

Manuel de développeur

RobHub

L'équipe Robobo

Ce manuel de développeur a pour but d'expliquer la mise en place de l'environnement ainsi que la structure et les choix d'implémentation du projet.

Sommaire

Introduction.....	3
Installation de l'environnement	4
1. WampServer.....	4
2. Git	4
3. Base de données.....	5
4. RobDex, le serveur de compilation des comportements	7
5. Editeur de texte (étape facultative)	9
6. Test	10
Structure de la base de données.....	11
1. La table user	11
2. La table behavior	11
3. La table mark	12
Structure et choix d'implémentation	13
1. Structure générale.....	13
2. Choix d'implémentation	14
2.1 Génération des pages web.....	14
2.2 Les pages d'include.....	15
2.3 Ajouter une page web	15
2.4 Ajouter une page de formulaire	16
2.5 Stockage des comportements et des vidéos.....	16
2.6 API.....	16
Table des illustrations.....	18

Introduction

Le projet **RobHub** est un réseau social permettant le partage de comportements développés avec le framework **RobDev**. Il est possible de créer un compte sur le réseau social afin de mettre en ligne ces comportements et de mettre également en ligne des vidéos démontrant la fonctionnalité du comportement. L'utilisateur est également capable de noter un comportement.


Ce réseau social a été développé en utilisant **HTML 5/CSS 3/JavaScript** pour la partie client et **PHP/MySQL** pour la partie serveur et la base de données. Le design du réseau social est basé sur le template **Bootstrap** [Lesser](#).

Installation de l'environnement

Avant de détailler la structure du projet, nous allons tout d'abord voir comment mettre en place l'environnement de développement. Nous tenons également à préciser que RobHub a été développé sous **Windows** mais qu'il n'y a aucune raison pour que cela ne fonctionne pas sur un autre type de système d'exploitation.

1. WampServer

Pour avoir un serveur local **PHP** sur l'ordinateur, nous avons utilisé le logiciel **WampServer** disponible à cette adresse : <http://www.wampserver.com/>. A l'heure où est écrit ce manuel, c'est la version 3.0.6 qui est utilisée. **WampServer** permet également d'avoir une base de données locale **MySQL**.

Une fois que **WampServer** est installé puis exécuté, vous devriez avoir cette icône  dans la barre de notification.

Remarque 1 :

Nous vous conseillons fortement de consulter [cette page](#) pour installer correctement **WampServer**. Elle vous sera fortement utile en cas de manque de **dll** lors du lancement de **WampServer**.

Remarque 2 :

Vous pouvez laisser le chemin d'installation par défaut **C:\wamp**.

Remarque 3 :

Si l'icône est de couleur orange ou rouge, c'est que le service **WampServer** n'est pas correctement exécuté. Il faut alors chercher d'où provient de problème à l'aide de Google.

2. Git

Le projet **RobHub** étant hébergé sur **GitHub**, nous allons avoir besoin de l'outil **Git** (command line) pour récupérer le projet et "commiter" de nouvelles modifications. Commencez par télécharger **Git**. Pensez à cocher la case de l'ajout de la commande **git** dans la variable d'environnement **PATH**. Une fois que **Git** est installé, ouvrez l'**Invite des commandes** et tapez les commandes suivantes :

1. `cd C:\wamp\www`
2. `git clone https://github.com/RoboboUPMC2016/RobHub.git`

Vous aurez donc maintenant un dossier **C:\wamp\www\RobHub**, le dossier **RobHub** contenant le code source du projet.

3. Base de données

Pour mettre en place la base de données, ouvrez un navigateur et entrez l'adresse suivante : <http://localhost/phpmyadmin/>. Vous serez alors une page d'authentification. Par défaut, l'utilisateur est **root** et le mot de passe est vide.

