré par Jinja, moteur de template Python.

https://twig.symfony.com/

variable

```
{{ var }}
```

if

```
{% if users|length > 0 %}
    <ul>
        {% for user in users %}
            <li>{{ user.username|e }}</li>
        {% endfor %}
    </ul>
{% endif %}
```

for

```
{% for user in users %}
* {{ user.name }}
{% else %}
No users have been found.
{% endfor %}
```

require

```
{% include 'header.html' %}
```

Blade

Laravel Blade est le moteur de modèles par défaut du framework Laravel. Il vous permet d'utiliser des variables, des boucles, des instructions conditionnelles et d'autres fonctionnalités PHP directement dans votre code HTML.

https://laravel.com/docs/11.x/blade

variable

```
Hello, {{ $name }}.
```

if

```
@if (count($records) === 1)
    I have one record!
@elseif (count($records) > 1)
    I have multiple records!
@else
    I don't have any records!
@endif
```

for

```
@for ($i = 0; $i < 10; $i++)
    The current value is {{ $i }}
@endfor

@foreach ($user->posts as $post)
    <p>This is user {{ $user->id }}</p>
@endforeach
```

require

```
@include('footer.html')
```

⚙️ | Regex

Les **expressions régulières** (Regex) sont des modèles de recherche avancés qui permettent de détecter, valider, extraire ou manipuler des chaînes de caractères. En PHP, elles sont couramment utilisées pour valider des formats de données, comme des adresses email, des numéros de téléphone, des codes postaux, etc.

Plein d'outils existent pour tester et formatter ses expressions regex avant de les utiliser dans son code, par exemple https://regex101.com/.

Syntaxe des Expressions Régulières

Une expression régulière est constituée de **modèles** qui décrivent un motif de caractères à rechercher. Quelques éléments de syntaxe courants :

- `^` : Début de la chaîne
- `$` : Fin de la chaîne
- `.` : N'importe quel caractère
- `*` : 0 ou plusieurs répétitions du caractère précédent
- `+` : 1 ou plusieurs répétitions du caractère précédent
- `[]` : Classe de caractères (ex : `[a-z]` pour toutes les lettres minuscules)
- `\d` : Un chiffre
- `\w` : Un caractère alphanumérique

Groupes de capture

Les **groupes de capture** en expressions régulières permettent de regrouper et d'extraire des sous-parties spécifiques d'une chaîne de caractères. Ils sont définis en entourant une partie de l'expression régulière entre parenthèses `( ... )`. Lorsqu'un modèle avec un groupe de capture est trouvé, le contenu entre parenthèses est sauvegardé et peut être récupéré séparément pour des traitements spécifiques. Par exemple, pour extraire le nom de domaine d'une adresse email, on pourrait utiliser l'expression régulière `/(.+)@(.+\..+)/`, où `( .+ )` capture tout ce qui précède le symbole `@` (l'identifiant), et `( .+\..+ )` capture le domaine.

Les groupes de capture sont également utiles pour **remplacer** certaines parties d'une chaîne avec `preg_replace()` en utilisant les références `$1`, `$2`, etc., qui correspondent aux groupes capturés, ce qui permet des manipulations de texte avancées.

Exemples

- `/^\d{5}$/` : Correspond à un code postal français à 5 chiffres
- `/^[\w\-\.]+@[a-z]+\.[a-z]{2,4}$/` : Correspond à un format d'email simple

Utilisation des Regex en PHP

PHP offre deux méthodes principales pour travailler avec les expressions régulières :

preg_match()

`preg_match()` recherche un modèle dans une chaîne et retourne `true` si le modèle est trouvé.

Exemple : Validation d'un email

```php
$email = "exemple@test.com";
if (preg_match("/^[\w\-\.]+@[a-z]+\.[a-z]{2,4}$/", $email)) {
    echo "Email valide.";
} else {
    echo "Email invalide.";
}
```

En PHP, les groupes de capture peuvent être accédés grâce à la fonction `preg_match()`, qui stocke chaque groupe trouvé dans un tableau indexé. Voici un exemple :

```php
$email = "exemple@domaine.com";
preg_match("/(.+)@(.+\..+)/", $email, $matches);
echo "Identifiant : " . $matches[1]; // Affiche "exemple"
echo "Domaine : " . $matches[2]; // Affiche "domaine.com"
```

preg_replace()

`preg_replace()` remplace toutes les occurrences d'un modèle par une autre chaîne.

Exemple : Masquer une partie d'un numéro de téléphone

```php
$telephone = "0123456789";
$masque = preg_replace("/\d{6}$/", "******", $telephone);
echo $masque;  // Affiche "0123******"
```

preg_split()

`preg_split()` divise une chaîne en un tableau selon un modèle.

Exemple :

```php
$phrase = "Bonjour, comment ça va ?";
$mots = preg_split("/[\s,]+/", $phrase);
print_r($mots);  // Sépare les mots en un tableau
```

📚 | TP 5 : Templating engine

> **INFO**
>
> Différemment des TD, les TP doivent s'effectuer seul. Ainsi, la réponse que vous trouverez à la problématique peut différer d'un étudiant à l'autre.

