

# Projet :

## Créer le site



# **KNOWLEDGE**

## **Bienvenue chez KnowledgeLearning**

## Sommaire

Introduction.....	4
Compétences du référentiel couvertes par le projet.....	4
Résumé du projet.....	5
Le projet.....	6
Cahier des charges.....	6
Contraintes et Livrables.....	7
Environnement humain et technique.....	9
Environnement humain.....	9
Environnement technique.....	9
La réalisation du projet back-end.....	11
Modélisation et gestion de la base de données.....	11
API REST et logique métier.....	12
Sécurité.....	13
La réalisation du projet front-end.....	16
Maquettes et conception UX/UI.....	16
Schéma de navigation.....	19
Développement et intégration.....	19
Jeu d'essai représentatif.....	22
Objectifs du test.....	22
Données en entrée.....	22
Données attendues.....	25
Données obtenues.....	27
Analyse des écarts.....	27
Conclusion.....	27
Tests Unitaires.....	28
Veille technologique et sécurité.....	30
Méthodologie de veille.....	30
Vulnérabilités identifiées.....	30
Correctifs et bonnes pratiques mises en place.....	30
Résultats de la veille.....	31
Conclusion.....	31
Conclusion.....	32

Annexes.....	33
Documentation.....	33
Lien vers le GitHub du projet.....	44
ReadMe du projet.....	44

## Introduction

### ***Compétences du référentiel couvertes par le projet***

#### **Développer la partie front-end d'une application web ou web mobile sécurisée**

1. Installer et configurer son environnement de travail en fonction du projet web ou web mobile
2. Maquetter des interfaces utilisateur web ou web mobile
3. Réaliser des interfaces utilisateur statiques web ou web mobile
4. Développer la partie dynamique des interfaces utilisateur web ou web mobile

L'application est un site web disponible en local et en ligne. L'affichage des pages est géré par **Vue.js**, un framework JavaScript permettant de construire des interfaces dynamiques et réactives. Grâce à ses composants modulaires et à la gestion native du DOM virtuel, l'interface s'adapte facilement aux différentes résolutions d'écran. Pour concevoir les maquettes et définir l'ergonomie du site, j'ai utilisé le logiciel **Figma**.

#### **Développer la partie back-end d'une application web ou web mobile sécurisée**

5. Mettre en place une base de données relationnelle
6. Développer des composants d'accès aux données SQL et NoSQL
7. Développer des composants métier côté serveur

Les routes et l'ensemble des fonctionnalités de l'application ont été développées en **JavaScript** à l'aide du framework **Express.js**. Celui-ci fournit une structure légère et flexible permettant de gérer facilement les requêtes HTTP et la création d'API REST. Pour la base de données, j'ai utilisé **Mongoose**, un ODM (Object Data Modeling) spécialement conçu pour MongoDB, qui simplifie la définition des schémas et les interactions avec les collections.

L'objectif principal de ce projet est de passer au moins une fois par chaque étape clé de conception d'une application web complète : analyse des besoins, maquettage, intégration, développement.

## Résumé du projet

La société **Knowledge** est une maison d'édition spécialisée dans la conception et la vente de supports de formation, tels que des livres et des kits pédagogiques qui distribue principalement ses produits en librairie.

Afin d'élargir son offre et de s'adapter aux nouveaux usages numériques, **Knowledge souhaite proposer ses formations en ligne** via une plateforme e-learning nommée *Knowledge Learning*. Ce site permettra aux utilisateurs d'apprendre à distance, en toute autonomie. Il offrira à la fois une interface client pour consulter et acheter des contenus de formation, ainsi qu'un espace administrateur pour gérer l'ensemble des données (utilisateurs, contenus, achats). Les clients pourront obtenir des **certifications** après avoir validé l'ensemble des leçons d'un cursus donné.

Dans le cadre de ce projet, ma mission a été de **concevoir le back-end de la plateforme**. Cela a inclus la **modélisation de la base de données**, la **création des API** permettant la gestion des comptes utilisateurs, des formations et des achats, ainsi que la **mise en place de la logique métier** pour la validation des leçons et l'attribution des certifications. J'ai également intégré une solution d'achat avec mode bac à sable afin de simuler les transactions et livrer un prototype fonctionnel.

## Le projet

### Cahier des charges

L'objectif principal de la plateforme **Knowledge Learning** est de proposer un environnement d'apprentissage en ligne intuitif et sécurisé, permettant à la société Knowledge de diversifier son offre de formations. Le cahier des charges a été élaboré à partir des besoins identifiés à la fois côté utilisateurs et côté administrateurs, en intégrant également les contraintes techniques et les objectifs de qualité.

#### Interface client

L'interface dédiée aux apprenants doit leur offrir une navigation simple et fluide. Les utilisateurs doivent pouvoir parcourir les contenus de formation par thèmes, accéder au détail des cursus disponibles et acheter directement des parcours pédagogiques. Une fois inscrits, ils doivent disposer d'un espace personnel leur permettant de suivre leur progression dans les différentes leçons, de visualiser leur avancement global et, à l'issue d'un cursus complet, de générer automatiquement une certification attestant de leur réussite. L'expérience utilisateur doit être pensée pour être accessible aussi bien sur ordinateur que sur mobile, afin de s'adapter aux usages actuels.

#### Interface administrateur

Un espace d'administration est nécessaire pour permettre la gestion de l'ensemble des données de la plateforme. Cet espace doit offrir aux administrateurs la possibilité de créer, modifier et supprimer des utilisateurs, de gérer les contenus pédagogiques (cursus, leçons, thèmes), ainsi que de superviser les commandes et les paiements effectués par les clients. L'interface doit être claire et sécurisée, afin que les administrateurs puissent intervenir efficacement sur la plateforme sans risquer d'erreurs ou de failles.

#### Système de paiement

La plateforme intègre un système d'achat en ligne, basé sur la solution Stripe. Pour la phase de conception et de test, l'API Stripe a été utilisée en mode sandbox afin de simuler des transactions sans impact financier réel. Ce système permet d'assurer la traçabilité des commandes et de sécuriser les échanges lors des paiements.

#### Sécurité

La sécurité des données et des échanges constitue un enjeu central. Les connexions doivent être protégées par un système d'authentification robuste, basé sur l'utilisation de tokens JWT. Les bonnes pratiques en matière de développement web ont été appliquées : validation stricte des données en entrée, gestion des rôles (administrateurs et utilisateurs), chiffrement des mots de passe avec un algorithme de hachage sécurisé, et surveillance des potentielles vulnérabilités liées aux technologies utilisées.

