Logo

Description automatically generated

**RAPPORT DE PROJET**

**SITE WEB DE FORMATION**

*DEVELOPPEMENT D’APPLICATION WEB*

*2022-2023*

Membres du groupe n°2 :

SALAH ISMAIL

BEAUJON ALEXANDRE

MALOIGNE ANTHONY

PERE BRANDON

ROBINET PERRINE

STRINCHAMPS Clothilde

Table des matières

[Introduction et présentation de l’équipe 2](#_Toc134132730)

[I. Introduction 2](#_Toc134132731)

[II. Présentation de l’équipe et répartition des tâches 2](#_Toc134132732)

[1. L’équipe de développement : 2](#_Toc134132733)

[2. La répartition des tâches : 3](#_Toc134132734)

[Modélisation de la plateforme 4](#_Toc134132735)

[I. Présentation du projet 4](#_Toc134132736)

[II. Analyse des besoins 4](#_Toc134132737)

[III. Diagramme Use Case 5](#_Toc134132738)

[Implémentation 6](#_Toc134132739)

[I. Organisation du site 6](#_Toc134132740)

[II. Le modèle MVC 6](#_Toc134132741)

[Architecture et programmation 8](#_Toc134132742)

[I. Dossier « model » 8](#_Toc134132743)

[1. Diagramme de classes de l’api base de données 8](#_Toc134132744)

[2. Explication des classes de l’API : 9](#_Toc134132745)

[3. Modélisation de la base de données : 10](#_Toc134132746)

[a. Points forts de cette modélisation de base de données : 11](#_Toc134132747)

[II. Dossier « core » 12](#_Toc134132748)

[III. Dossier « Controller » 13](#_Toc134132749)

[IV. Dossier « Views » 13](#_Toc134132750)

[Conclusion 14](#_Toc134132751)

# Introduction et présentation de l’équipe

## Introduction

Ce projet de création d’un site de formation a pour objectif de nous familiariser au développement des applications web dynamique en faisant usage des technologies PHP, HTML, CSS, XML et Javascript. C’est l’occasion de mettre en pratique les cours présentés par le module Développement d’Application Web (DAW). Le travail consiste à réaliser une plateforme de formations destinées à des enseignants/étudiants.

Ce document fait l’objet d’un compte rendu détaillé de la modélisation du site en question et de la gestion du projet. Il a été écrit en collaboration avec les 6 membres qui constituent le groupe n°2.

## Présentation de l’équipe et répartition des tâches

Nous avons utilisé la méthode agile pour la réalisation de ce projet. La méthode Agile offre une approche collaborative qui favorise une communication régulière entre les membres de l'équipe ce qui améliore l'efficacité du travail en équipe. De plus, la méthode Agile est conçue pour être adaptative et flexible. Enfin, la méthode Agile est centrée sur la satisfaction du client, ce qui est une priorité importante pour la réussite d’un projet avec de nombreuses contraintes évaluées. Nous avons donc choisi la méthode Agile pour notre projet en raison de son approche collaborative, de son adaptabilité et de sa priorité accordée à la satisfaction des consignes.

### L’équipe de développement :

* SALAH ISMAIL
* BEAUJON ALEXANDRE
* MALOIGNE ANTHONY
* PERE BRANDON
* ROBINET PERRINE
* STRINCHAMPS Clothilde

### La répartition des tâches :

* Architecture générale du projet / Core : Ismail
* Implémentation de la base de données : Ismail / Anthony
* Programmation de la partie modèle : Ismail / Anthony
* Programmation d’une API pour les fichier xml des QCM : Ismail / Anthony
* Programmation des Contrôleurs : Alexandre / Brandon / Perrine
* Programmation des Vues- HTML et CSS : Clothilde /Perrine

# Modélisation de la plateforme

## Présentation du projet

Notre objectif a été de développer un site web de formation en ligne permettant l’échange des cours entre les enseignants et les étudiants.

