République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université du 20 Août 1955 - Skikda

Faculté des Sciences Département d'Informatique

Mémoire de fin d'études

Pour obtenir le diplôme de Master

Spécialité

Réseaux et Systèmes Distribués

Thème

Plateforme d'Apprentissage Assisté

Rédigé par:

Mr. Bensalah Aymen Badreddine

Encadré par:

Mr. L. FOUGHALI

JUIN 2025

Remerciements

Je tiens à exprimer ma profonde gratitude à mon encadreur, Mr. L. FOUGHALI, pour son soutien indéfectible et son dévouement exemplaire. Je souhaite lui témoigner toute ma reconnaissance pour sa disponibilité, son accompagnement pédagogique, et sa bienveillance à mon égard, tenant compte de mes particularités avec une empathie remarquable. Bien plus qu'un mentor exceptionnel, il a endossé le rôle d'un deuxième père pour moi. Ses conseils éclairés et son mentorat ont profondément marqué mon épanouissement tant académique que personnel. Je lui suis infiniment reconnaissant pour le respect et la considération dont il a toujours fait preuve envers mes efforts. Je me considère extrêmement chanceux d'avoir pu bénéficier d'un tel guide.

Je souhaite également adresser mes remerciements les plus sincères à l'ensemble de mes enseignants pour leurs précieux conseils et leur soutien tout au long de mon parcours universitaire. Votre passion pour l'enseignement, votre engagement et votre confiance en mes capacités ont joué un rôle déterminant dans ma progression et mes réussites. Les connaissances et compétences que vous m'avez transmises constituent un bagage inestimable que je porterai avec moi dans mes futures ambitions. Merci d'avoir durablement influencé ma formation et contribué à faire de moi la personne que je suis aujourd'hui.

Enfin, je tiens à remercier chaleureusement les membres du jury d'examen pour l'intérêt porté à mon travail, malgré leurs emplois du temps chargés et obligations personnelles.

LISTE DE CONTENU :

Introduction **1**

Chapitre 1 Plateforme d'Apprentissage **4**

1.1 Plateforme d'apprentissage: 5

Système de gestion de contenu (CMS): 5

Système de gestion d'apprentissage (LMS) 6

1.2 Rôles et Permissions dans une plateforme E-Learning: 6

1.3 Design d'une plateforme d'apprentissage: 7

1.4 Les objectifs d'une plateforme d'apprentissage : 8

1.5 Conclusion 8

Chapitre 2 Le Framework Django **9**

2.1 Le Framework Django: 10

2.2 Mécanisme et Processus Django: 10

2.2.1 Modèle: 10

2.2.2 Vue: 11

2.2.3 Template: 11

2.2.4 URLs: 11

2.3 L'architecture Django: 12

2.4 Configurer Django 12

2.4.1 Installer Python: 12

2.4.2 Configurer un environnement virtuel Python: 12

2.5 Conclusion: 18

Chapitre 3 Conception **20**

3.1 Diagrammes de cas d'utilisations: 21

3.2 Diagrammes de classe : 24

3.3 Les modèles : 25
3.4 Les diagrammes de séquence: 32
3.5 Conclusion : 36
Chapitre 4 Implémentation 37
4.1 Structure du projet: 38
4.2 Paramètres globales: 40
4.3 Les vues : 43
4.4 L’interface du site web : 49
4.5 Conclusion 60
Conclusion 61

TABLE DE FIGURES :

FIGURE 1 L'ARCHITECTURE DJANGO 11

FIGURE 2 LA STRUCTURE D'UN PROJET DJANGO 13

FIGURE 3 LA PAGE PAR DEFAUT DU SERVEUR DE DEVELOPPEMENT DJANGO 16

FIGURE 4 LA STRUCTURE PROJET/APPLICATION DE DJANGO 17

FIGURE 5 LA STRUCTURE D'UNE APPLICATION DJANGO 18

FIGURE 6 DIAGRAMME D'UN VISITEUR 21

FIGURE 7 DIAGRAMME D'UN ECOLIER 22

FIGURE 8 DIAGRAMME DE L'ADMIN 23

FIGURE 9 DIAGRAMME DE CLASSE DE LA STRUCTURE DU PROGRAMME D'ETUDES 24

FIGURE 10 DIAGRAMME DE CLASSE POUR LE SYSTEME DE QUIZ 24

FIGURE 11 DIAGRAMME DE CLASSE DU SYSTEME DE SUIVI 24

FIGURE 12 DIAGRAMME DE SEQUENCE POUR CREATION DU CONTENU 32

FIGURE 13 DIAGRAMME DE CLASSE POUR ENREGISTREMENT DANS UN COURS . 33

FIGURE 14 DIAGRAMME DE SEQUENCE POUR AFFICHER LE CONTENU D'UN MODULE 35

FIGURE 15 STRUCTURE DU PROJET, L'APP COURSES ET STUDENTS 38

FIGURE 16 LES TEMPLATES DE L'APP COURSES 39

FIGURE 17 LES TEMPLATES DE L'APP STUDENTS 40

FIGURE 18 LA PAGE D'ACCUEIL MOBILE 49

FIGURE 19 LA PAGE D'ACCUEIL LARGE 50

FIGURE 20 LA LISTE DES MATIERES SUR MOBILE 51

FIGURE 21 LA LISTE DES MATIERES 52

FIGURE 22 LE CONTENU D'UNE LEÇON SUR MOBILE 53

FIGURE 23 LE CONTENU D'UNE LEÇON AVEC LA LISTE DES LEÇONS DISPONIBLES 54

FIGURE 24 LA PAGE PROFILE ET PROGRES 55

FIGURE 25 LA PAGE PROFILE ET PROGRES 56

FIGURE 26 L'AJOUT DU CONTENU (MOBILE) 57

FIGURE 27 L'AJOUT DU CONTENU 58

Introduction

Introduction :

Le ministre de l'Éducation nationale algérien, Mohamed Sghir Saadaoui, a reconnu en 2024 lors d'une intervention à l'Assemblée populaire nationale que les élèves se tournent "massivement vers les cours particuliers", qualifiant même ce phénomène de "fléau" [1]. Cette situation reflète, selon le ministre, des lacunes dans l'enseignement en classe et met en question la capacité du système éducatif public à répondre aux attentes des élèves et de leurs familles. Le recours généralisé aux cours particuliers témoigne d'un besoin accru d'un système éducatif performant et équitable, ces cours étant désormais perçus par beaucoup comme un palliatif indispensable à la réussite scolaire.

Cette situation révèle un déséquilibre préoccupant dans le système éducatif national, où les inégalités socio-économiques se traduisent directement par des inégalités d'accès au savoir. De nombreux élèves issus de milieux modestes se retrouvent exclus de ces dispositifs d'appoint faute de moyens, ce qui compromet sérieusement leurs chances de réussite. À cela s'ajoute une saturation du rythme scolaire, une perte d'autonomie chez les apprenants, et une pression accrue sur les familles.

Il apparaît donc impératif de proposer une alternative innovante, inclusive et économiquement viable. Une plateforme d'apprentissage numérique gratuite et accessible représente une réponse concrète à cette problématique. Elle permettrait non seulement d'offrir un accès équitable à des contenus pédagogiques de qualité, mais aussi de favoriser l'autonomie, la régularité dans l'apprentissage, et l'engagement des élèves grâce à des outils interactifs et personnalisés. Cette approche numérique pose également les bases pour de futures évolutions technologiques, notamment l'intégration de l'intelligence artificielle qui pourrait, à terme, permettre une personnalisation adaptative des parcours d'apprentissage, une assistance intelligente pour l'orientation scolaire, et une optimisation des méthodes pédagogiques basée sur l'analyse des données d'apprentissage.

Ainsi, ce mémoire a pour objectif principal la conception et l'implémentation d'une plateforme numérique d'apprentissage assisté. Cette plateforme vise à constituer une alternative efficace aux cours particuliers en Algérie. Pour traiter rigoureusement cette problématique, un plan

méthodologique structuré a été établi : dans un premier temps, nous définirons précisément ce qu'est une plateforme d'apprentissage assisté, ses objectifs pédagogiques, ses cas d'usage, ainsi que les différents rôles des utilisateurs impliqués pour optimiser l'interaction pédagogique (Chapitre I). Le Chapitre II sera consacré à la présentation des technologies choisies pour concevoir cette plateforme, notamment l'utilisation de Framework offrant des bibliothèques de code réutilisable afin d'assurer la qualité technique du produit final. Dans le Chapitre III, nous illustrerons les fonctionnalités proposées par des diagrammes précis. Enfin, le Chapitre IV détaillera et expliquera l'implémentation pratique du projet à travers le code développé.

Nous conclurons ce mémoire par une synthèse des résultats obtenus, ainsi que par une présentation des perspectives d'évolution de la plateforme, notamment les possibilités d'intégration future de l'intelligence artificielle pour enrichir davantage l'expérience d'apprentissage, démontrant ainsi l'efficacité et le potentiel d'évolution de la solution proposée.

Chapitre 1

Plateforme d'Apprentissage

Dans ce chapitre nous allons découvrir le monde des plateformes d’apprentissage, aussi appelé “E-learning platforms”, ces plateformes ont redéfinies l’éducation en offrant un environnement virtuel qui donne accès à des ressources éducatives et la capacité de collaboration entre écolier et enseignant. Les composants principaux d’une plateforme E-learning sont comme suit: système de gestion du contenu (CMS), système de gestion d’apprentissage (LMS). De plus, on va voir les différents rôles utilisateur dans une plateforme d’apprentissage, leurs permissions, et les objectifs, bénéfices et inconvénients de cette technologie.

1.1 Plateforme d’apprentissage:

Une plateforme d'apprentissage est un dispositif technologique et humain qui intègre des outils informatisés à des fins d'enseignement et d'apprentissage. Il a pour finalité l'accès à distance aux contenus pédagogiques, l'individualisation de l'apprentissage et le télé tutorat. Il peut contenir les fonctionnalités suivantes (ensemble ou séparés):

- Système de gestion du contenu (CMS)
- Système de gestion d’apprentissage (LMS)
- Apprentissage Collaboratif Assisté par Ordinateur (CSCL)

Système de gestion de contenu (CMS):

CMS est l’acronyme de Content Management System. Ce terme désigne les logiciels qui servent à créer, gérer, et mettre à jour des sites internet et des applications mobiles. Les caractéristiques d’un CMS sont :

- Il peut être utilisé par plusieurs personnes en même temps ;
- Il propose une interface pour publier des contenus (pages, articles, images, vidéos, etc.) ;
- Il permet de gérer séparément la forme et le contenu du site web ou de l’appli mobile.

Un CMS est également capable de hiérarchiser ses utilisateurs, selon des critères et des autorisations particulières (compte auteur, administrateur, contributeur, SEO manager, abonné, etc.). C’est pourquoi il repose sur l’exploitation de nombreuses données marketing.[3]

Système de gestion d’apprentissage (LMS)

Un LMS peut être un logiciel d’apprentissage ou une plateforme de formation, permettant de centraliser et gérer des activités d’apprentissage en ligne (e-learning).Il offre des fonctionnalités permettant de créer, publier et vendre des contenus pédagogiques (cours, exercices, évaluations, etc.). Il permet aussi de gérer les inscriptions des apprenants, suivre leur progression et évaluer leurs compétences.

Un LMS est généralement utilisé par :

- Des enseignants
- Des formateurs
- Des entreprises
- Des organismes de formation
- Tout autre acteur de la formation, de l’éducation, de la pédagogie ou de l’apprentissage

Un LMS peut aussi être utilisé pour soutenir et gérer l’apprentissage en présentiel.[4]

L’Apprentissage Collaboratif Assisté par Ordinateur (CSCL) est une méthode d'apprentissage collaboratif utilisant l'informatique et Internet. Son objectif est d'accompagner les écoliers dans un apprentissage collaboratif efficace.

1.2 Rôles et Permissions dans une plateforme E-Learning:

Dans un LMS, chaque utilisateur a un rôle basé-permission pour déterminer le niveau d’accès à différentes régions du LMS.

Utilisateur:

Principalement les écoliers qui ont accès au contenu LMS. Souvent un accès limité à la page d'accueil et les cours ou ils sont enregistrés.

Administrateur:

Les administrateurs du LMS ont accès complet au système, ils peuvent gérer les utilisateurs, créer des cours.

Administrateur groupe:

Chaque groupe a des administrateurs qui supervisent et contrôlent l’activité au sein de leurs groupes respectifs.

1.3 Design d’une plateforme d’apprentissage:

L'expérience utilisateur (UX) est un aspect essentiel de la conception et du développement d'une plateforme d’apprentissage efficace. Les principes de l'UX contribuent à minimiser la charge mentale liée au traitement de ce que l’élève voit à l’écran tout au long de leur parcours d'apprentissage. Les caractéristiques d'une bonne conception UX sont les suivantes

- Cohérence: Maintenez des éléments de conception, des mises en page et une navigation cohérente pour plus de familiarité.
- N’afficher que les principaux éléments de navigation et d'interaction: Surveillez l’utilité de tous les éléments sur la page, affichez les éléments optionnels seulement si nécessaire; supprimer les éléments d’interaction inutiles et n’afficher que les éléments de navigation les plus utiles.
- Feedback: Fournissez un feedback régulier via des quiz, des exercices et des rapports d'avancement pour suivre les progrès et améliorer l'expérience d'apprentissage.
- Accessibilité: assurer la compatibilité avec les technologies d’assistance, s’adapter aux utilisateurs ayant des capacités différentes et améliorer l’engagement.

1.4 Les objectifs d'une plateforme d'apprentissage :

- La flexibilité dans le travail : En choisissant de poursuivre vos études en ligne, vous bénéficierez de plus de souplesse pour concilier votre carrière, vos études et votre vie privée, sans subir les inconvénients d'un horaire fixe.[5]
- Développer des compétences et d'acquérir des connaissances, souvent dans des cours formels ou semi-informels. Voici des exemples d'utilisation de solutions numériques pour l'apprentissage :[6]
  - Cours d'autoformation numérique
  - Formations en ligne
  - Jeux d'apprentissage numériques, jeux sérieux
  - Formations en ligne ouvertes à tous (MOOC)
  - Micro apprentissage
  - Formations qui se déroulent dans des salles virtuelles (par exemple, des visioconférences)
  - Enseignement ou cours soutenus sur des plateformes numériques

1.5 Conclusion

Les plateformes numériques d'apprentissage bouleversent les pratiques éducatives traditionnelles en redéfinissant l'accès au savoir et la manière d'enseigner. Elles offrent aux apprenants une accessibilité immédiate à des contenus riches, variés et constamment mis à jour. Grâce à une interface interactive et immersive, les élèves évoluent dans un cadre motivant, propice à la concentration et à la progression autonome.

Ces outils représentent bien plus qu'un simple support : ils facilitent la planification des cours, permettent un meilleur suivi des acquis et ouvrent la voie à des approches pédagogiques différenciées. L'intégration d'exercices interactifs, de feedbacks instantanés et de tableaux de bord personnalisés renforce l'efficacité de l'enseignement et l'engagement des élèves.

Au-delà de la classe, ces plateformes instaurent une culture d'apprentissage continue, centrée sur l'élève, agile et évolutive. Elles préfigurent l'école de demain : inclusive, connectée et résolument tournée vers l'excellence.

Chapitre 2

Le Framework Django

Dans ce chapitre, nous explorons les mécanismes et processus du Framework Django. Nous discutons des composants principaux, avantages, instructions étape par étape pour commencer avec Django.

2.1 Le Framework Django:

Django est un Framework open-source pour applications web backend basé sur Python - l'un des langages de développement web les plus populaires. Ses objectifs principaux sont la simplicité, flexibilité, fiabilité et scalabilité. Lorsqu'on découvre Django, Python et ses fonctionnalités se révèlent sous un nouveau jour.

Django possède son propre système de nommage pour toutes ses fonctions et composants (par exemple, les réponses HTTP sont appelées "views"). Il inclut également un panneau d'administration, considéré comme plus facile à utiliser que ceux de Lavarel ou Yii, ainsi que d'autres fonctionnalités techniques comme :

- Une syntaxe simple
- Son propre serveur web
- Une architecture cœur de type MVC (Modèle-Vue-Contrôleur)
- Le principe "Batteries included" (inclut tous les éléments essentiels pour résoudre les cas courants)
- Un ORM (Object Relational Mapper)
- Des bibliothèques HTTP
- Le support de middleware
- Un Framework de tests unitaires Python

2.2 Mécanisme et Processus Django:

Django suit le modèle de conception MVT (Modèle-Vue-Template).

- Modèle : Les données à présenter, généralement issues d'une base de données
- Vue : Un gestionnaire de requêtes qui renvoie le Template et contenu appropriés - basé sur la requête de l'utilisateur
- Template : Un fichier texte (comme un fichier HTML) contenant la mise en page de la page web, avec la logique d'affichage des données

2.2.1 Modèle:

Un modèle est la source d’information unique et définitive à propos de vos données. Il contient les champs et le comportement essentiels des données que vous stockez. Généralement,

chaque modèle correspond à une seule table de base de données. Les modèles se trouvent généralement dans un fichier appelé *models.py*

2.2.2 Vue:

Une vue est une fonction ou méthode qui prend des requêtes HTTP comme arguments, importe le(s) modèle(s) pertinent(s), détermine quelles données envoyer au Template, et retourne le résultat final. Les vues sont généralement situées dans un fichier appelé views.py.

