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

STRAINCHAMPS CLOTHILDE

Table des matières

Introduction et présentation de l'équipe 2
I. Introduction 2
II. Présentation de l'équipe et répartition des tâches 2
1. L'équipe de développement : 2
2. La répartition des tâches : 3
Modélisation de la plateforme 4
I. Présentation du projet 4
II. Analyse des besoins 4
III. Diagramme Use Case 5
Implémentation 6
I. Organisation du site 6
II. Le modèle MVC 6
Architecture et programmation 8
I. Dossier « model » 8
1. Diagramme de classes de l'api base de données 8
2. Explication des classes de l'API : 9
3. Modélisation de la base de données : 11
a. Points forts de cette modélisation de base de données : 12
II. Dossier « core » 13
III. Dossier « controller » 15
1. Représentation graphique de la partie « controller » 15
2. L'interêt du contrôleur 16
3. Users 16
4. Forum 18
5. Lessons 19
6. QCMs 20
7. ViewLauncher 21
IV. Dossier « vues » 22
1. Le sous-dossier « composants » 22
2. Le sous-dossier « pages » 23
3. Charte graphique du site 24
Conclusion 26

INTRODUCTION ET PRESENTATION DE L'EQUIPE

I. 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.

II. PRESENTATION DE L'EQUIPE ET REPARTITION DES TACHES

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.

1. L'équipe de développement :

- SALAH ISMAIL
- BEAUJON ALEXANDRE
- MALOIGNE ANTHONY
- PERE BRANDON
- ROBINET PERRINE
- STRAINCHAMPS CLOTHILDE

2. La répartition des tâches :

- Architecture générale du projet et pensées de ces fonctionnalités : Tout le groupe
- Programmation de la partie Modèle: Ismail / Anthony
- Programmation des Contrôleurs : Alexandre / Brandon
- Programmation des Vues: Clothilde / Perrine / Anthony

MODELISATION DE LA PLATEFORME

I. PRESENTATION 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 obtenus d'un QCM
- Participer dans le forum de discussion entre les apprenants

II. 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.

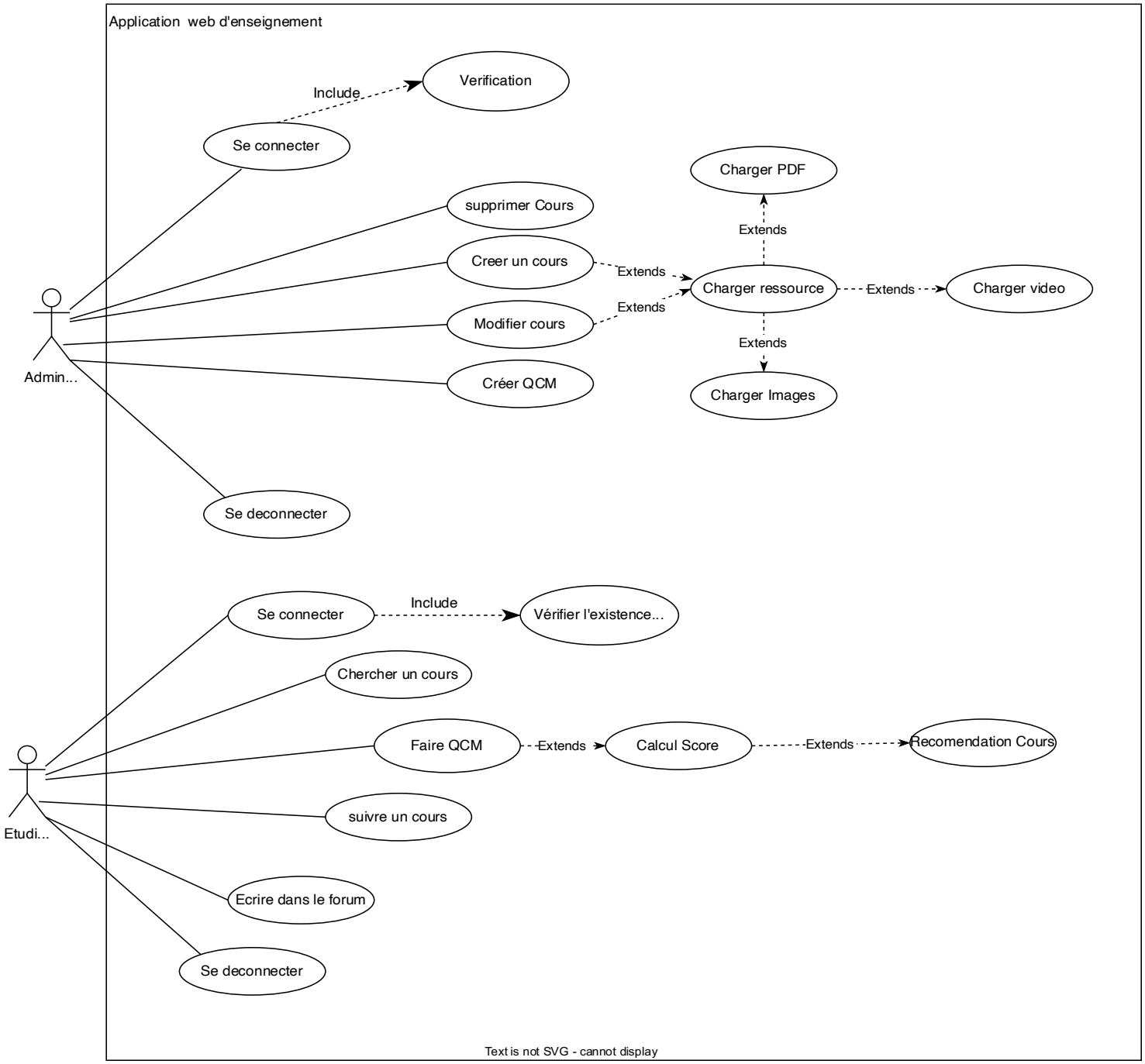
Notre site doit permettre :

