

# Créer des vues avec Le moteur de template TWIG

<https://twig.symfony.com/doc/>

- Optimiser la production de code HTML
- Séparer la forme et les contenus

# Optimiser la production de code HTML

- Dans un projet web **statique** « artisanal » les différentes pages sont codées en HTML :
  - Les éléments présents sur plusieurs pages doivent être **dupliqués** (copiés/collés)
    - Exemples : entête, base de page, menu, mise en page d'articles, blocs répétés d'une page à l'autre, etc.
  - Modifier un élément implique des corrections dans tous les fichiers => **risques d'erreurs ou d'omission**
- Les moteurs de template évitent cela
  - Exemple de moteurs : Smarty (PHP), Twig (Symfony), Blade (Laravel), Liquid (Shopify)...

# Le moteur de template TWIG

---

- Twig est un **moteur de template** qui permet
  - de produire du code HTML à partir de données
  - de séparer les **gabarits** ou **patrons** et les **contenus**
- **Gabarits** ou **patrons** de pages (template) est un fichier HTML à **trous** qui contient
  - Des balises HTML classiques
  - Des blocs dont le contenu change d'une page à l'autre
- Twig fusionne un gabarit + un jeu de données (un ensemble de variables) pour générer une page

# Gabarit ou Template **Twig**

---

- Un **gabarit** ou template Twig est un fichier HTML contenant des « balises » spéciales entre accolades
  - Des variables : `{{ pseudo }}` `{{ livre.titre }}`
  - Des instructions : `{% for %}` `{% if %}`
- Les **contenus** sont placés dans des variables PHP
  - Des variables simples : `$titre = 'Ma Galerie Web'`
  - Des tableaux : `$jours = ['lundi' ... 'dimanche']`
- On utilise Twig pour « **injecter** » les variables PHP dans le gabarit Twig et générer une page HTML
  - Un gabarit peut être utilisé avec différentes variables

# Exemple page HTML à trous

```
<html>
  <head> ... </head>
  <body>
    <h1> [ ] </h1>
    <p>Bienvenue [ ] </p>
    
      ... Un nombre variable d'images ...
    
  </body>
</html>
```

# Exemple contenus à mettre dans les trous

---

- Deux variables simples

```
$titre = 'Galerie';
```

```
$pseudo = 'David';
```

- Un tableau PHP pour les URL des images

```
$taburl = ['image1.jpeg', ..., 'image4.jpeg'];
```

# Exemple fichier 'essai.twig'

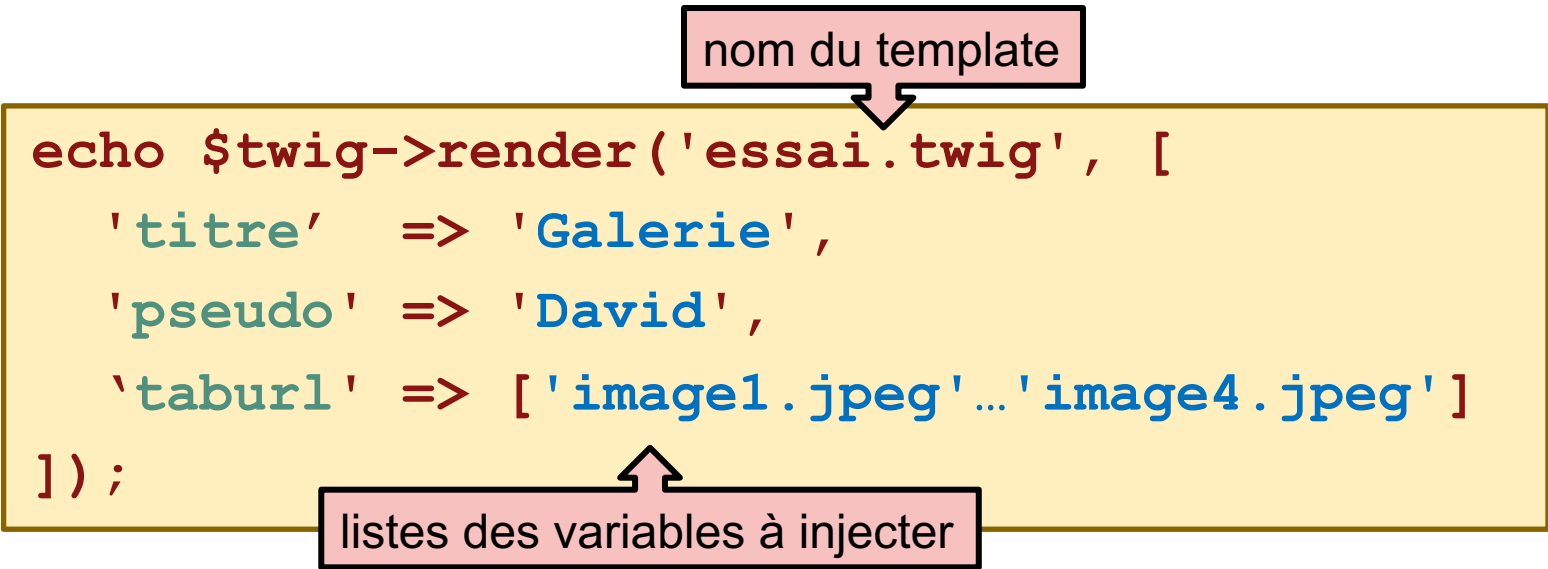
- Les contenus sont remplacés par du code Twig

```
<html>
  <head> ... </head>
  <body>
    <h1>{{ titre }}</h1>
    <p>Bienvenue {{ pseudo }}</p>
    {% for url in taburl %}
      <img src={{ url }}>
    {% endfor %}
  </body>
</html>
```