Faire le TP

1. Créer la structure du projet

> ⚠ **ATTENTION**
>
> Partir de la correction du TP4 (copier/coller du dossier). lien

- **Séance 5/TP5/**
  - public/
    - ...
  - framework/
    - `View.php` - Étend `Response.php` - Classe PHP représentant une vue/template
      - *$file* - String - Nom du template.
      - *$data* - Array - Variables à transférer dans le template.
      - Autres éléments de `Response.php`...
      - **render()** - Fonction - Permet de faire le rendu du moteur de templating. Insérer le code suivant : Fonction render()
    - ...
  - controllers/
    - `MyViewController.php` - Classe PHP qui renverra des vues.
    - Autres contrôleurs...
  - views/
    - `Footer.html` - Fichier HTML.
    - `MyTemplate.dacs` - Fichier/template HTML comportant des éléments de templating.
    - Autres fichiers HTML/dacs

2. Ajouter un autoloader

Un autoloader est un fichier qui charge automatiquement toutes les classes PHP de votre code, simplifiant ainsi leur import et leur utilisation.

À la racine de votre projet, ajouter ce fichier :

autoloader.php

```php
<?php

/* Code from internet, automatically load classes if they are currently
not loaded */

// https://www.php.net/manual/en/language.oop5.autoload.php

spl_autoload_register(function ($className) {
    $filename = __DIR__ . DIRECTORY_SEPARATOR . str_replace('\\', '/',
$className) . '.php';
    if (file_exists($filename)) require_once($filename);
});
```

Pour utiliser l'autoloader, il suffira d'ajouter cette ligne au tout début de votre `index.php` :

```php
// Import the autoloader, which loads all the namespaces and classes
require(__DIR__ . '/../autoloader.php');
```

3. Lancer le serveur

Une fois à la racine de votre projet `/Séance 5/TP5/`, il faut utiliser la commande suivante :

```
php -S localhost:8080 -t public
```

> ⚠ **ATTENTION**
>
> À noter que le dossier servi est `public` et non `TP5`. Cela permet d'avoir des accès utilisateurs différents, mon dossier public va être accessible par quasi-tout me monde (774). Là où mon dossier TP4 n'est accessible que par l'utilisateur courant (700).
>
> Cependant, cela n'a pas d'importance dans notre TP

4. Fonction render() de `View.php`

```php
public function render(): string {
    $file = file_get_contents('../views/' . $this->file);

    $file = str_replace("{{", "<?php", $file);
    $file = str_replace("}}", "?>", $file);

    $file = preg_replace("/@echo\(\s*(.+)\s*\)/", "<?php echo($1); ?>",
$file);

    $file = preg_replace("/@if\(\s*(.+)\s*\)/", "<?php if($1): ?>",
$file);
    $file = preg_replace("/@elseif\(\s*(.+)\s*\)/", "<?php else if($1): ?
>", $file);
    $file = str_replace("@else", "<?php else: ?>", $file);
    $file = str_replace("@endif", "<?php endif; ?>", $file);

    $file = preg_replace("/@foreach\(\s*(.+)\s*\)/", "<?php foreach($1): ?
>", $file);
    $file = str_replace("@endforeach", "<?php endforeach; ?>", $file);

    $file = preg_replace("/@require\(\s*(.+)\s*\)/", "<?php
require('../views/' . $1); ?>", $file);

    ob_start();
    extract($this->data);

    eval('?>' . $file);

    $content = ob_get_contents();
    ob_end_clean();

    return $content;
}
```

5. Exemple de vue dans un contrôleur

Nouvelle route

routes.php

```php
<?php

// Contain all the routes
return [
    // Create a new GET route linked to the /api route
    new Route(Method::GET, '/api', IndexController::class, 'index'),
    new Route(Method::GET, '/index', ViewController::class, 'index')
];
```

Ma vue

index.dacs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Index</title>
</head>
<body>
    <p>Page de test</p>
    <p>{{ echo $nom }}</p>

    @if($ma_var == 2)
        <p>ma var est égale à 3!!!</p>
    @else
        <p>ma var n'est pas égale à 3</p>
    @endif

    <ul>
        @foreach($items as $item)
            <li>élément : @echo($item)</li>
        @endforeach
    </ul>

    <footer>
        @require("footer.html")
    </footer>
</body>
</html>
```

Mon contrôleur

MyViewController.php

```php
<?php

namespace controllers;

use framework\Response;
use framework\ResponseType;
use framework\View;

/**
 * Basic HTTP controller
 */
class ViewController {
    /**
     * Route linked to /index
     *
     * @param array $request
     * @return Response
     */
    function index(array $request) : Response {
        $vars = [
            "nom" => "Julien",
            "ma_var" => 3,
            "items" => [
                1,
                2
            ]
        ];

        // Comme View étend Response, il peut ainsi être utilisé comme tel
        return new View("index.dacs", $vars);
    }
}
```