1. Explication structure du projet (classes, MVC, arborescence)
2. Gestion de projet (sprints, ClickUp, planification du temps, pourquoi pas de forum)

Bien réorganiser les chapitres

Bien revoir les transitions des chapitres, temps conjugaison utilisé (présent ou passé ?)

Pied de page, annexes, page de garde

Faire les todos, et notes de fin de page

Table des matières

[II. Introduction 4](#_Toc102453095)

[A. Rappels du sujet 4](#_Toc102453096)

[III. Diagrammes de cas d’utilisation 5](#_Toc102453097)

[A. Acteurs 5](#_Toc102453098)

[B. Fonctionnalités globales 5](#_Toc102453099)

[C. Fonctionnalités du visiteur 6](#_Toc102453100)

[1. S’inscrire 6](#_Toc102453101)

[D. Fonctionnalités de l’apprenant 7](#_Toc102453102)

[1. Gérer un profil 7](#_Toc102453103)

[2. Consulter la liste des cours 7](#_Toc102453104)

[3. Consulter la liste des QCM 8](#_Toc102453105)

[4. Consulter un cours 8](#_Toc102453106)

[5. Remplir un QCM 9](#_Toc102453107)

[E. Fonctionnalités de l’admin 10](#_Toc102453108)

[1. Gérer les utilisateurs 10](#_Toc102453109)

[2. Gérer les cours 10](#_Toc102453110)

[3. Gérer les QCM 11](#_Toc102453111)

[IV. Détails d’implémentation des fonctionnalités 12](#_Toc102453112)

[A. Version PHP 12](#_Toc102453113)

[B. URL Rewriting 12](#_Toc102453114)

[C. Formats de cours 12](#_Toc102453115)

[D. Thèmes CSS 13](#_Toc102453116)

[E. Fichier XML des QCM 13](#_Toc102453117)

[F. Base de données 13](#_Toc102453118)

[G. Sessions 14](#_Toc102453119)

[H. JavaScript, DOM 14](#_Toc102453120)

[I. Ajax, requête API 15](#_Toc102453121)

[J. Sécurité 15](#_Toc102453122)

[V. Maquettes de l’interface utilisateur 16](#_Toc102453123)

[A. Charte graphique 16](#_Toc102453124)

[B. Conception des composants 16](#_Toc102453125)

[C. Conception des pages 17](#_Toc102453126)

[VI. Diagramme de classes 19](#_Toc102453127)

[A. Classes en lien avec l’utilisateur 21](#_Toc102453128)

[B. Classes en lien avec les cours 21](#_Toc102453129)

[C. Classes en lien avec les QCM 21](#_Toc102453130)

[D. Classes en lien avec les réponses de QCM 22](#_Toc102453131)

[VII. Modèle relationnel 23](#_Toc102453132)

[A. Tables en lien avec l’utilisateur 23](#_Toc102453133)

[B. Tables en lien avec les cours 23](#_Toc102453134)

[C. Tables en lien avec les QCM 23](#_Toc102453135)

[VIII. Structure du projet 24](#_Toc102453136)

[A. Arborescence du projet 24](#_Toc102453137)

[B. Architecture MVC 24](#_Toc102453138)

[1. Modèles 24](#_Toc102453139)

[2. Vues 24](#_Toc102453140)

[3. Contrôleurs 25](#_Toc102453141)

[C. Classes de base de données 26](#_Toc102453142)

[D. Classe de gestion de session 26](#_Toc102453143)

[E. Classes de gestion de fichier 26](#_Toc102453144)

[F. Classe du parser XML 26](#_Toc102453145)

# Introduction

Dans ce rapport, nous allons décrire l’ensemble des fonctionnalités implémentées ainsi que les technologies utilisées. Nous expliquerons la modélisation UML du site web, et enfin nous verrons la gestion de projet que nous avons mis en œuvre. Mais tout d’abord, nous allons rappeler le sujet et les critères de réalisation.

## Rappels du sujet

L’objectif de ce projet était de créer un site de formation destiné à des apprenants. Bien que le sujet fût assez libre, il nous imposait les fonctionnalités suivantes :

* Partie administrateur :
* Importer des cours sous plusieurs formats (vidéo, texte, etc.)
* Gérer les utilisateurs (création, modification, suppression)
* Gérer les cours et les QCM
* Partie apprenant :
* Proposer de répondre à des QCM afin de lui recommander des cours
* Forum de discussion entre les apprenants

Nous avons implémenté toutes ces fonctionnalités mis à part le forum de discussion, faute de temps, mais nous y reviendrons plus tard lors de l’explication de la gestion de projet et de la planification des tâches.

Par ailleurs, le sujet nous imposait les consignes techniques suivantes que nous avons respectées :

* Architecture MVC
* Utilisation des sessions / cookies
* Utilisation de PHP, JavaScript sans outils non vus en cours
* Format XML des QCM
* 2 thèmes différents (CSS)

# Diagrammes de cas d’utilisation

Dans cette première partie, nous allons expliquer les diagrammes de cas d’utilisation que nous avons conçu au début du projet permettant de clarifier les besoins et les fonctionnalités à implémenter tout au long du projet afin que tous les membres soient sur la même longueur d’onde. Mais tout d’abord, ci-dessous une vue d’ensemble des fonctionnalités implémentées divisées en quatre grandes catégories :

* Gestion de la connexion
  + En tant que visiteur, s’inscrire
  + En tant que visiteur, se connecter
  + En tant que connecté, se déconnecter
* Gestion du profil
  + En tant que connecté, consulter son profil
  + En tant que connecté, supprimer son compte
  + En tant que connecté, éditer son profil (e-mail, mot de passe, image, etc.)
  + En tant qu’admin, consulter le profil des apprenants
  + En tant qu’admin, supprimer les comptes des apprenants
  + En tant qu’admin, modifier le profil des apprenants
  + En tant qu’admin, lister les apprenants et en rechercher selon plusieurs filtres
* Gestion des cours
  + En tant que connecté, lister les cours disponibles et en rechercher selon plusieurs filtres
  + En tant que connecté, consulter un cours : en fonction du format de celui-ci (vidéo, texte), il doit être possible de visionner son contenu directement sur la page
  + En tant que connecté, marquer un cours comme « terminé »
  + En tant que connecté, avoir des recommandations de cours en fonction de la moyenne obtenue aux QCM répondus
  + En tant que connecté, consulter son historique des cours commencés et terminés
  + En tant qu’admin, créer, modifier et supprimer des cours
* Gestion des QCM
  + En tant que connecté, lister les QCM disponibles et en rechercher selon plusieurs filtres
  + En tant que connecté, répondre à un QCM avec des questions à choix unique, choix multiple ou de saisie d’une réponse. Une moyenne sur 20 est obtenue ce qui propose des recommandations de cours adaptés
  + En tant que connecté, consulter son historique des QCM commencés et terminés
  + En tant que connecté, recommencer de zéro un QCM commencé ou terminé
  + En tant qu’admin, créer, modifier et supprimer des QCM

## Acteurs

Nous avons voulu rester assez simple compte-tenu le retard du module de « Développement d’application web ». Trois acteurs sont définis :

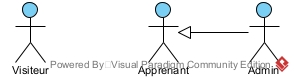


Diagramme de cas d'utilisation 1 : Acteurs

L’acteur « visiteur » est l’internaute qui n’est pas connecté à un compte, contrairement à « apprenant » et « admin ». De plus « admin » peut faire tout ce que « apprenant » peut faire (consulter des cours, répondre à des QCM, etc.).

## Fonctionnalités globales

Dans ce diagramme de cas d’utilisation du plus haut niveau, nous y avons décrit les fonctionnalités principales du site web. Ces cas d’utilisation correspondent aux fonctionnalités accessibles sur les pages principales du site web. Nous détaillerons plus en détail ces cas d’utilisation dans des sous-diagrammes.

