Rapport de projet

Développement d'application web

Rapport expliquant la modélisation et l'architecture du site web, ainsi que la stratégie de gestion de projet utilisée.

MEMBRES DE L'EQUIPE

Eddy DRUET
Rémy AULOY
Clément GILI
Tom BARBIER
Guillaume SONVICO
Lilian MANZANO

Groupe n°12

Université de Bourgogne
Licence 3 – Informatique
2021 – 2022

TABLES DES MATIERES

I. 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.

A. 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[1]
- Utilisation des sessions/cookies
- Utilisation de PHP, JavaScript sans outils non vus en cours
- Format XML des QCM
- 2 thèmes différents

[1] « Modèle – Vue – Contrôleur »

II. 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 le compte 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

A. 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.).

B. FONCTIONNALITES 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.

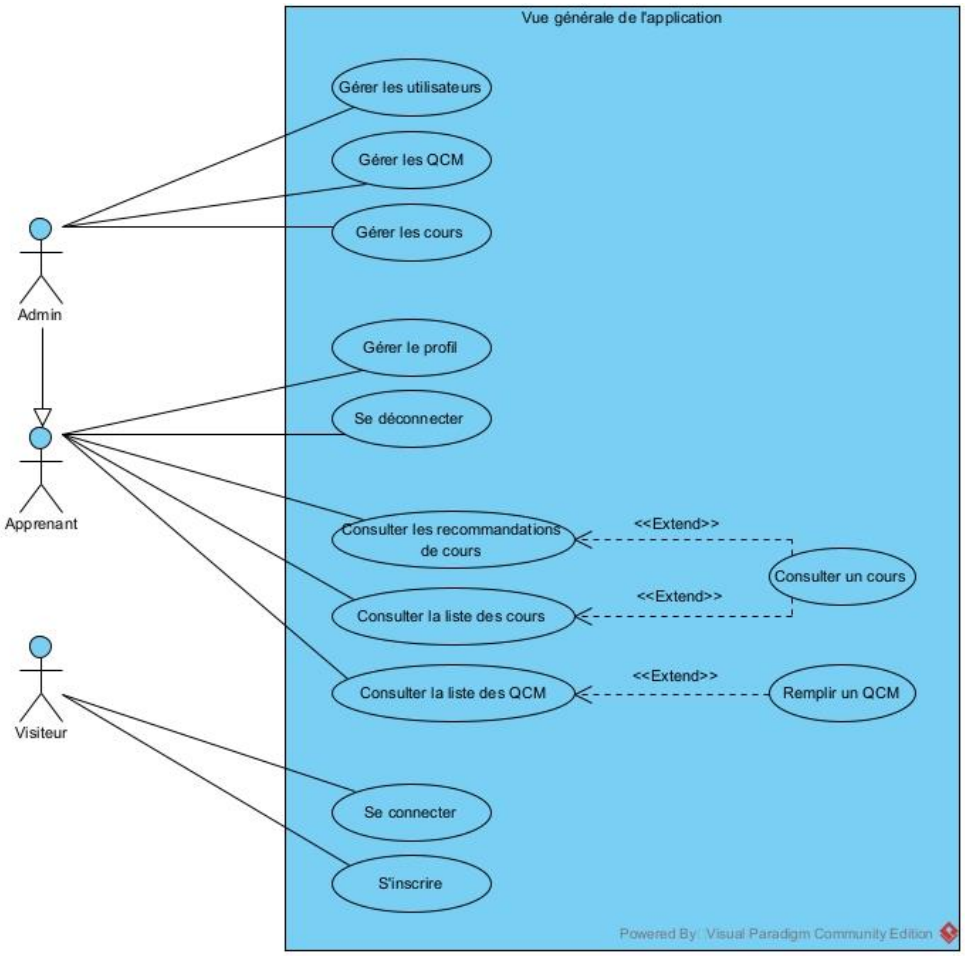

Diagramme de cas d'utilisation 2 : Fonctionnalités globales

Le visiteur n'a accès à 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.

C. FONCTIONNALITES DU VISITEUR

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

1. 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.

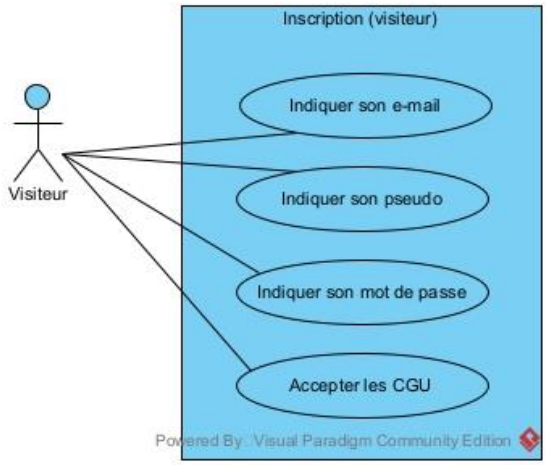

Diagramme de cas d'utilisation 3 : S'inscrire (visiteur)

D. FONCTIONNALITES DE L'APPRENANT

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

1. GERER 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.

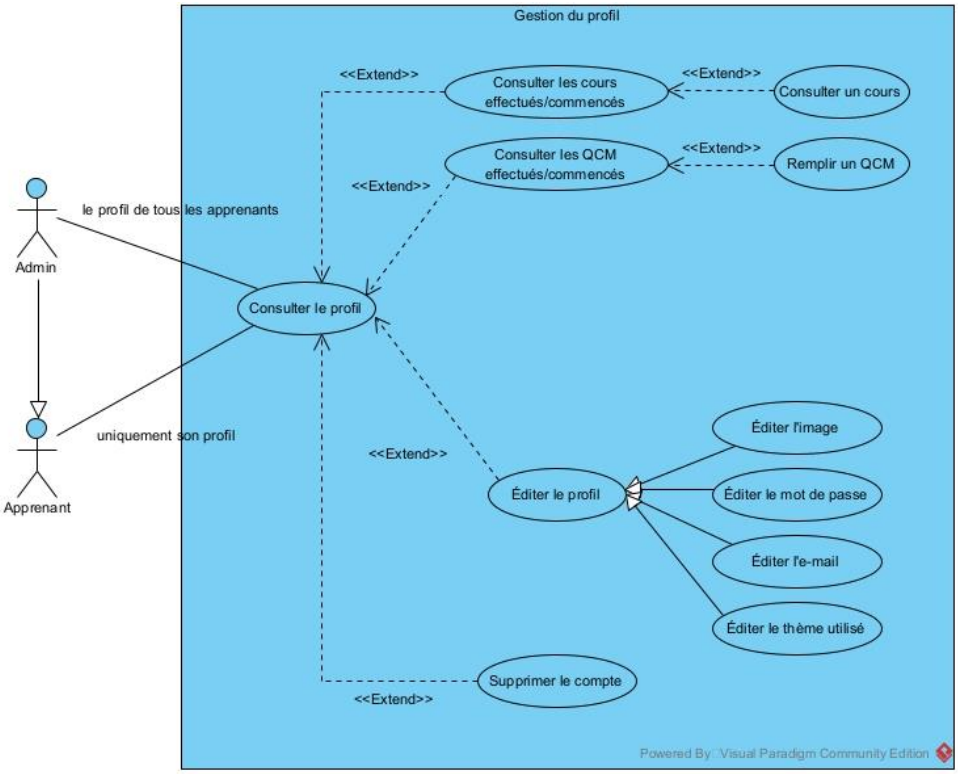

Diagramme de cas d'utilisation 4 : Gérer un profil (apprenant, admin)

En consultant un profil, il est possible d'avoir accès à l'historique des cours et des QCM commencés/terminés, et d'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.

2. 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.

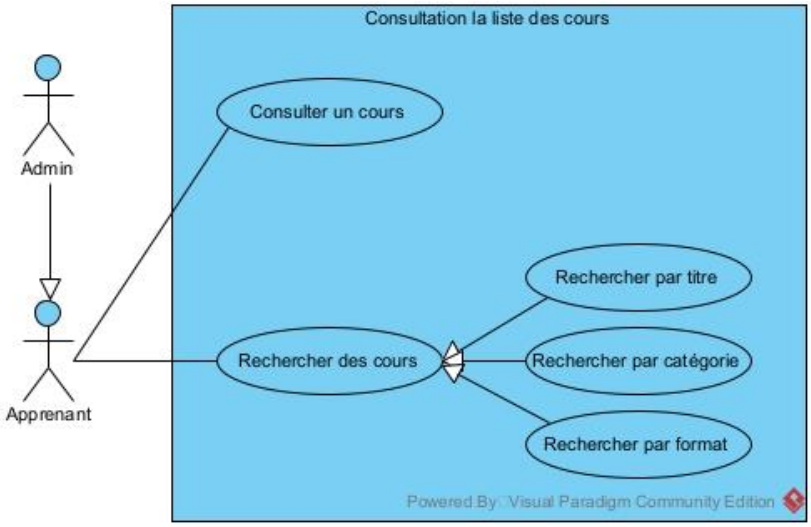

Diagramme de cas d'utilisation 5 : Consulter la liste de tous les cours (apprenant, admin)

À 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).

3. 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.

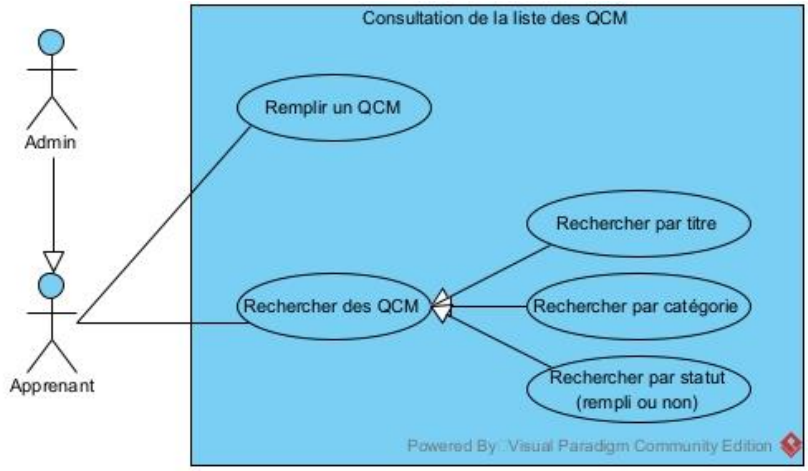

Diagramme de cas d'utilisation 6 : Consulter la liste de tous les QCM (apprenant, admin)

À 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).

4. CONSULTER UN COURS

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

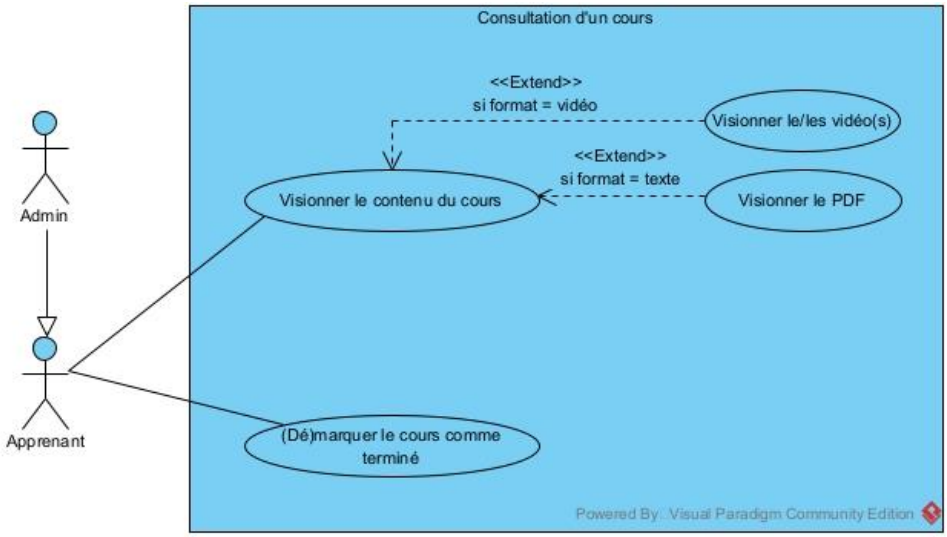

Diagramme de cas d'utilisation 7 : Consulter un cours (apprenant, admin)

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 » constitués 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.

