TP 1 : Architecture d'un projet d'application web

L'objectif de ce TP est de présenter une structure standard et modulaire pour un projet d'application PHP. Une proposition d'architecture MVC appuyée sur une programmation orientée objet va vous être présentée. Dans ce TP, vous devrez comprendre, implémenter et tester cette architecture.

Pré-requis :

Afin de bien comprendre l'architecture proposée, il sera utile de visionner le code des fichiers évoqués. Un squelette de projet prêt à l'emploi est téléchargeable sur la plateforme e-uapv. Dézipper l'archive dans un répertoire dénommé IMPERATIVEMENT public_html à la racine de votre home. Le serveur Apache lancé sur la machine pedago01a.univ-avignon.fr ira lire directement à la racine du répertoire public_html dès lors que vous utiliserez l'url http://pedago01a.univ-avignon.fr/~uapvXXXX/ dans un navigateur. Vous devrez également donner (1) les droits d'exécution sur vos répertoires home, public_html et tout répertoire à l'intérieur de public_html, accessibles par le serveur apache - droits sur "autres", (2) les droits de lecture à tous les fichiers accessibles par ce même serveur - droits sur "autres".

L'architecture proposée sera basée sur une implémentation simplifiée de l'architecture MVC (Modèle Vue Contrôleur) vue en cours.

