

SPECIFICATION

DETAILLEE

Alexis LEPAGE

Table des matières

[Rappel 3](#_Toc451415291)

[a) Le contexte 3](#_Toc451415292)

[b) Réalisation du projet 3](#_Toc451415293)

[c) Contraintes 3](#_Toc451415294)

[Architecture détaillée 4](#_Toc451415295)

[a) Environnement de développement. 4](#_Toc451415296)

[b) Schéma Détaillée du projet 7](#_Toc451415297)

[Modélisation de l’application 8](#_Toc451415298)

[a) Diagramme de classe 8](#_Toc451415299)

[Complétude de la définition du dictionnaire des données. 9](#_Toc451415300)

[a) Dictionnaire des données 9](#_Toc451415301)

[b) Dictionnaire des données – Application 12](#_Toc451415302)

[c) Définition du dictionnaire des données 14](#_Toc451415303)

[Maquettage du BackOffice 17](#_Toc451415304)

[a) Maquette Home 17](#_Toc451415305)

[b) MAQUETTE Ajout qCM 19](#_Toc451415306)

[c) Maquette Liste Qcm 21](#_Toc451415307)

[d) Maquette Afficher Qcm 23](#_Toc451415308)

[e) Maquette mise à jour Qcm 24](#_Toc451415309)

[f) Maquette de suppression d’un Qcm 25](#_Toc451415310)

[g) Note 25](#_Toc451415311)

[Maquettage Application mobile 26](#_Toc451415312)

[a) Maquette de connexion 26](#_Toc451415313)

[b) Maquette Liste catégorie. 29](#_Toc451415314)

[c) Maquette liste Qcm. 31](#_Toc451415315)

[d) Maquette questions 33](#_Toc451415316)

[Choix de la méthode de développement 35](#_Toc451415317)

[a) Test Unitaire 35](#_Toc451415318)

[b) Versionning 36](#_Toc451415319)

[c) Convention de nommage 36](#_Toc451415320)

[d) Commentaires 36](#_Toc451415321)

[e) Le choix symfony 2 37](#_Toc451415322)

[f) Les design pattern présent dans le projet 37](#_Toc451415323)

# Rappel

### Le contexte

La société TACTFactory souhaite remplacer les QCM papier traditionnel par une solution technologique qui permettrait un gain de temps considérable pour les formateurs. Ce projet est destiné à tous les élèves suivants des formations au sein des établissements concernés. Celui-ci sera une application mobile disponible sur 3 plateformes : Android, IOS et Windows Phone et permettra un gain de temps considérable pour les formateurs. Les QCM seront accessibles grâce à une authentification et disposeront de diverses questions à choix multiple ainsi que du contenu multimédia. La disponibilité d’un QCM et sa durée sera limité et il appartiendra à une catégorie. Enfin, les résultats seront mis à la disposition du formateur par mail.

### Réalisation du projet

La réalisation de ce projet correspond à :

* 3 applications mobiles (Android, IOS et Windows Phone) : Ces applications devront offrir différentes fonctions à l’utilisateur, une authentification, l’affichage des différentes catégories où des QCM sont disponibles, les QCM disponibles, l’affichage de toutes les questions, les réponses et leurs médias associés.
* Un web Service : Le web service aura pour fonction principal de recevoir les QCM des élèves, de les calculer et de retourner le résultat au formateur associé.
* Un Backend : Le Backend sera accessible à tous les administrateurs et permettra la gestion des QCM, de leurs questions, de leurs réponses ainsi que des médias qui lui sont liés. De plus, la gestion des groupes et des utilisateurs seront possible.

### Contraintes

Le client TACTFactory souhaite que son application soit disponible sur 3 plateformes : Android, IOS et Windows phone. La version d’Android devra être au minimum de 4.1 et celle d’IOS de 8.0. Il souhaite également que le web service soit fait avec Symfony 2. Enfin le Backend sera fait avec le bundle Sonata Admin.

La consultation des résultats se fera sur le Backend et un mail sera envoyé à chaque fin de correction d’un QCM à l’administrateur qui correspond.

Le thème couleur de l’application est aux couleurs de la société TACTFactory.

# Architecture détaillée

### Environnement de développement.

WebService et BackOffice

Le Web service ainsi que le BackOffice ont été réalisé à l’aide du Framework Symfony2. Ce Framework fonctionne avec un serveur Apache et une base de données MySQL inclut directement au sein de Wamp. Il a été développé sur Sublime Text 3.

Au sein de ce Framework, nous utilisons différents Bundle qui permettent d’ajouter des fonctionnalités diverses au projet de base. Voici la liste des Bundles utilisés et leurs fonctions :

|  |  |
| --- | --- |
| Nom du Bundle | Fonction |
| TwigBundle | Simplification de la mise en place du code PHP au sein du code HTML ainsi que la syntaxe à utiliser. |
| DoctrineBundle | Mise en place d’un ORM permettant la création d’un ensemble de données au sein du projet. |
| SonataAdminBundle | Bundle permettant la mise en place d’un BackOffice qui gère les données au sein du projet. |
| FOSRestBundle | Permet la création et l’affichage de page JSON lié aux données du projet. |
| FOSUserBundle | Permet de sécuriser son projet avec l’ajout de fonctionnalité (Chiffrement du mot de passe utilisateur, connexion sécurisé). |
| JMSSerializerBundle | Permet la gestion des données présentent au sein des pages JSON. |

Afin de gérer au mieux la totalité des Bundles nous avons mis en place un outil qui s’appelle Composer. Celui-ci répertorie la totalité des Bundles présents au sein du projet et permet la mise à jour de ceux-ci et de leur dépendance.

Le rôle du WebService est de fournir aux différentes applications un flux JSON qui contient les données présentent au sien de la base de données du serveur en question. Celui-ci reçoit également des flux JSON en provenance des applications contenant les réponses d’un élève concernant un Qcm. Voici un exemple de flux JSON envoyé par le WebService aux applications :



Application Android

L’application Android est optimisé pour les versions 4.1 et supérieur à celle-ci. Elle utilise une base de données SQLite au sein du téléphone qui est réduite un peu par rapport aux schémas de base de données du WebService. Elle est réalisée à l’aide de l’outil Android studio, qui est l’outil de référence pour les développements Android ainsi que l’outil d’émulation associé à celui-ci. Le développement Android respecte les conventions de nommage du langage Java.

Application IOS

L’application IOS est utilisable pour les versions 8.0 et supérieur. Le logiciel de développement utilisé est XCode présent uniquement sur MacOs. L’application utilise le langage Objective C et respect les conventions de nommage associé à ce langage.

Application Windows Phone

L’application Windows Phone utilise la version 8.1. Le logiciel de développement utilisé est Visual Studio en version 2015 Enterprise ainsi que le plugin d’émulation associé à celui-ci. Le développement Windows Phone respecte les conventions de nommage du langage C#.

### Schéma Détaillée du projet

**JSon** - Bundle APIRest

**WebService**

**BackOffice** - Bundle Sonata

Serveur apache

Base de données MySQL

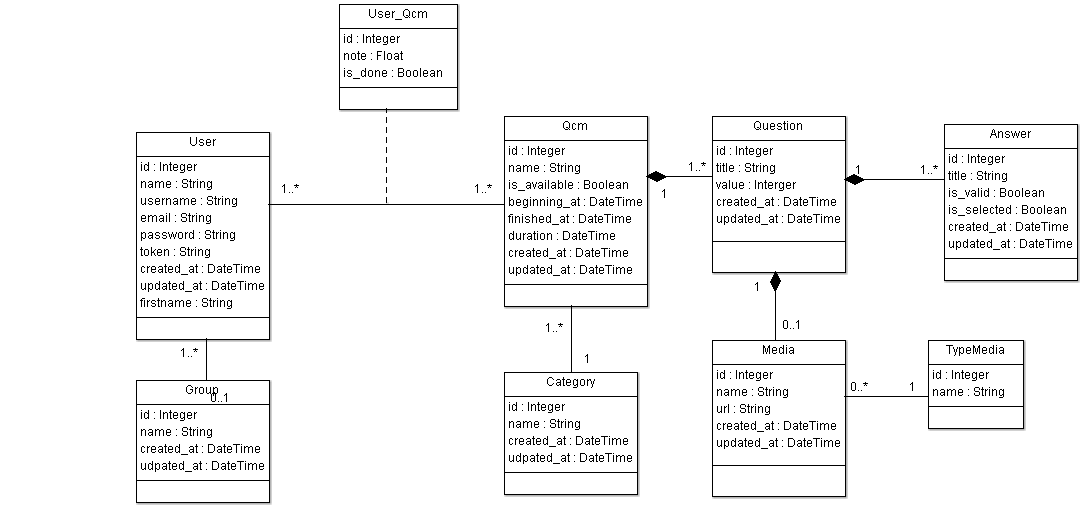
Applications

Au sein du projet nous utilisons actuellement une base de données MySQL avec un serveur apache en local. Nous avons choisi d’utiliser le Framework Symfony2 comme outil de développement pour le WebService et le BackOffice. Celui-ci gère les données présentent au sein de la base de données, il permet à l’utilisateur de créer, modifier, afficher ou supprimer la totalité des données souhaitées dans la configuration.

Le Web Service permet également la génération de page JSON grâce au Bundle APIRestBundle ainsi que la récupération de JSON envoyé depuis les applications afin de calculer le résultat des utilisateurs. Quant à elles, les applications viennent récupérer les dits JSON afin de synchroniser les données de l’application avec celles du Web Service.