Notre site est capable d’assurer le bon fonctionnement des deux parties « utilisateur » :

Partie Administrateur :

* Charger les cours sous forme de diapos, vidéos ou simplement des fichiers
* Gérer les QCM

Partie apprenant :

* Suivre un cours et visualiser son contenu
* S’inscrire à un cours
* Faire les QCM des différents cours
* Avoir des recommandations de cours basé sur les résultats obtenue d’un QCM
* Participer dans le forum de discussion entre les apprenants

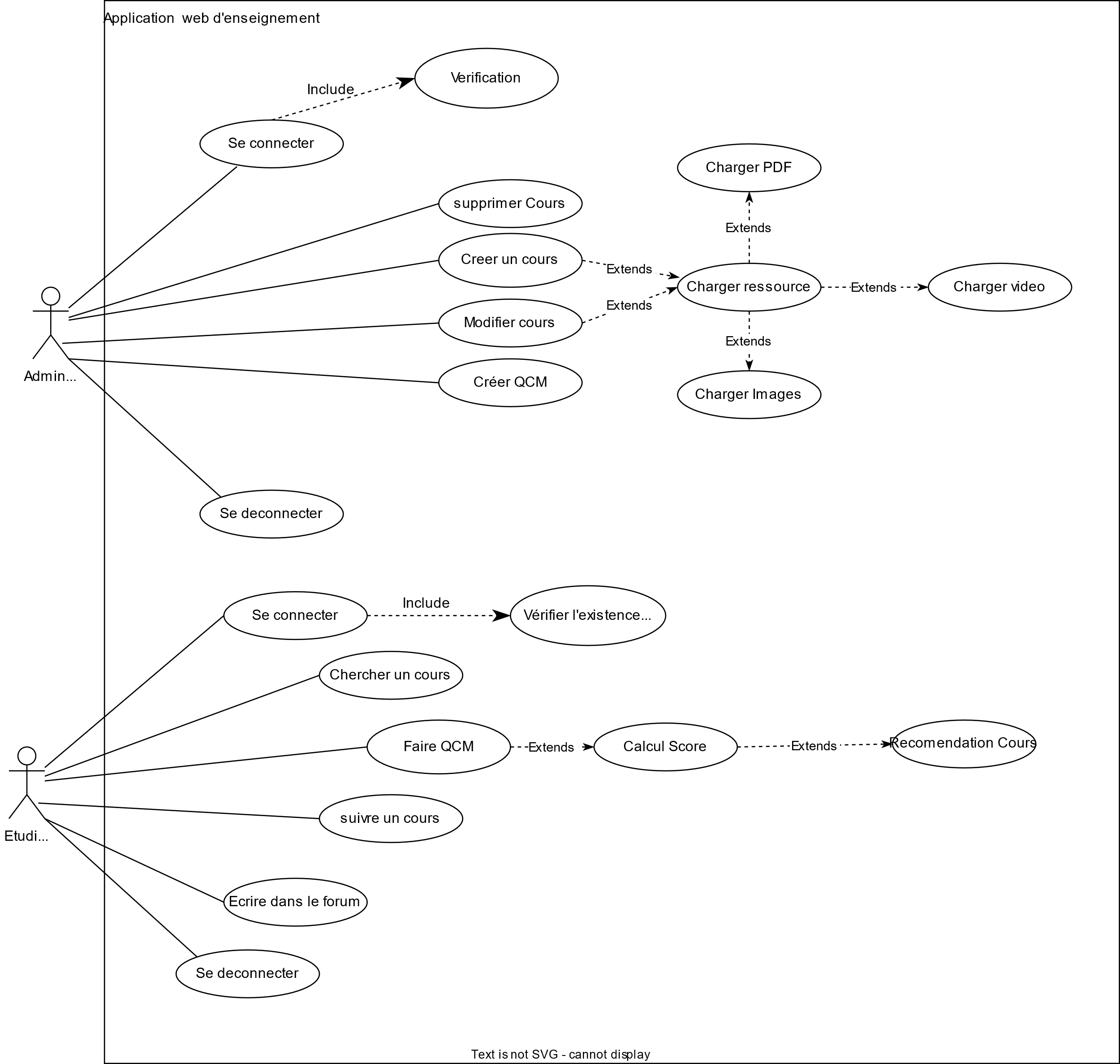
## Analyse des besoins

Dans cette partie, nous avons identifié les acteurs principaux du site web, leurs rôles ainsi que les contraintes d’accès et les limites de chaque acteur.

Notre site doit permettre :

* **A l’enseignant de :**
* **Créer un compte** : sur la page d’accueil, il doit cliquer sur la case enseignant afin de pouvoir surfer sur l’espace adapté
* **S’authentifier** : il doit s’authentifier pour accéder à son espace privé
* **Gérer un cours :** l’enseignant peut créer, modifier ou supprimer un cours et également ajouter des ressources
* **Gestion des QCM :** ajouter, modifier, supprimer des QCM
* **A l’étudiant de :**
* **S’authentifier :** il doit s’authentifier pour accéder à son espace privé, son compte doit être crée auparavant par un des enseignants
* **D’accéder au contenu des cours :** l’étudiant doit être capable de voir la totalité du cours avec toutes les ressources
* **Consulter la liste des cours recommandés :** en fonction de son niveau
* **D’accéder au forum :** afin de participer aux discussions avec d’autres étudiants

## Diagramme Use Case



# Implémentation

Notre site web de formation en ligne a été mis en place en utilisant les langages de programmation PHP, JAVASCRIPT, HTML, CSS et la base de données est implémentée en SQL avec MySQL.