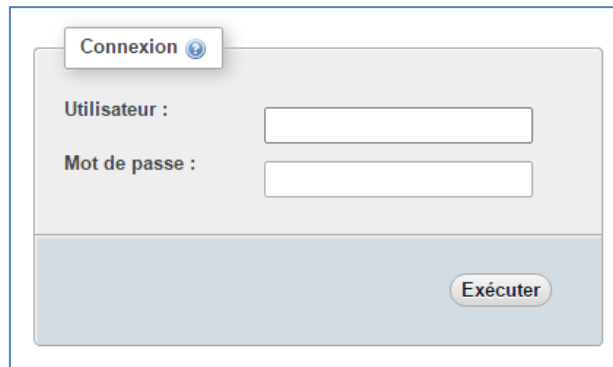
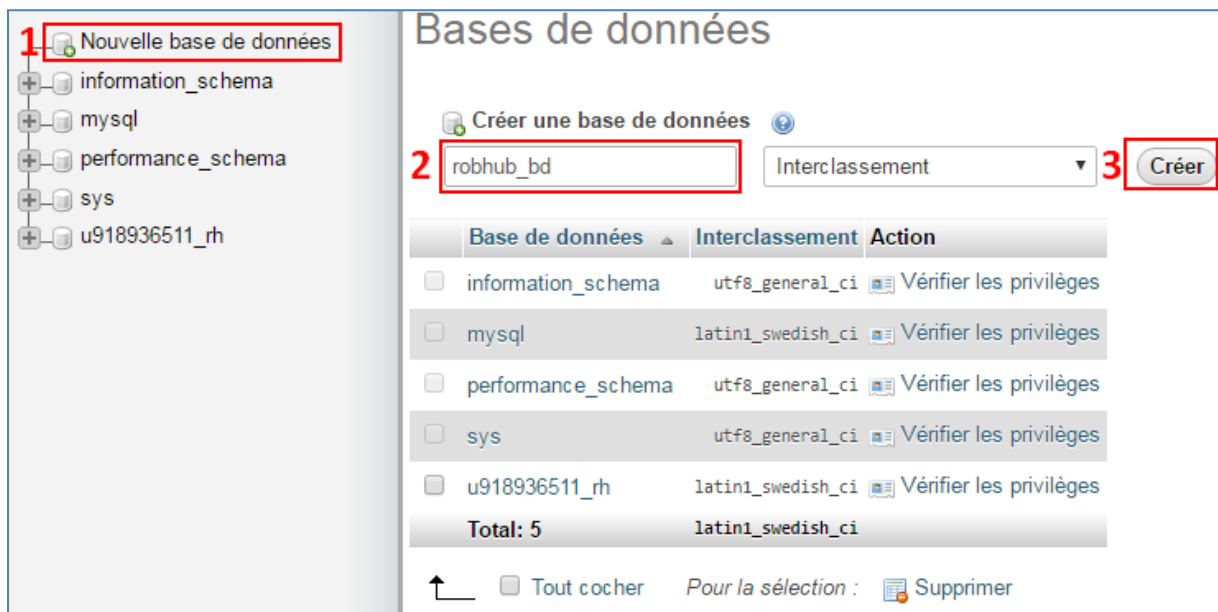


Figure 1 : Page d'authentification phpMyAdmin

Il faut maintenant créer la base de données :

1. Cliquez sur **Nouvelle base de données**.
2. Entrez un nom pour la base de données.
3. Cliquez sur le bouton **Créer**.



Base de données	Interclassement	Action
<input type="checkbox"/> information_schema	utf8_general_ci	Vérifier les privilèges
<input type="checkbox"/> mysql	latin1_swedish_ci	Vérifier les privilèges
<input type="checkbox"/> performance_schema	utf8_general_ci	Vérifier les privilèges
<input type="checkbox"/> sys	utf8_general_ci	Vérifier les privilèges
<input type="checkbox"/> u918936511_rh	latin1_swedish_ci	Vérifier les privilèges
Total: 5	latin1_swedish_ci	

Figure 2 : Création de la base de données

Nous allons ensuite créer les tables à l'aide d'un script **SQL**. Pour cela, vous devez :

1. Cliquez sur **Import**.
2. Sélectionnez le fichier qui se trouve dans le chemin **.wamp_server_path/RobHub/docs/create_robhub_db.sql**.
3. Cliquez sur le bouton **Exécuter**.