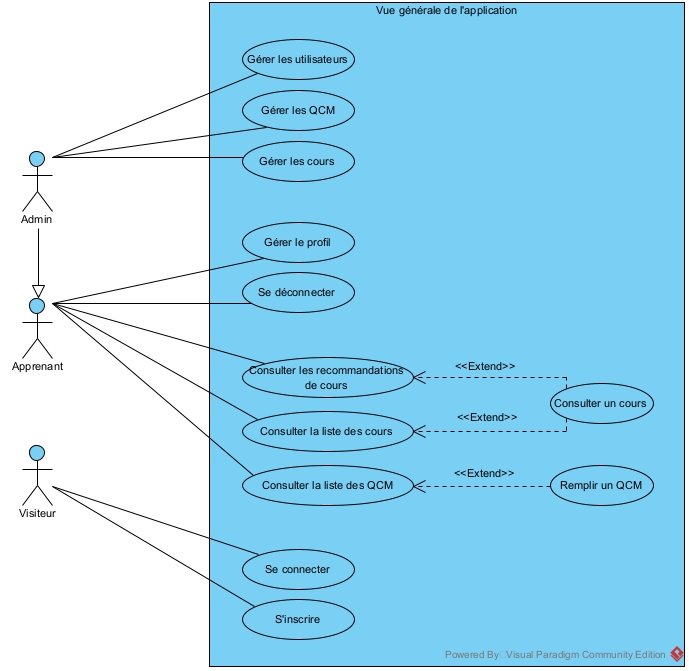


Diagramme de cas d'utilisation 2 : Fonctionnalités globales

Le visiteur n’a accès a aucune fonctionnalité, si ce n’est de pouvoir s’inscrire ou se connecter.

L’admin est responsable de gérer les données du site web, il gère le back-end, il peut ainsi créer, modifier, et supprimer les utilisateurs, les cours et les QCM.

L’apprenant peut gérer son profil comme le consulter, également, il peut consulter la liste des cours, des QCM et voir ses recommandations de cours.

## Fonctionnalités du visiteur

Nous allons décrire les fonctionnalités du visiteur par les sous-diagrammes associées aux cas d’utilisation du diagramme principal.

### S’inscrire

Ce sous-diagramme décrit les cas possibles pour le visiteur lorsqu’il veut s’inscrire. Il doit donc spécifier un e-mail, un pseudo et un mot de passe.

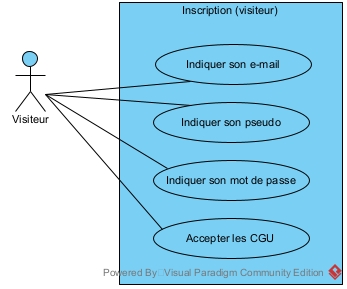


Diagramme de cas d'utilisation 3 : S’inscrire (visiteur)

## Fonctionnalités de l’apprenant

Nous allons décrire les fonctionnalités de l’apprenant par les sous-diagrammes associées aux cas d’utilisation du diagramme principal.

### Gérer un profil

Ce sous-diagramme décrit les cas possibles pour l’apprenant lorsqu’il veut gérer son profil ou pour l’admin lorsqu’il veut gérer le profil d’un autre apprenant.

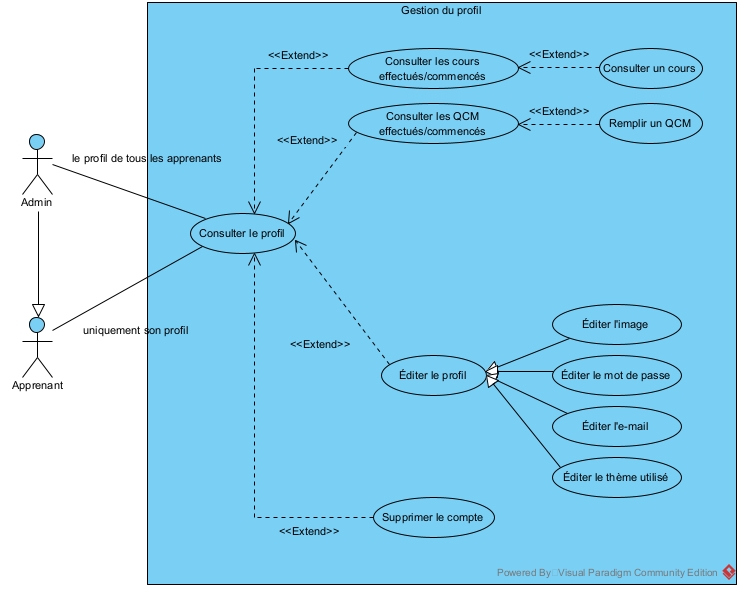


Diagramme de cas d'utilisation 4 : Gérer un profil (apprenant, admin)

En consultant un profil, il est possible d’avoir accès à l’historique des cours et des QCM commencés/terminés, et accéder à ceux-ci si souhaité. Il est également possible d’éditer le profil (e-mail, mot de passe, etc.) ou de supprimer le compte.

### Consulter la liste des cours

Ce sous-diagramme décrit les cas possibles pour l’apprenant lorsqu’il veut consulter la liste de tous les cours.

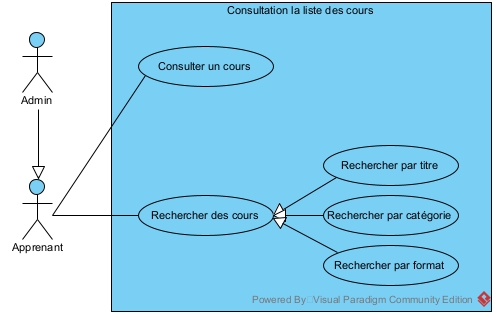


Diagramme de cas d'utilisation 5 : Consulter la liste de tous les cours (apprenant, admin)

A partir de la liste, il est ainsi possible de consulter un cours, ou bien de rechercher plus spécifiquement certains cours avec des filtres (titre, catégorie, format).

### Consulter la liste des QCM

Ce sous-diagramme décrit les cas possibles pour l’apprenant lorsqu’il veut consulter la liste de tous les QCM.

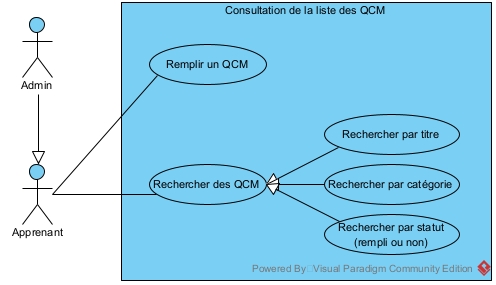


Diagramme de cas d'utilisation 6 : Consulter la liste de tous les QCM (apprenant, admin)

A partir de la liste, il est ainsi possible de consulter un QCM, ou bien de rechercher plus spécifiquement certains QCM avec des filtres (titre, catégorie, ou n’afficher que les QCM répondus).

### Consulter un cours

Ce sous-diagramme décrit les cas possibles pour l’apprenant lorsqu’il consulte la page d’un cours.

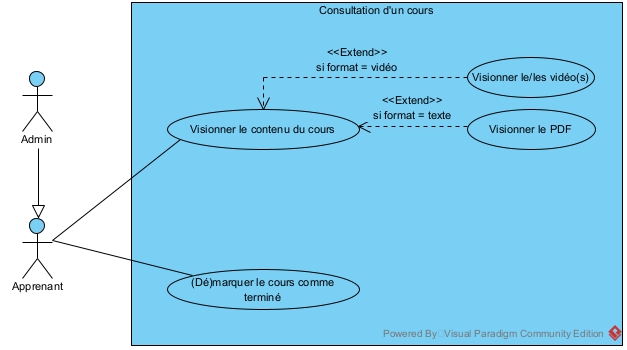


Diagramme de cas d'utilisation 7 : Consulter un cours (apprenant, admin)

Il doit être possible de visionner le contenu du cours selon son format (vidéo, texte). Comme, il est indiqué, nous proposons deux formats : les cours « vidéo » constitués d’une ou plusieurs vidéos YouTube, et les cours « texte » constitué d’un unique fichier PDF. Nous avons fait ce choix, car il aurait été peu praticable d’importer et d’héberger nos propres fichiers vidéo (pouvant être lourds) sur nos machines.

Enfin, il est possible de marquer un cours comme « terminé » ou inversement.

### Remplir un QCM

Ce sous-diagramme décrit les cas possibles pour l’apprenant lorsqu’il répond à un QCM.