# Exemple fichier PHP utilisant Twig

```
$loader = new  
    \Twig\Loader\FilesystemLoader('templates');  
  
$twig = new \Twig\Environment($loader);
```

nom du template



```
echo $twig->render('essai.twig', [  
    'titre'    => 'Galerie',  
    'pseudo'   => 'David',  
    'taburl1'  => ['image1.jpeg'...'image4.jpeg']  
]);
```

listes des variables à injecter

# Exemple résultat de l'injection

---

```
<html>
  <head> ... </head>
  <body>
    <h1>Galerie</h1>
    <p>Bienvenue David</p>
    
    
    
    
  </body>
</html>
```

# Comment coder avec Twig ?

---

- Un projet utilisant Twig à besoin :
  - Le dossier « vendor » qui contient le code de Twig
  - Le dossier « include » qui contient les fichiers PHP n'étant pas affichés :
    - Déclaration des variables et des contenus
    - Fonctions utilitaires (fichier « twig.php »)
  - Le dossier « templates » qui contient les gabarits
    - Tous vos modèles de pages à « trous »
  - Les pages placées à la racine du projet
    - Une page = un fichier PHP liant des variables à un modèle
    - Il faut autant de fichier PHP que de pages sur le site

# Exemple description d'un client

- Les contenus peuvent être structurés dans des tableaux associatifs

```
$client = [  
    'nom' => 'De Niro',  
    'prénom' => 'Robert',  
    'adresse' => '1 rue principale, Haguenau',  
    'code_client' => 'XYZ123'  
];
```

- Rappel pour lire une case du tableau en PHP

```
echo $client['nom'];
```

# Exemple fiche « client »

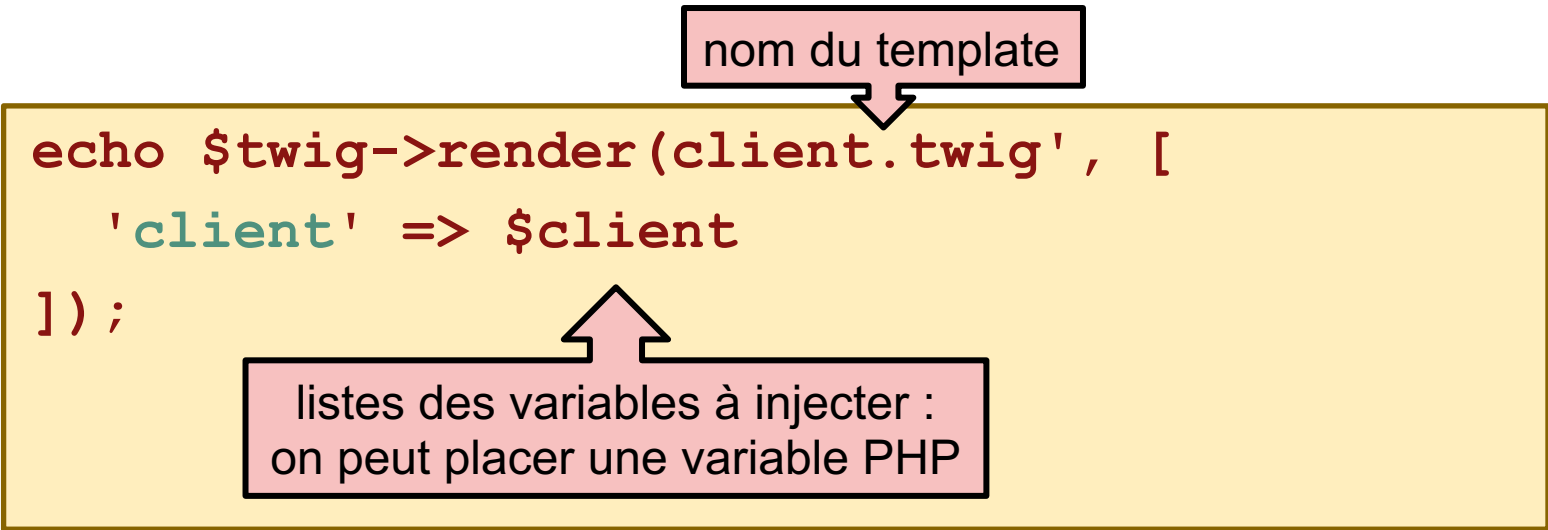
- Une structure HTML commune à tous les clients

```
// Fichier detail_client.twig
{# Modèle d'une page client #}
<div class="client">
  <h3>Client n°{{client.code_client }}</h3>
  <p>Nom : {{ client.nom }}</p>
  <p>Prénom : {{ client.prenom }}</p>
  <p>Adresse : {{ client.adresse }}</p>
</div>
```

