
Une application web en Bootstrap et Symfony :
Soutien scolaire en ligne

Rapport du TER

Etudiants :

DESBOS Marie-Lou
GARCIA Léa
HARTI Abdellah
LUO Stephen

Encadrant :

Mr. Abdelhak-Djamel SERIAI

Table des matières

1	Présentation du sujet	3
1.1	Introduction	3
1.2	Motivations en lien avec notre parcours scolaire	3
1.3	Cahier des charges	3
2	Étude du domaine métier et technologique	5
2.1	Métier	5
2.2	Technologies utilisées	5
2.2.1	Contexte	5
2.2.2	Architecture et initiation à Symfony	6
2.2.3	Bootstrap	8
3	Analyse et Conception	11
3.1	Les maquettes	11
3.2	Diagramme de Use-case	12
3.3	Diagramme de séquences	12
3.4	Diagramme de classes	12
4	Développement logiciel	16
4.1	Fonctionnalités de l'interface graphique	16
4.1.1	La page d'accueil	16
4.1.2	Le compte des étudiants	18
4.1.3	Le compte des enseignants	19
4.1.4	Le compte des parents	21
4.1.5	Le compte des administrateurs	22
4.2	L'éditeur de cours	22
4.2.1	Mustache	22
4.2.2	Les modals	23
4.2.3	Le corps de l'éditeur de cours	23
4.2.4	Récupération du contenu JSON et affichage lors d'une modification	23
4.3	Bootstrap	24
4.4	Intégration à l'aide de Twig	25
4.5	Utilisation de la base de données avec les Forms	25
4.6	Les sessions	26

5	Gestion de projet	28
5.1	Organisation générale	28
5.2	Outils utilisés	28
5.2.1	Google Drive	28
5.2.2	Overleaf	28
5.2.3	GitLab	28
5.2.4	PhpStorm	29
5.2.5	Figma	29
5.3	Organisation des tâches dans le temps	30
6	Conclusion	31
7	Bibliographie	32
8	Annexes	33
8.1	Mode d'Emploi	33
8.2	Les maquettes	34

1 Présentation du sujet

1.1 Introduction

Dans le cadre du TER, module HLIN601, du second semestre de troisième année de licence d'informatique, nous avons choisi de travailler sur le développement d'une application web pour le soutien scolaire en ligne. Cette application doit être réalisée en Bootstrap et Symfony, des technologies très utilisées dans les entreprises de nos jours.

Nous formons un groupe composé de Marie-Lou Desbos, Léa Garcia, Abdellah Harti et Stephen Luo. Notre encadrant est Monsieur Abdelhak-Djamel Seriai, que nous remercions d'avoir su nous guider et bien nous expliquer les attentes du sujet, tout au long de ce projet.

1.2 Motivations en lien avec notre parcours scolaire

Durant cette troisième année de licence, nous avons été amenés à travailler des langages de front-end et back-end mais nous n'avions jamais eu l'occasion d'utiliser des frameworks tels que Bootstrap et Symfony. Si nous avons choisi ce projet c'est d'une part pour pouvoir approfondir nos connaissances dans les langages informatiques utilisés par ces frameworks mais aussi pour nous permettre découvrir les outils utilisés par les professionnels du développement d'applications web.

1.3 Cahier des charges

Objectif L'objectif de ce projet de TER est de développer une application web de soutien scolaire en ligne.

Contraintes Cette application est à développer en utilisant Bootstrap pour la création de l'interface graphique (Front-end) et Symfony pour créer la partie serveur (Back-end).

Besoins fonctionnels Cette application devra gérer 5 types d'utilisateurs :

1. Les étudiants

Ils devront pouvoir s'inscrire, pour cela ils devront choisir un niveau puis une filière puis choisir s'ils veulent suivre une, plusieurs ou toutes les matières du niveau. Ils pourront se connecter pour accéder à leur compte et consulter le contenu disponible (voir les pdfs, vidéos et cours ainsi que faire les quizz) pour les matières choisies lors de l'inscription. Ils pourront éventuellement faire évoluer leur abonnement.

2. Les parents

Ils devront pouvoir s'inscrire, se connecter et inscrire leur(s) enfant(s) sur le site. Ils seront donc liés au compte de leur enfant et devront avoir accès aux mêmes ressources que lui (eux) (cours, quizz, vidéos et pdfs) et pourront également consulter son (leur) historique (temps de connexion, notes de quizz).

3. Les utilisateurs anonymes

Ils devront pouvoir : Voir une vidéo résumant les fonctionnalités de l'application, faire une navigation réduite sur le site et s'inscrire en tant qu'étudiant, parent ou enseignant.

4. Les enseignants

Ils devront pouvoir s'inscrire et se connecter.

Un enseignant est le créateur de contenu du site. Il doit être capable d'ajouter du contenu qui sera consultable et/ou manipulable, en fonction du type de contenu, par les étudiants. Il y a quatre types de contenu : les pdf, les vidéos, les cours (template) et les quizz. L'enseignant ne devra voir que les matières pour lesquelles il intervient.

Afin d'homogénéiser le contenu du site importé par les enseignants, il faudra créer un éditeur de cours grâce à un formulaire et un template (un modèle) pour qu'un enseignant puisse créer une page html+css sans pour autant qu'il sache comment programmer. La même chose devra être réalisé pour les quizz, l'enseignant aura accès à un éditeur de quizz qui par la suite permettra d'afficher dans un template les questions et les réponses possibles pour que les étudiants puissent y répondre. Les enseignants pourront voir et noter les quizz réalisés par les élèves.

Les enseignants pourront s'inscrire et se désinscrire eux-mêmes des matières auxquelles ils enseignent.

5. Les administrateurs

Ils devront pouvoir se connecter, inscrire des enseignants, recevoir des alertes d'ajout de contenu.

Ils seront capables de le consulter, ensuite de le valider, de le refuser ou bien de demander une rectification du contenu. Il peut aussi voir, supprimer et modifier les contenus déjà ajoutés par les enseignants. Enfin, ils géreront la liste des matières, des niveaux et des filières et pourront ajouter des types de questions pour les quizz..

2 Étude du domaine métier et technologique

2.1 Métier

La formation en ligne est un ensemble de technologies permettant l'apprentissage depuis chez soi sans devoir se déplacer au lieu d'assister aux cours en présentiel. Bien que ces technologies existent dans une certaine mesure depuis de nombreuses années, elles étaient plus souvent considérées comme des fonctionnalités pratiques que comme des outils essentiels à l'apprentissage en ligne. Même les plateformes en ligne telles que Moodle étaient surtout utilisées pour consulter les annonces, voir les horaires des cours et télécharger occasionnellement des diapositives.

Avec l'apparition de Covid-19, la formation en ligne a connu un tournant drastique avec beaucoup plus d'accent mis sur l'apprentissage ; cela peut typiquement être divisé en deux grandes catégories.

- Si la communication entre l'enseignant et les étudiants se fait en direct (par exemple par un appel Zoom), c'est ce qu'on appelle l'apprentissage «synchrone».
- En revanche, s'il y a un délai dans la communication (par exemple, une vidéo est téléchargée et est visionnée plus tard lorsque l'élève a le temps), il s'agit d'un apprentissage «asynchrone». Notre application de site web scolaire se concentre principalement sur ce domaine.

En général, outre les raisons sanitaires, l'avantage d'adopter l'apprentissage en ligne réside dans sa flexibilité en termes de programmation. Si les cours sont asynchrones, les leçons peuvent être consultées quand on le juge opportun. Il n'y a pas de temps perdu dans les déplacements, et si une leçon en direct est manquée pour une raison quelconque, elle peut être visionnée plus tard, à condition qu'elle ait été enregistrée pendant la session en direct.

Bien entendu, il convient également de noter que le succès de l'apprentissage en ligne en tant qu'outil dépend en partie de la motivation des étudiants et de la qualité de leur connexion Internet.

2.2 Technologies utilisées

2.2.1 Contexte

Les technologies choisies pour la réalisation de ce projet font partie des plus utilisées actuellement mais d'autres frameworks de développement back-end et front-end existent et ont leur propres avantages.

Frameworks back-end

Les frameworks de développement back-end sont nombreux et se différencient par un langage qui leur est propre. Par exemple, on retrouve le framework Spring pour le langage Java, Django pour le langage Python ou encore Ruby on Rails pour le langage Ruby. Cependant, le langage PHP reste majoritaire pour le développement côté serveur et est utilisé par des frameworks tels que Laravel, CodeIgniter et Symfony.

Les trois frameworks énoncés pour le langage PHP fonctionnent tous sur le modèle d'architecture MVC que nous détaillerons dans la prochaine partie. Ils se ressemblent donc fortement mais ont chacun leurs avantages et leurs inconvénients. Laravel est le plus utilisé dans le monde actuellement car il permet de gérer des applications web complexes plus facilement et plus rapidement que les autres frameworks et car il permet une grande liberté d'organisation du code. CodeIgniter quant à lui, est un framework léger et puissant qui est souvent utilisé pour les débutants en PHP car il est facile à prendre en main. Enfin, Symfony est puissant, sécurisé et flexible, tout en imposant un mode de développement moins libre ce qui rend au final, le code plus propre comme nous avons pu le découvrir tout au long de ce projet.

Frameworks front-end

En ce qui concerne les frameworks front-end, nous pouvons les diviser largement en deux catégories distinctes : les frameworks CSS, qui sont principalement orientés vers le design et se concentrent sur les modèles d'interface utilisateur interactifs (et qui peuvent être accompagnés de certaines fonctions JavaScript), et les frameworks JavaScript, qui dépendent fortement du code JavaScript pour leur comportement.

Les frameworks CSS

Parmi tous les frameworks CSS disponibles, Bootstrap est le plus populaire. Environ 1 site web sur 5 utilise Bootstrap, y compris Twitter (son fondateur), Spotify et Lyft. Il met à disposition un large panel de mises en page, de thèmes, de panneaux d'administration et de support communautaire. Il offre également une grande variété de composants d'interface utilisateur tels que la navigation, les formulaires, les cartes, les boutons et les alertes.

L'un des plus grands concurrents de Bootstrap est Foundation. Contrairement à Bootstrap, Foundation donne à ses développeurs front-end un contrôle total sur leurs interfaces utilisateur, ce qui signifie qu'il s'agit d'un framework beaucoup plus flexible que Bootstrap. Cependant, l'inconvénient de cette flexibilité est qu'il est plus compliqué de d'apprendre à l'utiliser, notamment pour les nouveaux utilisateurs.

Bulma est un ajout relativement récent dans le monde des frameworks CSS, ayant fait son entrée sur la scène en 2019. Malgré le fait qu'il soit récent, il monte rapidement en terme de popularité (avec plus d'étoiles sur Github que Foundation). Il a une approche stricte uniquement CSS, sans composants JavaScript, et possède des classes CSS extrêmement lisibles.

Les frameworks JavaScript

Les frameworks Javascript ne faisant pas partie de notre projet, nous n'allons pas trop les détailler. Mais sachez que les trois plus populaires sont React, VueJS et Angular.

React a démarré comme un projet interne à Facebook, et permet la ré-utilisation des composants ce qui facilite la collaboration entre les équipes. Cependant, en raison des mises à jour constantes du framework, la documentation appropriée est en retard, de sorte que React a une courbe d'apprentissage plus raide.

La nature robuste d'Angular signifie que des applications web extrêmement complexes peuvent être développées à l'échelle de l'entreprise, mais qu'elles peuvent ne pas convenir aux projets de portée plus limitée ou aux équipes plus restreintes.

VueJS a été créé pour être une version plus minimaliste d'Angular et donc il est simple, direct et de petite taille. Pour cette raison, et grâce à son excellente documentation, VueJS est facile à apprendre pour les débutants.

2.2.2 Architecture et initiation à Symfony

Symfony est un framework MVC libre, un ensemble de composants PHP, écrit en PHP. Le but de ce framework, comme tous les autres, est de faciliter et d'accélérer le développement d'applications web. La première version date de 2005 et a été lancée par l'agence web Française SensioLabs.

Avant ce projet, aucun d'entre nous n'avait eu l'occasion d'utiliser de framework PHP. Grâce aux formations et contenus en ligne dont nous avons parlé dans la partie précédente, nous avons appris à nous servir de cet outil de développement qui s'est révélé extrêmement utile pour les tâches longues et répétitives qui reviennent fréquemment lors de la création d'une application web.