2.2.3 Template:

Les templates (en français on utilise les mots " patron " ou " gabarit ") est un moyen de séparer le contenu d'un site internet au code . Ainsi chaque cœur de métier peut se concentrer sur ses besoins. Pour faire simple tout ce qui est HTML se trouve dans les templates .

2.2.4 URLs:

Django fournit également un moyen de naviguer entre les différentes pages d'un site web. Quand un utilisateur demande une URL, Django décide à quelle vue l'envoyer. Ceci est géré dans un fichier appelé *urls.py*.

La figure suivante montre comment Django traite les requêtes et comment le cycle requête/réponse est géré avec les différents composants principaux de Django : URLs, vues, modèles et templates:

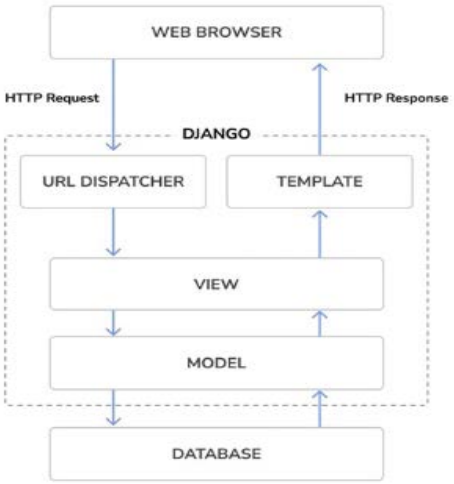

Figure 1 L'architecture Django[8]

2.3 L'architecture Django:

Lorsque vous avez installé Django et créé votre première application web Django, et que le navigateur demande une URL, voici ce qui se produit :

1. Django reçoit l'URL, consulte le fichier urls.py et appelle la vue correspondant à cette URL.
2. La vue, située dans views.py, recherche les modèles pertinents.
3. Les modèles sont importés depuis le fichier models.py.
4. La vue envoie ensuite les données à un Template spécifique dans le dossier des templates.
5. Le Template contient du HTML et des balises Django, et avec les données reçues, il renvoie un contenu HTML final au navigateur.

Django peut faire beaucoup plus que cela, mais ce sont les bases que vous apprendrez dans ce tutoriel, et ce sont les étapes fondamentales pour une application web simple développée avec Django.

2.4 Configurer Django

2.4.1 Installer Python:

Django 4 supporte les versions Python 3.8 à 3.10, dans Linux/MacOS, Python sera déjà installé. Si vous avez Windows, vous pouvez télécharger Python depuis un navigateur web.

2.4.2 Configurer un environnement virtuel Python:

Pour créer un environnement virtuel Python, ouvrez une invite de commande, naviguez au dossier du projet, et entrez la commande suivante:

- Pour Linux/MacOS:

```
aymen@home~\myapp\educa>python -m venv my_env
```

- Pour Windows:

```
D:\myapp\educa>py -m venv my_env
```

La commande précédente va créer un environnement Python dans un nouveau répertoire nommé my_env/

Pour activer votre environnement virtuel, exécutez la commande suivante:

- Pour Linux or macOS:

```
source my_env/bin/activate
```

- Pour Windows:

```
.\my_env\Scripts\activate
```

Installer Django:

Pip est un système de gestion de paquets utilisé pour installer et gérer des librairies écrites en Python. On va l'utiliser pour installer et importer plusieurs bibliothèques. Python 3.10 inclut pip préinstallé. Pour installer Django avec pip, exécutez la commande suivante dans l'invite de commande:

```
pip install Django
```

Créer un projet:

Pour les nouveaux utilisateurs de Django, la configuration initiale nécessite de générer le code pour créer un projet Django. Cela inclut les paramètres pour l'instance Django, la configuration de la base de données et les options spécifiques à l'application. Voici un exemple pour créer une application:

```
(my_env) D:\myapp>django-admin startproject elearning
```

Ceci va créer un projet avec le nom "elearning". Voici la structure du projet créé:

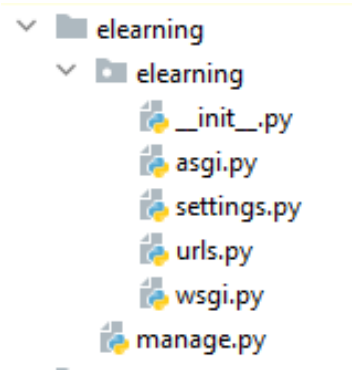

Figure 2 La structure d'un projet Django

Le répertoire externe elearning/ est le conteneur de notre projet. Il contient les fichiers suivants :

- elearning/ : Ceci est le package Python de votre projet, qui comprend les fichiers suivants :
  - __init__.py : Un fichier vide qui indique à Python de traiter le répertoire elearning comme un module Python.
  - asgi.py : Ce fichier configure l'exécution de votre projet en tant qu'application ASGI (Asynchronous Server Gateway Interface) avec des serveurs web compatibles ASGI. ASGI est la nouvelle norme Python pour les serveurs et applications web asynchrones.
  - settings.py : Ce fichier contient les paramètres et configurations de votre projet, incluant les réglages par défaut initiaux.
  - urls.py : C'est ici que se trouvent les motifs d'URL de votre projet. Chaque URL définie ici est associée à une vue.
  - wsgi.py : Ce fichier configure l'exécution de votre projet en tant qu'application WSGI (Web Server Gateway Interface) avec des serveurs web compatibles WSGI.
- manage.py : Ceci est un utilitaire en ligne de commande utilisé pour interagir avec votre projet. Vous n'avez pas besoin de modifier ce fichier.

Appliquer les migrations initiales dans la base des données:

Les applications Django nécessitent une base de données pour le stockage des données. Configurez la base de données dans le fichier settings.py. Utilisez la liste INSTALLED_APPS pour les applications par défaut. Créez des modèles pour le mappage des données et utilisez les migrations pour créer les tables. Exécutez les commandes suivantes :

```
(my_env) D:\myapp>cd elearning
(my_env) D:\myapp\elearning>python manage.py migrate
```

Vous verrez une sortie qui se termine par les lignes suivantes :

```
Operations to perform:
Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
     Applying contenttypes.0001_initial... OK
     Applying auth.0001_initial... OK
     Applying admin.0001_initial... OK
```

```
    Applying admin.0002_logentry_remove_auto_add... OK
    Applying admin.0003_logentry_add_action_flag_choices... OK
    Applying contenttypes.0002_remove_content_type_name... OK
    Applying auth.0002_alter_permission_name_max_length... OK
    Applying auth.0003_alter_user_email_max_length... OK
    Applying auth.0004_alter_user_username_opts... OK
    Applying auth.0012_alter_user_first_name_max_length... OK
    Applying sessions.0001_initial... OK
```

Exécution du serveur de développement:

Django inclut un serveur web léger qui permet d'exécuter le code sans configurer le serveur. Démarrez le serveur de développement en utilisant cette commande dans l'invite de commande:

```
python manage.py runserver
```

La console va alors afficher :

```
Watching for file changes with StatReloader
Performing system checks...
System check identified no issues (0 silenced).
June 07, 2025 - 12:30:27
Django version 5.2.2, using settings 'elearning.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Maintenant, ouvrez http://127.0.0.1:8000/ dans votre navigateur. Vous devriez voir une page indiquant que le projet fonctionne correctement, comme illustré dans la figure suivante :

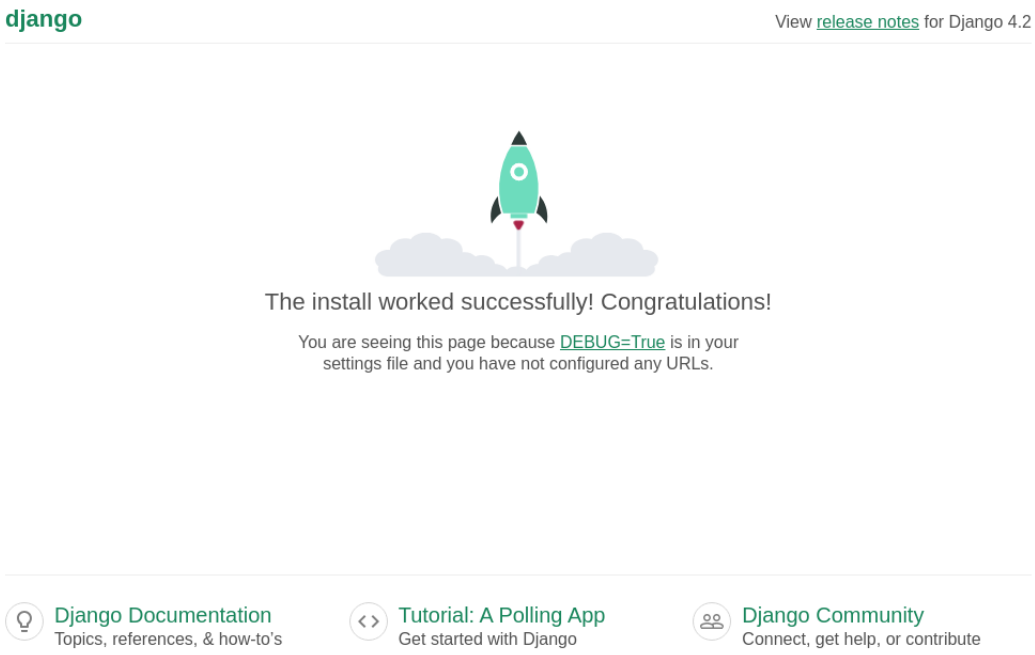

Figure 3 La page par défaut du serveur de développement Django

Les paramètres du projet:

- *DEBUG :* Un booléen qui active ou désactive le mode débogage. Lorsqu'il est défini sur True, des pages d'erreur détaillées s'affichent pour les exceptions non capturées. En production, il doit être défini sur False pour éviter d'exposer des données sensibles.
- *ALLOWED_HOSTS :* Un paramètre qui spécifie les domaines/hôtes autorisés à héberger votre site Django. Il n'est pas appliqué en mode débogage ou pendant les tests. Vous devez ajouter votre domaine/hôte à ce paramètre lors du passage en production avec DEBUG défini sur False.
- *INSTALLED_APPS :* Un paramètre qui liste les applications actives de votre site.
- *MIDDLEWARE :* Les middlewares effectuent diverses fonctions pendant le traitement des requêtes/réponses.
- *ROOT_URLCONF :* Spécifie le module Python où sont définies les configurations URL racines de votre application.

- *DATABASES :* Un dictionnaire contenant les paramètres de toutes les bases de données utilisées dans le projet. Il doit toujours y avoir une base de données par défaut, et la configuration par défaut utilise SQLite3.
- *LANGUAGE_CODE:* Définit le code de langue par défaut pour le site Django.
- *USE_TZ:* Indique à Django d'activer ou désactiver le support des fuseaux horaires. Django prend en charge les dates et heures sensibles au fuseau horaire. Par défaut, ce paramètre est True lors de la création d'un nouveau projet.

Créer une application:

Dans Django, un projet est considéré comme une installation Django avec certains paramètres. Une application est un ensemble de modèles, vues, templates et URLs. Les applications interagissent avec le Framework pour fournir des fonctionnalités spécifiques et peuvent être réutilisées dans divers projets.

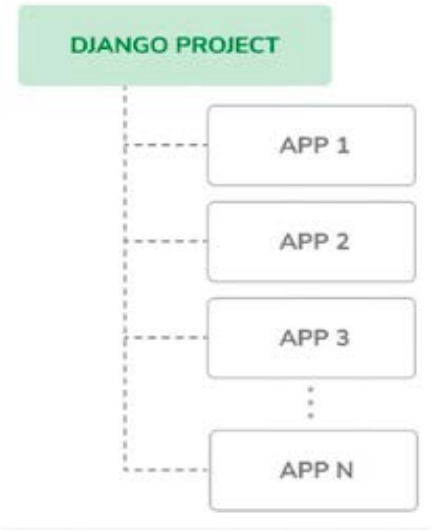

Figure 4 La structure projet/application de Django

Exécutez la commande suivante dans l'invite de commande:

```
(my_env) D:\myapp\elearning>python manage.py startapp myplatforme
```

Ceci créera la structure de base de l'application :

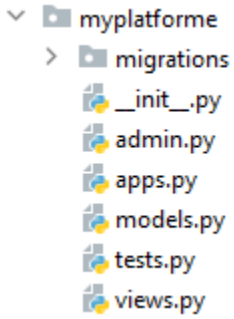

Figure 5 La structure d'une application Django

Les fichiers dans le répertoire blog sont :

- migrations : Répertoire pour gérer les migrations de base de données.
- init.py : Un fichier vide indiquant qu'il s'agit d'un module Python.
- admin.py : Enregistre les modèles pour le site d'administration de Django (optionnel).
- apps.py : Fichier de configuration principal pour l'application blog.
- models.py : Contient les modèles de données pour l'application.
- tests.py : Fichier pour ajouter des tests à l'application.
- views.py : Fichier pour gérer la logique et traiter les requêtes HTTP.

Avec la structure de l'application en place, vous pouvez commencer à créer des modèles de données, des URLs, des vues (Views) et des templates pour le blog.

2.5 Conclusion:

Ce chapitre a offert une vue d’ensemble claire du Framework Django, en expliquant ses mécanismes fondamentaux et les principales étapes de son utilisation. Nous avons appris à configurer un environnement virtuel, à créer un projet, puis à y intégrer une application. Ces étapes constituent le socle de tout développement web avec Django.

En suivant cette démarche, nous sommes désormais en mesure de structurer nos propres applications en tirant pleinement parti des atouts du Framework : une architecture robuste, une organisation modulaire, une gestion simplifiée des bases de données et une interface d'administration puissante. Django ne se contente pas d'accélérer le développement : il impose de bonnes pratiques, renforce la sécurité et garantit la maintenabilité du code à long terme.

Ainsi, maîtriser Django, c'est acquérir un outil professionnel, éprouvé, et capable de répondre aux exigences des projets les plus ambitieux.

Chapitre 3

Conception

Ce chapitre présente des diagrammes UML personnalisés, incluant des diagrammes de cas d'utilisation, de classes, de séquence et des présentations ORM, pour notre application e-learning. Ces diagrammes offrent une représentation visuelle de la structure, des fonctions et des relations au sein de notre application.

Les diagrammes UML présentés mettent en lumière l'architecture et les interactions de notre application, favorisant la communication et la collaboration tout au long des étapes de conception et de développement.

3.1 Diagrammes de cas d'utilisations:

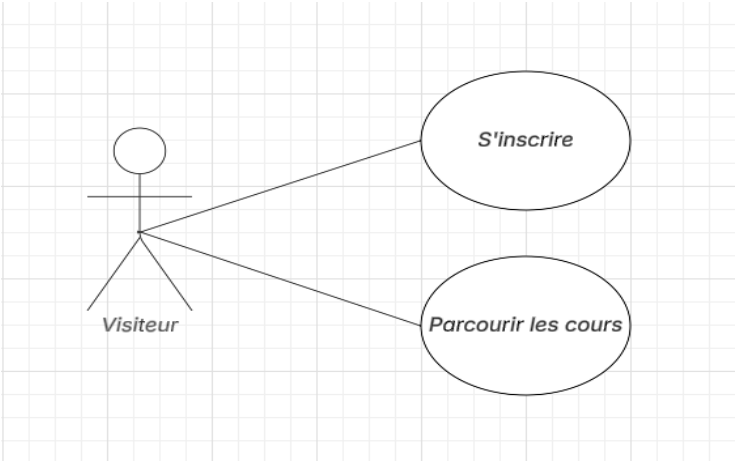

Figure 6 Diagramme d'un visiteur

Le diagramme illustre un acteur nommé **Visiteur** (un utilisateur non connecté) qui interagit avec un système (ex : plateforme éducative) via 2 cas d'utilisation :

- **S'inscrire** : Le visiteur peut créer un compte pour devenir utilisateur enregistré.
- **Parcourir les cours** : Le visiteur peut consulter la liste des cours disponibles sans s'authentifier.

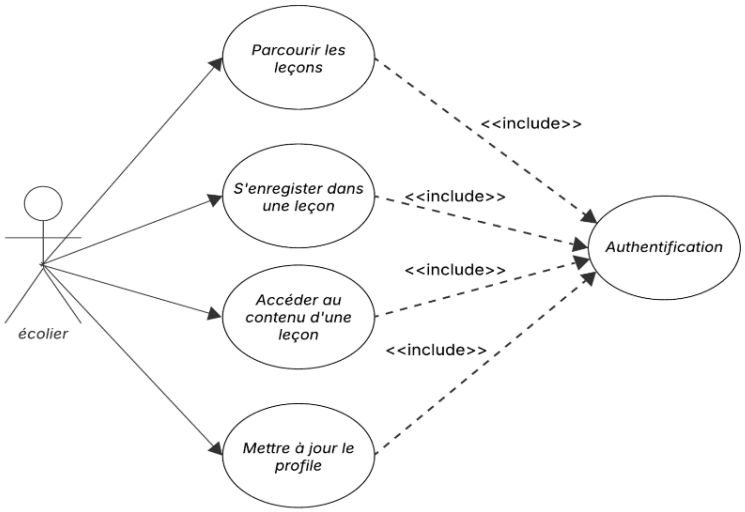

Figure 7 Diagramme d'un écolier

Un **écolier** est un utilisateur connecté ayant un accès limité au contenu pédagogique de la plateforme plus la possibilité de modifier son propre profile.

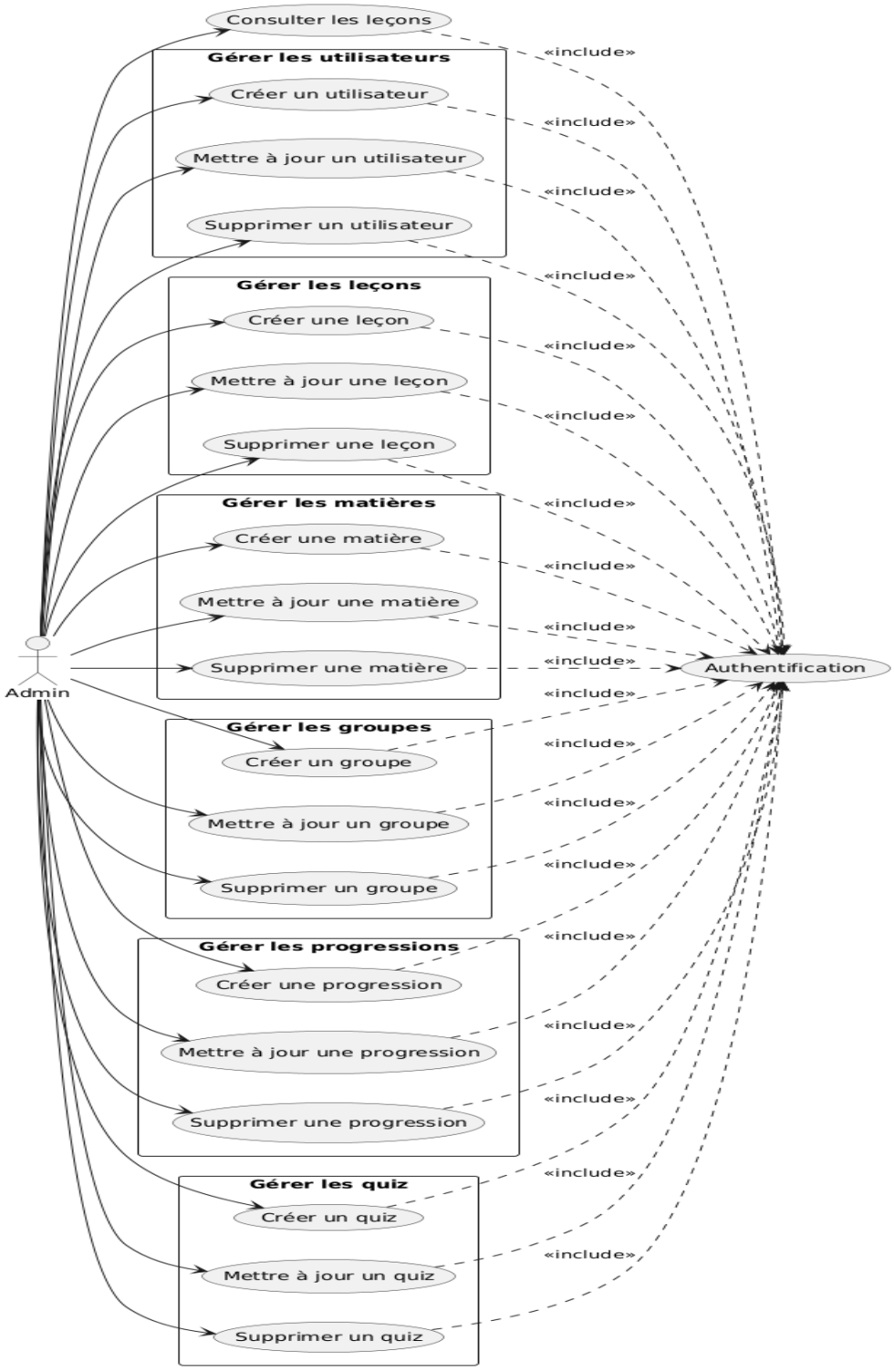

Figure 8 Diagramme de l'admin

L'**admin** a contrôle total de la plateforme, de la gestion du contenu à la gestion des utilisateurs ou visiteurs.

3.2 Diagrammes de classe :

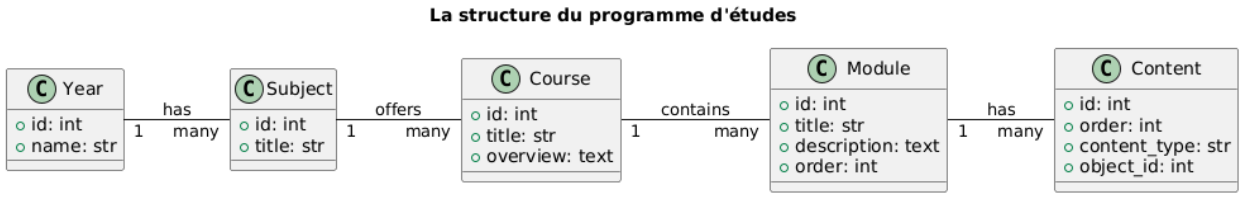

Figure 9 Diagramme de classe de la structure du programme d'études

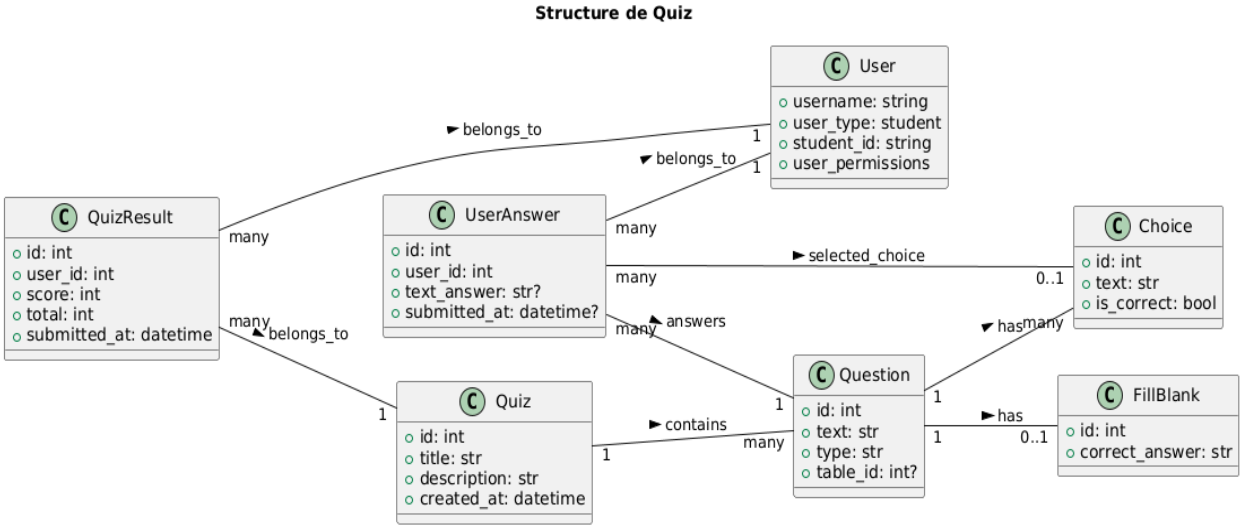

Figure 10 Diagramme de classe pour le système de quiz

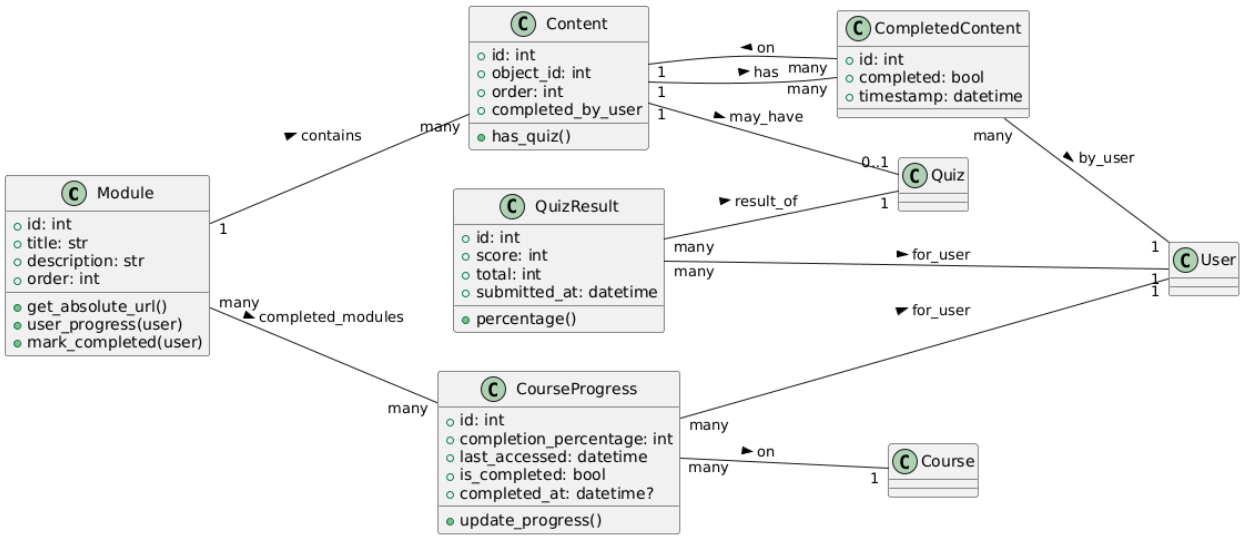

Figure 11 Diagramme de classe du système de suivi

3.3 Les modèles :

Les classes de modèles de Django sont un composant essentiel du système de mappage objet-relationnel (ORM) du Framework. Elles représentent les tables de base de données et définissent la structure et le comportement des données qu'elles contiennent. L'ORM de Django est un outil puissant pour gérer les requêtes intégrées au Framework. Les migrations sont une fonctionnalité utile. Les requêtes de l'ORM de Django simplifient la récupération, le filtrage et la gestion des données. Voici quelques modèles selon les fonctionnalités existantes dans notre plateforme :

```
class Year(models.Model):
    name = models.CharField(max_length=20, unique=True)
    class Meta:
        ordering = ['id']
    def __str__(self):
        return self.name
class Subject(models.Model):
    title = models.CharField(max_length=200, null=True, blank=True)
    year = models.ForeignKey(Year, on_delete=models.CASCADE)
    slug = models.SlugField(max_length=200, unique=True)
    description = RichTextUploadingField(null=True, blank=True)
    def __str__(self):
        return f"{self.year.name} - {self.title}”
```

La classe Year représente une année scolaire, par exemple "2024", "6e", ou "1ère année". Elle contient un champ name pour le nom de l’année, qui doit être unique. Cette classe est utilisée pour regrouper les matières par année scolaire. Lorsqu’on affiche un objet de cette classe, c’est son nom qui est retourné.

La classe Subject correspond à une matière scolaire, comme "Français" ou "Mathématiques". Chaque matière est liée à une année (Year). Elle possède un titre, un identifiant (slug) utilisé dans les URL, et une description au format enrichi grâce à un éditeur de texte comme CKEditor. Quand on affiche un objet Subject, on voit à la fois l’année et le nom de la matière.

```
class Course(models.Model):
   owner = models.ForeignKey(settings.AUTH_USER_MODEL,
                             related_name='courses_created',
                             on_delete=models.CASCADE)
    subject = models.ForeignKey(
        Subject,
        related_name='courses',
        on_delete=models.CASCADE
    )
```

```
    title = models.CharField(max_length=200, db_index=True)
    slug = models.SlugField(max_length=200, unique=True, db_index=True)
    overview = RichTextUploadingField(null=True, blank=True)
    published = models.BooleanField(default=False)
    published_date = models.DateTimeField(null=True, blank=True)
    students = models.ManyToManyField(
        settings.AUTH_USER_MODEL,
        related_name='courses_joined',
        blank=True
    )
    def save(self, *args, **kwargs):
        if not self.slug:
            self.slug = slugify(self.title)
            original_slug = self.slug
            counter = 1
            while Course.objects.filter(slug=self.slug).exists():
                self.slug = f"{original_slug}-{counter}"
                counter += 1
        super().save(*args, **kwargs)
    def get_absolute_url(self):
        return reverse('course_detail', args=[self.slug])
    class Meta:
        ordering = ['-published_date']
    def __str__(self):
        return self.title
```

La classe Course représente un cours précis créé par un admin. Chaque cours appartient à une matière (Subject) et contient un titre, une introduction enrichie, un état de publication, une date de publication, et une liste d'élèves inscrits. Le champ slug permet de générer automatiquement une adresse URL unique à partir du titre du cours. Lorsqu'un cours est publié, il peut être consulté par les élèves. Le lien entre cours et créateur est assuré par le champ owner.

```
class Module(models.Model):
    course = models.ForeignKey(
        Course,
        related_name='modules',
        on_delete=models.CASCADE
    )
    title = models.CharField(max_length=200)
    description = models.TextField(blank=True)
    order = OrderField(blank=True, for_fields=['course'])
    class Meta:
        ordering = ['order']
    def __str__(self):
        return f'{self.order}. {self.title}'
    def get_absolute_url(self):
```

```
        return reverse('student_course_detail_module', args=[self.course.id,
self.id])
    def user_progress(self, user):
        total = self.contents.count()
        if total == 0:
            return {'completed': 0, 'total': 0, 'percentage': 0}
        completed = CompletedContent.objects.filter(
            user=user,
            content__module=self
        ).count()
        return {
            'completed': completed,
            'total': total,
            'percentage': int((completed / total) * 100)
        }
    def mark_completed(self, user):
        CompletedContent.objects.get_or_create(
            user=user,
            content__module=self,
            defaults={'completed': True}
        )
        progress, _ = CourseProgress.objects.get_or_create(
            user=user,
            course=self.course
        )
        progress.completed_modules.add(self)
        progress.update_progress()
```

La classe **Module** est une subdivision d'un cours. Par exemple, un cours de français peut contenir plusieurs modules. Chaque module est lié à un cours et contient un titre, une description, et un ordre d'apparition. La classe contient aussi des méthodes pour suivre la progression de l'élève dans le module, comme le pourcentage de contenus terminés, et pour marquer un module comme complété.

```
class Content(models.Model):
    module = models.ForeignKey(
        Module,
        related_name='contents',
        on_delete=models.CASCADE
    )
    content_type = models.ForeignKey(
        ContentType,
        on_delete=models.CASCADE,
        limit_choices_to={
            'model__in': ('text', 'video', 'image', 'file')
        },
```

```
    )
    object_id = models.PositiveIntegerField()
    item = GenericForeignKey('content_type', 'object_id')
    order = OrderField(blank=True, for_fields=['module'])
    class Meta:
        ordering = ['order']
    def __str__(self):
        if self.item:
            return f"{self.module.title} - {str(self.item)}"
        return f"{self.module.title} - {self.content_type.model} (ID:
{self.object_id})"
    def has_quiz(self):
        try:
            return bool(self.quiz)
        except Quiz.DoesNotExist:
            return False
    @property
    def completed_by_user(self):
        return self.CompletedContent_set.filter(user=self._request.user).exists()
```

La classe **Content** représente un bloc de contenu générique appartenant à un module d'un cours. Elle utilise le système de contenu générique de Django (GenericForeignKey) pour permettre à chaque instance de faire référence dynamiquement à un objet d'un autre modèle, comme **Text**, Video, Image ou File. Cela rend la structure flexible et extensible pour différents types de contenus éducatifs. Chaque contenu est également associé à un module via une clé étrangère et ordonné grâce au champ personnalisé OrderField, ce qui permet d'afficher les contenus dans un ordre précis. La classe inclut une méthode has_quiz() pour vérifier si un quiz est lié au contenu, ainsi qu'une propriété (à améliorer) pour savoir si ce contenu a été complété par un utilisateur. Cette architecture permet d'organiser les contenus d'apprentissage de manière modulaire et réutilisable tout en gardant une interface commune.

Un **système de quiz** est un outil interactif d'évaluation intégré dans une plateforme d'apprentissage. Il permet aux écoliers de tester leurs connaissances en répondant à une série de questions de types variés, comme les choix multiples, le vrai ou faux, ou les textes à compléter. Chaque quiz est généralement associé à un module ou une leçon spécifique, et les réponses des utilisateurs sont automatiquement corrigées pour fournir un score ou un feedback immédiat. Ce type de système facilite l'autoévaluation. Voici nos modèles pour ce système :

```
class Quiz(models.Model):
    content = models.OneToOneField(Content, on_delete=models.CASCADE,
related_name='quiz')
    title = models.CharField(max_length=200)
    description = models.TextField(blank=True)
    created_at = models.DateTimeField(auto_now_add=True)
    def __str__(self):
        return self.title
    def render(self):
        from django.template.loader import render_to_string
        return render_to_string('courses/content/quiz.html', {'quiz': self})
class Question(models.Model):
    quiz = models.ForeignKey(Quiz, on_delete=models.CASCADE,
related_name='questions')
    text = models.TextField()
    type = models.CharField(
        max_length=50,
        choices=[
            ('choice', 'Choice'),
            ('fill_blank', 'Fill in the blank'),
        ],
        default='choice'
    )
    def __str__(self):
        from django.utils.html import strip_tags
        return strip_tags(self.text)[:50]
class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE,
related_name='choices')
    text = models.CharField(max_length=255)
    is_correct = models.BooleanField(default=False)
    def __str__(self):
        return self.text
class FillBlank(models.Model):
    question = models.OneToOneField(Question, on_delete=models.CASCADE,
related_name='fill_blank')
    correct_answer = models.CharField(max_length=255)
      def __str__(self):
        return self.correct_answer
```

Quiz représente un questionnaire lié à un bloc de contenu ; il porte un titre, une description optionnelle et la date de création, et il peut être rendu sous forme de Template HTML pour l'affichage aux apprenants.

Question appartient à un quiz donné ; elle stocke l'énoncé, le type de question (choix multiple, texte à trous, glisser-déposer, etc.) et, si nécessaire, une référence vers une table associée pour les questions basées sur un tableau.

Choice correspond à une proposition de réponse pour une question de type choix multiple ; elle mémorise le texte affiché et indique si cette réponse est la bonne.

FillBlank gère la réponse correcte d'une question à texte; il est lié à la question concernée et enregistre simplement la solution attendue.