- **A l'enseignant de :**
  - **Créer un compte** : sur la page d'accueil ou en passant par l'onglet s'authentifier
  - **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 :**
  - **Créer un compte** : sur la page d'accueil ou en passant par l'onglet s'authentifier
  - **S'authentifier :** il doit s'authentifier pour accéder à son espace privé
  - **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

III. DIAGRAMME USE CASE



IMPLEMENTATION

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.

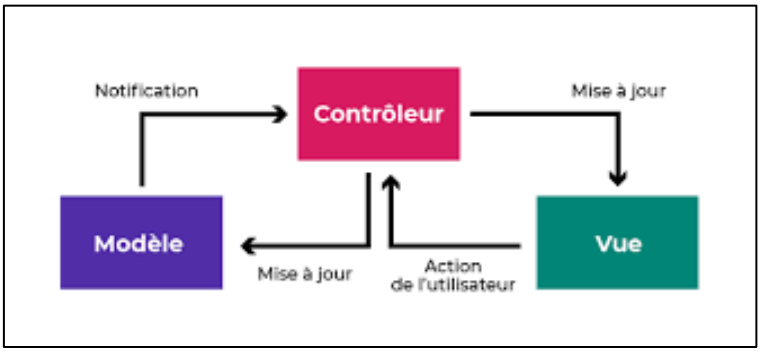
I. ORGANISATION DU SITE

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 ».

II. LE MODELE 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.



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 principalement du code PHP mais aussi 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.