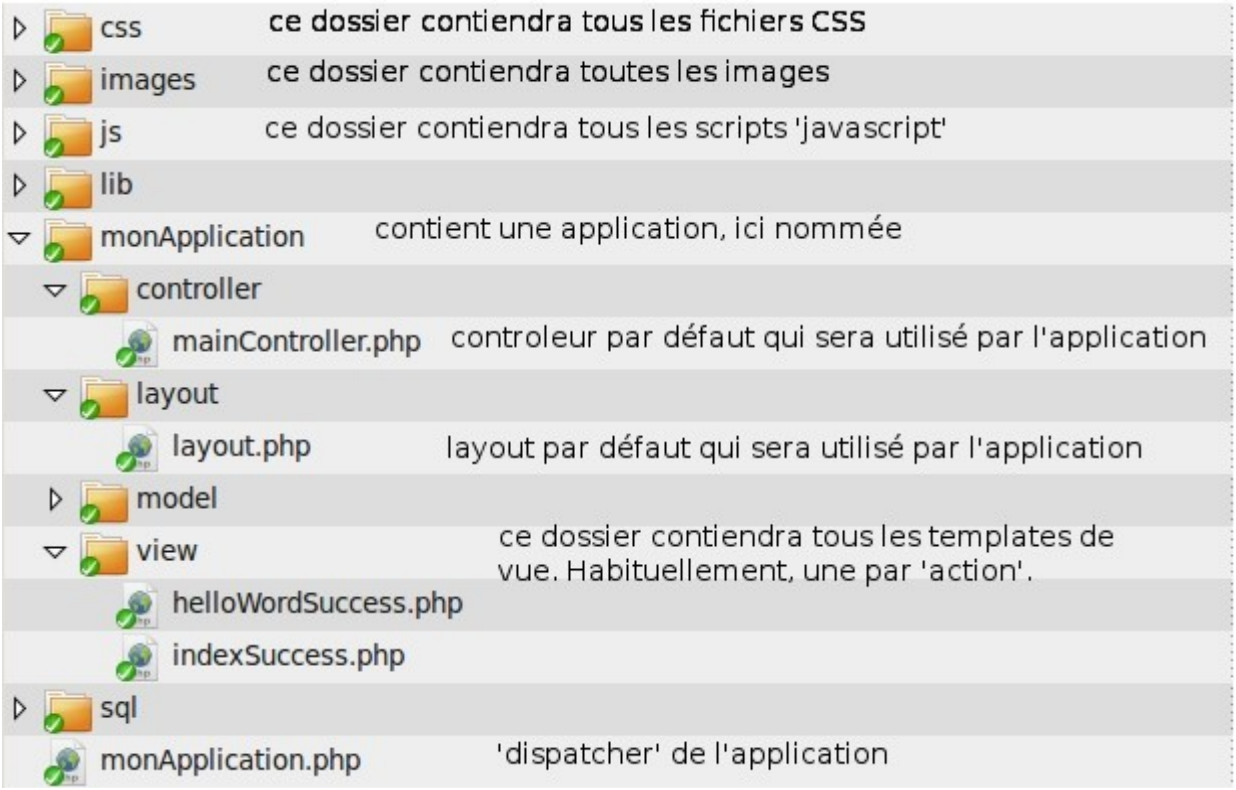


Figure 1: structure de l'architecture MVC

Fonctionnement général :

