

Rapport de projet

Modap Systèmes et  
Applications Distribuées

Ludovic Eschard  
Julien Manteau

2A TR ENSEEIHT

# Table des matières

<b>Présentation générale</b>	<b>1</b>
Objectif du sujet	1
Notre Objectif	1
<b>Architecture</b>	<b>2</b>
Framework	3
Lien classes - base de donnée : couche ORM	4
<b>Système distribué</b>	<b>6</b>
Processes	6
Monitoring	7
Restriction d'accès au machines distantes	7
<b>Déploiement</b>	<b>8</b>
Système requis	8
Installeur	8
Ré-initialisation de la base de donnée.	8
<b>Intérêts du framework</b>	<b>8</b>
Instantiation de classe communes	8
Library de génération de code	9
ajout dynamique de header	9
<b>Sécurité</b>	<b>9</b>
Mot de passe	9
Autorisation d'accès aux contrôleurs	10
<b>Interface utilisateur</b>	<b>10</b>
<b>Éléments externes utilisés</b>	<b>11</b>
<b>Conclusion</b>	<b>11</b>
Les apports que nous tirons de ce projet	11

# Présentation générale

## ▸ Objectif du sujet

L'objectif de ce module applicatif était de développer une application web légère exécutée sur un serveur web dont l'installation nous était laissée. Cette application devait être capable d'assurer un monitoring sur un parc de machine ainsi que le lancement de processus à distance.

## ▸ Notre Objectif

Ayant déjà une première expérience dans le développement web en technologies PHP et MySQL, et — pour un des membres du binôme — une expérience de stage dans le développement web sur framework *Django* en langage Python, l'objectif que nous nous sommes fixé pour ce projet est le suivant : développer l'application en utilisant le concept des frameworks web afin d'assurer une plus grande qualité au niveau de la maintenabilité de la sécurité et de la qualité de code.

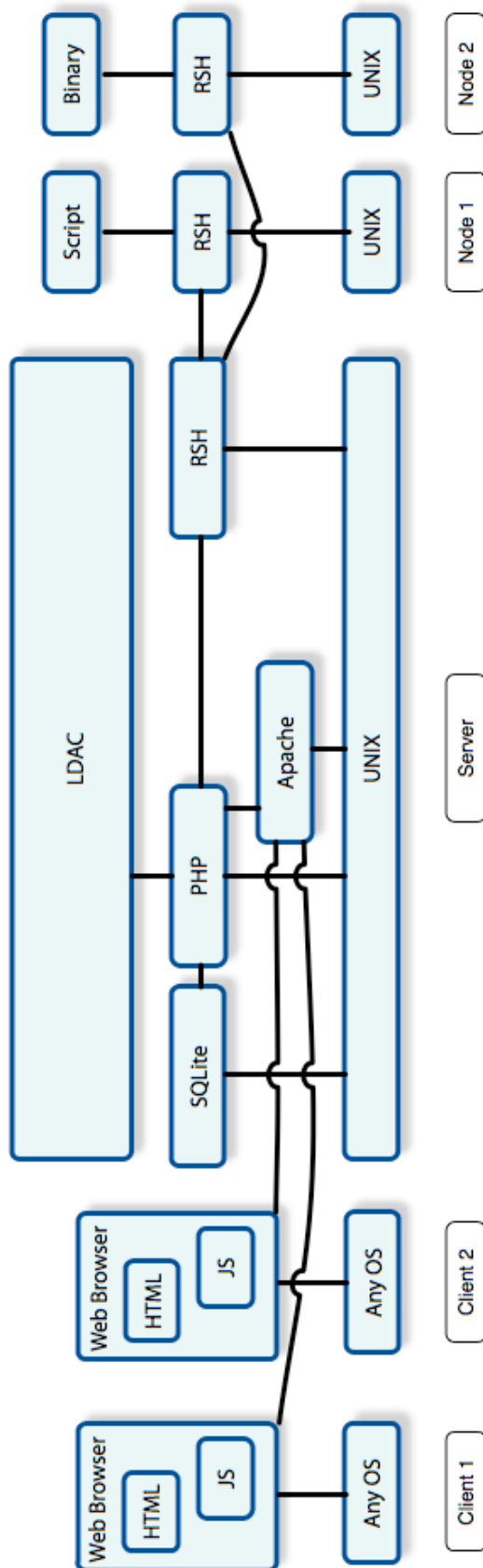
Deux principaux concepts utilisés par les frameworks web sont les suivants:

- Adoption d'une architecture MVC (Modèle-Vue-Contrôleur) afin de mieux isoler les différentes parties de l'application.
- Utilisation d'une couche de mapping-objet (appelée ORM pour Objet-Relational Mapping) permettant une correspondance entre le modèle relationnel de la base de donnée et le modèle objet de l'application.

Notre choix de technologie s'est porté sur l'utilisation du langage PHP dans sa version orientée-objet complétée par SQLite, une base de donnée très légère disponible en standard sur les stations Solaris.

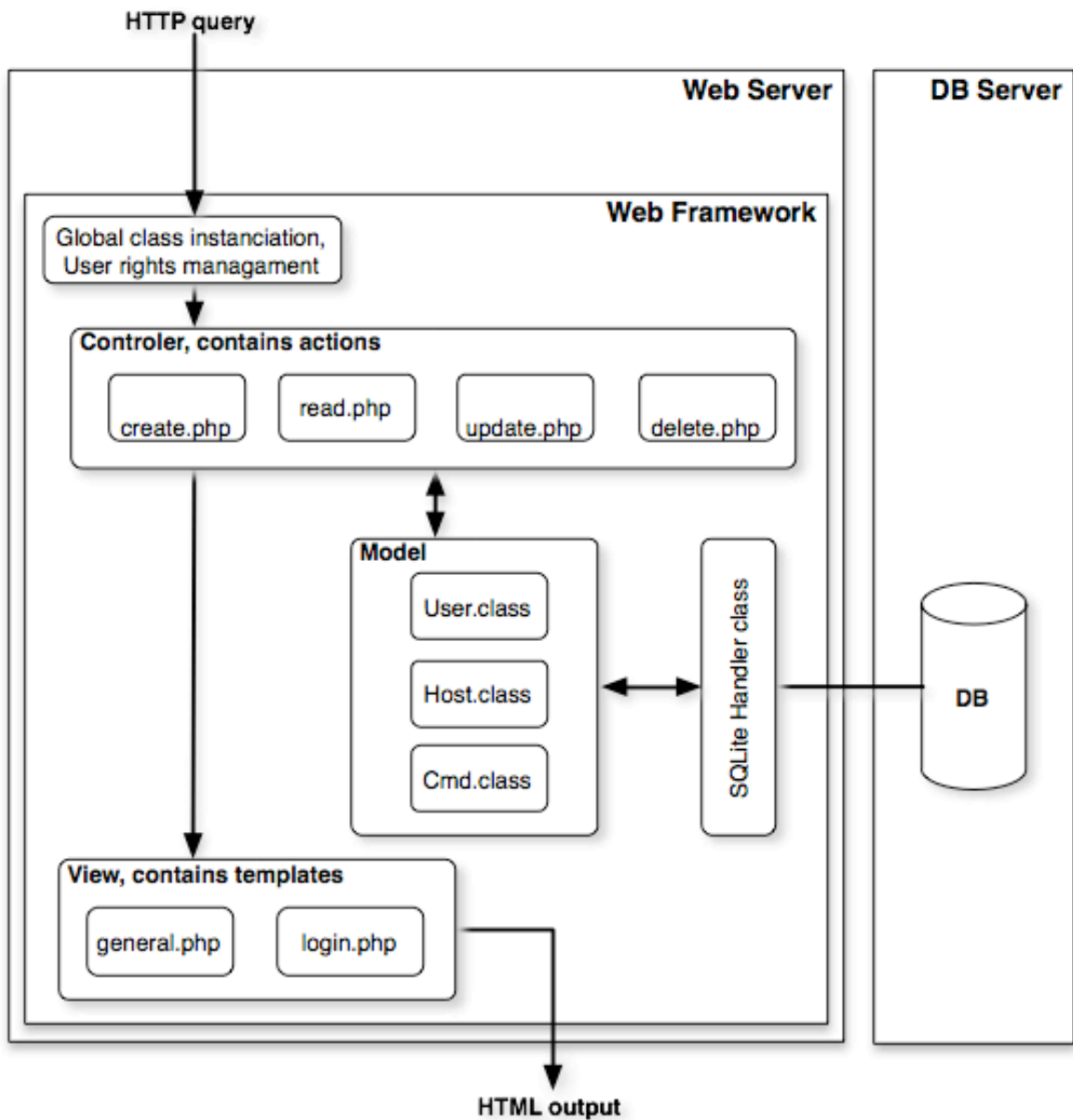
Le cheminement que nous avons suivi était de premièrement développer notre propre petit framework léger pour ensuite s'en servir de base pour y implanter les fonctionnalités désirées pour la gestion de parc Solaris.