5. REMPLIR UN QCM

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

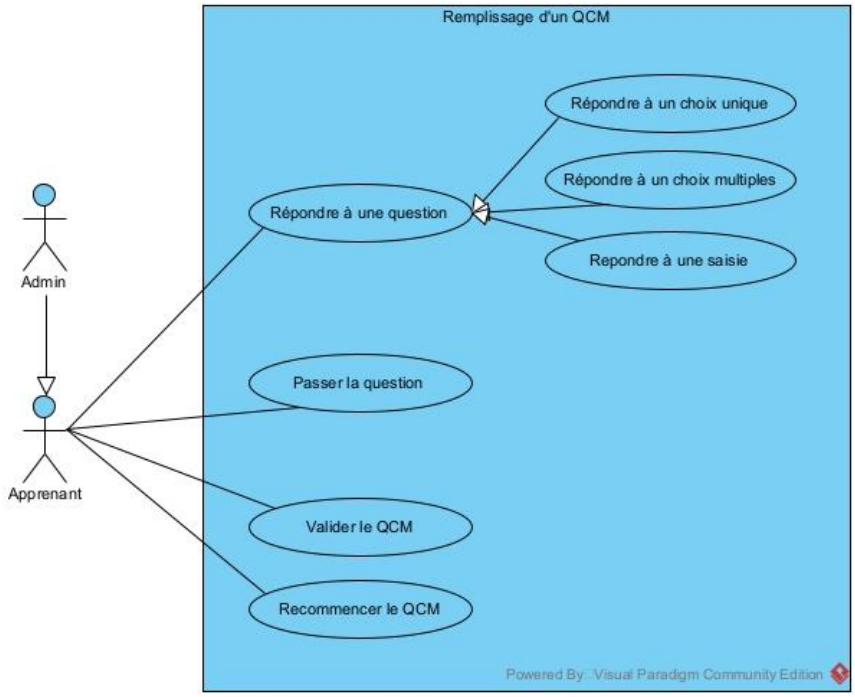

Diagramme de cas d'utilisation 8 : Remplir un QCM (apprenant, admin)

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).

E. FONCTIONNALITES DE L'ADMIN

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

1. GERER LES UTILISATEURS

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

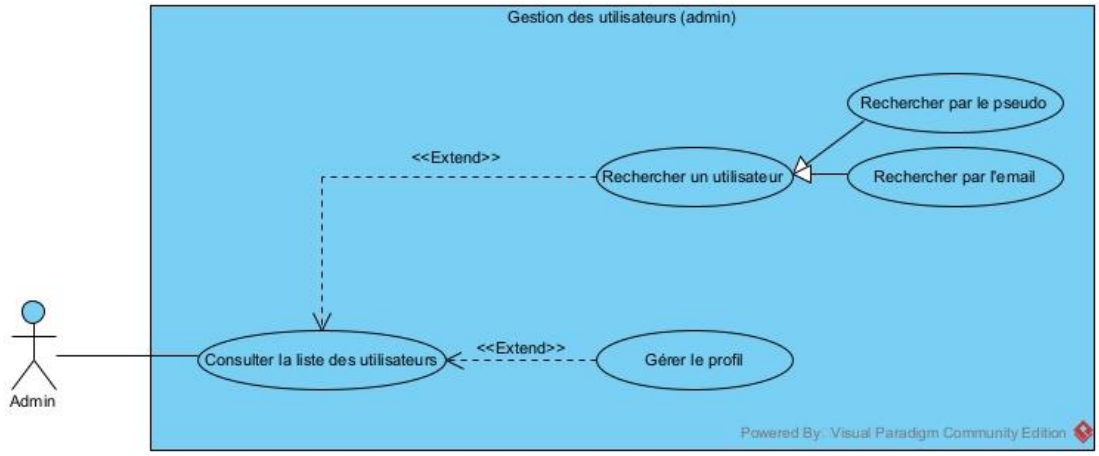

Diagramme de cas d'utilisation 9 : Gérer les utilisateurs (admin)

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).

2. GERER LES COURS

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

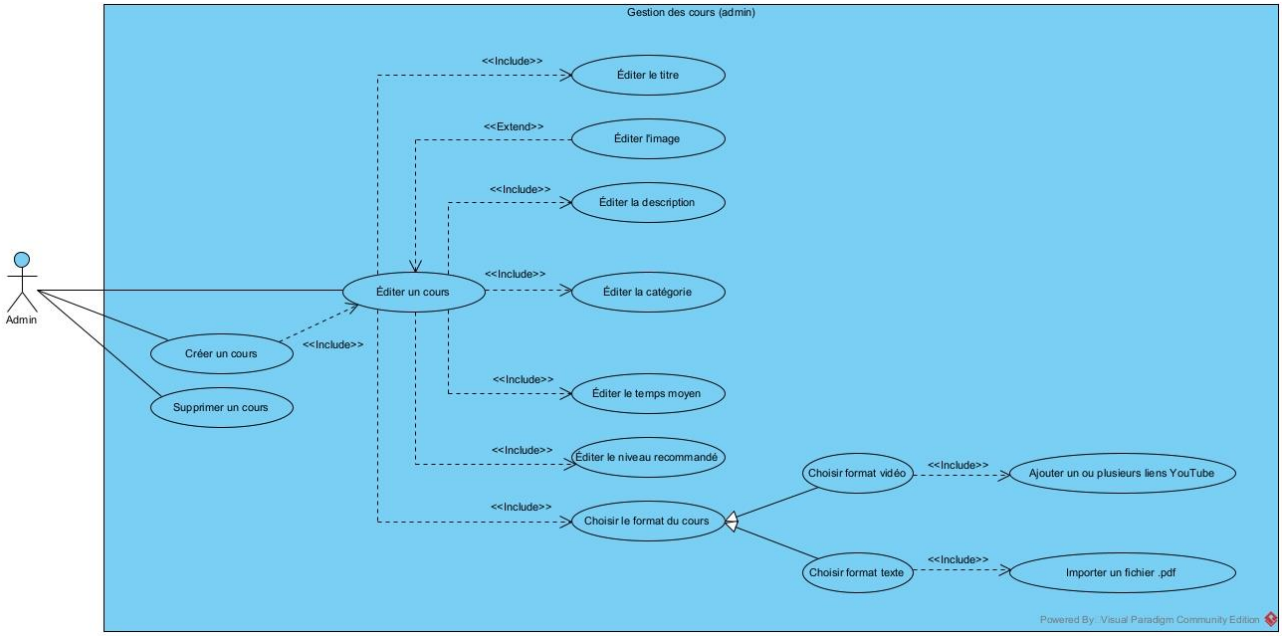

Diagramme de cas d'utilisation 10 : Gérer les cours (admin)

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

3. GERER LES QCM

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

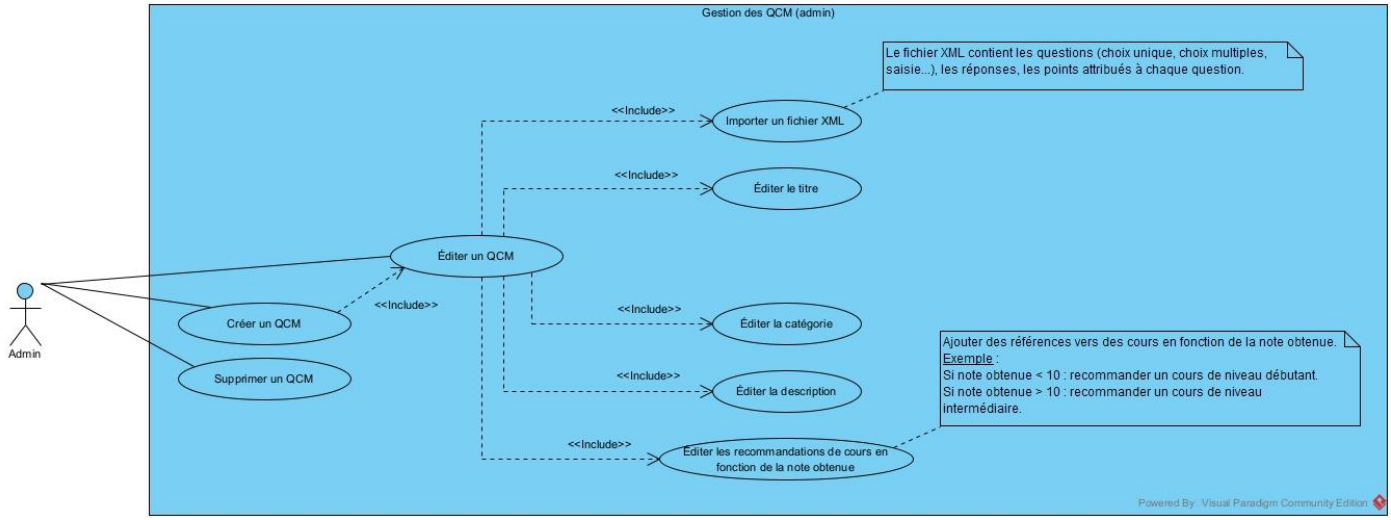

Diagramme de cas d'utilisation 11 : Gérer les QCM (admin)

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.