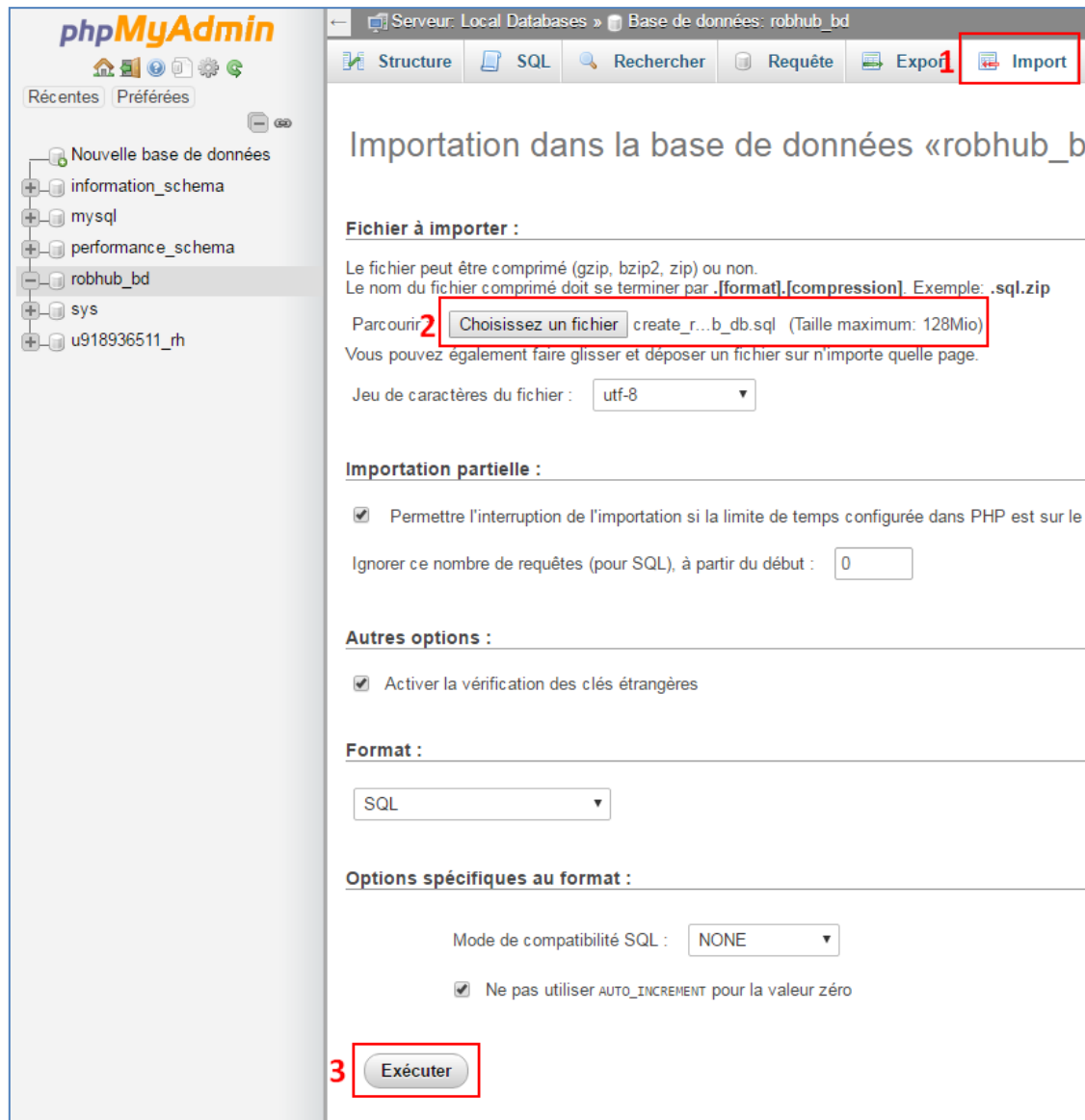


Figure 3 : Création des tables

La base de données avec les tables sont maintenant bien créées. Nous reviendrons plus tard sur la structure des tables de la base de données.

Remarque :

Le script de la base de données ne devrait pas être stocké sur le **GitHub** pour des problèmes de sécurité (tentatives d'attaque sur la base de données). Cependant pour rédiger ce manuel il fallait fournir un lien vers le script **SQL** en question. Une des solutions serait par exemple d'avoir un **repository GitHub** privé (qui est payant).

Dans le dossier **wamp_server_path/RobHub/config/**, créez un fichier nommé **config-database.json** avec la structure suivante :

```
{
    "host": "localhost",
    "dbname": "robhub_bd",
    "user": "root",
    "password": "",
    "dbchar": "utf8"
}
```

Bien sûr, il faut mettre à jour ces informations si la base de données se trouve sur un serveur distant.

4. RobDex, le serveur de compilation des comportements

Les comportements qui sont mis en ligne sont des fichiers **Java**. Ils doivent donc être compilés en fichiers **Dex** pour pouvoir éviter à l'utilisateur de le faire lui même comme cela peut être un peu fastidieux. Une fois ces comportements compilés, les fichiers **Java** et **Dex** sont stockés sur le serveur **RobHub**. Cependant ce n'est pas **RobHub** qui s'occupe de faire la compilation des fichiers **Java** mais un autre serveur nommé **RobDex**.

RobDex a été développé en **Java** et il peut donc être exécuté via **Eclipse**. Après avoir importé **RobDex** dans votre environnement **Eclipse**, il faut rajouter les différents arguments d'exécution de **RobDex** :

- -j RobDev.jar : disponible à [cette adresse](#), correspond au framework **RobDev**.
- -r RetroLambda.jar : disponible à [cette adresse](#), permet d'écrire de compiler des fichiers **Java** contenant des **lambdas expression** (**Android** ne supporte pas encore entièrement **Java 8**).
- -x dx.bat : le chemin de la commande **dx** qui permet de compiler des fichiers **.class** générés par l'outil **javac** en **.dex**.