# Architecture



## ► Framework

Voici l'architecture du framework que nous avons utilisée.



L'architecture MVC consiste à séparer

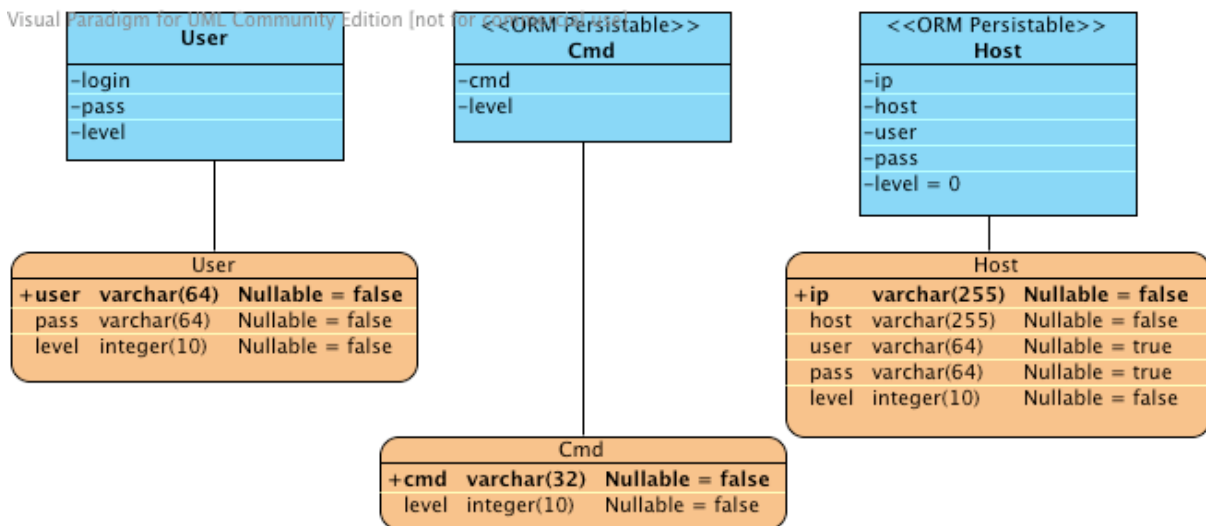
- le **Modèle** : les classes métiers, ici principalement les Utilisateurs (User), les Hosts (Host) et les Commandes (Cmd)
- le **Contrôleur** : ensemble des actions effectuées sur le Modèle tels que l'ajout, l'édition, la modification d'utilisateurs ou bien l'authentification des utilisateurs ou bien encore le monitoring des machines distantes.