# Modélisation de l’application

### Diagramme de classe



Ce diagramme de classe représente l’intégralité de la base de données présente sur Webservice.

# Complétude de la définition du dictionnaire des données.

### Dictionnaire des données

Table Fos\_user

|  |  |  |  |
| --- | --- | --- | --- |
| Colonne | Type | Null | Commentaires |
| Id | int(11) | Non | Identifiant de l’utilisateur |
| user\_type\_id | int(11) | Oui | Identifiant du type d’utilisateur |
| user\_group\_id | int(11) | Oui | Identifiant du groupe utilisateur |
| username | varchar(255) | Non | Username de l’utilisateur |
| email | varchar(255) | Non | Email de l’utilisateur |
| password | varchar(255) | Non | Mot de passe de l’utilisateur |
| firstname | varchar(255) | Oui | Prénom de l’utilisateur |
| created\_at | datetime | Non | Date de création de l’utilisateur |
| updated\_at | datetime | Non | Date de modification de l’utilisateur |
| name | varchar(255) | Oui | Nom de l’utilisateur |

Table GroupUser

|  |  |  |  |
| --- | --- | --- | --- |
| Colonne | Type | Null | Commentaires |
| id | int(11) | Non | Identifiant du groupe d’utilisateur |
| name | varchar(255) | Non | Nom du groupe d’utilisateur |
| created\_at | datetime | Non | Date de création du groupe d’utilisateur |
| updated\_at | datetime | Non | Date de la modification du groupe d’utilisateur |

Table User\_Qcm

|  |  |  |  |
| --- | --- | --- | --- |
| Colonne | Type | Null | Commentaires |
| id | int(11) | Non | Identifiant du Qcm\_User |
| qcm\_id | int(11) | Non | Identifiant du QCM |
| user\_id | int(11) | Non | Identifiant de l’utilisateur |
| is\_done | tinyint(1) | Non | L’utilisateur a répondu au qcm ou non |
| note | double | Oui | Note du Qcm\_User |

Table Qcm

|  |  |  |  |
| --- | --- | --- | --- |
| Colonne | Type | Null | Commentaires |
| id | int(11) | Non | Identifiant du QCM |
| category\_id | int(11) | Non | Identifiant de la catégorie |
| name | varchar(255) | Non | Nom du QCM |
| Is\_available | Tinyint(1) | Non | Disponibilité du QCM |
| beginning\_at | datetime | Non | Date de début du QCM |
| Finished\_at | datetime | Non | Date de fin du QCM |
| duration | int(11) | Non | Durée du QCM |
| created\_at | datetime | Non | Date de création du QCM |
| updated\_at | datetime | Non | Date de modification du QCM |

Table Category

|  |  |  |  |
| --- | --- | --- | --- |
| Colonne | Type | Null | Commentaires |
| id | int(11) | Non | Identifiant de la catégorie |
| name | varchar(255) | Non | Nom de la catégorie |
| created\_at | datetime | Non | Date de création de la catégorie |
| updated\_at | datetime | Non | Date de modification de la catégorie |

Table Question

|  |  |  |  |
| --- | --- | --- | --- |
| Colonne | Type | Null | Commentaires |
| id | int(11) | Non | Identifiant de la question |
| qcm\_id | int(11) | Non | Identifiant du QCM |
| title | longtext | Non | intitulé de la question |
| value | Int(11) | Non | Valeur en point de la question |
| created\_at | datetime | Non | Date de création de la question |
| updated\_at | datetime | Non | Date de la modification de la question |

Table answer

|  |  |  |  |
| --- | --- | --- | --- |
| Colonne | Type | Null | Commentaires |
| id | int(11) | Non | Identifiant de la réponse |
| question\_id | int(11) | Non | Identifiant de la question |
| title | varchar(255) | Non | Contenu de la réponse |
| is\_valid | tinyint(1) | Non | Réponse valide ou non |
| created\_at | datetime | Non | Date de création de la réponse |
| updated\_at | datetime | Non | Date de la modification de la réponse |

Table Media

|  |  |  |  |
| --- | --- | --- | --- |
| Colonne | Type | Null | Commentaires |
| id | int(11) | Non | Identifiant du média |
| question\_id | int(11) | Non | Identifiant de la question |
| Type\_media\_id | int(11) | Non | Identifiant du type de média |
| name | varchar(255) | Non | Nom du média |
| created\_at | datetime | Non | Date de création du média |
| updated\_at | datetime | Non | Date de la modification du média |

Table TypeMedia

|  |  |  |  |
| --- | --- | --- | --- |
| Colonne | Type | Null | Commentaires |
| id | int(11) | Non | Identifiant du type du média |
| name | varchar(255) | Non | Nom du type média |

### Dictionnaire des données – Application

Au sein des applications les entités utilisées ne sont pas les mêmes que sur le Webservice, elles n’ont pas besoin de la totalité des informations de celui-ci pour fonctionner. En effet, les tables *User*, *Group* et *User\_Qcm* ne sont pas présente sur les applications dût au fait que nous récupérons seulement à partir des entités *Qcm* et *Category*.

Table Qcm

|  |  |  |  |
| --- | --- | --- | --- |
| Colonne | Type | Null | Commentaires |
| id | int(11) | Non | Identifiant du QCM |
| category\_id | int(11) | Non | Identifiant de la catégorie |
| name | varchar(255) | Non | Nom du QCM |
| Is\_available | Tinyint(1) | Non | Disponibilité du QCM |
| beginning\_at | datetime | Non | Date de début du QCM |
| Finished\_at | datetime | Non | Date de fin du QCM |
| duration | int(11) | Non | Durée du QCM |
| created\_at | datetime | Non | Date de création du QCM |
| updated\_at | datetime | Non | Date de modification du QCM |

Table Category

|  |  |  |  |
| --- | --- | --- | --- |
| Colonne | Type | Null | Commentaires |
| id | int(11) | Non | Identifiant de la catégorie |
| name | varchar(255) | Non | Nom de la catégorie |
| created\_at | datetime | Non | Date de création de la catégorie |
| updated\_at | datetime | Non | Date de modification de la catégorie |

Table TypeMedia

|  |  |  |  |
| --- | --- | --- | --- |
| Colonne | Type | Null | Commentaires |
| id | int(11) | Non | Identifiant du type du média |
| name | varchar(255) | Non | Nom du type média |

Table Question

|  |  |  |  |
| --- | --- | --- | --- |
| Colonne | Type | Null | Commentaires |
| id | int(11) | Non | Identifiant de la question |
| qcm\_id | int(11) | Non | Identifiant du QCM |
| title | longtext | Non | intitulé de la question |
| value | Int(11) | Non | Valeur en point de la question |
| created\_at | datetime | Non | Date de création de la question |
| updated\_at | datetime | Non | Date de la modification de la question |

Table Answer

|  |  |  |  |
| --- | --- | --- | --- |
| Colonne | Type | Null | Commentaires |
| id | int(11) | Non | Identifiant de la réponse |
| question\_id | int(11) | Non | Identifiant de la question |
| title | varchar(255) | Non | Contenu de la réponse |
| is\_valid | tinyint(1) | Non | Réponse valide ou non |
| is\_selected | tinyint(1) | Non | Réponse sélectionné ou non par l’utilisateur. |
| created\_at | datetime | Non | Date de création de la réponse |
| updated\_at | datetime | Non | Date de la modification de la réponse |

Table Media

|  |  |  |  |
| --- | --- | --- | --- |
| Colonne | Type | Null | Commentaires |
| id | int(11) | Non | Identifiant du média |
| question\_id | int(11) | Non | Identifiant de la question |
| type\_media\_id | int(11) | Non | Identifiant du type de média |
| name | varchar(255) | Non | Nom du média |
| url | Varchar(255) | Non | Url du média |
| created\_at | datetime | Non | Date de création du média |
| updated\_at | datetime | Non | Date de la modification du média |

### Définition du dictionnaire des données

Note : *la définition du dictionnaire des données ne prendra pas en compte le champ id qui est une propriété unique de chaque objet au sein de la totalité des tables.*

Table User :

La table User contient un Name et un Firstname qui correspondent au nom et prénom de l’utilisateur. Cependant c’est le champ Username qui permet à l’utilisateur de se connecter sur les applications. L’utilisateur devra également renseigner un Password qui sera ensuite chiffré par le Bundle FosUserBundle. Au sein des applications nous avons voulu inclure l’adresse email dans les propriétés de la table User afin que celui-ci puisse changer son mot de passe ou recevoir des emails concernant les Qcm ainsi que les notes de ceux déjà effectué.

Un champ Token a été créé dans le but de pouvoir ajouter une fonctionnalité de notification push aux applications et ainsi enregistrer le Token du téléphone pour qu’il puisse recevoir ces notifications.

Enfin les champs Created\_at et Updated\_at correspondent à la date de création de l’utilisateur et à la dernière date de modification de cet utilisateur. Cela permet d’avoir un suivit de l’utilisateur en question au sein de la base de données.

Table Group :

Cette table a été créée afin de distinguer les utilisateurs et les administrateurs au sein du projet. En effet seul les administrateurs peuvent avoir accès au BackOffice qui permet la gestion de toutes les applications. Au sein de cette table nous trouvons un champ Name qui correspond au nom du groupe en question.

De plus les champs Created\_at et Updated\_at correspondent à la date de création du groupe et à la dernière date de modification de ce groupe. Cela permet d’avoir un suivit du groupe en question au sein de la base de données.

