

Développement d'Applications Web

# Rapport de projet

## « Création de site de formation »

---



<b>Équipe :</b>
<b>STIZ Romain, POUZIN Pierre-Emmanuel, THOMAS Allan, GIE Jimmy, VOUETTE Maxence, RAJILLAH Abderrahmane</b>

## Table des matières

<b>Introduction.....</b>	<b>3</b>
Architecture MVC (Model View Controller).....	3
Base de données (MariaDB).....	4
<b>I) Authentification.....</b>	<b>5</b>
<b>II) Partie administrateur.....</b>	<b>6</b>
Gestion des utilisateurs.....	6
Gestion des cours.....	9
<b>III) Partie apprenant.....</b>	<b>11</b>
Construction du profil de l'apprenant.....	11
Recommandation de cours.....	12
Forum de discussion entre les apprenants.....	13
<b>Conclusion.....</b>	<b>15</b>

# Introduction

Ce rapport, structuré en parties distinctes, présentera le projet que nous avons dû réaliser dans le cadre de cette unité d'enseignement. Ce dernier consiste en la réalisation d'un **site de formation** en ligne à destination d'étudiants. l'application est composée de deux parties :

- Une partie **administrateur**, permettant de charger les cours sous différents formats, de gérer les utilisateurs et les QCM.
- Une partie **apprenant**, intégrant un espace personnel et de gestion de cours, ainsi qu'un forum de discussion.

Ce compte-rendu détaillera les fonctionnalités implémentées dans chacune des deux parties décrites ci-dessus, en suivant les nombreuses contraintes imposées. Parmi elles, la nécessité de réaliser l'application avec une **architecture MVC**, l'obligation d'utiliser des **sessions** ou des **cookies**, l'implémentation de deux **présentations graphiques** différentes, et bien d'autres encore que nous préciserons par la suite.

## Architecture MVC (Model View Controller)

Avant d'entrer dans le vif du sujet, il est important de définir la notion d'**architecture MVC**. En effet, ce type de structure est utilisé pour chacune des fonctionnalités implémentées, il est donc nécessaire d'en préciser la nature.

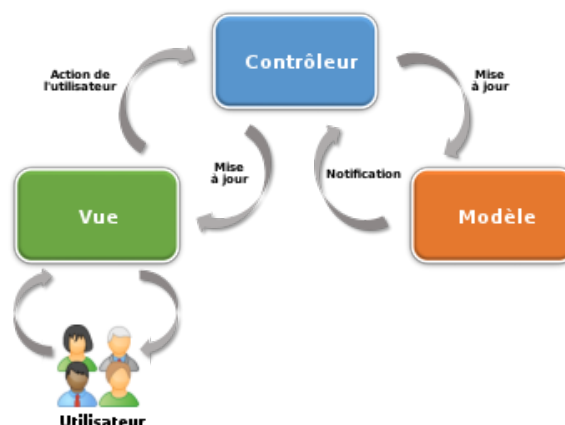


Figure 1 : Interactions entre le modèle, la vue et le contrôleur (wikipédia)

**MVC** est un motif d'**architecture logicielle** destiné aux interfaces graphiques, lancé en 1978 et très populaire pour les **applications web**. Le motif est composé de trois types de modules ayant trois responsabilités différentes : les modèles, les vues et les contrôleurs.

- Un modèle contient les **données** ainsi que la structure en rapport avec ces dernières.
- Une vue contient la présentation de l'**interface graphique**.
- Un contrôleur **traite** les actions de l'utilisateur, et **modifie** les données du modèle et de la vue.

Cette architecture permet d'établir des **rôles** et des limites claires entre les différents programmes qui composent l'application, ce qui facilite la **maintenance** générale du code et sa **lisibilité, ainsi que l'évolutivité de notre site de formation en ligne**. C'est donc celle-ci que nous avons utilisée.

## Base de données (MariaDB)

Afin de conserver les données de nos utilisateurs, nous utilisons un SGBD (Système de Gestion de Base de Données) sur un serveur MariaDB. Nous en définissons la structure à l'aide du fichier **projet.sql**, dans lequel nous créons les différentes tables et réalisons les insertions nécessaires au bon fonctionnement de l'application. Le contenu des tables peut être visualisé à tout moment dans **phpmyadmin**.

