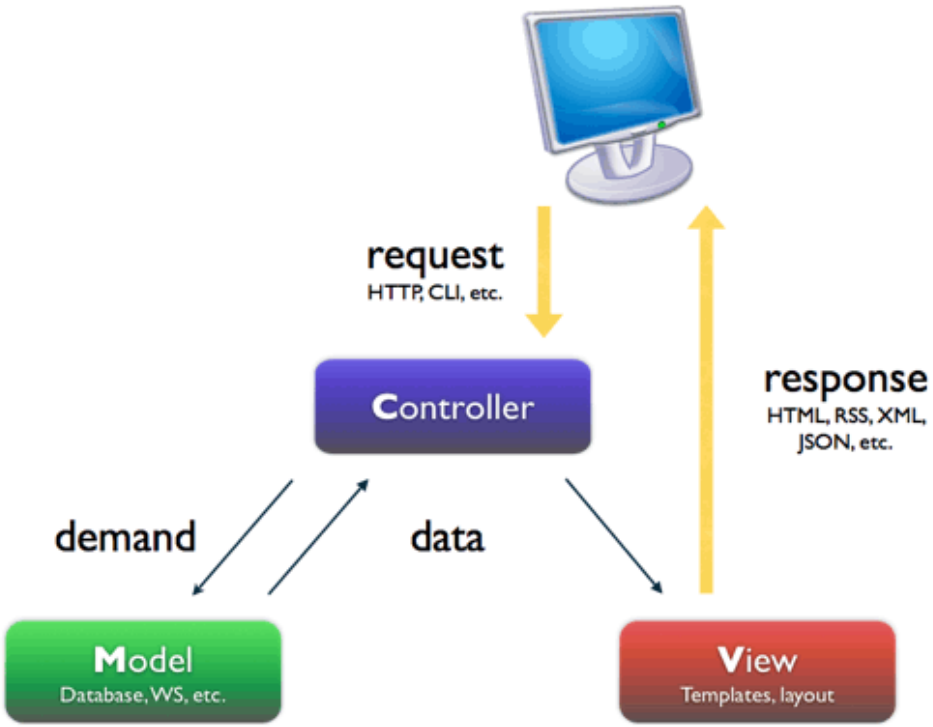
PHP

March 5, 2024

Structure générale MVC



Extrait de la documentation du framework Symfony

http://bpesquet.developpez.com

Le MVC

On peut distinguer trois parties :

- Modèles : gestion des données, accès aux base de données. Correspond généralement au besoin métier.
- Vue : gestion du html renvoyé au navigateur (contient du code dynamique, conditions, boucles...)
- Contrôleur : gère les données reçues de l'utilisateur, les transmet au modèle, et fait les appels au modèle pour obtenir les données. Il appelle les vues en fonction des contenus pour générer la page attendue

Pourquoi ce découpage ?

- Cloisonner le code en fonction des actions
- Rendre plus facilement maintenable le code
- Mutualiser des parties de codes
- Permettre le travail en équipe ou chacun pourra travailler sur sa partie

Le MVC

On peut distinguer trois parties :

- Modèles : gestion des données, accès aux base de données. Correspond généralement au besoin métier.
- Vue : gestion du html renvoyé au navigateur (contient du code dynamique, conditions, boucles...)
- Contrôleur : gère les données reçues de l'utilisateur, les transmet au modèle, et fait les appels au modèle pour obtenir les données. Il appelle les vues en fonction des contenus pour générer la page attendue

Pourquoi ce découpage ?

- Cloisonner le code en fonction des actions
- Rendre plus facilement maintenable le code
- Mutualiser des parties de codes
- Permettre le travail en équipe ou chacun pourra travailler sur sa partie

Le MVC

On peut distinguer trois parties :

- Modèles : gestion des données, accès aux base de données. Correspond généralement au besoin métier.
- Vue : gestion du html renvoyé au navigateur (contient du code dynamique, conditions, boucles...)
- Contrôleur : gère les données reçues de l'utilisateur, les transmet au modèle, et fait les appels au modèle pour obtenir les données. Il appelle les vues en fonction des contenus pour générer la page attendue

Pourquoi ce découpage ?

- Cloisonner le code en fonction des actions
- Rendre plus facilement maintenable le code
- Mutualiser des parties de codes
- Permettre le travail en équipe ou chacun pourra travailler sur sa partie

Le MVC

On peut distinguer trois parties :

- Modèles : gestion des données, accès aux base de données. Correspond généralement au besoin métier.
- Vue : gestion du html renvoyé au navigateur (contient du code dynamique, conditions, boucles...)
- Contrôleur : gère les données reçues de l'utilisateur, les transmet au modèle, et fait les appels au modèle pour obtenir les données. Il appelle les vues en fonction des contenus pour générer la page attendue

Pourquoi ce découpage ?

- Cloisonner le code en fonction des actions
- Rendre plus facilement maintenable le code
- Mutualiser des parties de codes
- Permettre le travail en équipe ou chacun pourra travailler sur sa partie

Le MVC

On peut distinguer trois parties :

- Modèles : gestion des données, accès aux base de données. Correspond généralement au besoin métier.
- Vue : gestion du html renvoyé au navigateur (contient du code dynamique, conditions, boucles...)
- Contrôleur : gère les données reçues de l'utilisateur, les transmet au modèle, et fait les appels au modèle pour obtenir les données. Il appelle les vues en fonction des contenus pour générer la page attendue

Pourquoi ce découpage ?

- Cloisonner le code en fonction des actions
- Rendre plus facilement maintenable le code
- Mutualiser des parties de codes
- Permettre le travail en équipe ou chacun pourra travailler sur sa partie

Le MVC

On peut distinguer trois parties :

- Modèles : gestion des données, accès aux base de données. Correspond généralement au besoin métier.
- Vue : gestion du html renvoyé au navigateur (contient du code dynamique, conditions, boucles...)
- Contrôleur : gère les données reçues de l'utilisateur, les transmet au modèle, et fait les appels au modèle pour obtenir les données. Il appelle les vues en fonction des contenus pour générer la page attendue

Pourquoi ce découpage ?

- Cloisonner le code en fonction des actions
- Rendre plus facilement maintenable le code
- Mutualiser des parties de codes
- Permettre le travail en équipe ou chacun pourra travailler sur sa partie

Le MVC

On peut distinguer trois parties :

- Modèles : gestion des données, accès aux base de données. Correspond généralement au besoin métier.
- Vue : gestion du html renvoyé au navigateur (contient du code dynamique, conditions, boucles...)
- Contrôleur : gère les données reçues de l'utilisateur, les transmet au modèle, et fait les appels au modèle pour obtenir les données. Il appelle les vues en fonction des contenus pour générer la page attendue

Pourquoi ce découpage ?