## Organisation du site

Nous avions initialement organisé notre site en deux parties, app et public, formant deux dossiers principaux.   
Nous avons, au départ, développé le site avec cette architecture mais nous avons rapidement été contraint de revoir notre décision. En effet, cette architecture impose de nombreuses contraintes et demande surtout une documentation assez lourde afin d’être maintenu à jour. Certes, notre premier choix était plus intéressant en termes de sécurité informatique et d’implementation du concept MVC mais cela nous ralentissait beaucoup trop.

C’est pourquoi, pour une raison de praticité, nous avons développé l’ensemble de notre site dans la partie « app ».

## Le modèle MVC

Nous avons choisi l’architecture MVC afin de faciliter l’organisation du code source, cette dernière nous aide à séparer la logique du code en trois parties que l’on retrouve dans des fichiers distincts.

Une image contenant diagramme

Description générée automatiquement

***Modèle*** : cette partie gère les données de notre site. Son rôle est d'aller récupérer les informations « brutes » dans la base de données, de les organiser et de les assembler pour qu'elles puissent ensuite être traitées par le contrôleur. On y retrouve que du code PHP.

Ainsi que la partie responsable de gérer les fichiers XML des QCM.

***Vue*** : cette partie se concentre sur l'affichage. Elle ne fait presque aucun calcul et se contente de récupérer des variables/listes pour savoir ce qu'elle doit afficher. On y trouve essentiellement du code HTML pour les contenus, du css pour le design et du javascript pour les animations, ainsi que du PHP pour l’affichage personnalisé selon l’utilisateur connecté.