#### Documentation

Enfin, pour garantir la maintenabilité et la compréhension du projet, l'ensemble des routes et fonctionnalités du back-end ont été documentées à l'aide de Swagger. Cette documentation interactive permet non seulement aux développeurs de disposer d'une vue claire des API disponibles, mais également de tester directement les différentes routes via

une interface conviviale.

En résumé, ce cahier des charges définit les besoins essentiels auxquels doit répondre l'application : offrir une expérience fluide et motivante aux utilisateurs, fournir aux administrateurs des outils de gestion fiables et sécurisés, assurer la sécurité des données et des transactions, et maintenir une documentation technique claire pour faciliter l'évolution future du projet.

## ***Contraintes et Livrables***

Le développement de la plateforme **Knowledge Learning** a été mené en tenant compte d'un ensemble de contraintes techniques, organisationnelles et sécuritaires, afin de garantir la fiabilité du prototype livré et de répondre aux besoins exprimés dans le cahier des charges.

### **Contraintes techniques**

Le choix des technologies s'est porté sur une architecture moderne et modulable, permettant d'assurer la maintenabilité du projet. Le back-end a été développé avec Node.js et le framework Express.js, offrant une grande flexibilité pour la création d'API REST et la gestion des routes. Le front-end repose sur Vue.js, un framework JavaScript orienté composants, qui permet de construire une interface réactive et adaptée à différents supports. La base de données a été conçue avec MongoDB, choisie pour sa capacité à gérer efficacement des données non relationnelles et pour sa compatibilité avec l'écosystème Node.js.

Par ailleurs, les transactions en ligne ont été simulées grâce à l'API Stripe, utilisée en mode sandbox afin de tester le processus d'achat en conditions réelles, mais sans risque financier. L'ensemble du projet a été versionné sur GitHub, garantissant un suivi précis des évolutions et une traçabilité des différentes étapes de développement.

### **Contraintes organisationnelles**

Le projet a été réalisé dans un cadre individuel, ce qui impliquait de gérer à la fois les aspects front-end, back-end et base de données. Cette contrainte a nécessité une organisation rigoureuse, avec une planification des tâches et une priorisation des fonctionnalités essentielles à livrer. Le temps disponible a également orienté les choix techniques vers des solutions efficaces et bien documentées, afin de maximiser la productivité tout en assurant un niveau de qualité suffisant.

### **Contraintes de sécurité**

La sécurité des données constitue un enjeu majeur, en particulier pour une plateforme de formation en ligne manipulant des informations personnelles et des données financières. Le projet a donc intégré plusieurs mécanismes de protection : authentification sécurisée avec JWT, validation stricte des données reçues côté serveur, chiffrement des mots de passe avec bcrypt, et gestion différenciée des rôles (utilisateur simple / administrateur). L'utilisation de Stripe, certifié PCI-DSS, permet par ailleurs d'assurer la conformité du module de paiement avec les standards de sécurité en vigueur.

### **Livrables attendus**

À l'issue du projet, les livrables suivants ont été produits :

- Un prototype fonctionnel de la plateforme Knowledge Learning, intégrant l'interface utilisateur et l'espace administrateur.
- Une base de données MongoDB structurée, avec ses schémas conceptuel et physique, ainsi que les scripts nécessaires à son initialisation et à sa mise à jour.
- Une API REST opérationnelle, documentée avec Swagger, permettant d'interagir avec les différentes fonctionnalités de la plateforme (gestion des utilisateurs, des cursus, des leçons et des commandes).
- Une documentation technique comprenant la description des choix technologiques, des fonctionnalités implémentées et des bonnes pratiques de sécurité mises en place.
- Un jeu de tests unitaires et fonctionnels, permettant de vérifier le bon fonctionnement des fonctionnalités critiques telles que l'inscription, l'authentification, l'achat et la validation des leçons.

L'ensemble de ces livrables constitue une base solide pour l'évolution future du projet, que ce soit en enrichissant les fonctionnalités existantes ou en préparant une mise en production réelle de la plateforme.

## Environnement humain et technique

Le projet **Knowledge Learning** a été conçu et développé dans un cadre pédagogique et professionnel visant à démontrer la capacité à mener à bien un développement web complet, du back-end au front-end, en passant par la gestion des données et la sécurité. L'environnement humain et technique constitue un facteur essentiel pour comprendre les conditions de réalisation et les choix opérés.

### ***Environnement humain***

Le projet a été réalisé de manière **individuelle**, ce qui a nécessité de maîtriser un large éventail de compétences couvrant l'ensemble du cycle de développement d'une application web. Cela incluait :

- la modélisation et la conception de la base de données,
- le développement du back-end et la mise en place des API,
- l'intégration du front-end et la création d'interfaces utilisateur,
- la réalisation des tests unitaires et fonctionnels,
- ainsi que la documentation technique du projet.

Cette autonomie complète a renforcé ma capacité à m'organiser et à prioriser les tâches, en adoptant une approche proche de la méthode agile : découpage des fonctionnalités en modules, itérations courtes et tests réguliers. Bien que réalisé seul, ce projet a été pensé de manière à pouvoir être poursuivi dans un cadre collaboratif, grâce à l'utilisation d'outils de versionnement (Git/GitHub) et à une documentation claire facilitant la reprise du code par d'autres développeurs.

### ***Environnement technique***

Le choix de l'environnement technique s'est orienté vers des outils modernes, largement utilisés dans l'industrie et adaptés aux besoins du projet :

- **Back-end :**
  - **Node.js** pour la rapidité d'exécution et la compatibilité avec les architectures asynchrones.
  - **Express.js**, framework minimaliste, facilitant la création d'API RESTful et la gestion des routes.
- **Base de données :**
  - **MongoDB**, base NoSQL permettant une gestion flexible des données et une intégration fluide avec l'écosystème JavaScript.
  - **Mongoose**, ODM (Object Data Modeling) simplifiant la définition des schémas et les interactions avec la base.
- **Front-end :**
  - **Vue.js**, framework JavaScript progressif permettant de construire des interfaces modulaires, réactives et adaptées aux différents supports (desktop, tablette, mobile).