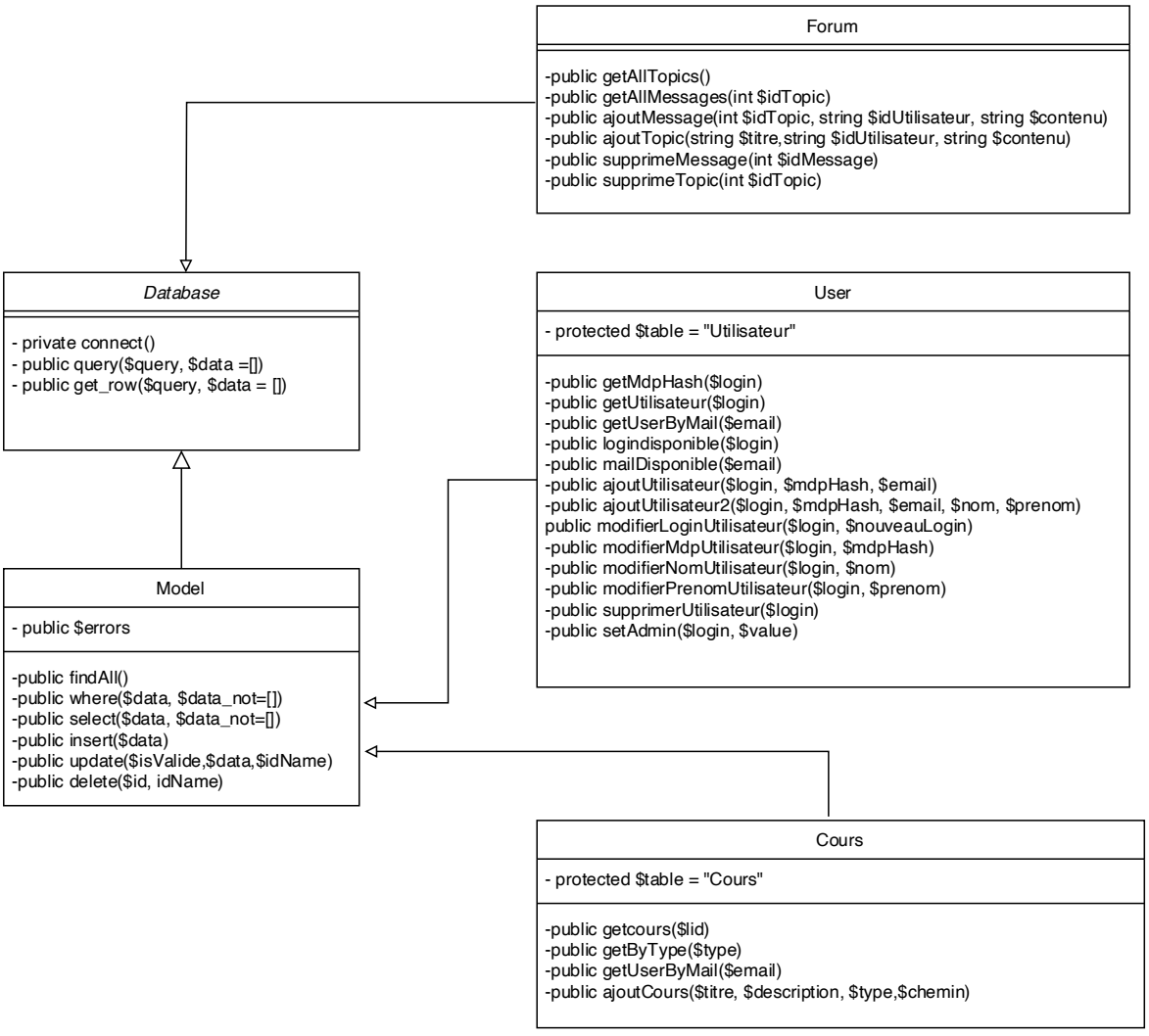
Contrôleur : cette partie gère la logique du code qui prend des *décisions*. C'est en quelque sorte l'intermédiaire entre le modèle et la vue : le contrôleur va demander au modèle les données, les analyser, prendre des décisions et renvoyer le texte à afficher à la vue. Le contrôleur contient exclusivement 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

I. DOSSIER « MODEL »

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

1. Diagramme de classes de l'api base de données



2. 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. 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 faciliter 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.

3. 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_idUtilisateur2" assurent que le sujet et l'utilisateur associés existent bien dans les tables "Topic" et "Utilisateur".

a. 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.

II. 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.

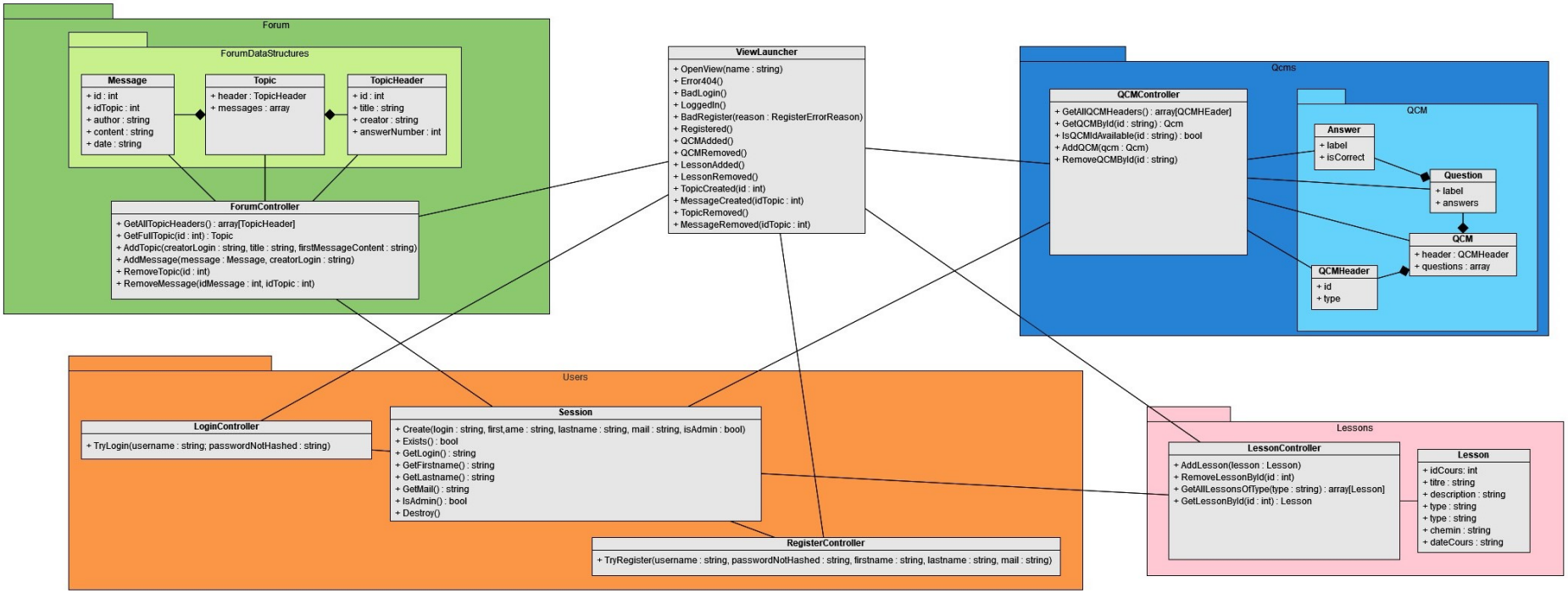
spl_autoload_register() : enregistre une fonction qui sera appelée automatiquement par PHP lorsqu'il rencontre une classe qui n'est pas encore définie. Cette fonction prend en paramètre le nom de la classe à charger, en l'occurrence ici, la fonction charge la classe en appelant le fichier correspondant 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.