Dans cette partie nous allons tout d'abord expliquer ce qu'est Symfony, les installations nécessaires, son architecture, son squelette et enfin deux commandes importantes.

Les installations nécessaires à Symfony

Afin de bien pouvoir utiliser le framework, il est nécessaire d'avoir installé certaines choses. Il faut un environnement de serveur local qui va permettre de tester l'application web en local. Nous avons choisi MAMP pour windows et celui-ci propose un serveur MySQL, un serveur HTTP, la dernière version de PHP. Il faut également installer Doctrine, la couche d'abstraction utilisée par Symfony qui va permettre de

faire le mapping entre les entités et la base de données du serveur local. Enfin, il est nécessaire d'installer Composer qui sert à installer les dépendances de Symfony ainsi que les composants PHP extérieurs dont on peut avoir besoin. Pour notre projet nous en avons installé et utilisé un certain nombre et c'est grâce au fichier `composer.lock` que l'on est certain de pouvoir tous les ré-installer à chaque fois qu'il le faut.

Une architecture avec MVC (Modèle Vue Contrôleur)

Symfony impose une division du code source avec le design pattern MVC. C'est un modèle d'architecture qui divise le code source d'une application web en trois parties. Cela permet par exemple, lorsque l'on modifie l'interface graphique, de ne pas toucher à la logique de l'application ni aux structures des données qui seront affichées. Voici le rôle de chacune de ces trois parties :

1. Modèles

Le modèle représente la structure des données, il est éloigné de la logique et de l'affichage. Le modèle reçoit des demandes de données de la part du contrôleur, il s'occupe de rassembler les données demandées et de les renvoyer au contrôleur.

2. Vues

Les vues servent d'interface graphique, elles contiennent tout le design de l'application mais également les données à afficher. Les vues reçoivent des ordres d'affichage de données de la part des contrôleurs ou elles leur envoient les requêtes http émises par le client via l'interface graphique. Lorsqu'un contrôleur demande à une vue d'afficher les données, celle-ci envoie au client la page html (qui correspond à la requête émise au départ par le client). Les vues sont écrites en html et twig et ont donc l'extension ".html.twig"

3. Contrôleurs

Les contrôleurs sont les intermédiaires entre les vues et le modèle. Lorsqu'un client émet une requête http via une vue, celle-ci est dirigée vers un contrôleur qui s'occupera de faire une demande au modèle pour récupérer les données qu'il lui faut et qui s'occupera d'ordonner à la vue de répondre au client avec les bonnes données.

Commandes essentielles à Symfony

Création d'une entité Pour créer une entité il faut utiliser la commande "`php bin/console make :entity`". Il faut nommer l'entité puis dans une boucle, il nous est demandé d'écrire le nom de l'attribut, son type ou éventuellement sa relation avec un autre attribut d'une autre table (ManyToMany, ManyToOne, OneToMany). La création de cette entité va créer un fichier entité dans "Entity", un fichier formulaire dans "Form" ainsi qu'un répertoire dans "Repository". Les entités sont vues comme des objets et c'est l'ORM qui s'occupera de les persister dans notre base de données.

Mettre à jour la base de données Après avoir créé les entités, il faut les enregistrer dans notre base de données, pour cela on utilise le DoctrineMigrationBundle et la commande "`php bin/console doctrine:migrations:generate`", qui va créer une migration contenant toutes les requêtes SQL nécessaires pour ajouter ou mettre à jour la base de données. Ensuite, la commande "`php bin/console doctrine:migrations:migrate`" va appliquer le fichier de migration à notre base de données et mapper notre entité à celle qui sera dans la base de données.

Squelette de Symfony 4

Dans la figure ci-après, on peut voir ce à quoi ressemble notre projet Symfony et où se situent les principaux dossiers et fichiers dont nous allons expliquer le rôle principal :

- `src/Controller/` : Ce dossier contient tous les contrôleurs de notre application, c'est eux qui sont sollicités lorsqu'un client agit sur l'interface graphique. En fonction des "routes", c'est un certain contrôleur puis une certaine fonction de ce contrôleur qui est appelée afin de répondre à la demande
- `src/Entity/` : Ce dossier contient toutes les entités que l'on a créées dont leurs noms, leurs ids, leurs types, leur accesseurs, les fonctions associées, les relations entre entités
- `src/Form/` : Ce dossier contient les formulaires personnalisés, associés aux entités créées.
- `templates/` : Ce dossier contient toutes les vues de notre application ainsi que les "bases".

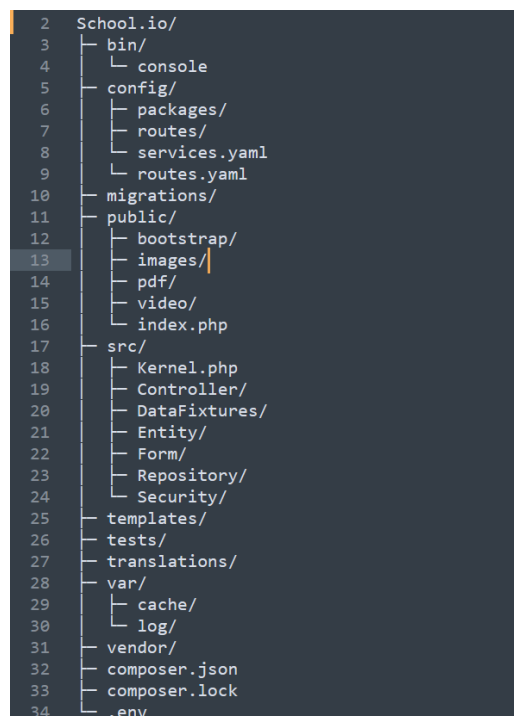


FIGURE 2.1 – Hiérarchie de notre application web appelée School.io

2.2.3 Bootstrap

Bootstrap est un framework CSS avec du contenu CSS et HTML qui permet de créer des sites web et des applications web de manière rapide. Il a été créé à l'origine par Twitter dans le but d'encourager la cohérence entre les outils internes, mais a ensuite été publié en tant que projet open source pour le public en 2011. Au fil du temps, des fonctionnalités telles que la réactivité (responsiveness en anglais) ont été ajoutées.

Comment utiliser Bootstrap ?

Pour intégrer Bootstrap dans un projet, rendez-vous d'abord sur getbootstrap.com. De là, vous pouvez soit télécharger du code compilé prêt à l'emploi pour Bootstrap et le déposer directement dans votre projet, soit utiliser un CDN via jsDelivr pour obtenir une version en cache des CSS et JS compilés de Bootstrap livré à votre projet. Son utilisation permet d'accélérer le rechargement des pages mais toutefois, en cas de problème de connexion Internet, le CDN ne fonctionnera plus. Il est donc préférable de conserver le code dans le projet.

Bootstrap vs CSS

Bien que Bootstrap soit un framework incroyablement utile, il existe un débat légitime sur le moment où il faut l'utiliser par rapport à l'utilisation de simples HTML/CSS. Dans notre cas particulier, l'utilisation et la familiarisation avec Bootstrap était une obligation, mais la question reste posée dans un contexte plus large.

Lors de la réalisation d'un projet qui contiendra de nombreuses fonctionnalités front-end uniques, il est probablement préférable d'éviter d'utiliser Bootstrap, car vous devrez travailler contre lui plutôt que d'utiliser ses fonctionnalités intégrées le plus souvent.

D'un autre côté, si le projet est davantage axé sur le back-end et que l'expérience de l'utilisateur n'est qu'une réflexion après coup, Bootstrap est un excellent outil pour rendre un site Web opérationnel rapidement. En outre, si le calendrier est une contrainte (par exemple, en raison des exigences et des délais du client), Bootstrap est également une bonne option pour travailler. Ainsi, même si Bootstrap n'était pas une exigence qui nous était imposée, cela aurait été une bonne décision compte tenu du temps limité dont nous disposions ce semestre.

Bases et Fonctionnalités de Bootstrap

Bootstrap offre une grande variété de classes pour faciliter le développement front-end. La classe de conteneur (container en anglais) est l'un des éléments de base qui contient et aligne le contenu. Les conteneurs sont nécessaires lorsque vous utilisez le système de grille par défaut (expliqué plus loin dans cette section) et sont de trois types.

Un container par défaut :

```
<div class="container">...</div>
```

est un conteneur réactif à largeur fixe. Cela signifie que lorsque la fenêtre s'agrandit, la largeur maximale du conteneur change également aux points de rupture spécifiés.

Les conteneurs peuvent également être spécifiés avec une taille, comme suit :

```
<div class="container-md">...</div>
```

Cela maintiendra le conteneur à 100% de largeur jusqu'à ce que le point de rupture moyen soit atteint.

Les conteneurs fluides

```
<div class="container-fluid">...</div>
```

s'étendent sur toute la largeur de la fenêtre, qu'elle soit petite ou grande.

Le système de grille de Bootstrap utilise un ensemble de conteneurs, de lignes et de colonnes pour mettre en page et aligner le contenu. La grille est réactive et peut prendre en charge jusqu'à 12 colonnes. À titre d'exemple, voici comment créer deux colonnes de largeur égale :

```
<div class="container">
  <div class="row">
    <div class="col-sm">
      Première colonne
    </div>
    <div class="col-sm">
      Deuxième colonne
    </div>
  </div>
</div>
```

Notez que les lignes fonctionnent ici comme des enveloppes pour les colonnes, ce qui signifie que sur chaque ligne vous pouvez choisir d'avoir un nombre différent de colonnes. Cela se fait facilement, comme suit :

```
<div class="container">
  <div class="row">
    <div class="col">
      1 of 1
    </div>
  </div>
  <div class="row">
    <div class="col">
      1 of 2
    </div>
    <div class="col">
      2 of 2
    </div>
  </div>
</div>
```

Les tailles des colonnes individuelles peuvent également être spécifiées à l'aide des classes .col. Comme la grille peut prendre en charge jusqu'à 12 colonnes, cette classe :

```
<div class="col-6">
```

représente une colonne qui prend la moitié de l'espace disponible, alors que ceci :

```
<div class="row">
  <div class="col-sm-8">col-sm-8</div>
  <div class="col-sm-4">col-sm-4</div>
</div>
```

représente deux colonnes qui occupent tout l'espace disponible sur la ligne, mais la première colonne est deux fois plus large que la seconde.

3 Analyse et Conception

3.1 Les maquettes

Après s'être intéressé à la logique de notre application, nous nous sommes intéressé à la partie front donc à l'interface graphique. Avant de commencer à coder, nous avons réalisé des maquettes à l'aide de l'outil en ligne Figma et en suivant les règles du material design. Nous avons modélisé les maquettes pour les pages des enseignants ainsi que la page d'accueil de l'application web.

Le material design dont nous avons suivi les règles d'usage, est un ensemble de règles de design (imposées par Google) telles que les jeux d'ombres qui sur-élèvent les formes sur la page afin de leur donner de l'importance, les couleurs qui doivent être limitées en nombre et avoir une certaine intensité, les boutons qui doivent avoir une couleur qui tranche et permet aux utilisateurs de les remarquer plus facilement, et bien d'autres...

