

Knowledge Learning – Plateforme e-learning de formation en ligne

Présentation du projet

À l'ère du numérique, la formation en ligne s'impose comme un pilier essentiel de l'apprentissage moderne. Qu'il s'agisse de développer de nouvelles compétences, de se reconverter ou de compléter un cursus existant, de plus en plus d'apprenants se tournent vers des solutions d'e-learning accessibles à distance, flexibles et adaptées à leurs contraintes. Cette évolution est encore renforcée par les mutations du marché du travail, la généralisation du télétravail, et la nécessité d'une mise à jour constante des compétences dans de nombreux domaines.

Malgré l'abondance de plateformes d'apprentissage existantes, de nombreux freins subsistent : interfaces peu ergonomiques, surcharge d'informations, manque de lisibilité dans la progression, coûts d'abonnement élevés, ou encore absence de fonctionnalités réellement centrées sur l'utilisateur. En parallèle, les petites structures (formateurs indépendants, associations, centres de formation locaux) peinent à trouver des solutions simples et abordables pour diffuser leurs contenus sans devoir investir dans des outils complexes ou des développements sur mesure.

C'est dans ce contexte qu'est né le projet **Knowledge Learning**, une application web de formation en ligne simple, moderne et sécurisée, conçue pour permettre à des utilisateurs non techniques d'acheter des leçons ou des cursus complets, les suivre, et obtenir une certification.

Les différents objectifs de notre plateforme sont de rendre cette dernière responsive et simple d'accès pour les utilisateurs inscrits, quel que soit l'outil de connexion.

Les utilisateurs auront la possibilité d'acheter des leçons à l'unité ou bien opter pour l'achat d'un cursus complet. Il sera intégré un système de suivi de progression pour les utilisateurs avec la validation des leçons et ou des cursus, ces derniers donnant lieu automatiquement à l'obtention d'une certification.

Par ailleurs, une interface administrateur est mise en place uniquement accessible si l'utilisateur connecté a le rôle admin. Cette interface permet de consulter, créer, mettre à jour ou encore supprimer les leçons, les cursus et les utilisateurs ainsi que supprimer les achats effectués par les utilisateurs dont le rôle est client.

Afin de sécuriser ces données par rapport aux paiements, et de faciliter les paiements, il a été intégré l'outil Stripe.

Le projet d'une plateforme e-learning s'adresse principalement à trois types de clients :

- Des particuliers, divisibles en trois tiers en fonction de l'âge :

- Un premier tiers composé majoritairement d'étudiants qui perçoivent dans la plateforme e-learning une nouvelle manière d'apprendre en autonomie,
 - Un deuxième tiers qui recherchera dans l'e-learning l'épanouissement de leur vie personnelle et,
 - Un dernier tiers de seniors (retraités), de plus en plus connectés, s'ouvrant à de nouvelles expériences ne nécessitant pas un déplacement.
- Des professionnels en quête de spécialisation dans un domaine ou plus radicalement entrant dans un processus de reconversion ; l'apprentissage à distance leur permet de poursuivre leur activité professionnelle et d'être en autonomie sur leur apprentissage.
 - Des organismes de formation souhaitant proposer des cours via un outil clé en main à leurs collaborateurs qui permet de suivre la validation de chacun tout en proposant le même support pour tous.

Le projet *Knowledge Learning* a vu le jour pour répondre à deux attentes complémentaires. D'une part, celle des utilisateurs souhaitant accéder à des formations en ligne facilement, sans contraintes techniques, dans un environnement clair et efficace. D'autre part, celle des administrateurs ou formateurs, qui ont besoin d'un outil simple pour publier des contenus, gérer les parcours de formation et suivre la progression des apprenants.

Afin de structurer le développement de l'application, les besoins ont été analysés sous deux angles : les **besoins fonctionnels** liés aux attentes des utilisateurs, et les **besoins techniques**, qui concernent la qualité, la fiabilité et la pérennité de l'outil.

Besoins fonctionnels

Du côté des clients, plusieurs fonctionnalités se sont révélées essentielles afin de garantir une plateforme de qualité. Premièrement il est primordial de permettre la possibilité de créer un compte et de se connecter de manière sécurisée, afin de pouvoir effectuer les achats de formations souhaités.

La navigation au sein de la plateforme devra être simple, intuitive et épurée, les clients devront atteindre la formation souhaitée le plus rapidement possible. Le nombre de “click” doit être limité.

Le système d’achat dans la continuité du désir de rapidité et de clarté devra être simple d'utilisation, rapide car nos clients souhaitent se former au plus vite, fonctionnel afin de toujours garantir un service. Ces points participent à la confiance que la clientèle donne pour effectuer ses achats sur la plateforme.

Afin de permettre aux différents utilisateurs qui ont acheté sur la plateforme un suivi de progression qui récompense, Il doit être mis en place une possibilité de valider chaque cursus de ce qui a été appris; ainsi que des leçons achetées individuellement. Par conséquent, le système de récompense s'articulera autour d'une certification qui sera remise automatiquement à la validation d'un cursus ou à celle de l'ensemble des leçons (appartenant au même cursus) qui auront été achetées individuellement.

Pour les administrateurs, l'application doit proposer un espace de gestion du contenu pédagogique c'est-à-dire Les thèmes, les cursus et les leçons, en donnant à l'administrateur, pour ces derniers, la possibilité de les consulter, de les modifier, voire de les supprimer, ou d'en ajouter de nouveaux si besoin.

De plus, ce même espace de gestion doit permettre à l'administrateur de gérer les différents utilisateurs de la plateforme et de leur mettre le rôle admin si besoin. L'administrateur sera ainsi en capacité de modifier ou supprimer n'importe quel utilisateur. Il sera aussi possible par l'administrateur de visualiser ou supprimer les achats des différents utilisateurs.

L'interface devra être minimaliste, claire et répartie en cinq parties :

- les thèmes
- les cursus
- les leçons.

sachant qu'une leçon créée devra appartenir à un cursus appartenant lui-même à un thème.

- les utilisateurs listé avec leur rôle admin ou clients et,
- l'intégralité des achats effectués sur la plateforme avec le nom du client acheteur.

La plateforme devra satisfaire un certain nombre d'exigences. Elle devra, notamment, être accessible à tout moment afin de permettre la flexibilité d'apprentissage au client, proposer une interface responsive qui doit être adaptée à

tout type d'écran aussi bien détaillé d'écran horizontal que vertical afin que les usagers puissent valider leur formation à n'importe quel endroit et depuis n'importe quel outil : Ordinateur, tablette ou smartphone.

La plateforme doit garantir la sécurité des utilisateurs, qu'il s'agisse des informations personnelles ou bien de l'historique des achats ainsi que du moyen de paiement. Dans un souci de sécurité, l'authentification devra être solide et les rôles entre utilisateurs clients et utilisateur administrateur nettement séparés.

L'architecture de l'interface doit être modulaire afin de permettre les évolutions nécessaires dans le futur comme l'ajout de nouveaux thèmes de nouveaux cursus ou de nouvelles leçons qui devront facilement s'ajouter visuellement dans l'interface au sein des éléments déjà existants.

Afin de garantir le bon fonctionnement de toutes les possibilités d'achat de la plateforme ainsi que des créations d'utilisateur une série de tests automatiques devrait être implémentée.

Cette analyse a permis de formaliser un cahier des charges structuré, qui a ensuite servi de base à la conception technique et fonctionnelle de l'ensemble du projet.

Analyse fonctionnelle

Pour concevoir une application web réellement adaptée aux attentes des utilisateurs et des administrateurs, une phase d'analyse fonctionnelle a été menée. Cette étape a permis de traduire les besoins exprimés en fonctionnalités concrètes à travers des **cas d'usage** et des **user stories**, afin de mieux organiser le développement et anticiper les interactions avec le système.

Les conclusions de l'analyse fonctionnelle reprennent l'ensemble des possibilités qu'un utilisateur ou un administrateur est capable de réaliser sur la plateforme.

Sur la plateforme Knowledge Learning un utilisateur ayant le rôle client sera dans la capacité de se créer un compte et de s'y connecter à partir des informations transmises lors de la création du compte. Une fois connecté à son compte, il est dans la capacité de consulter la liste des leçons et cursus répertoriés sur la plateforme dans le thème approprié. Lors de la consultation des leçons du cursus le client sera en capacité d'acheter une leçon à l'unité ou un cursus complet. Il sera en mesure de retrouver l'intégralité de ses achats leçons comme cursus dans la section mes achats qui lui sera affectée. La consultation et la lecture d'une leçon entraînera sa validation. La validation de l'intégralité des leçons d'un cursus validera ce cursus. Une fois le cursus validé l'utilisateur recevra une certification correspondante; l'ensemble des certifications de l'utilisateur client seront visibles dans la section associée.

Concernant l'utilisateur ayant le rôle administrateur, il sera le seul à pouvoir accéder à l'onglet admin qui donne accès au panel administrateur. Il sera bien évidemment en capacité d'ajouter, de modifier ou de supprimer des leçons des cursus ou des thèmes. Afin de respecter l'organisation du site imposé par la librairie Knowledge Learning lors de la création d'une leçon, il devra indiquer à quel cursus elle appartient en utilisant le cursus id. De même, lors de la création d'un cursus il est impératif de signaler le thème parents de ce cursus. Il est à noter pour les administrateurs que la suppression en cascade sera activée par conséquent lors de la suppression d'un thème l'intégralité des cursus et par conséquent l'intégralité des leçons appartenant à ces cursus seront supprimés.

Le projet **Knowledge Learning** repose sur un socle technologique moderne, robuste et adapté aux exigences d'une application web full stack. Le choix des différentes technologies s'est appuyé sur plusieurs critères essentiels : la stabilité, la popularité, la documentation, la facilité de prise en main, la compatibilité avec des outils modernes de déploiement, ainsi que la capacité à garantir une architecture polyvalente, simple et évolutive. L'ensemble de l'environnement technique a été pensé pour assurer la cohérence entre le front-end, le back-end, la base de données, la gestion de la sécurité, les tests et le déploiement.

Front-end : Vue.js et CSS

L'interface utilisateur du projet a été développée avec **Vue.js**, un framework JavaScript moderne particulièrement adapté à la création d'applications web interactives. Grâce à son système basé sur des composants réutilisables, **Vue.JS** a permis d'organiser le code de manière claire et modulaire. Sa documentation bien fournie, sa courbe d'apprentissage progressive et sa communauté active ont facilité sa prise en main et ont permis une mise en œuvre efficace des différentes vues de l'application.

L'avantage majeur de vue JS et sa capacité d'adaptation dans le cas d'un ajout de nouveaux cursus ou de nouvelles leçons dans la plateforme la page s'adapte dynamiquement sans problème.