III. DOSSIER « CONTROLLER »

1. Représentation graphique de la partie « controller »



2. L'interêt du contrôleur

Le contrôleur est un élément clé du modèle MVC en développement web. Il est chargé de la gestion des requêtes utilisateur et de la coordination entre la vue et le modèle. Plus précisément, le contrôleur reçoit les demandes de l'utilisateur, traite les données associées et décide ensuite de la vue qui doit être affichée.

Le contrôleur effectue des tâches telles que la validation des données entrées par l'utilisateur et la mise à jour du modèle en fonction de ces données. Il est également utilisé pour l'authentification des utilisateurs, la gestion des erreurs et des exceptions.

En utilisant un contrôleur, nous avons pu maintenir une séparation claire entre la logique de présentation de l'interface utilisateur (dans la vue) et la logique métier (dans le modèle). Cela permet une plus grande flexibilité et facilite la maintenance et l'extension de l'application, car chaque composant peut être modifié indépendamment des autres, facilitant ainsi le développement de l'application selon la répartion des tâches que nous avons choisie.

En résumé, le contrôleur est un élément central du modèle MVC qui facilite la gestion des requêtes utilisateur et la coordination entre la vue et le modèle, tout en permettant une plus grande modularité et une plus grande flexibilité dans le développement de l'application.

3. Users

Le partie « Users » permet la gestion de connexion au site, d'inscription et également la gestion des sessions. Il est composé de trois fichiers :

Sessions.php

Le but de ce fichier est de gérer les sessions utilisateur en utilisant la variable $_SESSION en PHP.

Le fichier commence par une instruction de session_start() qui permet de démarrer une session ou de reprendre une session existante. Ensuite, une classe Session est définie qui contient différentes méthodes statiques pour gérer les sessions utilisateur.

La première méthode Create() prend en paramètre les informations de l'utilisateur telles que le login, le prénom, le nom, le mail et le statut d'administrateur, et les stocke dans des variables de session en utilisant la variable $_SESSION en PHP. Cette méthode est utilisée pour créer une nouvelle session utilisateur.

La méthode Exists() vérifie si une session utilisateur est déjà en cours en vérifiant si la variable $_SESSION['login'] est définie. Cette méthode est utilisée pour vérifier si l'utilisateur est déjà connecté ou non.

Les méthodes GetLogin(), GetFirstName(), GetLastName(), GetMail() et IsAdmin() sont utilisées pour récupérer les informations de l'utilisateur stockées dans la variable de session $_SESSION. Si la session n'existe pas, ces méthodes lèvent une exception pour indiquer que la récupération des informations est impossible.

Enfin, la méthode Destroy() est utilisée pour détruire la session de l'utilisateur en utilisant la fonction session_destroy() en PHP. Cette méthode est appelée lorsque l'utilisateur se déconnecte du site.

En somme, ce fichier est un composant essentiel pour gérer les sessions utilisateur et sécuriser le site web en vérifiant l'authentification de l'utilisateur et en permettant d'accéder aux informations stockées dans la session.

LoginController.php

Ce fichier est un contrôleur pour une page de connexion au site.

Dans un premier temps, le code vérifie si les champs "username" et "passwordNotHashed" ont été passés en POST, ce qui indique que l'utilisateur a soumis un formulaire de connexion. Si c'est le cas, la fonction "TryLogin" du contrôleur est appelée, passant les informations d'identification saisies.