- Cloisonner le code en fonction des actions
- Rendre plus facilement maintenable le code
- Mutualiser des parties de codes
- Permettre le travail en équipe ou chacun pourra travailler sur sa partie

Les vues

On peut distinguer trois types de vues :

- Le formulaire de saisie qui vise à obtenir des informations de l'utilisateur. Celui-ci dispose en général d'un bouton pour envoyer au serveur les informations saisies.
- La page de réponse qui ne sert qu'à donner de l'information à l'utilisateur. Celle-ci dispose souvent d'un ou de plusieurs liens permettant à l'utilisateur de poursuivre l'application avec une autre page.
- La page mixte : le contrôleur a envoyé au client une page contenant des informations qu'il a générées. Cette même page va servir au client pour fournir au contrôleur de nouvelles informations provenant de l'utilisateur.

Les vues

On peut distinguer trois types de vues :

- Le formulaire de saisie qui vise à obtenir des informations de l'utilisateur. Celui-ci dispose en général d'un bouton pour envoyer au serveur les informations saisies.
- La page de réponse qui ne sert qu'à donner de l'information à l'utilisateur. Celle-ci dispose souvent d'un ou de plusieurs liens permettant à l'utilisateur de poursuivre l'application avec une autre page.
- La page mixte : le contrôleur a envoyé au client une page contenant des informations qu'il a générées. Cette même page va servir au client pour fournir au contrôleur de nouvelles informations provenant de l'utilisateur.

Les vues

On peut distinguer trois types de vues :

- Le formulaire de saisie qui vise à obtenir des informations de l'utilisateur. Celui-ci dispose en général d'un bouton pour envoyer au serveur les informations saisies.
- La page de réponse qui ne sert qu'à donner de l'information à l'utilisateur. Celle-ci dispose souvent d'un ou de plusieurs liens permettant à l'utilisateur de poursuivre l'application avec une autre page.
- La page mixte : le contrôleur a envoyé au client une page contenant des informations qu'il a générées. Cette même page va servir au client pour fournir au contrôleur de nouvelles informations provenant de l'utilisateur.

Les vues

On peut distinguer trois types de vues :

- Le formulaire de saisie qui vise à obtenir des informations de l'utilisateur. Celui-ci dispose en général d'un bouton pour envoyer au serveur les informations saisies.
- La page de réponse qui ne sert qu'à donner de l'information à l'utilisateur. Celle-ci dispose souvent d'un ou de plusieurs liens permettant à l'utilisateur de poursuivre l'application avec une autre page.
- La page mixte : le contrôleur a envoyé au client une page contenant des informations qu'il a générées. Cette même page va servir au client pour fournir au contrôleur de nouvelles informations provenant de l'utilisateur.

Les vues

Chaque vue donnera naissance à une page PHP. Pour chacune de celles-ci :

- On dessinera l'aspect de la page
- On déterminera quelles sont les parties dynamiques de celle-ci :
  Les informations à destination de l'utilisateur qui devront être fournies par le contrôleur en paramètres à la vue PHP.
- Les données de saisie qui devront être transmises au programme principal pour traitement.
  Celles-ci devront faire partie d'un formulaire HTML (balise <form>).

Les vues

Chaque vue donnera naissance à une page PHP. Pour chacune de celles-ci :

- On dessinera l'aspect de la page
- On déterminera quelles sont les parties dynamiques de celle-ci :
  Les informations à destination de l'utilisateur qui devront être fournies par le contrôleur en paramètres à la vue PHP.
- Les données de saisie qui devront être transmises au programme principal pour traitement.
  Celles-ci devront faire partie d'un formulaire HTML (balise <form>).

Les vues

Chaque vue donnera naissance à une page PHP. Pour chacune de celles-ci :

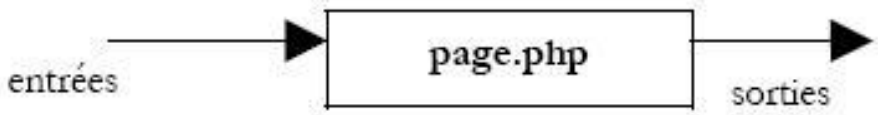
- On dessinera l'aspect de la page
- On déterminera quelles sont les parties dynamiques de celle-ci : **Les informations à destination de l'utilisateur qui devront être fournies par le contrôleur en paramètres à la vue PHP**.
- Les données de saisie qui devront être transmises au programme principal pour traitement.
  Celles-ci devront faire partie d'un formulaire HTML (balise <form>).

Les vues

Chaque vue donnera naissance à une page PHP. Pour chacune de celles-ci :

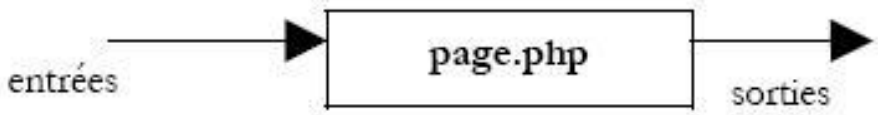
- On dessinera l'aspect de la page
- On déterminera quelles sont les parties dynamiques de celle-ci : **Les informations à destination de l'utilisateur qui devront être fournies par le contrôleur en paramètres à la vue PHP**.
- Les données de saisie qui devront être transmises au programme principal pour traitement.
  Celles-ci devront faire partie d'un formulaire HTML (balise <form>).

Les vues



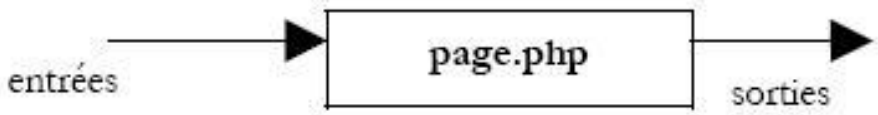
- Les entrées sont les données que devra fournir le contrôleur à la page PHP
- Les sorties sont les données que devra fournir la "page PHP" au contrôleur de l'aplication.
  - Elles font partie d'un formulaire HTML
  - et le contrôleur les récupérera par $_GET["param"] ou $_POST["param"].

Les vues



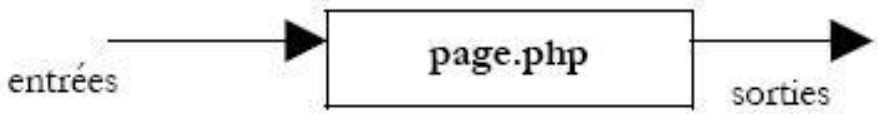
- Les entrées sont les données que devra fournir le contrôleur à la page PHP
- Les sorties sont les données que devra fournir la "page PHP" au contrôleur de l'aplication.
  - Elles font partie d'un formulaire HTML
  - et le contrôleur les récupérera par $_GET["param"] ou $_POST ["param"].

Les vues



- Les entrées sont les données que devra fournir le contrôleur à la page PHP
- Les sorties sont les données que devra fournir la "page PHP" au contrôleur de l'aplication.
  - Elles font partie d'un formulaire HTML
  - et le contrôleur les récupérera par $_GET["param"] ou $_POST ["param"].

Les vues



- Les entrées sont les données que devra fournir le contrôleur à la page PHP
- Les sorties sont les données que devra fournir la "page PHP" au contrôleur de l'aplication.
  - Elles font partie d'un formulaire HTML
  - et le contrôleur les récupérera par $_GET["param"] ou $_POST ["param"].