Enfin, cette table peut accueillir dans de prochaines améliorations, de nouveaux groupes afin de séparer encore plus les droits des applications.

tABLE uSER\_qCM :

Cette table est créée comme liaison entre la table User et Qcm. Elle permet notamment à l’administrateur de renseigner quel utilisateur aura accès à quel Qcm au sein de BackOffice. Au sein de cette table nous avons une propriété note qui corresponds à la note d’un utilisateur pour un Qcm ainsi qu’une propriété is\_done pour savoir si le Qcm a été effectué.

Table Qcm :

La table Qcm contient un champ Name qui correspond au nom visible sur les applications de celui-ci. Il possède également un champ is\_available afin de savoir si le Qcm est disponible ou non sur les applications. Le client souhaitait également que le Qcm ne soit disponible que pour une période précise, les champs beginning\_at et finished\_at qui correspondent respectivement à la date de début de disponibilité du Qcm et à la date de fin de disponibilité du Qcm.

Enfin les champs Created\_at et Updated\_at correspondent à la date de création du Qcm et à la dernière date de modification de ce Qcm. Cela permet d’avoir un suivit du Qcm en question au sein de la base de données.

Table Category :

La table Category correspond à une matière au sein de notre application. En effet, un Qcm doit obligatoirement appartenir à une Catégorie. Au sein de cette table nous trouvons simplement un champ Name qui correspond au nom de la catégorie. Comme la plupart des tables, les champs created\_at et updated\_at sont présent et représentent les dates de création et de dernière modification de la catégorie.

Table Question :

La table Question correspond aux différentes questions au sein d’un Qcm, par conséquent une question doit obligatoirement être associée à un Qcm. Le nombre de question associée à un Qcm n’est pas limité. Au sein de cette table nous avons un champ Title qui correspond à l’intitulé de la question ainsi qu’un champ value qui indique le nombre de point que vaut la question au sein du Qcm.

Enfin les champs Created\_at et Updated\_at correspondent à la date de création d’une question et à la dernière date de modification de cette question. Cela permet d’avoir un suivit des questions au sein de la base de données.

Table Answer :

La table Answer correspond aux différentes réponses au sein d’une question, par conséquent une réponse doit obligatoirement être associée à une question. Le nombre de réponses associées à une Question n’est pas limité. Au sein de cette table on trouve un champ Title qui correspond à l’intitulé de la réponse. De plus nous avons un champ is\_valid qui permet de savoir si une réponse est correcte ou non. Un champ is\_selected est également présent sur les applications afin de savoir ce que l’utilisateur a sélectionné. Dans le cas où l’utilisateur coche cette réponse le booléen passe à True et l’on sait lors du calcul de la note sur le webservice ce que l’utilisateur a répondu.

Enfin les champs Created\_at et Updated\_at correspondent à la date de création d’une réponse et à la dernière date de modification de cette réponse. Cela permet d’avoir un suivit des réponses au sein de la base de données.

Table Media :

La table Media correspond aux différents médias qui vont être lié avec une question (Image, Vidéos ou Audio). Le champ Name correspond au nom du média en question tandis que le champ url correspond à l’emplacement du média situé sur le serveur.

Enfin les champs Created\_at et Updated\_at correspondent à la date de création d’un média et à la dernière date de modification de ce média. Cela permet d’avoir un suivit des réponses au sein de la base de données.

Table Type\_Media

La table TypeMedia correspond aux différents types de médias qui caractérisent un média. Un média ne peut avoir qu’un seul type de média. Au sein de cette table on trouve seulement un champ Name qui correspond au nom du type média en question.

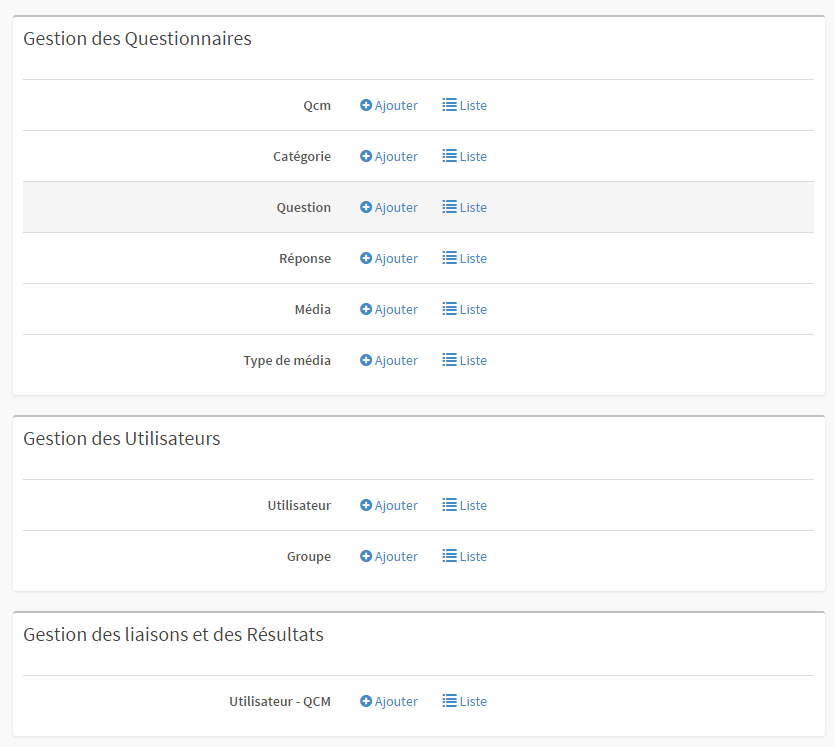
# Maquettage du BackOffice

### Maquette Home

Voici ci-joint l’écran principal du BackOffice, il permet la redirection sur les différentes interfaces permettant la gestion des entités présente sur le Webservice. Chaque table est représentée sous forme de ligne avec deux options principales : *Ajouter* et *Lister*.

Les Entités ont été classées par groupe pour améliorer la lisibilité de l’utilisateur lors de ces actions. En effet, 3 groupes ont été créés :

* Gestion des questionnaires : Cela concerne toute la gestion des Qcm et leurs dépendances. Au sein de ce groupe on retrouve donc les Catégories, les Questions, Les réponses, les Médias et les TypeMedia.
* Gestion des utilisateurs : Au sein de ce groupe l’administrateur peut gérer les utilisateurs ainsi que les groupes d’utilisateurs.
* Gestion des liaisons et des résultats : Ce dernier groupe permet de lier un Utilisateur à un Qcm. Il permet également de visualiser les résultats d’un Utilisateur pour un Qcm.



Ci-dessous les détails des actions possibles sur la page principal du BackOffice avec le groupe associé, la destination et la méthode appelée pour aller jusqu’à celle-ci.