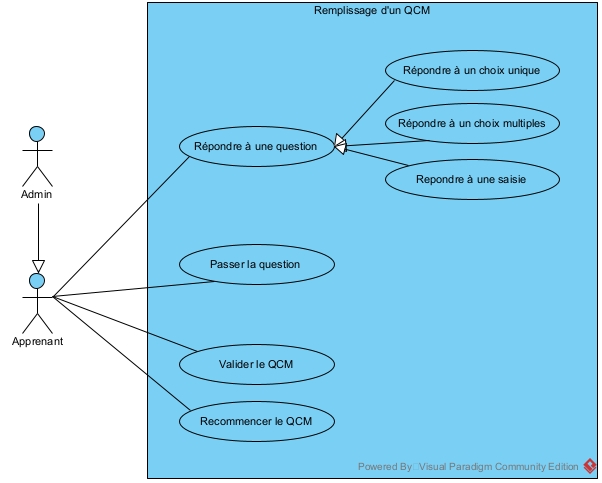


Diagramme de cas d'utilisation 8 : Remplir un QCM (apprenant, admin)

Lors du procédé de remplissage d’un QCM, l’apprenant peut répondre à des questions de plusieurs types : question à choix unique, à choix multiple, ou saisir une réponse textuelle.

Il peut passer une question et recommencer le QCM s’il le souhaite, autrement il peut valider le QCM (pour obtenir son résultat).

## Fonctionnalités de l’admin

Nous allons décrire les fonctionnalités de l’admin par les sous-diagrammes associées aux cas d’utilisation du diagramme principal.

### Gérer les utilisateurs

Ce sous-diagramme décrit les cas possibles pour l’admin lorsqu’il veut gérer les utilisateurs.

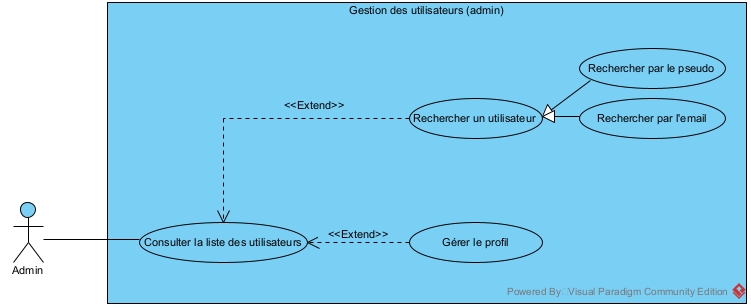


Diagramme de cas d'utilisation 9 : Gérer les utilisateurs (admin)

L’admin peut consulter une liste de tous les utilisateurs inscrits, et à partir de cette liste, il peut gérer le profil de l’utilisateur sélectionné, ou bien de rechercher plus spécifiquement certains utilisateurs avec des filtres (pseudo, e-mail).

### Gérer les cours

Ce sous-diagramme décrit les cas possibles pour l’admin lorsqu’il veut gérer les cours.

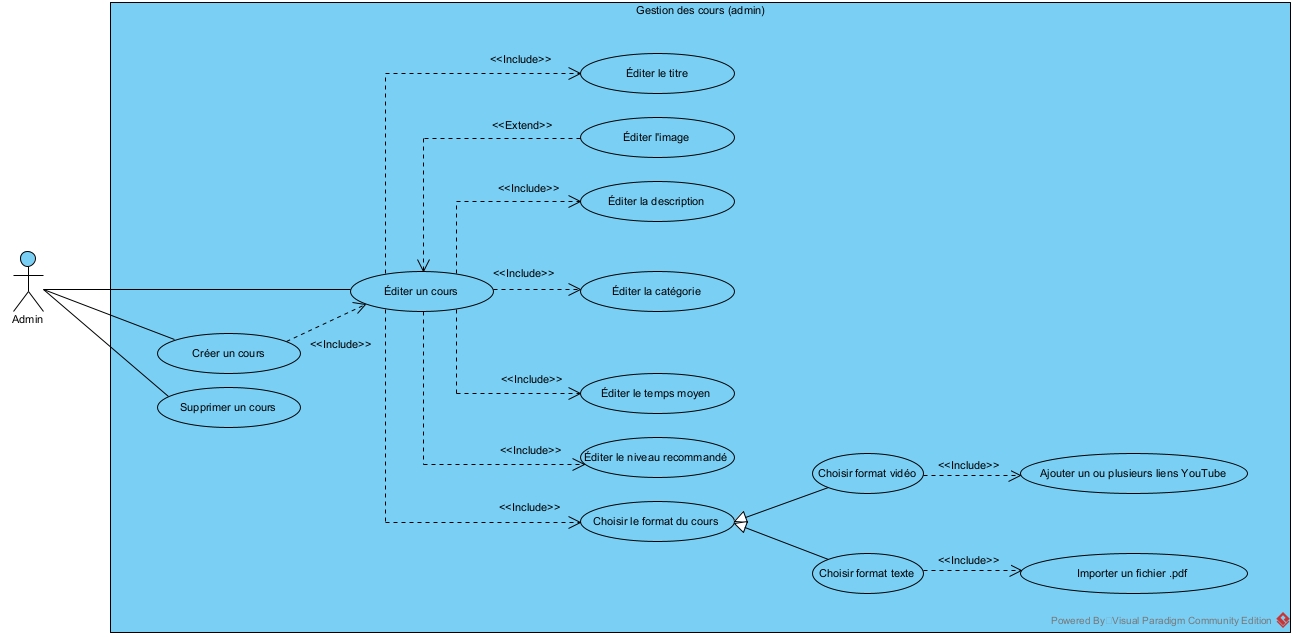


Diagramme de cas d'utilisation 10 : Gérer les cours (admin)

L’admin peut créer un cours, en modifier ou en supprimer un existant.

Lorsqu’il souhaite créer ou éditer un cours, il peut spécifier un titre, une image, une description, la catégorie, le temps moyen de complétion (en heures), et le niveau recommandé (débutant, intermédiaire, avancé) du cours. Enfin, il doit choisir le format du cours :

* Format « vidéo » : ajouter un ou plusieurs liens YouTube
* Format « texte » : importer un fichier PDF

### Gérer les QCM

Ce sous-diagramme décrit les cas possibles pour l’admin lorsqu’il veut gérer les QCM.

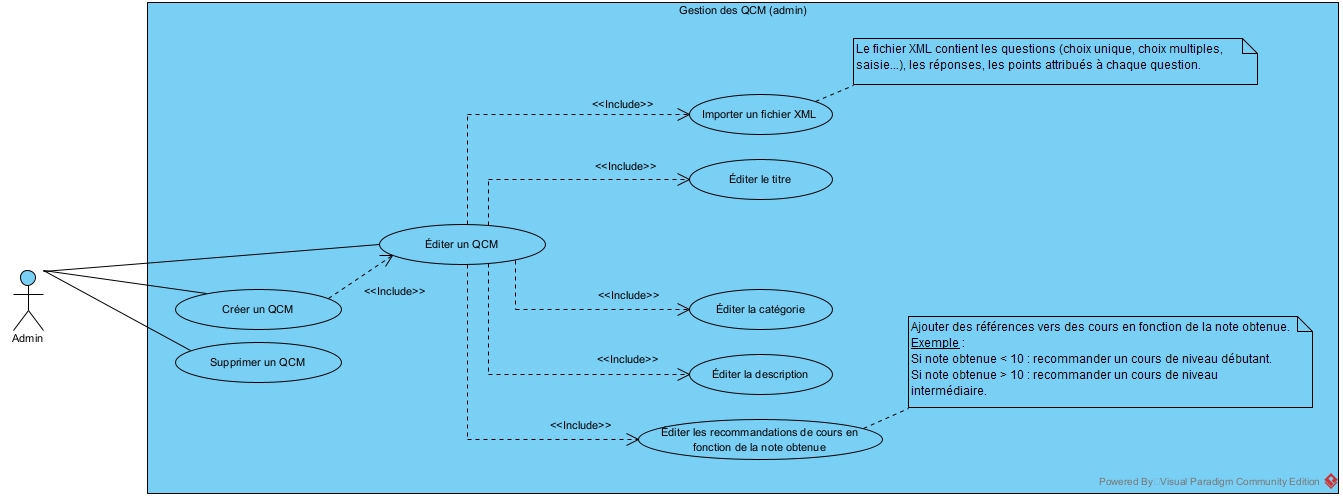


Diagramme de cas d'utilisation 11 : Gérer les QCM (admin)

L’admin peut créer un QCM, en modifier ou en supprimer un existant.