La fonction "TryLogin" essaie de connecter l'utilisateur en vérifiant si le nom d'utilisateur existe et si le mot de passe fourni correspond au mot de passe enregistré pour cet utilisateur. Si les informations d'identification sont incorrectes, la fonction "BadLogin" du "ViewLauncher" est appelée pour lancer la vue adaptée. Si les informations d'identification sont correctes, la fonction "LoggedIn" du "ViewLauncher" est appelée pour rediriger l'utilisateur vers une vue de connexion réussie et une session est créée pour cet utilisateur avec les informations de l'utilisateur nécessaires stockées en utilisant la classe "Session".

En résumé, LoginController.php définit un contrôleur qui gère le processus de connexion pour les utilisateurs d'un site web en vérifiant les informations d'identification soumises, en créant une session pour l'utilisateur connecté et en redirigeant l'utilisateur vers une page appropriée en fonction du résultat de la tentative de connexion.

RegisterController.php

Ce fichier contient le contrôleur de la page d'inscription. Il permet à un utilisateur de s'inscrire en fournissant son nom d'utilisateur, son mot de passe, son prénom, son nom de famille et son adresse e-mail.

On commence par vérifier si les paramètres requis sont présents dans la requête POST (nom d'utilisateur, mot de passe, prénom, nom de famille et adresse mail). Si tel est le cas, la fonction TryRegister du contrôleur est appelée, prenant en charge le traitement de la demande.

La fonction TryRegister vérifie si le nom d'utilisateur n'est pas déjà utilisé en interrogeant la base de données via l'instance User du modèle. Si le nom d'utilisateur est déjà pris, une vue d'erreur est renvoyée. Si le mot de passe est jugé trop faible, une autre vue d'erreur est retournée. Dans le cas contraire, le mot de passe est haché et les informations de l'utilisateur sont ajoutées à la base de données via les différentes fonctions fournies par l'instance User.

Enfin, une session est créée pour l'utilisateur nouvellement inscrit et une vue de confirmation est retournée pour indiquer que l'inscription a réussi. Si une erreur se produit lors de la création d'un utilisateur ou de la modification de ses informations, une vue d'erreur générique est retournée pour indiquer que l'inscription a échoué.

4. Forum

La partie « Forum » fait le lien entre les fonctions du modèle de notre forum avec les vues associées.

ForumDataStructures.php

Ce fichier contient la définition de trois classes qui sont utiles pour la gestion de messages et de sujets de discussion. La première classe, appelée "Message", contient les propriétés d'un message, telles que son identifiant, l'identifiant du sujet associé, l'auteur, le contenu et la date de création. La deuxième classe, appelée "TopicHeader", représente les informations générales d'un sujet, telles que son identifiant, son titre, le nom de son créateur et le nombre de réponses qu'il a reçues. Enfin, la troisième classe, "Topic", contient une instance de la classe "TopicHeader" et un tableau de messages associés à ce sujet. Ces classes fournissent une structure utile pour stocker les données des messages et des sujets dans un forum et permettent une manipulation efficace des données associées à ces derniers.

ForumController.php

Ce fichier correspond au contrôleur de la partie Forum de notre site. Au chargement de la page, la variable POST "whatToDo" est vérifiée pour appeler les fonctions demandées.

La classe ForumController contient deux fonctions principales:

- La fonction GetAllTopicHeaders() qui retourne tous les en-têtes de topic du forum en créant un objet Forum du modèle, en appelant la méthode getAllTopics(), en créant un objet User du modèle pour chaque topic brut et en récupérant les données nécessaires à la création de l'en-tête de chaque topic dans un tableau d'instances de TopicHeader.
- La fonction GetFullTopic() qui retourne un topic complet à partir de son id en créant un objet Forum du modèle, en appelant la méthode getTopic() avec l'id du topic, en créant un objet User du modèle et en récupérant les données nécessaires à la création de l'en-tête du topic et de tous ses messages dans un objet Topic.

Nous avons également les fonctions "AddTopic", "AddMessage", "RemoveTopic" et "RemoveMessage" qui peuvent être également appelées selon la valeur de la variable POST "whatToDo". Elles prennent des paramètres supplémentaires obligatoires passés en POST, tels que le login du créateur, le titre du topic, le contenu du premier message, l'id du topic ou l'id du message à supprimer. Si un ou plusieurs paramètre(s) POST sont inexistant(s), une exception est lancée pour indiquer que l'opération est impossible.

5. Lessons

La partie « Lessons » fait le lien entre les fonctions du modèle, de gestion des cours, avec les vues associées.