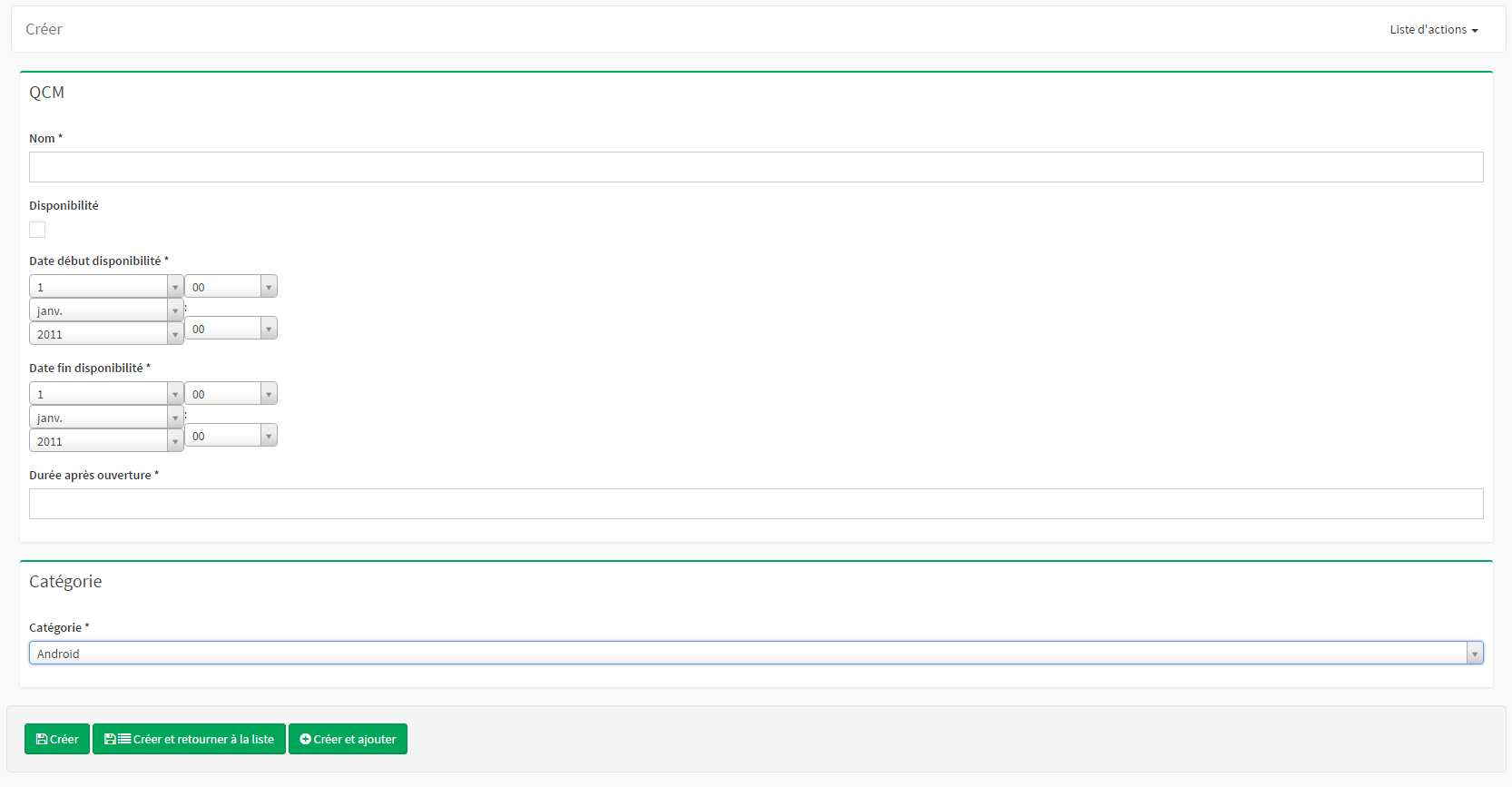
|  |  |  |  |
| --- | --- | --- | --- |
| Elément | Groupe | Destination | Méthode appelé |
| Ajouter Qcm | Questionnaires | admin/qcm/webservice/qcm/create | QcmAdmin/configureFormFields(FormMapper $formMapper) |
| Lister Qcm | Questionnaires | admin/qcm/webservice/qcm/list | QcmAdmin/configureListFields(ListMapper $listMapper) |
| Ajouter Catégorie | Questionnaires | admin/qcm/webservice/category/create | CategoryAdmin/configureFormFields(FormMapper $formMapper) |
| Lister Catégorie | Questionnaires | admin/qcm/webservice/category/list | CategoryAdmin/configureListFields(ListMapper $listMapper) |
| Ajouter Question | Questionnaires | admin/qcm/webservice/question/create | QuestionAdmin/configureFormFields(FormMapper $formMapper) |
| Lister Question | Questionnaires | admin/qcm/webservice/ question /list | QuestionAdmin/configureListFields(ListMapper $listMapper) |
| Ajouter Réponse | Questionnaires | admin/qcm/webservice/answer/create | AnswerAdmin/configureFormFields(FormMapper $formMapper) |
| Lister Réponse | Questionnaires | admin/qcm/webservice/answer /list | AnswerAdmin/configureListFields(ListMapper $listMapper) |
| Ajouter Média | Questionnaires | admin/qcm/webservice/media/create | MediaAdmin/configureFormFields(FormMapper $formMapper) |
| Lister Média | Questionnaires | admin/qcm/webservice/media /list | MediaAdmin/configureListFields(ListMapper $listMapper) |
| Ajouter Type Média | Questionnaires | admin/qcm/webservice/typemedia /create | TypeMediaAdmin/configureFormFields(FormMapper $formMapper) |
| Lister Type Média | Questionnaires | admin/qcm/webservice/typemedia /list | TypeMediaAdmin/configureListFields(ListMapper $listMapper) |
| Ajouter Utilisateur | Utilisateurs | admin/qcm/webservice/user/create | UserAdmin/configureFormFields(FormMapper $formMapper) |
| Lister Utilisateur | Utilisateurs | admin/qcm/webservice/user/list | UserAdmin/configureListFields(ListMapper $listMapper) |
| Ajouter Groupe | Utilisateurs | admin/qcm/webservice/group/create | GroupAdmin/configureFormFields(FormMapper $formMapper) |
| Lister Groupe | Utilisateurs | admin/qcm/webservice/group/list | GroupAdmin/configureListFields(ListMapper $listMapper) |
| Ajouter Utilisateur - Qcm | Liaisons utilisateurs - résultats | admin/qcm/webservice/user\_qcm/create | UserQcmAdmin/configureFormFields(FormMapper $formMapper) |
| Lister Utilisateur - Qcm | Liaisons utilisateurs - résultats | admin/qcm/webservice/user\_qcm/list | UserQcmAdmin/configureListFields(ListMapper $listMapper) |

### MAQUETTE Ajout qCM

Ci-dessous le formulaire Sonata, d’ajout d’un Qcm. Les champs affichés reprennent les propriétés de l’entité Qcm en question, seul les propriétés renseigné automatiquement ne sont pas présente au sein de ce formulaire (Id, Created\_at et Updated\_at).

La fonction *configureFormFields()* présente dans *Admin\AdminBundle\Admin\QcmAdmin* permet la construction de ce formulaire. Au sein de celle-ci est renseigné les différents champs présent sur ce formulaire ainsi que leur type (Date, Boolean, Entity, String).

De plus, au sein de l’entité *Qcm* présente dans *QCM\webserviceBundle\Entity* on a spécifié les champs qui sont obligatoire ou ceux qui ont des caractéristiques spécifiques (taille minimum, taille maximum, extension particulière)



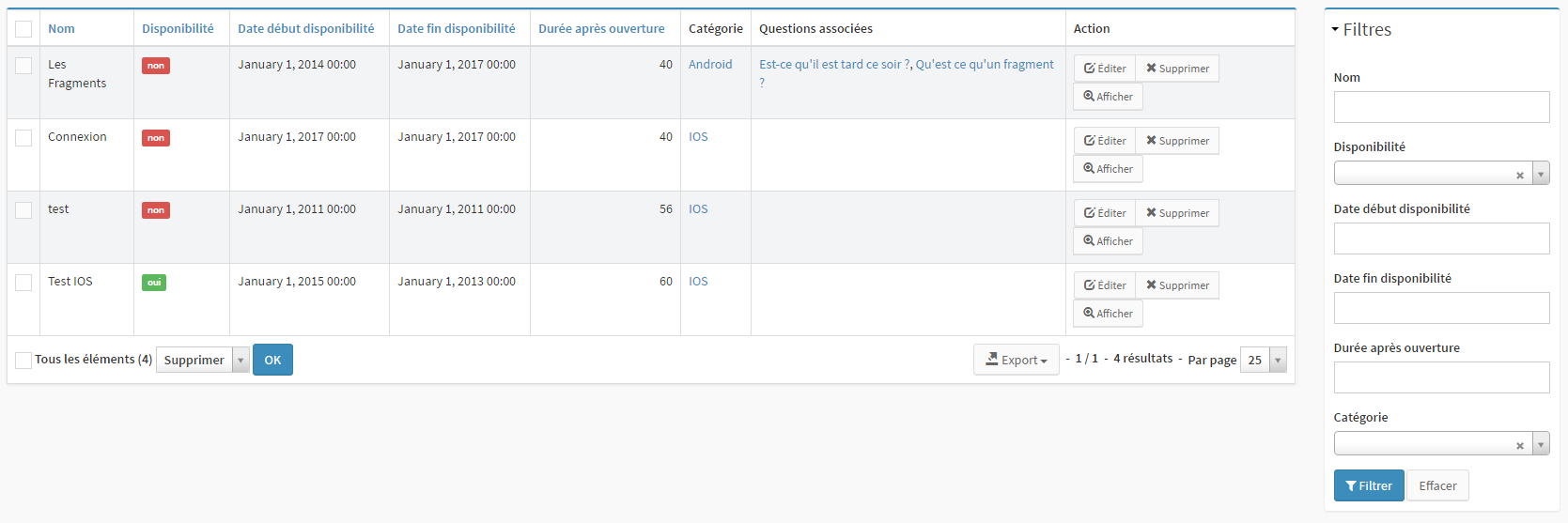
Ci-dessous le détail des champs présents sur le formulaire d’ajout de Qcm.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Elément | Type | Obligatoire | Nom en base | Valeur | Erreur |
| Nom | String | Oui | name | Nom du Qcm. | « Veuillez renseigner ce champs. » |
| Disponibilité | Boolean | Non | isAvailable | Valeur de disponibilité du Qcm sur l’application. |  |
| Date de début de disponibilité | Date | Oui | beginningAt | Date de début de disponibilité du Qcm sur l’application. | « Veuillez renseigner ce champs. » |
| Date de fin de disponibilité | Date | Oui | finishedAt | Date de début de disponibilité du Qcm sur l’application. | « Veuillez renseigner ce champs. » |
| Durée après ouverture | Integer | Oui | duration | Durée du Qcm lors du lancement de la première question. | « Veuillez renseigner ce champs. » |
| Catégorie | Entity | Oui | category | Lien avec l’entité Catégorie, permet d’attribuer une catégorie a un Qcm. |  |

### Maquette Liste Qcm

Ci-dessous la page Sonata de liste des Qcm. Les champs affichés reprennent les propriétés de l’entité Qcm en question, seul les propriétés renseignées au sein de la fonction *configureListFields()* présente dans *Admin\AdminBundle\Admin\QcmAdmin* sont affichées dans le tableau. Cette fonction permet la construction du tableau ainsi que les actions disponibles pour chaque ligne (Editer, Supprimer et Afficher).

De plus, des filtres sont disponibles sur la même page grâce à la méthode *configureDatagridFilters()* présente dans *Admin\AdminBundle\Admin\QcmAdmin.* Cette fonction permet de constituer les champs présents au sein de la partie filtre de la page. Une fois le filtre appliqué la liste se rafraichit et affiche simplement les résultats qui correspondent à la recherche.