Pour plus d'informations sur les arguments, n'hésitez pas à consulter la [page GitHub](#) de **RobDex**.

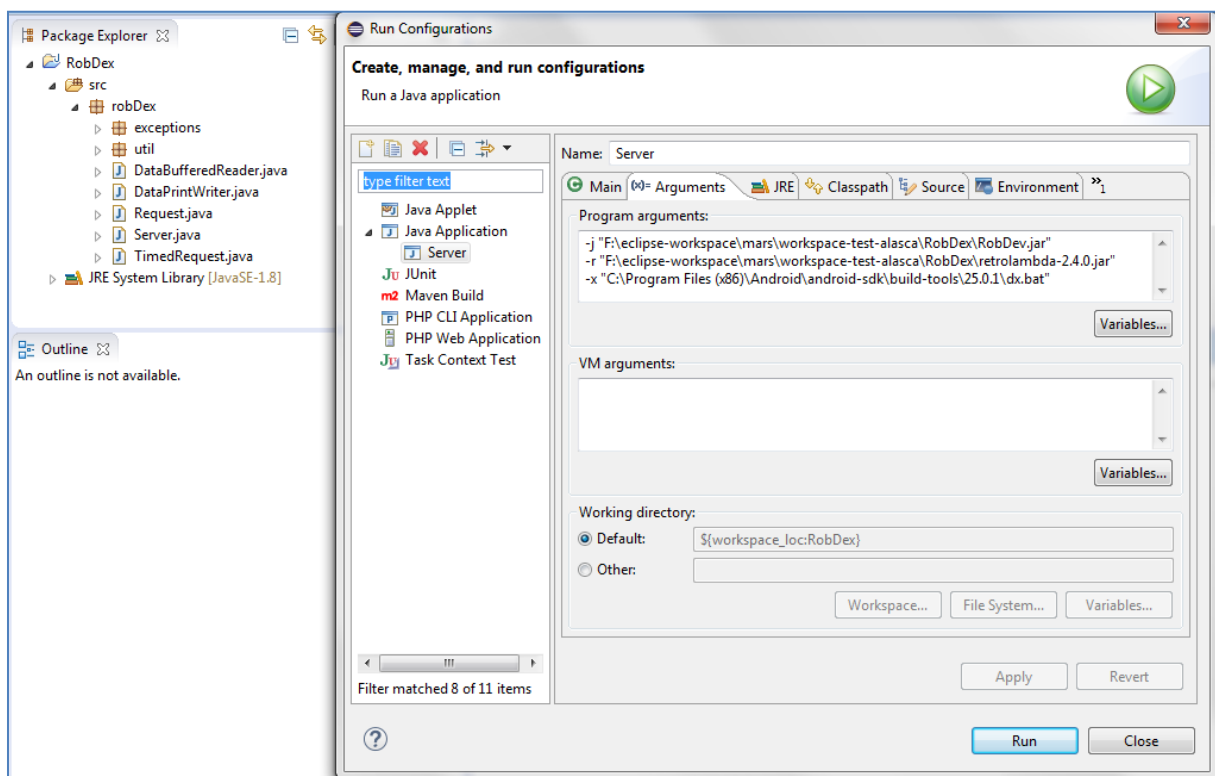


Figure 4 : Configuration de RobDex

Une fois que vous avez lancé **RobDex**, vous devriez avoir le message suivant dans la console : "**Server launched on port 5668. Waiting for connections.**".

Remarque 1 :

Le numéro de port peut être changé via les arguments.

Remarque 2 :

RobDex n'est pas "verbeux", c'est le seul message qu'il affichera tout au long de son exécution (sauf les **exceptions**), il agit comme un serveur.

Tout comme pour la base de données, nous devons définir un fichier **config-socket.json** dans le dossier **wamp_server_path/RobHub/config/** avec la structure suivante :

```
{
    "protocol": "tcp",
    "address": "localhost",
    "port": "5668"
}
```

Dans notre cas **RobHub** et **RobDex** tourne sur la même machine et c'est donc **localhost** qu'il faut mettre dans **address**. Si **RobDex** tourne sur un autre serveur, il faudra certainement mettre à jour les champs **address** et **port**.

Remarque 1 :

Il faut comprendre que **RobHub** et **RobDex** communique via des **sockets**.

Remarque 2 :

Si **RobDex** et **RobHub** ne sont pas connectés, le réseau social pourra toujours fonctionner mais il sera alors impossible de mettre en ligne des comportements.