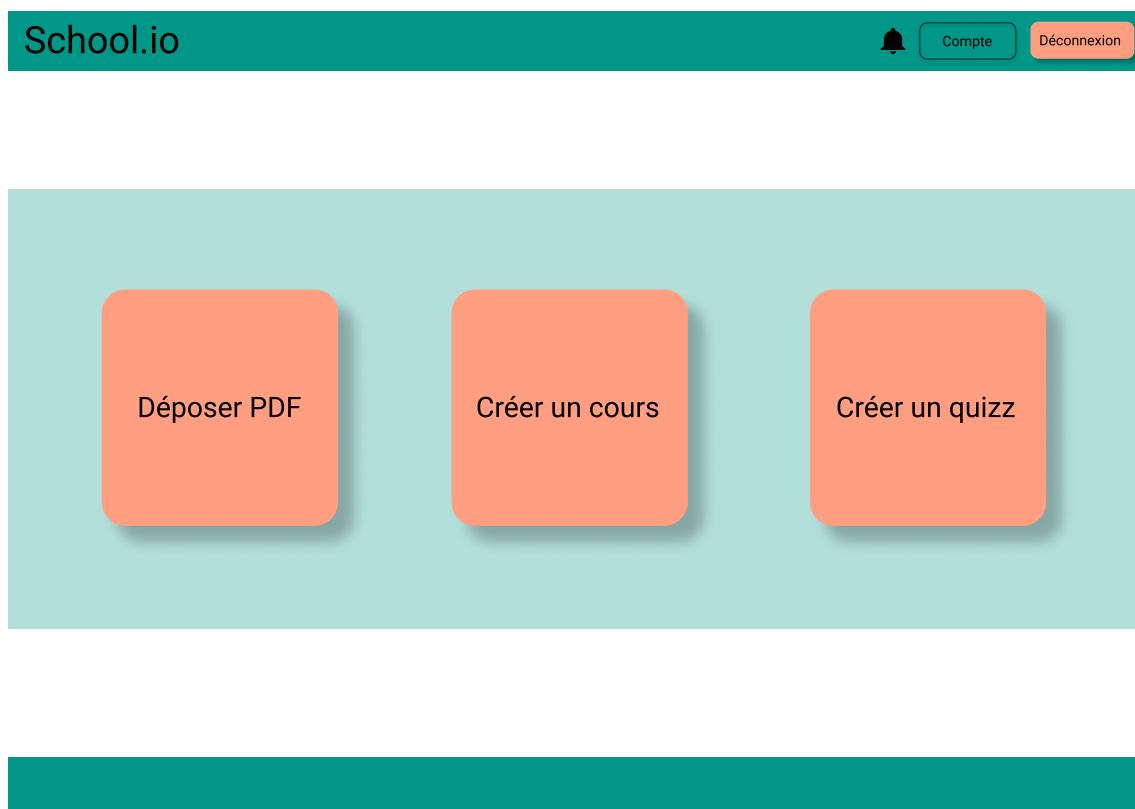


FIGURE 3.1 – Page d'accueil du compte enseignant

Remarque Dans cette partie nous allons expliquer les diagrammes que nous avons réalisés mais vous les trouverez dans les figures des pages suivantes dans les figures 3.2, 3.3 et 3.4.

3.2 Diagramme de Use-case

Un diagramme de Use-case est un diagramme qui fournit une vue de plus haut niveau du système. Ils donnent une représentation graphique des différents utilisateurs et de leurs interactions possibles avec le système et entre eux.

Dans notre diagramme particulier (voir figure 3.2), nos principaux utilisateurs sont les enseignants, les élèves, les parents et les administrateurs. Chaque ovale représente un cas d'utilisation (usecase), ou une action qui accomplit une sorte de tâche dans notre système. Par exemple, dans notre application scolaire, un enseignant peut ajouter, modifier, supprimer et consulter les notes de cours et les cours vidéos. Pour cela, il doit d'abord se connecter, ce qui est marqué par une authentification «include». Ceci est un exemple de relation include-relationship, où le cas d'utilisation de base nécessite l'exécution du cas d'utilisation inclus pour être complet.

3.3 Diagramme de séquences

Un diagramme de séquence décrit les interactions entre des objets disposés selon une séquence chronologique. Les lignes parallèles du diagramme représentent des objets ou des processus, tandis que les flèches horizontales représentent l'ordre dans lequel les interactions se sont produites.

Notre diagramme (voir figure 3.3) illustre une fonctionnalité où un parent peut inscrire un enfant à un compte, puis consulter l'ensemble de ses cours. Pour ce faire, le parent doit d'abord accéder à son propre compte. Si la connexion est réussie, une option permettant d'inscrire un enfant s'affiche. Dans le cas contraire, le parent sera renvoyé à la page de connexion. Une fois l'inscription réussie, le parent pourra accéder au compte de son enfant à tout moment à partir de son propre compte parental.

3.4 Diagramme de classes

Sur ce diagramme sont représentées les classes et les relations qui ont été transformées en entités dans notre base de données. Nous avons essayé d'optimiser au maximum les relations tout en conservant le plus de logique possible pour les utilisateurs, du côté interface graphique.

Les entités principales de notre application (enseignants, étudiants, parents et administrateurs) sont liées à une entité utilisateur depuis laquelle ils héritent des attributs email et mot de passe. Si on s'intéresse à la relation entre utilisateur et enseignant on peut voir qu'un enseignant est un utilisateur et qu'un utilisateur peut être un enseignant. Les deux entités sont liées par une relation 1 à 1. Ensuite, nous avons créé les tables document, questionnaire, pdf, vidéo (que l'on va nommer contenu pour expliquer plus clairement dans ce paragraphe) qui permettent de stocker les contenus ajoutés par les enseignants. Chacun de ces contenus est lié à une matière et à un enseignant. Cependant, un enseignant peut être lié à plusieurs contenus c'est pour cela qu'on les a liés avec un relation de un à plusieurs c'est à dire 1 d'un côté et 0 à n de l'autre. C'est exactement la même chose pour matière, chaque matière peut être liée à plusieurs contenus. L'entité matière est très importante car c'est seulement au travers de cette table que l'on va pouvoir accéder au niveau et à la filière d'un certain contenu de notre application. Cette table est également lié aux enseignants et aux étudiants car lors de leur inscription sur le site (ou a posteriori pour les enseignants) ils doivent choisir des matières afin de pouvoir soit ajouter du contenu (pour les enseignants) soit le consulter (pour les étudiants).