Ci-dessous les détails de la liste de Qcm :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Elément | Type | Nom en base | Valeur | Destination |
| Nom | String | name | Nom du Qcm. |  |
| Disponibilité | Boolean | isAvailable | Valeur de disponibilité du Qcm sur l’application. |  |
| Date de début de disponibilité | Date | beginningAt | Date de début de disponibilité du Qcm sur l’application. |  |
| Date de fin de disponibilité | Date | finishedAt | Date de début de disponibilité du Qcm sur l’application. |  |
| Durée après ouverture | Integer | duration | Durée du Qcm lors du lancement de la première question. |  |
| Catégorie | Entity | category | Lien avec l’entité Catégorie, permet d’attribuer une catégorie à un Qcm. | admin/qcm/webservice/category/#/edit |
| Questions | Entity | question | Questions associées à ce Qcm. | admin/qcm/webservice/question/#/edit |

Ci-dessous les détails des actions possibles sur une ligne de la liste de Qcm :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Action | Bouton | Méthode | Destination | Description |
| Edit | Editer | configureFormFields() | admin/qcm/webservice/qcm/#/edit | Au clique du bouton nous sommes redirigés sur la page en question. |
| Delete | Supprimer |  | admin/qcm/webservice/qcm/#/delete |
| Show | Afficher | configureShowFields() | admin/qcm/webservice/qcm/#/show |

Ci-dessous les détails concernant le tableau de filtre des Qcm :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Elément | Type | Nom en base | Valeur | Méthode |
| Nom | String | name | Nom du Qcm. | Lors du clique sur le bouton « Filtrer » une méthode de Sonata est appelée afin de rafraichir la liste des Qcm présent sur la partie Tableau des Qcm. Cette fonction est inclut directement dans le Bundle Sonata et ne peux donc pas être modifié. |
| Disponibilité | Boolean | isAvailable | Valeur de disponibilité du Qcm sur l’application. |
| Date de début de disponibilité | Date | beginningAt | Date de début de disponibilité du Qcm sur l’application. |
| Date de fin de disponibilité | Date | finishedAt | Date de début de disponibilité du Qcm sur l’application. |
| Durée après ouverture | Integer | duration | Durée du Qcm lors du lancement de la première question. |
| Catégorie | Entity | category | Lien avec l’entité Catégorie, permet d’attribuer une catégorie à un Qcm. |

### Maquette Afficher Qcm

Ci-dessous un exemple de maquette présent sur la BackOffice et permettant de visualiser le Qcm sélectionné grâce au bouton « Afficher » au sein de la maquette « Liste des Qcm ». La méthode appelée pour afficher cette page est *configureShowFields(),* présente dans la classe *Admin\AdminBundle\Admin\QcmAdmin.*

Au sein de cette fonction on renseigne les champs qui seront visible sur la page « Afficher Qcm ». Un lien est toujours disponible afin de visualiser la catégorie associé au Qcm.



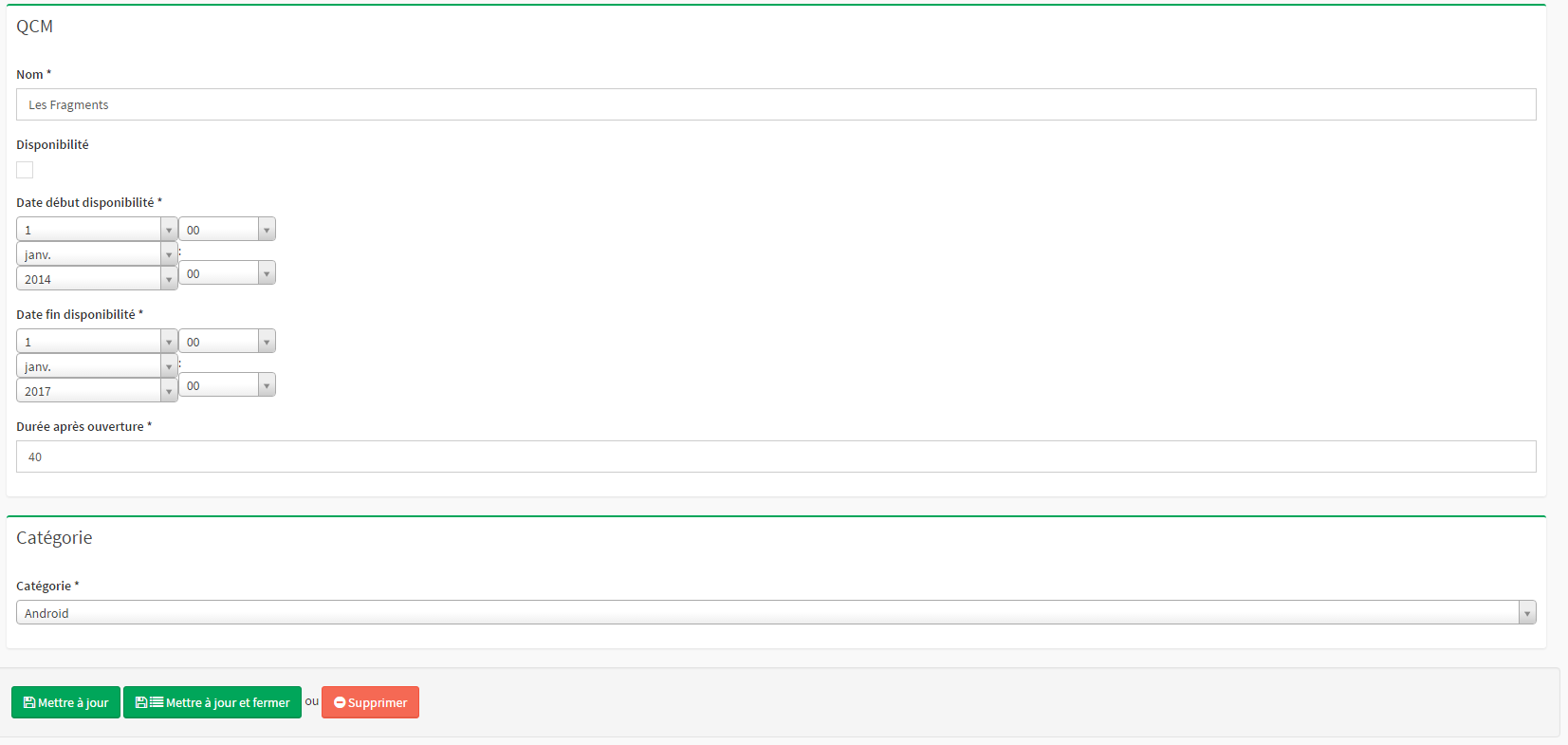
Ci-dessous le détail des propriétés présentent sur cette page.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Elément | Type | Obligatoire | Nom en base | Valeur |
| Nom | String | Oui | name | Nom du Qcm. |
| Disponibilité | Boolean | Non | isAvailable | Valeur de disponibilité du Qcm sur l’application. |
| Date de début de disponibilité | Date | Oui | beginningAt | Date de début de disponibilité du Qcm sur l’application. |
| Date de fin de disponibilité | Date | Oui | finishedAt | Date de début de disponibilité du Qcm sur l’application. |
| Durée après ouverture | Integer | Oui | duration | Durée du Qcm lors du lancement de la première question. |
| Catégorie | Entity | Oui | category | Lien avec l’entité Catégorie, permet d’attribuer une catégorie a un Qcm. |

### Maquette mise à jour Qcm

Ci-dessous un exemple de maquette présent sur la BackOffice et permettant d’éditer le Qcm sélectionné grâce au bouton « Editer » au sein de la maquette « Liste des Qcm ». La méthode appelée pour afficher cette page est *configureFormFields(),* présente dans la classe *Admin\AdminBundle\Admin\QcmAdmin.*

Cette fonction est identique à celle appelée lors de l’ajout d’un Qcm et reprends donc les champs présents dans celle-ci. De plus, un bouton de suppression est apparu en bas de la page permettant la suppression de l’élément en question. La méthode appelée lors du clique est une méthode Sonata et ne peux donc pas être modifié.



Ci-dessous le détail des champs présents sur le formulaire d’édition de Qcm.

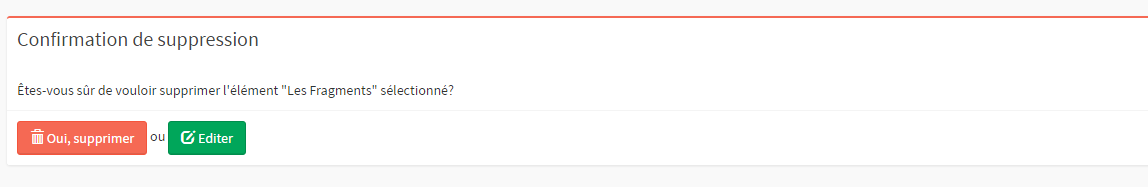
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Elément | Type | Obligatoire | Nom en base | Valeur | Erreur |
| Nom | String | Oui | name | Nom du Qcm. | « Veuillez renseigner ce champs. » |
| Disponibilité | Boolean | Non | isAvailable | Valeur de disponibilité du Qcm sur l’application. |  |
| Date de début de disponibilité | Date | Oui | beginningAt | Date de début de disponibilité du Qcm sur l’application. | « Veuillez renseigner ce champs. » |
| Date de fin de disponibilité | Date | Oui | finishedAt | Date de début de disponibilité du Qcm sur l’application. | « Veuillez renseigner ce champs. » |
| Durée après ouverture | Integer | Oui | duration | Durée du Qcm lors du lancement de la première question. | « Veuillez renseigner ce champs. » |
| Catégorie | Entity | Oui | category | Lien avec l’entité Catégorie, permet d’attribuer une catégorie a un Qcm. |  |

### Maquette de suppression d’un Qcm

Ci-dessous un exemple de maquette présent sur la BackOffice et permettant de supprimer le Qcm sélectionné grâce au bouton « Supprimer » au sein de la maquette « Liste des Qcm ». La méthode appelée pour afficher cette page est une méthode de Sonata et ne peux donc pas être modifiée.