Lesson.php

Le fichier Lesson.php contient la classe Lesson, qui définit les propriétés d'un cours en ligne, telles que l'identifiant du cours, le titre, la description, le type, le chemin d'accès et la date du cours. Ces propriétés sont définies comme étant publiques, ce qui signifie qu'elles peuvent être accessibles et modifiées depuis n'importe où dans le code. La classe Lesson peut être utilisée pour créer des instances de cours en ligne, ce qui facilite la gestion des données et la manipulation des cours.

LessonController.php

Le fichier LessonController.php est utilisé pour manipuler les informations de cours, que ce soit pour ajouter, supprimer ou récupérer des cours.

Le fichier commence par vérifier si une action a été demandée via la méthode POST et la variable 'whatToDo'. Si une action est demandée, le code exécute la fonction correspondante en fonction de la valeur de 'whatToDo'. Le fichier contient quatres fonctions principales : AddLesson, RemoveLessonById, GetAllLessonsOfType et GetLessonById.

La fonction AddLesson permet d'ajouter un cours à la base de données en prenant une instance de la classe Lesson en tant que paramètre. Avant de procéder à l'ajout, la fonction vérifie si l'utilisateur est connecté et a les droits d'administrateur. Si les vérifications sont correctes, la fonction appelle une fonction d'un modèle appelé Cours pour effectuer l'ajout en base de données. En cas de réussite, une vue est chargée.

La fonction RemoveLessonById permet de supprimer un cours de la base de données en utilisant l'ID du cours en tant que paramètre. Cette fonction utilise également le modèle Cours pour effectuer la suppression. Après la suppression, une vue est chargée.

La fonction GetAllLessonsOfType récupère tous les cours du type spécifié en paramètre en utilisant également le modèle Cours pour effectuer la requête en base de données. La fonction crée ensuite une instance de la classe Lesson pour chaque cours retourné et les ajoute à un tableau qui est finalement retourné. Cette fonction est utilisée pour afficher la liste des cours d'un type spécifique dans l'interface utilisateur.

La fonction GetLessonById permet de récupérer un cours en particulier dans la base de données en spécifiant son ID en paramètre. Cette fonction retourne une instance de la classe Si le cours est trouvé, une instance de la classe Lesson est créée et initialisée avec les informations du cours récupérées depuis le modèle, sinon elle retourne null.

6. QCMs

La partie « QCMs » fait le lien entre les fonctions du modèle, de gestion des qcms, avec les vues associées.

QCM.php

Le fichier QCM.php contient la définition de la classe QCM, qui permet de stocker les informations relatives à un questionnaire à choix multiples. Cette classe contient deux propriétés : une instance de QCMHeader et un tableau de questions. La classe QCMHeader représente l'en-tête d'un questionnaire et contient l'identifiant unique du questionnaire ainsi que son type. La classe Question représente une question et contient une chaîne de caractères correspondant à l'énoncé de la question et un tableau de réponses possibles, représentées par des instances de la classe Answer. La classe Answer représente une réponse possible et contient une chaîne de caractères correspondant à l'intitulé de la réponse ainsi qu'un booléen indiquant si cette réponse est la bonne réponse à la question ou non.

L'utilisation de cette classe permet de faciliter la gestion des données et la manipulation des qcms.

QCMController.php

Ce fichier définit plusieurs fonctions pour manipuler les données des QCM stockés sur le site.

Le contrôleur gère les endpoints qui sont envoyés en POST en utilisant la variable 'whatToDo'. Si la variable 'whatToDo' existe, le contrôleur utilise une structure de contrôle de flux (switch) pour appeler la fonction associée à cet endpoint. Il y a deux endpoints disponibles : 'addQCM' et 'removeQCM', qui permettent d'ajouter ou de supprimer un QCM. Les paramètres pour ces fonctions doivent être passés en POST.

Ensuite, le contrôleur a plusieurs fonctions pour interagir avec le modèle. La fonction GetAllQCMHeaders() récupère tous les identifiants et types de QCM stockés sur le site. La fonction QetQCMById() permet de récupérer un QCM à partir de son identifiant. La fonction IsQCMIdAvailable() permet de savoir si un identifiant de QCM est disponible ou non.

7. ViewLauncher

Le fichier ViewLauncher.php contient une classe qui permet de gérer les vues de l'application. Cette classe propose des méthodes statiques qui permettent d'ouvrir différentes vues en fonction des événements qui se produisent dans l'application, tels que la connexion ou l'inscription d'un utilisateur, l'ajout ou la suppression d'un QCM ou d'un cours, ou encore la création ou la suppression d'un topic ou d'un message sur le forum. En outre, cette classe contient une énumération qui permet de spécifier les raisons pour lesquelles une inscription peut échouer. Ce fichier est donc un élément clé pour la gestion des vues et des redirections dans l'application.