## I) Authentification

La première page à s'exécuter lorsque nous arrivons sur l'application est **index.php**. En effet, elle affiche une page de connexion et nous propose de rentrer nos identifiants (nom, prénom et mot de passe). Le bouton « se connecter » du formulaire de connexion exécute alors une fonction javascript présente dans **page\_connexion.js**, qui récupère les informations et les enregistre dans les cookies du navigateur.

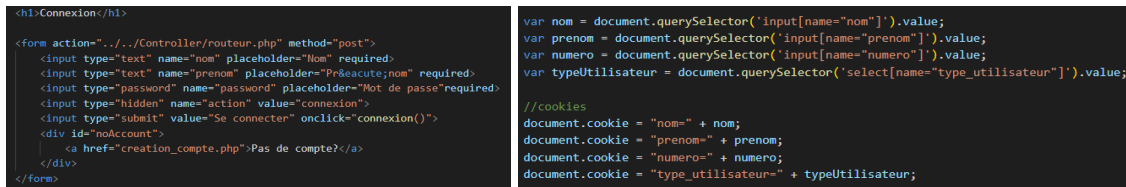


Figure 2 et 3 : formulaire de connexion à gauche, corps de la fonction javascript à droite

Le formulaire est ensuite envoyé au **routeur.php**, le programme qui permet d'appeler différentes méthodes en fonction de l'action reçue. Dans notre cas, le type d'action envoyée est « connexion » : le routeur appelle alors la méthode « connexion() » de **Contrôleur\_Utilisateur.php** (Figure 4). Cette dernière, conformément au modèle MVC, effectue des vérifications et appelle la méthode « connexion() » de **modele\_Utilisateur.php**, qui vérifie si les identifiants saisis sont bien présents dans la base de données à l'aide de requête SQL et les enregistre dans des variables de session (Figure 5), ou renvoie un message d'erreur en cas d'anomalie (mot de passe incorrect, nom d'utilisateur incorrect, etc...). Si les identifiants saisis sont corrects, l'utilisateur sera redirigé vers **page\_accueil.php**, qui récupérera les identifiants et le rôle de compte (utilisateur ou administrateur) par l'intermédiaire des variables de session, et différents accès à l'application lui seront attribués (Figure 6).

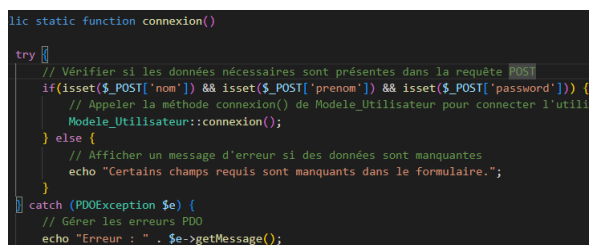


Figure 4

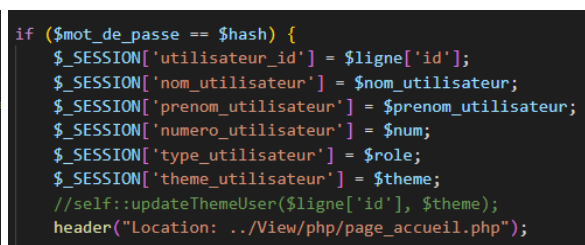


Figure 5

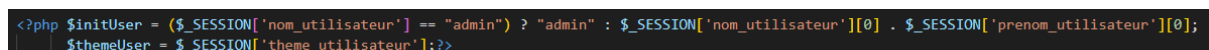


Figure 6

## II) Partie administrateur

Dans cette section, deux fonctionnalités seront décrites : celle permettant de gérer les utilisateurs de l'application, et celle permettant d'ajouter, supprimer ou modifier des cours. Seuls les administrateurs ont accès à ces fonctionnalités. Les apprenants ne peuvent que consulter les cours mis en ligne.

### Gestion des utilisateurs

La gestion des utilisateurs est une composante essentielle de notre application, permettant de gérer les profils des apprenants ainsi que des administrateurs. Le contrôleur **Contrôleur\_Utilisateur** joue un rôle central dans cette fonctionnalité en interagissant avec le modèle **modele\_Utilisateur** pour effectuer diverses opérations sur les utilisateurs. Voici un aperçu des fonctionnalités implémentés dans cette section.

Sur la page d'accueil, l'administrateur a la possibilité d'aller dans les **paramètres de l'application** en se rendant dans son tableau de bord : en cliquant sur **admin** en haut à gauche puis sur **paramètres**. La possibilité de passer du thème clair au thème sombre est également présente ici.