Le design a été pensé pour être à la fois sobre, intuitif et compatible avec tous les types d'appareils. Les styles ont été réalisés à l'aide de feuilles de style personnalisées, avec une attention particulière portée à la cohérence visuelle et à la lisibilité sur mobile comme sur ordinateur. Cette approche a permis de conserver une certaine flexibilité dans le rendu, tout en garantissant une expérience utilisateur fluide et agréable.

Back-end : Node.js, Express et Sequelize

Le serveur a été développé avec **Node.js**, associé au framework **Express.js**, pour bénéficier d'une architecture non bloquante (asynchrone) et d'une excellente performance en gestion d'API REST.

Express, par sa facilité d'utilisation et sa souplesse, permet de structurer efficacement les routes, les middlewares et les contrôleurs selon une logique MVC (Modèle-Vue-Contrôleur).

Pour la gestion de la base de données, l'ORM **Sequelize** a été utilisé afin de faciliter les interactions avec la base PostgreSQL. Sequelize permet d'abstraire les requêtes SQL tout en conservant un contrôle fin sur les relations entre entités (associations, migrations, validations...).

Base de données : PostgreSQL

La persistance des données est assurée par **PostgreSQL**, un système de gestion de base de données relationnelle reconnu pour sa robustesse, sa conformité aux standards SQL et sa capacité à gérer des volumes de données importants. PostgreSQL a été choisi pour ses performances, sa sécurité, mais aussi pour sa compatibilité native avec Sequelize et render. Il permet de structurer efficacement les différentes entités du projet (utilisateurs, leçons, cursus, achats...) tout en garantissant l'intégrité des données.

Authentification et sécurité : JWT et CSRF

L'authentification repose sur un système **JWT (JSON Web Token)**, qui permet de gérer les sessions utilisateurs de manière sécurisée et sans stockage côté serveur. Chaque utilisateur reçoit un token signé, utilisé pour accéder aux routes protégées.

La protection contre les attaques de type **Cross-Site Request Forgery (CSRF)** a été mise en place côté back-end via un middleware spécifique. Ce dispositif garantit que seules les requêtes initiées depuis le site de l'utilisateur authentifié sont acceptées, renforçant ainsi la sécurité globale de l'application.

Déploiement pour test : Netlify (front-end) et Render (API)

Le projet a été découpé en deux entités indépendantes : le front-end (Vue.js) et le back-end (Node/Express).

- Le front-end est hébergé sur **Netlify**, une plateforme moderne de déploiement continue qui facilite la mise en ligne des applications statiques ou SPA (Single

Page Applications). Netlify gère automatiquement les builds, les redirections et le déploiement à chaque push sur le dépôt **GitHub**.

- L'API Express est quant à elle déployée sur **Render**, un PaaS (Platform-as-a-Service) permettant l'hébergement d'applications Node.js avec gestion automatique des dépendances, des variables d'environnement, des redémarrages automatiques en cas d'erreur ou à chaque push sur le dépôt **GitHub**, utilise aussi **PostgreSQL** et des certificats HTTPS.

Tests : Mocha et Supertest

Afin d'assurer la fiabilité du code et la validation des fonctionnalités, une campagne de tests a été mise en place via **Mocha**, un framework de test pour Node.js, et **Supertest**, une bibliothèque permettant de simuler des requêtes HTTP sur une API Express.

Les tests couvrent aussi bien les cas de succès que les cas d'erreurs (validation, authentification, autorisations), garantissant la stabilité du système et facilitant la détection de régressions lors des évolutions futures.

Outils complémentaires

Le développement a été réalisé avec **Visual Studio Code**, un éditeur de code polyvalent et extensible. Tandis que **Postman** a servi à tester manuellement les routes de l'API, tout au long du développement.

Le versionnement du projet est géré avec **Git** et **GitHub**, permettant un suivi précis de l'évolution du code source.

Pour la conception des maquettes et le prototypage de l'interface, l'outil **Figma** a été utilisé.

Beekeeper Studio a permis d'interroger la base de données et d'y extraire diverses données au format json.

Maquettes

Avant de débiter le développement de l'application Knowledge Learning, j'ai conçu une série de maquettes représentant les principales interfaces du parcours utilisateur. Cette étape m'a permis de réfléchir à l'ergonomie, à la hiérarchie de l'information, ainsi qu'à la cohérence visuelle entre les différentes pages. L'objectif était de poser les bases d'une navigation claire, intuitive et accessible, aussi bien pour les utilisateurs que pour les administrateurs. Chaque maquette a été pensée dans une logique fonctionnelle, en lien direct avec les besoins identifiés lors de l'analyse. Ces visuels ont ensuite servi de référence tout au long du développement de l'application.

Page d'accueil

La page d'accueil de la plateforme a été pensée comme un point d'entrée clair et engageant. Elle souhaite transmettre immédiatement l'objectif du site, avec un message de bienvenue centré, accompagné d'un bouton pour explorer les formations, ainsi que deux autres boutons pour l'inscription et la connexion. Un encart est réservé à une vidéo de présentation, afin de guider visuellement les nouveaux utilisateurs. La navigation principale est positionnée en haut, avec un accès rapide à l'inscription et à la connexion. Ce choix renforce l'accessibilité dès l'arrivée sur la plateforme, sans nécessiter de recherche ou de scrolling.

Page de connexion

L'interface de connexion a été pensée dans une logique de sobriété et d'accessibilité. Le formulaire est centré sur la page, dans un encart bleu clair, avec deux champs simples (email et mot de passe) et un bouton de validation clairement identifiable. Le bouton "Connexion" en vert contraste efficacement avec le fond, attirant l'œil sans nuire à la lisibilité. Ce choix permet à l'utilisateur de s'authentifier rapidement, sans distraction ni surcharge. La page est volontairement minimaliste pour que l'utilisateur ne se pose aucune question : il comprend immédiatement ce qu'on attend de lui.

Page d'inscription

L'écran d'inscription reprend les mêmes principes ergonomiques que celui de la connexion, en y ajoutant un champ supplémentaire pour le nom. L'ensemble est disposé de manière verticale avec des espacements suffisants pour une utilisation fluide sur mobile. Le bouton "S'inscrire" adopte une couleur vert claire, renforçant la perception d'action positive. La simplicité visuelle vise ici à éviter toute friction dans le processus de création de compte, en réduisant les étapes au strict minimum.

Page des thèmes de formation

Cette page permet à l'utilisateur de découvrir les différentes thématiques disponibles sur la plateforme. Chaque thème est présenté sous forme de bouton arrondi avec un fond bleu clair, facilitant le repérage visuel et le clic. Le titre "Thèmes de formation" est bien mis en évidence, et la navigation reste présente en haut pour garantir un accès constant aux autres rubriques. Le choix d'un affichage centré et épuré offre une bonne lisibilité, y compris sur tablette ou petit écran. Chaque thème sert de point d'entrée vers un ensemble de cursus spécifiques.

Page des cursus disponibles

L'objectif de cette page est d'inciter à l'achat en mettant en avant l'offre pédagogique. Chaque cursus est affiché avec son titre, son prix et un bouton rouge

“Acheter”. L’usage de cette couleur crée un contraste fort, attirant immédiatement l’attention. Le contenu est disposé de manière verticale, avec un encart par cursus, ce qui permet d’identifier rapidement les offres et de faciliter la prise de décision. Le design reste sobre, avec une structure répétitive qui crée une régularité visuelle rassurante pour l’utilisateur.

Page des leçons d’un cursus

Une fois un cursus sélectionné, l’utilisateur accède à la liste des leçons qui le composent. Chaque leçon est affichée dans un encart individuel contenant son titre, son prix et un bouton d’achat rouge. Ce système d’affichage uniforme favorise une lecture rapide et une prise de décision efficace. La structure verticale permet une navigation fluide, même sur mobile, tandis que les couleurs choisies assurent un bon contraste entre les éléments. Cette page est centrale dans le parcours d’achat, et son design vise à maximiser la clarté et la simplicité.

Page des certifications

Cette interface permet aux utilisateurs de consulter les certifications qu’ils ont obtenues. Chaque certification est présentée dans un encart contenant le nom du cursus, la date de validation et l’identité de l’utilisateur. L’affichage est clair, centré, et reprend la charte graphique de la plateforme pour une continuité visuelle. Cette page valorise le travail accompli par l’apprenant et renforce l’aspect professionnalisant du projet. Elle constitue également un bon levier de motivation dans le suivi de formation.

Dashboard utilisateur

Le tableau de bord offre un aperçu global de l’activité de l’utilisateur sur la plateforme. Il regroupe les raccourcis vers les principales sections : explorer les contenus, consulter les achats, vérifier les certifications obtenues, etc. Chaque bouton est suffisamment espacé pour garantir une bonne expérience sur tous types d’écrans. L’interface utilise des couleurs douces et des pictogrammes explicites, facilitant la compréhension et la navigation. Ce dashboard constitue le point de départ de l’expérience utilisateur après connexion.

Page achats / progression

Cette page liste les leçons et cursus que l’utilisateur a achetés. Chaque bloc contient le titre du contenu, son état d’avancement, et un bouton “Valider” pour enregistrer la progression. Le design est conçu pour séparer clairement les leçons des cursus, à l’aide d’encadrés différenciés. L’objectif ici est de permettre à l’apprenant de suivre sa progression en autonomie, tout en conservant un accès rapide à ses contenus. L’ajout d’un bouton d’action visible dans chaque bloc renforce l’aspect interactif et pédagogique.

Interface administrateur

L'espace admin est divisé en plusieurs sections : gestion des thèmes, des cursus, des leçons, des utilisateurs et des achats. Chaque module comprend des boutons pour créer, modifier ou supprimer des entrées. Le design est structuré sous forme de blocs distincts, avec un fond clair et une interface volontairement minimaliste pour éviter toute surcharge visuelle. Cette organisation permet une gestion complète de la plateforme tout en restant accessible à des profils non techniques. L'ensemble a été pensé pour maximiser la productivité et simplifier la mise à jour des contenus pédagogiques.

3. Développement

Avant d'aborder en détail les fonctionnalités développées, voici un aperçu simplifié de l'arborescence du projet. Elle met en évidence l'organisation en modules et la séparation des responsabilités selon mon architecture MVC (Modèle – Vue – Contrôleur), ainsi que l'intégration d'un front-end dans le même dépôt dans le dossier `knowledge-learning-frontend`.

Cette architecture m'a permis de développer de façon claire , en séparant les différents rôles (routes, contrôleurs, modèles, middlewares, services, tests, etc.). La logique repose sur un back-end Node.js Express connecté à une base PostgreSQL, couplé à un front-end Vue.js dans le dossier `knowledge-learning-frontend`.