IV. DOSSIER « VUES »

Le dossier « Vues » est composé de deux sous dossiers, un nommé « composants » et un « pages » avec pour chacun des sous dossiers contenant les éléments associés à leur nom et ce dont ils ont besoins comme ressources.

Pour notre site web nous avons opté pour des vues au design simple.

1. Le sous-dossier « composants »

Nos vues étant composées d'éléments reproductibles, comme les parties HTML tel que le head, le header et le footer, nous avons décidé de créer des fonctions qui nous renverrons sur chacune de nos vues quand nous les appellerons.

Dans le dossier header nous pouvons trouver le fichier header.php où nous trouverons les implémentations de nos fonctions *head()* et *navBar()*.

Pour commencer la fonction *head()* prend en paramètres le titre de notre page courante et le lien vers son fichier de style CSS. Cette fonction nous affichera du code HTML quand nous ferons nos pages de vues.

La fonction *navBar()* prend en paramètre un booléen ce qui nous permet d'adapter la barre de navigation selon si l'utilisateur est connecté ou non. Elle est composée d'une fonction anonyme *connecter* qui vérifiera si l'utilisateur est connecté, professeur, élève et affichera en fonction de ces cas différents éléments dans notre menu de navigation. → Si l'utilisateur n'est pas connecté seul les liens *'Se connecter'* et 'S'inscrire' seront afficher. → Si l'utilisateur est connecté il pourra alors voir les différentes parties de notre site et les liens *'QCM'*, *'Forum'*, *'Cours'* et *'Se déconnecter'* seront alors affichés. Cette fonction nous affichera du code HTML quand nous ferons nos pages de vues.

Ensuite viens le fichier de style header.css qui définit le style de notre navBar. Nous avons fait une navbar responsive ce qui veut dire que si la page de navigateur de l'utilisateur descend en dessous de 900 pixels de largeur celle-ci prendra une forme différente. Elle se composera d'un « menu hamburger ».

Le fichier header.js nous servira à afficher notre « menu hamburger » lors du clic sur l'icône associé.

Maintenant dans le dossier footer nous pouvons trouver le fichier footer.php où nous trouverons l'implémentations de notre fonction *footer()* qui nous servira à afficher notre footer sur chacune de nos pages.

Il possède également son fichier de style qui est footer.css.

2. Le sous-dossier « pages »

Nous y trouverons nos différentes pages qui seront affichées à l'utilisateur. Dans le même esprit que la navbar les pages auront deux versions différentes mais définit dans la même page. PHP nous permettra de gérer quelle version nous afficherons, soit l'une (l'élève), soit l'autre (l'enseignant).

Nous allons maintenant définir le contenu de chaque page.

La première est la page d'accueil du site, notre site est un moodle. La première page est relativement simpliste et nous donne juste une indication sur le but de notre site. Lors de l'arrivée de l'utilisateur sur notre site, il n'a qu'une action réalisable qui est de se connecter ou bien de s'inscrire pour pouvoir découvrir le reste de notre moodle.

Vient ensuite les vues nécessaires à la connexion et l'inscription de l'utilisateur sur notre site. Il peut y accéder en cliquant sur le lien « se connecter » présent dans la nav bar. La partie vue de la connexion est gérer par le fichier 'connexion.php'. Elle est principalement composée d'un formulaire demandant à l'utilisateur son login et son mot de passe. Les informations entrées par l'utilisateur seront dès un clic sur le bouton 'connexion' envoyées au contrôleur avec la méthode *POST.* Ensuite si les informations sont validées par le contrôleur, sa session commencera et il sera rediriger sur la page d'accueil.

Même chose pour l'inscription de l'utilisateur de notre site, pour accéder à cette page il peut cliquer sur le lien 's'inscrire' dans la nav bar. De plus un lien est fait entre les pages d'inscription et de connexion. La partie vue de l'inscription est gérée par le fichier 'inscription.php'. Elle est principalement composée d'un formulaire demandant à l'utilisateur son login, nom, prénom, mot de passe et email. Les informations entrées par l'utilisateur seront dès un clic sur le bouton 'inscription' envoyer au contrôleur avec la méthode *POST.* Ensuite si les informations sont validées par le contrôleur, sa session commencera et il sera rediriger sur la page d'accueil.

Les pages d'accueil, de connexion et d'inscription sont les seules où l'affichage final est commun aux différents types d'utilisateurs.