Les vues

Une solution simple est la suivante :

- Le contrôleur met dans un dictionnaire $dReponse les informations qu'il veut fournir à une vue V
- Le contrôleur fait afficher la vue V.
  Si celle-ci correspond au fichier source V.php, l'affichage est obtenu par ***include V.php***.

Les vues

Une solution simple est la suivante :

- Le contrôleur met dans un dictionnaire $dReponse les informations qu'il veut fournir à une vue V
- Le contrôleur fait afficher la vue V.
  Si celle-ci correspond au fichier source V.php, l'affichage est obtenu par ***include V.php***.

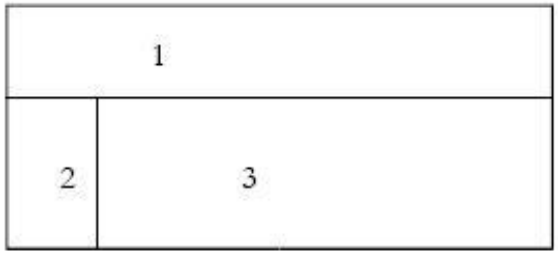
Les vues

Une solution simple est la suivante :

- Le contrôleur met dans un dictionnaire $dReponse les informations qu'il veut fournir à une vue V
- Le contrôleur fait afficher la vue V.
  Si celle-ci correspond au fichier source V.php, l'affichage est obtenu par ***include V.php***.

Les vue composées

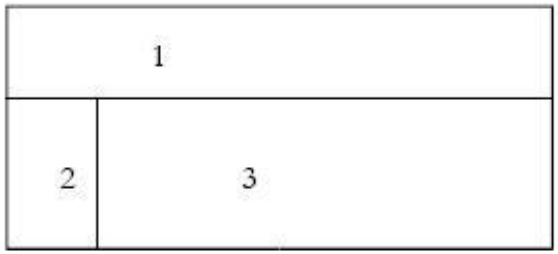
Une vue peut être une composition de vues. Exemple :



La zone 1 peut être un bandeau de titre, la zone 2 un bandeau de menu, la zone 3 une zone de contenu.

Les vue composées

Une vue peut être une composition de vues. Exemple :



La zone 1 peut être un bandeau de titre, la zone 2 un bandeau de menu, la zone 3 une zone de contenu.

Les vue composées

En PHP cette composition peut être obtenue par le code HTML/PHP suivant :

```
<table>
  <tr>
    <td><?php include zone1.php ?></td>
  </tr>
  <tr>
    <td><?php include zone2.php ?></td>
    <td><?php include zone3.php ?></td>
  </tr>
</table>
```

Les vue composées

En PHP cette composition peut être obtenue par le code HTML/PHP suivant :

```
<table>
  <tr>
    <td><?php include zone1.php ?></td>
  </tr>
  <tr>
    <td><?php include zone2.php ?></td>
    <td><?php include zone3.php ?></td>
  </tr>
</table>
```

Les vues composées

On écrira le code PHP/HTML de chaque vue élémentaire, par exemple :

```
<?php
  // éventuellement qqs initialisations notamment en
     phase de débogage
...
?>
<balise>
...   // on cherchera ici à minimiser le code php
</balise>
```

Remarque

Le cadre html, body, est généralement dans le modèle et pas dans la vue élémentaire

Les vues composées

On écrira le code PHP/HTML de chaque vue élémentaire, par exemple :

```
<?php
  // éventuellement qqs initialisations notamment en
     phase de débogage
...
?>
<balise>
...    // on cherchera ici à minimiser le code php
</balise>
```

Remarque

Le cadre html, body, est généralement dans le modèle et pas dans la vue élémentaire

Les vues composées

On écrira le code PHP/HTML de chaque vue élémentaire, par exemple :

```
<?php
  // éventuellement qqs initialisations notamment en
     phase de débogage
...
?>
<balise>
...    // on cherchera ici à minimiser le code php
</balise>
```

Remarque

Le cadre html, body, est généralement dans le modèle et pas dans la vue élémentaire

Les vue composées

- La vue composée peut être le format unique de l'application (= modèle de vues).
- Elle sera construite à partir des trois URL à charger dans les trois zones.
- On peut généraliser à plusieurs modèles possibles de vues.
- La réponse au client devra donc :
  - Fixer le modèle à utiliser
  - Fixer les éléments (vues élémentaires) à inclure dans celui-ci
  - Demander l'affichage du modèle

Les vue composées

- La vue composée peut être le format unique de l'application (= modèle de vues).
- Elle sera construite à partir des trois URL à charger dans les trois zones.
- On peut généraliser à plusieurs modèles possibles de vues.
- La réponse au client devra donc :
  - Fixer le modèle à utiliser
  - Fixer les éléments (vues élémentaires) à inclure dans celui-ci
  - Demander l'affichage du modèle

Les vue composées

- La vue composée peut être le format unique de l'application (= modèle de vues).
- Elle sera construite à partir des trois URL à charger dans les trois zones.
- On peut généraliser à plusieurs modèles possibles de vues.
- La réponse au client devra donc :
  - Fixer le modèle à utiliser
  - Fixer les éléments (vues élémentaires) à inclure dans celui-ci
  - Demander l'affichage du modèle

Les vue composées

- La vue composée peut être le format unique de l'application (= modèle de vues).
- Elle sera construite à partir des trois URL à charger dans les trois zones.
- On peut généraliser à plusieurs modèles possibles de vues.
- La réponse au client devra donc :
  - Fixer le modèle à utiliser
  - Fixer les éléments (vues élémentaires) à inclure dans celui-ci
  - Demander l'affichage du modèle

Les vue composées

- La vue composée peut être le format unique de l'application (= modèle de vues).
- Elle sera construite à partir des trois URL à charger dans les trois zones.
- On peut généraliser à plusieurs modèles possibles de vues.
- La réponse au client devra donc :
  - Fixer le modèle à utiliser
  - Fixer les éléments (vues élémentaires) à inclure dans celui-ci
  - Demander l'affichage du modèle

Les vue composées

- La vue composée peut être le format unique de l'application (= modèle de vues).
- Elle sera construite à partir des trois URL à charger dans les trois zones.
- On peut généraliser à plusieurs modèles possibles de vues.
- La réponse au client devra donc :
  - Fixer le modèle à utiliser
  - Fixer les éléments (vues élémentaires) à inclure dans celui-ci
  - Demander l'affichage du modèle

Les vue composées

- La vue composée peut être le format unique de l'application (= modèle de vues).
- Elle sera construite à partir des trois URL à charger dans les trois zones.
- On peut généraliser à plusieurs modèles possibles de vues.
- La réponse au client devra donc :
  - Fixer le modèle à utiliser
  - Fixer les éléments (vues élémentaires) à inclure dans celui-ci
  - Demander l'affichage du modèle

Vue Définition de la logique applicative

Les vues doivent retourner leur réponse vers un unique script (le contrôleur).