Lorsqu’il souhaite créer ou éditer un QCM, il peut spécifier un titre, une description, la catégorie, et importer un fichier XML. Ce fichier XML comporte les questions (choix unique, choix multiple, saisie d’une réponse textuelle), les réponses de chaque question mais aussi les points attribués à chaque question. Par ailleurs, il est possible de définir des recommandations de cours en fonction de la moyenne obtenue comme indiqué dans l’exemple.

# Détails d’implémentation des fonctionnalités

Après la vue d’ensemble des fonctionnalités de la partie précédente, nous allons détailler l’implémentation de certaines fonctionnalités pour mettre en évidence les mécanismes et principes utilisés conformément aux consignes techniques du sujet, ou pour montrer que nous avons essayé d’utiliser un maximum de techniques vues dans les CM du module.

## Version PHP

Nous avons utilisé la version 7.0.3 de PHP, ainsi que la version 5.7.11 de MySQL, car ceux-ci étaient inclus dans le logiciel tout-en-un Apache – PHP – MySQL : « UwAmp ». Ainsi, il est possible que notre site web ne fonctionne pas avec des versions inférieures.

## URL Rewriting

Nous avons préféré ne pas utiliser de contrôleur principal qui permet d’accéder aux différents pages du site web en spécifiant par exemple un paramètre d’URL « ?page=accueil ». Nous avons jugé cette solution peu élégante et contraignante, c’est pourquoi, nous avons utilisé le principe de réécriture d’URL.

Ce principe permet de créer des URL alias pointant vers un fichier de notre site web. Ainsi, nous avons un alias par page pointant vers le contrôleur associé à la page à afficher. Ces réécritures d’URL ont été faites dans le fichier Apache « .htaccess » :

Options +FollowSymlinks

RewriteEngine on

RewriteRule ^accueil$             /controllers/pages/accueil/accueil.php [L]

RewriteRule ^connexion$           /controllers/pages/connexion/connexion.php [L]

RewriteRule ^deconnexion$         /controllers/pages/connexion/deconnexion.php [L]

RewriteRule ^inscription$         /controllers/pages/inscription/inscription.php [L]

RewriteRule ^profil$              /controllers/pages/profil/profil.php [L]

RewriteRule ^profil/supprimer$    /controllers/pages/profil/supprimer.php [L]

RewriteRule ^profil/modifier$     /controllers/pages/profil/modifier.php [L]

RewriteRule ^utilisateurs$        /controllers/pages/utilisateurs/rechercher.php [L]

RewriteRule ^cours$               /controllers/pages/cours/cours.php [L]

RewriteRule ^cours/rechercher$    /controllers/pages/cours/rechercher.php [L]

RewriteRule ^cours/supprimer$     /controllers/pages/cours/supprimer.php [L]

RewriteRule ^cours/marquer$       /controllers/pages/cours/marquer.php [L]

RewriteRule ^cours/affichage$     /controllers/pages/cours/affichage.php [L]

RewriteRule ^cours/editer$        /controllers/pages/cours/edition.php [L]

RewriteRule ^qcm$                 /controllers/pages/qcm/remplir/remplir.php [L]

RewriteRule ^qcm/rechercher$      /controllers/pages/qcm/rechercher.php [L]

RewriteRule ^qcm/supprimer$       /controllers/pages/qcm/supprimer.php [L]

RewriteRule ^qcm/edition$         /controllers/pages/qcm/editer.php [L]

Fichier 1 : « .htaccess » – Définition des réécritures d’URL

## Formats de cours

Nous avons deux formats de cours :

* Format « vidéo » : comporte une ou plusieurs vidéos YouTube
* Format « texte » : comporte un fichier PDF

Le lecteur vidéo de YouTube est directement intégré à la page du cours, de même que pour les fichiers PDF, la visionneuse PDF du navigateur y est aussi intégrée.

Ce choix a été fait puisque nous trouvions contraignant d’importer et d’héberger nos propres fichiers vidéo (pouvant être lourds) sur nos machines. Par ailleurs, l’avantage est que la visionneuse PDF est déjà fournie en fonctionnalités comme avec le zoom, le chapitrage, le téléchargement du PDF, etc.

## Thèmes CSS

Conformément au sujet, nous avons deux thèmes CSS sur notre site web : un thème clair et un thème sombre. Le thème est automatiquement appliqué en détectant le thème du navigateur ou du système d’exploitation en utilisant la caractéristique média « prefers-color-scheme (TODO) » de CSS.

Par exemple, sur Windows, il est possible de choisir entre le thème clair et le sombre, et à partir de celui-ci, nos fichiers CSS le détectent et appliquent le thème correspondant.

## Fichier XML des QCM

Comme le sujet nous demandait d’utiliser XML pour les QCM, nous avons choisi de déclarer les questions du QCM dans un fichier XML. Dans ce fichier XML, nous pouvons définir des questions de deux types :

* Question de saisie d’une réponse textuelle (dans un champ de texte) :

Il est possible de déclarer la bonne réponse, et le *placeholder* du champ de texte.

* Question à choix :

Il est possible de déclarer si choix multiple autorisé, ainsi que les différents choix de la question.

Pour tous les types de question, nous pouvons définir la question posée, et les points attribués en cas de bonne réponse, ou au contraire, les points retirés en cas de sélection d’une mauvaise réponse.

Les informations du QCM en lui-même (titre, description, catégorie) sont définies dans le formulaire d’édition d’un QCM.

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

<**QCM**>

  <**QuestionSaisie** bonneReponse="1996" points="1" question="Quelle est l'année de création de JS ?" placeholder="Saisir l'année de création..." />

  <**QuestionSaisie** bonneReponse="console.log('Bonjour');" points="2" question='Donner la fonction pour afficher dans la console "Bonjour"' placeholder='\_\_\_.\_\_\_("Bonjour");' />

  <**QuestionChoix** isMultiple="true" question="Quels sont tous les mots-clés pour déclarer une variable ?">

    <**Choix** isValide="true" points="0.5">var</**Choix**>

    <**Choix** isValide="false" points="-0.5">set</**Choix**>

    <**Choix** isValide="true" points="0.5">let</**Choix**>

    <**Choix** isValide="false" points="-0.5">new</**Choix**>

    <**Choix** isValide="true" points="0.5">const</**Choix**>

  </**QuestionChoix**>

  <**QuestionChoix** isMultiple="false" question="Qu'est-ce qu'une expression ternaire ?">

    <**Choix** isValide="true" points="1">Un moyen concis d'écrire un if-else</**Choix**>

    <**Choix** isValide="false" points="-1">Un moyen d'appeler trois fonctions en même temps</**Choix**>

    <**Choix** isValide="false" points="-1">Un calcul mathématique impliquant trois variables</**Choix**>

  </**QuestionChoix**>

</**QCM**>

Fichier 2 : Exemple d’un fichier XML définissant les questions d’un QCM

Pour analyser le fichier XML au moment de l’importation du fichier, nous avons utilisé l’API « SimpleXML » de PHP vue en CM.

## Base de données

Pour manipuler la base de données MySQL avec PHP, nous avons utilisé l’interface d’accès « PDO[[1]](#footnote-1) ».

## Sessions

Nous avons utilisé les sessions PHP afin de maintenir la connexion au compte de l’utilisateur active même en changeant de page. Nous stockons dans la session, l’identifiant de l’utilisateur si celui-ci est connecté, sinon rien.