- Intégration du **responsive design** grâce à l'utilisation des technologies CSS modernes (Flexbox, Grid).
- **Sécurité :**
  - Authentification basée sur des **tokens JWT** (JSON Web Tokens).
  - **bcrypt** pour le chiffrement des mots de passe.
  - Validation systématique des données côté serveur pour prévenir les injections et autres vulnérabilités.
  - Intégration du paiement via **Stripe sandbox**, garantissant une sécurité conforme aux normes PCI-DSS.
- **Tests :**
  - **Mocha**, **Chai** et **Sinon** pour mettre en place des tests unitaires et fonctionnels, couvrant notamment l'inscription des utilisateurs, l'authentification et la gestion des transactions.
- **Documentation :**
  - **Swagger** pour générer une documentation interactive et faciliter l'exploration des API.
- **Gestion du projet :**
  - **GitHub** pour le versionnement et le suivi de l'évolution du code.
  - **VS Code** comme environnement de développement intégré (IDE), avec des extensions facilitant le linting, le debugging et la gestion des dépendances.

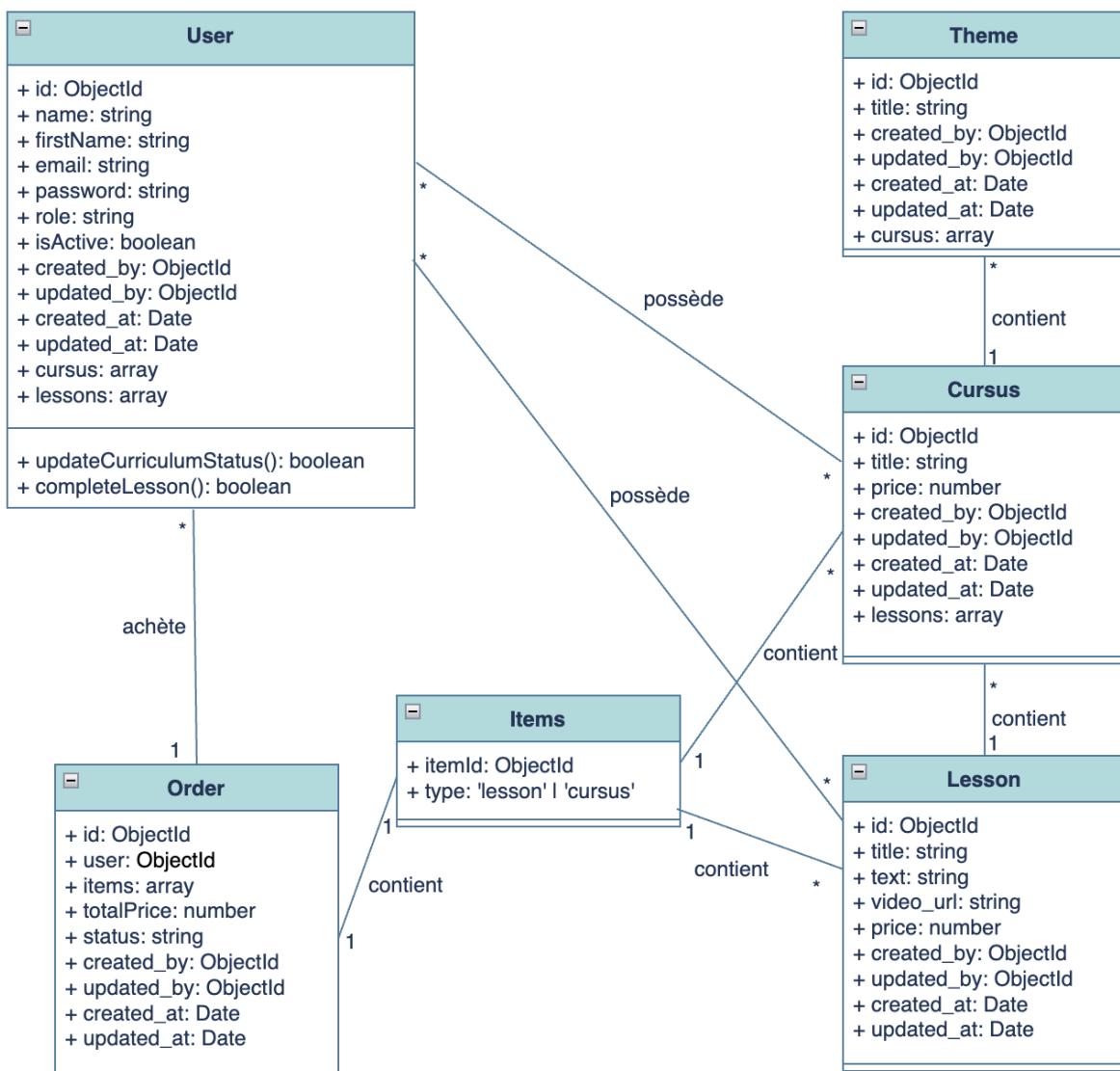
Cet environnement technique a permis de répondre aux exigences du projet tout en adoptant des solutions en adéquation avec les standards actuels du développement web. Il offre une base solide et évolutive, qui pourrait être étendue ou adaptée facilement dans le cadre d'un déploiement en production.

## La réalisation du projet back-end

Le **back-end** de la plateforme Knowledge Learning a été conçu afin de fournir une infrastructure robuste, sécurisée et évolutive permettant de gérer l'ensemble des données et fonctionnalités métier. L'architecture repose sur **Node.js** et le framework **Express.js**, choisis pour leur performance, leur simplicité de mise en œuvre et leur popularité dans le développement d'API modernes.

### **Modélisation et gestion de la base de données**

La base de données repose sur **MongoDB**, une solution NoSQL particulièrement adaptée pour des applications orientées contenu. Les collections principales correspondent aux entités clés de l'application :



- **Users** : informations des utilisateurs (nom, email, mot de passe chiffré, rôle).
- **Cursus** : regroupement de leçons constituant une formation complète.

- **Lessons** : leçons individuelles rattachées à un cursus.
- **Orders** : commandes passées par les utilisateurs, incluant l'état du paiement.
- **Themes** : classification thématique des cursus.

La modélisation a été réalisée avec **Mongoose**, un ODM (Object Data Modeling) qui facilite la définition des schémas et les interactions avec MongoDB. Des relations logiques ont été mises en place pour assurer la cohérence entre les collections (par exemple : un « Cursus » contient une liste de « Lessons », et un « User » peut posséder plusieurs « Orders »).

## **API REST et logique métier**

Le back-end expose une **API RESTful** documentée avec **Swagger**, permettant d'interagir avec l'ensemble des données de la plateforme. Les principales routes gèrent :

- l'inscription et l'authentification des utilisateurs,
- la consultation des cursus, leçons et thèmes,
- l'achat et le suivi des commandes,
- la validation des leçons et l'attribution des certifications,
- les opérations d'administration (création, modification et suppression des données).