```
class UserAnswer(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice = models.ForeignKey(Choice, null=True, blank=True,
on_delete=models.SET_NULL)
    text_answer = models.TextField(null=True, blank=True)
    submitted_at = models.DateTimeField(auto_now_add=True, null=True)
    def is_correct(self):
        return self.choice and self.choice.is_correct
    def __str__(self):
        return f'{self.user} - {self.question}'
class QuizResult(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    quiz = models.ForeignKey(Quiz, on_delete=models.CASCADE)
    score = models.IntegerField()
    total = models.IntegerField()
    submitted_at = models.DateTimeField(auto_now_add=True)
    class Meta:
        unique_together = ('user', 'quiz')
    def percentage(self):
        return round((self.score / self.total) * 100) if self.total > 0 else 0
    def __str__(self):
        return f"{self.user.username} - {self.quiz.title}:
{self.score}/{self.total}"
```

UserAnswer conserve la réponse soumise par un utilisateur ; il relie l'apprenant à la question et stocke soit l'option choisie, soit un texte libre, ainsi que l'horodatage de la soumission et une méthode pour vérifier la justesse de la réponse.

QuizResult récapitule le score obtenu par un utilisateur pour un quiz donné ; il enregistre le nombre de bonnes réponses, le total, la date de soumission et fournit un calcul du pourcentage de réussite.

Pour la progression des écoliers, on a créé les modèles suivants :

```
class CompletedContent(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    content = models.ForeignKey(Content, on_delete=models.CASCADE)
    completed = models.BooleanField(default=False)
    timestamp = models.DateTimeField(auto_now_add=True)
    class Meta:
        unique_together = ('user', 'content')
    def __str__(self):
        try:
            return f'{self.user.username} - {self.content.title} - {"Completed" if
self.completed else "Not Completed"}'
        except AttributeError:
            return f'{self.user.username} - {self.content} - {"Completed" if
self.completed else "Not Completed"}'
```

La classe **CompletedContent** permet de suivre la progression d'un utilisateur sur un contenu spécifique. Elle enregistre si l'utilisateur a complété ce contenu à l'aide du champ booléen completed, ainsi que la date et l'heure de cette interaction grâce au champ timestamp. La contrainte unique_together garantit qu'une association entre un même utilisateur et un même contenu n'existe qu'une seule fois dans la base de données. La méthode __str__ fournit une représentation lisible de chaque enregistrement, indiquant le nom de l'utilisateur, le titre du contenu (ou une valeur par défaut s'il n'est pas disponible), et l'état de complétion.

```
class CourseProgress(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    course = models.ForeignKey(Course, on_delete=models.CASCADE)
    completed_modules = models.ManyToManyField(Module, blank=True)
    completion_percentage = models.IntegerField(default=0)
    last_accessed = models.DateTimeField(auto_now=True)
    is_completed = models.BooleanField(default=False)
    completed_at = models.DateTimeField(null=True, blank=True)
    class Meta:
        unique_together = ('user', 'course')
    def update_progress(self):
        total_modules = self.course.modules.count()
        completed = self.completed_modules.count()
        self.completion_percentage = int((completed / total_modules) * 100) if
total_modules else 0
        self.is_completed = (self.completion_percentage == 100)
        if self.is_completed and not self.completed_at:
            self.completed_at = timezone.now()
        self.save()
```

La classe **CourseProgress** permet de suivre l'avancement d'un utilisateur dans un cours. Elle enregistre les modules terminés à l'aide d'un champ ManyToManyField vers **Module**, calcule le pourcentage de progression via le champ completion_percentage, et garde une trace de la dernière consultation grâce à last_accessed. Le champ is_completed indique si le cours est terminé, et completed_at enregistre la date de complétion. La méthode update_progress() met à jour ces données en calculant le pourcentage à partir des modules complétés par rapport au nombre total de modules du cours. La contrainte unique_together empêche l'enregistrement de plusieurs suivis pour le même utilisateur et le même cours.

3.4 Les diagrammes de séquence:

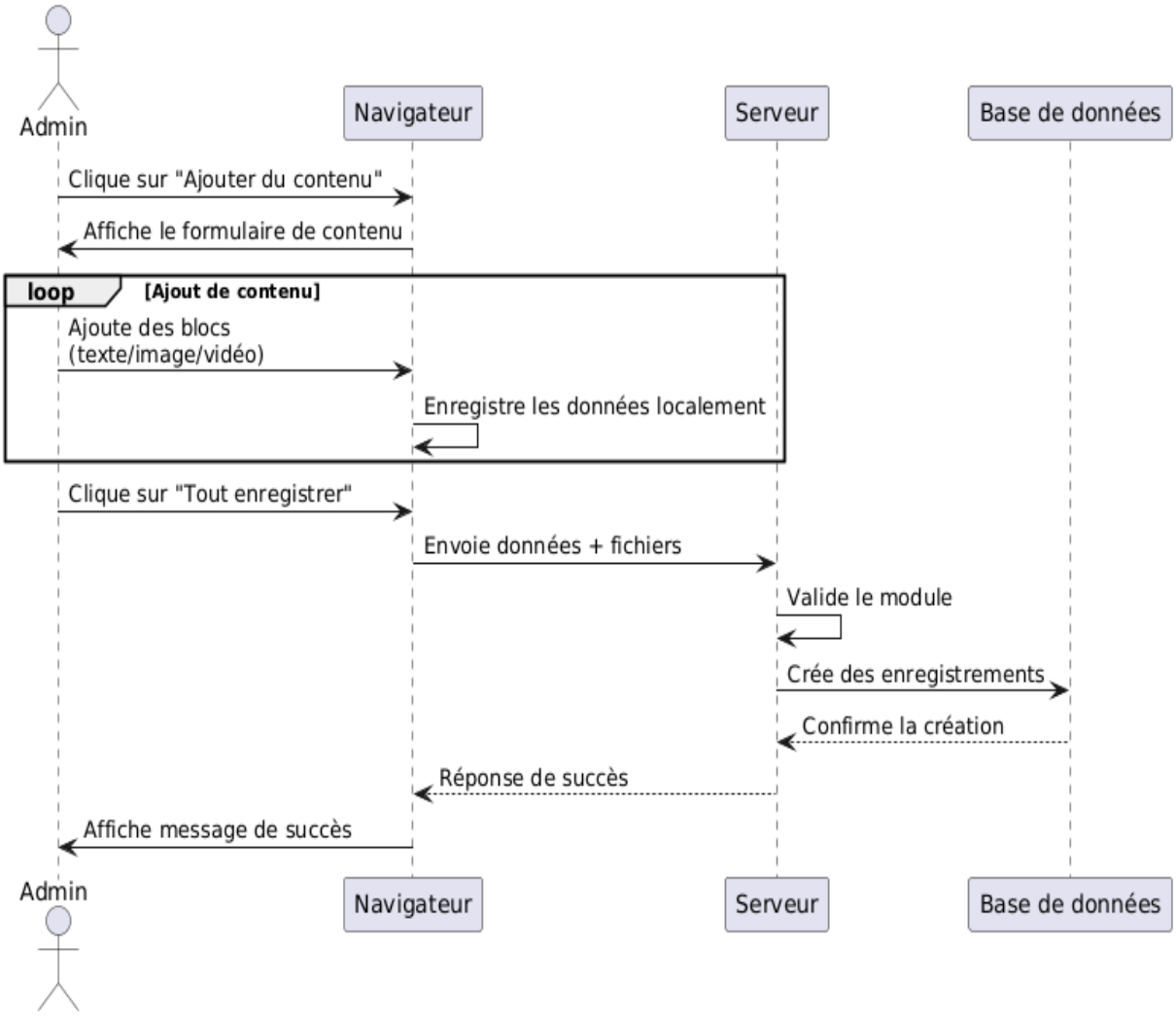

Figure 12 Diagramme de séquence pour création du contenu

Ce diagramme de séquence représente le processus d'ajout de contenu à un module sur une plateforme d'apprentissage. L'admin interagit d'abord avec l'interface via son navigateur en cliquant sur un bouton "Ajouter du contenu". Le navigateur lui affiche alors un formulaire qui permet l'ajout de différents types de blocs (texte, image, vidéo). L'utilisateur peut ajouter plusieurs blocs, qui sont temporairement enregistrés localement dans le navigateur. Une fois tous les contenus saisis, l'admin clique sur "Tout enregistrer". Le navigateur envoie alors l'ensemble des données, y compris les fichiers, au serveur. Le serveur valide les informations reçues, puis crée les enregistrements nécessaires dans la base de données. Une fois la création réussie, la base de données renvoie une confirmation au serveur, qui répond ensuite au navigateur avec un message de succès. Enfin, le navigateur affiche un retour visuel à l'admin confirmant que le contenu a été correctement enregistré.

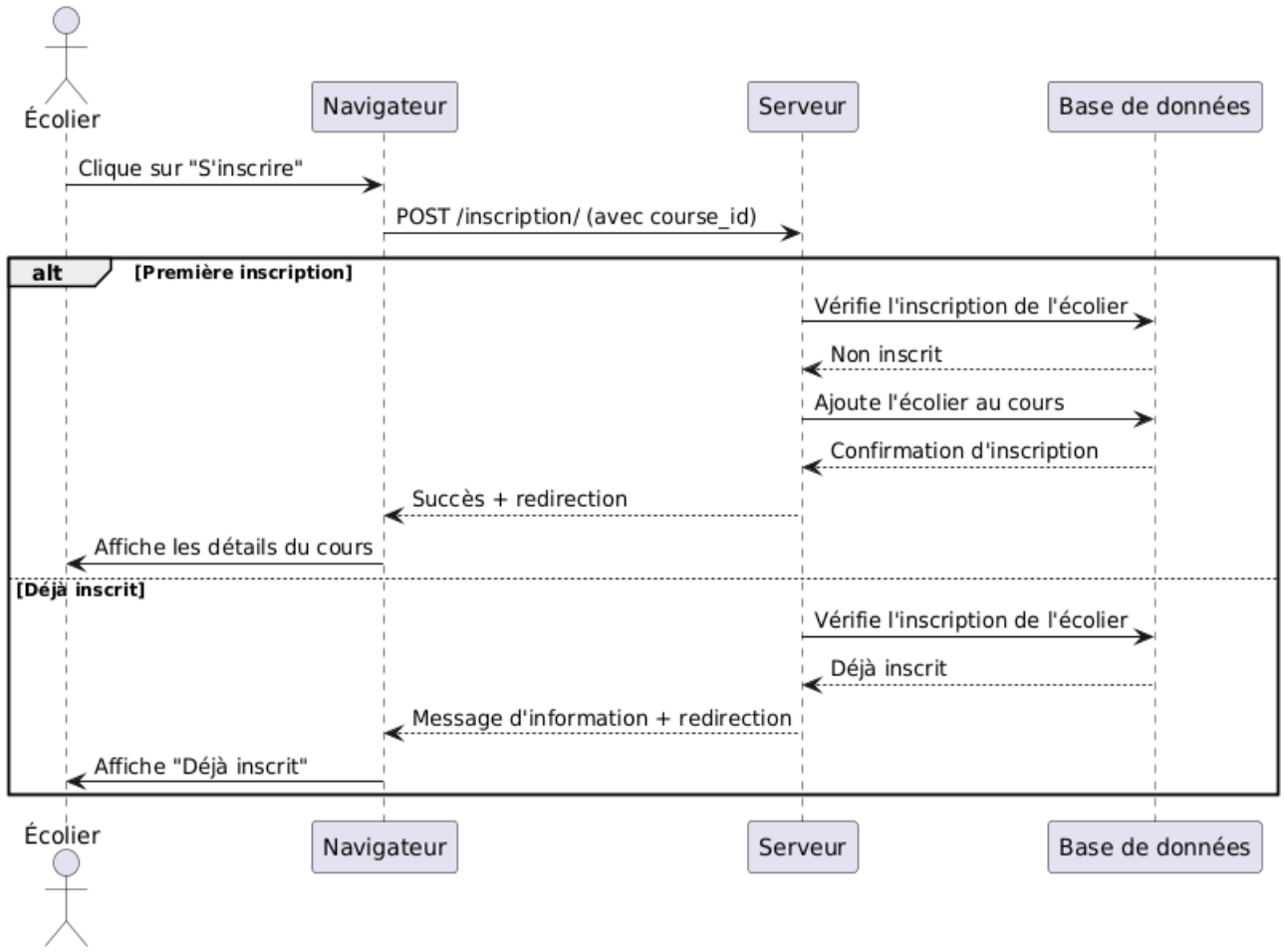

Figure 13 Diagramme de classe pour enregistrement dans un cours

Ce diagramme de séquence illustre le processus d'inscription d'un écolier à un cours via une interface web. Lorsque l'écolier clique sur le bouton **"S'inscrire"**, le navigateur envoie une requête POST au serveur contenant l'identifiant du cours sélectionné. Le serveur interroge la base de données pour vérifier si l'écolier est déjà inscrit à ce cours.
Deux scénarios sont alors possibles :

- **Première inscription** : la base de données indique que l'écolier n'est pas encore inscrit. Le serveur ajoute donc l'écolier au cours, enregistre cette information, puis envoie une confirmation de réussite au navigateur. Celui-ci redirige ensuite l'écolier vers la page de détails du cours.

- **Déjà inscrit** : la base de données indique que l'écolier est déjà inscrit. Le serveur envoie alors un message d'information au navigateur, qui affiche à l'écolier un message indiquant qu'il est déjà inscrit.

Ce diagramme met en évidence la logique conditionnelle de vérification et les échanges entre le navigateur, le serveur et la base de données.

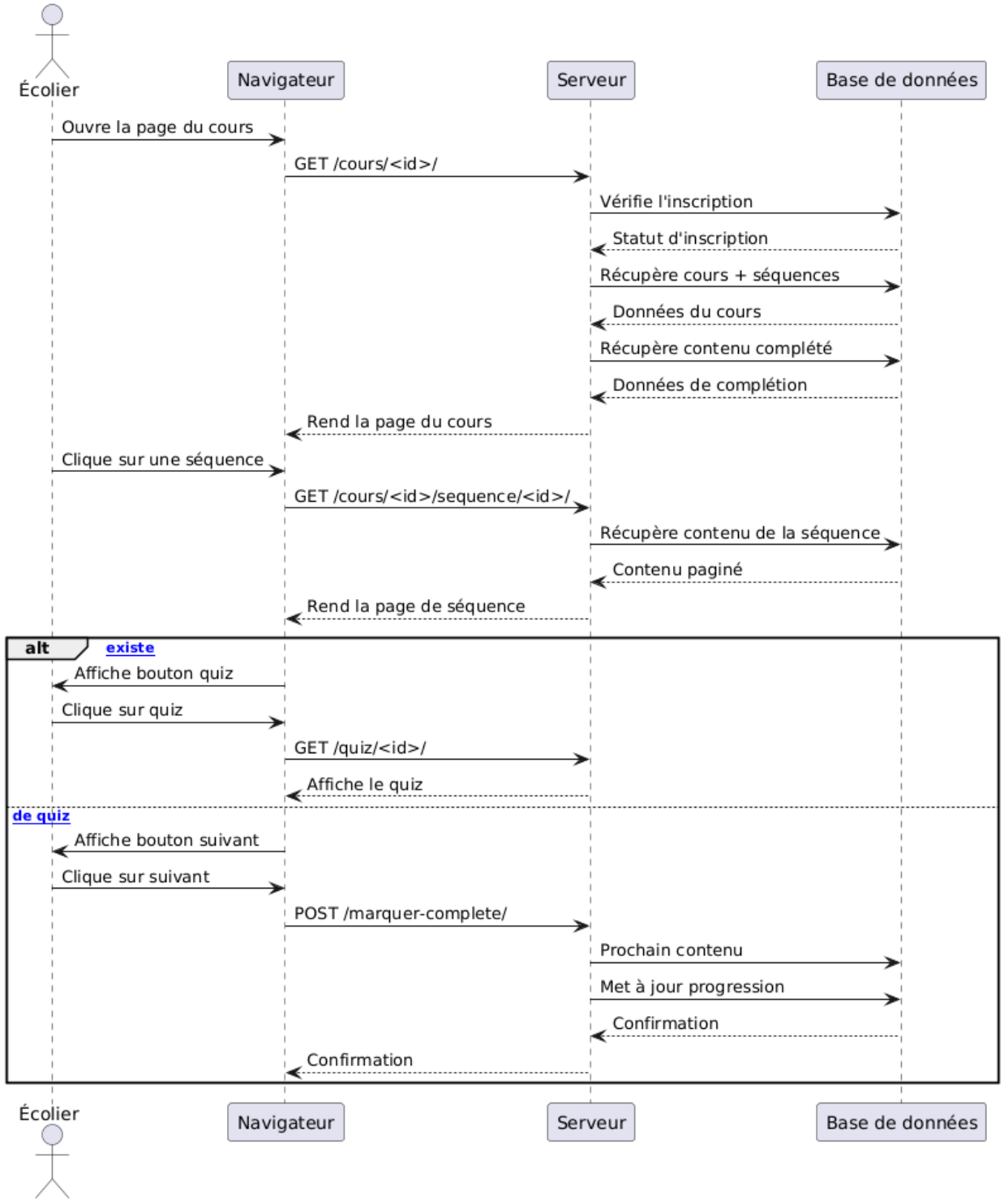

Figure 14 diagramme de séquence pour afficher le contenu d'une leçon

Ce diagramme de séquence décrit le parcours d'un écolier accédant à un cours et naviguant à travers ses séquences, avec ou sans quiz associé.

Lorsque l’écolier ouvre la page du cours. Le navigateur envoie une requête GET au serveur, qui vérifie dans la base de données si l’écolier est bien inscrit. Si c’est le cas, le serveur récupère ensuite les informations du cours, les séquences associées, ainsi que les contenus déjà complétés, avant de rendre la page du cours.

Ensuite, l’écolier clique sur une séquence pour la consulter. Le navigateur envoie une nouvelle requête GET au serveur pour obtenir le contenu de la séquence sélectionnée. Le serveur récupère les données depuis la base (sous forme paginée) et les affiche dans le navigateur.

Deux cas sont alors possibles :

- Un quiz est associé à la séquence : le navigateur affiche un bouton permettant à l’écolier de lancer le quiz. Lorsqu’il clique, une requête est envoyée au serveur pour charger le quiz correspondant.
- Aucun quiz n’est associé : un bouton “Suivant” est affiché. En cliquant dessus, une requête POST est envoyée pour marquer le contenu comme complété. Le serveur met alors à jour la progression de l’écolier dans la base de données, puis retourne une confirmation.

Ce diagramme met en valeur le fonctionnement dynamique de la plateforme d’apprentissage, qui adapte les actions disponibles selon la présence ou non de quiz, tout en assurant un suivi de la progression de l’écolier.

3.5 Conclusion :

Ce chapitre se concentre sur une introduction sélective aux diagrammes UML de base, tels que les diagrammes de cas d'utilisation critiques, les diagrammes de classes et les diagrammes de séquences, qui couvrent uniquement les fonctions essentielles du système. Il explore également les classes du modèle Django et les relations entre les champs des tables à l'aide du code ORM. L'accent est mis sur la présentation des fonctions système essentielles à une représentation efficace de la conception.

Chapitre 4

Implémentation

Dans ce chapitre, nous allons explorer en détail notre application Django et sa structure, la configuration de ses URL, ses paramètres globaux et certaines vues associées à chaque application. Nous présenterons également le résultat de nos efforts via une démonstration de l'interface utilisateur.

4.1 Structure du projet:

La structure d'un projet Django consiste en des sections comme les templates qui contiennent des fichiers HTML pour l'interface utilisateur, des fichiers statiques CSS, JavaScript et des fichiers multimédia. Les vues (views) déterminent la logique de traitement des requêtes utilisateur et génèrent les pages HTML à l'aide des modèles. Ces derniers définissent la structure et les relations de la base de données. On va voir quelques captures représentant la structure de notre projet :

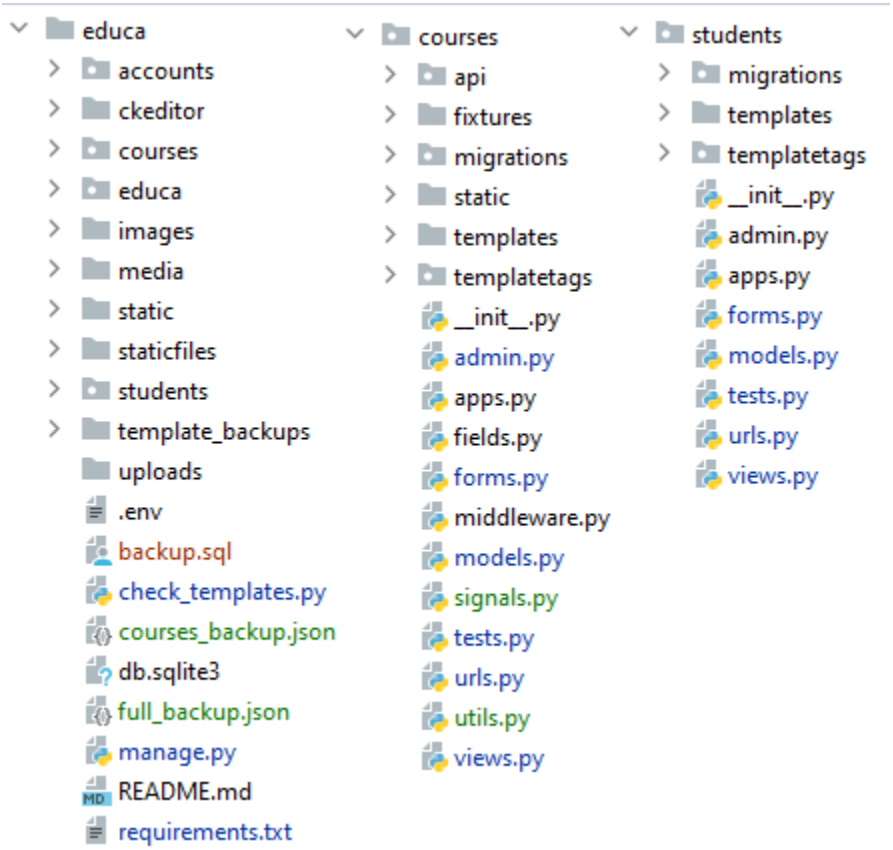

Figure 15 Structure du projet(educa), l'app courses et students

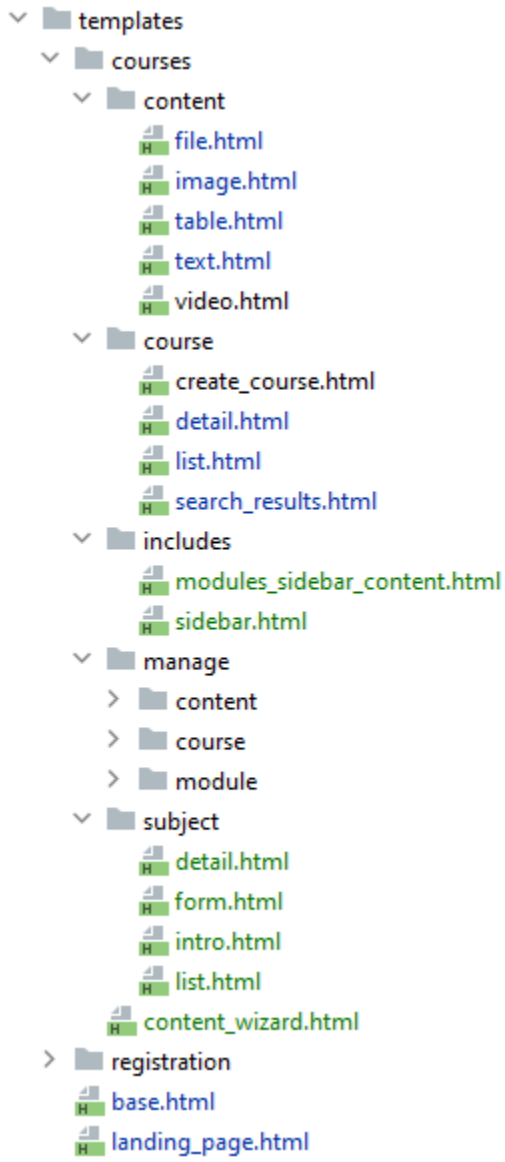

Figure 16 Les templates de l'app Courses

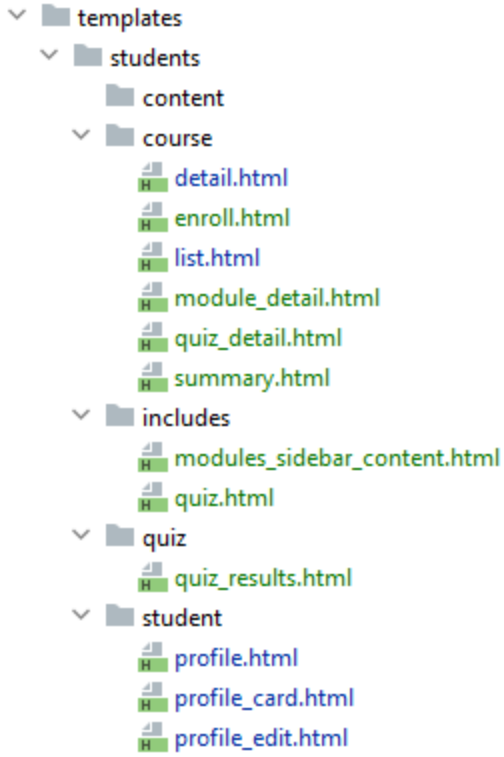

Figure 17 Les templates de l'app Students

4.2 Paramètres globales:

Les paramètres globaux d'un projet Django sont stockés dans le fichier « settings.py » et incluent les configurations des bases de données, des intergiciels, des applications installées et d'autres paramètres à l'échelle du projet. Voici quelques paramètres globaux de notre projet:

```
INSTALLED_APPS = [
    #  Apps natives de Django
    'django.contrib.admin',              # Interface d'administration Django
    'django.contrib.auth',               # Système d'authentification et gestion
des utilisateurs
    'django.contrib.contenttypes',       # Gestion des types de contenus génériques
    'django.contrib.sessions',           # Gestion des sessions utilisateur
    'django.contrib.messages',           # Système de messages temporaires
(notifications)
    'django.contrib.staticfiles',        # Gestion des fichiers statiques (CSS, JS,
images)
    #  Apps personnalisées du projet
    'accounts.apps.AccountsConfig',      # App pour gérer les comptes utilisateurs
(profils, rôles, etc.)
    'courses.apps.CoursesConfig',        # App principale pour les cours, modules
et contenus
```

```
    'students',                             # Partie du site dédiée aux élèves
    #  Apps tierces (externes)
    'embed_video',                          # Pour intégrer des vidéos (YouTube,
Vimeo) dans le contenu
    'ckeditor',                             # Éditeur de texte riche (WYSIWYG)
    'ckeditor_uploader',                    # Ajoute la possibilité d'uploader des
fichiers avec CKEditor
    'redisboard',                           # Tableau de bord de surveillance pour
Redis
    'rest_framework',                       # Framework pour construire des APIs REST
avec Django
    'pylint',                               # (Optionnel) Analyseur de code Python –
pas forcément utile ici comme app Django
    'django_extensions',                   # Fournit des extensions utiles : shell
amélioré, graphiques de modèles, etc.
]
```

URLs globales :

```
urlpatterns = [
# Page d'accueil
    path('', LandingPageView.as_view(template_name='landing_page.html'),
name='home'),
# URL pour CKEditor (gestion de l'upload des fichiers via l'éditeur)
    path('ckeditor/',
         include('ckeditor_uploader.urls'
    )),
# Authentification
    path(
        'login/',
        CustomLoginView.as_view(), name='login'),
    path('logout/', CustomLogoutView.as_view(), name='logout'),
    path('logged_out/', LoggedOutView.as_view(), name='log_out'),
    path(
        'admin/', admin.site.urls
    ),
    path('teachers/', include('teachers.urls')),
    path('courses/', include('courses.urls')),
    path('students/', include('students.urls')),
    path('search/', CourseSearchView.as_view(), name='course_search'),
# Ajout des fichiers médias en mode développement
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

URLs de l'application courses :

```
urlpatterns = [
    path('login/', views.CustomLoginView.as_view(), name='login'),
    path('logout/', views.CustomLogoutView.as_view(), name='logout'),
    path('logged_out/', views.LoggedOutView.as_view(), name='log_out'),
    path('register/', views.UserRegistrationView.as_view(), name='register'),
    path('courses/', views.CourseListView.as_view(), name='course_list', ),
    path('mine/', views.ManageCourseListView.as_view(), name='manage_course_list',
),
    path('create/', views.CourseCreateView.as_view(), name='course_create', ),
    path('<pk>/edit/', views.CourseUpdateView.as_view(), name='course_edit', ),
    path('<pk>/delete/', views.CourseDeleteView.as_view(), name='course_delete', ),
    path('<pk>/module/', views.CourseModuleUpdateView.as_view(),
name='course_module_update', ),
    path('module/<int:module_id>/content/create/select/',
views.ContentTypeSelectView.as_view(), name='module_content_select'),
    path('module/<int:module_id>/content/create/',
views.ContentCreateView.as_view(), name='module_content_create'),
    path('quiz/create/<int:content_id>/', views.QuizCreateView.as_view(),
name='quiz_create'),
    path('quiz/<int:pk>/edit/', views.QuizUpdateView.as_view(), name='quiz_edit'),
    path('subjects/add/', views.SubjectCreateView.as_view(),
name='subject_create'),
    path('module/<int:module_id>/content/<str:model_name>/<id>/',
views.ContentUpdateView.as_view(), name='module_content_update',),
    path('content/<int:id>/delete/', views.ContentDeleteView.as_view(),
name='module_content_delete', ),
    path('module/<int:module_id>/', views.ModuleContentListView.as_view(),
name='module_content_list', ),
    path('subjects/<slug:slug>/edit/', views.SubjectIntroUpdateView.as_view(),
name='subject_intro_edit'),
    path('content/order/', views.ContentOrderView.as_view(), name='content_order',
),
    path('subject/<slug:slug>/intro/', views.SubjectIntroView.as_view(),
name='subject_intro'),
    path('subject/<slug:subject>/', views.CourseListView.as_view(),
name='course_list_subject', ),
    path('subjects/', views.SubjectListView.as_view(), name='subject_list'),
    path('<slug:slug>/', views.CourseDetailView.as_view(), name='course_detail', ),
    path('subjects/<slug:slug>/', views.SubjectDetailView.as_view(),
name='subject_detail'),
]
```

4.3 Les vues :

Maintenant, on va voir les différentes vues (views) dans notre projet:

```
class SubjectListView(ListView):
    model = Subject
    context_object_name = 'subjects'
    template_name = 'courses/subject/list.html'
    def get_queryset(self):
        return
Subject.objects.select_related('year').annotate(total_courses=Count('courses'))
class SubjectIntroView(DetailView):
    model = Subject
    template_name = 'courses/subject/intro.html'
    context_object_name = 'subject'
    def get_queryset(self):
        return Subject.objects.select_related('year').annotate(
            total_courses=Count('courses')
        )
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['courses'] = Course.objects.filter(
            subject=self.object,
            published=True
        ).order_by('-published_date')
        return context
```

SubjectListView : Cette classe affiche une liste paginée de toutes les matières (Subject). Elle optimise les requêtes avec select_related('year') pour éviter les problèmes de performance (N+1) et ajoute une annotation total_courses qui compte le nombre de cours associés à chaque matière.