- contraintes : toutes les vues, liens, i.e. requête ou action de l'utilisateur doivent "pointer" vers le même script
- solution : il faut différentier les actions demandées à l'aide d'un paramètre

Pour les formulaires :

```
<form ... action="/C/main.php" method="post"...>
<input type="hidden" name="action" value="uneAction">
</form>
Si la requête provient d'un lien, on peut paramétrer
    celui-ci :
<a href="/C/main.php?action=uneAction">lien</a>
```

Pour les liens :

Vue Définition de la logique applicative

Les vues doivent retourner leur réponse vers un unique script (le contrôleur).

- contraintes : toutes les vues, liens, i.e. requête ou action de l'utilisateur doivent "pointer" vers le même script
- solution : il faut différentier les actions demandées à l'aide d'un paramètre

Pour les formulaires :

```
<form ... action="/C/main.php" method="post" ...>
<input type="hidden" name="action" value="uneAction">
</form>
```

Pour les liens :

```
Si la requête provient d'un lien, on peut paramétrer
    celui-ci :
<a href="/C/main.php?action=uneAction">lien</a>
```

Vue Définition de la logique applicative

Les vues doivent retourner leur réponse vers un unique script (le contrôleur).

- contraintes : toutes les vues, liens, i.e. requête ou action de l'utilisateur doivent "pointer" vers le même script
- solution : il faut différentier les actions demandées à l'aide d'un paramètre

Pour les formulaires :

```
<form ... action="/C/main.php" method="post" ...>
<input type="hidden" name="action" value="uneAction">
</form>
Si la requête provient d'un lien, on peut paramétrer
    celui-ci :
<a href="/C/main.php?action=uneAction">lien</a>
```

Pour les liens :

Vue Définition de la logique applicative

Les vues doivent retourner leur réponse vers un unique script (le contrôleur).

- contraintes : toutes les vues, liens, i.e. requête ou action de l'utilisateur doivent "pointer" vers le même script
- solution : il faut différentier les actions demandées à l'aide d'un paramètre

Pour les formulaires :

```
<form ... action="/C/main.php" method="post" ...>
<input type="hidden" name="action" value="uneAction">
```

Pour les liens :

```
</form>
Si la requête provient d'un lien, on peut paramétrer
    celui-ci :
<a href="/C/main.php?action=uneAction">lien</a>
```

Vue Définition de la logique applicative

Les vues doivent retourner leur réponse vers un unique script (le contrôleur).

- contraintes : toutes les vues, liens, i.e. requête ou action de l'utilisateur doivent "pointer" vers le même script
- solution : il faut différentier les actions demandées à l'aide d'un paramètre

Pour les formulaires :

```
<form ... action="/C/main.php" method="post"...>
<input type="hidden" name="action" value="uneAction">
</form>
Si la requête provient d'un lien, on peut paramétrer
    celui-ci :
<a href="/C/main.php?action=uneAction">lien</a>
```

Vue Définition de la logique applicative

Les vues doivent retourner leur réponse vers un unique script (le contrôleur).

- contraintes : toutes les vues, liens, i.e. requête ou action de l'utilisateur doivent "pointer" vers le même script
- solution : il faut différentier les actions demandées à l'aide d'un paramètre

Pour les formulaires :

```
<form ... action="/C/main.php" method="post"...>
<input type="hidden" name="action" value="uneAction">
</form>
Si la requête provient d'un lien, on peut paramétrer
    celui-ci :
<a href="/C/main.php?action=uneAction">lien</a>
```

Le contrôleur

Traitement par le contrôleur

- Le contrôleur récupère la valeur du paramètre ,
- puis délègue le traitement au module adéquate.

Si le contrôleur = un script unique (appelé main.php).

- Si l'application doit traiter des actions action1, action2. . . , actionx. On peut :
  - créer au sein du contrôleur une fonction par action (attention s'il y a beaucoup d'actions)
  - créer des scripts action1.php, action2.php. . . ,actionx.php (1 script par action).
  - Le contrôleur se contentera de charger les scripts idoines (include ou require,...).

Le contrôleur

Traitement par le contrôleur

- Le contrôleur récupère la valeur du paramètre ,
- puis délègue le traitement au module adéquate.

Si le contrôleur = un script unique (appelé main.php).

- Si l'application doit traiter des actions action1, action2. . . , actionx. On peut :
  - créer au sein du contrôleur une fonction par action (attention s'il y a beaucoup d'actions)
  - créer des scripts action1.php, action2.php. . . ,actionx.php (1 script par action).
  - Le contrôleur se contentera de charger les scripts idoines (include ou require,...).

Le contrôleur

Traitement par le contrôleur

- Le contrôleur récupère la valeur du paramètre ,
- puis délègue le traitement au module adéquate.

Si le contrôleur = un script unique (appelé main.php).

- Si l'application doit traiter des actions action1, action2. . . , actionx. On peut :
  - créer au sein du contrôleur une fonction par action (attention s'il y a beaucoup d'actions)
  - créer des scripts action1.php, action2.php. . . ,actionx.php (1 script par action).
  - Le contrôleur se contentera de charger les scripts idoines (include ou require,...).

Le contrôleur

Traitement par le contrôleur

- Le contrôleur récupère la valeur du paramètre ,
- puis délègue le traitement au module adéquate.

Si le contrôleur = un script unique (appelé main.php).

- Si l'application doit traiter des actions action1, action2. . . , actionx. On peut :
  - créer au sein du contrôleur une fonction par action (attention s'il y a beaucoup d'actions)
  - créer des scripts action1.php, action2.php. . . ,actionx.php (1 script par action).
  - Le contrôleur se contentera de charger les scripts idoines (include ou require,...).

Le contrôleur

Traitement par le contrôleur

- Le contrôleur récupère la valeur du paramètre ,
- puis délègue le traitement au module adéquate.

Si le contrôleur = un script unique (appelé main.php).

- Si l'application doit traiter des actions action1, action2. . . , actionx. On peut :
  - créer au sein du contrôleur une fonction par action (attention s'il y a beaucoup d'actions)
  - créer des scripts action1.php, action2.php. . . ,actionx.php (1 script par action).
  - Le contrôleur se contentera de charger les scripts idoines (include ou require,...).

Le contrôleur

Traitement par le contrôleur

- Le contrôleur récupère la valeur du paramètre ,
- puis délègue le traitement au module adéquate.

Si le contrôleur = un script unique (appelé main.php).

- Si l'application doit traiter des actions action1, action2. . . , actionx. On peut :
  - créer au sein du contrôleur une fonction par action (attention s'il y a beaucoup d'actions)
  - créer des scripts action1.php, action2.php. . . ,actionx.php (1 script par action).
  - Le contrôleur se contentera de charger les scripts idoines (include ou require,...).

Le contrôleur

Traitement par le contrôleur

- Le contrôleur récupère la valeur du paramètre ,
- puis délègue le traitement au module adéquate.

Si le contrôleur = un script unique (appelé main.php).

- Si l'application doit traiter des actions action1, action2. . . , actionx. On peut :
  - créer au sein du contrôleur une fonction par action (attention s'il y a beaucoup d'actions)
  - créer des scripts action1.php, action2.php. . . ,actionx.php (1 script par action).
  - Le contrôleur se contentera de charger les scripts idoines (include ou require,...).