La logique métier a été codée directement dans les contrôleurs, avec une séparation claire entre les couches :

- **couche routes** : définition des endpoints,

```
import express from 'express';
const router = express.Router();

import service from '../services/lesson.js';
import middleware from '../middlewares/private.js';

router.get('/', middleware.checkJWT, service.getAll);
```

- **couche contrôleurs** : logique applicative,

```
import Lesson from "../models/lesson.js";

// Get all lessons
const getAll = async (req, res) => {
  try {
    const lessonList = await Lesson.find();
    return res.status(200).json(lessonList);
  } catch (error) {
    return res.status(500).json({ error: error.message });
  }
};
```

- **couche modèles** : interaction avec la base via Mongoose.

```
import mongoose from "mongoose";

const { Schema } = mongoose;

const LessonSchema = new Schema(
  {
    title: { type: String, required: true },
    text: { type: String, required: true },
    video_url: { type: String },
    price: { type: Number, required: true },
    created_by: { type: Schema.Types.ObjectId, ref: "User" },
    updated_by: { type: Schema.Types.ObjectId, ref: "User" },
  },
  { timestamps: true }
);

export default mongoose.model("Lesson", LessonSchema);
```

## Sécurité

Plusieurs mécanismes de sécurité ont été intégrés pour protéger les données et limiter les risques de vulnérabilités :

- **Authentification JWT (JSON Web Token)** : génération d'un token à la connexion, utilisé pour sécuriser les appels API.

```
import jwt from 'jsonwebtoken';

const JWT_SECRET = process.env.JWT_SECRET;

const checkJWT = (req, res, next) => {
  try {
    let token = req.headers['x-access-token'] || req.headers['authorization'];

    // Verification of the format of the token
    if (token && token.startsWith('Bearer ')) {
      token = token.slice(7, token.length);
    }

    if (!token) {
      return res.status(401).json({ message: 'Token required' });
    }

    // Verification and decoding of the token
    jwt.verify(token, JWT_SECRET, (err, decoded) => {
      if (err) {
        return res.status(401).json({
          message:
            err.name === 'TokenExpiredError' ? 'Token expired' : 'Invalid token',
        });
      }

      if (decoded) {
        req.decoded = decoded;

        const expireIn = 24 * 60 * 60; // 24 heures
        const newToken = jwt.sign({ user: decoded.user }, JWT_SECRET, {
          expiresIn: expireIn,
        });
        res.setHeader('Authorization', 'Bearer ' + newToken);

        return next();
      } else {
        return res.status(401).json({ message: 'Invalid token data' });
      }
    });
  } catch (error) {
    console.error('JWT Verification Error:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
};
```

- **Gestion des rôles et permissions** : distinction entre utilisateur simple et administrateur, avec restriction d'accès aux routes sensibles.
  - middleware CheckAdmin pour vérifier si l'utilisateur est un administrateur :

```
const checkAdmin = (req, res, next) => {
  // Verification of the role of the user
  if (req.decoded && req.decoded.role === 'admin') {
    return next();
  } else {
    return res
      .status(403)
      .json({ message: 'Accès interdit. Vous devez être administrateur.' });
  }
};
```

- Utilisation du middleware dans les routes seulement accessibles aux administrateurs :

```
import express from 'express';
const router = express.Router();

import service from '../services/lesson.js';
import middleware from '../middlewares/private.js';

router.post('/add', middleware.checkJWT, middleware.checkAdmin, service.add);
```

- **Chiffrement des mots de passe** : utilisation de **bcrypt** pour stocker les mots de passe de manière sécurisée.

Le chiffrement du mot de passe est géré dans le model « User », par un pre-save hook :

```
import mongoose from "mongoose";
import bcrypt from "bcrypt";

// Middleware to hash the password before saving the user
UserSchema.pre("save", function (next) {
  if (!this.isModified("password")) {
    return next();
  }

  this.password = bcrypt.hashSync(this.password, 10);
  next();
});
```

Dans le service d'authentification de l'utilisateur, on utilise à nouveau bcrypt pour comparer le mot de passe entré à celui enregistré en base de données :

```
// Password validation
const isPasswordValid = await bcrypt.compare(password, user.password);
if (!isPasswordValid) {
  return res.status(403).json({ error: "Invalid credentials" });
}
```

- **Validation des entrées** : contrôles systématiques des données reçues (par exemple, vérification du format email ou de la longueur des mots de passe).

Dans le service « User », on vérifie la longueur et le format du mot de passe :

```
const validatePassword = (password) => {
  const regex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,})$/;
  return regex.test(password);
};
```

Cette fonction est utilisée lors de l'inscription d'un utilisateur :

```
(!validatePassword(password)) {
  return res.status(400).json({
    error:
      "Le mot de passe doit contenir au moins 8 caractères, une majuscule, une minuscule, un chiffre et un caractère spécial.",
  });
};
```