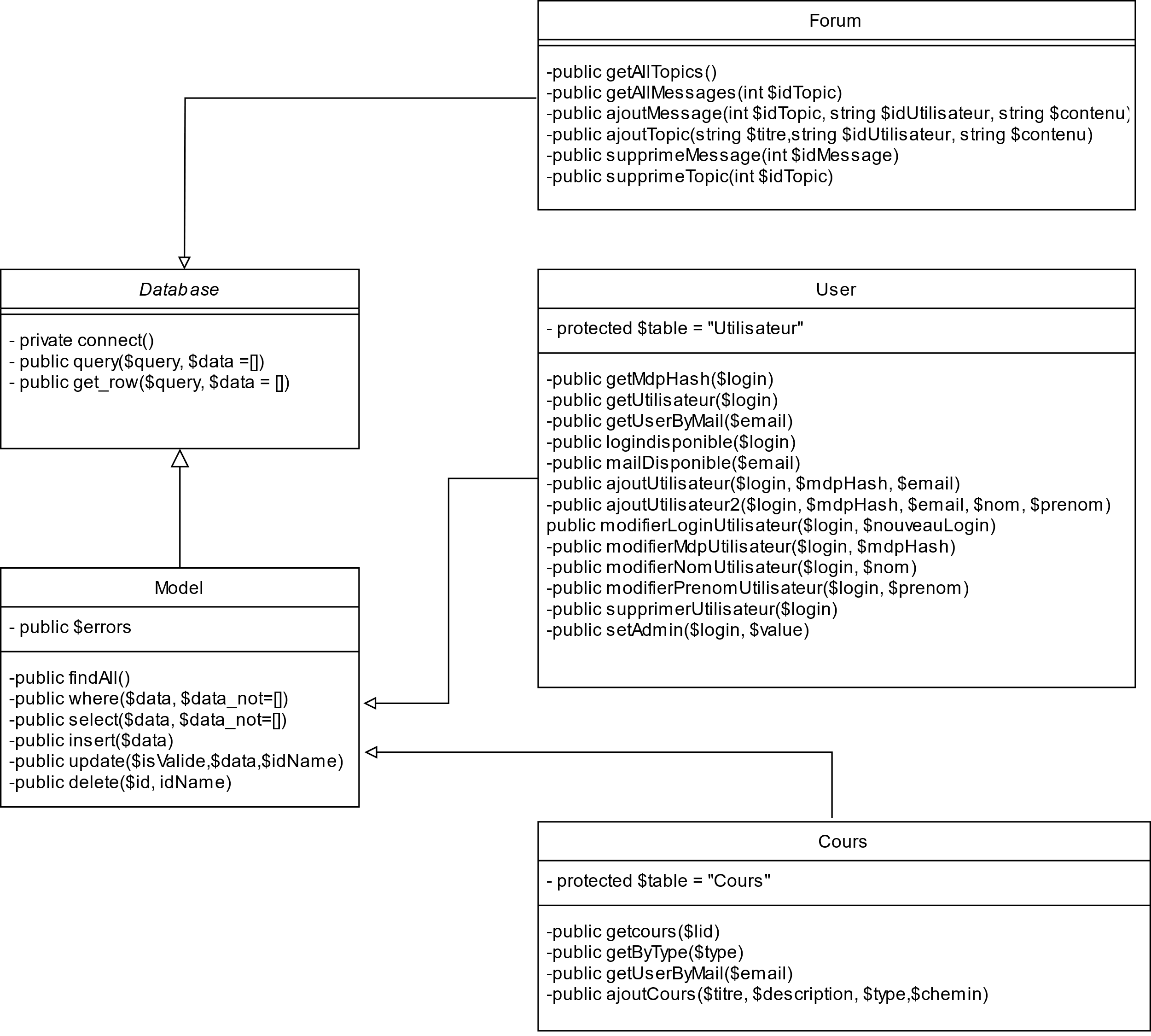
***Contrôleur*** : cette partie gère la logique du code qui prend des décisions. C'est en quelque sorte l'intermédiaire entre le modèle et la vue : le contrôleur va demander au modèle les données, les analyser, prendre des décisions et renvoyer le texte à afficher à la vue. Le contrôleur contient exclusivement du PHP. C'est notamment lui qui détermine si le visiteur a le droit de voir la page ou non (gestion des droits d'accès).