Ci-dessous, l'arborescence simplifiée (certains fichiers système ou de dépendances comme `node_modules` ou `.git` sont volontairement exclus) :

```
├─ app.js
├─ config/
|   └─ db.js
├─ controllers/
|   ├─ admin/
|   ├─ auth.controller.js
|   ├─ user.controller.js
|   └─ ...
├─ docs/ (support PDF, slides, visuels)
```

```
├─ knowledge-learning-frontend/
|   ├─ index.html
|   ├─ package.json
|   └─ src/
|       ├─ App.vue
|       ├─ components/
|       ├─ pages/
|       ├─ router/
|       └─ ...
├─ middlewares/
|   ├─ auth.middleware.js
|   └─ csrf.middleware.js
├─ models/
|   ├─ user.model.js
|   └─ ...
├─ routes/
|   ├─ admin/
|   ├─ auth.routes.js
|   └─ ...
├─ services/
|   └─ mailer.service.js
├─ test/
|   └─ api.test.js
```

└─ server.js

└─ seed.js

Cette structure a été conçue pour assurer la clarté du code, faciliter la navigation entre les fichiers et rendre le projet facilement maintenable par un autre développeur.

Étapes de développement

Authentification

L'authentification des utilisateurs constitue l'une des premières fonctionnalités mises en place pour le site e-commerce *Knowledge Learning*. Elle permet de sécuriser les accès aux contenus de formations payants et à l'espace personnel de l'utilisateur connecté, tout en différenciant les droits selon le rôle (client ou administrateur).

J'ai opté pour une authentification avec un JWT (**JSON Web Token**), une solution stateless, largement utilisée par les sites web modernes. Lorsqu'un utilisateur se connecte avec des identifiants valides, un token signé est généré côté serveur et envoyé au client. Les informations contenues par ce token sont essentielles comme l'identifiant de l'utilisateur et son rôle. Le token est donc requis pour accéder aux routes protégées.

Voici les grandes étapes mises en œuvre :

Premièrement il a été mis en place les routes `/auth/register` et `/auth/login` dans le respect de l'architecture MVC. Les routes mises dans le fichier `routes/auth.routes.js` de la logique métier du contrôleur `controllers/auth.controller.js`.

Le mot de passe recueilli par la route `/auth/register` et validé s'il correspond à tous les critères de sécurité mis en place comme la longueur, la présence d'un chiffre et d'une majuscule... Il sera hashé par la fonction de dérivation `bcrypt` importée à notre projet.

Lors de l'utilisation de la route `/auth/login` les identifiants sont comparés avec ceux en base de données. A la validation par notre contrôleur le json web token est généré et transmis par cookie HTTP-only.

Le fichier `Middlewares/ auth.checkJWT` permettra de vérifier et décoder le token à chaque requête privée qui fera appel à un `checkJWT`

Dans le but de toujours renforcer la sécurité, le token est stocké côté client sous forme de **cookie HTTP-only**, ce qui empêche son accès via JavaScript (XSS). J'ai

également intégré une protection contre les attaques **CSRF** en générant un token CSRF secondaire, stocké dans un cookie sécurisé et vérifié via un middleware Express situé dans le fichier `middlewares/csrf.middleware`.

Ce système permet une navigation sécurisée entre les pages privées (dashboard, achats, certifications...) avec une protection d'accessibilité par le token JWT. Le rôle de l'utilisateur limite la possibilité d'accès au dashboard admin au rôle administrateur seulement. La protection contre les attaques csrf est aussi effective comme cité plus haut.

L'authentification fonctionne de manière fluide et sécurisée, aussi bien côté front que back. Une fois connecté, l'utilisateur peut accéder à son tableau de bord, acheter des contenus, et suivre sa progression. Si le token est absent ou invalide, l'accès est refusé et une redirection vers la page de connexion est envoyée.

Cette fonctionnalité, bien que complexe à mettre en place au départ, constitue le cœur de la sécurité applicative. Elle m'a également permis de mieux comprendre la gestion des sessions, ainsi que les enjeux liés à la sécurisation des échanges.

Gestion des utilisateurs

Une fois le système d'authentification développé, j'ai commencé la gestion des utilisateurs, un élément central dans la logique de l'application. Elle permet non seulement de distinguer les différents profils (utilisateur client et administrateur), mais aussi de gérer les droits d'accès, les données personnelles et les interactions avec les autres modules (achats, validations, certifications...).

Chaque utilisateur possède un rôle stocké en base de données : **CLIENT** ou **ADMIN**. Cette information est encodée dans le token JWT à la connexion, puis exploitée par le middleware d'authentification vu plus haut pour restreindre l'accès aux routes protégées.

Par exemple, un utilisateur standard ne pourra accéder qu'à ses propres données et à son tableau de bord, tandis qu'un administrateur aura accès à l'interface de gestion complète de la plateforme de e-learning.

Les principales opérations liées aux utilisateurs sont dans un premier temps la création d'un compte avec la route `/auth/register`, le contrôleur de cette route une fois la validation des identifiants, enverra un mail de vérification à l'utilisateur grâce au fichier `services/mailer.service`.

La connexion sécurisée depuis la route `/auth/login` se fait avec le mail et le mot de passe hashé et vérifié en base de données avant de délivrer le token; permettant l'accessibilité aux données personnelles de l'utilisateur ou bien l'espace administrateur.

Coté administrateur la gestion des utilisateurs se fait à l'aide des routes présentes dans le fichier `routes/admin/user.routes`, elles-même dépendantes du fichier contrôleur `controllers/admin/user.controller`.

L'accès aux données personnelles de l'intégralité des users se fait via la route `/admin/users/`. Les modifications ou suppressions d'utilisateur ne sont possibles que par l'ID de l'utilisateur concerné soit la route `/admin/users/:ID`

Ces routes sont sécurisées à l'aide de middlewares spécifiques qui vérifient non seulement la présence d'un token valide, mais aussi le rôle associé à l'utilisateur qui doit impérativement être admin.

Sur l'interface client, un utilisateur connecté après avoir créé et vérifié son compte, peut accéder à ses achats ou certifications via le dashboard.

Côté administrateur, une page dédiée permet de consulter la liste des comptes enregistrés, de visualiser leur rôle, et de suivre leur activité (achats réalisés, cursus validations, etc.).

Ce module offre un gage de fiabilité pour gérer l'ensemble des profils présents sur la plateforme. Il garantit la protection des données utilisateurs clients, les clients accèdent uniquement aux informations qui les concernent, tout en permettant à l'administrateur de piloter la base d'utilisateurs dans un environnement sécurisé. Il joue un rôle essentiel dans le projet.

Gestion des leçons

La gestion des leçons est au cœur de la plateforme *Knowledge Learning*, puisqu'elle constitue l'unité de base de la formation. Chaque leçon peut être achetée indépendamment, visualisée, et marquée comme validée par l'utilisateur. Cette fonctionnalité a été développée en lien étroit avec les modules d'achat, de progression et de certification.

La possibilité pour les utilisateurs connectés de voir les leçons achetables répond à la route public `/cursus/:cursusId/lessons`. Comme il est observable dans le fichier `index.js` des models, une leçon appartient obligatoirement à un cursus. Et bien évidemment une leçon peut avoir plusieurs achats par des utilisateurs différents. Aucune notion de stock car il s'agit de produit digital.

Coté administrateur la gestion des utilisateurs se fait à l'aide des routes présentes dans le fichier `routes/admin/lesson.routes`, elles-même dépendantes du fichier contrôleur `controllers/admin/lesson.controller`.

La création d'une nouvelle leçon répond aux critères du modèle d'une leçon dans le fichier `models/lesson.model`. Par conséquent une leçon a besoin du titre de la leçon,

de sa description, de son prix, du cursus ID correspondant et de l' URL de la Vidéo avec la formation.

Les leçons sont donc rattachées à un cursus pour former un parcours structuré. Les données sont stockées en base PostgreSQL via Sequelize, et les routes associées permettent de créer, modifier, consulter ou supprimer une leçon (</admin/lesson>).

L'utilisateur peut parcourir les leçons disponibles via une interface simple et claire. Concernant une leçon non achetée, il faut choisir le thème concerné, laissant le choix à possiblement plusieurs cursus eux-mêmes possédant plusieurs leçons. Une fois achetée, la leçon devient accessible depuis l'espace personnel.

Sur la page "Mes achats", l'utilisateur voit toutes les leçons qu'il possède, avec un bouton "Valider" qui lui permet d'indiquer qu'il a terminé la lecture. La validation de la leçon est ensuite enregistrée en base dans la table [ValidatedLesson](#).

Chaque validation est prise en compte dans le calcul global d'un cursus : si toutes les leçons rattachées à un cursus ont été validées, le cursus sera considéré comme validé et par conséquent une certification sera automatiquement générée. Le système repose donc sur une logique progressive, où chaque action utilisateur influence sa progression globale.

La gestion des leçons permet une grande souplesse pédagogique : les administrateurs peuvent publier ou retirer des contenus à tout moment à l'aide de la route </admin/lesson>, et les utilisateurs disposent d'un suivi clair et interactif.

Le code est organisé pour séparer les accès publics (affichage des leçons) des accès administrateurs (création ou modification côté admin), garantissant ainsi une structure propre et maintenable.

Gestion des cursus

Les cursus représentent un regroupement cohérent de plusieurs leçons appartenant à un thème. Cette fonctionnalité permet à la plateforme *Knowledge Learning* de proposer des parcours structurés, avec une logique de progression et de certification après validation.

La possibilité pour les utilisateurs connectés de voir les cursus acheteables répond à la route public </themes/:themeld/cursus>. Comme il est observable dans le fichier [index.js](#) des modèles, un cursus appartient obligatoirement à un thème et possède une à plusieurs leçons. Et bien évidemment un cursus peut avoir plusieurs achats par des utilisateurs différents. Aucune notion de stock car il s'agit d'un produit digital.

Côté administrateur la gestion des utilisateurs se fait à l'aide des routes présentes dans le fichier [routes/admin/cursus.routes](#), elles même dépendantes du fichier contrôleur [controllers/admin/cursus.controller](#).

La création d'un nouveau cursus répond aux critères du modèle d'un cursus dans le fichier `models/cursus.model`. Par conséquent un cursus a besoin d'un titre, de son prix et du thème ID correspondant.

Les cursus sont donc rattachés à un thème. Les données sont stockées en base PostgreSQL via Sequelize, et les routes associées permettent de créer, modifier, consulter ou supprimer une leçon (</admin/cursus>).

La relation entre cursus et leçons est gérée via une association dans la base de données, ce qui permet d'inclure une même leçon dans plusieurs cursus si nécessaire. Des routes protégées ont été développées pour créer, modifier ou supprimer un cursus (</admin/cursus>), avec contrôle des autorisations via JWT et rôle.

Sur l'interface publique, les cursus sont affichés avec leur titre, leur prix, et un bouton d'achat. Lorsqu'un utilisateur achète un cursus, l'ensemble des leçons associées lui devient accessible dans la page mes achats. Il peut alors suivre sa progression dans chaque leçon individuellement et la valider.