III. 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'interface utilisateur.

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

A. CHARTE GRAPHIQUE

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

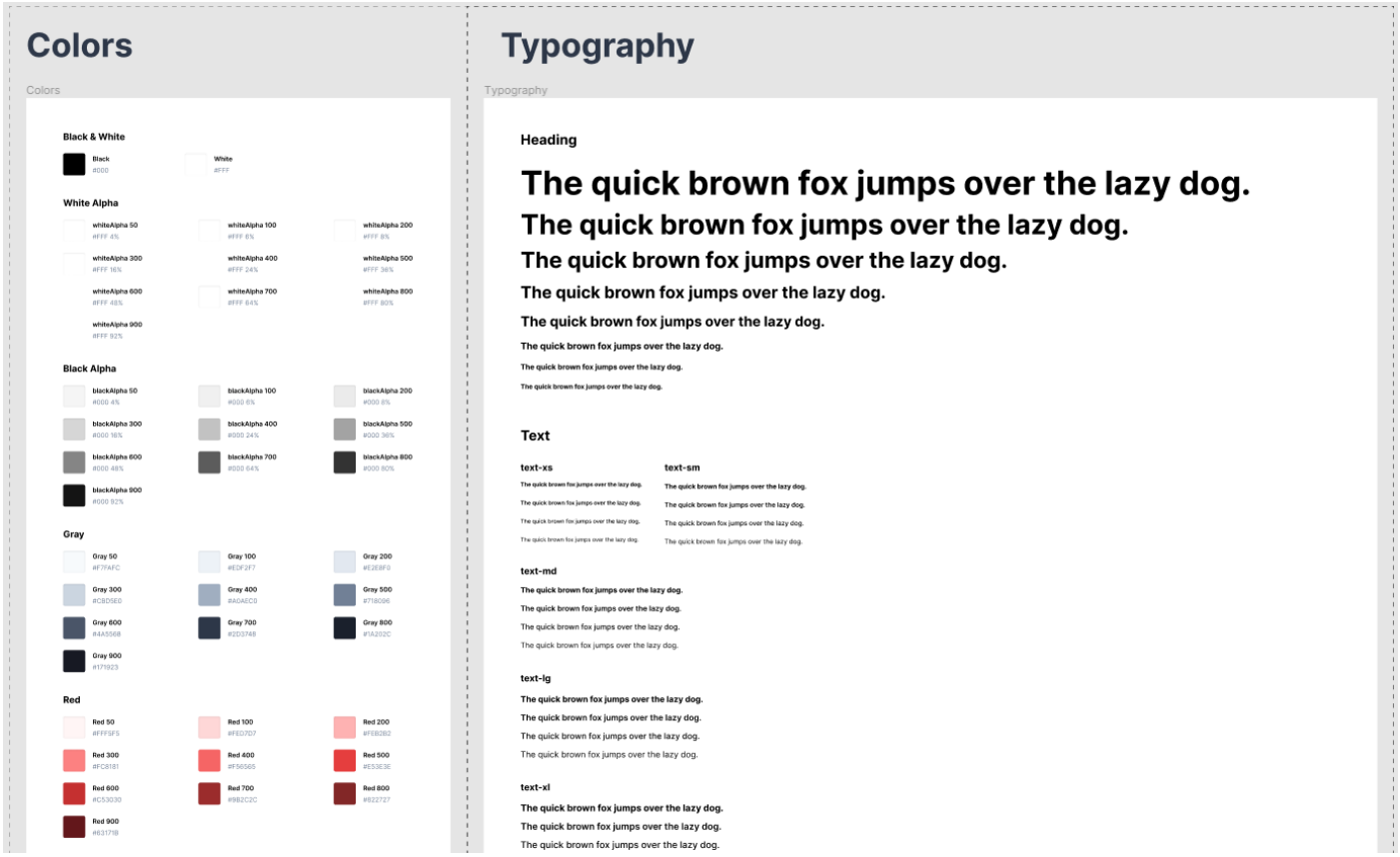

Image 1 : Charte graphique du site web

B. 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.

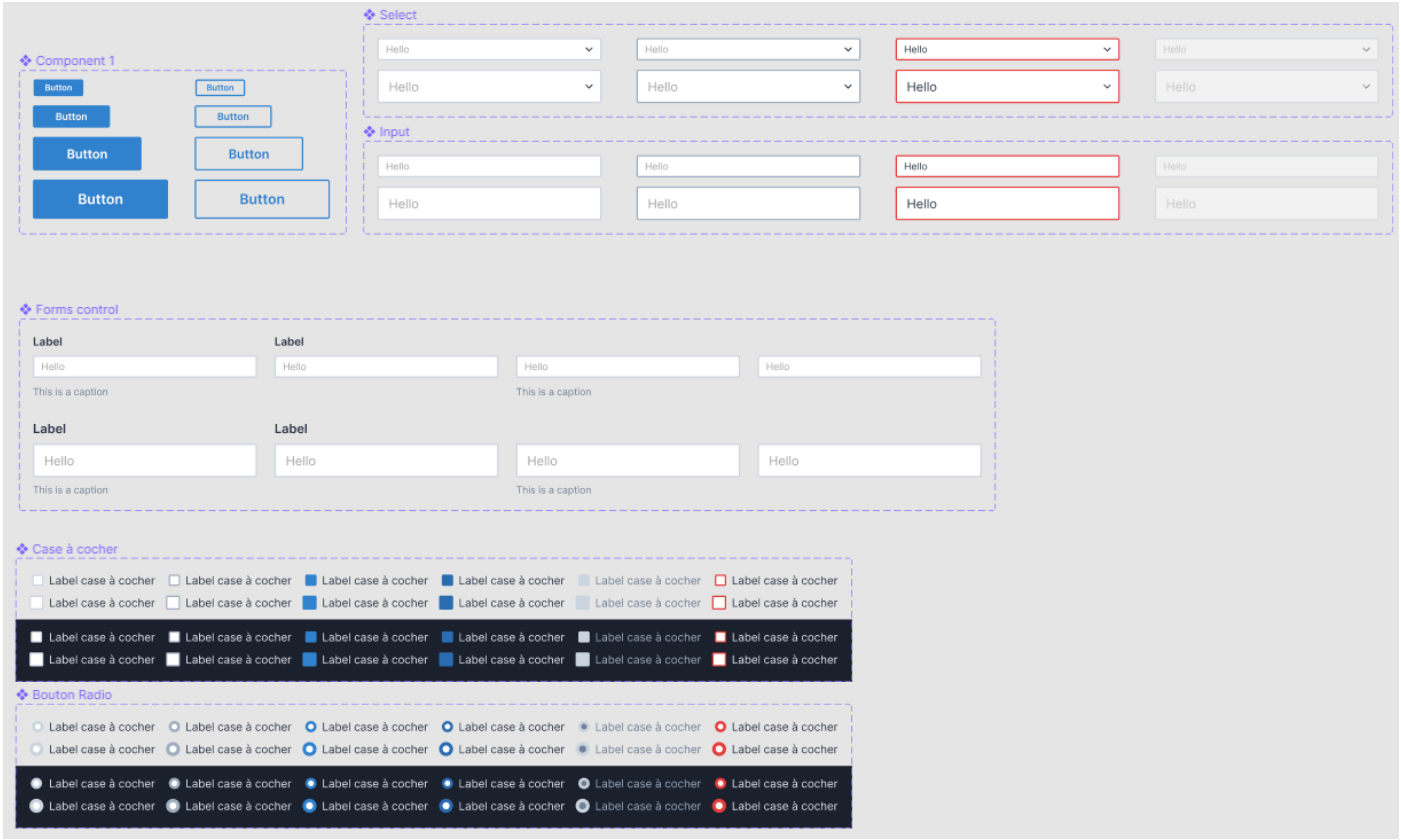

Image 2 : Composants élémentaires (boutons, champs de formulaire, etc.)

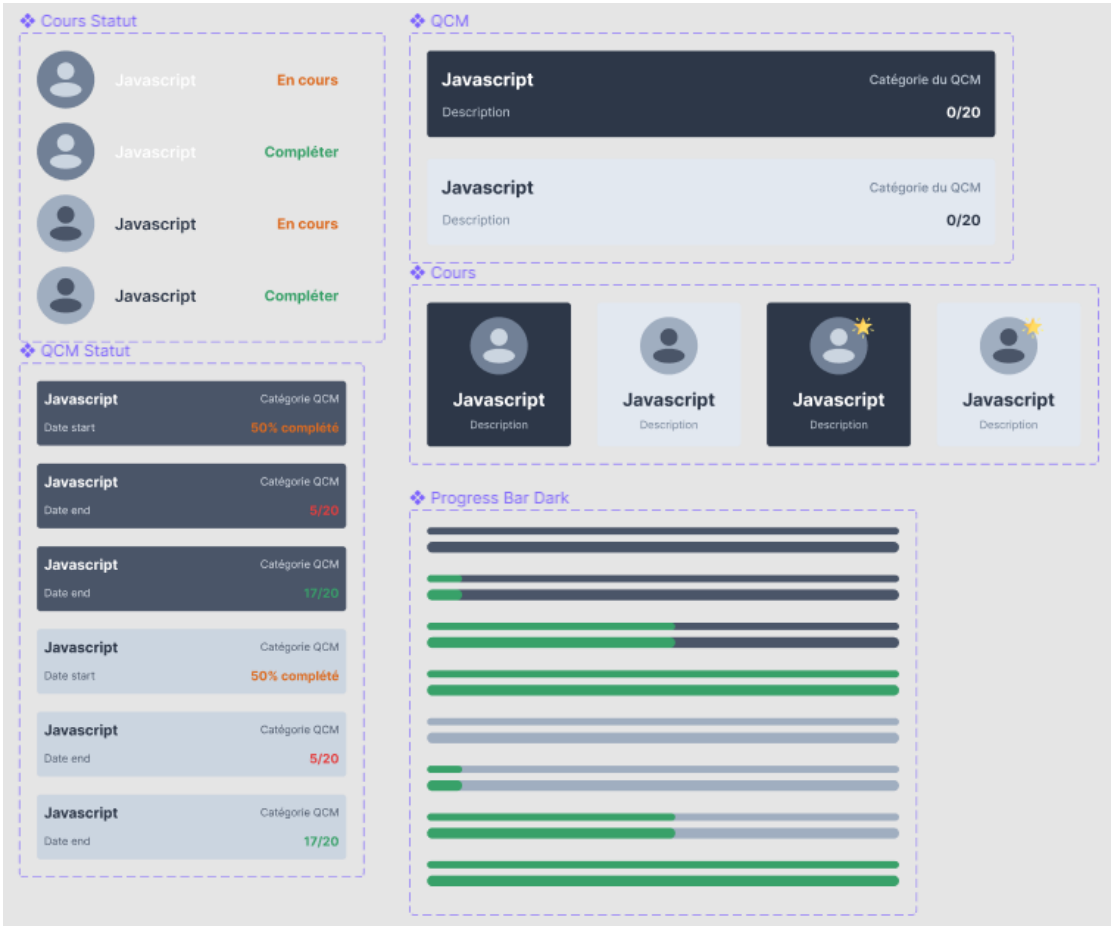

Image 3 : Composants plus complexes (boîte d'un cours, etc.)

C. CONCEPTION DES PAGES

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

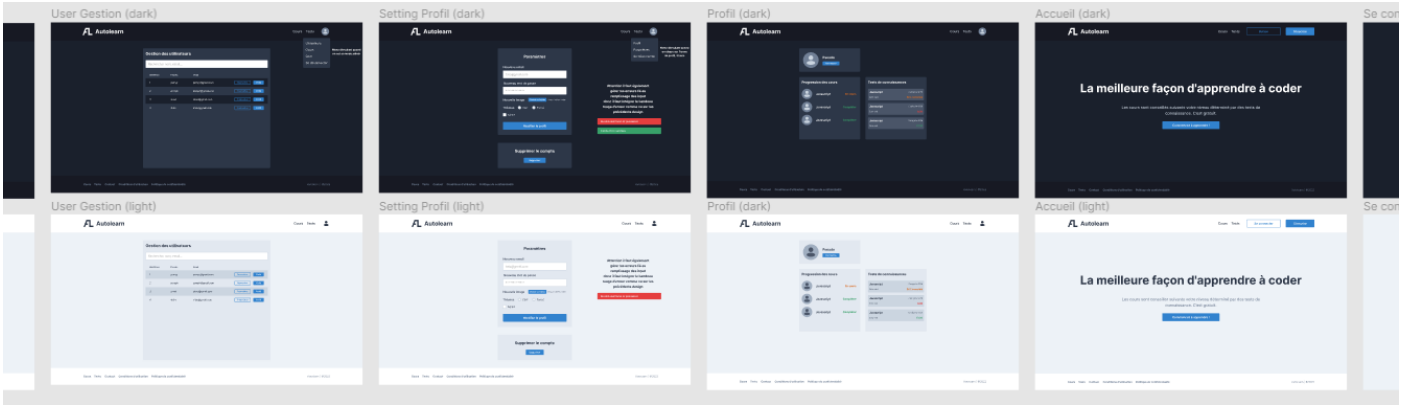

Image 4 : Ensemble des maquettes de page dans le thème clair et sombre

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).

IV. 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 utilisées aussi bien dans le front-end que dans le back-end.

Le diagramme de classes se trouve sur la prochaine page.