Le contrôleur

Traitement par le contrôleur

- Le contrôleur récupère la valeur du paramètre ,
- puis délègue le traitement au module adéquate.

Si le contrôleur = un script unique (appelé main.php).

- Si l'application doit traiter des actions action1, action2. . . , actionx. On peut :
  - créer au sein du contrôleur une fonction par action (attention s'il y a beaucoup d'actions)
  - créer des scripts action1.php, action2.php. . . ,actionx.php (1 script par action).
  - Le contrôleur se contentera de charger les scripts idoines (include ou require,...).

Le contrôleur

Les avantages de cette dernière méthode :

- On travaille en-dehors du code du contrôleur.
- Chaque développeur peut travailler sur le script de traitement d'une action actionx de façon relativement indépendante (attention format d'échange de données).
- Réaliser l'inclusion du code du script actionx.php dans le code du contrôleur au moment de l'exécution :
  - allégement du code chargé en mémoire,
  - seul le code de traitement de l'action en cours est chargé.

Inconvénient

- L'inclusion peut engendrer des "collisions' de variables entre script.

Le contrôleur

Les avantages de cette dernière méthode :

- On travaille en-dehors du code du contrôleur.
- Chaque développeur peut travailler sur le script de traitement d'une action actionx de façon relativement indépendante (attention format d'échange de données).
- Réaliser l'inclusion du code du script actionx.php dans le code du contrôleur au moment de l'exécution :
  - allégement du code chargé en mémoire,
  - seul le code de traitement de l'action en cours est chargé.

Inconvénient

- L'inclusion peut engendrer des "collisions' de variables entre script.

Le contrôleur

Les avantages de cette dernière méthode :

- On travaille en-dehors du code du contrôleur.
- Chaque développeur peut travailler sur le script de traitement d'une action actionx de façon relativement indépendante (attention format d'échange de données).
- Réaliser l'inclusion du code du script actionx.php dans le code du contrôleur au moment de l'exécution :
  - allégement du code chargé en mémoire,
  - seul le code de traitement de l'action en cours est chargé.

Inconvénient

- L'inclusion peut engendrer des "collisions' de variables entre script.

Le contrôleur

Les avantages de cette dernière méthode :

- On travaille en-dehors du code du contrôleur.
- Chaque développeur peut travailler sur le script de traitement d'une action actionx de façon relativement indépendante (attention format d'échange de données).
- Réaliser l'inclusion du code du script actionx.php dans le code du contrôleur au moment de l'exécution :
  - allégement du code chargé en mémoire,
  - seul le code de traitement de l'action en cours est chargé.

Inconvénient

- L'inclusion peut engendrer des "collisions' de variables entre script.

Le contrôleur

Les avantages de cette dernière méthode :

- On travaille en-dehors du code du contrôleur.
- Chaque développeur peut travailler sur le script de traitement d'une action actionx de façon relativement indépendante (attention format d'échange de données).
- Réaliser l'inclusion du code du script actionx.php dans le code du contrôleur au moment de l'exécution :
  - allégement du code chargé en mémoire,
  - seul le code de traitement de l'action en cours est chargé.

Inconvénient

- L'inclusion peut engendrer des "collisions' de variables entre script.

Le contrôleur

Les avantages de cette dernière méthode :

- On travaille en-dehors du code du contrôleur.
- Chaque développeur peut travailler sur le script de traitement d'une action actionx de façon relativement indépendante (attention format d'échange de données).
- Réaliser l'inclusion du code du script actionx.php dans le code du contrôleur au moment de l'exécution :
  - allégement du code chargé en mémoire,
  - seul le code de traitement de l'action en cours est chargé.

Inconvénient

- L'inclusion peut engendrer des "collisions' de variables entre script.

Le contrôleur

Les avantages de cette dernière méthode :

- On travaille en-dehors du code du contrôleur.
- Chaque développeur peut travailler sur le script de traitement d'une action actionx de façon relativement indépendante (attention format d'échange de données).
- Réaliser l'inclusion du code du script actionx.php dans le code du contrôleur au moment de l'exécution :
  - allégement du code chargé en mémoire,
  - seul le code de traitement de l'action en cours est chargé.

Inconvénient

- L'inclusion peut engendrer des "collisions' de variables entre script.

Le contrôleur

Les avantages de cette dernière méthode :

- On travaille en-dehors du code du contrôleur.
- Chaque développeur peut travailler sur le script de traitement d'une action actionx de façon relativement indépendante (attention format d'échange de données).
- Réaliser l'inclusion du code du script actionx.php dans le code du contrôleur au moment de l'exécution :
  - allégement du code chargé en mémoire,
  - seul le code de traitement de l'action en cours est chargé.

Inconvénient

- L'inclusion peut engendrer des "collisions' de variables entre script.

Le contrôleur : conception

Nous verrons que nous pouvons faire en sorte de limiter les variables du contrôleur à quelques variables bien définies qu'il faudra alors éviter d'utiliser dans les scripts.

- On cherchera systématiquement à isoler le code métier ou le code d'accès aux données persistantes dans des modules distincts. Le contrôleur est une sorte de chef d'équipe qui reçoit des demandes de ses clients (clients web) et qui les fait exécuter par les personnes les plus appropriées (les modules métier).
- Lors de l'écriture du contrôleur, on déterminera **l'interface** des modules métier à écrire.
  - Cela si ces modules métier sont à construire.
  - S'ils existent déjà, alors le contrôleur s'adaptera à l'interface de ces modules existants.

Le contrôleur : conception

Nous verrons que nous pouvons faire en sorte de limiter les variables du contrôleur à quelques variables bien définies qu'il faudra alors éviter d'utiliser dans les scripts.

- On cherchera systématiquement à isoler le code métier ou le code d'accès aux données persistantes dans des modules distincts. Le contrôleur est une sorte de chef d'équipe qui reçoit des demandes de ses clients (clients web) et qui les fait exécuter par les personnes les plus appropriées (les modules métier).
- Lors de l'écriture du contrôleur, on déterminera **l'interface** des modules métier à écrire.
  - Cela si ces modules métier sont à construire.
  - S'ils existent déjà, alors le contrôleur s'adaptera à l'interface de ces modules existants.

Le contrôleur : conception

Nous verrons que nous pouvons faire en sorte de limiter les variables du contrôleur à quelques variables bien définies qu'il faudra alors éviter d'utiliser dans les scripts.

- On cherchera systématiquement à isoler le code métier ou le code d'accès aux données persistantes dans des modules distincts. Le contrôleur est une sorte de chef d'équipe qui reçoit des demandes de ses clients (clients web) et qui les fait exécuter par les personnes les plus appropriées (les modules métier).
- Lors de l'écriture du contrôleur, on déterminera **l'interface** des modules métier à écrire.
  - Cela si ces modules métier sont à construire.
  - S'ils existent déjà, alors le contrôleur s'adaptera à l'interface de ces modules existants.

Le contrôleur : conception

Nous verrons que nous pouvons faire en sorte de limiter les variables du contrôleur à quelques variables bien définies qu'il faudra alors éviter d'utiliser dans les scripts.