## JavaScript, DOM

Dans le projet, nous avons utilisé du JavaScript notamment pour faire les sections dynamiques de certaines pages. Un exemple d’une telle section dynamique est le menu « burger » situé dans l’en-tête du site web qui apparaît/disparaît lorsque l’on clique sur l’image de profil :

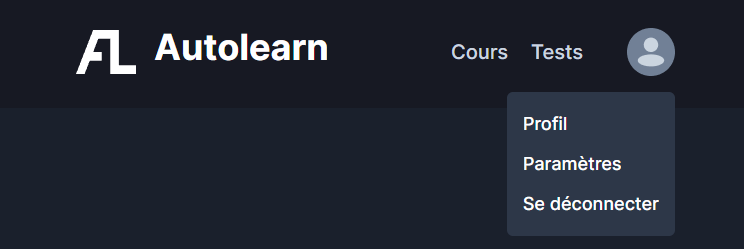


Image 1 : Menu « burger » s’ouvrant après un clique sur l’image de profil

Un autre exemple, est dans la page d’édition d’un cours vidéo, lorsque nous avons la possibilité d’ajouter de nouveaux liens de vidéo YouTube en cliquant sur un bouton sans actualisation de la page :

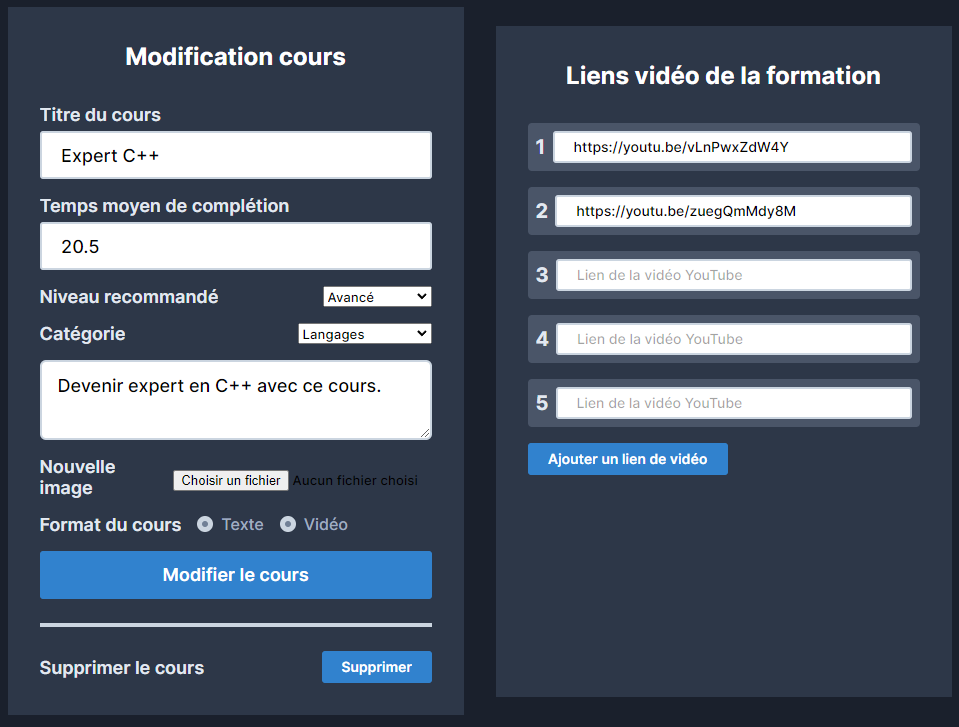


Image 2 : Ajout de liens vidéo YouTube supplémentaires dans la page d’édition d’un cours (panneau à droite)

Afin de faire cela, nous avons utilisé les évènements JavaScript pour détecter l’appuie sur un bouton par exemple, et nous avons dû manipuler le DOM pour modifier les classes CSS d’un élément HTML (image n°1), ou encore insérer des éléments HTML dans le DOM (image n°2).

Nous n’avons pas utilisé la bibliothèque « jQuery » car nous avons globalement peu utilisé JavaScript dans le projet, ce n’était donc pas forcément nécessaire.

## Ajax, requête API

Nous avons utilisé « Ajax » une seule fois, et c’est dans la page d’édition d’un QCM afin de récupérer la liste de tous les cours pour les afficher dans les boîtes de sélection où il faut choisir un cours à recommander lorsque la moyenne obtenue est comprise entre deux notes :

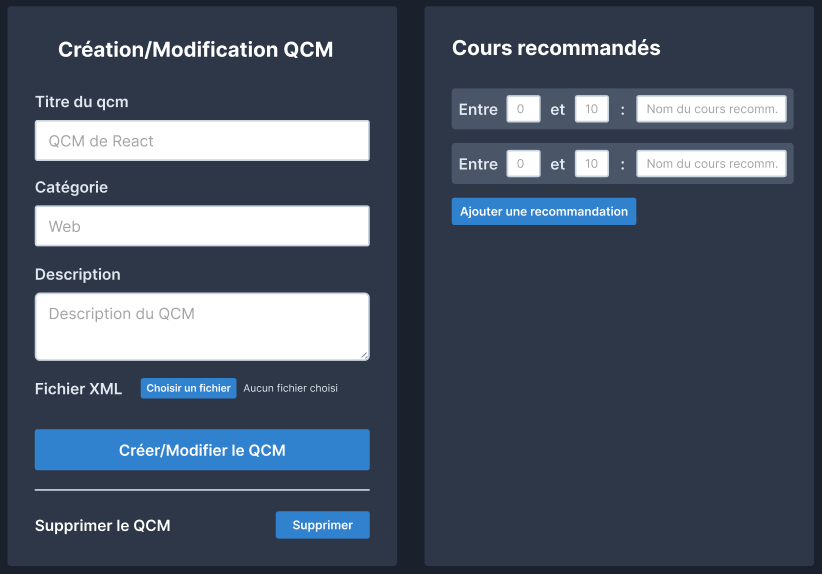


Image 3 : Choix d’un cours à recommander dans la page d’édition d’un QCM (panneau à droite)

Dans le back-end, nous avons un contrôleur responsable de retourner tous les cours dans un format JSON. Dans le front-end, nous faisons une requête XHR avec JavaScript à ce contrôleur et nous affichons les cours dans les boîtes de sélection.

## Sécurité

Concernant la sécurité, nous nous sommes prémunies des injections SQL en utilisant les requêtes préparées autant que nécessaire. Également, nous avons essayé de nous prémunir des injections JavaScript (XSS) en désinfectant les données reçus des formulaires ou des paramètres d’URL.

Par ailleurs, les contrôleurs font des vérifications concernant les autorisations d’accès à la page (ex. l’utilisateur est-il admin ?).

# Maquettes de l’interface utilisateur

Après avoir conçu les diagrammes de cas d’utilisation afin d’avoir une idée générale des fonctionnalités à implémenter, cela nous a permis de travailler sur les maquettes des pages du site web avant de réellement débuter la programmation. Pour réaliser ces maquettes, nous avons utilisé le logiciel « Figma » qui est un outil collaboratif de design d’interfaces utilisateur.

Nous avons réalisé une maquette par thème (clair et sombre).

## Charte graphique

En première étape, nous avons définit la charte graphique qui nous servira à la conception de toutes les pages. La charte graphique comprend la typographie ainsi que les couleurs.