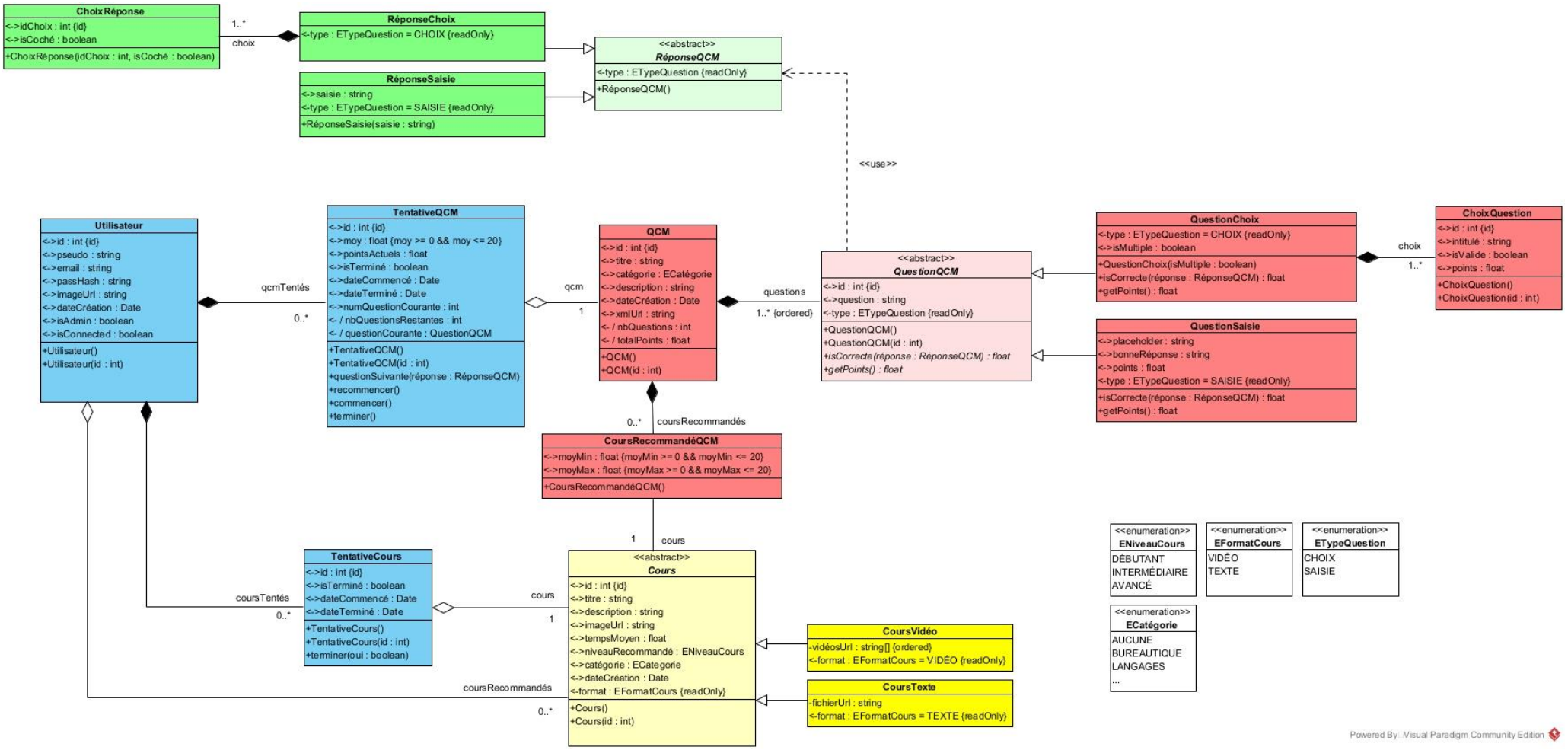

Image 5 : Diagramme de classes

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.

A. 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

B. 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.

C. 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 » possède 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 redéfinies 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 valent 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.

D. CLASSES EN LIEN AVEC LES REPONSES 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.

V. MODELE RELATIONNEL

Maintenant, que nous venons juste de voir le diagramme de classes, nous allons nous focaliser sur la conversion de 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

A. 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.

B. 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.

C. 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.

VI. 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.

A. 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
```

B. ARCHITECTURE MVC

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

1. MODELES

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. Attention, les modèles ne récupèrent pas les données depuis la base de données, ils servent juste à les stocker en mémoire, pour cela, nous verrons les classes de base de données peu après.

2. 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
<?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 1 : 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, 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.

3. CONTROLEURS

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 type que nous avons utilisé la plupart du temps :

```php
<?php

SessionManagement::session_start();

/* --------------------- Récupération des paramètres URL -------------------- */

$idCours = $_GET["id"] ?? null;

/* ------------------------------------------------------------------------- */
/*                              Vérifications                                */
```

```php
/* ------------------------------------------------------------------------ */

if (!$isAdmin) die("Vous devez être admin.");
if (!$idCours) die("Identifiant obligatoire.");

/* ------------------------------------------------------------------------ */
/*                          Récupération des données                        */
/* ------------------------------------------------------------------------ */

$crud  = new CoursCRUD();
$cours = $crud->getCoursById($idCours);


if (empty($_POST))
  showView();       // Afficher la vue si aucun formulaire envoyé ...
else
  handleForm();     // ... sinon, traiter le formulaire.


function showView()
{
  global $cours;
  afficherVue($cours); // Afficher la vue en lui passant l'instance du cours.
}

function handleForm()
{
  global $cours;

  // Traitement du formulaire...
  $_POST["..."];