- On cherchera systématiquement à isoler le code métier ou le code d'accès aux données persistantes dans des modules distincts. Le contrôleur est une sorte de chef d'équipe qui reçoit des demandes de ses clients (clients web) et qui les fait exécuter par les personnes les plus appropriées (les modules métier).
- Lors de l'écriture du contrôleur, on déterminera **l'interface** des modules métier à écrire.
  - Cela si ces modules métier sont à construire.
  - S'ils existent déjà, alors le contrôleur s'adaptera à l'interface de ces modules existants.

Le contrôleur : démarche d'implémentation

- On écrira le squelette des modules métiers nécessaires au contrôleur.
- Par exemple, si celui-ci utilise un module getCodes rendant un tableau de chaînes de caractères, on implémente une "maquette" de type :

```
function getCodes(){
    return array("code1","code2","code3");
}
```

Le contrôleur : démarche d'implémentation

- On écrira le squelette des modules métiers nécessaires au contrôleur.
- Par exemple, si celui-ci utilise un module getCodes rendant un tableau de chaînes de caractères, on implémente une "maquette" de type :

```
function getCodes(){
    return array("code1","code2","code3");
}
```

Le contrôleur : démarche d'implémentation

- On écrira le squelette des modules métiers nécessaires au contrôleur.
- Par exemple, si celui-ci utilise un module getCodes rendant un tableau de chaînes de caractères, on implémente une "maquette" de type :

```
function getCodes(){
    return array("code1","code2","code3");
}
```

Classes métier : test

On écrit enfin les classes métier dont a besoin le contrôleur

- = développement classique d'une classe PHP,
- partie le plus souvent indépendante de toute application web.
- Elle sera testée en dehors de cet environnement,
- avec une application console par exemple et/ou des tests unitaires.
- Lorsqu'une classe métier a été écrite, on l'intègre dans l'architecture de déploiement de l'application web et on teste son intégration.
- On procédera ainsi pour chaque classe métier.

Classes métier : test

On écrit enfin les classes métier dont a besoin le contrôleur

- = développement classique d'une classe PHP,
- partie le plus souvent indépendante de toute application web.
- Elle sera testée en dehors de cet environnement,
- avec une application console par exemple et/ou des tests unitaires.
- Lorsqu'une classe métier a été écrite, on l'intègre dans l'architecture de déploiement de l'application web et on teste son intégration.
- On procédera ainsi pour chaque classe métier.

Classes métier : test

On écrit enfin les classes métier dont a besoin le contrôleur

- = développement classique d'une classe PHP,
- partie le plus souvent indépendante de toute application web.
- Elle sera testée en dehors de cet environnement,
- avec une application console par exemple et/ou des tests unitaires.
- Lorsqu'une classe métier a été écrite, on l'intègre dans l'architecture de déploiement de l'application web et on teste son intégration.
- On procédera ainsi pour chaque classe métier.

Classes métier : test

On écrit enfin les classes métier dont a besoin le contrôleur

- = développement classique d'une classe PHP,
- partie le plus souvent indépendante de toute application web.
- Elle sera testée en dehors de cet environnement,
- avec une application console par exemple et/ou des tests unitaires.
- Lorsqu'une classe métier a été écrite, on l'intègre dans l'architecture de déploiement de l'application web et on teste son intégration.
- On procédera ainsi pour chaque classe métier.

Classes métier : test

On écrit enfin les classes métier dont a besoin le contrôleur

- = développement classique d'une classe PHP,
- partie le plus souvent indépendante de toute application web.
- Elle sera testée en dehors de cet environnement,
- avec une application console par exemple et/ou des tests unitaires.
- Lorsqu'une classe métier a été écrite, on l'intègre dans l'architecture de déploiement de l'application web et on teste son intégration.
- On procédera ainsi pour chaque classe métier.

Classes métier : test

On écrit enfin les classes métier dont a besoin le contrôleur

- = développement classique d'une classe PHP,
- partie le plus souvent indépendante de toute application web.
- Elle sera testée en dehors de cet environnement,
- avec une application console par exemple et/ou des tests unitaires.
- Lorsqu'une classe métier a été écrite, on l'intègre dans l'architecture de déploiement de l'application web et on teste son intégration.
- On procédera ainsi pour chaque classe métier.

Structuration des répertoire

- faire un dossier public qui sera la racine et dans lequel il n'y a que l'index.php
- apache est configurer avec public comme racine
- Les autres répertoires de l'appli seront au dessus

Structuration des répertoire

- faire un dossier public qui sera la racine et dans lequel il n'y a que l'index.php
- apache est configurer avec public comme racine
- Les autres répertoires de l'appli seront au dessus

Structuration des répertoire

- faire un dossier public qui sera la racine et dans lequel il n'y a que l'index.php
- apache est configurer avec public comme racine
- Les autres répertoires de l'appli seront au dessus

Avant propos

Rappel ou synthèse du framework MVC:

- Rendre indépendant
  - les traitement sur les données (le métier)
  - l'affichage (les vue)
- se fait via une gestion centralisé dans le contrôleur qui connaît le métier et les vues
- "les dépendance sont reportées sur lui"

Avantages

- Les objets métiers et les vues peuvent être facilement réutilisées dans d'autres contextes
- l'extension de l'application est facilité
- les traitements communs à toutes les actions peuvent facilement être centralisée (dans le contrôleur)

Avant propos

Rappel ou synthèse du framework MVC:

- Rendre indépendant
  - les traitement sur les données (le métier)
  - l'affichage (les vue)
- se fait via une gestion centralisé dans le contrôleur qui connaît le métier et les vues
- "les dépendance sont reportées sur lui"

Avantages

- Les objets métiers et les vues peuvent être facilement réutilisées dans d'autres contextes
- l'extension de l'application est facilité
- les traitements communs à toutes les actions peuvent facilement être centralisée (dans le contrôleur)

Avant propos

Rappel ou synthèse du framework MVC:

- Rendre indépendant
  - les traitement sur les données (le métier)
  - l'affichage (les vue)
- se fait via une gestion centralisé dans le contrôleur qui connaît le métier et les vues
- "les dépendance sont reportées sur lui"

Avantages

- Les objets métiers et les vues peuvent être facilement réutilisées dans d'autres contextes
- l'extension de l'application est facilité
- les traitements communs à toutes les actions peuvent facilement être centralisée (dans le contrôleur)

Avant propos

Rappel ou synthèse du framework MVC:

- Rendre indépendant
  - les traitement sur les données (le métier)
  - l'affichage (les vue)
- se fait via une gestion centralisé dans le contrôleur qui connaît le métier et les vues
- "les dépendance sont reportées sur lui"

Avantages

- Les objets métiers et les vues peuvent être facilement réutilisées dans d'autres contextes
- l'extension de l'application est facilité
- les traitements communs à toutes les actions peuvent facilement être centralisée (dans le contrôleur)

Avant propos

Rappel ou synthèse du framework MVC:

- Rendre indépendant
  - les traitement sur les données (le métier)
  - l'affichage (les vue)
- se fait via une gestion centralisé dans le contrôleur qui connaît le métier et les vues
- "les dépendance sont reportées sur lui"