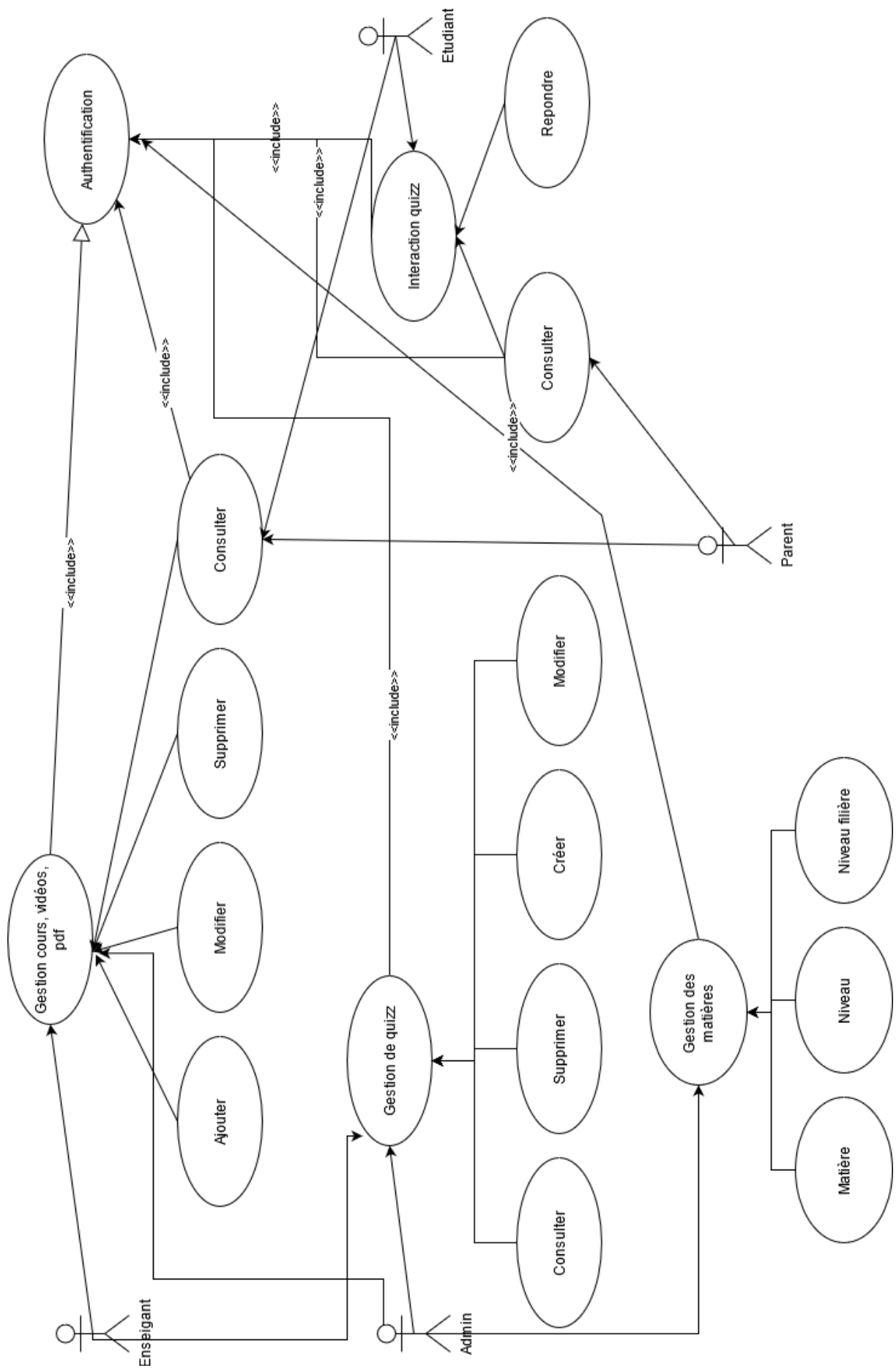


FIGURE 3.2 – Diagramme de usecase

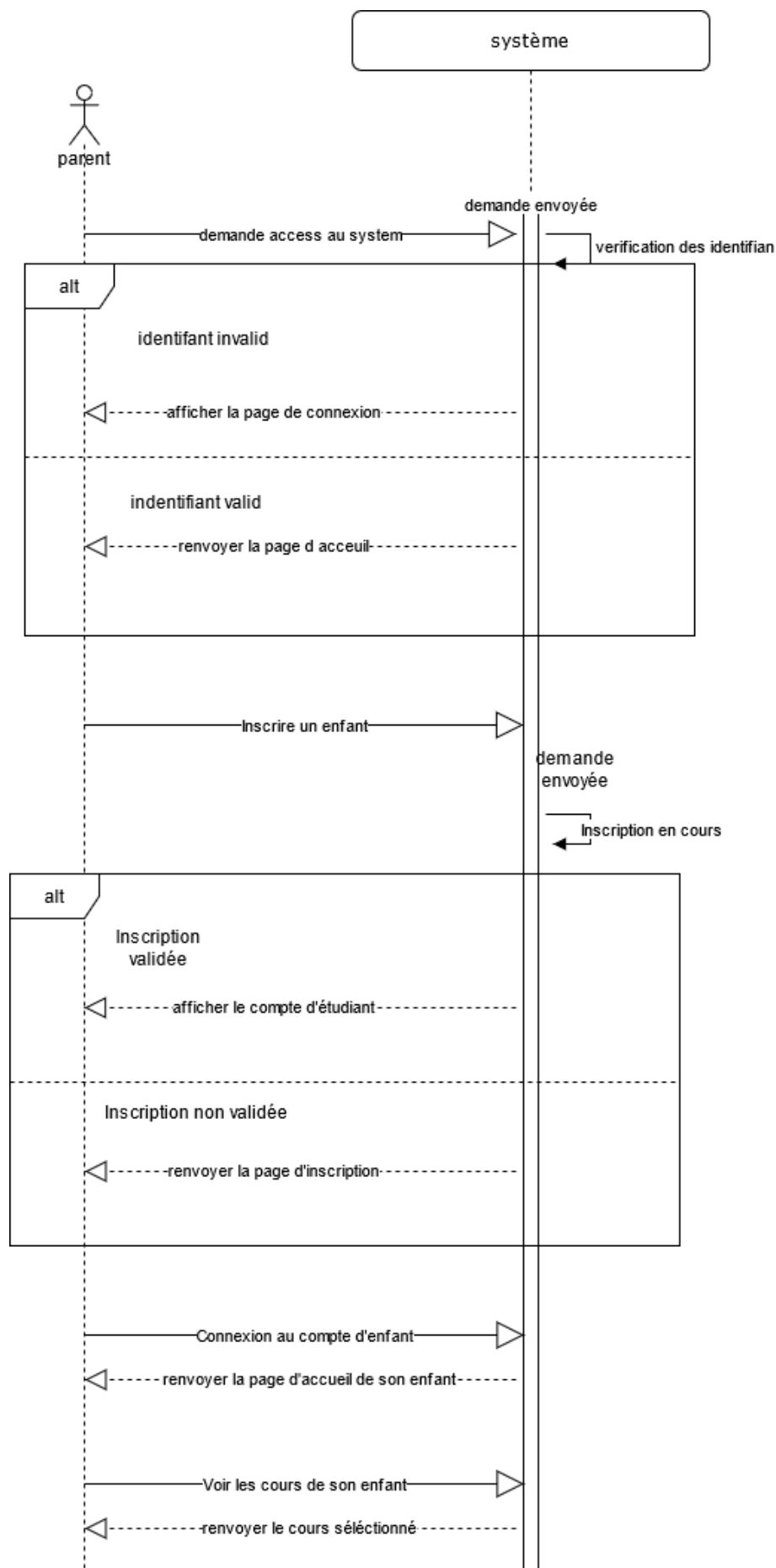
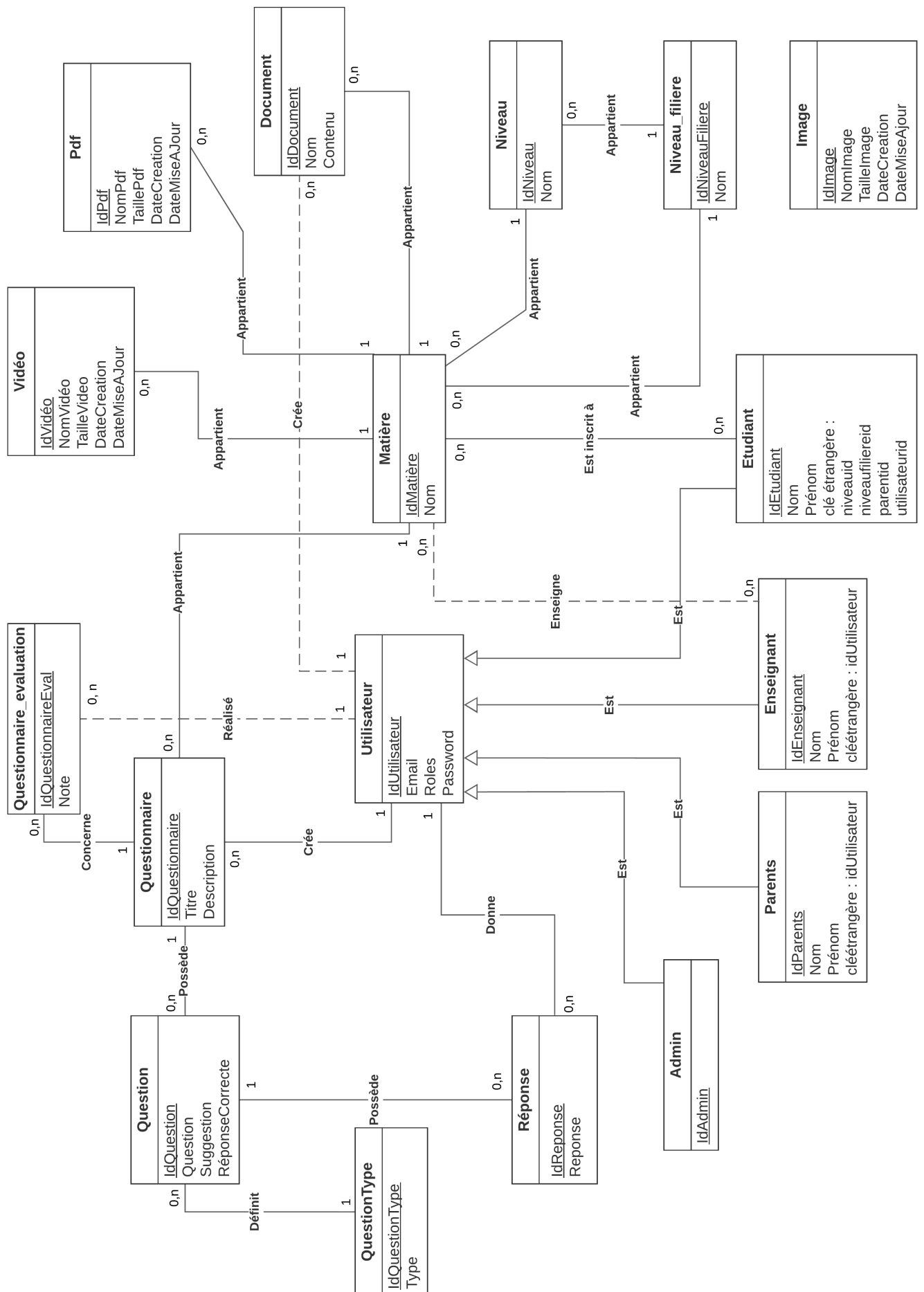


FIGURE 3.3 – Diagramme de séquence



15
FIGURE 3.4 – Diagramme de classe final de notre application

4 Développement logiciel

4.1 Fonctionnalités de l'interface graphique

Dans cette partie nous allons détailler et expliquer comment ont été réalisés les principales fonctionnalités de notre application web sans pour autant en faire une liste exhaustive. Les deux parties les plus développées sont enseignant et étudiant comme nous allons le voir. Nous avons également travaillé sur la partie parents et administrateur mais les fonctionnalités étant plus simples avons choisi de ne pas les développer ici.

4.1.1 La page d'accueil

Notre page d'accueil contient deux fonctionnalités : la possibilité de se connecter et de s'inscrire. Si une personne souhaite s'inscrire, il lui suffit de cliquer sur le bouton "inscription" et de là il lui est proposé de choisir son profil : c'est à dire de choisir de s'inscrire en tant qu'étudiant, enseignant ou parent. Une fois le profil choisi, le bouton sur lequel elle a cliqué appelle la route concernée donc soit `/etudiant/new`, `/enseignant/new` ou `/parents/new`. Chaque route est dans un contrôleur qui s'occupe d'exécuter la fonction associée qui va, selon le rôle, faire certaines actions. Pour tous, la fonction associée renvoie la vue associée à l'ajout d'un étudiant ou enseignant ou parent et intègre dans cette vue le formulaire associé c'est à dire celui d'ajout d'étudiant, d'enseignant ou de parent.

Inscription Voyons en détail la fonction `new` du contrôleur `Etudiant` qui est appelée dès qu'un étudiant veut s'inscrire. Lorsque c'est le cas, la fonction associée `"new"` va être lancée et va s'occuper de créer un nouvel étudiant grâce à la ligne `$etudiant = new Etudiant();` et l'entité `Étudiant`. Ensuite, la fonction va s'occuper de créer le formulaire associé et adapté, elle va vérifier qu'il est bien soumis, elle va créer un utilisateur avec les champs qui lui sont propres en plus de ceux de l'étudiant (mot de passe et email) et elle va lui associer le rôle d'étudiant ! Ensuite, ici nous avons une condition qui permet, lorsque c'est un parent qui demande l'inscription d'un étudiant, de récupérer l'utilisateur connecté donc le parent, et son id au champ de l'étudiant. Si c'est un étudiant qui fait la demande, le champ parent est mis à `"NULL"`. Ensuite, la fonction s'occupe de persister tout d'abord l'utilisateur puis l'étudiant et prenant le soin de lui rajouter son numéro d'utilisateur qui vient d'être créé. Une fois tout cela terminé, la fonction rend la page `"login"` si la personne est un étudiant, et renvoie sur le compte parent si la personne qui fait la demande d'inscription est un parent. Enfin, cette fonction appelle tout d'abord la vue `"etudiant/new.html.twig"` qui va faire afficher le formulaire c'est à dire `"etudiant/form.html.twig"` grâce à une création de vue sur celui-ci ! Vous trouverez ci-dessous le code correspondant à ce qui vient d'être expliqué.

```

/**
 * @Route("etudiant/new", name="etudiant_new", methods={"GET","POST"})
 */
public function new(Request $request, UserPasswordEncoderInterface $encoder): Response
{
    $etudiant = new Etudiant();
    $form = $this->createForm( type: EtudiantType::class, $etudiant);
    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid()) {
        $utilisateur = new Utilisateur();
        $utilisateur->setEmail($form->get('email')->getData());
        $utilisateur->setRoles(["ROLE_ETUDIANT"]);
        $utilisateur->setPassword($encoder->encodePassword($utilisateur, $form->get('password')->getData()));

        /** @var Utilisateur $parent */
        $parent = $this->getUser();
        if ($parent === NULL) {
            $etudiant->setParent( parent: NULL);
        } else {
            $etudiant->setParent($parent);
        }
        $entityManager = $this->getDoctrine()->getManager();
        $entityManager->persist($utilisateur);
        $entityManager->flush();

        $etudiant->setUtilisateur($utilisateur);
        $entityManager->persist($etudiant);
        $entityManager->flush();

        $parent = $this->getUser();
        if ($parent === NULL) {
            return $this->redirectToRoute( route: '/login');
        } else {
            $etudiant->setParent($parent);
        }
        return $this->redirectToRoute( route: 'CompteParent');
    }
    return $this->render( view: 'etudiant/new.html.twig', [
        'etudiant' => $etudiant,
        'form' => $form->createView(),
    ]);
}

```

FIGURE 4.1 – Code fonction new du contrôleur Etudiant.php

Choix des matières Lorsque les étudiant doivent choisir les matières auxquelles ils souhaitent s'inscrire, un script Javascript a été mis en place pour qu'une fois que l'étudiant a choisi son niveau, les filières associées à ce niveau remplissent l'input suivant et de la même façon, une fois qu'il a choisi sa filière, c'est l'input suivant des matières qui est rempli. Ce script se retrouve dans beaucoup de pages de notre application et il fonctionne grâce à deux fonctions :

- La première fonction vient du contrôleur de l'entité niveau et s'appelle "getFiliereByNiveau()". Elle permet - à l'aide de la fonction getNiveaufiliere() qui est définie grâce à la relation entre les entités Niveau et NiveauFiliere - de récupérer les filières pour un niveau donné, celui qui a été choisi dans le premier input. La fonction récupère les noms des niveaux à partir des id et renvoie le tout sous forme de tableau.
- La deuxième fonction vient du contrôleur de l'entité matière et s'appelle "getMatiereByNiveauAnd-Filiere()". Elle permet - à l'aide de la fonction findBy() du répertoire matière, d'un niveau et d'une filière - de récupérer les matières associées. Le résultat du "findBy(['niveau'=> idNiveau, 'niveauFiliere' => idFiliere])" va être stocké dans un tableau et pour chaque matière récupérée dans le tableau, on va aller chercher son id puis son nom pour ensuite pouvoir fournir ce résultat (sous format JSON) à l'input des matières. Voici le code de la deuxième fonction dont nous avons expliqué le fonctionnement :

```

* @Route("/getMatiereByNiveauAndFiliere/{idNiveau}/{idFiliere}", name="get_matiere_by_niveau_and_filiere_new", methods={"GET","POST"})
*/
public function getMatiereByNiveauAndFiliere(Request $request,$idNiveau,$idFiliere, MatiereRepository $matiereRepository): Response
{
    $matieres = $matiereRepository->findBy([
        'niveau'=> $idNiveau,
        'niveauFiliere' => $idFiliere
    ]);
    $matiereJson = [];
    if ($matieres) {
        foreach ($matieres as $matiere) {
            $matiereJson[$matiere->getId()] = $matiere->getNom();
        }
    }
    return $this->json($matiereJson);
}

```

FIGURE 4.2 – Fonction getMatiereByNiveauAndFiliere du contrôleur de matière

4.1.2 Le compte des étudiants

Leur page d'accueil

Une fois un étudiant connecté, le contenu de sa page d'accueil va varier en fonction des choix fait au moment de l'inscription : s'il a choisi de s'inscrire à une seule matière ou à plusieurs. Lorsqu'il se connecte et accède à son compte, si l'étudiant a choisi plusieurs matières c'est la route `@Route("/CompteEtudiant", name="CompteEtudiant")` qui est appelée et qui va l'amener sur la page `CompteEtudiant.html.twig` et qui va rentrer dans le "else" de la boucle twig que nous avons créée. Si l'étudiant n'est inscrit qu'à une seule matière, il est directement dirigé grâce à la route `@Route("/CompteEtudiant/idMatiere", name="CompteEtudiantMatiere")` qui va aussi l'amener sur la page `CompteEtudiant.html.twig` et qui va passer dans la condition "if" de notre boucle twig et va afficher les contenus disponibles pour une matière. Lorsqu'un étudiant clique sur le bouton d'une certaine matière, c'est la deuxième route prenant en paramètre l'id qui est appelée. Afin de pouvoir récupérer et tester dans la boucle twig le nombre de matières, il faut ajouter le paramètre `'matieres' => $userMatiere` à la fonction `render` qui s'occupe du rendu de la page.

Toutes nos pages ont la même base : un div de type "container-fluid" qui englobe tous les éléments de la page, puis un div de type "row" pour le menu de haut de page présent sur tout le site, et enfin un autre div de type "row" pour la bannière de bas de page.