Une fois redirigé vers la page d'accueil, l'utilisateur pourras naviguer dans notre site grâce à la barre de navigation. Il peut alors aller voir et faire les cours et qcm postés par l'enseignant du point de vue de l'élève et voir, éditer et modifier les cours et qcm du point de vue de l'enseignant. Les deux types d'utilisateur pourront également poster des topics et des réponses à ces topics sur le forum. Les enseignants auront la possibilité de supprimer des topics ou des réponses afin de pouvoir modérer le forum.

Pour les QCM, cours et forum, on utilise les mêmes vues pour les élèves et les enseignants et on se sert de l'attribut admin associé à la session pour modifier ce qui va s'afficher.

La partie qcm est composé de deux vues, la première est une liste les qcm disponibles qui est défini dans le fichier 'qcm.php' et la deuxième est le qcm en lui-même qui est défini dans le fichier 'vueQcm.php'. Pour la version élève, en cliquant sur un des sujets de qcm l'utilisateur sera redirigé vers la liste des questions du qcm. Ensuite l'élève fait le qcm et il sera vérifié par le contrôleur. Pour la version enseignante, sur la page de liste des qcm, l'enseignant pourra ajouter ou supprimer un qcm. La liste des qcms sera afficher de manière automatique sous

forme de liste numéroté en demandant au contrôleur quels qcm sont disponibles. Puis les questions de qcm seront récupérées grâce à l'URL.

La partie cours est composé de deux vues, la première est une liste des sujets de cours disponibles et est défini dans le fichier 'cours.php'. Puis dans la deuxième vue, qui est défini dans le fichier 'vueCours.php, on trouve les PDF, vidéos... du cours. Pour la version élève, en cliquant sur un des sujets de cours l'utilisateur sera redirigé vers la deuxième pages. Ensuite l'élève peut ouvrir les pdf grâce à l'ouverture pdf disponible avec son navigateur et pour les vidéos c'est également le lecteur par défaut de son navigateur qui prends le relais. Pour la version enseignante, sur la page de liste des sujets des cours, l'enseignant pourra ajouter ou supprimer un sujet de cours. La liste des cours sera afficher de manière automatique sous forme de liste numéroté en demandant au contrôleur quels sont les sujets de cours disponibles.

La partie forum est également décomposé en deux vues, une où les topics de discussion sont listés, défini dans le fichier 'forum.php', puis la deuxième affiche la discussion en rapport avec le topic choisi par l'utilisateur, défini dans le fichier 'topic.php'.

Un mode sombre est également disponible pour l'application.

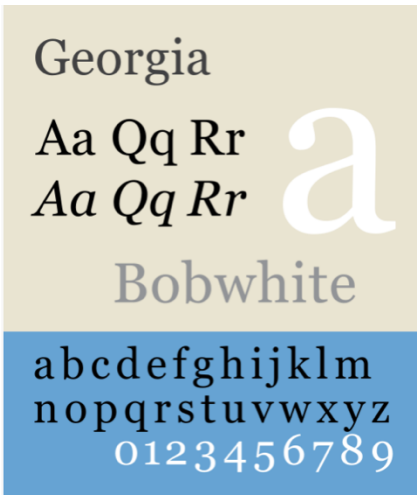
3. Charte graphique du site

Notre palette de couleur est composée essentiellement d'un dégrader d'orange pour les fonds des éléments comme la nav bar ou bien des différents formulaires, et de gris qui sont majoritairement utilisé pour les bordures, les écritures seront en noir et blanc.

L'orange a été choisie car c'est une couleur qui facilite la vitalité, le bonheur et l'apaisement. Pour un site de formation c'est important de se sentir heureux et apaisé pour pouvoir se

concentrer sur des cours. Elle attire également vite l'œil. De plus beaucoup de moodle sont fait dans ces teintes.

Nous avons harmonisé le style d'écriture de notre site en style Georgia, car elle était simple et sérieuse pour un site de formation.



Le fond de notre site web sera blanc en mode clair pour faciliter la lecture de l'utilisateur.

CONCLUSION

Notre équipe a travaillé dur pour réaliser ce projet de site web en architecture MVC pour une plateforme de formation, dans le cadre de la Licence d'informatique. Grâce à notre collaboration, nous avons réussi à concevoir et à développer une plateforme fonctionnelle, qui reflète les fonctionnalités pensées au départ et exigées par le sujet. Le développement de ce projet nous a permis de mieux comprendre l'importance du travail d'équipe et de la communication dans un projet informatique. Nous avons également amélioré nos compétences en programmation, en conception de bases de données et en gestion de projet. Nous sommes convaincus que ce projet peut constituer une base solide pour des améliorations futures et nous sommes fiers de ce que nous avons accompli en tant qu'équipe.