- **Intégration Stripe sandbox** : gestion des transactions de manière sécurisée, en conformité avec la norme PCI-DSS.

```
import Stripe from "stripe";

const stripe = new Stripe(process.env.STRIPE_SECRET_KEY);

export const createCheckoutSession = async (orderId, products, stripeInstance = stripe) => {
  try {
    const lineItems = products.map((product) => {
      if (!product.title || !product.price) {
        throw new Error("Produit invalide : title et price sont requis");
      }

      return {
        price_data: {
          currency: "eur",
          product_data: {
            name: product.title,
          },
          unit_amount: product.price * 100,
        },
        quantity: 1,
      };
    });

    const session = await stripeInstance.checkout.sessions.create({
      payment_method_types: ["card"],
      line_items,
      mode: "payment",
      success_url: `http://localhost:8080/success?orderId=${orderId}`,
      cancel_url: `http://localhost:8080/cancel`,
    });

    return session.url;
  } catch (error) {
    console.error("Erreur Stripe:", error);
    throw new Error("Erreur lors de la création de la session Stripe.");
  }
};

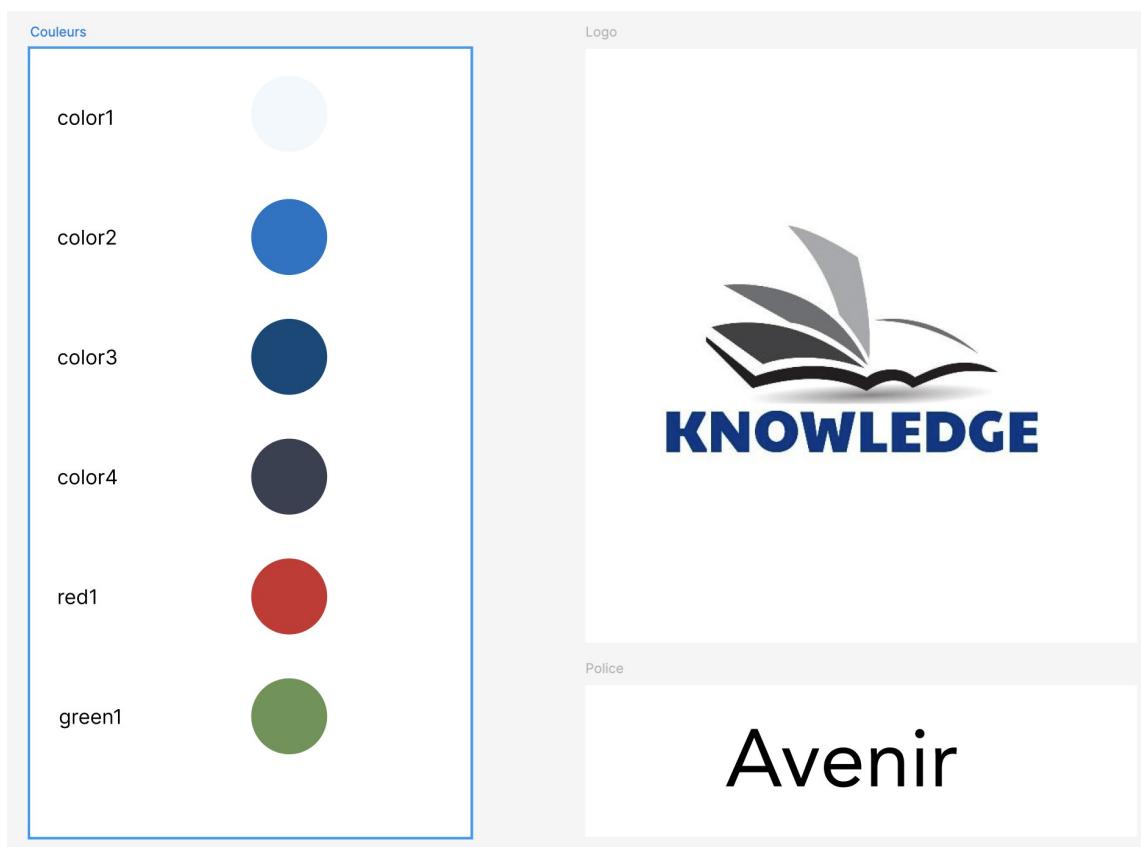
export default createCheckoutSession;
```

## La réalisation du projet front-end

Le développement du **front-end** de la plateforme Knowledge Learning a constitué une étape essentielle afin de garantir une expérience utilisateur fluide, intuitive et adaptée à différents supports (desktop, tablette, mobile). Le choix technologique s'est porté sur **Vue.js**, un framework JavaScript progressif particulièrement adapté à la conception d'interfaces dynamiques et modulaires.

### **Maquettes et conception UX/UI**

Avant toute intégration technique, un travail de **maquettage** a été réalisé avec **Figma**, permettant de définir la structure des interfaces et leur ergonomie, en tenant compte de la charte graphique :



Les maquettes ont été pensées pour mettre en valeur la simplicité d'usage :

- une page d'accueil claire donnant accès aux cursus par thèmes,
- des pages dédiées à l'achat et à la consultation des cours,
- un espace personnel permettant de suivre la progression de l'apprenant,
- un espace administrateur distinct, pour la gestion des utilisateurs, des cursus et des commandes.

Les maquettes ont été déclinées en **version desktop** et en **version mobile**, afin d'assurer une adaptation optimale à toutes les résolutions d'écran. En voici quelques exemples :

## Projet Knowledge Learning - Gwenaëlle Boussès

**Page de connexion**

1280 × 900

**Tableau de bord utilisateur**

1280 × 1048

**Page de présentation des thèmes**

1280 × 1704

Page de connection - ... </>

Bienvenue chez KnowledgeLearning

Veuillez vous connecter pour accéder à votre compte

Connexion

Email

Mot de passe

Se connecter

S'inscrire

390 × 844

Page de présentatio... </>

Les Thèmes des Leçons

Thème 1

Cursus 1 (XX€)

Leçon n°1 : Lorem ipsum (XX€)

Acheter le cours

Leçon n°2 : Lorem ipsum (XX€)

Acheter le cours

Acheter le cursus complet

Cursus 2 (XX€)

Leçon n°1 : Lorem ipsum (XX€)

Acheter le cours

Leçon n°2 : Lorem ipsum (XX€)

Acheter le cours

Acheter le cursus complet

Thème 2

Cursus 1 (XX€)

Leçon n°1 : Lorem ipsum (XX€)

Acheter le cours

Leçon n°2 : Lorem ipsum (XX€)

Acheter le cours

Acheter le cursus complet

Cursus 2 (XX€)

Leçon n°1 : Lorem ipsum (XX€)

Acheter le cours

Leçon n°2 : Lorem ipsum (XX€)

Acheter le cours

Acheter le cursus complet

390 × 1921

Tableau de bord - m... </>

Mes certifications

Voir mes certifications

Mes cours

Mes leçons

Leçon n°1 : Lorem ipsum

Voir la leçon

Cursus complété

Cursus n°1

Leçon n°1 : Lorem ipsum

Voir la leçon

Leçon n°2 : Lorem ipsum

Voir la leçon

390 × 1077

## Schéma de navigation

Un **schéma d'enchaînement des écrans** a été élaboré pour visualiser la navigation entre les différentes interfaces :

- authentification → accueil → choix d'un cursus → achat → consultation des leçons → progression → certification,
- côté administrateur : connexion → tableau de bord → gestion des utilisateurs/contenus → suivi des commandes.

Ce schéma garantit une logique de parcours claire et cohérente pour les deux types d'utilisateurs.

## Développement et intégration

Le front-end a été implémenté en **Vue.js**, en adoptant une architecture basée sur les **composants**. Chaque partie de l'interface (formulaire d'inscription, tableau de progression, liste des cours, panier, etc.) a été développée comme un composant réutilisable, facilitant ainsi la maintenabilité et l'évolution du projet.

Afin de garantir une **organisation claire du projet** et une bonne **séparation des responsabilités**, la structure des fichiers respecte les bonnes pratiques de Vue.js.