5. Editeur de texte (étape facultative)

Cette dernière étape consiste à installer un éditeur de texte afin de modifier le code source. Vous pouvez donc utiliser votre éditeur de texte préféré mais il est conseillé d'utiliser [Sublime Text 3](#). Il a pour avantage de pouvoir ouvrir le dossier **RobHub** comme projet et offre une très bonne coloration syntaxique.

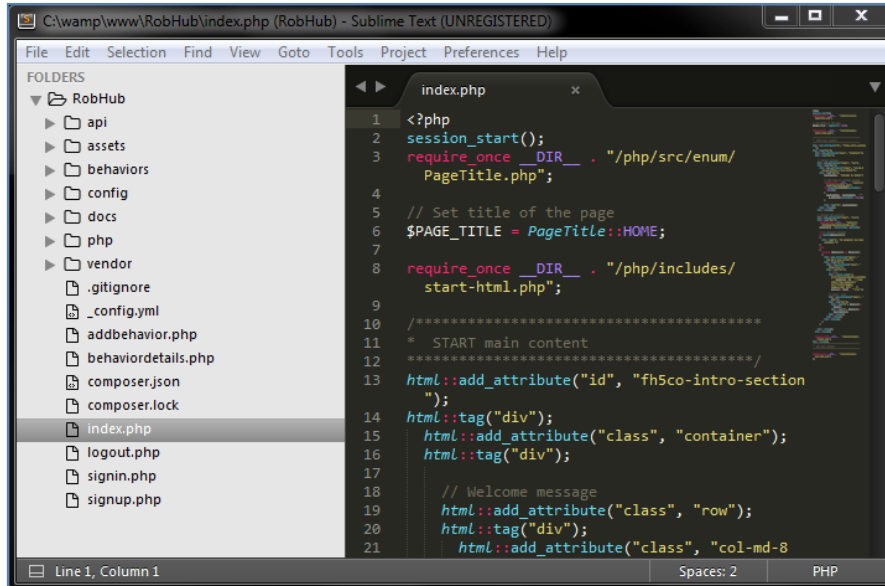


Figure 5 : Projet RobHub dans Sublime Text

6. Test

Vous pouvez dès maintenant vous rendre à l'adresse <http://localhost/robhub/index.php> pour tester si l'environnement est bien mis en place. Vous devriez vous trouver sur la page suivante :

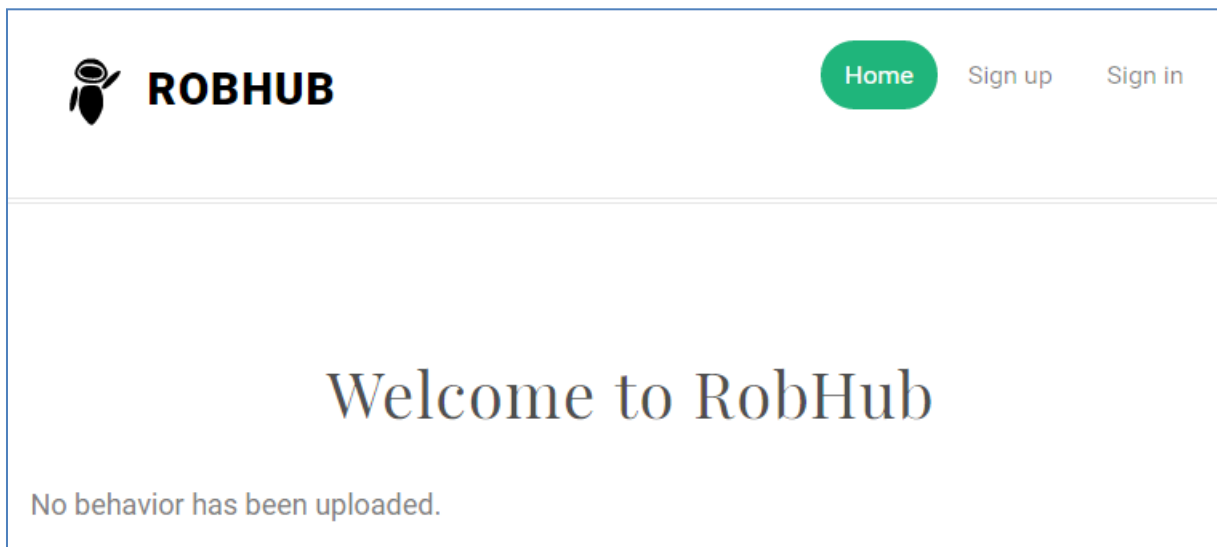


Figure 6 : Page d'accueil de RobHub

Remarque 1 :

Si la page ne charge pas, vérifiez que **WampServer** est bien démarré.