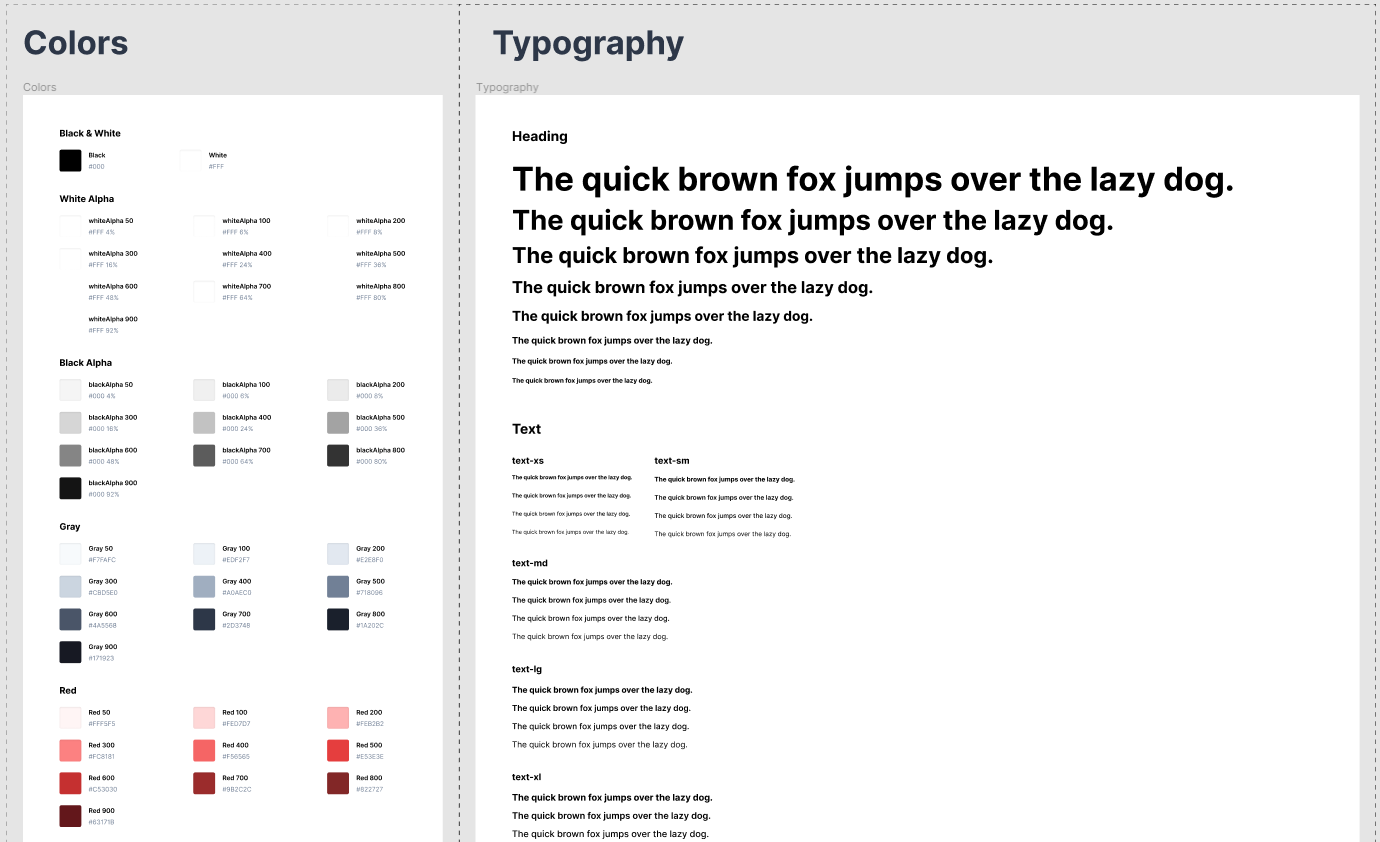


Image 4 : Charte graphique du site web

## Conception des composants

En deuxième étape, nous avons conçu le design des composants des pages du site web. Ces composants sont des éléments communs à plusieurs pages comme par exemple : les boutons, les champs de saisie, les cases à cocher, ou encore des sections comme la boîte d’un cours.

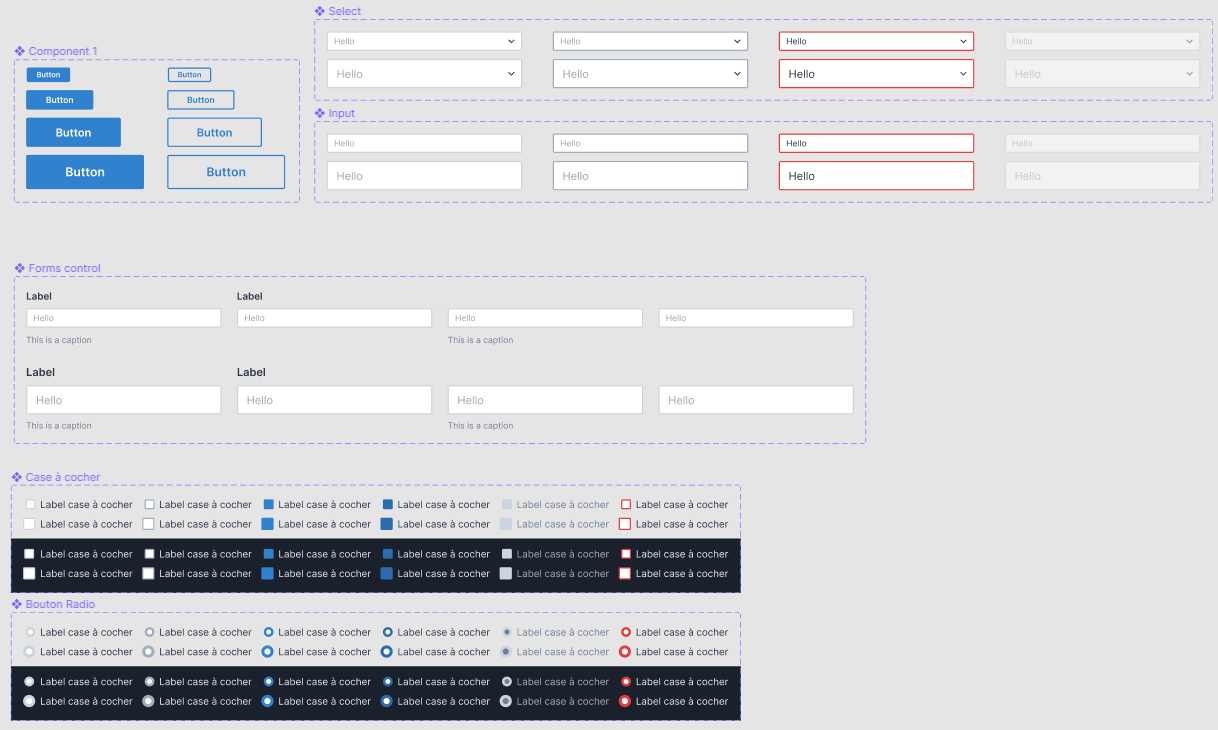


Image 5 : Composants élémentaires (boutons, champs de formulaire, etc.)

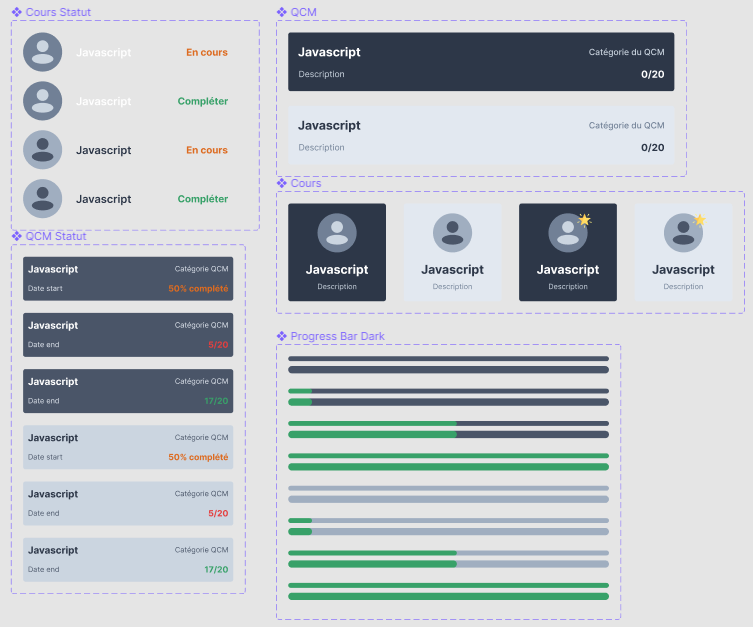


Image 6 : Composants plus complexes (boîte d’un cours, etc.)

## Conception des pages

La charte graphique et les composants (autrement nommés « design system ») étant réalisées, nous avons pu débuter le design des pages du site web.