L'illustration ci-dessous montre l'arborescence du projet front-end :

```
└─ frontend/kl-frontend
    ├ node_modules
    ├ public
    └─ src
        ├ assets
        ├ components
        ├ router
        ├ services
        └─ store
            └─ views
                └─ App.vue
                ┌── main.js
```

- les **composants** sont regroupés dans un dossier « components/ » et se concentrent sur l'affichage et les interactions utilisateur, comme dans cet exemple :

```
<template>
  <div class="theme-card">
    <h2>{{ theme.title }}</h2>
    <p>{{ theme.description }}</p>

    <!-- Liste des cursus associés -->
    <div v-if="theme.cursus && theme.cursus.length > 0">
      <CursusCard
        v-for="cursus in theme.cursus"
        :key="cursus.id"
        :cursus="cursus"
        @add-to-cart="$emit('add-to-cart', $event)"
      />
    </div>
    <p v-else>Pas de cursus associés à ce thème.</p>
  </div>
</template>

<script>
import CursusCard from "./CursusCard.vue";

export default {
  name: "ThemeCard",
  props: {
    theme: {
      type: Object,
      required: true,
    },
  },
  components: {
    CursusCard,
  },
};
</script>
```

Ce composant gère l'affichage d'un thème et de tous les cursus qui y sont associés

- les **vues** dans « views/ » définissent les pages principales et orchestrent les composants :

```
<template>
  <NavBar />
  <main>
    <div class="themes-page">
      <h1>Les Thèmes des Leçons</h1>

      <div v-if="error" class="error">
        <p>{{ error }}</p>
      </div>

      <div v-if="themes.length === 0" class="loading">
        <p>Chargement des thèmes...</p>
      </div>

      <div v-else>
        <ThemeCard
          v-for="theme in themes"
          :key="theme.id"
          :theme="theme"
          @add-to-cart="handleAddToCart"
        />
      </div>
    </div>
  </main>
</template>
```

Cette view affiche la barre de navigation et les différents thèmes.

- les **services** dans « services/ » gèrent les appels API et l'accès aux données :

```
import axiosInstance from "../services/axios";

export const getThemes = async () => {
  try {
    const response = await axiosInstance.get('http://localhost:3000/themes');
    return response;
  } catch (error) {
    console.error("Erreur lors de la récupération des thèmes", error);
    throw error;
  }
};
```

Ce service récupère tous les thèmes présents dans la base de données en utilisant une requête via Axios.

- les **routes** dans « router/ » assurent la navigation entre les différentes sections de l'application :

```
{
  path: '/themes',
  name: 'ThemesPage',
  component: Themes,
},
```

Cette route permet d'afficher la view « ThemesPage »

L'adaptation responsive a été assurée grâce à l'utilisation des propriétés **CSS modernes (Flexbox et Grid Layout)**, qui permettent de garantir un rendu homogène sur différents supports.

## Jeu d'essai représentatif

Afin de valider le bon fonctionnement de l'application **Knowledge Learning**, un jeu d'essai a été élaboré sur l'une des fonctionnalités les plus représentatives : **le parcours d'inscription d'un utilisateur suivi de l'achat d'un cursus de formation**. Ce scénario couvre à la fois l'interaction front-end et back-end, la gestion des données et l'intégration d'un service externe (Stripe).

### Objectifs du test

L'objectif est de vérifier :

- la bonne création d'un compte utilisateur avec enregistrement sécurisé des données et confirmation par mail,
- l'authentification de l'utilisateur et l'obtention d'un token JWT valide,
- l'ajout d'un cursus au panier et la création d'une commande,
- la simulation d'un paiement via Stripe en mode sandbox,
- la mise à jour de la base de données (commande validée, cursus disponible pour l'utilisateur).

### Données en entrée

#### 1. Formulaire d'inscription :

- Nom : *Dupont*
- Prénom : *Jean*
- Email : *dupont@example.com*
- Mot de passe : *Test1234!*

The screenshot shows a web page with a dark blue header. On the left is a logo with a stylized book icon and the word "KNOWLEDGE". On the right are links for "Nos cours" and "Se connecter". The main content area has a light blue background. At the top, it says "Créer un compte". Below that is a form with five input fields: "Nom" (Dupont), "Prénom" (Jean), "Email" (dupont@example.com), "Mot de passe" (redacted), and "Confirmer le mot de passe" (redacted). At the bottom of the form is a blue "S'inscrire" button.

#### 2. Validation du compte par mail

Utilisation de maildev pour le test

The screenshot shows a MailDev interface with a blue header bar. On the left, there's a search bar and a sidebar with the subject "Confirmation de votre inscription". The main content area displays an email message with the following details:

**Confirmation de votre inscription**  
From: knowledgelearning72@gmail.com  
To: dupont@example.com  
2025-09-27 17:34:46 (+0200)

The body of the email contains a confirmation message: "Merci de vous être inscrit. Cliquez sur le lien suivant pour confirmer votre compte : [Confirmer mon compte](#)".

### 3. Connexion :

- Identifiants : *dupont@example.com / Test1234!*

The screenshot shows the homepage of the KnowledgeLearning website. At the top, there's a dark header with the "KNOWLEDGE" logo on the left, "Nos cours" in the middle, and "Se connecter" on the right. The main content area features the "KNOWLEDGE" logo and the text "Bienvenue chez KnowledgeLearning". Below this, a message says "Veuillez vous connecter pour accéder à votre compte". A central "Connexion" form is displayed, containing fields for "Email" (with the value "dupont@example.com") and "Mot de passe" (with the value "....."). A blue "Se connecter" button is at the bottom of the form.

### 4. Achat d'un cursus :

- Cursus sélectionné : *Développement Web – Niveau débutant*
- Moyen de paiement : carte bancaire fictive fournie par Stripe (4242 4242 4242 4242).

Page de sélection des thèmes :

## Projet Knowledge Learning - Gwenaëlle Boussès

The screenshot shows a web browser window for 'localhost:8080/themes'. The header includes the 'KNOWLEDGE' logo, navigation links for 'Nos cours', 'Mon tableau de bord', 'Mon panier', and 'Se déconnecter', and a 'TEST MODE' indicator. The main content area displays three categories: 'Informatique' (with 'Cursus d'initiation au développement web (60 €)'), 'Musique' (with 'Cursus d'initiation au piano (50 €)'), and 'Jardinage' (with 'Leçon n°1 : Les outils du jardinier (16 €)' and 'Leçon n°2 : Jardiner avec la lune (16 €)'). Each category has three 'Acheter le cours' buttons. A success message box is visible on the right side of the 'Musique' section, stating: 'La leçon "Cursus d'initiation au développement web" a été ajoutée au panier'.