  // Redirection avec confirmation du résultat...
  if ($ok)
    redirect("/cours", "success", "OK !", ["id" => $idCours]);
  else
    redirect("/cours", "error", "Erreur !", ["id" => $idCours]);
}
```

Fichier 2 : Exemple d'un fichier PHP d'un contrôleur type dans notre projet

C. CLASSES DE BASE DE DONNEES

Pour respecter le principe de responsabilité unique des classes, nous avons séparé la manipulation des données de la base de données des modèles. Ainsi, d'un côté, les modèles stockent uniquement les données en mémoire et peuvent gérer la logique de celles-ci, et de l'autre côté, nous avons des classes qui manipulent les données de la base de données à partir des modèles.

Nous avons une classe « DatabaseManagement » permettant d'ouvrir la connexion avec la base de données. Puis, nous avons les classes « CRUD » qui permettent de créer, lire, mettre à jour et supprimer des données de la base de données ; nous avons globalement, une classe « CRUD » par modèle.

Par exemple, nous avons une classe « CRUD » pour le modèle « Cours » avec une méthode de lecture qui retourne une ou plusieurs instances de ce modèle. Pareillement, pour mettre à jour à jour un cours, nous passons l'instance du modèle en paramètre de la méthode de mise à jour.

D. CLASSE DE GESTION DE SESSION

Pour plus facilement gérer les sessions PHP, nous avons développé une classe utilitaire « SessionManagement ». Cette classe possède des méthodes qui permettent entre autres de démarrer la session, fermer la session, récupérer l'instance de l'utilisateur connecté, permet de connaître si un utilisateur est connecté, ou s'il est admin.

E. CLASSES DE GESTION DE FICHIER

Dans le projet, à plusieurs reprises, il est possible d'importer des fichiers à partir de formulaires. Afin, de faciliter le traitement des importations dans les contrôleurs, nous avons créé des classes pour chaque type de fichier (PDF, XML, images). Ces classes permettent de vérifier si le fichier importé respecte le poids autorisé, est du bon format et possède la bonne extension. Elles permettent aussi de sauvegarder le fichier s'il respecte bien ces conditions.

F. CLASSE DE PARSER XML

Nous avons une classe qui permet d'analyser un fichier XML pour les questions de QCM avec l'API « SimpleXML » de PHP. Elle retourne les instances de « QuestionQCM » correspondantes.

VII. DETAILS D'IMPLEMENTATION

Après la vue d'ensemble des fonctionnalités, 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.

A. VERSION PHP ET MYSQL

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.

B. REECRITURES D'URL

Nous avons préféré ne pas utiliser de contrôleur principal qui permet d'accéder aux différentes 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 3 : « .htaccess » – Définition des réécritures d'URL

C. 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.

D. THEMES 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 » 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.

E. 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 le choix multiple est 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
<?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 4 : 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.

F. BASE DE DONNEES

Pour manipuler la base de données MySQL avec PHP, nous avons utilisé l'interface d'accès « PDO[2] ».

G. 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.

H. 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 :

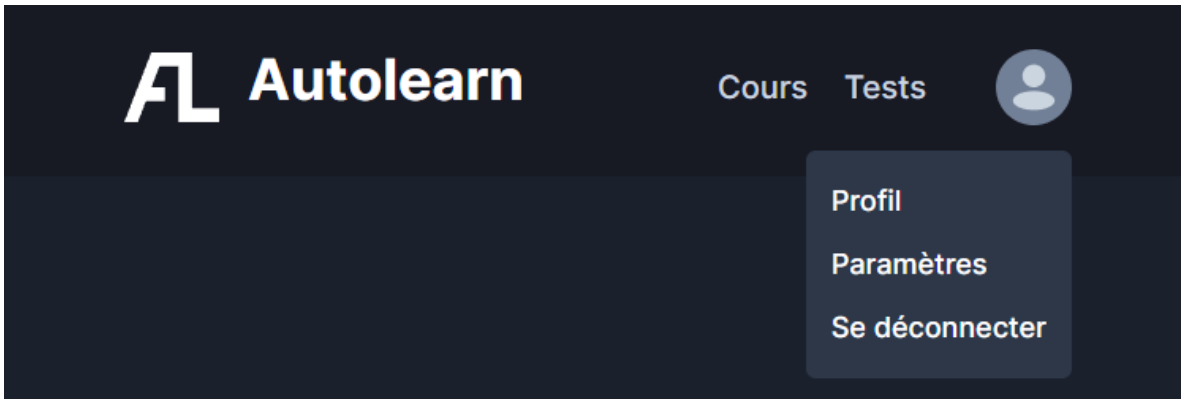

Image 6 : Menu « burger » s'ouvrant après un clique sur l'image de profil

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 :

[2] « PHP Data Objects » est une interface d'accès à une base de données avec PHP.

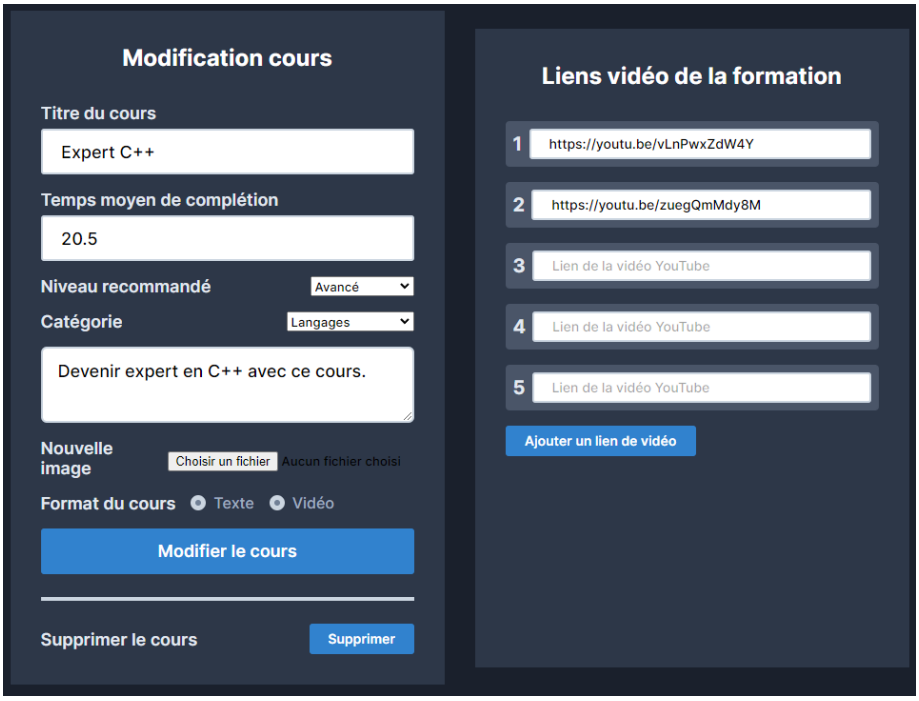

Image 7 : Ajout de liens vidéo YouTube supplémentaires dans la page d'édition d'un cours (panneau à droite)

Afin de faire cela, nous avons utilisé les événements 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.

I. AJAX, REQUETE 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 :

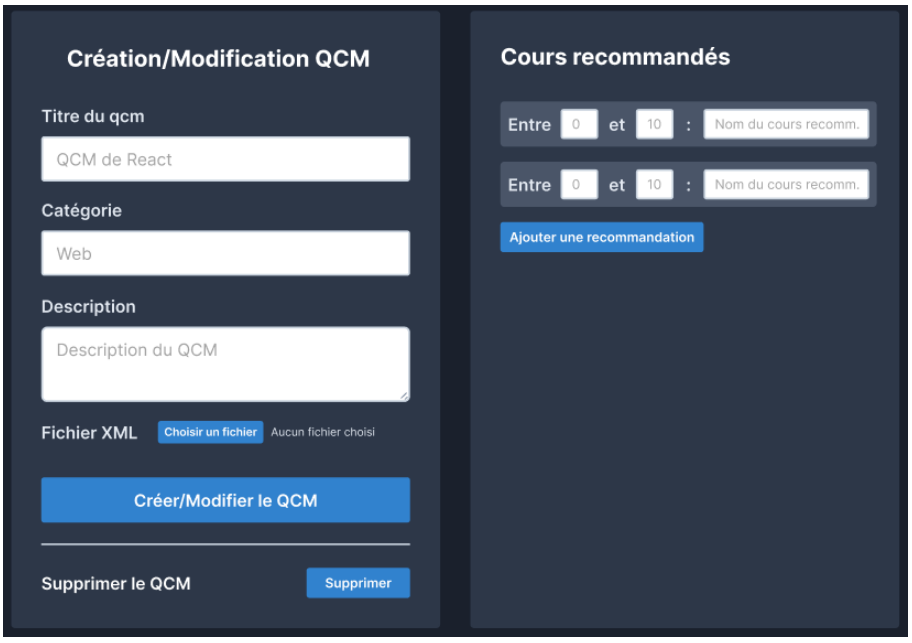

Image 8 : Choix d'un cours à recommander dans la page d'édition d'un QCM (panneau à droite)

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.

J. SECURITE

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 ?).

VIII. GESTION DE PROJET

Pour cette dernière partie, nous allons détailler notre gestion de projet avec la stratégie « Scrum ».

A. GESTIONNAIRE DE VERSIONS

Nous avons utilisé le gestionnaire de versions « GitHub » pour le projet.

B. OUTIL DE GESTION DE PROJET

Nous avons utilisé l'outil de gestion de projet « ClickUp » qui est une application web, et permet entre autres d'avoir un *workspace* privé où il est possible d'inviter d'autres personnes, de créer une planification de tâches, et de se répartir les tâches. En plus, il permet de gérer les dépendances, et les priorités. Nous avons choisi cet outil, car c'est l'un des outils gratuits le plus complet, et qu'il permet de gérer la stratégie Scrum.