# Exemple injection d'un client dans le template

```
$loader = new  
    \Twig\Loader\FilesystemLoader('templates');  
  
$twig = new \Twig\Environment($loader);
```

nom du template



```
echo $twig->render(client.twig', [  
    'client' => $client  
]);
```

listes des variables à injecter :  
on peut placer une variable PHP

- Dans un modèle on peut définir des blocs

```
{% block nom-du-bloc %}  
... Contenu du bloc  
{% endblock %}
```

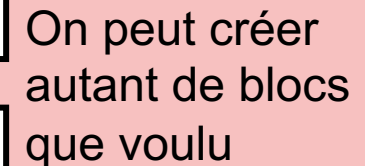
- Le bloc contient du code HTML ou Twig
- On peut définir plusieurs blocs dans une page
- Les blocs indiquent les parties du site qui changent d'une page à l'autre
- Ce qui est en dehors des blocs est identique sur toutes les pages (header, footer, etc.)

- Quand un modèle (fils) étend un 2<sup>ème</sup> modèle (père)
  - Il contient tout le contenu de son père
  - Il peut spécialiser (redéfinir) les blocs
  - Cela permet de réutiliser du code en ajoutant des fonctionnalités nouvelles
- Le contenu de fils est de ce fait très simple :

```
{% extends "père.twig" %}  
{% block nom-du-bloc %}  
... Nouveau contenu du bloc  
{% endblock %}
```

# Exemple créer un modèle de page

```
{# fichier base.twig #}  
<!DOCTYPE html>  
<html lang="fr">  
<head> ... feuilles de style ... </head>  
<body>  
  <header> ... les éléments communs de l'entête ...  
    <nav> ... le menu ... </nav>  
  </header>  
  <main>  
    {% block main %}  
    {% endblock %}  
  </main>  
  <footer> ... le bas de page ... </footer>  
</body>  
</html>
```



On peut créer  
autant de blocs  
que voulu

# Exemple utiliser le modèle de page

```
{# fichier accueil.twig #}  
{% extends "base.twig" %}  
{# les entêtes sont hérités de base.twig #}  
{# On définit un nouveau contenu pour 'main' #}  
{% block main %}  
<h1>Bienvenue  
    {{ utilisateur.prenom }}  
    {{ utilisateur.nom }}  
</h1>  
{% endblock %}
```

# Exemple étendre la hiérarchie

```
{# fichier deuxcolonnes.twig #}  
{% extends "base.twig" %}  
{# On définit un nouveau contenu pour 'main' #}  
{% block main %}  
<article class="gauche">  
    {% block gauche %}  
    {% endblock %}  
</article>  
<aside class="droite">  
    {% block droite %}  
    {% endblock %}  
</aside>  
{% endblock %}
```

# Exemple utiliser la hiérarchie

```
{% extends "deuxcolonnes.twig" %}  
{# On définit un nouveau contenu pour les blocs #}
```

```
{% block gauche %}  
<h2>{{ article.titre }}</h2>  
<p>{{ article.texte }}</p>  
{% endblock %}
```

```
{% block droite %}  
  <h3>{{ aside.titre }}</h3>  
  <p>{{ aside.texte }}</p>  
{% endblock %}
```

# Construire une hiérarchie

---

- On commence par écrire un fichier HTML correspondant à la maquette du site : « base.twig »
  - Exemple : page d'accueil ou page type
  - Éléments communs : entêtes, style, script JS, etc.
- Pour ajouter une page :
  - 1/ On repère le bloc qu'il faut changer
  - 2/ On déclare le bloc dans le père (s'il existe pas)
  - 3/ On étend le père et on spécialise le bloc
- Il est possible d'étendre un modèle déjà étendu
  - On construit alors une hiérarchie de modèles