SubjectIntroView Affiche une vue détaillée d'une matière spécifique (Subject), incluant ses cours publiés (published=True), triés par date de publication décroissante. Comme SubjectListView, elle utilise select_related et annotate pour des requêtes optimisées.

```
class StudentCourseDetailView(LoginRequiredMixin, DetailView):
    model = Course
    template_name = 'students/course/detail.html'
    def get_queryset(self):
        return super().get_queryset().filter(students__in=[self.request.user])
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        subject = self.get_object()
        course = self.get_object()
        context['subject'] = course.subject
```

```
        context['now'] = datetime.now()
        completed_contents = set(
            CompletedContent.objects.filter(
                user=self.request.user,
                content__module__course=course
            ).values_list('content_id', flat=True)
        )
        context['completed_contents'] = completed_contents
        module = course.modules.get(
            id=self.kwargs['module_id']) if 'module_id' in self.kwargs else
course.modules.first()
        context['module'] = module
        next_module = None
        if module:
            try:
                next_module =
course.modules.filter(order__gt=module.order).order_by('order').first()
            except (AttributeError, ValueError):
                pass
        context['next_module'] = next_module
        if module:
            context['progress'] = module.user_progress(self.request.user)
        contents = module.contents.order_by('order')
        paginator = Paginator(contents, 1)
        page_number = self.request.GET.get('page', 1)
        page_obj = paginator.get_page(page_number)
        context['page_obj'] = page_obj
        all_courses =
list(course.subject.courses.filter(published=True).order_by('id'))
        current_index = all_courses.index(course)
        next_course = all_courses[current_index + 1] if current_index + 1 <
len(all_courses) else None
        context['next_course'] = next_course
        quiz = None
        for content in page_obj.object_list:
            try:
                quiz = content.quiz
                break
            except ObjectDoesNotExist:
                continue
        if quiz:
            context['quiz'] = quiz
        return context
```

La classe **StudentCourseDetailView** permet aux écoliers de consulter un cours auquel ils sont inscrits. Elle enrichit le contexte avec des données essentielles telles que la progression de l'écolier (contenus complétés, pourcentage d'avancement par module), une navigation structurée (modules et cours suivants), et une pagination des contenus (affichés un par un). Elle gère également les quiz associés aux contenus. Optimisée pour éviter les requêtes redondantes et sécuriser l'accès, cette vue offre une expérience utilisateur fluide et interactive, tout en respectant les bonnes pratiques de Django.

```
class QuizDetailView(View):
    def get(self, request, pk):
        quiz, module, questions = self._get_quiz_data(pk)
        form = QuizSubmissionForm(questions=questions)
        self._prefill_form_with_user_answers(form, questions, request.user)
        questions_with_inputs = self._build_questions_with_inputs(questions)
        return render(request, 'students/course/quiz_detail.html', {
            'quiz': quiz,
            'form': form,
            'module': module,
            'questions': questions,
            'questions_with_inputs': questions_with_inputs,
        })
```

Le rôle de la méthode get est d'afficher le quiz avec ses questions et pré-remplit les réponses déjà données par l'utilisateur. Fonctionnalités clés :

- Récupère le quiz, son module et ses questions via _get_quiz_data.
- Prépare un formulaire dynamique (QuizSubmissionForm) adapté aux types de questions (choix multiples, texte, etc.).
- Prépopule les réponses existantes de l'utilisateur (via _prefill_form_with_user_answers).
- Génère des rendus HTML interactifs pour les questions complexes (menus déroulants, glisser-déposer) avec _build_questions_with_inputs.

```
    def post(self, request, pk):
        quiz, module, questions = self._get_quiz_data(pk)
        form = QuizSubmissionForm(request.POST, questions=questions)
        questions_with_inputs = self._build_questions_with_inputs(questions)
        if not form.is_valid():
            return self._render_invalid_form(quiz, module, form,
questions_with_inputs)
        score, question_results = self._process_answers(form, questions,
request.user)
        CompletedContent.objects.get_or_create(
            user=request.user,
```

```
            content=quiz.content,
            defaults={'completed': True}
        )
        QuizResult.objects.update_or_create(
            user=request.user,
            quiz=quiz,
            defaults={'score': score, 'total': questions.count()}
        )
        next_url = self._get_next_url(module, quiz)
        return render(request, 'students/quiz/quiz_results.html', {
            'quiz': quiz,
            'form': form,
            'question_results': question_results,
            'score': score,
            'total': questions.count(),
            'next_url': next_url,
            'questions': questions,
            'questions_with_inputs': questions_with_inputs,
        })
    def _render_invalid_form(self, quiz, module, form, questions_with_inputs):
        return render(self.request, 'students/course/quiz_detail.html', {
            'quiz': quiz,
            'form': form,
            'module': module,
            'questions_with_inputs': questions_with_inputs,
        })
    def _get_quiz_data(self, pk):
        quiz = get_object_or_404(Quiz, pk=pk)
        module = quiz.content.module
        questions = quiz.questions.prefetch_related(
            'choices').all()
        return quiz, module, questions
    def _prefill_form_with_user_answers(self, form, questions, user):
        for question in questions:
            user_answers = UserAnswer.objects.filter(user=user,
question=question)
            if not user_answers.exists():
                continue
            base_field_name = f"question_{question.id}"
            if question.type in ['text', 'fill_blank']:
                form.fields[base_field_name].initial =
user_answers.first().text_answer
            elif question.type == 'choice':
                choice = user_answers.first().choice
                if choice:
                    form.fields[base_field_name].initial = choice.id
    def _process_answers(self, form, questions, user):
        score = 0
```

```
        results = []
        for question in questions:
            base_field_name = f"question_{question.id}"
               UserAnswer.objects.filter(user=user, question=question).delete()
            if question.type == 'fill_blank':
                answer = form.cleaned_data.get(base_field_name, "").strip()
                correct = question.fill_blank.correct_answer.strip().lower() ==
answer.lower()
                if correct:
                    score += 1
                UserAnswer.objects.create(user=user, question=question,
text_answer=answer)
                results.append({'question': question, 'user_answer': answer,
'is_correct': correct})
            elif question.type == 'choice':
                choice_id = form.cleaned_data.get(base_field_name)
                choice = get_object_or_404(Choice, id=choice_id)
                UserAnswer.objects.create(user=user, question=question,
choice=choice)
                if choice.is_correct:
                    score += 1
                results.append({'question': question, 'user_answer': choice,
'is_correct': choice.is_correct})
        return score, results
    def _get_next_url(self, module, quiz):
        contents = list(module.contents.order_by('order'))
        quiz_content = quiz.content
        try:
            index = contents.index(quiz_content)
        except ValueError:
            index = None
        if index is not None and (index + 1) < len(contents):
            return f"{module.get_absolute_url()}?page={index + 2}"
        next_module = Module.objects.filter(course=module.course,
order__gt=module.order).order_by('order').first()
        return next_module.get_absolute_url() if next_module else
module.course.get_absolute_url()
```

- Valide les réponses avec **QuizSubmissionForm** :

```
class QuizSubmissionForm(forms.Form):
    def __init__(self, *args, questions=None, **kwargs):
        super().__init__(*args, **kwargs)
        self.questions = questions or []
        for question in self.questions:
            self._add_question_field(question)
    def _add_question_field(self, question):
        base_field_name = f"question_{question.id}"
        if question.type == 'choice':
```

```
            choices = [(choice.id, choice.text) for choice in
question.choices.all()]
            self.fields[base_field_name] = forms.ChoiceField(
                choices=choices,
                widget=forms.RadioSelect,
                label=question.text,
                required=True
            )
        elif question.type == 'text':
            self.fields[base_field_name] = forms.CharField(
                widget=forms.Textarea,
                label=question.text,
                required=False
            )
        elif question.type == 'fill_blank':
            self.fields[base_field_name] = forms.CharField(
                label=question.text,
                required=True
            )
```

- Si invalide, réaffiche le formulaire avec erreurs (_render_invalid_form).
- Si valide :
  - Calcule le score et enregistre les réponses (_process_answers).
  - Marque le contenu comme terminé (CompletedContent).
  - Sauvegarde le score global (QuizResult).
  - Redirige vers la page de résultats ou le contenu suivant (_get_next_url).
  - Template de succès : students/quiz/quiz_results.html