Les gros boutons situés au centre de la page des étudiants ainsi que dans beaucoup d'autres pages de notre site (compte enseignants, parents, administrateurs) sont placés sur la page grâce à une div de type "row" qui englobe des div de type "col" qui représentent les colonnes de la grille bootstrap. Ils ont tous le même design et sont faits à partir d'un div de type "card" qui, concrètement, est une boîte rectangulaire qui peut être composée d'un en-tête, d'un corps et d'un pied que nous avons personnalisé afin qu'il corresponde à nos attentes dans le fichier "landing-page.css" qui est lié à chaque page directement dans le "body" de ces pages. Nous avons choisi ce type de card par praticité afin que les boutons puissent être déclenchés sur la totalité de leur surface.

```

<div class="col-md-4 col-xl-3">
    <a style="color:white; text-decoration: none;" href="{{ path('/etudiant/pdf/list/{idMatiere}', {'idMatiere': matieres|first.id}) }}">
        <div class="card bg-c-color order-card" style="opacity: 0.2;">
            <div class="card-block">
                <h4 style="text-align:center; margin-top:40%; ">PDF</h4>
            </div>
        </div>
    </a>
</div>

```

FIGURE 4.3 – Utilisation des card dans notre code

Les matières

Après avoir cliqué sur le bouton d'une de ses matières, ou s'il ne s'est inscrit qu'à une matière, l'étudiant arrive sur une page qui contient tout le contenu de la matière. Il y a alors quatre boutons (identiques aux

précédents) qui permettent d'accéder respectivement à des listes de pdfs, de vidéos, de cours ou de QCM disponibles pour une matière. Lorsqu'il clique sur un des boutons, il accède à un index spécial étudiant créé afin de lister les matières en fonction de celle sur laquelle l'étudiant est. Ces listes ont demandé la création de plusieurs autres fonctions PHP qui permettent de récupérer dans chaque répertoire de contenu, la liste des éléments disponibles en fonction de la matière grâce à un `findBy()` auquel on fournit l'id de matière.

4.1.3 Le compte des enseignants

Leur page d'accueil

Voici le visuel de la page d'accueil des enseignants construit de manière presque identique à celle des étudiants :

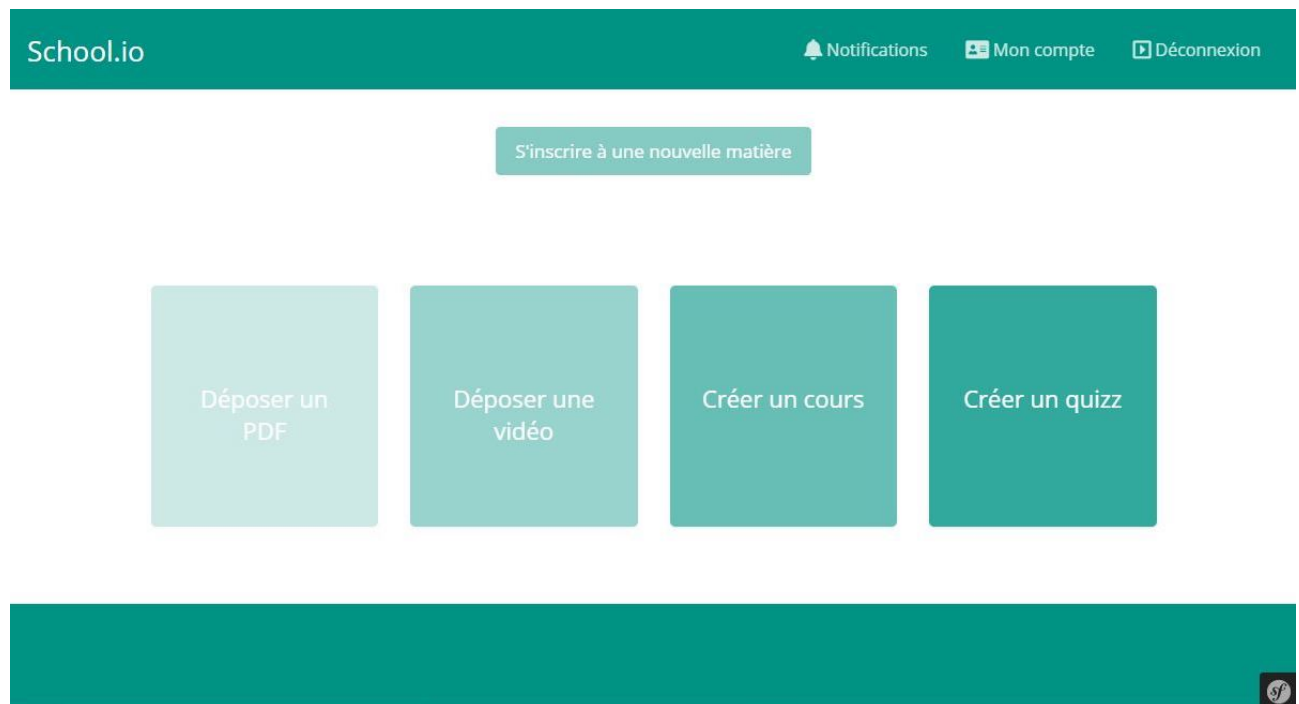


FIGURE 4.4 – Page d'accueil des enseignants

Déposer un pdf et une vidéo

Lorsque l'enseignant clique sur déposer un PDF ou une vidéo, il est redirigé vers la page "index.html.twig" associé à l'entité PDF ou vidéo, créé au moment de la création de l'entité grâce à la commande "make :crud". Depuis cet index on peut facilement accéder au contenu, le modifier ou en ajouter. Chaque bouton est lié à un lien menant vers une de ces pages (lecture, modification ou ajout).

Créer un cours

Sur cette page, il peut d'abord voir la liste des cours qu'il a déjà créés, les visualiser ou les modifier en cliquant sur le bouton correspondant au bout de la ligne qui contient le cours visé. Et il peut également en créer d'autres en cliquant sur le bouton "Créer un nouveau cours" situé en bas de la liste. Après avoir cliqué sur ce bouton il est redirigé sur une page qui contient un formulaire dynamique qui va l'aider à créer un cours. Il doit tout d'abord donner un nom à son cours et choisir la matière à laquelle il veut qu'il soit lié. Ensuite il peut commencer par ajouter des sections à son document grâce à un système de bouton "+". Après avoir créé au moins une section il peut alors commencer à ajouter du contenu comme des paragraphes, des définitions ou des photos toujours grâce au bouton "+" qui se placeront automatiquement aux endroits où il est possible d'ajouter du contenu. L'enseignant pourra également à tout moment déplacer les sections, les paragraphes, les définitions et les photos en le prenant avec sa souris et en le déposant à l'endroit voulu. Chaque élément du cours peut à tout moment être supprimé ou

modifié grâce à des boutons situés au niveau du contenu. Une fois son cours complété, il lui suffit d'appuyer sur le bouton "Envoyer" et son cours sera enregistré. Nous verrons dans une partie entièrement consacrée à cet éditeur, comment nous l'avons construit.

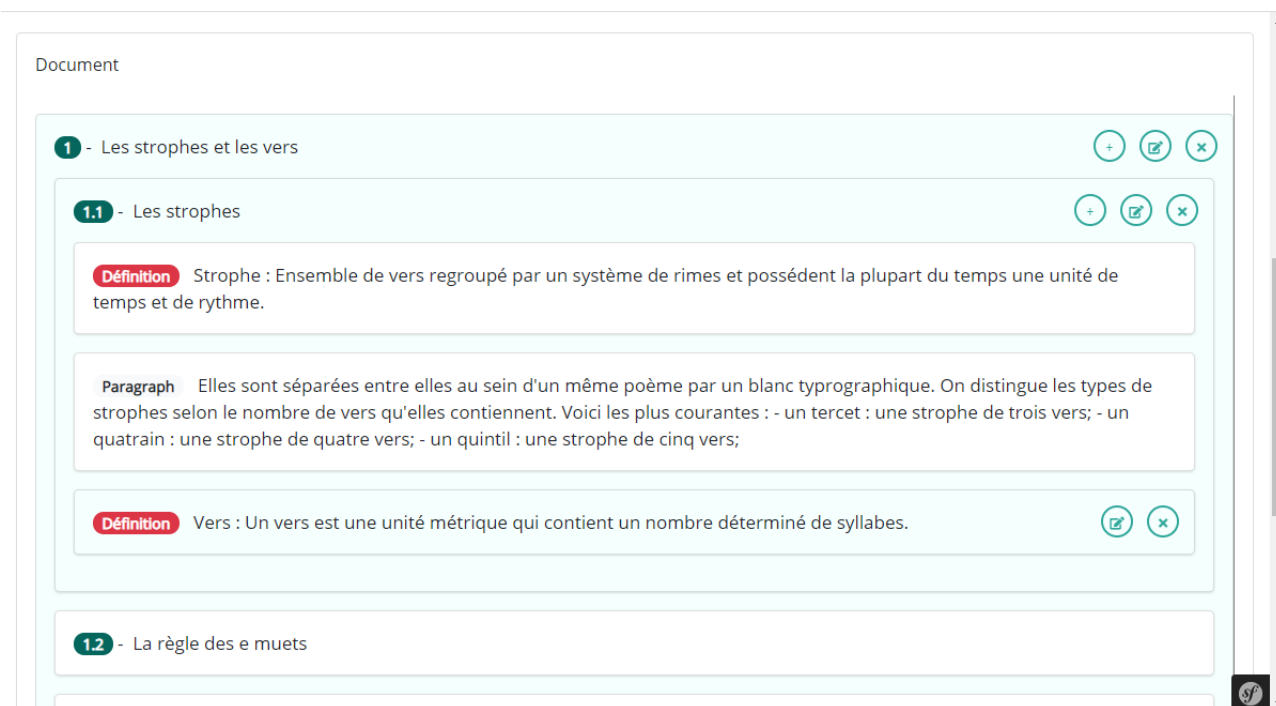


FIGURE 4.5 – Exemple de création d'un cours

Créer un quizz

De la même manière que les pages précédentes, cette page contient la liste des quizz déjà créés. Il peut alors, grâce à des boutons au bout de la ligne, soit le visualiser et ensuite décider de l'éditer ou de le supprimer, soit le modifier, soit noter les étudiants qui l'ont déjà réalisé. Pour cela il a un bouton qui l'amène sur une page avec la liste des étudiants qui l'ont déjà fait. Il peut également grâce au bouton "Créer un nouveau questionnaire" accéder à une page qui va lui permettre de créer un quizz facilement. Sur cette page se trouve un formulaire qui l'aide à créer son quizz. Il doit tout d'abord rentrer le titre, une description ainsi que la matière à laquelle le quizz sera relié (pour cela il rentre le niveau, puis la filière et ensuite sera affiché les matières correspondantes). Pour ajouter des questions, il lui suffit de cliquer sur "Ajouter une question".

Le script qui nous permet de rajouter plusieurs questions en affichant plusieurs formulaires est le suivant :

```
$('#body').on('click', '.add_item_link', function (e) {
    let $collectionHolderClass = $(e.currentTarget).data('collectionHolderClass');
    // add a new tag form (see next code block)
    addFormToCollection($collectionHolderClass);
    updateCorrectAnswerLabels();
    ListenToLablClick();
})
```

FIGURE 4.6 – Exemple de création d'un cours

Le bouton "ajouter une question" est de type "add item link" et il est lié avec le data-prototype qui contient le code pour le formulaire fils. Quand on clique sur ce bouton, le contenu de ce formulaire sera stocké dans la variable `collectionHolderClass`. Ensuite, on s'est servi du script `addFormToCollection()` disponible sur la documentation de Symfony pour exécuter sa fonctionnalité. On a remplacé les balises `` par des `div` de type `segment` pour pouvoir l'adapter à notre affichage.

De là, lui est proposé un cadre où il peut rentrer la question, le type de réponses attendu (choix unique, choix multiple ou texte) ainsi que les réponses possibles. Pour que les enseignants puissent mettre différentes possibilités de réponses nous avons utilisé le script `tagsinput` aussi disponible sur la documentation bootstrap. Enfin, pour limiter le nombre de bonnes réponses sélectionnable en fonction du type de question choisi, nous avons écrit un script javascript récupérant l'id du type de question et adaptant ainsi le nombre de champs que l'on peut sélectionner.