C. GESTION DE LA COMMUNICATION

Nous allons vous expliquer comment nous nous sommes organisés pour la gestion de communication dans l'équipe.

1. MODELISATION UML / MAQUETTES

Afin d'assurer que toute l'équipe soit bien au fait des besoins et des fonctionnalités à implémenter, les diagrammes de cas d'utilisation nous ont bien été utiles. Également, sur un point plus technique, le diagramme de classes nous a permis de bien cerner l'architecture de l'application aussi bien pour ceux du front-end pour comprendre comment afficher les données, ou ceux du back-end pour implémenter les classes et les contrôleurs.

Par ailleurs, les maquettes de l'interface utilisateur ont été essentielles pour concrétiser les fonctionnalités, et garder une cohérence visuelle sur l'ensemble du site web.

La modélisation a été donc véritablement utile pour limiter les dérives, les contradictions et maintenir une bonne cohésion dans l'équipe.

2. MOYEN DE COMMUNICATION

Pour communiquer dans l'équipe aussi bien par message que pour les réunions, nous avons utilisé le logiciel « Discord » dans lequel nous avons créé notre propre groupe de discussion privé. Nous avons pu nous entraider avec des partages d'écran, débattre de certaines choses, etc.

3. RASSEMBLEMENT D'INFORMATIONS

Pour limiter le désordre, les malentendus, ou les incompréhensions dans l'équipe, nous avons centralisé les informations utiles dans « ClickUp ».

En effet, nous y avons créé des documents récapitulant les informations importantes que nous avons pu avoir en réunion, et en échange, ou tout simplement pour préciser des points techniques pour aider les autres membres à réaliser leurs tâches. Par exemple, nous avons listé les URL des pages, expliquer comment créer de nouvelles URL, comment configurer l'environnement de développement, ou comment sont structurées les contrôleurs et les vues.

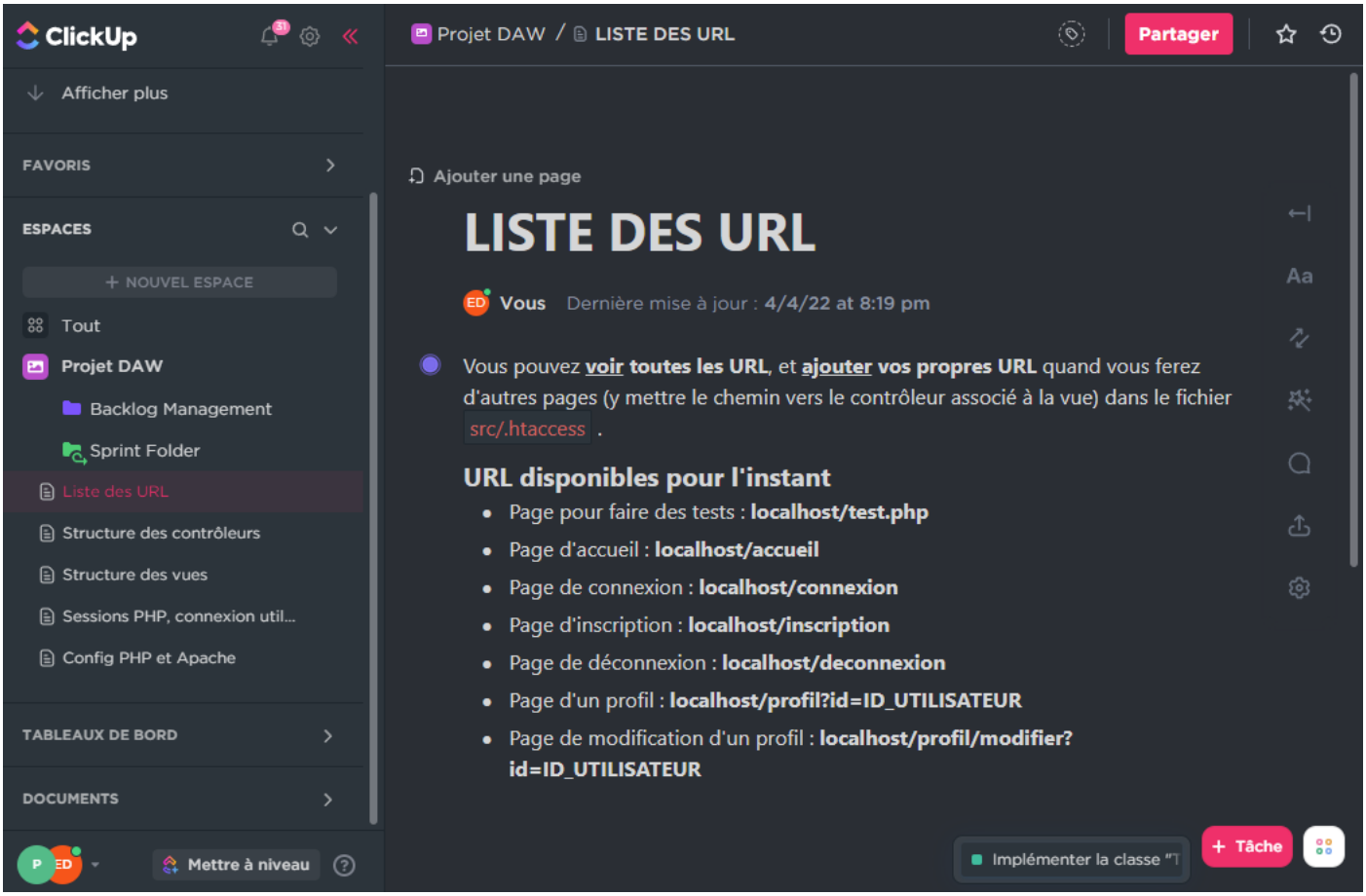

Image 9 : Documents (panneau à gauche) récapitulant les informations utiles pour l'équipe dans « ClickUp »

De plus, pour chacune des tâches, nous avons mis une description expliquant concrètement ce qu'il fallait réaliser, tout en fournissant des conseils, et des points techniques permettant la réalisation des tâches.

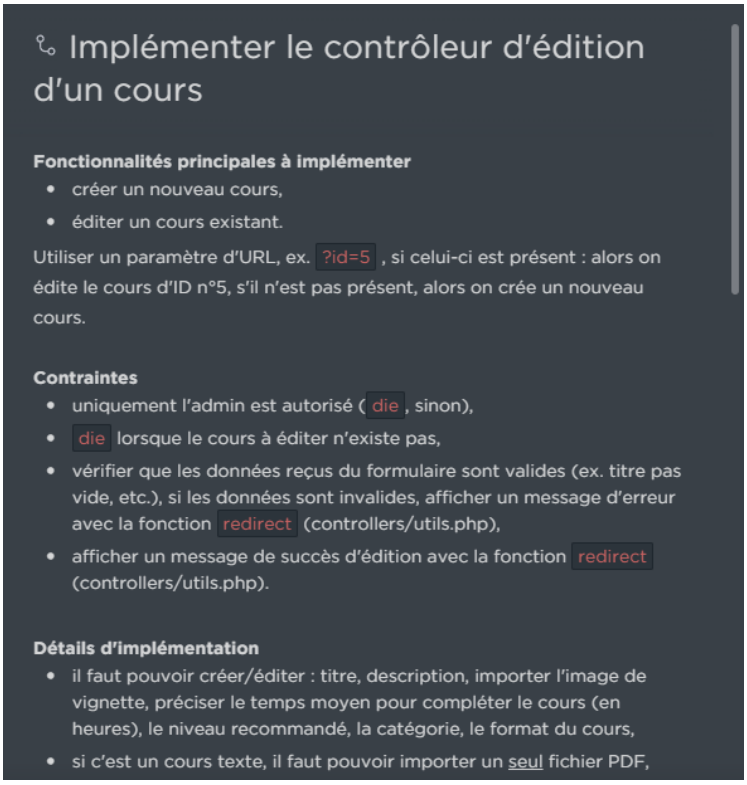

Image 10 : Description d'une tâche back-end expliquant ce qu'il faut implémenter, les contraintes à respecter, etc.

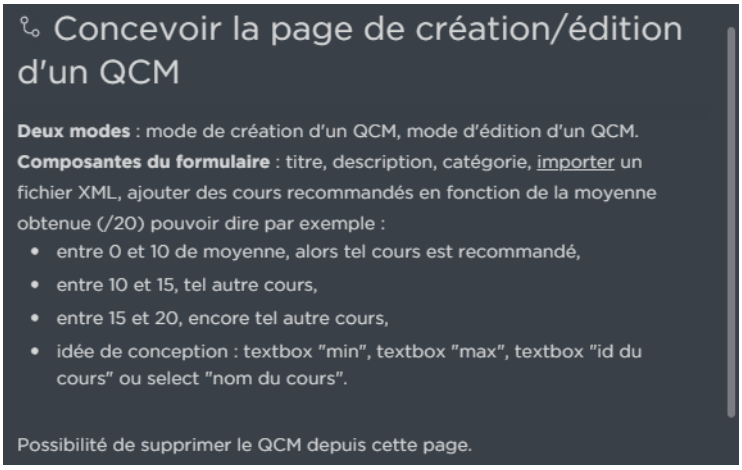

Image 11 : Description d'une tâche front-end expliquant ce qu'il faut implémenter

D. ORGANISATION SCRUM

Nous avons donc utilisé la stratégie de gestion de projet Scrum, et nous allons vous décrire comment nous avons géré ses principes et ses composantes.

1. PROJECT OWNER

Le « Product Owner » est Eddy, et il a d'abord été chargé de réaliser l'identification des besoins en réalisant une première version de la modélisation avec les diagrammes de cas d'utilisation, qu'il a ensuite communiqué à l'équipe pour leur apporter la vision du projet et qu'elle puisse accepter ou modifier cette première modélisation. Une fois, les besoins validés par l'équipe, il s'est occupé d'alimenter le *backlog* avec des « User Stories » décrivant simplement et de façon compréhensive les besoins.

De plus, Eddy s'est occupé de préparer le projet en créant le *workspace* ClickUp, le GitHub, et en préparant l'environnement de développement pour l'équipe.

Enfin, tout au long du projet, il a dû avoir un feedback régulier sur les tâches réalisées pour vérifier si elles respectaient la modélisation, les maquettes et les besoins, afin de faire des retours à l'équipe.

2. SCRUM MASTER

Le « Scrum Master » est Clément, et il a été chargé de préparer les réunions et de planifier les sprints. Il a aussi dû noter les domaines d'amélioration et les éléments à adopter pour les futurs sprints.

3. BACKLOG

Le backlog a été réalisé tout au début du projet après confirmation des besoins par l'équipe lors de la première réunion. Il contient les « User Stories » décrivant chaque fonctionnalité à réaliser pour la fin du projet. Ces « User Stories » ont été rangées dans quatre grandes catégories nommées « Epic Stories » :

- Gestion de la connexion
- Gestion du profil
- Gestion des cours
- Gestion des QCM

Image 12 : Contenu du backlog, « User Stories », et « Epic Stories »

E. REUNIONS DE SPRINT

Au total, nous avons eu quatre sprints de deux semaines, et pour chacun des sprints nous avons fait une réunion pour planifier et se répartir les nouvelles tâches tout en détaillant les points d'améliorations pour les prochains sprints.

Par ailleurs, pour chacun des sprints, après avoir choisi les fonctionnalités du backlog à réaliser pour le sprint, le *Product Owner* a créé les tâches techniques (ex. développer telle page, tel contrôleur, implémenter telle classe, etc.), ce sont ces tâches qui ont été réparti entre les membres.