Lors du clique sur le bouton « Oui, supprimer », l’élément sélectionné sera supprimé de la base de données et ne pourra être récupéré.

Lors du clique sur le bouton « Editer », la méthode *configureFormFields(),* présente dans la classe *Admin\AdminBundle\Admin\QcmAdmin* sera appelée et permettra la redirection de l’utilisateur sur la page *admin/qcm/webservice/qcm/#/edit* correspondant à la maquette « Editer un Qcm »*.*



### Note

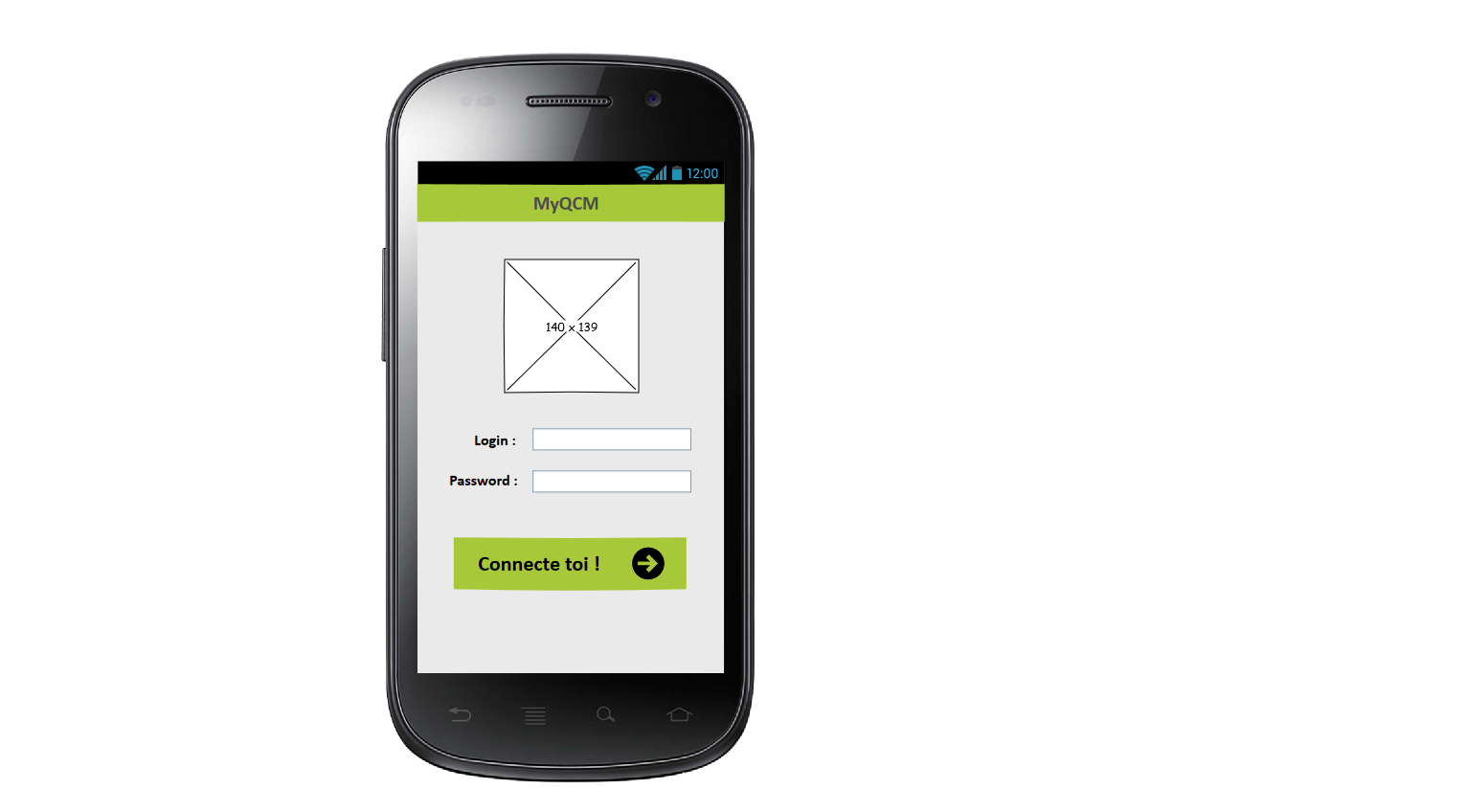
Au sein de ce document seul les différentes actions concernant l’entité *Qcm* ont étés détaillées. Les autres entités, à savoir *User*, *Group*, *Question*, *Answer*, *Category*, *Media*, *TypeMedia* et *UserQcm* fonctionne selon le même modèle seul les propriétés de chaque objet sont différentes*.* Nous avons donc décidé, dans un souci de longueur de document, de ne pas répéter l’opération de détail pour la totalité de ces entités.

# Maquettage Application mobile

### Maquette de connexion

Lors de l’ouverture de l’application, la première page qui apparaît est celle de connexion. Afin de pouvoir utiliser l’application dans sa globalité, l’utilisateur doit saisir son **username** et son **mot de passe** puis cliquer sur « **Se connecter »**. Le bouton de connexion devient cliquable que lorsque le champ **username** et le champ **mot de passe** est renseigné.

Une icône ou une image à l’effigie de TactFactory sera également prévu au sein de cette page de connexion.



Fonctions Appelées lors de la connexion :

|  |  |  |  |
| --- | --- | --- | --- |
| Classes | Nom de la fonction | Paramètre(s) | Retour |
| UserSQLiteAdapter | open() |  |  |
| UserWSAdapter | getUser() | String username / JsonHttpResponseHandler handler | JSONObject response |
| UserWSAdapter | jsonToItem() | JSONObject response | User item |
| CategorySQLiteAdapter | insert() | Category category | Long id |
| CategorySQLiteAdapter | open() | - | - |
| CategorySQLiteAdapter | close() | - | - |
| QcmSQLiteAdapter | open() | - | - |
| QcmSQLiteAdapter | close() | - | - |
| QcmSQLiteAdapter | Insert() | Qcm qcm | Long id |
| QuestionSQLiteAdapter | open() |  | - |
| QuestionSQLiteAdapter | close() |  |  |
| QuestionSQLiteAdapter | insert() | Question question | Long id |
| AnswerSQLiteAdapter | open() |  | - |
| AnswerSQLiteAdapter | close() |  |  |
| AnswerSQLiteAdapter | insert() | Answer answer | Long id |

Détails des fonctions :

**UserSQLiteAdapter** : Classe qui permet la relation avec la base de données (notamment avec la table User)

* **open()** : Permet d’ouvrir la base de donnée en lecture et écriture

Note : *la fonction open() de chaque classe permet d’ouvrir la base de donnée en lecture/écriture donc on ne détaillera pas ses fonctions des autres classes.*

* **close()**:Permet de fermer la connexion avec la base de données

Note : *la fonction close() de chaque classe permet de fermer la connexion avec la base de données donc on ne détaillera pas cette fonction pour les autres classes.*

**UserWSAdapter** : Classe qui permet de contacter le webservice.

* **getUser()** : Permet de récupérer un utilisateur avec toutes ses informations.
* **jsonToItem()** : Permet de convertir un JSONObject en objet User.

**CategorySQLiteAdapter** : Classe qui permet la relation avec la base de données (notamment avec la table Category).

* **insert()** : Permet d’insérer une catégorie en base de donnée.

**QcmSQLiteAdapter** : Classe qui permet la relation avec la base de données (notamment avec la table QCM).

* **insert()** : Permet d’insérer un QCM en base de données.

**QuestionSQLiteAdapter** : Classe qui permet la relation avec la base de données (notamment avec la table Question).

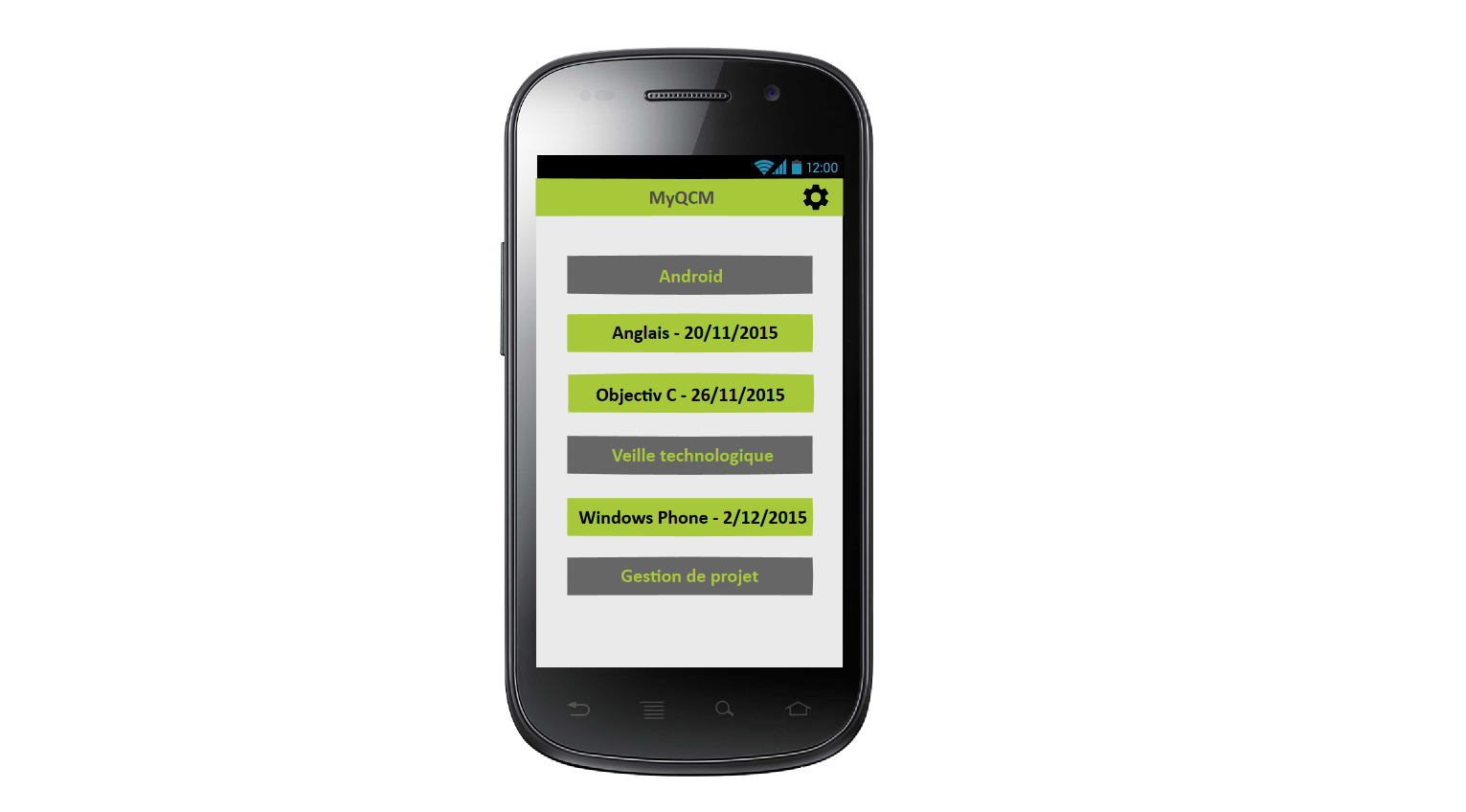
* **insert()** : Permet d’insérer une question en base de données.