Afin de noter quelle est la (ou les) bonne(s) réponse(s), il lui suffit de cliquer dessus, et elle sera enregistré comme bonne réponse. Après avoir complété le quizz avec toutes les questions qu'il voulait, il peut le valider en cliquant sur "Enregistrer" et il sera directement disponible pour les étudiants.

Inscription à des matières

Lorsque l'enseignant veut s'inscrire à d'autres matières, il clique depuis son menu sur le bouton "S'inscrire à une nouvelle matière". Il arrive sur une page qui lui propose de choisir d'abord un niveau, puis une filière comme partout grâce au script JavaScript. Puis, une liste des matières est formée et une autre liste est formée par celles où l'enseignant est déjà inscrit, un script Javascript en est également à l'origine, c'est une petite variante de celui pour les remplissage des input pour les choix de niveau, filière et matière que nous avons déjà vu. Un drag and drop est possible entre les deux listes grâce à la bibliothèque Sortable de JQuery dont nous avons ajouté et adapté la fonction et les propriétés.

School.io

Notifications Mon compte Déconnexion

Choisissez les matières pour lesquelles vous souhaitez ajouter du contenu :

Niveau Seconde

Niveau filière Générale

Matières pour ce niveau et cette filière:

- Mathématiques
- Français
- SES

Matières auxquelles je suis inscrit(e):

- Mathématiques
- Français
- SES

Ajouter

FIGURE 4.7 – Page pour ajouter des matières

Nous allons enfin vous parler brièvement des différentes fonctionnalités des comptes parents et administrateurs sans rentrer dans les détails techniques.

4.1.4 Le compte des parents

Leur page d'accueil

La page possède la liste des enfants du parent qui sont inscrit sur le site. A partir de cette liste il peut choisir d'aller voir le compte de son enfant grâce à un bouton situé sur la ligne de l'enfant dans la liste.

Inscrire un enfant

Cette page est identique à celle où les étudiants peuvent s'inscrire par eux mêmes. C'est en fait le même formulaire mais qui va recevoir une valeur pour le champ "parents", alors qu'il est mis à NULL sur

l'autre page, nous verrons ça plus précisément dans une partie consacrée à un exemple d'utilisation des contrôleurs, formulaires et entités avec php.

Voir le compte de son enfant

Une fois sur le compte de son enfant, il aura accès à toutes les ressources auxquelles son enfant à accès, la seule différence est qu'il ne pourra pas faire les quizz.

4.1.5 Le compte des administrateurs

Leur page d'accueil

Elle contient le menu habituel pour retourner sur sa page d'accueil grâce à "Mon compte" et pour se déconnecter avec "Déconnexion". La page d'accueil contient en son centre 8 boutons : "Gestion niveaux", "Gestion filières", "Gestion matières", "Gestion type de questions", "Gestion PDFs", "Gestion vidéos", "Gestion cours" et "Gestion quizz".

Gestion niveaux, filières, matières et type de questions

Lorsqu'un administrateur clique sur un de ces boutons, il est amené à la liste des niveaux (ou filières ou...) qui sont dans la base de données et peut en rajouter ou les modifier afin de faire évoluer le site au besoin.

Gestion PDFs, vidéos, cours et quizz

Grâce à ces boutons, il peut avoir accès à la liste de tout le contenu du site et est capable de le modifier ou de le supprimer.

Nous allons maintenant développer plus spécifiquement la partie de l'éditeur de cours afin d'expliquer en détail comment elle a été faite, et nous parlerons en suite des utilisations spécifiques à notre projet de Symfony, Bootstrap, la bases de donnée ou encore Twig.

4.2 L'éditeur de cours

Afin que les enseignants puissent proposer des cours homogènes, c'est à dire les afficher toujours d'une certaine manière sur notre application web, nous avons créé un éditeur dynamique de cours. Cet éditeur permet à l'enseignant de créer des cours sans se soucier de l'affichage puisqu'il lui suffit de créer chaque partie là où il le souhaite, les nommer, les remplir, puis l'éditeur et l'application se chargeront de le mettre en page pour lui. Pour réaliser ceci nous avons principalement travaillé en JavaScript mais aussi avec la base de données qui nous permet de stocker un objet JSON contenant la totalité du formulaire soumis par l'enseignant dans le but d'être affiché par la suite sur notre application. Nous allons détailler les parties les plus importantes de la construction de cet éditeur.

4.2.1 Mustache

Mustache est un moteur de template qui permet d'écrire des templates à l'aide de balises HTML qui sont ensuite contrôlées par du JavaScript (cachés, affichés, concaténés aux autres éléments). Nous avons utilisé Mustache pour écrire trois scripts pour trois templates différents entre des balises *"verbatim"* qui permettent d'empêcher à Twig d'interpréter le code Mustache écrit car celui-ci lui ressemble fortement. Nous avons un script qui affiche des modals nommés "modal", c'est la fenêtre pop-up qui apparaît lorsqu'on clique sur n'importe quel bouton ajout de notre éditeur.

Nous avons écrit un autre script pour des modals d'édition de section nommés "edit-modal", c'est la fenêtre qui apparaît lorsqu'on clique sur un bouton de modification.

Enfin, nous avons écrit un troisième script Mustache nommé "section" qui fait le rendu des éléments que l'enseignant a choisi et rempli pour agrémenter son cours.

C'est grâce à la fonction *Mustache.render()* JavaScript que les trois scripts peuvent être remplis et utilisés.

4.2.2 Les modals

Mustache permet de faire le rendu de modals. Ce sont des fenêtres pop-up qui apparaissent lorsqu'on clique pour ajouter ou modifier un élément. Les deux scripts modals contenant le code html permettent d'afficher en fonction de l'emplacement dans l'éditeur (racine ou non), les propositions d'ajouts d'éléments possibles pour le premier et d'afficher le contenu que l'on peut modifier pour le deuxième. Pour le modal d'ajout, en fonction du type de contenu choisi, le modal fait apparaître le champ, si c'est une définition il fait apparaître un input texte, si c'est une image il fait apparaître un bouton pour permettre d'insérer une image par exemple.

4.2.3 Le corps de l'éditeur de cours

Initialisation

La création d'un cours commence dès lors que l'enseignant appuie sur le bouton "+". Ce bouton est lié à un "data-target" qui va appeler un modal spécial (modal-0) pour le début de document ayant comme data-parent la racine "root". Mustache va faire le rendu de ce modal qui va proposer le seul type d'élément autorisé au niveau de la racine de notre document : les sections. Quand l'enseignant va vouloir l'ajouter, la nouvelle section va être concaténée au document, va recevoir un nom "child" et va avoir deux modals associés cachés, un pour la modification et un pour l'ajout (vu que c'est une section).

Évènements attendus

Notre éditeur contient une fonction d'écoute "addEventListener" qui permet de rendre le document dynamique en fonction des demandes de l'enseignant. Par exemple il y a un évènement qui écoute les clics sur le bouton d'ajout principal de l'éditeur et tous les boutons d'ajout des modals que l'on ne voit pas au début. Il y a aussi deux autres évènements qui vont s'enclencher dès que l'enseignant va passer sa souris au dessus d'un élément ou la retirer, ces évènements permettent de colorer la partie survolée ainsi que tous ses parents (nous verrons plus tard ce que sont les parents). Ensuite, il y a un évènement qui permet de supprimer un élément, cet évènement écoute les boutons de suppression qui sont ajoutés au moment de la création d'un élément comme nous allons le voir

Ajout d'un élément dans une section Lorsque l'enseignant a déjà créé une section au niveau de la racine et décide d'ajouter un élément à l'intérieur de celle-ci, il clique sur le bouton d'ajout depuis la section qui fait apparaître le modal associé créé. Lorsqu'il clique sur "Ajouter" après avoir rempli les champs, c'est la fonction "addEventListener" qui est déclenchée grâce à l'évènement nommé "addElement" et va appeler une fonction "addContent". C'est cette dernière fonction qui va permettre de récupérer ce qui a été saisi dans l'input ainsi que le data-parent de l'élément sur lequel on vient de cliquer (grâce au modal qui le contient), et de faire appel à la fonction "renderContent" qui est responsable de créer l'élément à l'aide du template "section" de Mustache, de le remplir avec les données saisies (par enseignant et data-parent) et de créer les modals associés à l'aide des deux templates modals. Ensuite c'est la fonction "addElement" qui s'occupe de concaténer le nouvel élément retourné, à son parent dans le document grâce à la ligne *parentElem.append(contentRender['sectionRendered'])* ; et de concaténer les deux modals cachés créés. Ce nouvel élément créé possède des boutons d'ajout, de modification et de suppression en fonction de son type, par exemple si c'est une section les trois boutons sont disponibles, par contre si c'est une image, seulement les deux derniers sont disponibles.

Modification et suppression d'un élément Lorsque que l'enseignant clique sur le bouton modifier, le modal de modification est appelé. Lorsqu'il clique sur le bouton de suppression, c'est l'évènement "removeContent" qui est appelé

Autres fonctions très importantes Nous avons écrit d'autres fonctions dont la fonction "updateSectionNumber" qui permet d'actualiser les numéros des sections en fonction de leur emplacement. Elle est activée et fait une boucle récursive sur tout le document afin de s'assurer que tout est dans l'ordre. Ensuite nous avons activé le plugin sortable de JQuery qui permet de sélectionner les éléments, les faire glisser pour les changer de place.

4.2.4 Récupération du contenu JSON et affichage lors d'une modification

La fonction getJsonDocument Cette fonction permet de faire une boucle sur toutes les sections puis

pour chaque section, une boucle sur chaque enfant et ainsi de suite afin de récupérer le texte JSON qui est ensuite indexé dans un objet JSON avec les trois champs "content", "contentType" et "children". Ensuite, la fonction "renderJson" va s'occuper de reconstruire tout le document, lorsqu'un enseignant va demander à le modifier. Chaque section est recrée à partir du JSON, chaque modal d'ajout et chaque modal de modification est recréé afin de permettre à l'enseignant de modifier tout ce qu'il souhaite et par la suite faire l'update vers la base de données en envoyant le nouvel objet JSON créé.

4.3 Bootstrap

Notre application web fonctionne comme préconisé par Symfony, à l'aide de "bases" qui sont des fichiers qui permettent de factoriser le code. Nous avons huit bases dont environ deux pour chaque type d'utilisateur. Dans ces bases, nous avons choisi de mettre la balise <head>, la liaison vers le page contenant notre CSS ainsi que le menu et le footer. Chaque base définit une nav-bar spéciale, relative au rôle de l'utilisateur connecté. Nous avons construit cette nav-bar dans les bases pour qu'à l'aide de twig, elle soit automatique inclus et que l'on puisse directement écrire le corps des pages. Cette factorisation de code a un autre avantage, lorsque l'on veut modifier quelque chose sur notre nav-bar, plus besoin de se rendre sur chaque page de notre site ce qui serait vraiment trop long. Nous avons aussi défini un footer simple qui pourrait être développé dans une suite de projet, avec des liens, peut-être un accès plus rapide à des parties du site.. Nous l'avons seulement inclus dans notre base comme nous l'avons fait avec la nav-bar. Par exemple, voici le code de notre nav-bar adapté au menu du site lorsque l'utilisateur n'est pas encore connecté :

```
<body class="landing-page landing-page1" style="...">

<div class="container-fluid">
  <div class="row">

    <nav class="navbar navbar-transparent navbar-top navbar-dark" style="background-color: #009283" role="navigation">
      <div class="container-fluid">
        <!-- Brand and toggle get grouped for better mobile display -->
        <div class="navbar-header">
          <a href="#">
            <div class="logo-container">
              <div class="brand">
                School.io
              </div>
            </div>
          </a>
        </div>
        <!-- Collect the nav links, forms, and other content for toggling -->
        <div class="float-right" id="example">
          <ul class="nav navbar-nav navbar-right" style="...">
            <li>
              <a href="/login">
                <i class="fas fa-sign-in-alt"></i>
                Connexion
              </a>
            </li>
            <li>
              <a href="{{ path('/Inscription') }}">
                <i class="fas fa-user-plus"></i>
                Inscription
              </a>
            </li>
          </ul>
        </div>
        <!-- /.navbar-collapse -->
      </div>
    </nav>
```

FIGURE 4.8 – Le navbar de Base Accueil

4.4 Intégration à l'aide de Twig

Les bases La plupart de nos vues sont construites grâce à Twig qui inclut au début du document, la base dont nous avons besoin pour afficher le bon menu et le bon footer. Il réalise cela grâce à : `{% extends 'BaseEnseignant.html.twig' %}` par exemple pour les pages qui concernent les enseignants. Ensuite, il permet de définir le corps des pages entre les balises : `% block body %` et `% endblock body %`. Enfin, nous l'utilisons dans l'affichage de nos templates de cours et quizz pour pouvoir afficher correctement du côté étudiant ou du côté enseignant pour la vérification.