Remarque 2 :

Si des warnings/erreurs **SQL** sont affichés, veuillez vérifier que vous avez correctement créé la base de données avec les tables ainsi que le fichier **wamp_server_path/RobHub/config/config-database.json**.

Structure de la base de données

La base de données est décomposée en trois tables :

1. **user.**
2. **behavior.**
3. **mark.**

1. La table user

Cette table définit un utilisateur du réseau social. Ci-dessous la structure de la table :

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
1	User_username	varchar(20)	utf8_unicode_ci		Non	Aucune		
2	User_password	varchar(40)	utf8_unicode_ci		Non	Aucune		
3	User_firstname	varchar(30)	utf8_unicode_ci		Non	Aucune		
4	User_lastname	varchar(30)	utf8_unicode_ci		Non	Aucune		

Figure 7 : Structure de la table user

Description des champs :

- User_username : identifiant de l'utilisateur.
- User_password : mot de passe de l'utilisateur.
- User_firstname : prénom de l'utilisateur.
- User_lastname : nom de l'utilisateur.

2. La table behavior

Cette table définit un comportement. Ci-dessous la structure de la table :

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
1	Behavior_id	bigint(20)			Non	Aucune		AUTO_INCREMENT
2	Behavior_label	varchar(50)	utf8_unicode_ci		Non	Aucune		
3	Behavior_description	varchar(500)	utf8_unicode_ci		Oui	NULL		
4	User_username	varchar(20)	utf8_unicode_ci		Oui	NULL		
5	Behavior_timestamp	date			Non	Aucune		

Figure 8 : Structure de la table behavior

Description des champs :

- Behavior_id : identifiant du comportant, auto-incrément.
- Behavior_label : label du comportement.
- Behavior_description : description du comportement.
- User_username : identifiant de l'utilisateur qui a mis en ligne le comportement.
- Behavior_timestamp : date de mise en ligne du comportement.

3. La table mark

Cette table définit une note donnée par un utilisateur pour un comportement. Ci-dessous la structure de la table :




#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
1	Mark_id 	int(11)			Non	Aucune		AUTO_INCREMENT
2	Mark_value	int(11)			Non	Aucune		
3	Behavior_id 	bigint(20)			Oui	NULL		
4	User_username 	varchar(20) utf8_unicode_ci			Oui	NULL		

Figure 9 : Structure de la table mark

Description des champs :

- Mark_id : identifiant de la note, auto-incrément.
- Mark_value : valeur de la note.
- Behavior_id : identifiant du comportement qui est noté.
- User_username : identifiant de l'utilisateur qui note le comportement.

Structure et choix d'implémentation

1. Structure générale

Les fichiers **.php** à la racine du dossier RobHub représentent les pages web du réseau social. On peut ici lister les pages web suivantes qui sont les vues :

- `addbehavior.php` : page d'ajout d'un comportement.
- `behaviordetails.php` : page de détails d'un comportement.
- `index.php` : page d'accueil.
- `logout.php` : page de déconnexion.
- `signin.php` : page de connexion.
- `signup.php` : page d'inscription.

Le dossier **api** contient une liste de services permettant de récupérer des informations de la base de données et d'effectuer certaines actions comme noter un comportement.

Le dossier **assets** contient l'ensemble des images et des fichiers CSS.

Le dossier **behaviors** répertorie tous les comportements qui ont été mis ligne ainsi que les vidéos associées.

Le dossier **config** dispose des fichiers de configuration (comme nous l'avons vu précédemment pour les informations sur la connexion à la base de données).

Le dossier **php** contient l'ensemble des classes **PHP** qui font office de contrôleurs ainsi que des fichiers d'**include PHP** afin d'éviter la duplication de code **HTML** (comme le pied de page par exemple ou le menu).

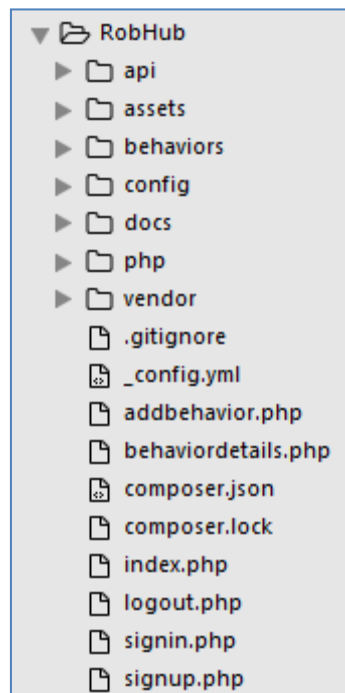



Figure 10 : Arborescence du projet

2. Choix d'implémentation

2.1 Génération des pages web

Bien que l'on puisse mélanger du **PHP** et de l'**HTML**, nous avons décidé d'utiliser une bibliothèque **PHP** nommée **html-writer** qui est en fait une simple classe **PHP** permettant de générer du code **HTML**. Cette classe est vraiment simple d'utilisation, il suffit de consulter [cette page](#) pour comprendre comment l'utiliser. Le code **PHP** n'est donc pas mélangé au code **HTML** et nous avons donc que du **PHP** à lire. Il est donc plus lisible à lire donc plus facilement maintenable.