Avantages

- Les objets métiers et les vues peuvent être facilement réutilisées dans d'autres contextes
- l'extension de l'application est facilité
- les traitements communs à toutes les actions peuvent facilement être centralisée (dans le contrôleur)

Avant propos

Rappel ou synthèse du framework MVC:

- Rendre indépendant
  - les traitement sur les données (le métier)
  - l'affichage (les vue)
- se fait via une gestion centralisé dans le contrôleur qui connaît le métier et les vues
- "les dépendance sont reportées sur lui"

Avantages

- Les objets métiers et les vues peuvent être facilement réutilisées dans d'autres contextes
- l'extension de l'application est facilité
- les traitements communs à toutes les actions peuvent facilement être centralisée (dans le contrôleur)

Avant propos

Rappel ou synthèse du framework MVC:

- Rendre indépendant
  - les traitement sur les données (le métier)
  - l'affichage (les vue)
- se fait via une gestion centralisé dans le contrôleur qui connaît le métier et les vues
- "les dépendance sont reportées sur lui"

Avantages

- Les objets métiers et les vues peuvent être facilement réutilisées dans d'autres contextes
- l'extension de l'application est facilité
- les traitements communs à toutes les actions peuvent facilement être centralisée (dans le contrôleur)

Avant propos

Rappel ou synthèse du framework MVC:

- Rendre indépendant
  - les traitement sur les données (le métier)
  - l'affichage (les vue)
- se fait via une gestion centralisé dans le contrôleur qui connaît le métier et les vues
- "les dépendance sont reportées sur lui"

Avantages

- Les objets métiers et les vues peuvent être facilement réutilisées dans d'autres contextes
- l'extension de l'application est facilité
- les traitements communs à toutes les actions peuvent facilement être centralisée (dans le contrôleur)

Avant propos

Rappel ou synthèse du framework MVC:

- Rendre indépendant
  - les traitement sur les données (le métier)
  - l'affichage (les vue)
- se fait via une gestion centralisé dans le contrôleur qui connaît le métier et les vues
- "les dépendance sont reportées sur lui"

Avantages

- Les objets métiers et les vues peuvent être facilement réutilisées dans d'autres contextes
- l'extension de l'application est facilité
- les traitements communs à toutes les actions peuvent facilement être centralisée (dans le contrôleur)

Le contrôleur

Il doit

- identifier l'action a réaliser
- déclencher les traitements auprès des objets métiers
- récupérer les données de retour
- passer les données de retour à objet vue pour l'affichage (construction de la page html de retour)

Le contrôleur

Il doit

- identifier l'action a réaliser
- déclencher les traitements auprès des objets métiers
- récupérer les données de retour
- passer les données de retour à objet vue pour l'affichage (construction de la page html de retour)

Le contrôleur

Il doit

- identifier l'action a réaliser
- déclencher les traitements auprès des objets métiers
- récupérer les données de retour
- passer les données de retour à objet vue pour l'affichage (construction de la page html de retour)

Le contrôleur

Il doit

- identifier l'action a réaliser
- déclencher les traitements auprès des objets métiers
- récupérer les données de retour
- passer les données de retour à objet vue pour l'affichage (construction de la page html de retour)

Le contrôleur

Il peut en plus, s'il n'y a pas d'objet applications ou autre en amont

- appliquer les règles de sécurités (vérifier l'identification, nettoyer les paramètres reçu,...,
- Gérer les connexions (session, cookies,...)
- ...

Une histoire sans fin

- La structure peut être très complexe et très générique,
- tout votre talent résidera dans voter capacité à proposer la structure adaptée à la situation
- bien identifier les résultats attendus et **les évolutions à venir**

Le contrôleur

Il peut en plus, s'il n'y a pas d'objet applications ou autre en amont

- appliquer les règles de sécurités (vérifier l'identification, nettoyer les paramètres reçu,...,
- Gérer les connexions (session, cookies,...)
- ...

Une histoire sans fin

- La structure peut être très complexe et très générique,
- tout votre talent résidera dans voter capacité à proposer la structure adaptée à la situation
- bien identifier les résultats attendus et **les évolutions à venir**

Le contrôleur

Il peut en plus, s'il n'y a pas d'objet applications ou autre en amont

- appliquer les règles de sécurités (vérifier l'identification, nettoyer les paramètres reçu,...,
- Gérer les connexions (session, cookies,...)
- ...

Une histoire sans fin

- La structure peut être très complexe et très générique,
- tout votre talent résidera dans voter capacité à proposer la structure adaptée à la situation
- bien identifier les résultats attendus et **les évolutions à venir**

Le contrôleur

Il peut en plus, s'il n'y a pas d'objet applications ou autre en amont

- appliquer les règles de sécurités (vérifier l'identification, nettoyer les paramètres reçu,...,
- Gérer les connexions (session, cookies,...)
- ...

Une histoire sans fin

- La structure peut être très complexe et très générique,
- tout votre talent résidera dans voter capacité à proposer la structure adaptée à la situation
- bien identifier les résultats attendus et **les évolutions à venir**

Le contrôleur

Il peut en plus, s'il n'y a pas d'objet applications ou autre en amont

- appliquer les règles de sécurités (vérifier l'identification, nettoyer les paramètres reçu,...,
- Gérer les connexions (session, cookies,...)
- ...

Une histoire sans fin

- La structure peut être très complexe et très générique,
- tout votre talent résidera dans voter capacité à proposer la structure adaptée à la situation
- bien identifier les résultats attendus et **les évolutions à venir**

Le contrôleur

Il peut en plus, s'il n'y a pas d'objet applications ou autre en amont

- appliquer les règles de sécurités (vérifier l'identification, nettoyer les paramètres reçu,...,
- Gérer les connexions (session, cookies,...)
- ...

Une histoire sans fin

- La structure peut être très complexe et très générique,
- tout votre talent résidera dans voter capacité à proposer la structure adaptée à la situation
- bien identifier les résultats attendus et **les évolutions à venir**

Le contrôleur

Il peut en plus, s'il n'y a pas d'objet applications ou autre en amont

- appliquer les règles de sécurités (vérifier l'identification, nettoyer les paramètres reçu,...,
- Gérer les connexions (session, cookies,...)
- ...

Une histoire sans fin

- La structure peut être très complexe et très générique,
- tout votre talent résidera dans voter capacité à proposer la structure adaptée à la situation
- bien identifier les résultats attendus et les évolutions à venir

Le contrôleur

Il peut en plus, s'il n'y a pas d'objet applications ou autre en amont

- appliquer les règles de sécurités (vérifier l'identification, nettoyer les paramètres reçu,...,
- Gérer les connexions (session, cookies,...)
- ...

Une histoire sans fin

- La structure peut être très complexe et très générique,
- tout votre talent résidera dans voter capacité à proposer la structure adaptée à la situation
- bien identifier les résultats attendus et **les évolutions à venir**

Le contrôleur

Il peut en plus, s'il n'y a pas d'objet applications ou autre en amont

- appliquer les règles de sécurités (vérifier l'identification, nettoyer les paramètres reçu,...,
- Gérer les connexions (session, cookies,...)
- ...