Comme nous pouvons le voir dans l'extrait de code ci-dessous, un lien présent sur **paramètres** redirige vers **page\_gestion.php**, que nous allons expliciter par la suite.

```
<button class="dropdown" onclick="dropdown(event)"></button>
<div class="dropdown-content" id="myDropdown">
  <span id="spanDropTop">Tableau de bord</span>
  <hr><a href="page_gestion.php"><i class="fa-solid fa-wrench fa-lg"></i><span class="spanDrop">Paramètres</span></a>
  <hr><form action=".../Controller/routeur.php" method="post"><a onclick="switchThemeLN()" id="switchTheme" name="action" value="switchTheme">
  <hr><form action=".../Controller/routeur.php" method="post">
    <button type="submit" id="loginButton" name="action" value="deconnexion"><i class="fa-solid fa-power-off fa-lg"></i></button>
```

Figure 7 : extrait du contenu de *page\_accueil.php*

Cette page permet donc de visualiser les comptes des utilisateurs de la plateforme, et de les supprimer. Elle permet également de changer son mot de passe. Pour ce faire, **page\_gestion.php** vérifie dans un premier temps si le type de compte est bien un

administrateur en récupérant la variable de session enregistrée dans `$_SESSION['type_utilisateur']` (Figure 8). Dans ce cas, la méthode `Controleur_Utilisateur::readAll()` est appelée (Figure 9). Cette dernière fait ensuite appel à `modele_Utilisateur::readAll()`, et tous les comptes de la plateforme sont affichés à l'écran. La méthode `readAll()` du modèle récupère tous les utilisateurs par une requête SQL dans la table `Utilisateurs`, et enregistre le retour dans une variable (Figure 10). A l'aide de la méthode `fetch()`, la boucle `while` récupère les valeurs dans une variable temporaire, et en affiche le contenu (Figure 11).

```
<?php if(isset($_SESSION['type_utilisateur']) && $_SESSION['type_utilisateur'] == 'Professeur'){
    Controleur_Utilisateur::readAll();
}??
```

Figure 8

```
public static function readAll()
{
    try {
        // Appeler la méthode readAll() de Modele_Utilisateur
        Modele_Utilisateur::readAll();
    } catch (PDOException $e) {
        // Gérer les erreurs PDO
        echo "Erreur : " . $e->getMessage();
    }
}
```

Figure 9

```
public static function readAll(){
    try{
        $sql = "SELECT * FROM Utilisateur";
        $conn = Model::$pdo;
        $rep = $conn->query($sql);
        $rep->setFetchMode(PDO::FETCH_ASSOC);
    }
```

Figure 10

```
echo "<ul>";
while ($row = $rep->fetch()) {
    echo "<li>";
    echo "ID: " . $row['id'] . ", ";
    echo "Nom: " . $row['Nom'] . ", ";
    echo "Prénom: " . $row['Prenom'] . ", ";
    echo '<form action="../../Controller/routeur.php" method="post">';
    echo '<input type="hidden" name="id" value="'. $row['id'] .'">';
    echo '<button type="submit" name="action" value="supprimer_utilisateur">Supprimer</button>';
    echo "</form>";
    echo "</li>";
}
echo "</ul>";
```

Figure 11

Comme expliqué précédemment, cette page de gestion permet également de modifier son mot de passe. En effet, l'administrateur rentre son nouveau mot de passe dans un formulaire qui sera par la suite envoyé au `routeur.php`. Le routeur traitera l'action reçue, qui est dans ce cas "modif" — routeur.php fonctionne au moyen d'un switch/case, pour rediriger le trafic —, et appellera la méthode associée `controleur_Utilisateur::modifier()`, qui appellera elle-même la méthode `modele_Utilisateur::modifier()` conformément au modèle MVC. Celle-ci vérifie s'il existe bien une session utilisateur active, et récupère les nouveaux

identifiants de l'utilisateur à l'aide de la variable POST du formulaire. Après cela, elle exécute une requête SQL de type « UPDATE » avec les nouvelles informations dans la base de données, avant de rediriger vers une nouvelle page après le succès de l'opération.

### Modification des informations

La méthode `modifier()` permet aux utilisateurs de modifier leurs informations personnelles telles que leur nom, prénom, numéro de téléphone et mot de passe. Cette méthode vérifie d'abord la présence des données nécessaires dans la requête POST, puis appelle la méthode `modifier()` du modèle pour effectuer la modification.