La progression est suivie automatiquement en fonction des validations effectuées sur les leçons composant le cursus. C'est à dire que lorsqu'un utilisateur valide toutes les leçons d'un même cursus, une certification est obtenue, sans intervention manuelle. Ce mécanisme repose sur une logique simple mais robuste : chaque validation alimente une table `ValidatedLesson`, et un contrôleur se charge de vérifier l'état d'avancement complet pour autoriser ou non l'émission du certificat [controllers/validation.controller](#).

Achats de contenus

Le système d'achat constitue une étape essentielle dans le fonctionnement de *Knowledge Learning*, puisqu'il détermine l'accès aux leçons et cursus. Bien que le projet soit en mode développement, une logique complète d'achat a été mise en place, simulant un processus d'acquisition simple et fluide pour l'utilisateur.

Lorsqu'un utilisateur souhaite acheter une leçon ou un cursus, il clique sur un bouton "Acheter" associé au contenu. L'action déclenche un appel API vers le back-end, qui enregistre l'achat dans la base de données. Aucun paiement réel n'est effectué, mais toute la logique de validation, de contrôle d'accès et de retour d'information est en place comme si un paiement avait eu lieu.

La possibilité pour l'utilisateur avec le rôle client d'acheter une leçon ou un cursus répond aux routes : </buy//cursus/:id> et </buy//cursus/:id> situé dans le fichier `routes/purchase.routes` dépendantes du contrôleur `controllers/purchase.controller`.

Ainsi le contrôleur permet de vérifier si la leçon ou le cursus choisi existe bien dans la base de données. Lors de l'achat d'un cursus, le contrôleur permet aussi l'achat

des leçons qui sont rattachées et lors de l'achat de l'intégralité des leçons d'un cursus, le cursus est considéré comme acheté. Il est à noter que la possibilité de faire l'achat d'un cursus est plus avantageux pécuniairement que l'achat des leçons du même cursus individuellement.

La création du modèle purchase dans le fichier `models/purchase.model` a permis la création d'une table dans la base de données permettant de stocker l'intégralité des achats. Conformément au modèle purchase, lors de la réalisation d'un achat sur la plateforme Knowledge Learning, l'achat est enregistré dans la base de données avec les éléments suivants : identifiant de l'utilisateur, l'identifiant du contenu acheté (leçon ou cursus) et la date de l'achat.

Un système de vérification est ensuite mis en place sur les pages privées : si l'utilisateur n'a pas acheté le contenu, l'accès est bloqué.

Deux contrôleurs gèrent les achats :

- `purchase.controller.js` pour les utilisateurs permettant les achats mais aussi la consultation leçons et cursus achetés.
- `admin/purchase.controller.js` pour les administrateurs (visualisation de toutes les ventes) et possibilité de les supprimer.

Une fois l'achat effectué, l'utilisateur retrouve immédiatement son contenu dans l'espace personnel. Les cursus et leurs leçons avec leurs contenus sont débloqués, et des boutons "Valider" deviennent disponibles. Cette interaction en temps réel renforce l'expérience utilisateur et donne un vrai sentiment de fluidité.

Même sans intégration de paiement bancaire en production, le système d'achat est entièrement fonctionnel grâce à l'intégration de **Stripe en mode bac à sable**. Cela permet de simuler un parcours d'achat complet, incluant la gestion des droits d'accès, l'enregistrement des transactions, la validation des cursus et l'émission automatique de certificats. L'architecture reste ouverte, et l'ajout d'autres solutions comme **Paypal**, **Lydia** ou tout autre prestataire est parfaitement envisageable à l'avenir. Ce module constitue une base solide pour toute plateforme de formation en ligne à visée commerciale.

Validation et progression

Afin de permettre à chaque utilisateur de suivre son avancée dans ses formations, un système de validation a été mis en place comme rapidement évoqué plus tôt. Il permet de marquer une leçon ou un cursus comme "validé". Dans le cas où seule une leçon d'un cursus est validée le cursus demeure non validé jusqu'à la validation de l'intégralité des leçons le composant. Cette logique joue un rôle central dans le déclenchement des certifications.

Une fois une leçon achetée et accessible, l'utilisateur peut cliquer sur un bouton "Valider" affiché dans son tableau de bord section mes achats. Cette action déclenche un appel API `/validate-protected/lesson/:ID` qui enregistre la validation dans la table `ValidatedLesson`.

Conformément au modèle `validated Lesson` consultable dans le fichier `models/Lesson.Model` L'enregistrement d'une leçon validée se fait dans cette table avec l'ID de l'utilisateur, l'ID de la leçon concernée et la date de validation. Cette opération est simple pour l'utilisateur, mais essentielle pour suivre la progression et verrouiller les accès aux certifications.

Un contrôle permet également d'empêcher une double validation, et d'afficher visuellement l'état de la leçon (validée / non validée) directement sur la page "Mes contenus".

La progression dans un cursus repose sur l'état de validation de chacune de ses leçons. Une fois toutes les leçons d'un cursus validées, une vérification est automatiquement faite pour déclencher la complétion du cursus. Cette logique est gérée côté back-end dans le contrôleur `validation.controller.js`, qui vérifie la correspondance entre les leçons validées et celles attendues pour le cursus.

Le résultat en back de nos contrôleurs conditionne la mise à jour côté front dans le tableau de bord, permettant à l'utilisateur de voir clairement ce qui lui reste à faire avant d'obtenir son certificat.

Toutes les validations sont historisées dans la base de données dans les tables `validatedlesson` et `validatedcrsus`. Cette base servira à terme à enrichir la plateforme de fonctionnalités futures comme des relances, des taux de complétion, ou des notifications automatiques.

Génération de certifications

La certification représente l'aboutissement du parcours utilisateur sur *Knowledge Learning*. Elle permet de matérialiser l'effort accompli par l'apprenant et d'attester de la complétion d'un cursus. Ce module renforce la dimension professionnelle de la plateforme tout en apportant une forte valeur perçue à l'expérience de formation.

La certification est effectuée dans le contrôleur `validation.controllers` dans la fonction qui sert à valider une leçon où dans celle qui sert à valider un cursus grâce à la méthode `FindOrCreate`. En effet, cette méthode me permet de vérifier lors de la validation d'une leçon ou d'un cursus si la certification est déjà existante et dans le cas où elle n'existe pas la méthode me permet de la créer.

Lors du chargement de la page "Mes certifications". Le contrôleur `controller/certification.controller` vérifie dans la base de données les certifications

créées comme vu précédemment et les transmet grâce à la route </certifications/my-certifications>.

Les certifications obtenues apparaissent sous forme de cartes avec Le nom du cursus, la date d'obtention ainsi que le nom de l'utilisateur qui a obtenu la certification.

Pour expliquer l'automatisation de la certification il suffit de regarder dans le fichier `models/index.js`, Il est parfaitement visible selon les termes Sequelize qu'une certification appartient à un cursus mais aussi à un utilisateur d'où la présence des deux clés étrangères `user ID` et `cursus ID`.

Ce design simple et lisible permet à l'utilisateur de visualiser rapidement ses réussites. À terme, une fonctionnalité d'export PDF ou d'impression pourra être ajoutée pour formaliser davantage ces attestations.

L'interface admin ne propose aucune possibilité d'action sur les certifications des utilisateurs ni la possibilité de les lire. Les certifications étant données à la validation des cursus les candidats peuvent toujours enlever cette validation et perdre la certification actuellement. Néanmoins les routes ont été créées pour un ajout possible dans le futur. (</routes/admin/certification.routes> et </controller/admin/certifications.controller>).

La génération de certificats apporte une vraie dimension de reconnaissance dans l'expérience utilisateur. Elle renforce l'engagement, stimule la motivation, et crédibilise l'ensemble du dispositif de formation.

Interface front-end

L'interface front-end de *Knowledge Learning* a été conçue dans un souci constant de simplicité, de lisibilité et de cohérence. Elle repose sur le framework `Vue.js`, avec une architecture basée sur des vues indépendantes reliées par un système de routes dynamiques, gérées via `Vue Router`, qui facilite aussi l'aspect responsive. Fidèle aux maquettes en annexe, chaque vue correspond à une page fonctionnelle de l'application : inscription, connexion, tableau de bord, exploration des contenus, gestion des achats ou encore affichage des certifications.

Interface utilisateur Client

Du côté du client, la navigation est pensée de manière linéaire. Après la connexion, l'utilisateur est dirigé vers son tableau de bord, depuis lequel il peut consulter et apprendre les leçons achetées, consulter ses certifications ou alors se rendre dans les thèmes et explorer parmi les cursus et leçons disponibles à l'achat.

Chaque page est épurée et va droit à l'essentiel : un bouton pour une action, une page pour un seul service. Les composants sont réutilisables (cartes de leçons,

boutons, sections thématiques), facilitant à la fois le développement et l'homogénéité visuelle. L'expérience est également responsive, c'est-à-dire adaptée à tous types d'écrans, du mobile au desktop.

Interface utilisateur administrateur

Une fois connecté avec un compte administrateur, l'interface change de logique. Le menu supérieur donne accès à un nouvel onglet nommé admin. Ce nouvel onglet est la possibilité à l'administrateur d'accéder au panel .admin.

L'interface admin reprend les composants du front classique, mais y ajoute des outils de gestion, des formulaires plus complets, ainsi que des boutons d'action spécifiques ("Modifier", "Supprimer", "Ajouter"). Toutes ces vues sont protégées par des middlewares d'authentification et de rôle, garantissant que seuls les comptes autorisés peuvent y accéder.

L'ensemble de l'interface front-end a été construit de manière modulaire, avec une logique de navigation claire, cohérente et sécurisée. Cette organisation a permis une évolution fluide du projet tout en assurant une expérience utilisateur intuitive et efficace, que ce soit pour l'apprenant ou pour l'administrateur de la plateforme.

Architecture des données, difficultés rencontrées & extraits de code

L'un des points les plus structurants — et aussi les plus complexes — du développement back-end de *Knowledge Learning* a été la définition de la base de données et des relations entre les modèles Sequelize.

Tout repose sur un fichier central, `models/index.js`, qui importe tous les modèles et établit entre eux les associations logiques nécessaires au bon fonctionnement du projet.

Ce fichier m'a permis de surtout centraliser et par conséquent faciliter l'ensemble des relations métier : utilisateurs, leçons, cursus, thèmes, achats, validations et certifications.