Voici un simple exemple :



```
<?php
    html::init();

    html::tag('html');
        html::add_attribute("class", "main")
    html::tag('body');
    html::close();
html::close();

print(html::get_buffer());
?>
```

↓ Génération et indentation

```
<html>
  <body class="main">
  </body>
</html>
```

Figure 11 : Exemple d'utilisation de la classe

Remarque :

Les appels `html::init();` et `print(html::get_buffer());` seront automatiquement appelés. Il ne sera donc nécessaire de les appeler quand vous rajouterez de nouvelles pages.

De plus, nous avons également défini une classe **HtmlWriterUtils.php** située dans le dossier **wamp_server_path/RobHub/php/src/util/**. Cette classe contient des fonctions pour générer des structures **HTML** telles qu'un **label** avec son **input** associé. Ces fonctions font gagner énormément de temps car nous sommes souvent amenés à créer des formulaires.

Il existe également des classes **entity** et **DAO** pour chacune de nos trois tables afin de représenter les entrées de la base de données en objet **PHP** (**entity**) et pour faire des communications avec la base de données comme l'ajout, la suppression, la mise à jour, la recherche (**DAO**).

2.2 Les pages d'include

Comme il a été mentionné précédemment, plusieurs éléments du site sont communs sur plusieurs pages. Dans le dossier **wamp_server_path/RobHub/php/includes/** se trouve donc l'ensemble des éléments communs qui sont principalement :

- le header : header.php (qui contient le menu de navigation).
- le footer : footer.php.

Il y a d'autres éléments communs comme le début et la fin de la structure **HTML**. A partir de cela, nous avons donc pu définir une structure par défaut pour toutes les pages qui correspond au fichier **wamp_server_path/RobHub/docs/default_template.php** :

```
1 <?php
2 session_start();
3 require_once __DIR__ . "/php/src/enum/PageTitle.php";
4
5 // Set title of the page
6 $PAGE_TITLE = //PageTitle::XXX; where XXX = the title of the page
7
8 require_once __DIR__ . "/php/includes/start-html.php";
9
10 /*****
11  * START main content
12  *****/
13 html::add_attribute("id", "fh5co-intro-section");
14 html::tag("div");
15     // INSERT WHAT YOU WANT HERE
16
17     require_once __DIR__ . "/php/includes/footer.php";
18 html::close();
19 /*****
20  * END main content
21  *****/
22
23 require_once __DIR__ . "/php/includes/end-html.php";
24 ?>
```

Figure 12 : Structure par défaut d'une page web

2.3 Ajouter une page web

Pour ajouter une nouvelle page web, il faut copier le fichier **wamp_server_path/RobHub/docs/default_template.php** et le coller dans le dossier **wamp_server_path/RobHub/** (ou dans un autre dossier si nécessaire). Il faut ensuite renommer le fichier comme vous le souhaitez, par exemple en **index2.php**. Dans la classe **wamp_server_path/RobHub/php/src/enum/PageTitle.php**, ajoutez une nouvelle variable pour le titre de votre page qui est une sorte d'identifiant :

```
abstract class PageTitle
{
    const TITLE_XXX = "TITLE_XXX";

    // Other titles
}
```

Figure 13 : Ajout d'un titre pour une nouvelle page

Si nous avons besoin de rajouter une section dans le menu de navigation ou afficher un menu avec des sections en plus ou en moins, il faut modifier la page d'**include wamp_server_path/RobHub/php/includes/header.php**. Vous pourrez donner le focus sur la section de votre dans le menu de navigation en comparant les variables **\$PAGE_TITLE** et **PageTitle::XXX**, où **PageTitle::XXX** est le nouveau titre de page que vous venez de définir.