## Page de récapitulatif de la commande :

The screenshot shows a 'Récapitulatif de votre commande' page. It displays a single item: 'Commande n° 68d80da51db73c362938c755' for 'Cursus d'initiation au développement web - 60 €'. The total amount is 'Total : 60 €' and there is a 'Payer' button.

## Page de session Stripe :

This screenshot shows a payment session on the Stripe platform. At the top left, it says 'Stubborn TEST MODE'. The main content area shows a summary: 'Cursus d'initiation au développement web' and a large bolded amount of '60,00 €'. To the right, there is a green button labeled 'Payer avec ⚡ link'. Below this are fields for 'E-mail' (containing 'dupont@example.com') and 'Moyen de paiement'. Under 'Informations de la carte', a card number '4242 4242 4242 4242' is shown along with expiration '06 / 26' and CVV '555'. Under 'Nom du titulaire de la carte', the name 'Jean Dupont' is entered. Under 'Pays ou région', 'France' is selected. A checkbox for 'Enregistrer mes informations pour régler plus rapidement' is checked, with a note explaining it allows payment via Link. At the bottom is a large blue 'Payer' button.

## Données attendues

- Création du compte utilisateur dans la collection **Users** avec un mot de passe chiffré.

```
_id: ObjectId('68d804161db73c362938c6e0')
name: "Dupont"
email: "dupont@example.com"
password: "$2b$10$LWpGRxk6j0mrAh8WD81sf0vQ1TucsAyVKuB0rE0DaWp8kVLWfLFZm"
isActive: true
role: "user"
cursus: Array (empty)
lessons: Array (empty)
createdAt: 2025-09-27T15:34:46.639+00:00
updatedAt: 2025-09-27T16:19:17.742+00:00
__v: 3
```

- Retour d'un **token JWT** valide après la connexion.

```
GET /users/confirm/eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJcI2VySWQiOiI20GQ4MDQxNjFkYjczYzM2Mjkz0GM2ZTAiLCJpYXQiOjE3NTg50DcyODYsImV4cCI6MTc10TA3MzY4Nn0.JYKorLdS9dKYKIBxj-sh9h9cIu87FNzVrIUCDJ2BLN4 200 56.3
50 ms - 83
```

- Création d'une entrée dans la collection **Orders** avec l'état *pending* avant paiement.

```
_id: ObjectId('68d80e531db73c362938c76b')
user: ObjectId('68d804161db73c362938c6e0')
items: Array (1)
  0: Object
    type: "cursus"
    itemId: ObjectId('676003071db79502df9a448d')
    _id: ObjectId('68d80e531db73c362938c76c')
    totalPrice: 60
    status: "pending"
    created_by: ObjectId('68d804161db73c362938c6e0')
    createdAt: 2025-09-27T16:18:27.242+00:00
    updatedAt: 2025-09-27T16:18:27.242+00:00
__v: 0
```

- Validation de la commande après le paiement sandbox → mise à jour de l'état en *completed*.

```
_id: ObjectId('68d80e531db73c362938c76b')
user: ObjectId('68d804161db73c362938c6e0')
items: Array (1)
  0: Object
    type: "cursus"
    itemId: ObjectId('676003071db79502df9a448d')
    _id: ObjectId('68d80e531db73c362938c76c')
    totalPrice: 60
    status: "completed"
    created_by: ObjectId('68d804161db73c362938c6e0')
    createdAt: 2025-09-27T16:18:27.242+00:00
    updatedAt: 2025-09-27T16:19:17.632+00:00
__v: 0
updated_by: ObjectId('68d804161db73c362938c6e0')
```

- Ajout automatique du cursus acheté dans l'espace utilisateur.

Dans la base de données :

```

_id: ObjectId('68d804161db73c362938c6e0')
name : "Dupont"
email : "dupont@example.com"
password : "$2b$10$LWpGRxk6j0mrAh8WD81sf0vQ1TucsAyVKuB0rE0DaWp8kVLWfLFZm"
isActive : true
role : "user"
cursus : Array (1)
  ▾ 0: Object
    id : ObjectId('676003071db79502df9a448d')
    ▾ data : Object
      _id : "676003071db79502df9a448d"
      title : "Cursus d'initiation au développement web"
      ▾ lessons : Array (2)
        ▾ 0: Object
          _id : "676002c81db79502df9a4488"
          title : "Leçon n°1 : Les langages Html et CSS"
          text : "Lorem ipsum"
          price : 32
          __v : 0
        ▾ 1: Object
          _id : "676002d21db79502df9a448a"
          title : "Leçon n°2 : Dynamiser votre site avec Javascript"
          text : "Lorem ipsum"
          price : 32
          __v : 0
        ▾ lessons : Array (2)
          ▾ 0: Object
            _id : ObjectId('676002c81db79502df9a4488')
            isCompleted : false
          ▾ 1: Object
            _id : ObjectId('676002d21db79502df9a448a')
            isCompleted : false
            isCompleted : false
            _id : ObjectId('68d80dd01db73c362938c762')
        ▾ lessons : Array (empty)
      createdAt : 2025-09-27T15:34:46.639+00:00
      updatedAt : 2025-09-27T16:16:16.151+00:00
      __v : 2
    
```

Sur le tableau de bord de l'utilisateur :

The screenshot shows a user dashboard with the following sections:

- Mes certifications**: Displays a message "Vous n'avez pas encore de certification".
- Mes cours**: Displays a message "Vous n'avez pas encore de leçons".
- Mes cursus**: Shows a card for the "Cursus d'initiation au développement web" with two lessons listed:
  - Leçon n°1 : Les langages Html et CSS
  - Leçon n°2 : Dynamiser votre site avec Javascript
 Each lesson has a "Voir la leçon" button next to it.

## **Données obtenues**

Les tests ont confirmé que l'ensemble des étapes étaient fonctionnelles :

- Le compte utilisateur est correctement créé et stocké en base avec un mot de passe haché via bcrypt.
- Le login retourne bien un token JWT, utilisé pour sécuriser les appels suivants.
- Une commande est générée avec un identifiant unique, puis validée après le paiement.
- L'utilisateur peut consulter immédiatement les leçons associées au cursus acheté.