Relation 1:n entre Thème et Cursus fichier `models/index.js`

```

/* Theme has many Cursus */
Theme.hasMany(Cursus, { foreignKey: 'ThemeId', as: 'cursus', onDelete: 'CASCADE' });
Cursus.belongsTo(Theme, { foreignKey: 'ThemeId', as: 'theme' });

/**
 * Cursus has many Lessons */
Cursus.hasMany(Lesson, { foreignKey: 'CursusId', as: 'lessons', onDelete: 'CASCADE' });
Lesson.belongsTo(Cursus, { foreignKey: 'CursusId', as: 'cursus' });

```

Ce bloc à titre d'exemple a été primordial pour bien comprendre toute sa subtilité; il m'a appris l'importance des alias dans Sequelize. Sans le champ `as`, les jointures dans les requêtes `include` ne fonctionnaient pas, ou retournaient des noms de colonnes inattendus ce qui fut une des premières difficultés. J'ai dû expérimenter via Postman pour comprendre comment Sequelize transformait mes modèles en relations SQL.

Le `onDelete: 'CASCADE'` a été ajouté dès le départ pour éviter d'avoir des cursus sans thème si un thème venait à être supprimé.

Relation 2 : relation n:n/many-to-many avec ValidatedCursus

```

/**
 * Many-to-many: Users <=> Validated Cursus */
User.belongsToMany(Cursus, { through: ValidatedCursus, foreignKey: 'UserId', otherKey: 'CursusId' });
Cursus.belongsToMany(User, { through: ValidatedCursus, foreignKey: 'CursusId', otherKey: 'UserId' });

ValidatedCursus.belongsTo(Cursus, { foreignKey: 'CursusId', onDelete: 'CASCADE' });
ValidatedCursus.belongsTo(User, { foreignKey: 'UserId', onDelete: 'CASCADE' });

User.hasMany(ValidatedCursus, { foreignKey: 'UserId', onDelete: 'CASCADE' });
Cursus.hasMany(ValidatedCursus, { foreignKey: 'CursusId', onDelete: 'CASCADE' });

```

Ce type de relation a été le plus difficile à mettre en place. J'avais besoin que chaque utilisateur puisse valider plusieurs cursus, et que chaque cursus puisse être validé par plusieurs utilisateurs comme le décrit une relation many to many.

Sequelize permet ça via le champ `through`, mais encore fallait-il bien comprendre comment gérer les clés.

Pendant plusieurs heures, je recevais des erreurs, jusqu'à ce que je découvre que l'ordre des clés et la présence d'`otherKey` avaient un impact direct sur la structure de ma table.

Ce problème m'a forcé à lire la documentation, et de comprendre comment Sequelize construit réellement ses tables de jointure. J'ai fini par créer un modèle `ValidatedCursus` séparé pour mieux gérer la logique métier.

Malgré l'ajout de `onDelete: 'CASCADE'` sur mes relations, certaines suppressions ne s'exécutaient pas correctement. Par exemple, lorsqu'un utilisateur était supprimé, ses validations restaient en base. J'ai d'abord cru à une erreur de syntaxe, mais le problème venait en réalité de **l'ordre d'importation des modèles dans `index.js`**, ainsi que d'un mauvais usage de `sequelize.sync()`.

Après plusieurs tentatives, j'ai compris que Sequelize ne créait pas toujours les clés étrangères dans le bon ordre si les modèles n'étaient pas bien déclarés. La correction a consisté à centraliser toutes les associations dans `index.js`, et à appeler la synchronisation après avoir défini toutes les relations.

Structure finale des modèles & utilité

Voici un aperçu du code exporté à la fin du fichier :

```
/**
 * Exports all models and Sequelize instance
 */
module.exports = {
  sequelize,
  User,
  Theme,
  Cursus,
  Lesson,
  Purchase,
  ValidatedLesson,
  ValidatedCursus,
  Certification,
};
```

L'ensemble des fichiers du projet (contrôleurs, services, seeders, etc.) est dépendant de cet export. Il permet d'importer directement les modèles Sequelize depuis un seul

point d'entrée. Ce design m'a permis de simplifier l'organisation globale, de mieux séparer les responsabilités, et de faciliter le test unitaire des différentes fonctionnalités.

Ce fichier m'a clairement posé des difficultés techniques, mais il m'a aussi permis d'apprendre énormément. Avant ce projet, je ne savais pas modéliser correctement une base relationnelle avec un ORM comme sequelize. Aujourd'hui, je suis capable d'utiliser Sequelize avec une vraie logique métier, de comprendre ses subtilités (alias, cascade, jointure personnalisée), et de structurer un modèle de données complet. Cette étape a été un tournant dans ma compréhension du back-end.

Gestion de la sécurité : authentication, tokens et CSRF

La sécurité étant un pilier essentiel de toute application web, plusieurs mécanismes ont été mis en place dans Knowledge Learning pour garantir la protection des données des utilisateurs, l'intégrité des requêtes et la différenciation des accès selon les rôles.

Authentication avec JSON Web Token (JWT)

Le système d'authentification repose sur les **JSON Web Tokens**, une méthode sécurisée et stateless qui permet d'authentifier un utilisateur. Lorsqu'un utilisateur se connecte, un token est généré côté serveur et renvoyé au client, qui le stocke (dans un cookie ou localStorage selon le besoin).

Exemple de génération de token :

```

const token = jwt.sign(
  {
    id: user.id,
    email: user.email,
    role: user.role
  },
  process.env.JWT_SECRET,
  { expiresIn: '24h' }
);

res.cookie('token', token, {
  httpOnly: true,
  sameSite: 'none',
  secure: true,
  maxAge: 1000 * 60 * 60 * 24
});

res.status(200).json({ message: 'Login successful', token });
} catch (error) {
  console.error('Login error:', error);
  res.status(500).json({ message: 'Server error' });
}
};

```

Ce token encode trois informations essentielles : l'identifiant de l'utilisateur, son email et son rôle (client ou admin). Il est signé à l'aide d'une **clé secrète** contenue dans une variable d'environnement (**JWT_SECRET**), rendant toute tentative de falsification inutile sans la clé.

Le token est ensuite envoyé au client sous forme de cookie sécurisé.

httpOnly: true : interdit l'accès au cookie via JavaScript, empêchant les attaques XSS.

secure: true : exige le protocole HTTPS.

sameSite: 'none' : nécessaire pour le bon fonctionnement des cookies en cross-origin (notamment en développement avec Netlify + Render).

maxAge : durée de vie fixée à 24h.

Enfin, une réponse sous format JSON est envoyée pour confirmer la réussite de la connexion.

Middleware de vérification

Chaque route protégée passe par un middleware `checkJWT`, ci-dessous.

```
exports.checkJWT = (req, res, next) => {
  const token =
    req.cookies?.token || req.headers?.authorization?.split(' ')[1];

  if (!token) {
    return res.status(401).json({ message: 'Token manquant dans le cookie.' });
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    return res.status(403).json({ message: 'Token invalide ou expiré.' });
  }
};

// 🛡️ Middleware de vérification du rôle (admin, etc.)
exports.checkRole = (role) => {
  return (req, res, next) => {
    if (!req.user || req.user.role !== role) {
      return res.status(403).json({ message: `Accès interdit. Rôle requis : ${role}` });
    }
    next();
  };
};
```

Ce middleware `checkJWT` est appelé à chaque requête vers une route protégée. Il vérifie la présence du token JWT soit dans les cookies, soit dans l'en-tête `Authorization`. Si le token est absent ou invalide, la requête est rejetée. Le token est ensuite déchiffré et validé à l'aide de la clé secrète (`JWT_SECRET`). En cas d'échec (ex. expiration, falsification), une erreur 403 est retournée.

Vérification des rôles

Un second middleware `checkRole` permet de restreindre l'accès à certaines routes selon le rôle (ci-dessus).

Ainsi, l'accès à des routes sensibles (ajout de leçons, gestion des utilisateurs) est strictement réservé à des utilisateurs autorisés par le bon rôle.

Ces deux middlewares sont des éléments clés de la sécurité back-end : ils assurent non seulement l'identification de l'utilisateur via JWT, mais aussi l'autorisation selon son rôle.

Protection contre les attaques CSRF

Pour éviter les attaques Cross-Site Request Forgery, un système de jeton CSRF a été intégré dans l'application via le middleware `csrf`.

```
const csrfProtection = csrf({ cookie: true })

router.get('/token', csrfProtection, (req, res) => {
  res.cookie('XSRF-TOKEN', req.csrfToken())
  res.status(200).json({ message: 'CSRF token envoyé' })
})

module.exports = router
```

Chaque requête POST ou DELETE doit être accompagnée d'un **token CSRF valide**, généré côté serveur et envoyé dans la page (ou par une route dédiée `/csrf-token`). Ce jeton est ensuite injecté dans les en-têtes des requêtes côté client.

```
onMounted(async () => {
  try {
    const res = await api.get('/security/csrf-token')
    csrfToken.value = res.data.csrfToken
  } catch (err) {
    console.error('Erreur CSRF', err)
  }
})

const handleLogin = async () => {
  try {
    // 1. POST /auth/login
    await api.post('/auth/login', {
      email: email.value,
      password: password.value
    }, {
      withCredentials: true,
      headers: { 'X-CSRF-Token': csrfToken.value }
    })
  }
}
```

Cela empêche un site malveillant de forcer l'utilisateur à exécuter des actions sans son consentement.

En combinant le jeton JWT pour l'authentification et CSRF pour la protection des requêtes, Knowledge Learning est sécurisé et protège la connexion des utilisateurs, bloque les tentatives d'usurpation ou de requêtes malveillantes et distingue clairement les utilisateurs clients des administrateurs.

Modélisation des données et initialisation avec `seed.js`

Afin d'avoir un environnement de développement reproductible et réaliste, j'ai conçu un script d'amorçage de la base de données : `seed.js`. Ce script a pour rôle d'initialiser toutes les données de base à l'application, en suivant un ordre logique qui respecte les dépendances entre modèles, essentiel pour tester et faire fonctionner la plateforme.

Le fichier commence par charger les variables d'environnement (`.env`), les modèles Sequelize (définis dans `models/`) et leur dépendance décrite dans `index.js` ainsi que `bcrypt` pour le hashage des mots de passe utilisateurs.

```
require('dotenv').config();
const { sequelize, User, Theme, Coursus, Lesson } = require('./models');
const bcrypt = require('bcrypt');
```

L'appel à `sequelize.sync({ force: true })` supprime toutes les tables existantes puis les crée de nouveau, assurant une remise à zéro totale de la base. Cette opération est précieuse pendant les phases de test et de mise au point.

Création des données pédagogiques

Le script commence par insérer quatre **thèmes** (Musique, Informatique, Jardinage, Cuisine), chacun servant de conteneur logique à plusieurs cursus. Ces thèmes sont créés en parallèle grâce à `Promise.all()` :

```
// Thèmes
const [musique, informatique, jardinage, cuisine] = await Promise.all([
  Theme.create({ name: 'Musique' }),
  Theme.create({ name: 'Informatique' }),
  Theme.create({ name: 'Jardinage' }),
  Theme.create({ name: 'Cuisine' })
]);
```

Ensuite, plusieurs **cursus** sont créés pour chaque thème, avec un prix défini. Chaque cursus est ensuite lié à plusieurs **leçons**, insérées grâce à `Lesson.bulkCreate()` avec des informations données et respectant le modèle

(titre, prix, description, lien vidéo). Ces vidéos sont issues de YouTube et simulées via des URLs génériques.