Template éditeur de cours Afin d'afficher correctement et de manière uniforme tous les cours de notre application web, nous avons eu recours à twig qui s'occupe d'afficher selon l'html et le css que l'on a choisi, les données que l'enseignant a fournis dans son formulaire de création de cours. Twig va parcourir à l'aide d'une boucle la totalité de l'objet JSON récupéré et va afficher section par section et sous section par sous section, tous les éléments.

Voici un exemple détaillé :

```
{% for content in contents %}
    {% if content["contentType"] == "section" %}
        <li><h1 style="color:#009283; opacity:0.75"> {{ rootIndex }}. {{ content["content"] }} </h1></li>
        {% set rootIndex = rootIndex+1 %}

        {% if content['children'] is not empty %}
            {{ _self.renderChildren(content,loop.index) }}
        {% endif %}

    {% elseif content["contentType"] == "paragraph" %}
        <p>{{ content["content"] }}</p>
    {% elseif content["contentType"] == "title" %}
        <h3>{{ content["content"] }} </h3>
    {% elseif content["contentType"] == "image" %}
        
    {% endif %}
{% endfor %}
```

FIGURE 4.9 – Twig

En parcourant les contenus du document, on vérifie le type de chaque contenu. Lorsqu'on tombe sur un type «section», il s'agit d'un titre d'une nouvelle section. Twig donc va l'afficher avec le style précisé – un titre h1, la couleur correspondant à 009283, et semi-opaque.

Le texte attendu après le titre d'un section est du type «paragraph », et Twig affichera le contenu tout simplement entre des balises `<p>...</p>`

Twig s'occupe aussi d'afficher des « définitions ». Dans les documents, les définitions sont mises en `<h3>` pour les mettre en évidence. Ici il y a une petite erreur sémantique, car le type n'est pas marqué « définition » mais plutôt «title». En revanche, tout marche comme il le faut.

Finalement, pour que les contenus de type « image » soient bien affichées, il suffit tout simplement de les mettre entre les balises ``, avec des spécifications de tailles qui nous convient.

Voilà maintenant un document affiché selon un format précisé grâce à Twig.

4.5 Utilisation de la base de données avec les Forms

Symfony inclut une fonction de formulaire puissante qui fournit tout un tas de fonctionnalités qui permet de traiter les cas les plus complexes. Dans la partie enseignant par exemple, on a utilisé la commande `make :crud` (dans le terminal) qui a permis de générer le formulaire associé à la création d'un enseignant. On a ensuite ajouté les champs "nom" "prenom" "email" "password" grâce à la fonction "add". Les champs enseignant et password sont récupérés depuis l'entité utilisateur. De plus pour "email"

et "password" nous avons ajouté le paramètre "EmailType : :class" et "PasswordType : :class" qui est un champ de texte rendu grâce à HTML5 qui permet de forcer l'utilisateur à rentrer un email et un mot de passe valide dans ces champs. Pour les rendre fonctionnel il a été nécessaire d'ajouter les Namespace pour chaque champ spécifique. A titre d'exemple le paramètre "EmailType : :class" comporte un message invalide par défaut si l'utilisateur rentre une mauvaise adresse mail mais aussi un format obligatoire qui est le format d'adresse mail classique (texte@texte.texte)

Ensuite, nous avons par exemple utilisé l'option "mapped => false" pour que tous les utilisateurs, au moment où c'est demandé, puisse choisir d'abord leur niveau, puis leur filière puis leur(s) matière(s). Un script JavaScript permet de les afficher en fonction de ce qui est choisi dans l'input précédent. Pour ce qui est de la base des données, le formulaire d'inscription aux matières pour chaque utilisateur récupère forcément les entités niveau et niveau filière et matière mais ne va récupérer pour stocker dans sa table, que le champ matière car c'est grâce à lui qu'on peut remonter aux deux autres champs comme nous l'avons vu dans le diagramme de classe. Afin de réaliser ceci il a été nécessaire d'ajouter le morceau de code ci dessous. On peut voir que le champ matière est ajouté aux champs de l'étudiant mais par contre niveau et niveauFiliere ne seront pas ajoutés.

```

    })
    ->add( child: 'niveau', type: EntityType::class,[
        'placeholder'=> 'Choisir un niveau',
        'class' => Niveau::class,
        'mapped' => false,
        'required' => false,
    ])
    ->add( child: 'niveauFiliere', type: EntityType::class,[
        'class' => NiveauFiliere::class,
        'placeholder'=> 'Choisir un niveau',
        'mapped' => false,
        'required' => false,
    ])
    ->add( child: 'matiere');
}

```

FIGURE 4.10 – Morceau de code de l'entité Etudiant

4.6 Les sessions

Les utilisateurs Afin de mettre en place les sessions, nous avons du installer un bundle Symfony appelé "security-bundle". Nous avons ensuite créé une entité "Utilisateur" que nous avons ensuite lié avec les entités étudiants, parents, enseignants et administrateurs. Lorsqu'un utilisateur veut se connecter, le formulaire d'authentification va faire appelle à une fonction "onAuthenticationSuccess" qui va les rediriger vers la bonne page en fonction de leur rôle.

Les rôles Afin de pouvoir interdire l'accès aux utilisateurs à des pages sur lesquelles ils ne sont pas sensé pouvoir aller, nous avons donné des rôles à nos utilisateurs. Les étudiants ont pour rôle "ROLE_ETUDIANT", les enseignants "ROLE_ENSEIGNANT", les parents "ROLE_PARENT" et les administrateurs "ROLE_ADMIN". Nous avons ainsi pu limiter l'accès au site grâce à des routes auxquelles seul des utilisateurs ayant un rôle particulier peuvent aller. Pour cela, nous avons configuré la section "access_control" du fichier "security.yaml" de manière à ce que les utilisateurs ayant le rôle "ROLE_ENSEIGNANT" puisse seulement accéder au page du site ayant une route commençant par "/CompteEnseignant".

La déconnexion Afin de donner la possibilité à un utilisateur de se déconnecter, nous avons dû activer le paramètre de configuration "logout" dans le fichier "security.yaml". On a ensuite dû créer une route

```
role_hierarchy:
    ROLE_ADMIN: [ROLE_ENSEIGNANT]

# Easy way to control access for large sections of your site
# Note: Only the *first* access control that matches will be used
access_control:
    - { path: ^/CompteAdmin, roles: ROLE_ADMIN }
    - { path: ^/CompteEnseignant, roles: ROLE_ENSEIGNANT }
    - { path: ^/CompteEtudiant, roles: ROLE_ETUDIANT }
    - { path: ^/CompteParent, roles: ROLE_PARENT }
```

FIGURE 4.11 – Les rôles associés à des routes spécifiques

"app_logout" pour cette URL dans "securitycontroller" vers la page d'accueil du site. Lorsqu'un utilisateur clique sur le bouton "Déconnexion", il va être envoyé vers la route "app_logout" et Symfony va le déconnecter et le rediriger vers la page d'accueil.

5 Gestion de projet

5.1 Organisation générale

Dès la première semaine et durant presque toute la durée du projet, nous avons convenu d'un rendez-vous hebdomadaire le jeudi en début d'après-midi. Avant chaque réunion nous étions tenus d'écrire et d'envoyer à notre encadrant un bilan détaillé contenant les attentes de la semaine et les avancées réellement faites. Ensuite, après chaque réunion nous devons également écrire un compte-rendu reprenant les corrections à faire sur notre travail ainsi que les explications sur les tâches à faire durant la semaine.

5.2 Outils utilisés

5.2.1 Google Drive

Afin de stocker tous les fichiers nécessaires au bon déroulement du projet dans un endroit accessible à tout le monde, nous avons choisi d'utiliser Google Drive. Ainsi nous pouvions travailler ensemble sur les bilans ou les compte-rendus de réunions grâce à Google Docs. Nous avons également stocké les diagrammes, les maquettes et tout autre fichier que notre encadrant nous a donné à faire, sur le Drive. Notre encadrant avait bien entendu accès au Drive afin de pouvoir contrôler notre travail.

5.2.2 Overleaf

Afin d'écrire notre rapport en Latex, nous avons utilisé l'éditeur en ligne "Overleaf" qui nous a permis de travailler tous ensemble sur le rapport sans avoir besoin de télécharger d'application spécialisée. Nous étions déjà habitués à travailler avec cet outil grâce au projet de seconde année. Une fois le rapport rédigé, il nous a suffi de télécharger le pdf qui a été généré automatiquement par ce que nous avons écrit en langage Latex.

5.2.3 GitLab

Grâce au module "Techniques de communication et conduite de projets" que nous avons eu lors de notre deuxième année de licence nous avons pu assez facilement mettre en place un dépôt Git, sur la plateforme GitLab fournie par l'université, pour permettre à chaque membre du groupe de travailler depuis chez soi. On peut démarrer un dépôt Git de deux manières : soit en prenant un répertoire sur notre machine et le transformer en dépôt Git, soit en clonant un dépôt Git existant sur un serveur distant avec la commande "*git clone*".

L'outil Git est un gestionnaire de versions qui permet à chaque membre d'un projet de travailler sur un dépôt dit "local" pour ne pas modifier le travail de l'ensemble du groupe. Une fois qu'une modification de l'application web est aboutie en "local", le membre met à jour le dépôt dit "distant" en fusionnant l'ancienne version avec la nouvelle. Pour cela il doit tout d'abord faire un "commit", c'est à dire enregistrer les fichiers modifiés sur son dépôt local grâce à la commande "*git commit*" suivit d'un commentaire pour expliquer les changements qui ont été apportés lors du commit. Une fois que c'est fait, il peut faire un "*git push*" qui va mettre son dépôt "local" dans le dépôt "distant" afin que tout le monde y ait accès. Les autres membres du groupe peuvent alors mettre à jour leur propre dépôt "local" pour utiliser la nouvelle fonctionnalité en utilisant la commande "*git pull*". Cependant lorsque plusieurs membres travaillent simultanément, les versions vont entrer en conflit, c'est à dire qu'un fichier aura une version dans chaque dépôt. Pour éviter qu'ils ne rentrent en conflit sur le dépôt distant, avant chaque commit,

un "rebase" doit être fait afin d'obtenir la dernière version du projet et de régler en local les éventuels conflits.

5.2.4 PhpStorm

Afin de faciliter la création de notre application web, nous avons utilisé PhpStorm. C'est un IDE (Interface Development Environment) qui propose une interface simple et efficace pour afficher les dossiers, sous-dossiers et fichiers d'un projet. L'avantage de PhpStorm est qu'il peut se lier à notre dépôt GitLab et que l'on peut donc faire des "commit", "push" et "pull", les trois commandes que nous utilisons le plus, directement depuis l'application. Il s'occupe également de colorer nos fichiers en fonction de leurs états : non connus de Git, modifiés ou ajoutés... Il peut également identifier le framework que l'on utilise (ici Symfony) et une fois reconnu, propose un certain nombre d'options et de raccourcis tels que l'auto-complétion. Utiliser cet IDE nous aura fait gagner de manière considérable, en temps et en productivité.

5.2.5 Figma

Afin de réaliser nos maquettes nous avons utilisé Figma qui est un éditeur de graphiques vectoriels et un outil de web design. De nombreuses fonctionnalités sont disponibles et le travail en groupe de manière simultanée est possible. Ainsi nous avons pu créer et modifier nos maquettes pour avoir une idée de l'interface graphique de notre projet avant de passer à l'étape de l'implémentation.