Ci-dessous, le compte-rendu de chacune des réunions de début de sprint.

1. SPRINT N°1 – 14/03/2022 AU 27/03/2022

Cette première réunion a tout d'abord servi à présenter le projet, réfléchir à l'environnement de développement, répartir les tâches pour le premier sprint au sein de cette stratégie de gestion de projet Scrum et de réfléchir aux moyens de résoudre les fonctionnalités demandées.

Au début de la réunion, le *Product Owner* a présenté la première modélisation des besoins du projet et a demandé à l'équipe s'ils avaient des points à améliorer ou à clarifier.

Ensuite, le *Scrum Master* à commencer à planifier les tâches pour le premier sprint avec l'équipe en prenant des fonctionnalités du *backlog*. Nous avons décidé de commencer par la gestion de la connexion et la gestion du profil puisque c'était les fonctionnalités de base du site web et les fondements des autres fonctionnalités, nous avons donc décidé par priorité et dépendances. Par la suite, les membres de l'équipe se sont concertés pour se répartir les tâches en fonction de leurs compétences et leurs préférences.

En fin de réunion, le *Product Owner* a expliqué à l'équipe comment installer l'environnement de développement via un partage d'écran.

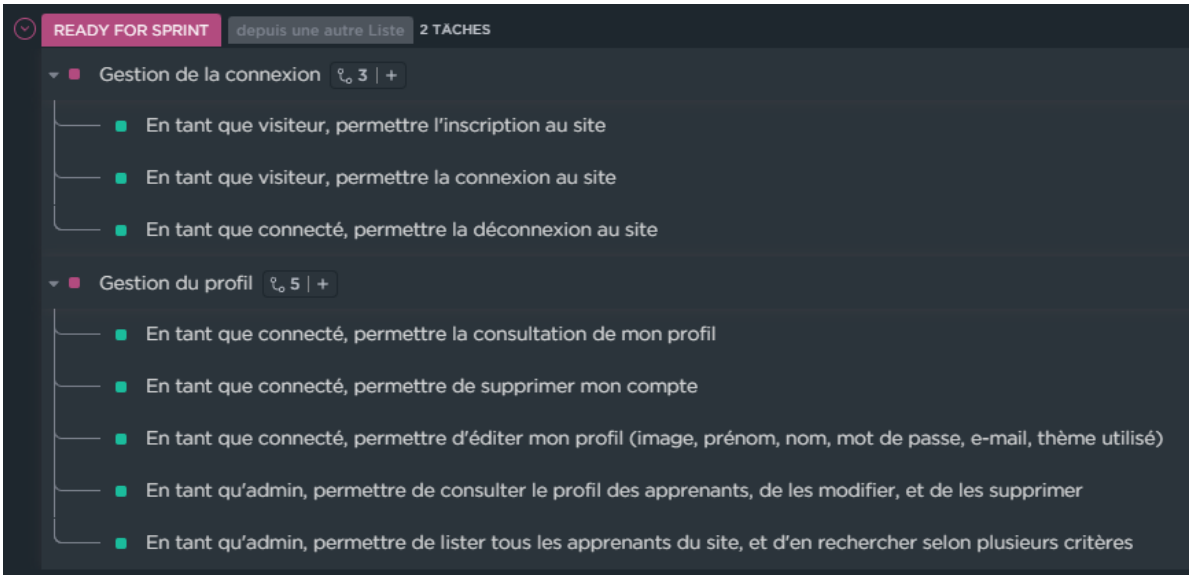

Image 13 : Sprint n°1 – Fonctionnalités à réaliser

Image 14 : Sprint n°1 – Planification et répartition des tâches techniques (1/4)

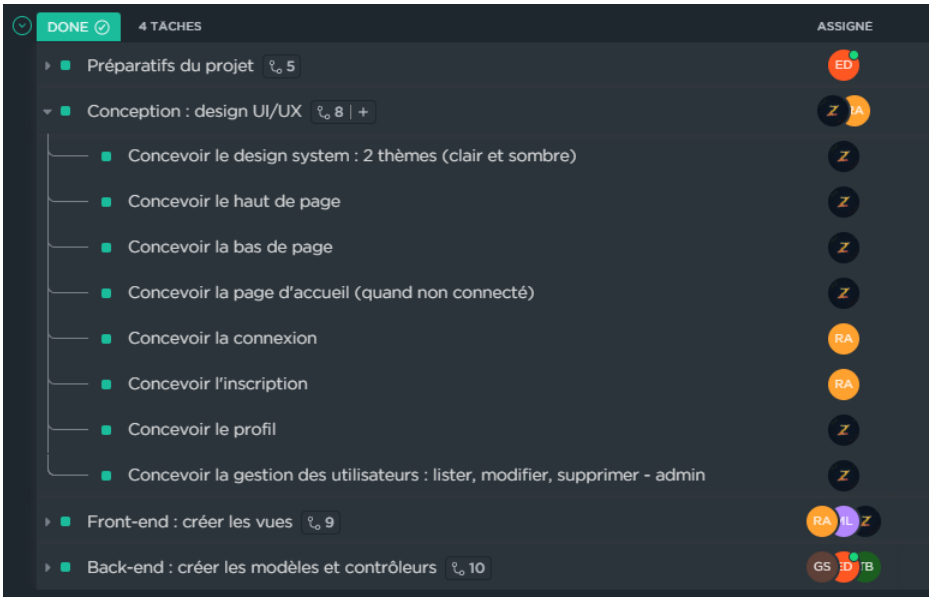

Image 15 : Sprint n°1 – Planification et répartition des tâches techniques (2/4)

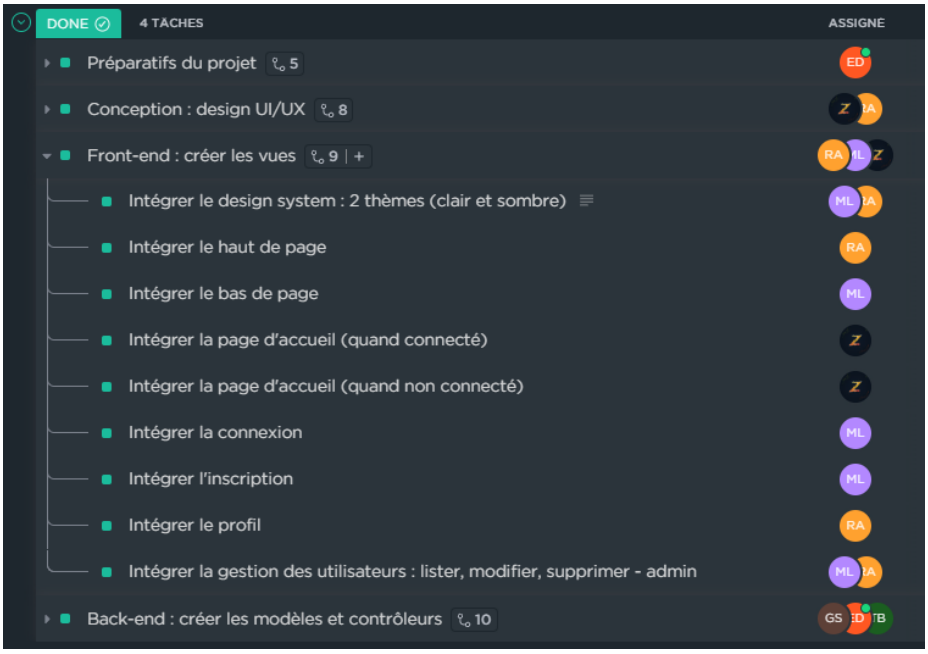

Image 16 : Sprint n°1 – Planification et répartition des tâches techniques (3/4)

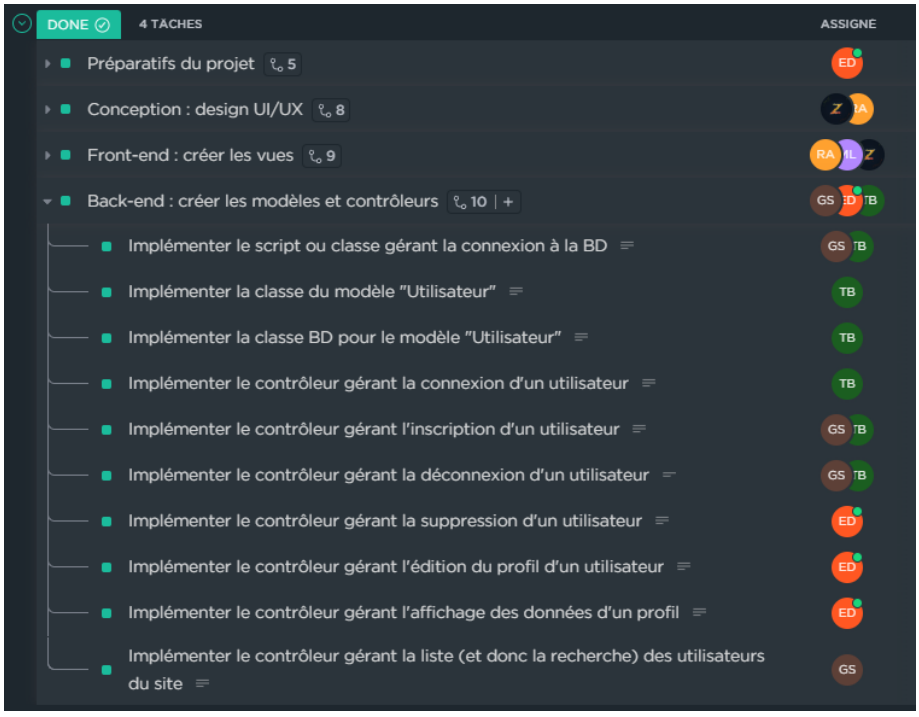

Image 17 : Sprint n°1 – Planification et répartition des tâches techniques (4/4)

2. SPRINT N°2 – 28/03/2022 AU 10/04/2022

Pour cette deuxième réunion, nous avons fait un récapitulatif du dernier sprint concernant les points à améliorer et nous avons planifié les tâches du deuxième sprint.

Au début de la réunion, le *Product Owner* et le *Scrum Master* ont fait part des tâches qui ne respectaient pas les critères d'acceptation (non-respect des maquettes, pas en adéquation avec les besoins décrits, bugs, etc.). Toutes les tâches du premier sprint ont été terminées avant le début du deuxième.

Par la suite, nous avons planifié et nous nous sommes réparti les tâches pour le deuxième sprint, et nous avons choisi uniquement la gestion des cours, car cette partie était assez conséquence pour remplir les deux semaines.

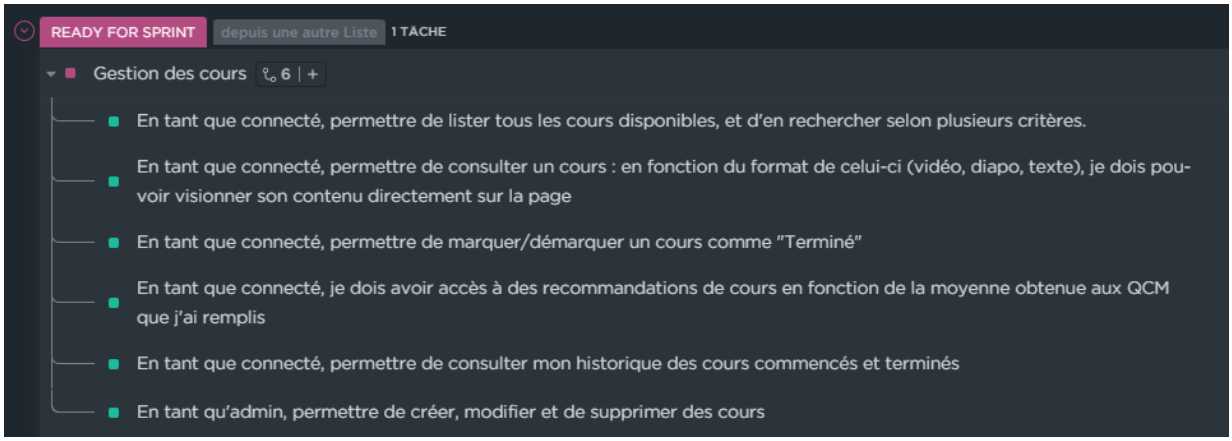

Image 18 : Sprint n°2 – Fonctionnalités à réaliser

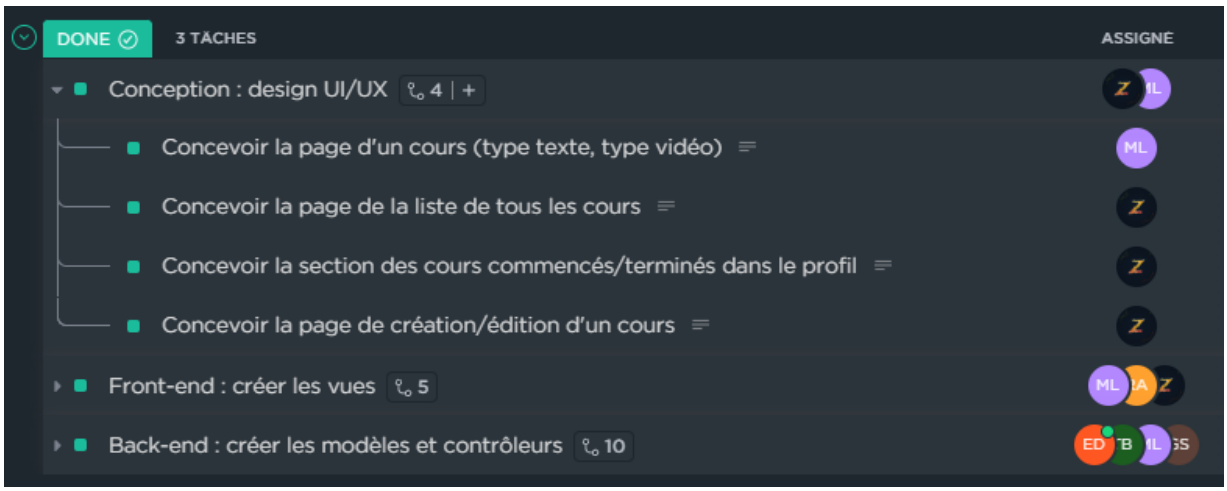

Image 19 : Sprint n°2 – Planification et répartition des tâches techniques (1/3)

Image 20 : Sprint n°2 – Planification et répartition des tâches techniques (2/3)

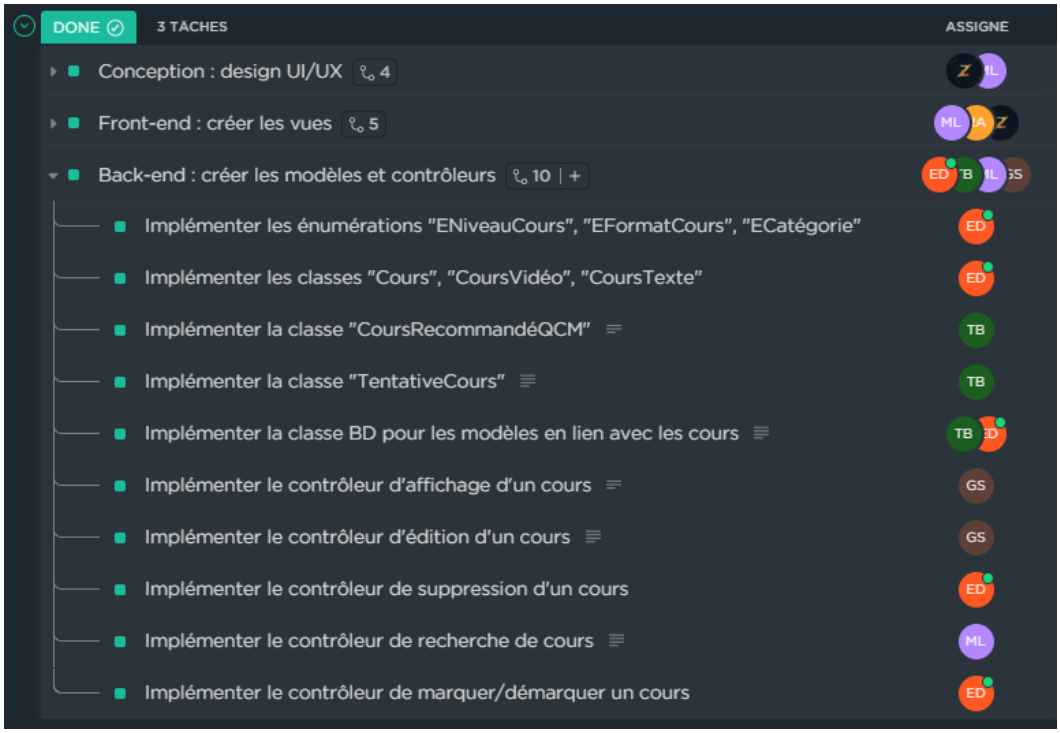

Image 21 : Sprint n°2 – Planification et répartition des tâches techniques (3/3)

3. SPRINT N°3 – 11/04/2022 AU 24/04/2022

Pour cette troisième réunion, comme pour les précédents sprints, nous avons fait un récapitulatif et nous avons planifié les nouvelles tâches.

Certaines tâches du dernier sprint n'ont pas été terminées à temps, faute de disponibilités de certains membres, et elles ont donc été transférés pour le troisième sprint, nous restant encore quatre semaines pour boucler le projet, ce n'était pas très important.

Cette fois-ci, nous avons choisi de s'occuper des dernières fonctionnalités du *backlog*, et c'est la gestion des QCM. Celle-ci est la dernière à être faite, puisqu'elle reposait sur les trois autres parties.

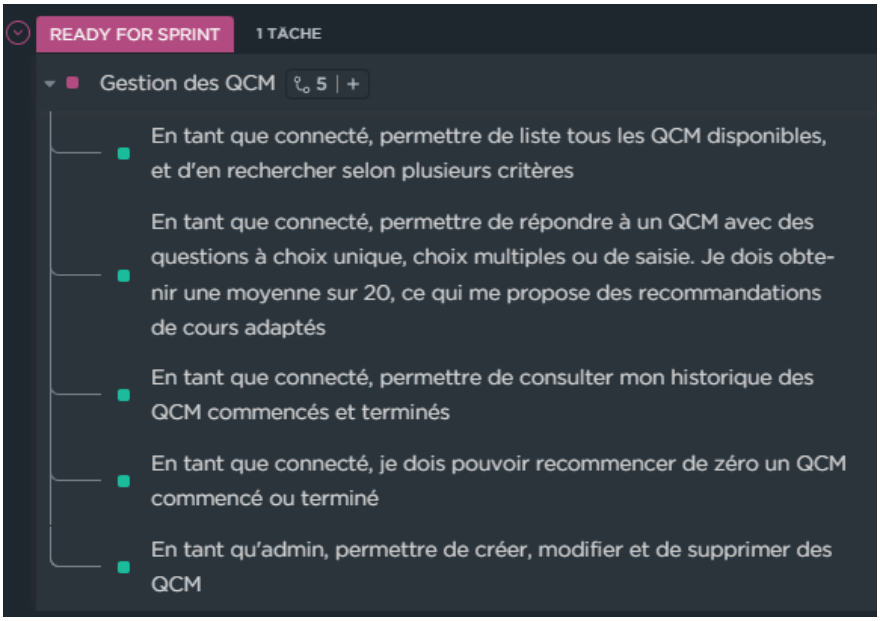

Image 22 : Sprint n°3 – Fonctionnalités à réaliser

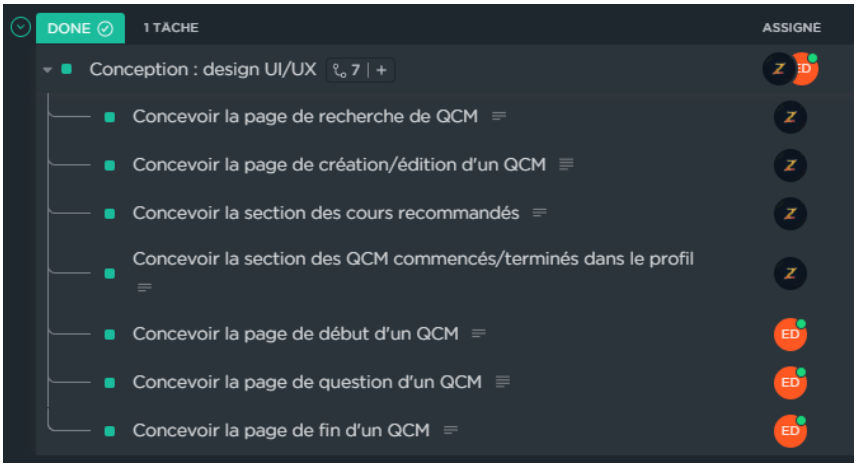

Image 23 : Sprint n°3 – Planification et répartition des tâches techniques (1/3)

Image 24 : Sprint n°3 – Planification et répartition des tâches techniques (2/3)

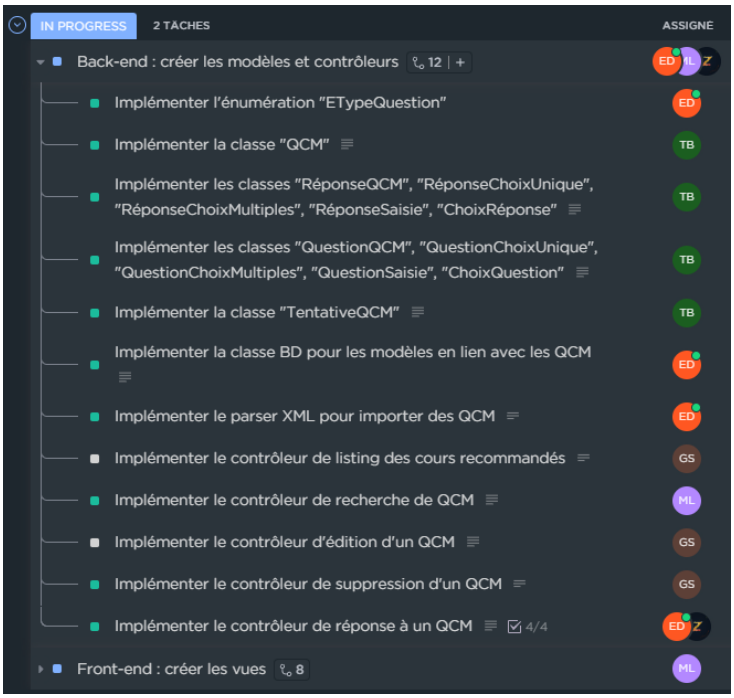

Image 25 : Sprint n°3 – Planification et répartition des tâches techniques (3/3)

4. SPRINT N°4 – 25/04/2022 AU 08/05/2022

Pour cette dernière réunion, nous avons fait un dernier récapitulatif de l'ensemble du projet pour vérifier les détails à peaufiner ou à améliorer, et que tout était conforme aux besoins, à la modélisation, et aux maquettes.

Nous nous sommes réparti les bugs à corriger, et nous avons transféré les tâches du précédent sprint qui n'étaient pas entièrement terminées pour ce dernier sprint.

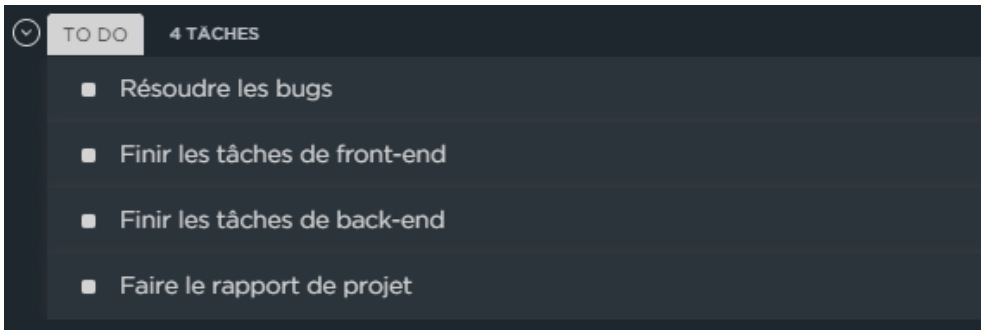

Image 26 : Sprint n°4 – Tâches de finitions

F. PLANIFICATION DES TACHES

Nous allons vous expliquer comment nous avons géré la planification des tâches, sur quels critères, et de quelles manières.

1. ESTIMATION DU TEMPS DE REALISATION