Cette structure imbriquée (Thème → Coursus → Leçons) illustre concrètement les relations définies dans `models/index.js`, et se retrouve dans la base de données.

Création d'utilisateurs

Enfin, deux comptes sont ajoutés à la base : un **client** et un **administrateur**. Leur mot de passe est hashé avec `bcrypt` pour simuler un comportement sécurisé dès le début de la base de données de notre projet.

```
// Utilisateurs
const hashedPassword = await bcrypt.hash('azerty123', 10);
await User.bulkCreate([
  {
    fullName: 'Client Test',
    email: 'client@example.com',
    password: hashedPassword,
    role: 'client',
    isActive: true
  },
  {
    fullName: 'Admin Test',
    email: 'admin@example.com',
    password: hashedPassword,
    role: 'admin',
    isActive: true
  }
]);
```

Ces comptes sont ensuite utilisés pour tester les différents scénarios d'authentification, de gestion de droits, d'achats, et de progression dans l'application.

Intérêt et utilisations

Le script `seed.js` permet de pouvoir réinitialiser la base à tout moment avec des données propres, de tester l'application de manière réaliste sans avoir à saisir manuellement des données à chaque fois, de vérifier la cohérence des relations entre les différents modèles du projet, de garantir que les interfaces client et

administrateur disposent de contenu à afficher dès le début afin de développer au mieux le côté frontend et donc visible du projet.

Ce script a été exécuté régulièrement au cours du projet pour tester la stabilité de l'application à différentes étapes. Il a également servi de base pour les tests automatisés.

Tests et validation du projet

Pour vérifier le bon fonctionnement et la fiabilité de notre plateforme, j'ai utilisé une combinaison d'outils qui m'ont permis de tester les fonctions principales de la plateforme. Les outils utilisés sont les suivants:

- **Jest** : Jest est un framework de test en JavaScript, principalement utilisé pour tester les applications React,, utilisé pour exécuter les tests unitaires.
- **Supertest** : Bibliothèque pour tester les serveurs HTTP node.js
- **Sequelize** : la base de données est réinitialisée à chaque exécution via `sequelize.sync({ force: true })`, garantissant une base propre pour chaque test.
- **Bcrypt** : pour hasher les mots de passe lors de l'initialisation des deux utilisateurs.
- **CSRF & JWT** : les tests intègrent les aspects de sécurité tels que les jetons CSRF (`X-CSRF-Token`) et l'authentification par token JWT.

Le fichier `test/auth.test.js` regroupe plusieurs tests clés qui se retrouvent en cas d'usage de la plateforme :

Le premier test effectué est la Création d'un compte utilisateur. Le test envoie une requête `POST /auth/register` avec un corps contenant les informations d'un nouvel utilisateur. Le test assure que la création renvoie un code `200` ou `201`.

Le deuxième test est une connexion utilisateur et la récupération du token JWT. Le login utilisé la route `/auth/login` pour la tester, avec en-tête CSRF. Le test vérifie la réception correcte du jeton JWT dans la réponse.

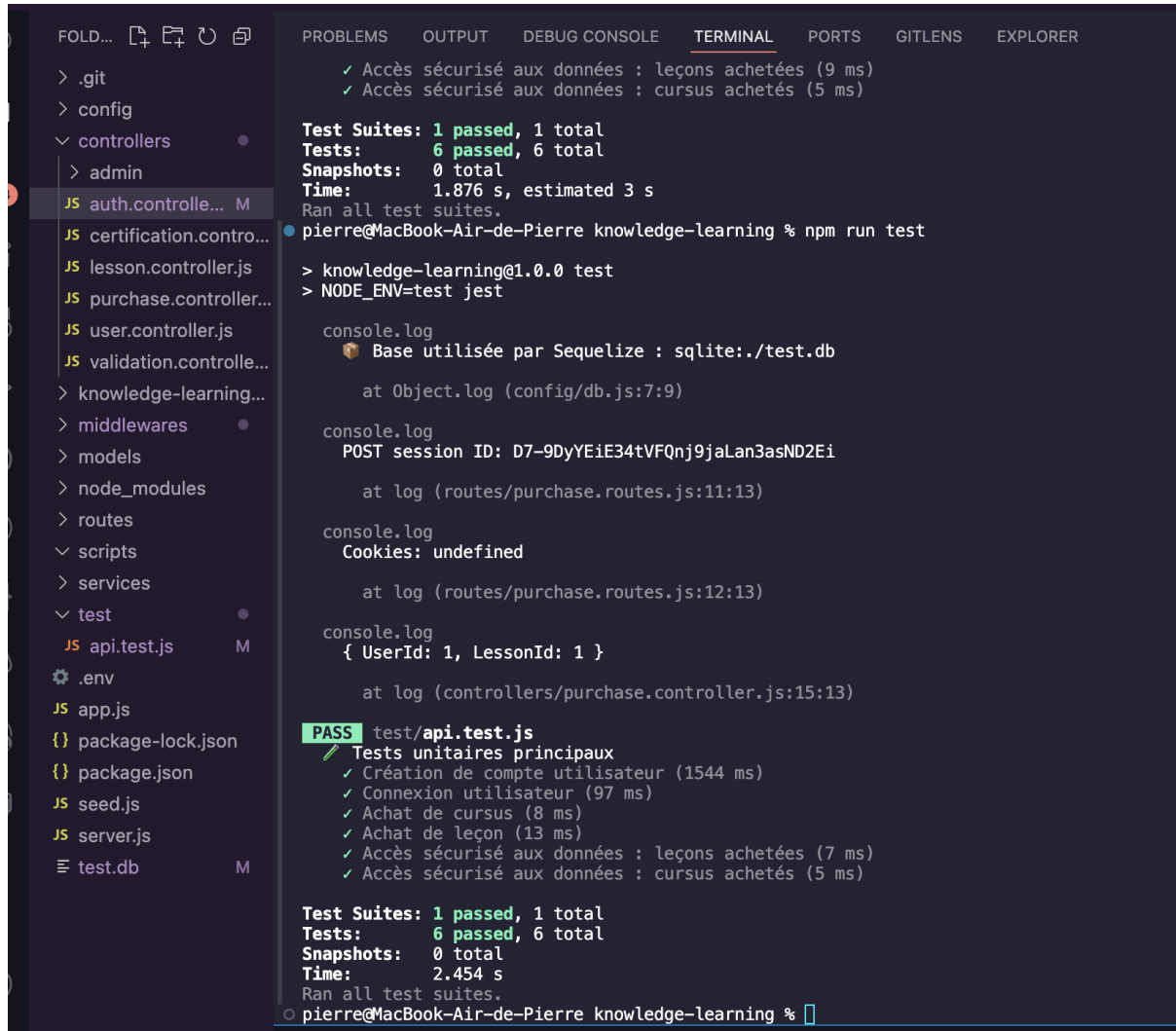
Les tests suivants sont les achats d'un cursus et d'une leçon. Pour ce faire les deux appels sont réalisés sur les routes sécurisées `/buy/cursus/:id` et `/buy/lesson/:id`. Ils vérifient que l'utilisateur authentifié peut effectuer un achat, et que le serveur répond avec succès (`201`).

Les derniers tests permettent la consultation des achats effectués précédemment l'utilisateur connecté interroge ses leçons et cursus achetés avec les routes `/buy/my-lessons` et `/buy/my-cursus`. Les réponses doivent être des tableaux (`200`), prouvant l'efficacité des relations et du middleware d'authentification.

Avant les tests, la base de données de test (différentes de celle de la production) est synchronisée et des données minimales sont injectées :

Ensuite, le jeton CSRF est récupéré grâce à la route `/security/csrf-token`, puis un token JWT est obtenu via une requête de login réussie. Ces deux éléments sont utilisés dans tous les tests sécurisés via les headers.

L'exécution des tests aboutit à une série de vérifications positives avec l'ensemble des routes critiques couvertes. Voici un aperçu du résultat attendu en console :



```
FOLD... [Icons]
> .git
> config
  > controllers
    > admin
    JS auth.controlle... M
    JS certification.contro...
    JS lesson.controller.js
    JS purchase.controller...
    JS user.controller.js
    JS validation.controlle...
  > knowledge-learning...
  > middlewares
  > models
  > node_modules
  > routes
  > scripts
  > services
  > test
    JS api.test.js M
    .env
    JS app.js
    {} package-lock.json
    {} package.json
    JS seed.js
    JS server.js
    test.db M

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS EXPLORER
  ✓ Accès sécurisé aux données : leçons achetées (9 ms)
  ✓ Accès sécurisé aux données : cursus achetés (5 ms)

Test Suites: 1 passed, 1 total
Tests: 6 passed, 6 total
Snapshots: 0 total
Time: 1.876 s, estimated 3 s
Ran all test suites.
pierre@MacBook-Air-de-Pierre knowledge-learning % npm run test
> knowledge-learning@1.0.0 test
> NODE_ENV=test jest

console.log
  Base utilisée par Sequelize : sqlite:./test.db
    at Object.log (config/db.js:7:9)

console.log
  POST session ID: D7-9DyYeiE34tVFQnj9jaLan3asND2Ei
    at log (routes/purchase.routes.js:11:13)

console.log
  Cookies: undefined
    at log (routes/purchase.routes.js:12:13)

console.log
  { UserId: 1, LessonId: 1 }
    at log (controllers/purchase.controller.js:15:13)

PASS test/api.test.js
  Tests unitaires principaux
    ✓ Création de compte utilisateur (1544 ms)
    ✓ Connexion utilisateur (97 ms)
    ✓ Achat de cursus (8 ms)
    ✓ Achat de leçon (13 ms)
    ✓ Accès sécurisé aux données : leçons achetées (7 ms)
    ✓ Accès sécurisé aux données : cursus achetés (5 ms)

Test Suites: 1 passed, 1 total
Tests: 6 passed, 6 total
Snapshots: 0 total
Time: 2.454 s
Ran all test suites.
pierre@MacBook-Air-de-Pierre knowledge-learning %
```

Tous les tests passent avec succès, confirmant le bon fonctionnement des logiques métiers, la sécurité des accès, et la solidité de l'API.

Accès et utilisation pour un utilisateur client

Créer un compte

Nom complet :

Email :

Mot de passe :

L'utilisateur peut créer un compte depuis la page d'inscription accessible depuis la barre de navigation. Il doit remplir pour ça son nom complet, son adresse mail et un mot de passe sécurisé.

✗ **Lien invalide ou expiré.**

Une fois le formulaire complété, il recevra un lien de validation par email. En cas de lien expiré, un message clair s'affiche.