**AnswerSQLiteAdapter** : Classe qui permet la relation avec la base de données (notamment avec la table Answer).

* **insert()** : Permet d’insérer une réponse en base de données

### Maquette Liste catégorie.

Une fois la connexion effectuée avec succès, l’utilisateur arrivera sur la page Catégorie qui liste les différentes catégories dans lesquelles un ou plusieurs Qcm est présent et disponible. Ci-dessous une maquette représentative de la version finale de l’application.



Fonctions Appelées lors de l’affichage de la liste :

|  |  |  |  |
| --- | --- | --- | --- |
| Classes | Nom de la fonction | Paramètre(s) | Retour |
| CategorySQLiteAdapter | open() |  |  |
| CategorySQLiteAdapter | getAllCursor | - | Cursor cursor |
| CategorySQLiteAdapter | close() | - | - |

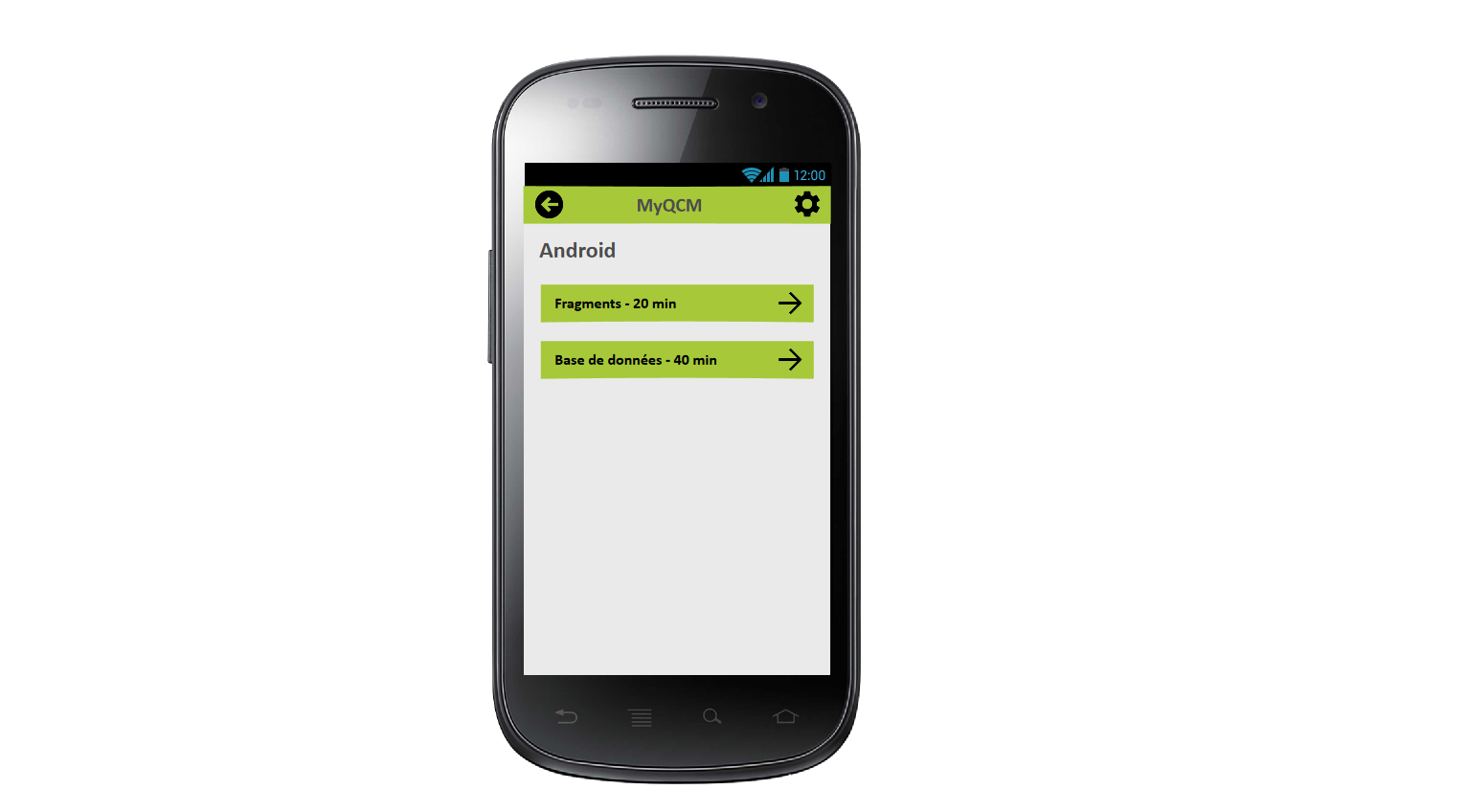
Détails des fonctions :

**CategorySQLiteAdapter** : Classe qui permet la relation avec la base de données (notamment avec la table Category).

* open() : Permet d’ouvrir la base de données en lecture/écriture.
* getAllCursor : Récupère toutes les catégories en base de données.
* close() : Permet de fermer la connexion avec la base de données.

### Maquette liste Qcm.

La page contenant la liste des QCM est appelé lors d’un clic sur l’une des catégories présente sur la maquette précédente. Son rôle est donc d’afficher la liste des QCM appartenant à la catégorie choisit par l’utilisateur grâce à son nom et on lui indique également la durée maximum pour réaliser celui-ci. Le nombre de QCM par catégorie n’est pas limité.



Fonctions appelées :

|  |  |  |  |
| --- | --- | --- | --- |
| Classes | Nom de la fonction | Paramètre(s) | Retour |
| QcmSQLiteAdapter | open() |  |  |
| QcmSQLiteAdapter | getAllCursorByCategory() | Long categoryId | Cursor cursor |
| QcmSQLiteAdapter | close() | - | - |

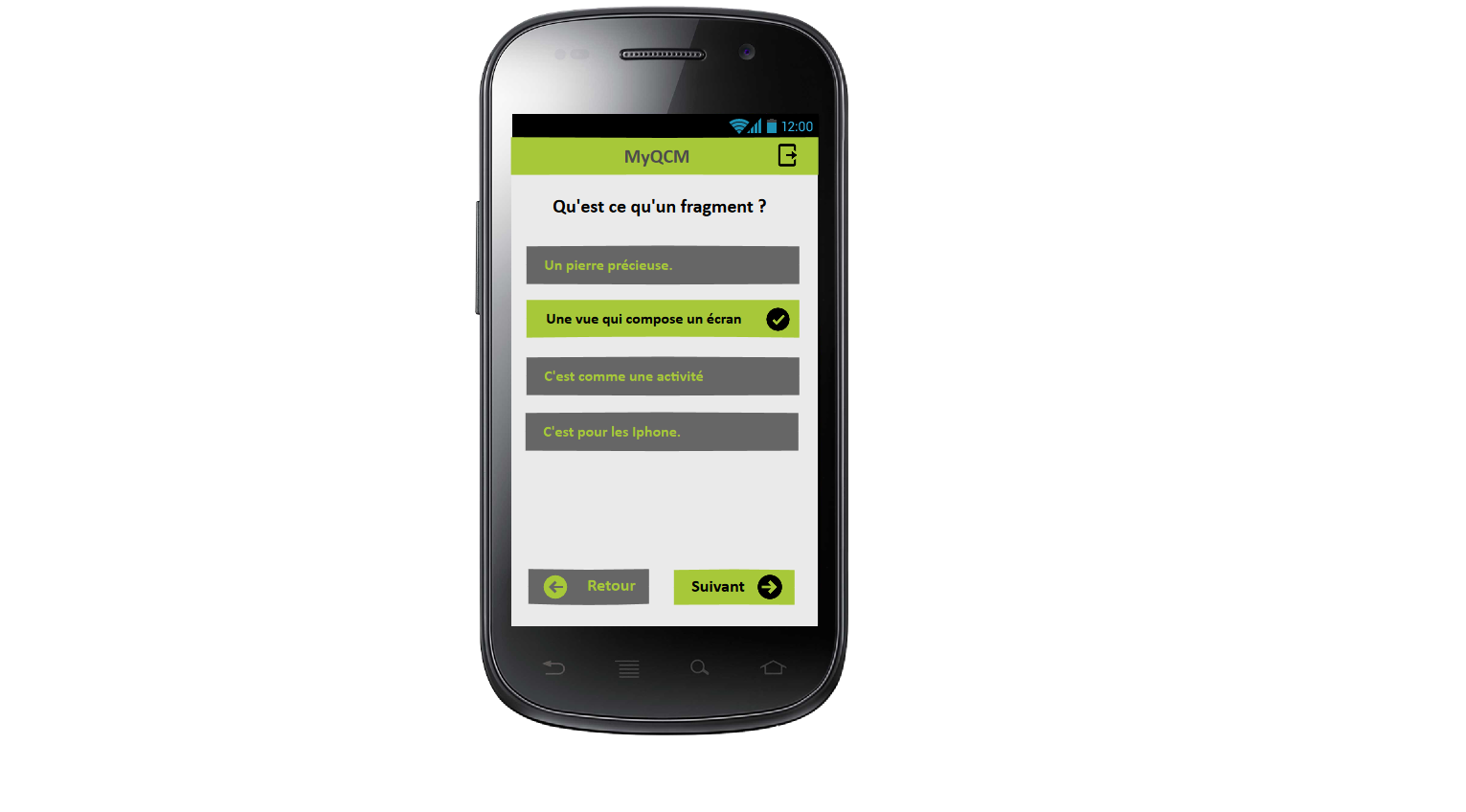
Détails des fonctions :