- la Vue : ensemble des templates contenant la structure HTML des pages. Le fait de séparer la vue du reste des autres éléments du framework, permet d'avoir une structure HTML unifié pour chacune des pages et permet une évolution généralisé au site entier en cas de modification ce cette vue.

### ► Lien classes - base de donnée : couche ORM

Notre projet utilise une base de données SQLite afin de stocker les utilisateurs, les hôtes ainsi que les commandes autorisées.

Nous avons utilisé pour cela le principe d'ORM (mapping entre l'orientation objet du modèle métier et le schéma relationnel des bases de données). Nous avons ici lié chacune des classes métiers à une table de la base de données.



Pour la modification du modèle nous utilisons le principe de classes persistantes. Ce principe nous permet de modifier les enregistrements des tables de la base de données sans jamais faire de requête SQL dans le contrôleur alors que c'est dans le contrôleur que les classes métiers sont manipulées.

Dès que l'on souhaite modifier un enregistrement dans une table, un objet correspondant à cet enregistrement est instancié avec tous ces paramètres enregistrés. Une fois instancié l'objet est manipulable en toute abstraction de la base. Enfin avant d'être libéré en fin de script, les valeurs des attributs de l'objet sont enregistrées dans la base de donnée.

## Exemples d'utilisation

`/* Voici des exemple de code que l'on pourrait trouver dans le controleur.`

`Il est à noter que l'ORM permet à ce niveau de faire totalement abstraction de la base de donnée sous-jacente.  
*/`

`// Création d'un utilisateur toto de niveau 3 :`

`$toto = new User("toto","mot de passe",3);`

`$toto->save();`

`// Obtenir un objet de type Host d'après son adresse IP :`

`$host = Host::getByIp("192.168.0.1");`

`$host->setLevel(200);`

`$host->save();`

`// Supprimer la commande "rm"`

`$command = Cmd::getByCmd("rm");`

`$command->delete();`

`// Afficher les logins de tout les utilisateurs`

`foreach(User::selectAll() as $user) {`

`print $user->getLogin();`

`}`

# Systeme distribue

Les fonctionnalites "applications et systemes distribues" de ce projet se retrouvent dans les menu Processes et Monitoring.

## ▸ Processes

La page Processes demande deux choses a l'utilisateur : la commande desiree ainsi que les hotes sur lesquels executer celle-ci.

### Selection des hotes

Les hotes sont selectionnes via une liste de case a cocher (« checkbox »). Cette selection des hotes est memorisee d'une commande a l'autre dans le cas ou l'on souhaiterait enchaîner plusieurs commandes sur le meme ensemble d'hotes.

### Commande

Pour des raisons de securite tous les utilisateurs ne peuvent effectuer toutes les commandes. Nous avons donc introduit des niveaux de securite.

Les utilisateurs le niveau 0 correspondent aux utilisateurs "root" : qui ont tous les droits. A partir du niveau 1 et pour toutes les valeurs superieures, les utilisateurs sont soumis aux restrictions de niveaux propres a chaque commande et hote. Plus le niveau est eleve, plus les restrictions sont fortes.

Chaque commande autorisee se voit attribuer un certain de niveau de securite. « **ls** » peut considerée comme une commande avec un niveau de securite assez faible (ie 5 ou superieur) alors que "**rm**" sera une commande critique de niveau 0. Ainsi un utilisateur n'a le droit d'executer une commande que s'il possede un niveau de securite inferieur ou egal a celui de la commande.

Commande et niveau
ps : 2
ls : 5
cd : 4

Utilisateur et niveau
Toto : 3
Bob : 5
God : 0

Exemple :

Bob ne peut executer que "**ls**". Toto lui a le droit d'executer **cd** et **ls**. Tandis que God a le droit de tout executer.