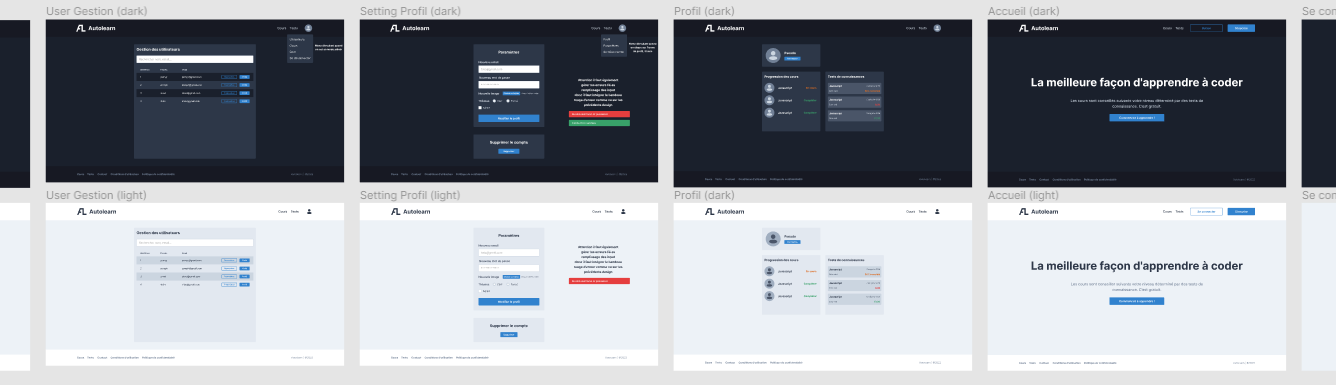


Image 7 : Ensemble des maquettes de pages dans le thème clair et sombre

Les intégrateurs front-end ont pu ainsi facilement développer les pages en HTML et en CSS en consultant les maquettes, les codes couleurs, et la typographie (polices, tailles).

# Diagramme de classes

Nous allons maintenant nous attarder sur le diagramme de classes, en effet, bien que nous ayons maintenant les fonctionnalités définis et les maquettes des pages du site web, il n’est pas encore possible de commencer le développement sans les classes métiers, car celles-ci vont être utilisé aussi bien dans le front-end que dans le back-end.

Le diagramme de classes se trouve sur la prochaine page.

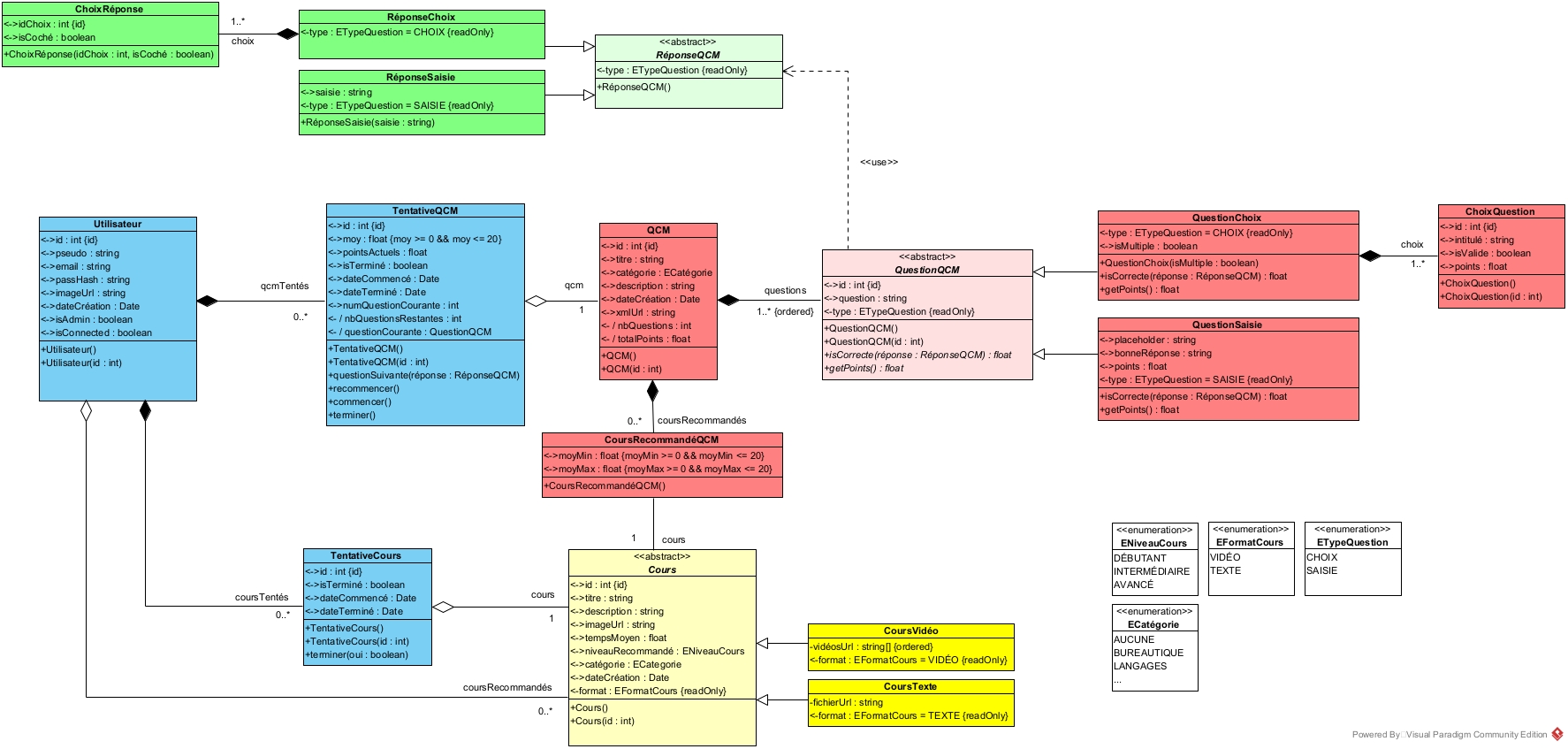


Image 8 : Diagramme de classes

Note :

* Le « <-> » devant les attributs et propriétés indiquent qu’ils ont soit uniquement un getter (« <- »), soit uniquement un setter (« -> »), soit les deux (« <-> »).
* Le « / » devant certaines propriétés correspond à des propriétés calculées (ex. dont la valeur est calculée à partir d’autres attributs).
* Les classes et les méthodes abstraites ont un nom en italique.

## Classes en lien avec l’utilisateur

Les classes présentées en bleu sont en lien avec l’utilisateur.

La classe « Utilisateur » représente les acteurs « apprenant » et « admin ». Le mot de passe est haché pour des raisons de sécurité. Un utilisateur possède soit aucune, soit une ou plusieurs tentatives de cours et de QCM. Ces tentatives sont instanciées au moment où l’utilisateur ouvre un cours ou un QCM. Par ailleurs, l’utilisateur possède une liste de cours qui lui sont recommandés.

La classe « TentativeCours » correspond à un cours que l’utilisateur a ouvert et par définition commencé. Elle est associée à une instance de la classe « Cours ». La méthode « terminer » permet de changer le statut de la tentative de cours en « terminé » ou inversement.

La classe « TentativeQCM » correspond à un QCM que l’utilisateur a ouvert et par définition commencé. Elle est associée à une instance de la classe « QCM ». Voici le comportement des méthodes :

* « commencer » : passer à la première question
* « recommencer » : remettre l’instance avec ces valeurs par défaut pour recommencer le QCM du début
* « questionSuivante » : passer à la prochaine question tout en vérifiant la validité de la réponse fournie et en attribuant les points, si c’était la dernière question, alors terminer le QCM
* « terminer » : calculer la moyenne, définir la date de complétion et terminer le QCM

## Classes en lien avec les cours

Les classes présentées en jaune sont en lien avec les cours.

La classe abstraite « Cours » permet de contenir les attributs communs à tous les formats de cours (titre, description, catégorie, etc.). Les classes descendantes sont « CoursVidéo » et « CoursTexte » qui respectivement représente un cours vidéo avec une liste ordonnée de liens vidéo YouTube, et un cours texte qui contient l’URL vers le fichier PDF.

## Classes en lien avec les QCM

Les classes présentées en rouge sont en lien avec les QCM.

Pour bien comprendre les associations, voici l’idée générale : un QCM contient un certain nombre de questions dans un ordre précis ; ces questions ont deux formes : les questions de saisie textuelle d’une réponse ou les questions à choix ; dans le cas d’une question à choix, celle-ci contient alors plusieurs choix possibles (représentant les cases à cocher).

La classe « QCM » contient une liste de cours à recommander en fonction de la moyenne obtenue, ceci est fait par la classe « CoursRecommandéQCM ». La classe « QCM » possèdent deux propriétés calculées « nbQuestions » et « totalPoints » qui comme leur nom indique, retourne le nombre de questions total et le nombre de points total possible d’obtenir.

La classe abstraite « QuestionQCM » représente une question et les méthodes abstraites « isCorrecte » et « getPoints » doivent être redéfini au cas par cas dans les classes descendantes.

