POLYTECH LILLE

Université Lille1 Sciences et Technologies

Rapport de création

Modifier et comprendre DataTable en profondeur

```php
/**
 * Transform a string with this format id:value,id:value... in an array
 * Return null if a column of id can't find
 * @param string $index index with this format: id:value,id:value
 * @return array|null
 */
private function parseIndex($index) {
    $error = false;
    $indexCSV = array();
    $allIndex = explode(',', $index);

    foreach ($allIndex as $aCSV) {
        $aIndex = explode(':', $aCSV);
        $name = $aIndex[0];
        $value = $aIndex[1];
        $indexCSV[$name] = $value;

        // Vérification que la colonne existe
        $col = $this->getCol($name);
        if ($col == null)
            $error = true;
    }

    if ($error)
        return null;
    else
        return $indexCSV;
}
```


Edouard COMTET

Génie Informatique et Statistiques - Promo 2014

Février 2014 sous le tutorat de Olivier CARON

Rapport de création

Modifier et comprendre DataTable en profondeur

1. Introduction

Lors de la création d'une partie d'administration, la plupart des développeurs mettent régulièrement en place des tableaux pour la gestion (tableaux des utilisateurs, des news, …).

Le but de ce framework est de permettre, en quelques lignes de code, de générer un tableau HTML/CSS/Javascript relié à une base de données. La modification du tableau sera, de manière automatique, répercutée dans la base.

Ce document est destiné aux développeurs qui devront modifier ou comprendre cet outil. En expliquant le fonctionnement interne de celui-ci et les passerelles entre les différentes parties, ce document permet d'en appréhender la complexité.

Dans la suite de ce document, on considèrera que le framework est installé dans un dossier du nom *DataTable* et que le fichier utilisant ce dernier est à la racine et se nomme *index.php*.

Ce document est rattaché à son homologue qui permet d'identifier les pré-requis nécessaire à l'utilisation de cet outil.

Ce framework se base sur DataTable, disponible à l'adresse http://datatables.net/. Le code source de cette amélioration est disponible depuis https://github.com/doudou34/DataTable.

2. Contexte & objectifs

Ce projet part de deux constats:

- Aujourd'hui, le web 2.0 est devenu incontournable et il a pour objectif de permettre à des utilisateurs n'ayant aucune connaissance technique en informatique d'interagir avec un système de manière très simple et fluide. Ceci se traduit par bien des façons et notamment par l'utilisation du javascript qui permet de naviguer dans un contenu sans devoir rafraîchir ce dernier.
- Lors de la création d'une partie d'administration, le développeur est continuellement obligé de produire des tableaux liés aux données présentes dans une table d'une base de données. Il doit donc implémenter les fonctions de mise à jour du tableau tout en le rendant interactif.

Le but de ce PFE est de réaliser un framework simple d'utilisation permettant d'afficher un tableau relié à une base de données. Ce tableau doit être "2.0".

Le projet se destine aux informaticiens du système d'information de Polytech Lille. L'objectif est d'améliorer le temps de développement des applications de l'équipe responsable dans laquelle je dois m'intégrer.

Au delà de cet objectif, il est aussi intéressant de proposer ce projet à la communauté des développeurs et d'en faire un framework open-source.

Le système d'information de l'école produit aujourd'hui un grand nombre d'applications (disponibles à l'adresse applications.polytech-lille.net) et souhaite aujourd'hui se tourner vers le web 2.0 avec notamment l'arrivée de bootstrap dans le processus de développement.

M. Olivier CARON a aussi remarqué que la majorité de leurs interfaces se présente sous la forme d'un tableau qu'il est nécessaire de modifier et de relier à une base de données (ici Postgresql, mais le framework se veut compatible avec d'autres types de bases comme MySQL).

L'objectif final serait de recréer un PhpMyAdmin en une interface plus malléable et plus "UserFriendly".

3. Généralités

A. Exécution des requêtes SQL

Pour gérer l'exécution des requêtes avec PHP, le framework utilise PDO, une extension de PHP. Pour des raisons de sécurité, l'intégralité des requêtes est d'abord préparée puis exécutée. Le connexion avec la base de données se fait grâce à la fonction *connect* de la classe *Table*, l'exécution des requêtes avec la fonction *prepareExecute* de la même classe qui prendra en paramètre la requête SQL et un tableau avec les paramètres de cette dernière. Cette fonction s'occupera également de vérifier la validité de la connexion à la base de données.

Ainsi, la plupart des fonctions qui créent une requête SQL retourne un tableau associatif comprenant une partie *requete* et une seconde *parameters*.