Une interdiction des caractères permettant de passer cette protection (par exemple : « `ls * | rm` ») est imposée. Ainsi toute commande contenant les caractères ( '&', '>', '<', '|', ';' ) est systématiquement rejetée excepté pour les utilisateur “root” de niveau 0.

Le retour obtenu suite à l'application de la commande sur chacun des hosts est affiché en dessous de la boîte de dialogue.

### ► **Monitoring**

La sélection d'hôtes à monitorer s'effectue de la même façon que dans la page Processes.

Ensuite vient la sélection des informations que l'on souhaite obtenir sur les hôtes distants.

Process Monitoring :

- Short list: affiche la liste des 15 premiers process en termes de CPU avec des renseignements.
- Full list: affiche tous les processus actifs sur la machine avec des renseignements.

Network Monitoring :

- Routing table: affiche la table de routage
- Sockets state: affiche l'état des sockets.

System Monitoring :

- State: renvoie la charge de la machine et le nombre d'utilisateurs dessus ainsi que le temps de marche.

En cliquant sur le nom de la machine on arrive sur une page qui récapitule toutes les infos précédentes sur une seule page pour la machine considérée.

### ► **Restriction d'accès au machines distantes**

Le principe d'accès par niveaux est également appliqué selon le niveaux de sécurité des hôtes. Ainsi les hôtes ayant une valeur de niveau inférieur à celui de l'utilisateur (là encore plus la valeur du niveau de sécurité est petit, plus l'accès est restrictif) ne sont pas proposés dans les pages Monitoring et Processes.

# Déploiement

## ▸ Système requis

Afin de pouvoir déployer notre application, il faut disposer de :

- un serveur Apache
- PHP (version 5 du fait de l'orientation objet) compilé avec l'extension SQLite
- des binaires SQLite (version 3) disponibles sur le serveur

## ▸ Installeur

Nous avons déployer un script d'installation qui est appelé automatiquement lors de la première exécution de l'application. Celui-ci crée la base de données et enlève la redirection automatique vers le script d'installation. Ce script d'installation n'est plus utilisable une fois que la base de donnée existe et ne peut donc pas servir malicieusement à écraser cette dernière.

## ▸ Ré-initialisation de la base de donnée.

Dans le but de faciliter le développement de l'application nous avons créer un script de réinitialisation de base de donnée. Ceci-ci effectue un DROP de toutes les tables de la base de donnée puis les reconstruit et ajoute enfin des enregistrements servant d'exemple.

# Intérêts du framework

## ▸ Instantiation de classe communes

Chacun des scripts du contrôleur (qui sont également les seuls scripts appelés par le navigateur au travers des URL) font un **include** avant leur code du script **framework.php**. Ce fichier

- instancie la classe Page (qui est la classe permettant de générer l'assembler les élément HTML selon la template associée ou bien de procéder à une redirection si le développeur le souhaite)
- gère la continuité des sessions PHP

- instancie un objet de type User correspondant à l'utilisateur loggué sur l'application
- spécifie le chemin du dossier contenant les classes du Modèle (obligatoire avec PHP 5 pour l'orientation objet)

### ▸ Library de génération de code

La classe Page propose des méthodes permettant la génération d'élément HTML (très) commun. Par exemple :

```
$page::imgCode("mon_image.png");
```

créé un élément image pointant vers le fichier mon\_image.png en ajoutant le chemin vers le dossier image.

```
$page::addLink("url_de_destination","Intitulé du lien");
```

créé un lien pointant vers **url\_de\_destination** en ajoutant la base de l'URL et avec pour intitulé **"Intitulé du lien"**.

Ces deux méthodes s'inscrivent dans l'orientation "développement rapide" emprunté par les frameworks web standard.

### ▸ ajout dynamique de header

Le chargement de fichiers javascript est permis au niveau du contrôleur. Par exemple :

```
$page->addJs("mon_script_JavaScript");
```

ajoute dynamiquement un script dans l'entête de la page HTML.