# Architecture et programmation

## Dossier « model »

Ceci représente l’API qui fait office d’interface entre le contrôleur et la base de données.

### Diagramme de classes de l’api base de données



### Explication des classes de l’API :

* **La classe Database :**

La classe nommée "database" a deux fonctions : connect() et query().

**La fonction connect()** : est une fonction privée qui établit une connexion à une base de données MySQL à l'aide de l'objet PDO (PHP Data Objects) et renvoie la connexion. Les informations de connexion sont stockées dans des constantes telles que DBHOST, DBNAME, DBUSER et DBPASS.

**La fonction query()** : est une fonction publique qui prend une requête SQL en tant que paramètre et éventuellement des données qui seront liées à la requête. Cette fonction appelle d'abord la fonction connect() pour établir une connexion à la base de données, puis prépare la requête SQL à l'aide de la méthode prepare() de l'objet PDO. La méthode execute() est ensuite appelée pour exécuter la requête SQL avec les données liées, si elles existent.   
Si la requête SQL s'exécute avec succès, la méthode fetchAll() est appelée pour récupérer toutes les lignes résultantes de la requête SQL sous forme d'objet PDO (FETCH\_OBJ). Si le résultat est un tableau et qu'il contient des éléments, alors la fonction retourne ce tableau.   
Si la requête SQL échoue, la fonction retourne false.   
En résumé, la classe database fournit une couche d'abstraction pour interagir avec une base de données MySQL en utilisant l'objet PDO et les fonctions préparées. La fonction query() est utilisée pour exécuter une requête SQL et récupérer les résultats sous forme d'objet PDO.

* **La classe Model :**

Le but de cette classe est d’établir une chaine de caractère qui sera exécuté ensuite en tant que requête SQL .   
Elle utilise la classe Database pour communiquer avec la base de données. Les méthodes suivantes sont définies dans cette classe :

**findAll () :** récupère tous les enregistrements de la table correspondante dans la base de données en fonction de l'ordre de tri, du nombre d'enregistrements et de l'offset définis pour cette instance de la classe. Elle appelle la méthode "query" de la classe "Database" pour exécuter la requête SQL.

**selectAll ():** récupère tous les enregistrements de la table correspondante dans la base de données qui correspondent aux conditions spécifiées dans le tableau $data, en excluant ceux qui correspondent aux conditions spécifiées dans le tableau $data\_not. Elle appelle également la méthode "query" de la classe "Database" pour exécuter la requête SQL.

**select ():** récupère le premier enregistrement correspondant aux conditions spécifiées dans le tableau $data, en excluant ceux qui correspondent aux conditions spécifiées dans le tableau $data\_not. Elle appelle également la méthode "query" de la classe "Database" pour exécuter la requête SQL.

**insert ():** insère une nouvelle ligne dans la table correspondante en utilisant les données spécifiées dans le tableau $data. Elle appelle également la méthode "query" de la classe "Database" pour exécuter la requête SQL.

**update ():** met à jour les enregistrements correspondant aux conditions spécifiées dans le tableau $data, en utilisant les données spécifiées dans le tableau $data. Elle appelle également la méthode "query" de la classe "Database" pour exécuter la requête SQL.

**delete ():** supprime les enregistrements correspondant à l'ID spécifié en utilisant la colonne ID spécifiée (par défaut, c'est "id"). Elle appelle également la méthode "query" de la classe "Database" pour exécuter la requête SQL.

Toutes ces fonctions sont accessibles pour le contrôleur, seulement afin de faciliter leur utilisation et pas ralentir l’avancée du projet, trois autres fichiers ont été créés afin d’offrir aux développeurs exactement la fonction dont ils ont besoin, et donc les classes Forum, Cours et User agissent directement sur les tables correspondantes (topic et message, cours et utilisateur) en utilisant les fonctions mises en place dans la classe Model.

* **La classe Forum :**

Cette classe propose des fonctions afin de lire, ajouter, modifier ou supprimer des éléments dans les tables Topic et Message de notre base de données.