4.4 L’interface du site web :

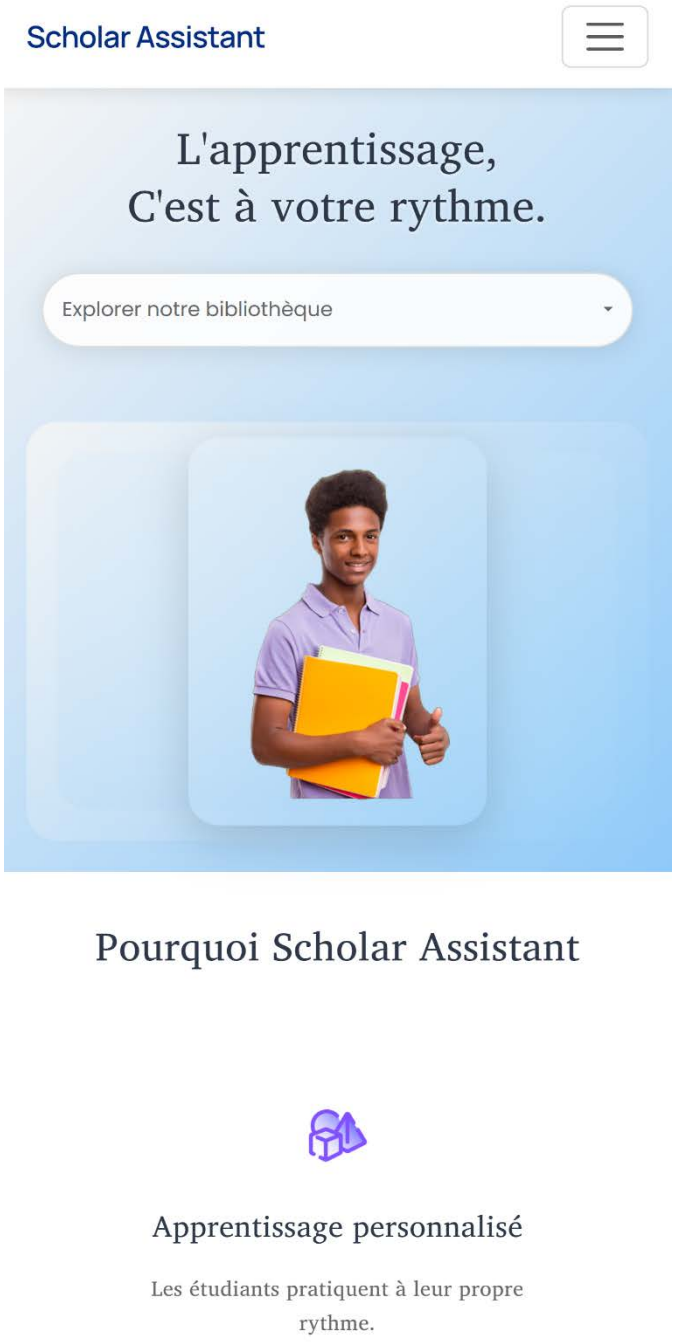

Figure 18 La page d'accueil mobile

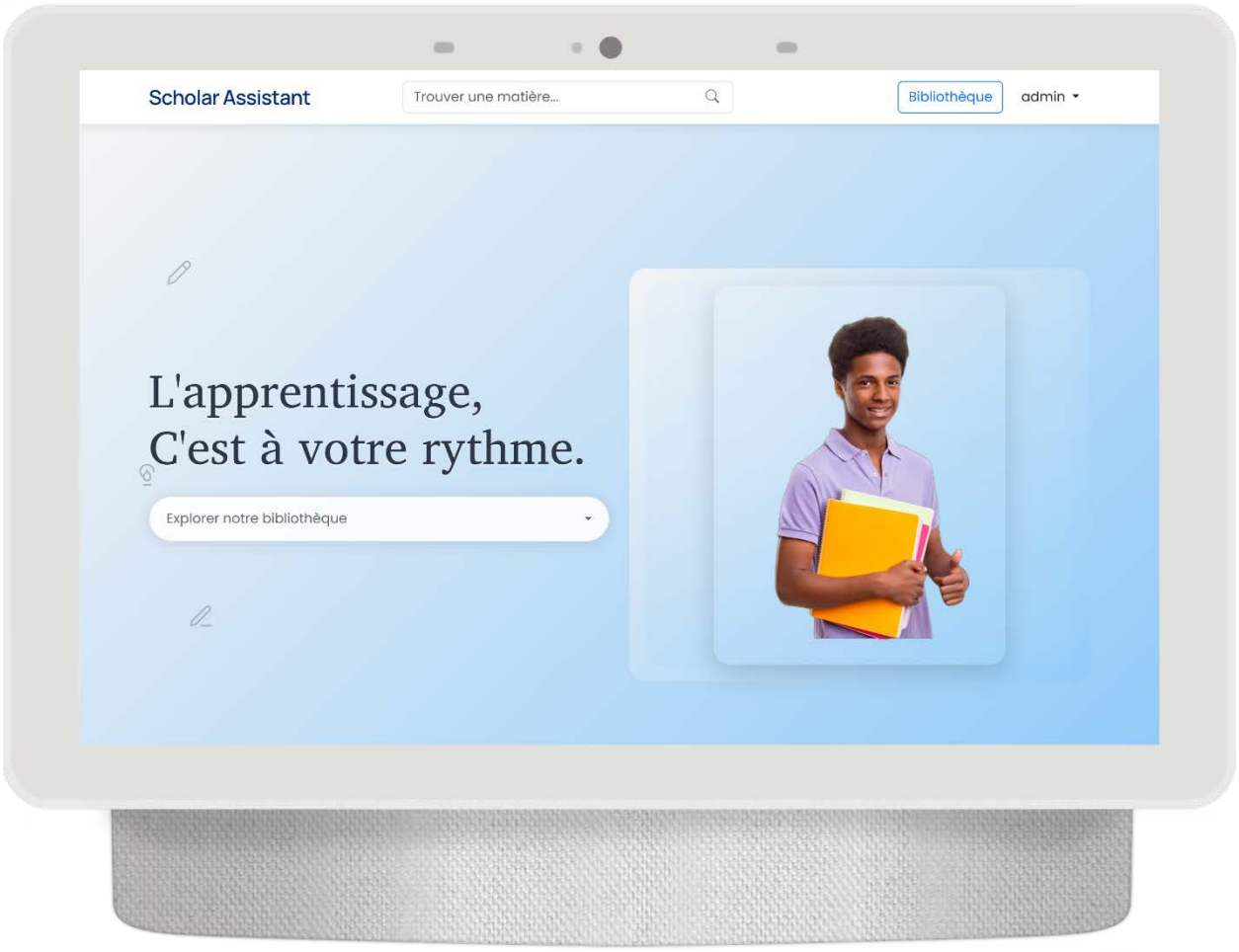

Figure 19 La page d'accueil large

Figure 20 La liste des matières sur mobile

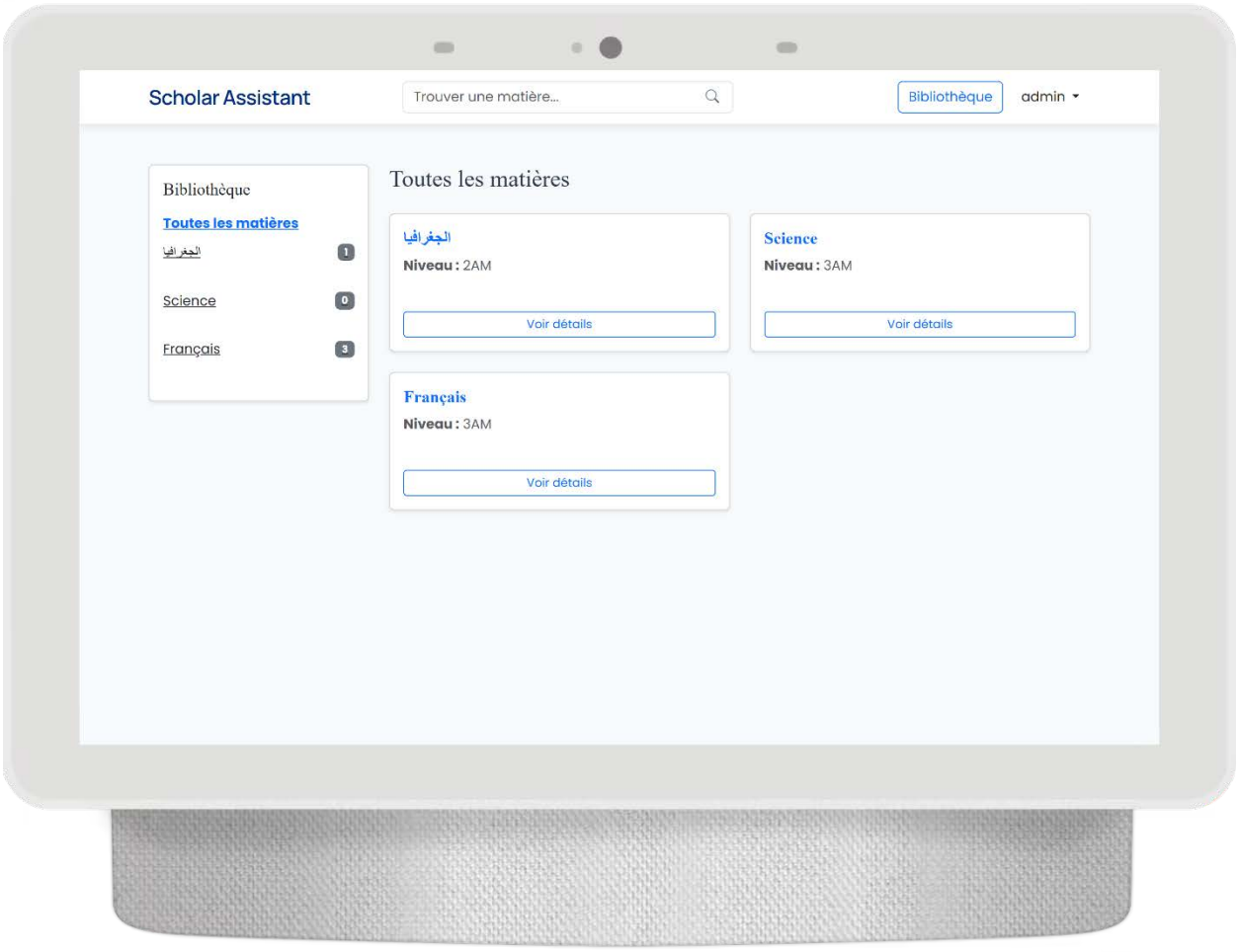

Figure 21 La liste des matières

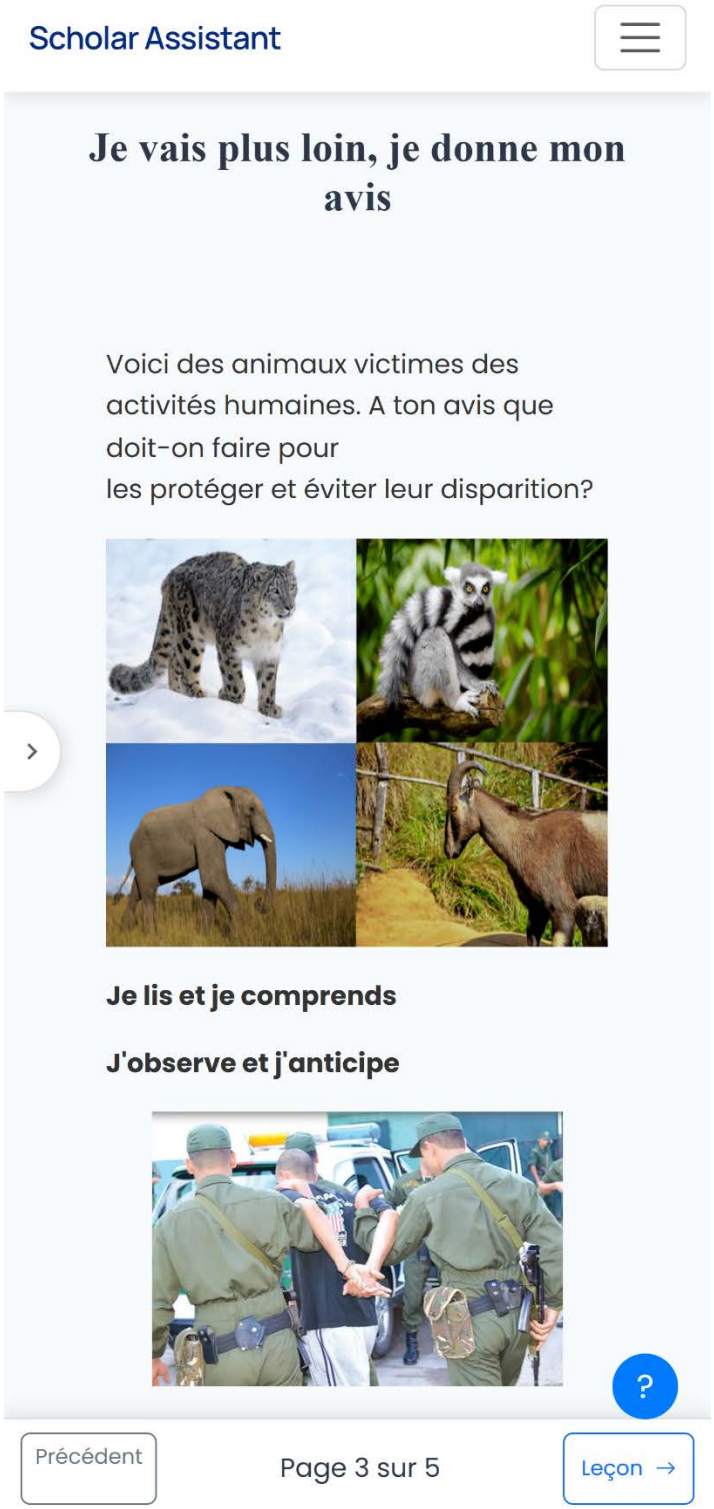

Figure 22 Le contenu d'une leçon sur mobile

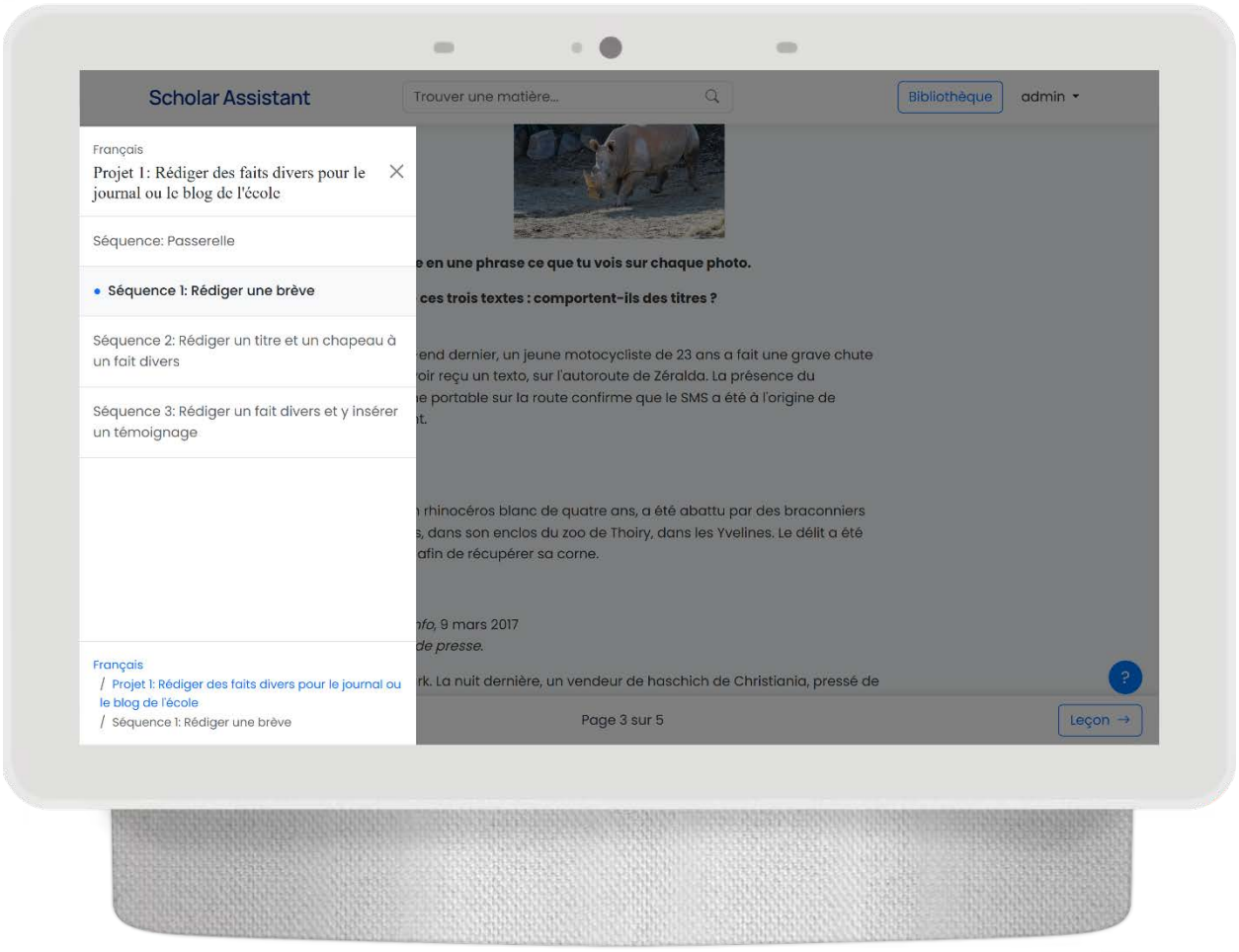

Figure 23 Le contenu d'une leçon avec la liste des leçons disponibles

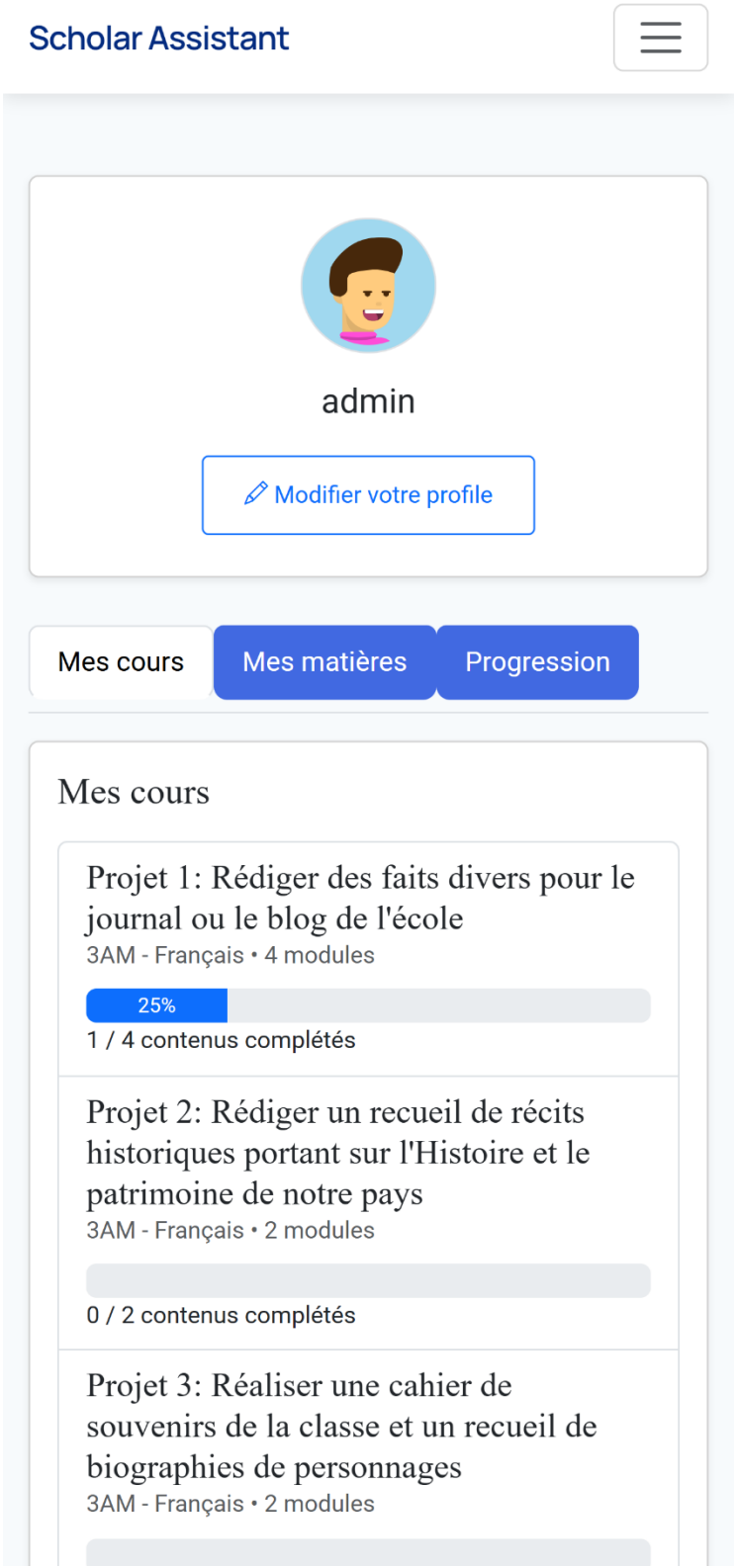

Figure 24 La page profile et progrès

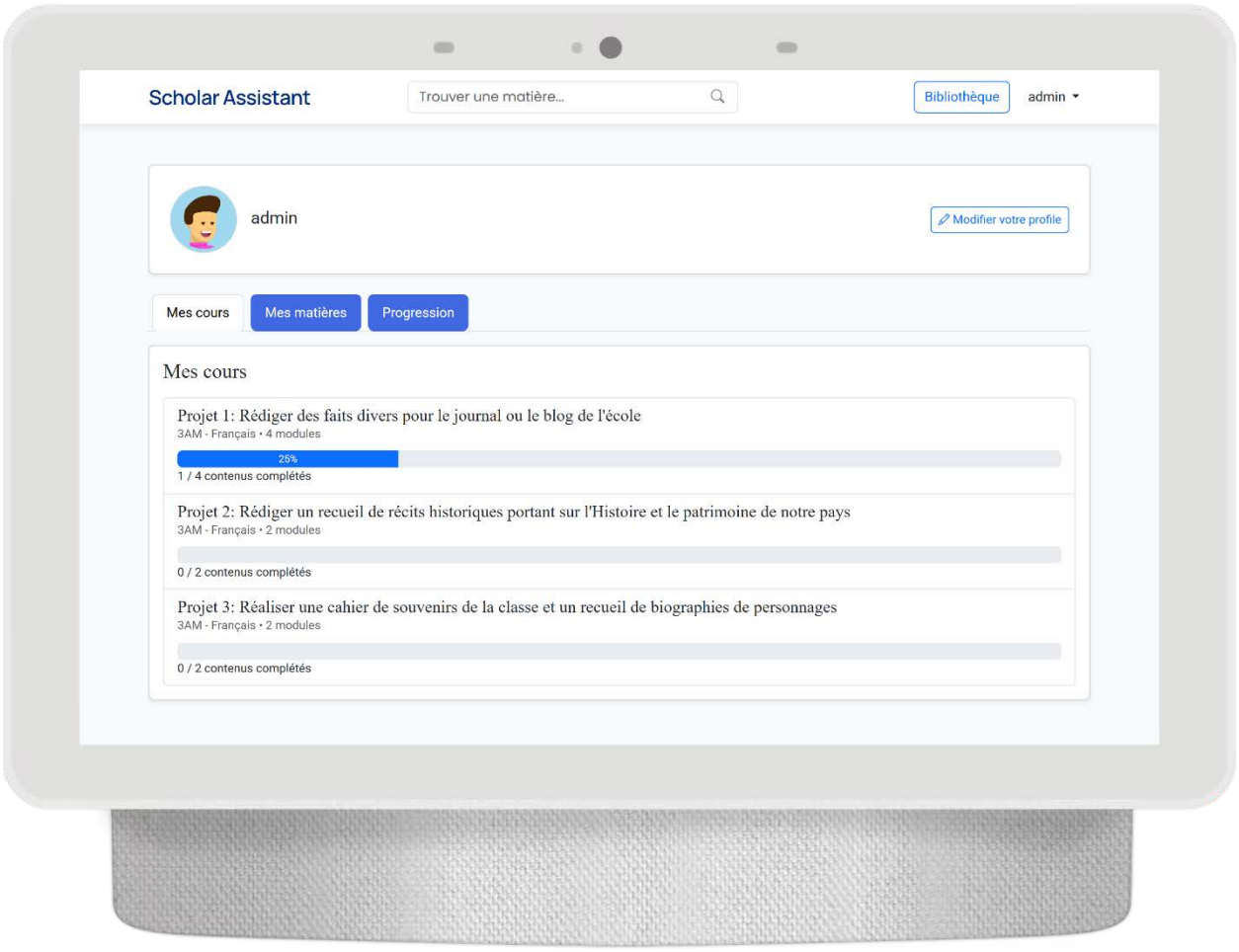

Figure 25 La page profile et progrès

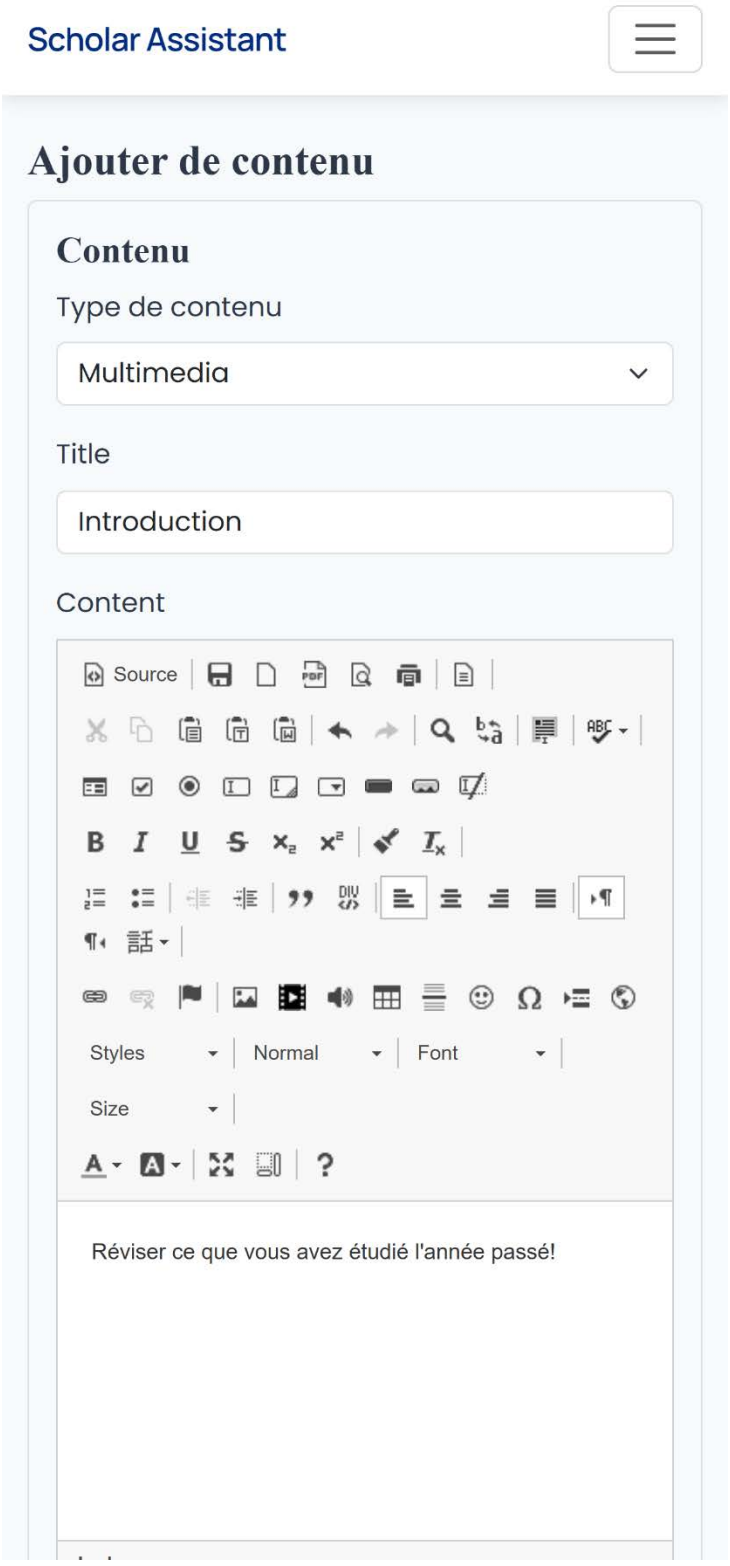

Figure 26 L'ajout du contenu (mobile)