## Sécurité

### ▸ Mot de passe

Les mots de passe des utilisateurs de l'applications sont stockés après passage dans l'algorithme de hashage SHA-256. Il est — avec l'état actuel de la technologie — impossible de retrouver le mot de passe original sans utiliser la méthode par "force brute".

## ► Autorisation d'accès aux contrôleurs

L'exécution des contrôleurs est conditionné par une première ligne simple :

```
$page->adminOnly();
```

effectuant une redirection si l'utilisateur n'a pas les droits d'administration ou

```
$page->authOnly();
```

effectuant une redirection si l'utilisateur n'est pas authentifié.

Voici les différentes autorisations associées à chacun des scripts du contrôleur.

Droits d'administration requis	Authentification requise	Pas de restriction
admin	monitoring	login
<b>User :</b> createUser editUser deleteUser	processes	logout
<b>Host :</b> createHost editHost deleted	index	
<b>Cmd :</b> createCmd editCmd deleteCmd	about	
	myPassword	

## Interface utilisateur

L'interface pour un utilisateur lambda est le premier contact avec l'application web. C'est donc un point important à ne pas négliger. Nous avons le choix d'une interface assez épurée sans pour autant négliger l'aspect pratique et esthétique. Pour cela le principe retenu est celui de distinguer le fond (pages HTML générées par PHP) de la forme (mise en style par des feuilles CSS).

L'interface est constituée au niveau visuel de 3 points distincts:

- le header et le footer fixés dans la page par la CSS qui indique le titre de la page courante dans le header et le copyright dans le footer.
- Le menu fixé dans la partie centrale.
- la partie principale appelée « Content » qui affiche les pages proprement dites et qui est défilable.

Une fois mis en place ce squelette, nous avons ajouté des éléments supplémentaires:

- Une bibliothèque d'images qui permet d'agrémenter les pages d'icônes.
- Des scripts AJAX qui permettent d'améliorer l'expérience de l'utilisateur en affichant les messages d'erreur avec un effet « fade » ou encore l'« accordéon » pour l'affichage de retour des commandes qui évite de surcharger la page.
- Les “box redimensionnables”. Nous avons créé (sous Inkscape) les images des menus et de la page principale que nous avons ensuite découpés pour les intégrer dans le site. Ceci permet notamment à la box de la partie « content » d'être redimensionnable verticalement et de garder son style sans aucun problème

## Éléments externes utilisés

Nous avons utilisé une bibliothèque Javascript pour les effets graphiques et l'auto-complétion ainsi qu'une classe standard de l'extension PEAR du projet PHP nous permettant de gérer les sessions (persistance des variables de sessions d'une page à l'autre).

## Conclusion

### ▸ Les apports que nous tirons de ce projet

#### **Compilation et déploiement de serveur Apache et binaires PHP**

La première phase du projet fut de compiler et déployer un serveur Apache ainsi que PHP.

### **Développement d'un léger framework MVC**

En suivant l'objectif que nous nous fixer au début du projet, nous avons développer un framework web léger utilisant le schéma MVC et une couche de mapping-objet.

### **Développement sur framework MVC + ORM**

Ce projet fut l'occasion pour le membre qui n'avait pas fait de développement sur framework web de découvrir cette méthode de développement.

### **Développement et utilisation d'une couche légère de mapping objet**

Ce projet fut l'occasion pour le membre qui a déjà fait du développement sur framework de voir plus en détail le fonctionnement d'un framework MVC et d'une couche de mapping objet.

### **Utilisation de librairies Javascript**

Aucun des deux membres n'avait auparavant utilisé de librairies Javascript permettant d'ajouter des effet de mise en page et de popup de confirmation.

### **Déploiement d'un serveur SVN**

Afin de gérer nos fichiers sources, nous avons déployé un serveur SVN accessible à distance sur un serveur personnel.