5.3 Organisation des tâches dans le temps

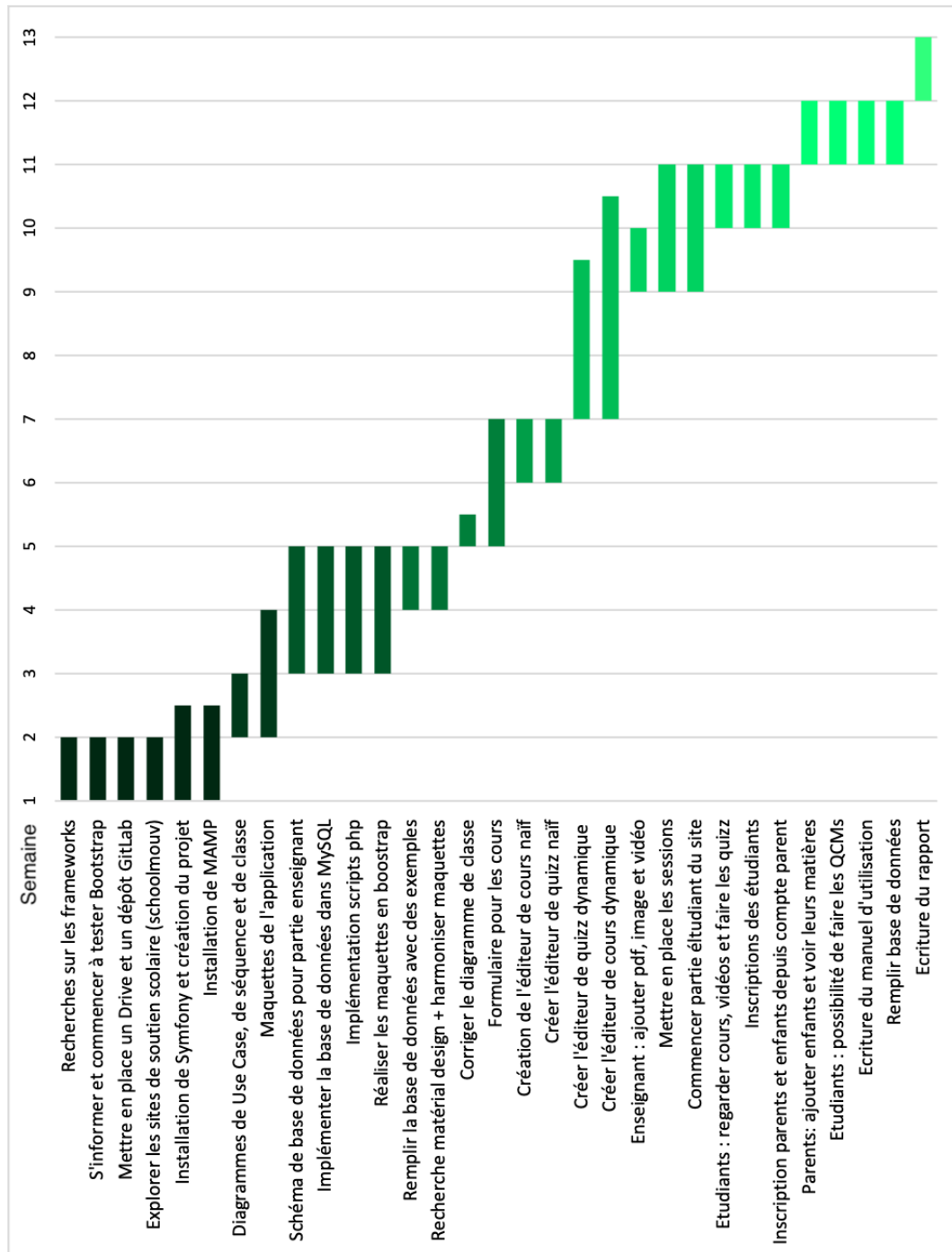


FIGURE 5.1 – Diagramme de Gantt

6 Conclusion

Pour conclure, ce projet s'est révélé très enrichissant car il nous a permis d'apprendre à travailler en groupe, à partager les tâches, à respecter les délais, à savoir présenter notre travail lors de réunions, à écrire des bilans et des comptes-rendus, à apprendre à se former nous-mêmes, à acquérir des compétences en développement logiciel front et back avec Symfony et Bootstrap...

Le problème le plus important que nous avons rencontré concerne l'éditeur de cours car il nous a demandé à tous beaucoup de travail de recherches, de réflexion et de temps. Nous n'avions jamais été confrontés à un exercice aussi difficile en JavaScript c'est pour cela que nous avons mis du temps à le réaliser et que les réunions afin de comprendre le fonctionnement ont été nombreuses. Le deuxième problème que nous avons rencontré est lié au fait que nous travaillions souvent à quatre pour essayer de résoudre les bugs, les erreurs signalées par Symfony et les parties difficiles. Travailler à quatre permet d'avoir de meilleurs résultats mais cela prend également énormément de temps.

Arrivé à la fin du projet, nous avons réussi à réaliser les fonctionnalités du cahier des charges les plus importantes pour notre application web. Cependant certaines fonctionnalités, comme le fait que l'administrateur puisse vérifier un contenu avant que l'enseignant le mette sur le site, ou encore la création de différents abonnements n'ont pas pu être mises en oeuvre par manque de temps. Ces différents points pourraient être approfondis pour avoir une application web fonctionnelle et qui puisse être hébergée et accessible au grand public.

En conclusion, nous sommes tous fiers du travail réalisé et des fonctionnalités proposées par notre application web. Nous maîtrisons, à notre niveau, les deux frameworks imposés Bootstrap et Symfony avec leurs langages : Css, Html, Twig, Php, et enfin JavaScript et ses bibliothèques.

7 Bibliographie

OpenClassrooms

<https://openclassrooms.com/fr/>

Documentation Symfony

<https://symfony.com/doc/current/index.html>

Documentation Bootstrap

<https://getbootstrap.com/docs/5.0/getting-started/introduction/>

Documentation Javascript

<https://developer.mozilla.org/fr/docs/Web/JavaScript>

Dépôt GitLab pour mustache

<https://github.com/janl/mustache.js/>

Material Design

<https://material.io/design>

Mamadou Aliou Diallo *Guide du débutant sur Symfony*

<https://www.kaherecode.com/tutorial/le-guide-du-debutant-sur-symfony4>

Lior Chamla *Chaîne contenant des vidéos d'initiation et de développement avec Symfony*

https://www.youtube.com/channel/UCS71mal_TkTW_PpZR9YLpIA

Documentation JQuery

<https://api.jquery.com/>

8 Annexes

8.1 Mode d'Emploi

Etape 1 Pour commencer, il faut télécharger et installer un environnement de serveur local. De préférence MAMP que vous pouvez trouver à cette adresse : <https://www.mamp.info/fr/downloads/>. Lors de l'installation, on vous propose d'installer MAMP Pro en plus de MAMP, ne le faite pas : décochez "MAMP PRO" et "Install Apple Bonjour". Puis mettez MAMP directement sur votre disque C : C:\MAMP

Etape 2 Il vous faut aussi le logiciel Composer, qui est un logiciel gestionnaire de dépendances pour PHP. <https://getcomposer.org/download/>.

Lors de l'installation, ne choisissez pas le mode développeur, cliquez simplement sur next. Ensuite lorsque l'on vous demande de choisir la ligne de commande PHP il faut cliquer sur browse et choisir le fichier php.ini se trouvant dans C :\MAMP\bin\php\php7.4.1\php.ini

Etape 3 Ouvrez le fichier php.ini se trouvant à C :\MAMP\bin\php\php7.4.1\php.ini avec le bloc notes. Puis faites "Ctrl"+"F" et recherchez "extension=pdo_mysql". Si un point virgule se trouve au début de la ligne, supprimez le, sinon ne faites rien.

Etape 4 Vérifiez que tout fonctionne bien en tapant sur votre environnement de commande php -v et puis composer -v. Vous devrez avoir au minimum la version PHP 7.2.4 et la version Composer 1.4.1

Etape 5 Installer git : <https://git-scm.com/downloads> et Symfony : <https://symfony.com/download>

Etape 6 Placez vous dans le dossier Symfony là où vous l'avez installé (normalement dans C :\Program Files\Symfony) et placez y le dossier du projet que nous vous avons envoyé

Etape 7 Lancez le projet avec votre éditeur de texte préféré (Par exemple Visual Studio Code, que vous pouvez télécharger ici : <https://code.visualstudio.com/>).

Etape 8 Vous pouvez maintenant lancer des commandes depuis le terminal de VS Code au lieu de votre environnement de commande extérieur.

Etape 9 Tapez la commande "composer install" dans le terminal de VS code

Etape 10 Tapez "composer require server -dev 4.4.2" dans le terminal de VS Code

Etape 11 Ouvrez MAMP, attendez que le serveur Apache et MySQL se lancent. Une fois qu'ils ont tous les deux un point vert, cliquez sur "Open WebStart page". Une fois la page internet ouverte, allez dans "Tool"->"PhpMyAdmin". Dans le menu de gauche, cliquez sur "New" afin de créer une base de données et nommez-là "bddprojet". Placez-vous dans la base de données que vous venez de créer puis cliquez sur "Import" puis "Choisir un fichier" et choisissez le fichier bddprojet.sql que nous vous avons envoyé puis cliquez sur "go".

Etape 12 Pour lancer l'application de notre projet dans votre navigateur, tapez "php bin/console server :run" dans le terminal de VS Code

8.2 Les maquettes

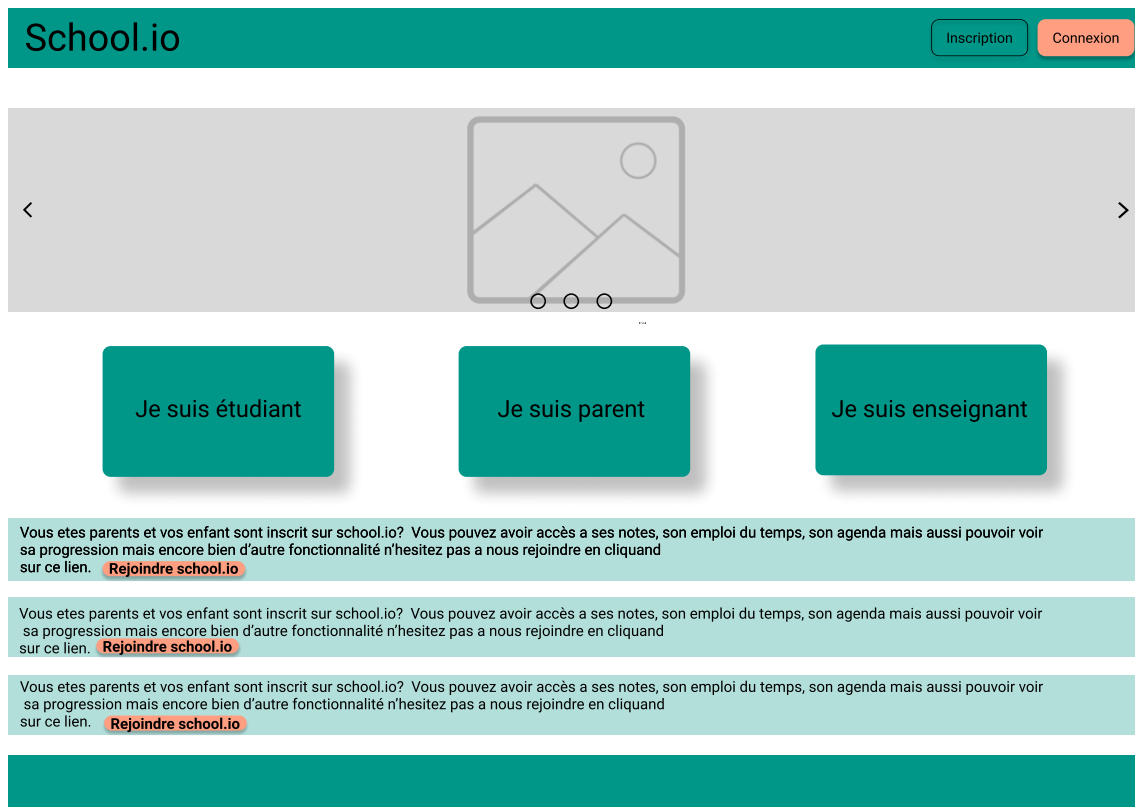



FIGURE 8.1 – Page d'accueil du site

School.io [Compte](#) [Déconnexion](#)

35

School.io



Compte

Déconnexion

Selectionner votre matière

Selectionner votre niveau

Ecrivez votre question...

Selectionner type de réponses

Cochez la ou les bonnes reponses :

☐ Réponse 1

☐ Réponse 2

☐ Réponse 3 (si choix multiple)

☐ Réponse 4 (si choix multiple)

- Choix multiple

- Vrai/Faux

+

Ajouter une autre question

FIGURE 8.4 – Page de création de quizz