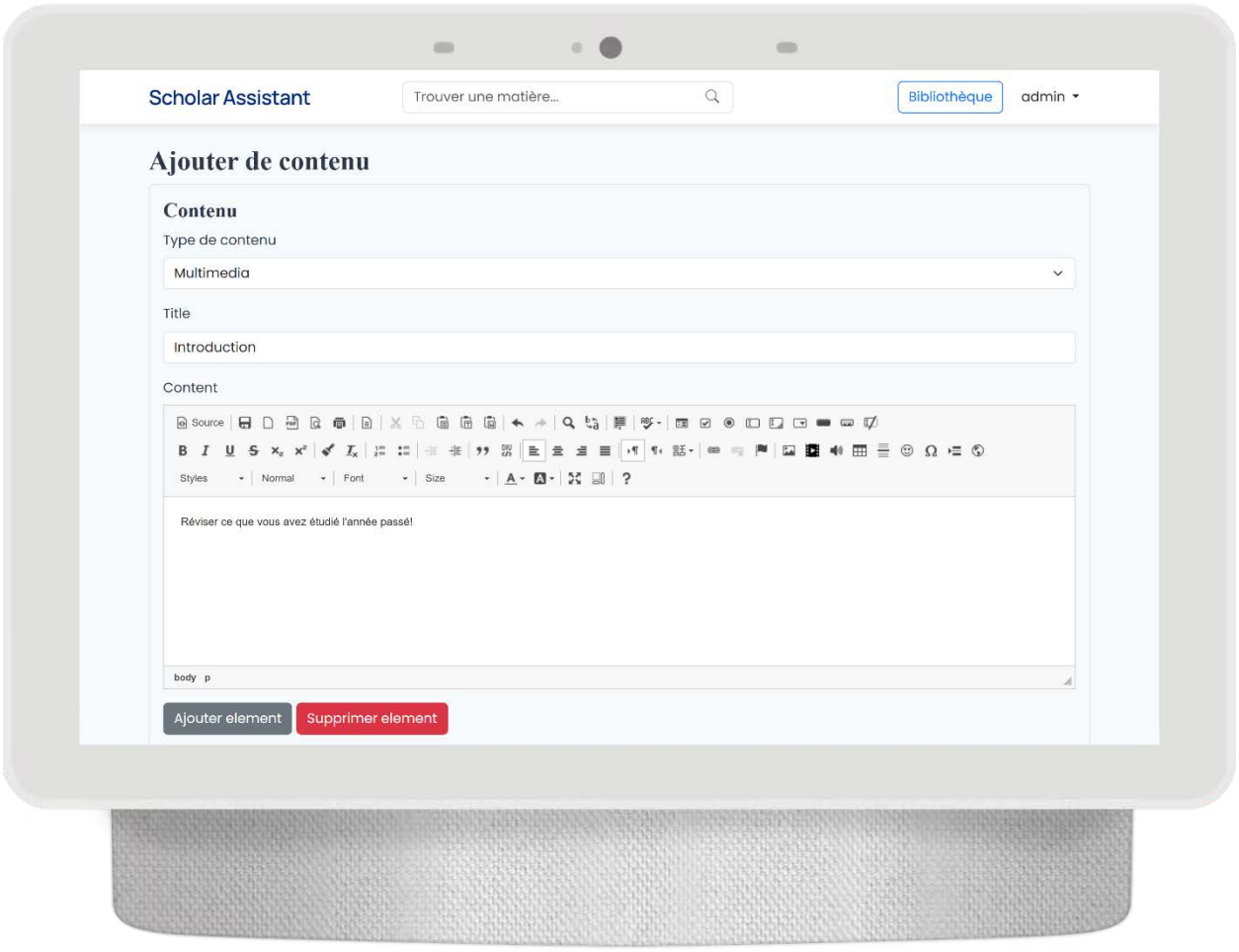

Figure 27 L'ajout du contenu

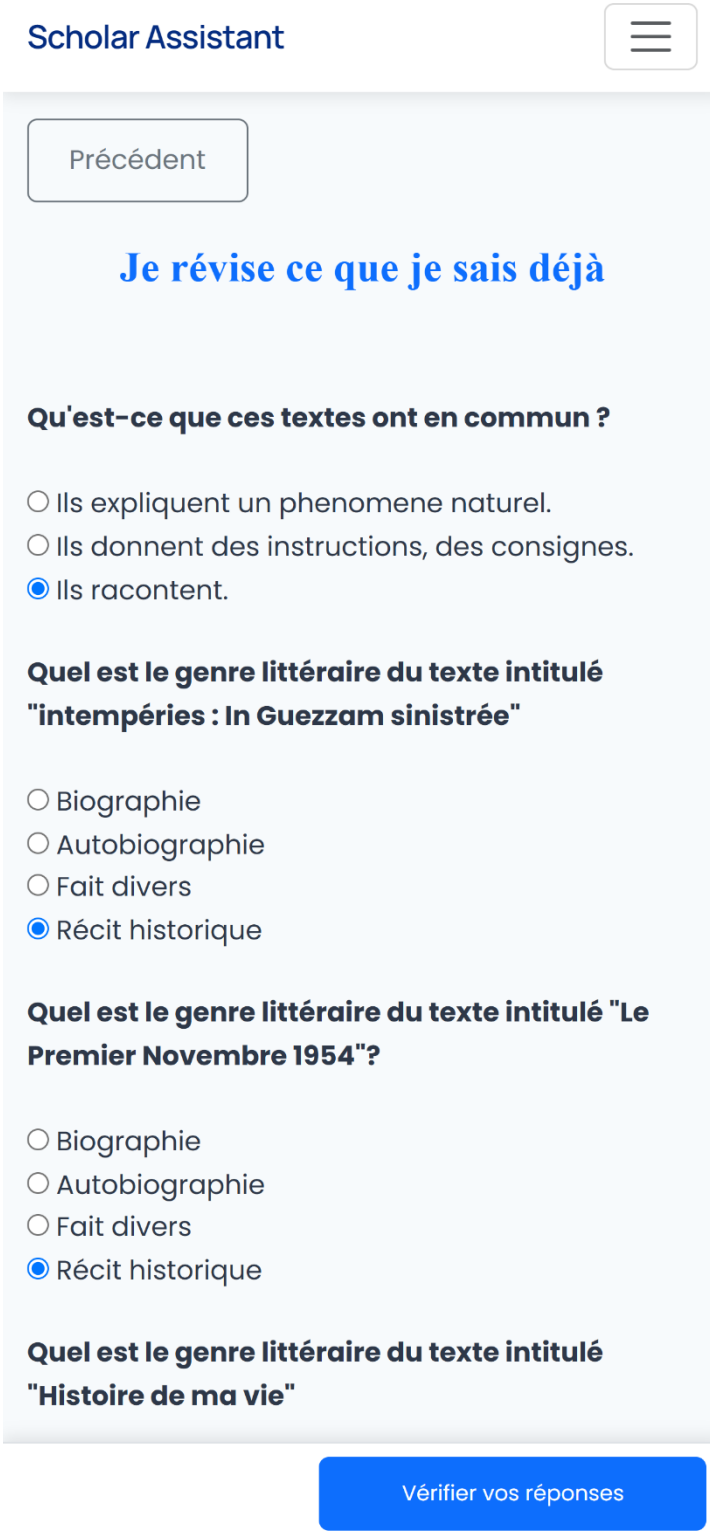

Figure 28 Page du quiz

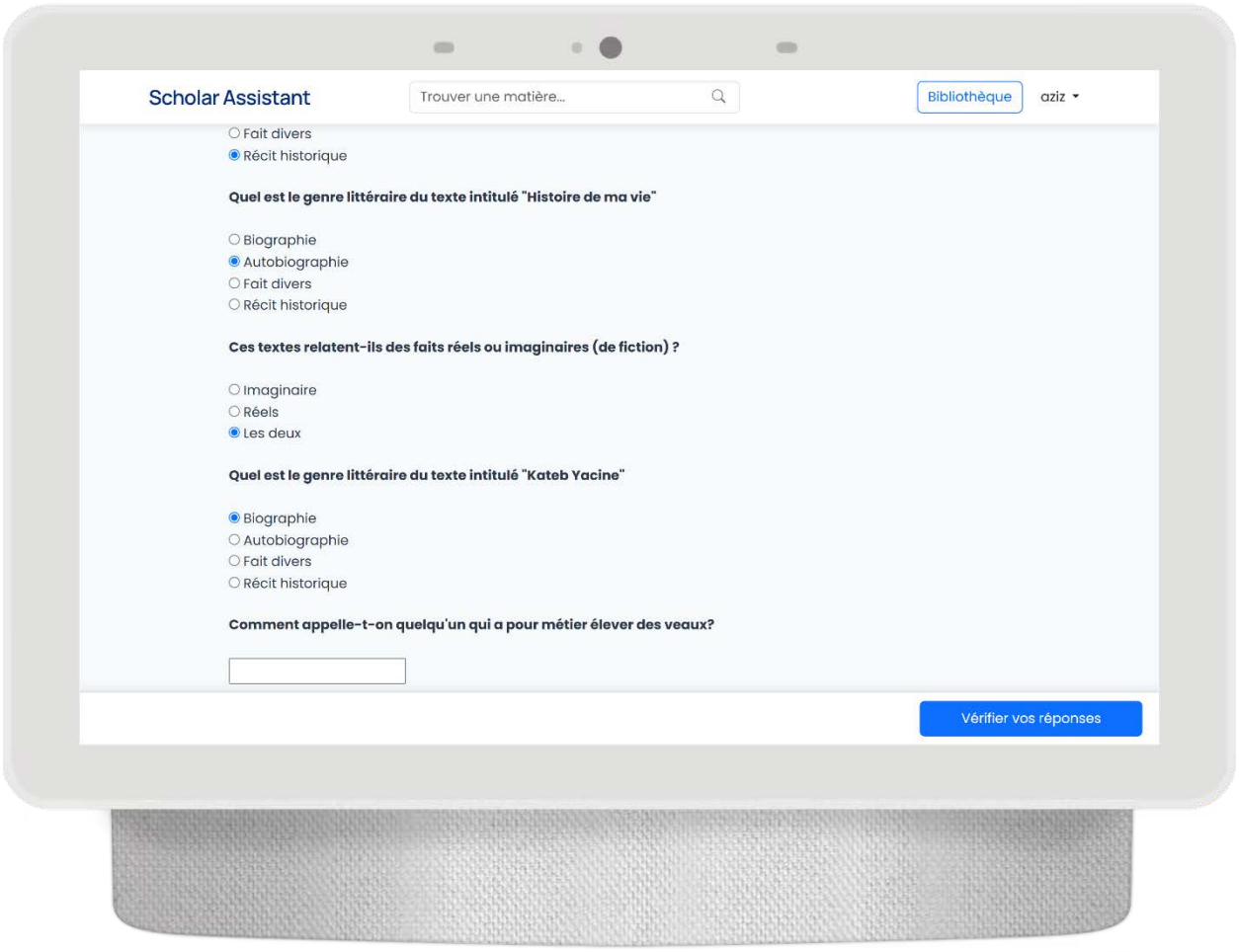

Figure 29 Page du quiz

4.5 Conclusion

Ce chapitre traite des détails de mise en œuvre du projet, incluant la structure du projet, les paramètres globaux, les URLs et diverses interfaces. Il couvre des sujets tels que les interfaces, les pages écoliers. Dans l'ensemble, le chapitre fournit un aperçu complet des aspects de mise en œuvre du projet, en se concentrant sur la façon dont les différentes interfaces sont conçues et utilisées.

Conclusion

Conclusion :

Ce mémoire s'est donné pour ambition de développer une plateforme d'apprentissage assisté, pensée pour accompagner durablement les écoliers tout au long de leur parcours scolaire. En proposant un accès direct à des contenus pédagogiques organisés et enrichis, cette plateforme vise à pallier la dépendance croissante aux cours particuliers. Elle offre, en parallèle, un système d'évaluation intelligent fondé sur des quiz personnalisés pour chaque matière, favorisant ainsi une progression autonome et ciblée de l'élève.

La convivialité de l'interface constitue un pilier fondamental de cette solution. Elle permet une navigation fluide : l'inscription, la connexion et le choix des cours adaptés au niveau scolaire s'effectuent avec une grande simplicité.

L'intégration d'outils technologiques performants a, en outre, permis l'enrichissement des contenus grâce au multimédia, tout en assurant une gestion administrative centralisée et efficace. Ce travail a ainsi abouti à la mise en place d'un environnement d'apprentissage interactif, stimulant et résolument tourné vers les exigences de l'éducation moderne.

Face au "recours massif" aux cours particuliers en Algérie, phénomène qualifié de "fléau" par le ministre de l'Éducation nationale, ce mémoire avait pour objectif de concevoir et développer une alternative numérique accessible et équitable : une plateforme d'apprentissage assisté gratuite.

Les travaux menés ont permis de répondre à cette problématique en développant une solution complète qui s'attaque aux causes profondes du recours aux cours particuliers. La plateforme conçue offre un accès direct et gratuit à des contenus pédagogiques organisés, éliminant ainsi les barrières économiques qui excluent de nombreux élèves issus de milieux modestes. Le système d'évaluation intelligent basé sur des quiz personnalisés favorise une progression autonome et ciblée, répondant directement aux lacunes identifiées dans l'enseignement traditionnel.

Cette réalisation démontre qu'il est possible de proposer une alternative concrète au système de cours particuliers, contribuant ainsi à réduire les inégalités éducatives et à redonner à l'école publique son rôle central dans la transmission du savoir.

Cette réalisation constitue également une base solide pour l'évolution future de la plateforme vers des technologies plus avancées, notamment l'intelligence artificielle, qui pourrait démultiplier l'impact pédagogique de cette solution numérique.

À l'avenir, il serait pertinent d'étudier l'impact de telles plateformes sur les performances scolaires réelles, d'explorer leur intégration dans les politiques éducatives nationales, et d'évaluer les opportunités d'enrichissement technologique pour maximiser leur efficacité pédagogique.

LISTE DE REFERENCES :

[1] https://www.algerie360.com/les-cours-particuliers-en-algerie-le-ministre-de-leducation-met-en-lumiere-un-defi-majeur/

[2] *Singapour : avant-gardiste de l'apprentissage hybride* https://www.learnthings.fr/top-10-des-pays-qui-ont-bouleverse-leducation-en-2024/

[3] https://www.custup.com/cms-content-management-system-definition/

[4] https://www.teachizy.fr/definition/lms/

[5] https://hospitalityinsights.ehl.edu/fr/avantages-apprentissage-en-ligne

[6] https://think-modular.com/fr/elearning_plattformen

[7] https://savoirslibres.fr/outils-et-technologies-pour-leducation/la-plateforme-dapprentissage-un-outil-essentiel-pour-leducation-moderne

[8] M. Antonio, Django 4 By Example, Livery Place: Packt Publishing Ltd, 2022.

[9] Hasnaoui, R. (2020, 28 mai). *Exemple de conclusion de mémoire*. Scribbr. https://www.scribbr.fr/memoire/exemple-conclusion-memoire/