Une histoire sans fin

- La structure peut être très complexe et très générique,
- tout votre talent résidera dans voter capacité à proposer la structure adaptée à la situation
- bien identifier les résultats attendus et **les évolutions à venir**

Le modèle

- Le modèle représente la structure des données de l'application et définit la logique métier
- Il contient généralement les classes et les méthodes,
- pour interagir avec la base de données et effectuer des opérations de traitement des données
- Le modèle ne se soucie pas de l'interface utilisateur,
- ou de la manière dont les données sont présentées à l'utilisateur

Le modèle

- Le modèle représente la structure des données de l'application et définit la logique métier
- Il contient généralement les classes et les méthodes,
- pour interagir avec la base de données et effectuer des opérations de traitement des données
- Le modèle ne se soucie pas de l'interface utilisateur,
- ou de la manière dont les données sont présentées à l'utilisateur

Le modèle

- Le modèle représente la structure des données de l'application et définit la logique métier
- Il contient généralement les classes et les méthodes,
- pour interagir avec la base de données et effectuer des opérations de traitement des données
- Le modèle ne se soucie pas de l'interface utilisateur,
- ou de la manière dont les données sont présentées à l'utilisateur

Le modèle

- Le modèle représente la structure des données de l'application et définit la logique métier
- Il contient généralement les classes et les méthodes,
- pour interagir avec la base de données et effectuer des opérations de traitement des données
- Le modèle ne se soucie pas de l'interface utilisateur,
- ou de la manière dont les données sont présentées à l'utilisateur

Le modèle

- Le modèle représente la structure des données de l'application et définit la logique métier
- Il contient généralement les classes et les méthodes,
- pour interagir avec la base de données et effectuer des opérations de traitement des données
- Le modèle ne se soucie pas de l'interface utilisateur,
- ou de la manière dont les données sont présentées à l'utilisateur

Application MVC de gestion de produits

- Pour cela on implémente notre modèle dans un dossier Model on aura : Model/Produit.php

```
class Produit {
    public $id_produit;
    public $nom;
    public $prix;
    private $liste_produits = []; // Liste des produits
    public function __construct($id_produit, $nom, $prix)
        {
        $this->id_produit = $id_produit;
        $this->nom = $nom;
        $this->prix = $prix;
    }
    // Méthode pour ajouter un produit à la liste
    public function ajouter_produit($produit) {
        $this->liste_produits[] = $produit;
    }
```

Application MVC de gestion de produits

- Pour cela on implémente notre modèle dans un dossier Model on aura : Model/Produit.php

```
class Produit {
    public $id_produit;
    public $nom;
    public $prix;
    private $liste_produits = []; // Liste des produits
    public function __construct($id_produit, $nom, $prix)
        {
        $this->id_produit = $id_produit;
        $this->nom = $nom;
        $this->prix = $prix;
    }
    // Méthode pour ajouter un produit à la liste
    public function ajouter_produit($produit) {
        $this->liste_produits[] = $produit;
    }
```

Application

```
// Méthode pour supprimer un produit de la liste par son
    ID
    public function supprimer_produit($id_produit) {
        foreach ($this->liste_produits as $key =>
            $produit) {
            if ($produit->id_produit == $id_produit) {
                unset($this->liste_produits[$key]);
                return;
            }
        }
    }

    // Méthode pour récupérer tous les produits de la
        liste
    public function liste_produits() {
        return $this->liste_produits;
    }
}
```

Application MVC de gestion de produits

- Pour cela on implémente notre vue dans un dossier Vue on aura : Vue/vueProduit.php

```
class VueProduit {
    public function afficher_produits($produits) {
        echo "<h2>Produits:</h2>";
        echo "<ul>";
        foreach ($produits as $produit) {
            echo "<li>{$produit->nom} - {$produit->prix}
                </li>";
        }
        echo "</ul>";
    }

    public function afficher_message($message) {
        echo "<p>{$message}</p>";
    }
}
```

Application MVC de gestion de produits

- Pour cela on implémente notre vue dans un dossier Vue on aura : Vue/vueProduit.php

```
class VueProduit {
    public function afficher_produits($produits) {
        echo "<h2>Produits:</h2>";
        echo "<ul>";
        foreach ($produits as $produit) {
            echo "<li>{$produit->nom} - {$produit->prix}
                </li>";
        }
        echo "</ul>";
    }

    public function afficher_message($message) {
        echo "<p>{$message}</p>";
    }
}
```

Application MVC de gestion de produits

- Pour cela on implémente notre vue dans un dossier Controleur on aura : Controleur/controleurProduit.php

```
class ControleurProduit {
    private $modele;
    private $vue;

    public function __construct($modele, $vue) {
        $this->modele = $modele;
        $this->vue = $vue;
    }

    public function ajouter_produit($produit) {
        $this->modele->ajouter_produit($produit);
        $this->vue->afficher_message("Produit ajouté avec
            succès.");
    }
```

Application MVC de gestion de produits

- Pour cela on implémente notre vue dans un dossier Controleur on aura : Controleur/controleurProduit.php

```
class ControleurProduit {
    private $modele;
    private $vue;

    public function __construct($modele, $vue) {
        $this->modele = $modele;
        $this->vue = $vue;
    }

    public function ajouter_produit($produit) {
        $this->modele->ajouter_produit($produit);
        $this->vue->afficher_message("Produit ajouté avec
            succès.");
    }
```

Application

```
public function supprimer_produit($id_produit) {
        $this->modele->supprimer_produit($id_produit);
        $this->vue->afficher_message("Produit supprimé
            avec succès.");
    }

    public function afficher_produits() {
        $produits = $this->modele->liste_produits();
        $this->vue->afficher_produits($produits);
    }
}
```

Application MVC de gestion de produits

- Pour cela on implémente notre fichier index on aura : index.php

```
require_once 'Model/Produit.php';
require_once 'Vue/vueProduit.php';
require_once 'Controleur/controleurProduit.php';

// Initialisation des composants MVC
$liste_produits = new Produits();
$vue = new vueProduit();
$controleur = new controleurProduit($liste_produits, $vue
    );
```

Application MVC de gestion de produits

- Pour cela on implémente notre fichier index on aura : index.php

```
require_once 'Model/Produit.php';
require_once 'Vue/vueProduit.php';
require_once 'Controleur/controleurProduit.php';

// Initialisation des composants MVC
$liste_produits = new Produits();
$vue = new vueProduit();
$controleur = new controleurProduit($liste_produits, $vue
    );
```

Application

```
// Ajout de quelques produits
$controleur->ajouter_produit(new Produit(1, "Ordinateur
    portable", 799));
$controleur->ajouter_produit(new Produit(2, "Téléphone
    portable", 599));
$controleur->ajouter_produit(new Produit(3, "Casque audio
    ", 99.99));

// Affichage de tous les produits
$controleur->afficher_produits();
// Suppression d'un produit
$controleur->supprimer_produit(2);
// Affichage mis à jour des produits
$controleur->afficher_produits();
```