B. Gestion de la clef primaire

Pour pouvoir identifier le tuple[1] de la base de données à modifier, il est nécessaire que le framework gère proprement la clef primaire d'une table. Pour cela, lors de l'affichage d'une ligne dans le tableau HTML, un attribut index est généré et se présente sous cette forme: index1:value1,index2:value2. La fonction *parseIndex* prend en paramètre une chaîne de caractère et retourne un tableau comportant les index sous la forme:

Clef du tableau	Valeur
index1	value1
index2	value2

C. Modifier le style CSS

Ce framework se base sur le style CSS Bootstrap. Pour modifier le style par défaut, il faut écraser celui de *table.css*. Pour modifier le style CSS Bootstrap, se référer à sa documentation.

[1] Ligne dans la base de données, parfois confondu avec le terme ligne

4. Mécanisme d'affichage d'une table

L'affichage d'une instance *Table* passe par l'appel à la fonction *show* qui prend pour seul paramètre le chemin relatif où est stocké le framework par rapport au fichier d'utilisation. Ceci est utile pour pouvoir afficher des images ou faire appel au fichier *ajax.php* décrit plus bas.

Cette fonction va récupérer le type des champs, afficher le tableau, la fenêtre d'ajout puis sérialiser l'instance de *Table* dans un fichier. Nous ne nous intéresserons, dans cette partie, qu'à la partie affichage du tableau et de la fenêtre.

A. Partie tableau HTML

Pour afficher le tableau HTML, le framework utilise la fonction *showTable* qui s'occupera de fournir la structure du tableau et délèguera l'affichage des tuples de la base de données à *showBody*.
Son rôle premier va être de construire un élément <*table* /> avec une succession d'attributs tel que la langue à utiliser, le chemin relatif, son identifiant unique et les différentes options choisies par l'utilisateur (fonction CRUD, tri, pagination, …).

La fonction *showBody* va récupérer les données et, pour chaque tuple, construire son champ index, sauvegarder sa valeur dans un attribut et générer l'élément <*tr* />.

B. Récupération des données

La fonction showBody va utiliser *getData* et *getDataSQL* pour récupérer les données. Cette dernière fonction comporte 3 variables majeures. Une première qui enregistre la clause SELECT, une seconde pour la clause FROM et la dernière pour la clause WHERE qui prendra en compte les filtres et les relations clefs primaires / clefs étrangères. La construction de la requête est des plus classiques.

C. Partie fenêtre d'ajout

Les fonctions *showModal* et *showInput* permettent de générer une fenêtre qui s'affichera lors du clique sur le bouton d'ajout. Pour cela, le framework utilise

Bootstrap. La fonction *showInput* permet de créer un champ spécifique suivant le type de la colonne.

5. Mécanisme de la partie javascript

A. Les différentes fonctions

Pour fonctionner, l'outil utilise la librairie jQuery. A la création d'un objet avec *$('table[dataTable]').dataTablePlus()*, la fonction *init* sera exécutée et une suite d'actions sera déclenchée (mise en place du CSS bootstrap, liaison des évènements avec ajax, …).

Parmi celle là, on peut noter l'utilisation de la fonction *send*, qui va envoyer grâce à AJAX une requête au fichier *ajax.php*. Les fonctions *setValue* et *getValue* permettent de gérer les types (suivant le type de la colonne dans la base de données, la manière de lui mettre une valeur ou de la récupérer sera différente). La fonction *transformCase* va mettre dans la cellule la bonne valeur (prise en compte notamment des booléens qui s'affichent avec des icônes).

B. Capturer les différents évènements

Lors de son fonctionnement, le framework déclenche plusieurs évènements, voici la liste:

- *fnExtension*: évènement déclenché lorsque l'utilisateur clique sur l'icône d'extension ou qu'il y dépose une ligne. Prend en paramètre la ligne concernée (*<tr />*).
- *fnRemove*: évènement déclenché lorsque l'utilisateur clique sur l'icône de suppression ou qu'il y dépose une ligne. Cet évènement se déclenche uniquement si l'événement a été enregistré dans la base de données. Prend en paramètre l'index concerné.
- *fnAdd*: évènement déclenché lorsque l'utilisateur ajoute une ligne et que cet ajout a été répercuté dans la base de données. Prend en paramètre un tableau avec les nouvelles valeurs.
- *fnUpdate*: évènement déclenché lorsque l'utilisateur modifie une cellule et que cette modification a été répercutée dans la base de données. Prend en paramètre la cellule concernée.