### Affichage de tous les utilisateurs

La méthode `readAll()` récupère et affiche la liste de tous les utilisateurs enregistrés dans la base de données. Elle appelle la méthode `readAll()` du modèle pour récupérer les données, et gère les erreurs PDO le cas échéant.



Nom de famille	Prénom	ID	Action
admin	admin	5	Supprimer
Dumont	Jean	6	Supprimer
STIZ	Romain	13	Supprimer
THOMAS	Allan	165489	Supprimer

### Ajout d'un nouvel utilisateur

La méthode `add()` permet aux administrateurs d'ajouter de nouveaux utilisateurs à la plateforme. Elle vérifie d'abord la présence de toutes les données nécessaires dans la requête POST, puis appelle la méthode `add()` du modèle pour ajouter le nouvel utilisateur.

### Suppression d'un utilisateur

La méthode `supprimer()` permet aux administrateurs de supprimer un utilisateur existant de la plateforme. Elle vérifie d'abord la présence de l'identifiant de l'utilisateur à



supprimer dans la requête POST, puis appelle la méthode `supprimer()` du modèle pour effectuer la suppression.

### Connexion et déconnexion des utilisateurs

Les méthodes `connexion()` et `deconnexion()` gèrent respectivement la connexion et la déconnexion des utilisateurs. La méthode `connexion()` vérifie d'abord la présence des données d'identification dans la requête POST, puis appelle la méthode `connexion()` du modèle pour vérifier les informations d'identification et connecte l'utilisateur. La méthode `deconnexion()` quant à elle, supprime simplement la session de l'utilisateur.

## Gestion des cours

La gestion des cours constitue une fonctionnalité cruciale de notre application. Cette partie de l'application permet aux administrateurs de charger, organiser et gérer les différents cours proposés aux apprenants. Cette partie de notre système est gérée par le contrôleur **`controleur_Cours`**, qui interagit avec le modèle **`model_Cours`** pour effectuer diverses opérations sur les cours. Voici une aperçu des fonctionnalités mises en oeuvres dans cette section:

### Ajout de matières

La méthode `ajouterMatiere()` permet aux administrateurs d'ajouter de nouvelles matières à la plateforme. Lorsqu'un administrateur soumet un formulaire contenant le nom d'une nouvelle matière, cette méthode vérifie d'abord si la matière existe déjà dans la base de données. Si elle n'existe pas, la méthode crée un nouveau dossier pour la matière dans le répertoire de stockage des cours et enregistre la matière dans la base de données. En revanche, si la matière existe déjà, un message d'erreur est renvoyé.

### Suppression de matières

La méthode `supprimerMatiere($matiere)` permet aux administrateurs de supprimer une matière existante de la plateforme. Cette méthode supprime d'abord le dossier correspondant

à la matière, ainsi que tous ses fichiers associés, puis supprime l'enregistrement de la matière dans la base de données.

### **Chargement de fichiers**

La méthode `chargerFichiers($matiere)` récupère et affiche la liste des fichiers associés à une matière donnée. Les fichiers peuvent être de différents formats tels que PDF, PowerPoint, ou vidéo. Les liens vers ces fichiers sont générés dynamiquement et permettent aux utilisateurs d'accéder directement au contenu des cours.

### **Suppression de fichiers**

La méthode `supprimerFichier($matiere, $fichier)` permet aux administrateurs de supprimer un fichier spécifique associé à une matière donnée. Cette méthode supprime le fichier du répertoire de stockage correspondant à la matière et met à jour la base de données pour refléter ce changement.

### **Importation de fichiers**

La méthode `importerFichier($matiere, $fichierImporte)` permet aux administrateurs d'importer de nouveaux fichiers dans une matière existante. Cette méthode enregistre le fichier dans le répertoire de stockage correspondant à la matière et met à jour la base de données pour inclure le nouveau fichier.

La méthode `getListeCours()` récupère et affiche la liste de toutes les matières disponibles sur la plateforme, avec des liens permettant aux utilisateurs d'accéder aux cours associées à chaque matière.

### III) Partie apprenant

Dans notre application, l'espace personnel des apprenants joue un rôle essentiel, offrant une interface sympathique pour gérer les cours et les interactions au sein de la plateforme d'apprentissage.