Exemple avec l'action *helloWord*

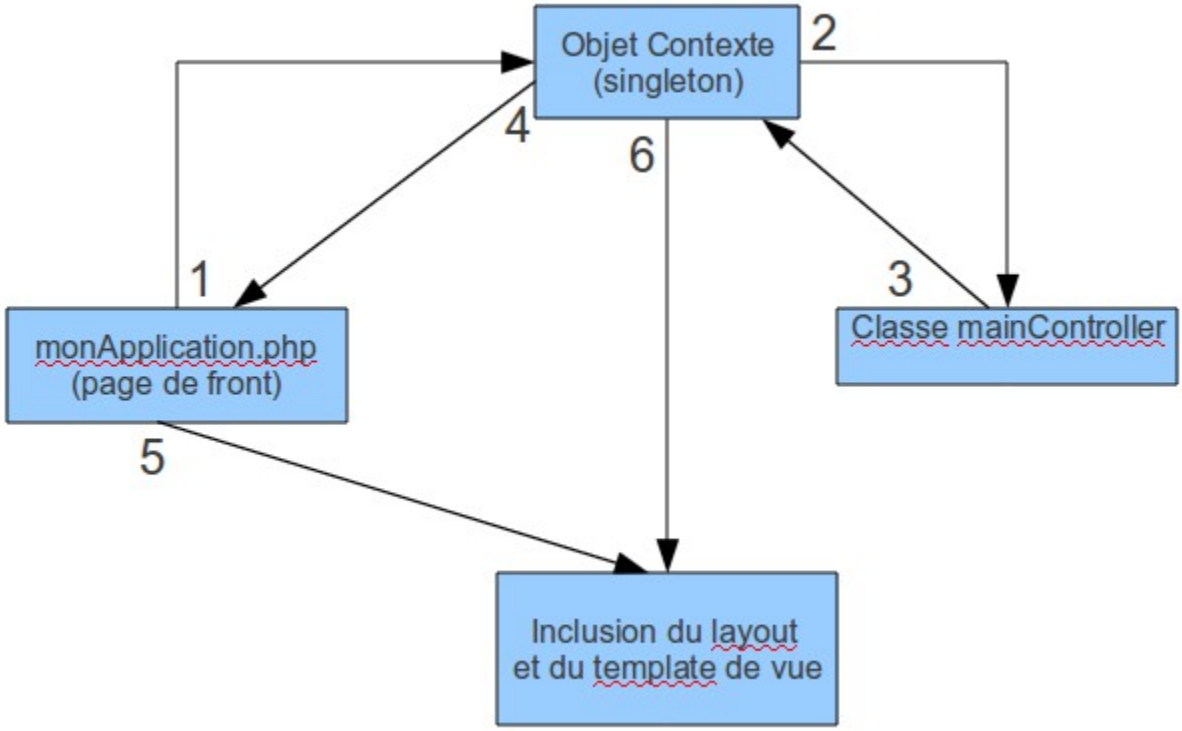
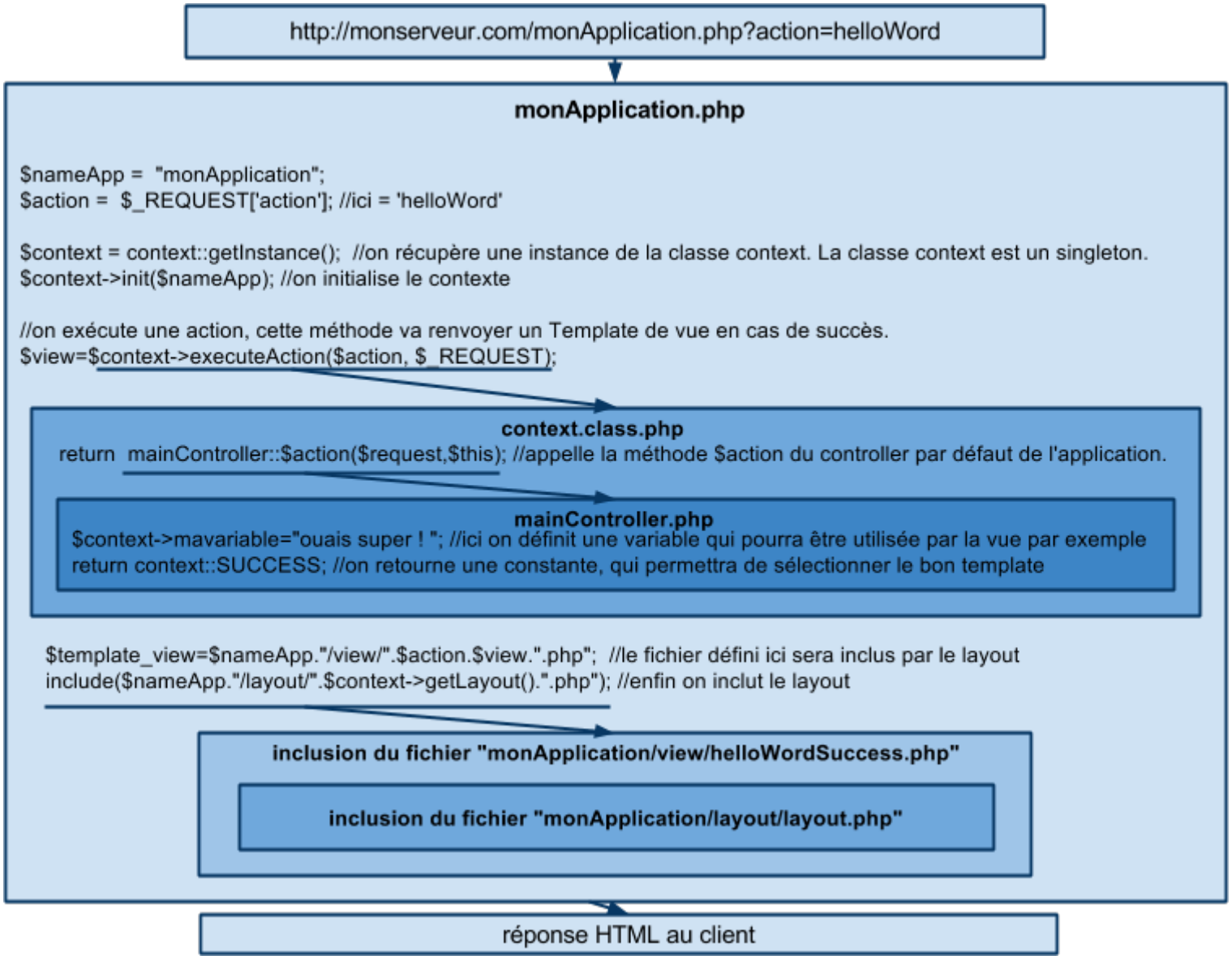