Voici le comportement des méthodes pour la classe descendante « QuestionSaisie » :

* « isCorrecte » : retourne le nombre de points gagnés si la réponse fournie est bonne, sinon aucun point
* « getPoints » : retourne le nombre de points que vaut la question

Voici le comportement des méthodes pour la classe descendante « QuestionChoix » :

* « isCorrecte » : retourne le nombre de points gagnés si les choix cochés de la réponse fournie sont bons, sinon aucun point et dans le pire des cas, un nombre de points négatifs, s’il y a des choix à points négatifs cochés
* « getPoints » : retourne le nombre de points que vaut tous les choix

La méthode « isCorrecte » utilise l’instance « RéponseQCM » fournie en paramètre pour vérifier la validité de la réponse fournie d’où l’association « use » sur la classe « QuestionQCM ».

À noter que les classes en lien avec les questions et le choix sont instanciées au moment de l’analyse du fichier XML d’un QCM.

## Classes en lien avec les réponses de QCM

Les classes présentées en vert sont en lien avec les réponses à des questions de QCM.

Ces classes sont calquées sur les classes de « QuestionQCM » et elles permettent de stocker la réponse d’un utilisateur à une question afin de pouvoir vérifier sa validité par rapport aux bonnes réponses.

Ces classes sont instanciées lors du traitement du formulaire de réponse à un QCM, elles ne sont pas stockées puisqu’elles ont une durée de vie très éphémères.

# Modèle relationnel

Maintenant, que nous venons juste de voir le diagramme de classes, nous allons nous focaliser sur la conversion de ce celui-ci en un modèle relationnel pour la base de données. Notez, d’ailleurs que les classes « RéponseQCM » ne sont pas persistantes donc elles n’ont pas été intégrées dans la base de données. Ci-dessous, le modèle relationnel :

**Conventions de notation** :

* *Clé primaire* #attribut
* *Clé étrangère* attribut
* *Les deux* #attribut

## Tables en lien avec l’utilisateur

**Utilisateur**(#id : int, pseudo\* : text, email\* : text, passHash : text, imageUrl : text, dateCréation : date, isAdmin : bool)

**TentativeCours**(#id : int, idCours : int, isTerminé : bool, dateCommencé : date, dateTerminé : date)

**TentativeQCM**(#id : int, idQCM : int, moy : float, pointsActuels : float, isTerminé : bool, dateCommencé : date, dateTerminé : date, numQuestionCourante : int)

**UtilisateurTentativesCours**(#idTentativeCours : int, #idUtilisateur : int)

**UtilisateurTentativesQCM**(#idTentativeQCM : int, #idUtilisateur : int)

\* contrainte d’unicité sur l’attribut.

## Tables en lien avec les cours

Cours(#id : int, titre : text, description : text, imageUrl : text, tempsMoyen : float, dateCréation : date, niveauRecomm : enum\*, catégorie : enum\*, format : enum\*)

CoursTexte(#idCours : int, fichierUrl : text)

CoursVidéo(#idCours : int)

CoursVidéosUrl(#idCours : int, #videoUrl : text, ordre : int)

\* prennent uniquement comme valeurs, celles des énumérations définies dans le diagramme de classes.

## Tables en lien avec les QCM

QCM(#id : int, titre : text, description : text, dateCréation : date, xmlUrl : text, catégorie : enum\*)

QuestionQCM(#id : int, idQCM : int, question : text, type : enum\*)

QuestionSaisie(#idQuestionQCM : int, placeholder : text, bonneRéponse : text, points : float)

QuestionChoix(#idQuestionQCM : int, isMultiple : bool)

ChoixQuestion(#id : int, idQuestion : int, intitule : text, isValide : bool, points : float)

CoursRecommandéQCM(#idQCM : int, #idCours : int, moyMin : float, moyMax : float)

\* prennent uniquement comme valeurs, celles des énumérations définies dans le diagramme de classes.

# Structure du projet

Nous avons terminé de passer en revue les parties concernant les préparatifs et la conception du projet. Nous allons maintenant expliquer la structure du projet, l’architecture mise en œuvre et les classes utiles de parler.

## Arborescence du projet

Ci-dessous, l’arborescence globale du projet en corrélation avec l’architecture MVC :

src/

├─ assets/          Contient les images, fichiers importés, etc.

├─ models/          Contient tous les modèles

├─ views/           Contient toutes les vues

├─ controllers/     Contient tous les contrôleurs

├─ databases/       Contient tout ce qui est en lien avec la base de données

├─ config.php       Contient les configurations et constantes

├─ index.php        Point d'entrée (redirige vers la page d'accueil)

Le fichier de configuration contient les paramètres de connexion à la base de données, et les constantes définissant les chemins d’accès aux répertoires où les fichiers importés sont stockés.

Ensuite, pour l’arborescence dans le répertoire des vues, voici un extrait :

views/

├─ components/          Contient les composants

│  ├─ button/           Exemple d'un composant "bouton"

│  │  ├─ button.css/

│  │  ├─ button.php/

│  │  ├─ button.js/

├─ pages/               Contient les pages

│  ├─ accueil/          Exemple d'une page "accueil"

│  │  ├─ accueil.css/

│  │  ├─ accueil.php/

│  │  ├─ accueil.js/

Enfin, pour l’arborescence dans le répertoire des contrôleurs, voici encore un extrait :

controllers/

├─ classes/             Contient des classes utiles (ex. parser XML pour QCM)

├─ pages/               Contient les contrôleurs des pages

│  ├─ accueil/          Exemple d'un contrôleur d'une page "accueil"

│  │  ├─ accueil.php/

├─ utils.php            Contient des fonctions utilitaires

## Architecture MVC

Nous allons détailler comment nous avons implémenté l’architecture MVC dans notre site web.

### Modèles

Il y a peu à dire sur les modèles, puisque les modèles sont simplement l’implémentation des classes métiers du diagramme de classes. Ces modèles sont utilisés aussi bien dans les vues pour afficher les données que dans les contrôleurs pour modifier leurs données.

### Vues

Concernant les vues, afin de séparer le plus la logique de l’affichage, les vues reçoivent les données à afficher en paramètres d’une fonction que le contrôleur correspondant appelle pour afficher la vue.

Pour être plus précis, nous avons dans chaque fichier PHP des vues, une fonction « afficher(…) » où le code HTML de la vue est à l’intérieur et les données à afficher sont situées en arguments de cette fonction, par exemple, voici un exemple de fichier vue :

<?php

/\*\*

 \* Afficher le formulaire d'édition d'un QCM.

 \* @param boolean $isEditMode Formulaire est-il en mode d'édition d'un QCM existant ?

 \* @param **QCM**|null $qcm       QCM à éditer, sinon `null` si en mode création.

 \* @return void

 \*/

function afficherFormulaire(bool $isEditMode, **QCM** $qcm = null)

{

  if ($isEditMode) {

    echo "<h1>" . $qcm->getTitre() . "</h1>";

  } else {

    echo "<h1>Créer un nouveau QCM";

  }

}

Fichier 3 : Exemple d’une fichier PHP d’une vue affichant un formulaire d’édition d’un QCM

Cette fonction est appelée par le contrôleur qui passe les données en paramètres pour afficher la vue.

Par ailleurs, au lieu de cela, nous utilisons des templates, c’est-à-dire, que nous décomposons les parties communes des pages en fichiers PHP que nous importons où nous voulons les insérer. Cela peut concerner l’en-tête et le pied de page, ou la boîte de notification.

### Contrôleurs

Concernant les contrôleurs, nous en avons un par page, il y a donc autant de vues que de contrôleurs.

En général, nos contrôleurs s’occupent d’afficher la vue si aucun formulaire n’a été envoyé, autrement le formulaire est traité avec redirection et confirmation de succès ou d’erreur à la clé. Certains contrôleurs ne sont associés à aucune vue, ce sont par exemple, ceux qui effectuent une action très basique comme supprimer un cours, marquer un cours comme « terminé », etc.

Ci-dessous, le modèle de contrôleur typique que nous avons utilisé la plupart du temps :

TODO

## Classes de base de données

## Classe de gestion de session

## Classes de gestion de fichier

## Classe du parser XML

1. « PHP Data Objects » est une interface d’accès à une base de données avec PHP. [↑](#footnote-ref-1)