La classe Forum, ayant été créée en parallèle, ne réutilise pas les fonctions de la classe Model. Elle contient directement les requêtes sql à exécuter. Pour éviter de devoir recréer ces requêtes à chaque fois, nous avons fait en sorte de les préparer et de ne modifier que les éléments qui changent d’une requête à l’autre, par exemple si on veut obtenir le topic d’identifiant i, seule la valeur de i devra être préparée au moment de l’appel de la fonction getTopic($id), le reste de la requête étant déjà prêt à être exécuté.

Cependant cette classe montre une grosse limite par rapport aux autres : la facilité d’ajout d’une nouvelle fonctionnalité. En effet, étant donné que pour ajouter une nouvelle fonctionnalité on doit écrire toute la requête, cela demande beaucoup plus d’efforts qu’avec les classes se basant sur la classe Model. Ceci permet donc de mettre en valeur l’utilité de la création de notre classe Model afin de facilité la création de requêtes adaptées.

* **Gestion des QCM :**

Pour modéliser les QCM, nous devions construire un document xml. Ainsi nous avons donc choisis de représenter chaque QCM par un document contenant un type et une suite de questions, chaque question contenant un texte et des propositions avec leur validité ou non. Ainsi un qcm suit le paterne suivant :

<?xml version="1.0" encoding="UTF-8"?>

<qcm>

  <type>Culture</type>

  <question>

    <text>Question</text>

    <choice correct="true">Proposition 1</choice>

    <choice correct="">Proposition 2</choice>

    <choice correct="">Proposition 3</choice>

    <choice correct="">Proposition 4</choice>

  </question>

  <question>

    …

  </question>

  …

</qcm>

L’identification de ces QCM se fait grâce à leur nom. En effet, chaque QCM possède un nom différent, ainsi on peut utiliser cette information pour identifier chaque QCM.

Notre classe QcmModel permet d’extraire, créer ou supprimer un QCM. Pour cela nous utilisons la classe PHP DOMDocument permettant de créer un document xml. Notre classe permet de faire un lien entre une liste contenant toutes les informations de notre QCM et sa version xml.

### Modélisation de la base de données :

Cette modélisation de la base de données a été pensée pour notre site qui permet la création de cours en ligne, la discussion entre utilisateurs et la gestion de la sécurité des données.

**La table "Cours"** : permet de stocker des informations sur les cours en ligne qui sont proposés sur le site web. Les colonnes "titre", "description", "type" et "chemin" permettent de stocker les informations principales sur chaque cours, tandis que la colonne "dateCours" permet de stocker la date de création du cours. Le type a été pensé afin de l’utiliser pour la recommandation du contenue , car le fichier XML des QCM ont aussi un type et dans le cas où la note d’un qcm est inférieur de de la moyenne tous les cours du même type seront recommandés.

**La table "Utilisateur"** permet de stocker les informations sur les utilisateurs du site web. Les colonnes "login" et "mdpHash" stockent les informations de connexion de chaque utilisateur, tandis que les colonnes "nom", "prenom" et "email" permettent de stocker les informations personnelles. La colonne "admin" permet de déterminer si un utilisateur est un administrateur ou non.

**La table "Topic"** permet de stocker des informations sur les sujets de discussion créés par les utilisateurs. Les colonnes "titre", "idUtilisateur" et "nbReponses" permettent de stocker les informations principales sur chaque sujet, tandis que la contrainte de clé étrangère "fk\_idUtilisateur" assure que l'utilisateur qui a créé le sujet existe bien dans la table "Utilisateur" mais aussi cela servirait au moment d’affichage afin de retrouver l’auteur.