Nous avons commencé le projet le 14 mars 2022, et la date de rendu est aux environs du 8 mai 2022, et en comptant environ 3 jours par semaine dédiés à la réalisation du projet (jours libres en semaine et week-ends) ainsi que la quasi-totalité des jours de la semaine à partir du 16 avril 2022 (date où nous avons plus cours dans l'emploi du temps) jusqu'à la date de rendu : cela nous fait environ 30 jours, soit 240 heures pour réaliser l'entièreté du projet.

Nous avons donc défini une mesure qui sont les « points de sprint », où 1 point correspond à une demi-journée (4 heures). Nous avons donc eu un total de 240 heures / 4 heures = 60 points à répartir dans les tâches du *backlog*. Ci-dessous, la répartition des points dans les « User Stories » du *backlog*.

Image 27 : Répartition des points de sprint dans les tâches du backlog (colonne à droite)

Nous avons donc consommé 58 points de sprint pour l'estimation de la réalisation de l'ensemble de ces tâches. Nous n'avons donc pas le temps de réaliser un forum de discussion avant la fin du projet, c'est pourquoi, nous l'avons omis.

Concernant le choix du nombre de points attribués, nous avons raisonné par le fait d'attribuer plus de points à des tâches demandant de réaliser une fonctionnalité complexe comme dans les cas où : il y a beaucoup de pages web à développer ou s'il elles sont compliquées en termes de contenu, il y a de gros formulaires et donc des contrôleurs conséquents à développer, etc.

Enfin, l'estimation du temps de réalisation des tâches a été faite par le *Product Owner* puisqu'il était plus apte à l'estimer compte tenu de ses connaissances et son expérience avancée dans le développement web.

2. DECOUPAGE DES FONCTIONNALITES PAR SPRINT

Comme vous l'avez vu dans le *backlog*, nous avons quatre grandes catégories de tâches, ce sont les « Epic Stories ». Les deux premières « Gestion de la connexion » et « Gestion du profil » étant assez rapide à réaliser en regardant l'estimation en points de sprint par rapport aux deux dernières catégories « Gestion des cours » et « Gestion des QCM », nous avons décidé de dédier le premier sprint, aux deux premières catégories ; le deuxième et troisième sprint aux deux dernières catégories ; et le quatrième sprint est réservé pour finir les éventuelles tâches non terminées, et faire les finitions de dernières minutes.

3. TÂCHES TECHNIQUES

Les tâches techniques (ex. développer telle page, tel contrôleur, implémenter telle classe, etc.), ont été créées durant les réunions par le *Product Owner* en lien avec les « User Stories » à réaliser. Ce sont ces tâches qui ont fait office d'une répartition entre les membres de l'équipe.

Ces tâches sont catégorisées en trois catégories :

- Tâches de conception des maquettes
- Tâches de front-end, avec création des vues en intégrant des maquettes
- Tâches de back-end, avec implémentation des classes et des contrôleurs

Les dépendances entre les tâches ont été définies.

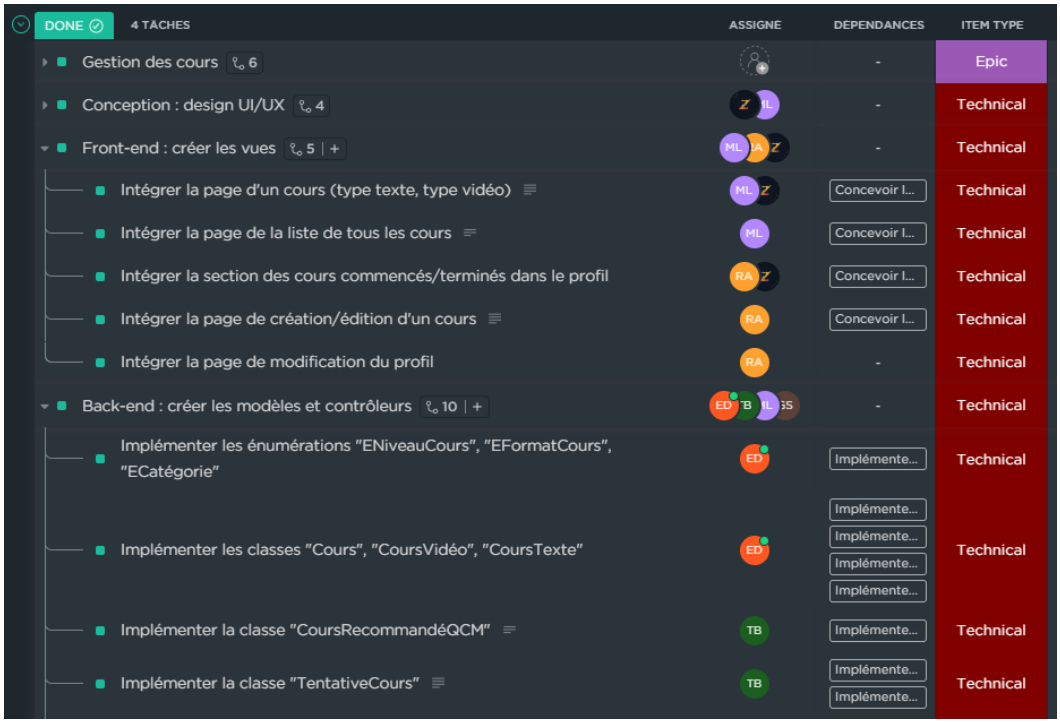

Image 28 : Tâches techniques du deuxième sprint avec les assignations et dépendances

Par ailleurs, comme précisé dans la partie IX.C.3 – Rassemblement d'informations, pour chacune de ces tâches, il y a une description complète sur ce qu'il y a à implémenter, les contraintes à respecter, etc.

4. SYSTEME DE SUIVI DES BUGS

Avec ClickUp, nous avons pu ajouter des bugs à résoudre, et les suivre en les affectant à des membres. Cela était très utile, puisque nous avions juste à ajouter les bugs que nous trouvions sur le site web. Par ailleurs, il était possible d'ajouter une priorité concernant les bugs à corriger.

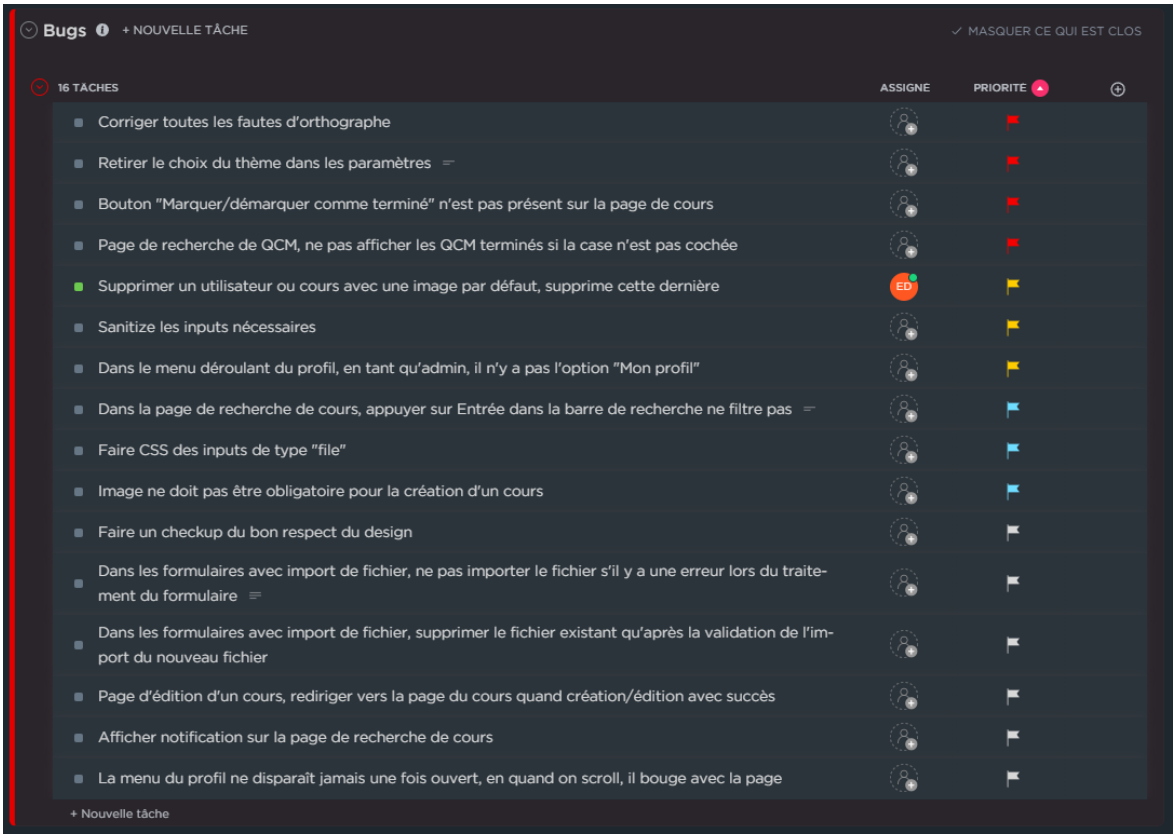

Image 29 : Suivi des bugs

G. REPARTITION DES TACHES

Concernant la répartition des tâches, nous nous sommes naturellement répartis en deux groupes : ceux qui font du front-end, et ceux qui font du back-end. Ceci a été fait en fonction des compétences et des préférences de chacun.

Du côté front-end, nous avons :

- Rémy AULOY
- Lilian MANZANO
- Clément GILI

Du côté back-end, nous avons :

- Eddy DRUET
- Guillaume SONVICO
- Tom BARBIER

Cette répartition est au final assez équitable, la proportion étant égale pour les deux côtés.

IX. CONCLUSION

Nous arrivons à la fin de ce rapport, et nous pensons avoir utilisé le maximum de principes et thèmes étudiés dans le module de « Développement d'application web » dans ce projet. Concernant la modélisation, nous avons décrit les besoins avec les diagrammes de cas d'utilisation, nous avons modélisé la logique métier avec le diagramme de classes, et nous avons réalisé des maquettes des pages web en amont avant de commencer le développement ce qui a grandement aidé pour la cohésion d'équipe. Enfin, concernant la gestion de projet, nous pensons avoir exploité au mieux de nos capacités la stratégie Scrum puisque globalement, toute l'équipe a travaillé équitablement et relativement sans encombre grâce à une centralisation des informations nécessaires à la réalisation des tâches sur « ClickUp ».

DIAGRAMMES DE CAS D'UTILISATION

IMAGES

FICHIERS