2.4 Ajouter une page de formulaire

Pour ajouter une page de formulaire, il faut en plus rajouter une classe qui va s'occuper de vérifier si les champs du formulaire sont corrects. Il faut rajouter cette classe de contrôle dans le dossier **wamp_server_path/RobHub/php/src/** et la nommer **XXXForm**, où **XXX** est le "nom" du formulaire. Dans cette classe, nous allons définir un ensemble de variables pour garder une référence sur les données du formulaires, nous allons définir un tableau associatif qui va contenir le message d'erreur pour chaque donnée si cette dernière est invalide. Nous pourrions donc ainsi récupérer ces erreurs et les afficher à l'utilisateur. Cette classe doit également contenir les attributs **name** des champs **HTML <input ...>** qui vont nous permettre de définir leur **name** et qui vont également correspondre aux clefs pour les messages d'erreur. Les données soumises via le formulaire doivent être passées via le constructeurs de la classe **XXXForm**. Avant de passer ces données au constructeur, il faut d'abord les "désinfecter" pour éviter toutes attaques de type injection. Pour cela une méthode **clean(\$str)** a été définie dans la classe **StringUtils.php** et il faut donc penser à "désinfecter" toutes les données de type **string**.

Pour comprendre plus en détails le fonctionnement d'un formulaire et de sa classe de validation, nous vous invitons à étudier les fichiers **wamp_server_path/RobHub/signup.php** et **wamp_server_path/RobHub/php/src/form/SignupForm.php** qui vont de pair et qui sont faciles à comprendre si vous souhaitez rajouter un nouveau formulaire. Toute page de formulaire a donc sa vue et son contrôleur.

2.5 Stockage des comportements et des vidéos

Lorsqu'un utilisateur souhaite mettre un comportement en ligne, il met en réalité un fichier **Java** qui doit être compilé en **Dex**. **RobHub** envoie ce fichier **Java** à **RobDex** qui lui nous renvoi le fichier **Dex**. Nous stockons ces deux fichiers dans le dossier **wamp_server_path/RobHub/behaviors/id/**, où **id** est l'identifiant (unique) du comportement qui vient d'être mis en ligne.

Les vidéos sont placées dans le dossier **wamp_server_path/RobHub/behaviors/id/videos/username/**, où **username** est l'identifiant (unique) de l'utilisateur qui a mis en ligne la vidéo. Si une vidéo qui porte le même nom qu'une vidéo existante est mise en ligne, un renommage est effectué afin d'éviter d'écraser le fichier vidéo existant.

2.6 API

Une **API** a été créée afin de permettre d'effectuer plusieurs actions pour le système de notation des comportements. Cela permet de faire des appels **AJAX** au serveur et donc de mettre à jour la note d'un comportement dans la base de données sans recharger la page en question. Nous avons décidé de faire cela avec **AJAX** car c'est ce qui se fait principalement sur les autres applications **Web** lorsque nous notons quelque chose.

Une **API Json getbehaviors.php** a également été créée afin de permettre à l'application **RobApp** de récupérer l'ensemble des informations (label, description, moyenne des notes, **URL** vers les fichiers **Dex**) de tous les comportements afin de pouvoir les télécharger.

Remarque :

Actuellement, **RobHub** est déployé en **LOCAL** et donc son adresse est <http://localhost/robhub/index.php>. Il est donc normal que **RobApp** ne puisse pas accéder à l'**API Json** de **RobHub**. Pour tester que cette **API Json** fonctionne correctement et que **RobApp** puisse y accéder pour télécharger les comportements, nous avons utilisé un service d'hébergement gratuit **Hostinger**. La version gratuite d'**Hostinger** ne permet pas de communiquer avec des ressources extérieures. Il n'était donc pas possible de compiler des comportements en se basant sur la version de **RobHub** hébergée sur **Hostinger** comme la communication avec **RobDex** n'était pas possible. Nous avons donc effectué la compilation des comportements via la version locale de **RobHub** (avec **RobDex** qui est exécuté sur la même machine) puis nous avons créé **MANUELLEMENT** les comportements dans la base de données ainsi que le dossier **/behaviors/id/**. Nous avons ensuite mis les fichiers **Java** et **Dex** dans ces dossiers de comportements. Nous aurions dû avoir un environnement de développement en ligne fourni par un administrateur système notre université mais ne l'avons finalement pas eu.

Table des illustrations

Figure 1 : Page d'authentification phpMyAdmin	5
Figure 2 : Création de la base de données	5
Figure 3 : Création des tables	6
Figure 4 : Configuration de RobDex	7
Figure 5 : Projet RobHub dans Sublime Text	9
Figure 6 : Page d'accueil de RobHub	10
Figure 7 : Structure de la table user	11
Figure 8 : Structure de la table behavior	11
Figure 9 : Structure de la table mark	12
Figure 10 : Arborescence du projet	13
Figure 11 : Exemple d'utilisation de la classe	14
Figure 12 : Structure par défaut d'une page web	15
Figure 13 : Ajout d'un titre pour une nouvelle page	15