**La table "Message"** permet de stocker les messages postés par les utilisateurs dans les sujets de discussion. Les colonnes "idTopic", "idUtilisateur", "contenu" et "dateMessage" permettent de stocker les informations principales sur chaque message, tandis que les contraintes de clé étrangère "fk\_idTopic" et "fk\_idUtilisateur" assurent que le sujet et l'utilisateur associés existent bien dans les tables "Topic" et "Utilisateur".

#### Points forts de cette modélisation de base de données :

Cette modélisation suit les principes de la normalisation, ce qui signifie que les tables sont structurées de manière à réduire la redondance et à faciliter la maintenance. Par exemple, la table Topic contient uniquement les informations relatives à un sujet de discussion, tandis que les messages sont stockés dans la table Message. Cela permet d'optimiser l'utilisation de l'espace de stockage et d'éviter les anomalies de mise à jour.

Les clés étrangères sont utilisées pour établir des relations entre les tables. Cela garantit l'intégrité référentielle, c'est-à-dire que les données dans les tables liées sont cohérentes. Par exemple, la table Message contient une clé étrangère qui référence la table Topic pour garantir que chaque message est associé à un sujet existant.

Cette modélisation peut être performante car elle utilise des index sur les colonnes clés pour accélérer les recherches dans les tables. Les index permettent de réduire le temps de recherche, notamment lorsqu'il s'agit de tables volumineuses.

Cette modélisation utilise une table Utilisateur pour stocker les informations d'identification des utilisateurs, notamment le nom d'utilisateur et le mot de passe haché. Les informations d'identification sont stockées de manière sécurisée, ce qui réduit les risques de piratage.

## Dossier « core »

Ce dossier contient les éléments centraux de l'application, c'est-à-dire les classes et les fonctions qui gèrent la logique de l'application, ainsi il contient aussi les éléments fondamentaux de l'application et est donc essentiel pour le fonctionnement de l'architecture MVC.

**Le fichier  « config.php » :**

Le fichier config.php contient des constantes pour la configuration de l'application web, notamment:

- ROOT: le chemin racine de l'application web

- DBNAME, DBHOST, DBUSER, DBPASS: les informations de connexion à la base de données

- APP\_NAME: le nom de l'application web.

**Le fichier  « functions.php » :**

Ce fichier contient trois fonctions utiles pour le projet mais qui n’ont pas leur place dans les autres fichiers.

**Show() :** cette fonction prend un argument et l'affiche à l'écran en utilisant la fonction `print\_r()` et en encadrant le résultat avec les balises HTML `<pre>`. Elle est utile pour le débogage et l'affichage de tableaux et autres données complexes.

**Skip() :** cette fonction prend une entrée et la nettoie en retirant les espaces blancs en début et fin de chaîne, en retirant les antislashs et en remplaçant les caractères spéciaux par leurs équivalents HTML. Elle est utile pour protéger contre les attaques XSS et CSRF en nettoyant les données d'entrée.

**Redirect() :** cette fonction prend un chemin et redirige l'utilisateur vers cette page en utilisant la fonction `header()`. Elle est utile pour les redirections après un traitement de formulaire ou après une validation.

**Le fichier  « init.php » :**

Le fichier `init.php` a pour rôle d'initialiser l'application en chargeant les différents fichiers nécessaires à son bon fonctionnement.

**spl\_autoload\_register() :** enregistre une fonction qui sera appelée automatiquement par PHP lorsqu'il rencontre une classe qui n'est pas encore définie. Cette fonction prend en paramètre le nom de la classe à charger, en l'occurrence ici, la fonction charge la classe en appelant le fichier correspondant situé dans le répertoire `../app/model/`.

**Les instructions `require`** sont utilisées pour charger les fichiers contenant les constantes de configuration, les fonctions utilitaires, les classes de base, la classe principale du contrôleur et la classe principale du modèle, ainsi que la classe principale de l'application.

En somme, le fichier `init.php` prépare le terrain pour que le reste de l'application puisse être utilisé correctement.

## Dossier « Controller »

## Dossier « Views »

# Conclusion