**QcmSQLiteAdapter** : Classe qui permet la relation avec la base de données (notamment avec la table Qcm)

* open() : Permet d’ouvrir la base de données en lecture/écriture
* getAllCursorByCategory() : Récupère tous les QCMS en base de données en fonction de l’id de la catégorie
* close() : Permet de fermer la connexion avec la base de données

### Maquette questions

La page contenant les questions du QCM est appelée lors du clic de l’utilisateur sur un QCM sur la maquette précédente. L’utilisateur arrive alors sur une liste de question présenté aléatoirement. Plusieurs choix sont disponibles pour chacune d’entre elle. Il a également la possibilité de revenir en arrière pour modifier les questions précédemment renseigné. De plus, si l’utilisateur quitte l’application ou dépasse le temps qui était impartit pour réaliser le QCM, l’application enregistre les réponses apporté et quitte celui-ci.



Fonctions appelees :

|  |  |  |  |
| --- | --- | --- | --- |
| Classes | Nom de la fonction | Paramètre(s) | Retour |
| QuestionSQLiteAdapter | open() |  |  |
| QuestionSQLiteAdapter | getAllByQcm | Long qcmId | ArrayList<Question> questionList |
| QuestionSQLiteAdapter | close() | - | - |
| AnswerSQLiteAdapter | open() | - | - |
| AnswerSQLiteAdapter | getAllByQuestion() | Long questionId | ArrayList<Answer> answerList |
| AnswerSQLiteAdapter | close() | - | - |

Détails des fonctions :

**QuestionSQLiteAdapter** : Classe qui permet la relation avec la base de données (notamment avec la table Question)

* open() : Permet d’ouvrir la base de données en lecture/écriture

Note : *la fonction open() de chaque classe permet d’ouvrir la base de donnée en lecture/écriture donc on ne détaillera pas ses fonctions des autres classes.*

* getAllByQuestion() : Récupère toutes les questions en base de données en fonction de l’id du QCM
* close() : Permet de fermer la connexion avec la base de données

Note : *la fonction close() de chaque classe permet de fermer la connexion avec la base de données donc on ne détaillera pas cette fonction pour les autres classes.*

**AnswerSQLiteAdapter :** Classe qui permet la relation avec la base de données (notamment avec la table Answer)

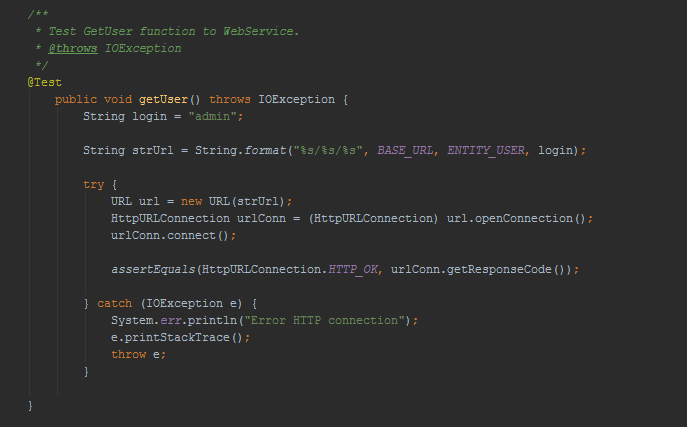
* getAllByQuestion() : Récupère toutes les réponses en base de données en fonction de l’id de la question

# Choix de la méthode de développement

Ci-dessous le schéma de développement qui a été mis en place durant la réalisation du projet Qcm. Il répertorie chronologiquement les étapes importantes qui ont permis au projet d’avancer dans des conditions adéquates.

### Test Unitaire

Au sein du projet des tests unitaires ont été codé afin de vérifier que tout au long de l’avancement du projet les fonctions restaient valides. Ces tests unitaires n’ont étés effectué que au sein des applications et test également la récupération de flux JSON et la connexion au Web Service. Ci-dessous un exemple de test sur l’application Android permettant de tester la connexion au Web Service depuis l’application.



### Versionning

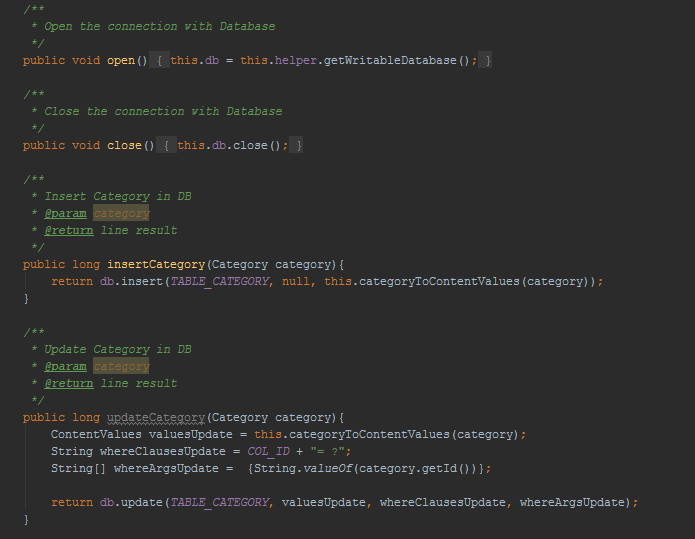
Au sein du projet Qcm nous avons utilisé Git comme outil de versionning. En effet, l’ensemble des dépôts (Applications et Web Service) est stocké sur l’outil Git Hub. Chaque application a son dépôt distant ainsi que le Webservice. De plus deux branches ont étés créer à l’intérieur de chaque dépôt. L’une est une branche *Master* permettant les push de fin de version tandis que l’autre, la branche *Développement* permet des push régulier pour garantir une sauvegarde quotidienne des projets développés.

Enfin, afin de simplifier la gestion de tous ces dépôts nous avons opté pour un outil de gestion de dépôt qui se nomme Source Tree. Il permet de visualiser et de gérer simplement et rapidement ces dépôts depuis son ordinateur.

### Convention de nommage

Au sein du projet est également présent une convention de nommage que nous respectons au sein de notre projet Qcm. En effet cette convention de nommage indique la marche à suivre quant au nommage de différentes informations présentes au sien du code source. Nous utilisons plusieurs langages différents tel que le Java, le C#, le PHP ou encore l’Objective C. Toutes les règles de nommage sont présentent au sien de ce document.

### Commentaires

Tout au long du projet nous commenterons notre projet afin que celui-ci soit maintenu et lisible de tous. Différentes documentations seront mises à votre disposition afin que vous visualisiez le rôle de chacune des méthodes présentes au sein des applications et du Web Service. Ci-dessous un exemple de commentaire au sein du code Android.

### Le choix symfony 2

Symfony 2 étant un Framework PHP puissant du fait qu’il intègre de nombreux Bundle au sein de sa structure, il est également conçut sur une architecture MVC. C’est un outil qui permet une augmentation importante de production. Au sein de notre projet nous avions besoin d’un BackOffice simple et rapide à mettre en place. Sonata étant un Bundle facilement intégrable à Symfony nous avons opté pour celui-ci.

Au final, pour des raisons de productions et de richesse le Bundle Symfony s’avérait être un choix judicieux et pertinent quant à la réalisation du projet.

### Les design pattern présent dans le projet

MVC (Modèle – View – Controleur) :

Nous utilisons le design pattern MVC au sein du développement du Web Service. En effet, Symfony 2 utilise une architecture MVC.

mvvm (Modèle – View – View Model:

Nous utilisons le design pattern MVVM au sein du développement de l’application mobile Windows Phone. Windows phone est un langage adéquate pour l’utilisation d’une tel architecture notamment grâce au Binding et aux commandes que propose le langage avec des liaisons faibles.