#### Construction du profil de l'apprenant

La construction du profil de l'apprenant débute par l'évaluation de son niveau et de ses compétences. Pour ce faire, des questionnaires à choix multiples (QCM) sont proposés à l'apprenant afin de définir son niveau dans différents domaines d'études. Voici comment chaque méthode du modèle `Modele_Quiz` contribue à cette fonctionnalité :

- Lecture de tous les quiz (`readAll_Admin` et `readAll`):
  - `readAll_Admin()` : charge et affiche tous les quiz à partir d'un fichier XML.
  - `readAll()` : fonction similaire à `readAll_Admin`, mais conçue pour une utilisation par les apprenants.
  - Pour `readAll_Admin()`, affiche également un formulaire pour supprimer un quiz si nécessaire.
- Lecture d'un seul quiz (`readQuiz`):
  - Affiche un seul quiz avec toutes ses questions et réponses possibles.
  - Permet à l'utilisateur de sélectionner des réponses et de soumettre le quiz.
  - Calcule le score en fonction des réponses soumises.
  - Enregistre le score dans la base de données pour suivre la progression de l'apprenant.
- Ajout d'un nouveau quiz (`addQCM`):

- Affiche un formulaire pour créer un nouveau quiz avec une question initiale et ses réponses possibles.
- Permet à l'apprenant de créer de nouveaux quiz pour élargir son éventail de sujets d'étude.
- Suppression d'un quiz (supprimer):
  - Supprime un quiz spécifique à partir d'un fichier XML.
  - Permet de maintenir la pertinence et la qualité des quiz disponibles pour les apprenants.
- Enregistrement d'un nouveau quiz (save):
  - Prend en charge la réception des données d'un nouveau quiz depuis un formulaire HTML.
  - Enregistre le quiz dans un fichier XML avec toutes les informations nécessaires, y compris les questions, les réponses et les images associées.

## Recommandation de cours

Après avoir passé avec succès le quiz, nous proposons de recommander des cours en fonction des matières abordées dans le quiz. Cette recommandation est basée sur les performances de l'étudiant et vise à renforcer ses connaissances dans les domaines où il pourrait avoir besoin d'amélioration:

- Enregistrement du score:
  - Lorsque l'étudiant soumet ses réponses au quiz, le système enregistre son score dans la base de données. Pour ce faire, nous utilisons une requête SQL pour vérifier si l'étudiant a déjà un enregistrement dans la table des scores. Si c'est le cas, nous mettons à jour son score global pour le quiz correspondant en utilisant une requête UPDATE. Sinon,

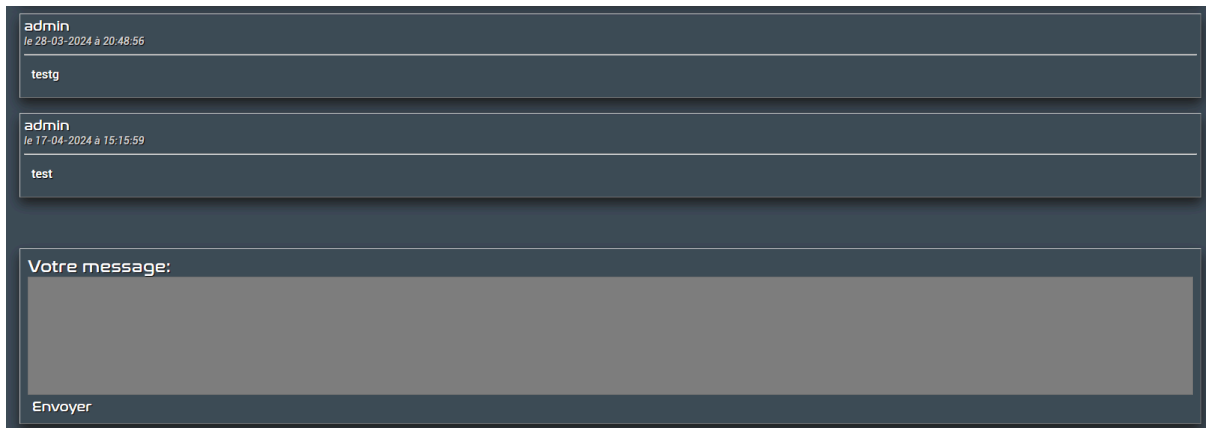
nous insérons un nouveau score pour l'étudiant en utilisant une requête INSERT.