Figure 2: schématisation de l'appel à une fonction dans l'architecture MVC

1. La méthode *executeAction('helloWord',$_REQUEST)* de l'objet *contexte* est appelée.
2. La méthode statique *helloWord* du contrôleur (classe mainController) est appelée. Elle contiendra **tout le code relatif à cette action**.
3. Toutes les éventuelles données devant être affichées sont enregistrées dans l'instance de l'objet *contexte*. La méthode *mainController::helloWord* renvoie le type de vue qui devra être chargé : 3 types de vues possibles sont définis par 3 constantes dans la classe Contexte : *SUCCESS*, *ERROR* et *NONE*.
4. La page de front a maintenant le type de vue qui devra être chargé (valeur retournée par la méthode *executeAction* appelée en 1).
5. Suivant le type de vue renvoyé par l'action, le layout est inclus, il s'agit ici de *layout.php*. Il va lui même charger le template de la vue correspondante. Dans le cas de l'action *helloWord,* c'est le fichier *view/helloWordSuccess.php* qui sera inclus. Comme décrit précédemment il y a 3 types de vues possibles :
    - *SUCCESS :* le layout est appelé incluant le fichier *view/$actionSuccess.php*.
    - *ERROR :* le layout est appelé incluant le fichier *view/$actionError.php.*
    - *NONE :* dans ce cas là, rien n'est inclus. Ce type de réponse est utilisé pour une méthode AJAX, pour laquelle un flux JSON ou XML est attendu par exemple.
6. Le template de vue peut faire appel à toutes les données enregistrées dans l'instance de l'objet *contexte*.



igure 3: code correspondant à l'appel d'une action

Chaque application possède sa propre structure dans un dossier. Toutes les actions (et donc les urls) de votre application pointeront sur *"monApplication.php"* qui est le handler/dispatcher d'accès à cette application.

A chaque fois qu'une action est appelée, une instance unique (singleton) de la classe *"context"* est créée. Le *"context"* sert à centraliser toutes les informations pour l'action en cours. Par exemple, les variables définies par le contrôleur, l'accès aux données de session, etc. Cette instance unique de *"context"* est accessible à n'importe quel endroit du code de l'application, ce qui permet une communication rapide et efficace des différents composants.

L'objet *"context"* permet également l'exécution d'une "action", qui lors de son appel va appeler la méthode statique correspondant du contrôleur de l'application (représenté ici par la classe *mainController*). Dans le contrôleur on pourra effectuer des opérations sur le modèle, les classes métiers, etc...
Le contrôleur renvoie une constante définie dans la classe *"context"*, qui suivant sa valeur permettra l'appel d'un certain type de vue. S'il s'agit par exemple de la valeur *"Success"* récupérée dans la variable *$view*, la vue *$action.$view.php* sera appelée comme expliqué dans le schéma ci-dessus.

Vous devez comprendre, l'architecture présentée ci-dessus. Pour cela, n'hésitez pas à ouvrir les différents fichiers fournis, à commenter le code déja présent, etc. Prenez le temps de comprendre ce que vous faites et n'hésitez pas à poser des questions, c'est capital pour la suite !

Prise en main :

1. Déployez l'archive et personnalisez la (changez le nom de l'application...).
2. Implémentez la structure et le contrôleur objet de votre application. Testez le "helloWord" pour vérifier son bon fonctionnement.
3. Définissez une action "superTest" qui prend deux paramètres dans l'url (param1 et param2 par exemple) et affichez ce message: "j'ai compris <VALEUR PARAM1> , super : <VALEUR PARAM2>".
4. Commencez à concevoir le fichier *layout* pour afficher un cadre à votre application, et un menu.

Ça devient sérieux !

A présent, vous allez implémenter la fonction d'authentification de l'application :

1. Créez une action *login* par l'ajout d'une méthode correspondante depuis le contrôleur. Elle doit afficher un formulaire d'authentification classique (login/password). Ce formulaire enverra ses données sur cette même action (donc *votreAppli.php?action=login*)
2. Ajoutez dans cette action le traitement des données de ce formulaire pour gérer l'authentification
    a. utilisez la méthode prête à l'emploi getUserByLoginAndPass() qui renvoie false en cas d'échec, ou un objet de la classe *utilisateur* en cas de succès.
    b. en cas d'échec affichez un message.
    c. en cas de succès enregistrez l'utilisateur courant en session (utilisez les méthodes de la classe *context*).
    d. faites en sorte qu'il faille être identifié pour exécuter la méthode helloWord. Testez !