6. Etablissement de la liaison entre le PHP et le javascript

La liaison entre les deux langages, et qui donne son côté web 2.0 au framework, se fait grâce à AJAX. Un fichier *ajax.php* récupère l'ensemble des requêtes et les exécuter au fur et à mesure.

Pour garder le côté sécurisé de l'application, l'instance *Table* est sérialisée et sauvegardée dans le répertoire *temp* grâce à la fonction *serializeTable*. La fonction *checkTempFile* permet de supprimer les fichiers postérieur à une heure (pour ne pas garder un historique trop conséquent). Lorsque que la page est quittée par l'utilisateur, un dernier évènement AJAX est déclenché pour supprimer le fichier de sauvegarde de la sérialisation grâce à la fonction *removeSerializedTable*.

Chaque instance de *Table* est identifiée par un numéro composé de sa place en mémoire et de sa date de création. Pour chaque requête AJAX, la partie javascript transmet l'identifiant, la fonction a exécuté et les paramètres grâce à la fonction *send*.

Les opérations de mise à jour, de suppression et d'ajout se font par l'intermédiaire des 3 fonctions publiques de la classe *Table* du framework: *updateData*, *deleteDate* et *createData*. Les requêtes SQL, plus complexes à générer dans les fonctions *deleteData* et *createData*, ont été externalisées dans *deleteSQL* et *createSQL*.

7. Ajout d'un driver

Pour récupérer le type des champs, le framework s'adapte selon le driver de la base (MySQL, PostGreSQL, …).

Deux exemples de drivers sont à disposition:
- PGSQL qui correspond à une base de données sous PostGreSQL.
- MySQL et Default qui correspondent à une base de données sous MYSQL.

Pour ajouter votre propre driver, la nouvelle classe doit hériter de *TableDriver* et vous devez modifier la fonction *setConcreteDriver*. Les différentes fonctions de la classe permettent de:
- *setTable*: sauvegarde la *table* pour l'exécution des requêtes.
- *getTable*: retourne la *table* actuellement utilisée.
- *getSQL*: retourne la requête SQL à exécuter sous la forme d'un tableau associatif requête / paramètres.
- *getAllColumn*: retourne la requête SQL à exécuter sous la forme d'un tableau associatif requête / paramètres. Cette requête retourne l'intégralité des colonnes.
- *getColumnName*: prend en paramètre une ligne du retour de la requête et retourne le nom de la colonne pour cette ligne (en théorie le nom de la colonne correspondant).
- *getColumnType*: prend en paramètre une ligne du retour de la requête et retourne le type de la colonne (en théorie le nom de la colonne correspondant).
- *getColumnLength*: prend en paramètre une ligne du retour de la requête et retourne la longueur maximale de la colonne(en théorie le nom de la colonne correspondant).
- *isIndex*: prend en paramètre une ligne du retour de la requête et retourne vrai si la colonne est une clef primaire.

8. Ajout d'une colonne

La gestion des colonnes est essentielle dans ce framework. Voici un guide pour ajouter une colonne et répondre à vos attentes.

Tout d'abord, il faut créer votre nouvelle classe qui devra étendre la classe de base: *Col.php*. Il vous faudra aussi ajouter votre fonction *addColXXX* dans la classe *Table* avec les différents paramètres nécessaires.

La classe *Col* définit les fonctions élémentaires nécessaires au bon fonctionnement du framework. Rien ne vous empêche de rajouter des fonctions dans votre classe pour plus de clarté comme dans la classe *ColLinked* qui prend en compte la gestion des filtres. Il sera alors possible d'accéder à votre objet avec la fonction *getCol* de l'objet *Table*.

Voici une description des fonctions utilisées par le framework:

- *setType*: enregistre le type de la colonne (voir la partie Gestion des types). Si le type est linked, la fonction *getSelectOption* sera utilisée.
- *getHeadLabel*: retourne le libellé de la colonne (afficher dans le tableau HTML).
- *getHeadName*: retourne le nom de la colonne dans la base de données.
- *getBody*: retourne la valeur pour les données fournies (généralement retourne l'index portant le nom de la colonne dans les données fournies).
- *getIndex*: retourne une chaîne de caractère contenant la valeur de l'index pour ce champ (vide si ce champ n'est pas une clef primaire). Le paramètre *force* permet de forcer cette fonction à retourner un index, même si elle n'est pas un index (utile dans les colonnes de type *ColLinked*).
- *getTables*: retourne un tableau de chaîne de caractère contenant les tables utilisées pour ce champ (utilisée pour la clause WHERE).
- *getField*: retourne la ou les colonnes utilisées pour l'affichage dans la base de données.
- *getLinks*: retourne une chaîne de caractère contenant les différentes conditions à mettre dans la clause WHERE lors de la récupération des données.
- *getSelectOption*: retourne la requête SQL permettant de récupérer les différentes options à afficher lors de la modification ou de l'ajout de ce champ (utilisée uniquement si le champ a pour type *linked*).
- *isVisible*: retourne vrai si la colonne est visible par l'utilisateur.