- Identification de la matière:
  - Après avoir enregistré le score de l'étudiant, nous récupérons le nom de la matière du quiz réussi à partir des données du quiz. Cela implique de parcourir le fichier XML du quiz pour extraire cette information.
- Insertion dans la table de recommandation:
  - Une fois que nous avons identifié la matière du quiz, nous recherchons l'ID correspondant du cours associé à cette matière dans notre base de données. Pour cela, nous utilisons une requête SQL SELECT. Après avoir récupéré l'ID du cours, nous insérons une recommandation dans la table de recommandation en utilisant une requête INSERT. La recommandation comprend l'ID du cours, l'ID de l'utilisateur et la date à laquelle la recommandation a été faite.

Cette approche technique garantit que l'étudiant reçoive des recommandations personnalisées en fonction de ses performances dans les quiz, ce qui contribue à améliorer son apprentissage dans des domaines spécifiques.

## Forum de discussion entre les apprenants

Un forum de discussion interactif est mis à la disposition des apprenants pour favoriser les échanges et les discussions sur les sujets abordés dans les cours, de poser des questions et de partager des connaissances. Cette section du système est gérée par le contrôleur `controleur_Forum`, le contrôleur `controleur_Message`, et leur modèles correspondants.



The screenshot shows a forum interface with a dark blue header and footer. The first message is from 'admin' on 'le 28-03-2024 à 20:48:56' with the subject 'testg'. The second message is also from 'admin' on 'le 17-04-2024 à 15:15:59' with the subject 'test'. Below the messages is a large text area for a reply, labeled 'Votre message:', and a button labeled 'Envoyer' at the bottom.

- Lecture des forums :
  - La fonction `readAll()` dans le modèle `Modele_Forum` permet de lire tous les forums depuis la base de données. Elle récupère les titres des forums et les affiche sous forme de liens cliquables. Lorsque les utilisateurs cliquent sur un lien, ils sont redirigés vers la page du forum correspondant.
- Ajout d'un nouveau forum :
  - Les utilisateurs peuvent également ajouter de nouveaux forums grâce à la fonction `add()` dans le contrôleur `controleur_Forum.php`. Lorsqu'un utilisateur soumet un titre de forum via un formulaire, le titre est enregistré dans la base de données en utilisant la fonction `save()` du modèle `Modele_Forum()`. En cas d'erreur, un message d'erreur est affiché pour informer l'utilisateur.
- Suppression d'un forum :
  - La fonction `supprimer()` dans le contrôleur `controleur_Forum.php` permet la suppression d'un forum spécifique et de tous ses messages associés. Lorsqu'un utilisateur clique sur le bouton "Supprimer" à côté d'un forum, un message de confirmation s'affiche, et si l'utilisateur confirme, le forum et ses messages sont supprimés de la base de données.

- Lecture des messages dans un forum :
  - Le contrôleur `controleur_Message.php` et le modèle `modele_Message.php` gèrent la lecture des messages dans un forum spécifique. La fonction `readAll()` dans le modèle `Modele_Message` récupère tous les messages associés à un forum donné depuis la base de données et les affiche dans l'ordre chronologique inverse. Chaque message est présenté avec le nom de l'utilisateur, la date et le contenu du message.

Cette fonctionnalité de forum de discussion favorise la collaboration entre les apprenants, encourageant ainsi l'apprentissage actif et l'échange de connaissances.

L'espace personnel et les fonctionnalités de gestion de cours pour les apprenants constituent des éléments essentiels de notre plateforme d'apprentissage, contribuant à offrir une expérience éducative engageante, personnalisée et collaborative.

## Conclusion

Le développement de cette application d'apprentissage représente une étape importante dans la mise en place d'un outil efficace pour la formation en ligne. A travers deux parties distinctes, celle de l'administrateur et celle de l'apprenant, l'application offre un éventail de fonctionnalités visant à faciliter la gestion des cours et à améliorer l'expérience d'apprentissage des utilisateurs.

En conclusion, ce projet démontre l'importance de concevoir des solutions éducatives flexibles et adaptables aux besoins des utilisateurs. En combinant des fonctionnalités administratives robustes avec des outils d'apprentissage personnalisés et collaboratifs, notre application vise à offrir une expérience d'apprentissage enrichissante et efficace pour tous les utilisateurs.