Connexion

Email :

Mot de passe :

Une fois le client inscrit et activé, l'utilisateur peut se connecter sur la page de connexion. Pour cela, il doit juste renseigner son **email** et **mot de passe**. Et si les informations sont valides, il est redirigé vers son **tableau de bord** personnalisé.

Tableau de bord

Bienvenue sur votre tableau de bord

Vous pouvez accéder à vos leçons, cursus, certifications et effectuer des achats.

[Explorer les thèmes](#)

[Mes certifications](#)

[Mes achats](#)

[Espace administrateur](#)

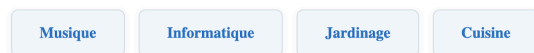
Le tableau de bord est la page principale de navigation pour les clients. Il permet d'accéder à toutes les fonctionnalités, **explorer les thèmes** va permettre de découvrir les cursus disponibles (ex. développement web, jardinage,

cuisine...), le menu **mes achats** liste les leçons et cursus achetés et **mes certifications** répertorie les certifications obtenues après validation.

Le bouton « **Espace administrateur** » n'est visible que pour les comptes administrateurs.

Explorer les thèmes

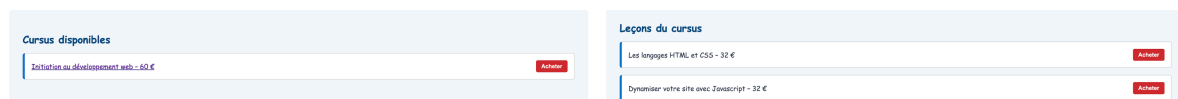
Thèmes de formation



L'utilisateur accède à une liste de thèmes (informatique, cuisine, jardinage...).

Chaque thème regroupe plusieurs cursus.

Achat d'un cursus ou d'une leçon



Les leçons et cursus sont affichés avec leurs prix.

Un simple clic sur le bouton rouge **Acheter** déclenche le processus d'achat via **Stripe (sandbox)**.

Mes achats



Les leçons et/ou cursus achetés apparaissent dans Mes achats du dashboard. L'utilisateur peut les **valider** (bouton vert) pour obtenir sa certification.

Chaque leçon a une **vidéo intégrée** et un **texte explicatif**.

Certifications

Mes certifications

✓ Certification - Initiation au jardinage - TESTEUR
Obtenu le : 26/06/2025

✓ Certification - Initiation à la cuisine - TESTEUR
Obtenu le : 26/06/2025

Une fois un cursus validé, l'utilisateur obtient automatiquement une certification avec la date d'obtention, l'intégralité des certifications sont dans la section mes certifications du tableau de bord .

Utilisation pour un utilisateur Administrateur

L'administrateur accède à une interface dédiée grâce à son rôle (**admin**) détecté au moment de la connexion. Cette interface permet la gestion complète des contenus pédagogiques et des utilisateurs. Les principales fonctionnalités sont la gestion des thèmes, des cursus, des leçons et des utilisateurs.

The screenshot shows the 'Interface d'administration' with two main sections: 'Thèmes' and 'Leçons'. The 'Thèmes' section lists 'Musique', 'Informatique', 'Jardinage', and 'Cuisine', each with a 'Supprimer' button. Below this is a form to 'Ajouter' a new theme with fields for 'Nouveau thème' and 'ID', and a 'Modifier' button. The 'Leçons' section lists various lessons like 'Les accords et les gammes - 26 €' and 'Découverte de l'instrument - 26 €', each with a 'Supprimer' button.

The screenshot shows the 'Cursus' section, which lists various courses like 'Initiation à la guitare - 50 €' and 'Initiation au piano - 50 €', each with a 'Supprimer' button. Below the list is a form to 'Ajouter' a new course with fields for 'Titre', 'ID', and 'Nouveau titre', and a 'Modifier' button. A note at the bottom says 'Sélectionnez un élément dans la liste.'

Gestion des thèmes : les thèmes (musique, jardinage, informatique, etc.) peuvent être ajoutés, renommés ou supprimés.

Gestion des cursus : création, modification, suppression de cursus et de leur prix.

Gestion des leçons : affichage de toutes les leçons avec leur prix, possibilité de les supprimer.

The screenshot shows the 'Utilisateurs' section, which lists two users: 'Client Test - client@example.com (client)' and 'Admin Test - admin@example.com (admin)', each with 'Modifier' and 'Supprimer' buttons. Below this is a form to 'Ajouter un utilisateur' with fields for 'Nom complet', 'Email', 'Mot de passe (laisser vide)', and a dropdown for 'client', with 'Créer' and 'Annuler' buttons. At the bottom, there is a section for 'Achats'.

La **Gestion des utilisateurs** : création, modification, suppression.
L'administrateur peut changer les rôles et voir la liste complète des comptes.

Le **Suivi des achats** : tous les achats effectués sont visibles dans l'interface admin pour contrôle. L'administrateur peut ainsi facilement enrichir la plateforme en contenus ou corriger d'éventuelles erreurs.

La plateforme a été développée dès le début pour évoluer dans le temps. Plusieurs pistes d'amélioration sont envisagées, actuellement seul l'intégration de Stripe en mode sandbox représente un moyen de paiement. À l'avenir il sera entièrement possible d'en ajouter d'autres comme paypal, Lydia ou Apple Pay. Concernant le contenu de la plateforme il sera possible d'ajouter des thèmes des cursus ou des leçons, le site est d'ores et déjà prêt à les accueillir. Pour du contenu innovant il est imaginable que la plateforme intègre un système de visioconférence pour des cours ou des formations en direct, ou alors un système de messagerie afin d'échanger avec de potentiels professeurs ou formateurs. Enfin pour terminer on peut imaginer que l'aspect ludique des formations évolue en des jeux ou en des systèmes autres que des vidéos.

Bilan

Ce projet m'a donné l'opportunité de consolider et de tester toutes mes connaissances acquises lors de mon apprentissage en tant que développeur Web et mobile. En effet, de l'analyse du besoin jusqu'à la mise en ligne de l'application j'ai été amené à réaliser toutes les étapes du développement de l'application dans un environnement réaliste.

Sur le plan technique, ce projet m'a amené à m'imprégner de fond en comble des outils modernes de développement web full-stack, y compris Node.js, Express, Sequelize, Vue.js, et l'intégration de services externes tels que stripe. Plus précisément, j'ai renforcé mes compétences en matière de sécurité en ce qui concerne l'authentification JWT, la protection CSRF, l'organisation des bases de données relationnelles, ainsi que ce qui se rapporte à une architecture MVC propre.

En plus des compétences acquises, j'ai également appris à mieux m'organiser au niveau du travail, à mieux documenter proprement son code, et à m'impliquer davantage sur les prévisions de besoins des utilisateurs, que ces derniers soient des clients ou des administrateurs. J'ai également été plus rigoureux dans les tests et dans la validation de mon code, sur la documentation de l'application.

De façon plus générale, ce projet m'a permis de me sensibiliser à l'importance des aspects UX et UI. J'ai essayé ici de développer une application à la fois intuitive et accessible, autant pour mes collègues professionnels du domaine et les départements de recherche que pour tout utilisateur de tout âge. En outre, ce travail

m'a donné une vue claire des aspects sur lesquels je vais travailler à l'avenir, afin de toujours améliorer mes pratiques, apprendre et progresser.

En conclusion, je suis fier du travail accompli, de l'autonomie acquise, et de la montée en compétence progressive que ce projet m'a apportée. Il constitue une base solide sur laquelle je souhaite m'appuyer pour continuer à progresser, intégrer une équipe technique stimulante, et contribuer à des projets web utiles, maintenables et ambitieux.

FAIVRE Pierre

Annexe 1 Maquettes home et connexion



ConnexionInscription


Créer un compte

Nom

Email

Mot de passe

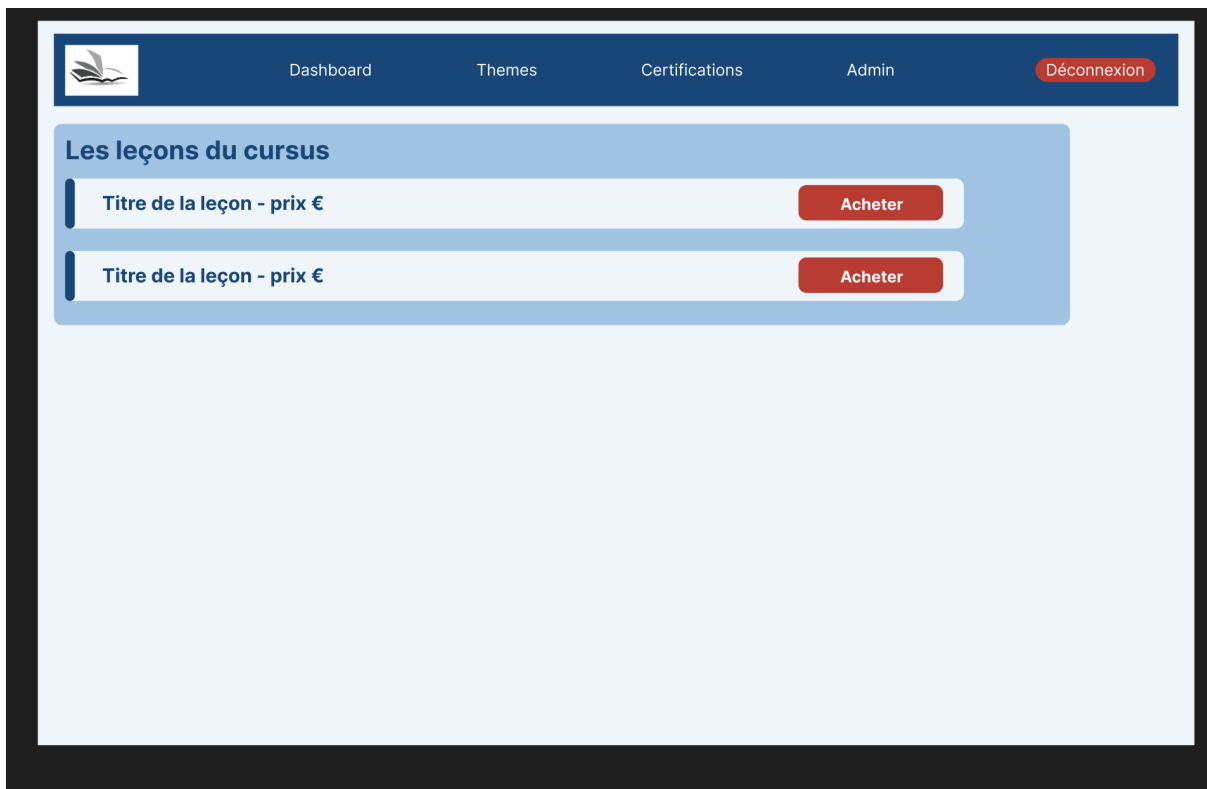
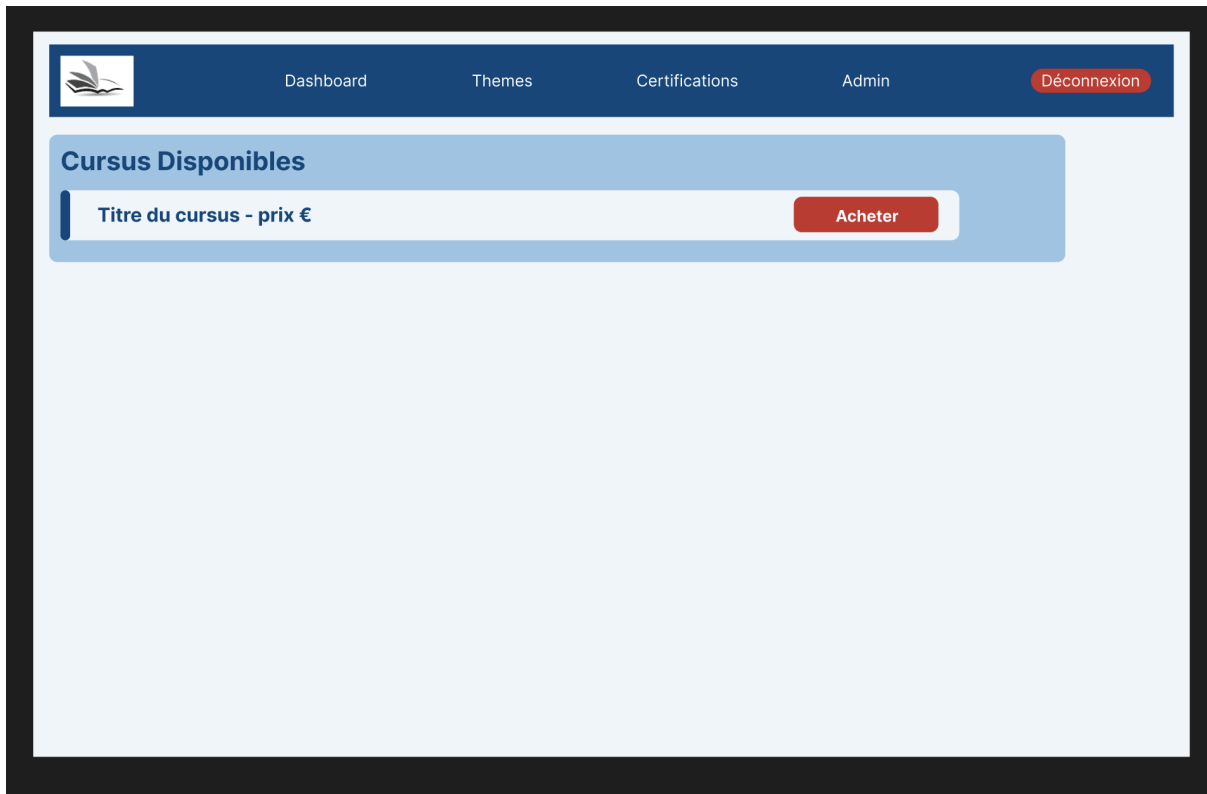
S'inscrire