- *isCreatable*: retourne vrai si, lors de l'ajout d'une ligne, l'utilisateur doit renseigner ce champ.
- *getValue*: retourne la requête SQL permettant de récupérer pour ce champ et pour l'*index* fourni la valeur enregistrée dans la base de données.
- *update*: retourne la requête SQL pour mettre à jour ce champ à l'*index* fournit avec la valeur *new*.
- *create*: retourne la valeur de ce champ lors de la création avec *value* la valeur saisie par l'utilisateur (retourne vide pour ne pas prendre ce champ en compte lors de la requête INSERT INTO).

9. Gestion des types

Les différents types de champs (*text*, *varchar*, *date*, …) sont gérés à plusieurs endroits du framework. Si vous voulez rajouter la gestion d'un type, voici la liste des endroits à prendre en compte:

- Dans le fichier *Table.php*, dans la fonction *showInput*, vous devez rajouter l'affichage pour votre type.
- Dans le fichier *table.js*, dans les fonctions *setValue* et *getValue*, vous devez rajouter votre type pour que le framework puisse récupérer sa valeur et la mettre à jour. Les fonctions prennent en paramètre l'élément jQuery où il faut récupérer ou mettre à jour la valeur, le type du champ et, pour la fonction *setValue*, la nouvelle valeur.

10. Apports & Conclusion

Au final, ce projet m'aura permis comme je l'espérais, de découvrir les différentes facettes de jQuery. Je peux aujourd'hui dire que j'ai une expérience de ce langage en en maitrisant les aspects basiques.

J'ai aussi pu découvrir les outils de gestion de version (SubVersion et Git) qui permettent d'améliorer le travail en équipe d'une manière assez simple.

La réalisation de ce projet en open-source m'a sensibilisé aux différents besoins des développeurs, aux aspects de sécurité et à la lisibilité d'un code. Les commentaires pour le rendre réutilisable ne sont pas toujours une chose simple, la patience et la clarté sont les clefs de la réussite.

J'ai aussi pu m'apercevoir que je n'ai pas du tout respecté le planning de base. Tout au long du projet, j'ai eu 2 semaines d'avance grâce notamment aux vacances de Noel et une bonne gestion de mon temps.

Je tiens à remercier tout particulièrement Juliette MAGNIES et Olivier CARON pour leurs tests et leur suivi. Je porte également une attention toute particulière à Boris OBREMSKI et Valentin WUILBEAUX, mes deux bêta testeurs qui ont su prendre leur mal en patience pour m'aider à corriger mes erreurs autant dans le code que dans la documentation.

Finalement ce projet est une belle réussite. Malgré quelques améliorations à apporter et des bugs à corriger, son déploiement dans un enivrement de production, réalisé par Juliette MAGNIES sur l'application de mobilité, montre qu'il est puissant et permet de créer une application en quelques lignes de code. Le développeur doit maintenant mettre le focus sur son schéma relationnel et non sur les aspects techniques de son code.

11. Bibliographie & Annexes

PHP: http://www.php.net/manual/fr/

PHP est un langage informatique utilisé dans ce framework. Certains aspects complexes, comme la notion d'objet, sont utilisés dans celui-ci.

Bootstrap: http://getbootstrap.com/

Bootstrap est un framework qui permet de créer facilement et simplement des pages HTML en définissant des classes CSS. Grâce à sa partie jQuery, il permet également d'afficher des fenêtres, des tooltips, …

DataTable: http://datatables.net/

DataTable est un framework qui permet, grâce à jQuery, de formater un tableau HTML avec une pagination, un champ de recherche, …

PDO: http://php.net/manual/fr/book.pdo.php

PDO est une librairie de PHP qui permet de gérer des connexions avec une base de données.

Jquery: http://jquery.com/

Jquery est une librairie javascript qui permet d'étendre les fonctions du langage.

GitHub: https://github.com/

GitHub est un service internet qui permet de proposer à la communauté un espace de stockage pour un projet informatique. Il permet aussi d'améliorer le travail en équipe au même titre que SubVersion.