DashboardThemesCertificationsAdminDéconnexion

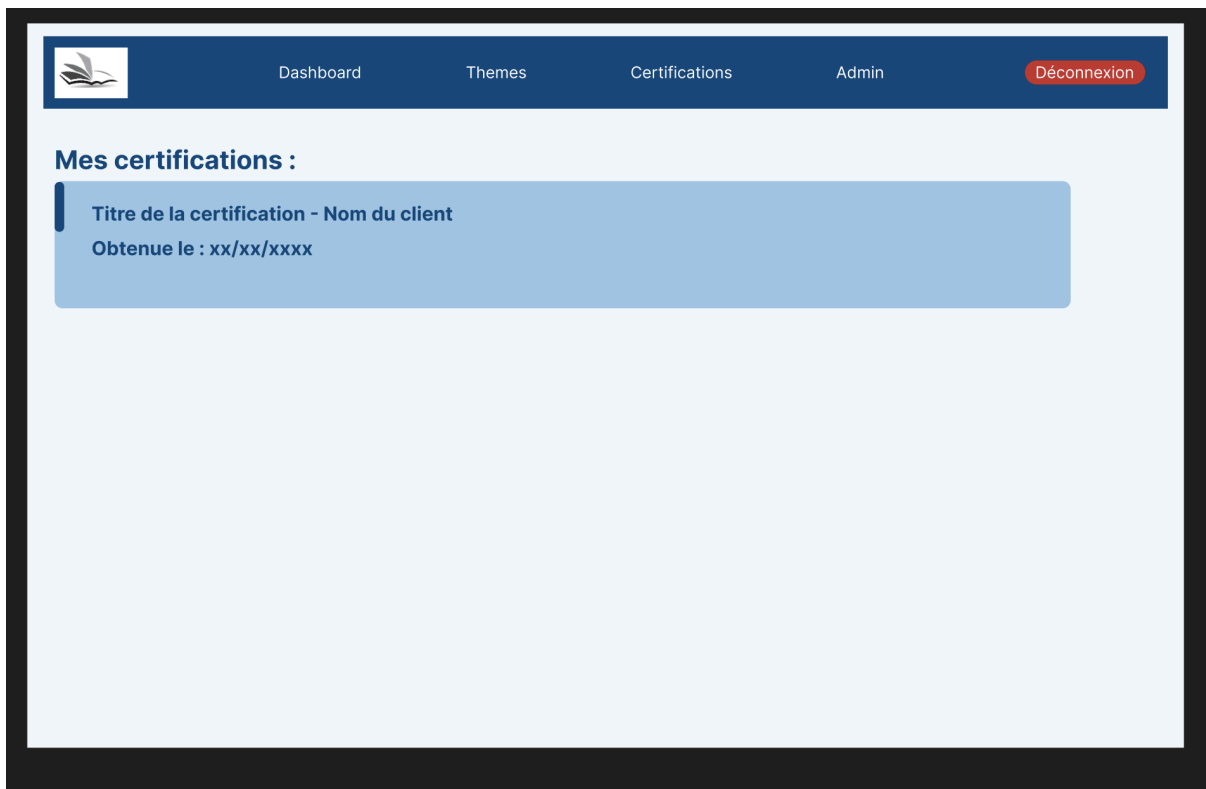
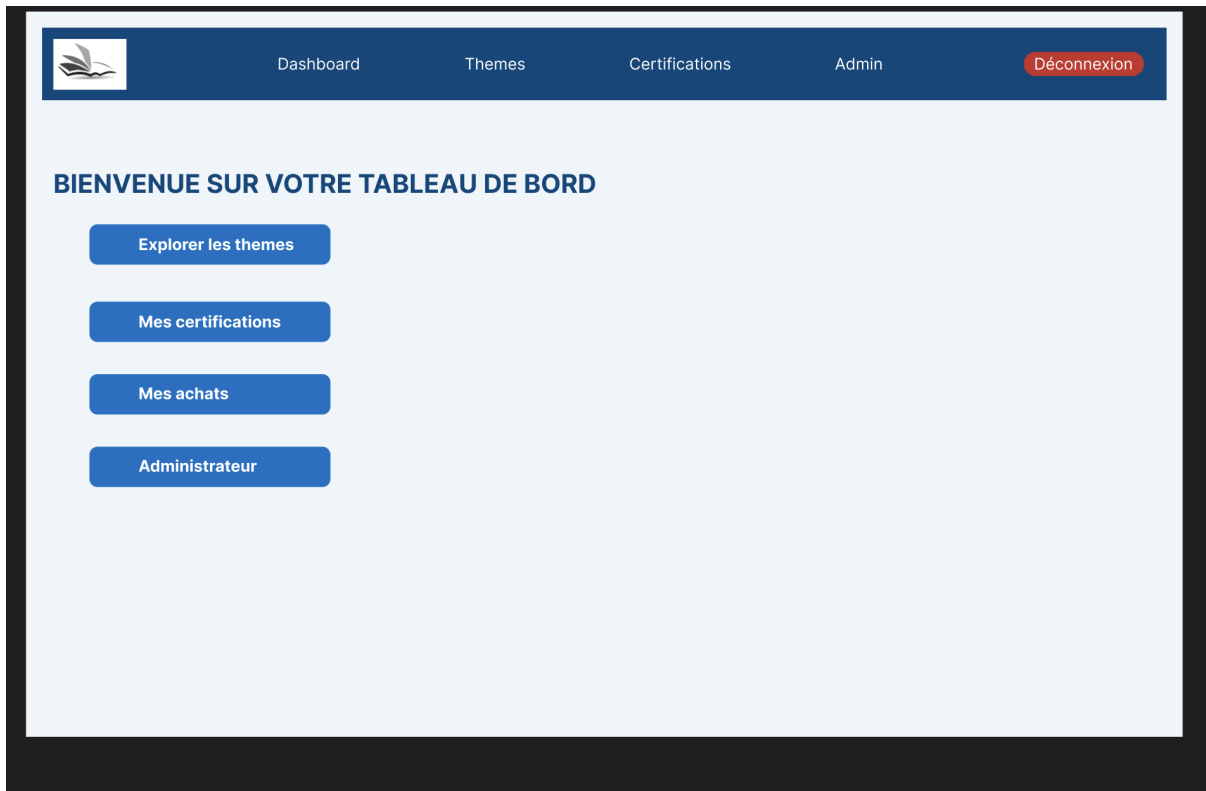
Thèmes de formation

MusiqueInformatiqueJardinageCuisine


Annexe 3 Maquettes Coursus et Leçons



Annexe 4 Maquettes Certifications et tableau de bord



Annexe 5 Maquette mes achats

[Dashboard](#)[Themes](#)[Certifications](#)[Admin](#)[Déconnexion](#)

Mes cursus achetés

Titre du cursus

Valider

Titre du cursus

Valider

Mes leçons achetées

Titre de la leçon

Valider

Description

Vidéo


Titre de la leçon

Valider

Description

Vidéo

Annexe 6 Maquette Interface Admi

[Dashboard](#)[Themes](#)[Certifications](#)[Admin](#)[Déconnexion](#)

Interface d'administration

Thèmes

[Supprimer](#)

[Supprimer](#)

[Ajouter](#)

[Modifier](#)

Cursus

[Supprimer](#)

[Supprimer](#)

[Ajouter](#)

[Modifier](#)

Leçons

[Supprimer](#)

[Supprimer](#)

[Ajouter](#)

[Modifier](#)

Utilisateurs

[Supprimer](#)

[Supprimer](#)

[Ajouter](#)

[Modifier](#)

Achats

Nom user	Titre leçon/cursus	
Nom user	Titre leçon/cursus	Supprimer
Nom user	Titre leçon/cursus	Supprimer
Nom user	Titre leçon/cursus	Supprimer