## **Analyse des écarts**

Lors des premiers tests, un problème a été rencontré sur la validation du mot de passe : les règles de complexité n'étaient pas suffisamment strictes (acceptation de mots de passe trop simples). Ce point a été corrigé en renforçant les règles de validation côté serveur (longueur minimale, présence de majuscules, minuscules, chiffres et caractères spéciaux).

Après correction, aucun écart n'a été constaté entre les données attendues et les données obtenues.

## **Conclusion**

Ce jeu d'essai a permis de valider une chaîne fonctionnelle complète, couvrant l'authentification, la gestion des commandes et l'intégration d'un service externe de paiement. Il constitue un test représentatif de l'application, démontrant la robustesse des fonctionnalités essentielles et la cohérence des données entre le front-end, le back-end et la base de données.

## Tests Unitaires

Les test unitaires ont été mis en place en utilisant Mocha, Chai pour simuler les appels API, et Sinon pour les tests impliquant l'envoi de mail lors de l'inscription et le paiement par Stripe.

### Exemple de test : user.test.mjs

```
import { describe, it, before, after, beforeEach, afterEach } from "mocha";
import sinon from "sinon";
import request from "supertest";
import { expect } from "chai";
import app from "../app.js";
import mongoose from "mongoose";
import User from "../models/user.js";
import nodemailer from "nodemailer";
import bcrypt from "bcrypt";

describe("User Authentication", () => {
  let findOneStub, compareStub;

  beforeEach(() => {
    findOneStub = sinon.stub(User, "findOne").returns({
      _id: "user_id",
      email: "testuser@example.com",
      password: bcrypt.hash("Password123!", 10),
      role: "user",
      isActive: true,
    });
    compareStub = sinon.stub(bcrypt, "compare").resolves(true);
  });

  afterEach(() => {
    sinon.restore();
  });

  it("should authenticate the user and return a token", async () => {
    const res = await agent
      .post("/users/authenticate")
      .set("X-XSRF-TOKEN", csrfToken)
      .send({
        email: "testuser@example.com",
        password: "Password123!",
      });
  });
});
```

```

    expect(res.status).to.equal(200);
    expect(res.body.message).to.equal("Authenticate_succeed");
    expect(res.body).to.have.property("token");
});

it("should return 403 if password is invalid", async () => {
    compareStub.resolves(false);

    const res = await agent
        .post("/users/authenticate")
        .set("X-XSRF-TOKEN", csrfToken)
        .send({
            email: "testuser@example.com",
            password: "wrongpassword"
        });
        (property) Chai.Assertion.equal: Chai.Equal
        (value: any, message?: string) => Chai.Assertion
    expect(res.status).to.equal(403);
    expect(res.body.error).to.equal("Invalid credentials");
});

it("should return 404 if user not found", async () => {
    findOneStub.returns(null);

    const res = await agent
    any post("/users/authenticate")
        .set("X-XSRF-TOKEN", csrfToken)
        .send({
            email: "nonexistent@example.com",
            password: "Password123!"
        });

    expect(res.status).to.equal(404);
    expect(res.body.error).to.equal("User not found");
});
});

```

À l'exécution des tests, il passent tous avec succès.

```

User Services
  User Authentication
    GET /csrf-token 200 2.450 ms - 52
    Connected to MongoDB
    POST /users/authenticate 200 217.673 ms - 488
    POST /users/authenticate 200 82.234 ms - 496
    POST /users/authenticate 200 1.055 ms - 353
      ✓ should authenticate the user and return a token
    GET /csrf-token 200 0.324 ms - 52
    POST /users/authenticate 200 84.205 ms - 488
    POST /users/authenticate 200 93.652 ms - 496
    POST /users/authenticate 403 0.402 ms - 31
      ✓ should return 403 if password is invalid
    GET /csrf-token 200 0.336 ms - 52
    POST /users/authenticate 200 84.591 ms - 488
    POST /users/authenticate 200 82.029 ms - 496
    POST /users/authenticate 404 0.520 ms - 26
      ✓ should return 404 if user not found

  3 passing (2s)

```

## Veille technologique et sécurité

La sécurité étant un enjeu majeur pour toute application web manipulant des données personnelles et financières, une veille technologique régulière a été réalisée durant le développement du projet **Knowledge Learning**. L'objectif était d'identifier les vulnérabilités potentielles liées aux technologies utilisées (Node.js, Express.js, MongoDB, Vue.js, Stripe) et de mettre en place des correctifs adaptés afin de renforcer la robustesse de la plateforme.

### Méthodologie de veille

La veille a été effectuée à partir de plusieurs sources fiables :

- **Bases de données de vulnérabilités** : OWASP Top 10, CVE (Common Vulnerabilities and Exposures).
- **Blogs et documentations officielles** : Node.js Security Releases, MongoDB Security Documentation, Stripe API updates.
- **Communautés de développeurs** : forums spécialisés, GitHub issues, Stack Overflow.

Cette veille m'a permis d'anticiper les risques et d'appliquer les bonnes pratiques de sécurité recommandées.

### Vulnérabilités identifiées

- **Injection NoSQL (MongoDB)** : risque d'injections dans les requêtes si les données utilisateur ne sont pas correctement validées.
- **Failles XSS (Vue.js)** : possibilité d'injection de scripts malveillants dans les champs de saisie affichés sans échappement.
- **Vol de sessions (JWT)** : risque si les tokens sont mal protégés ou stockés de manière inadaptée dans le navigateur.
- **Failles liées au paiement en ligne (Stripe)** : nécessité de respecter les standards PCI-DSS et d'empêcher toute interception des données sensibles.

### Correctifs et bonnes pratiques mises en place

- **Validation et assainissement des données** : mise en place de contrôles stricts côté serveur (express-validator) pour vérifier la conformité des données (format d'email, mot de passe, identifiants).
- **Protection contre les injections NoSQL** : utilisation de requêtes préparées avec Mongoose, limitant la possibilité de manipulation malveillante des champs.
- **Sécurisation de l'authentification** :
  - chiffrement des mots de passe avec **bcrypt**,
  - génération de tokens JWT avec une durée de vie limitée,
  - stockage sécurisé des tokens côté client (en-têtes HTTP, jamais dans le localStorage).
- **Protection contre le XSS** : échappement automatique des données affichées dans

les templates Vue.js, ajout de règles Content Security Policy (CSP).

- **Sécurité des paiements** : utilisation exclusive de l'API Stripe en mode sandbox pour les tests, garantissant que les données bancaires ne transitent jamais par l'application. Stripe gère lui-même le chiffrement et la conformité PCI-DSS.

## **Résultats de la veille**

Grâce à cette veille, certaines failles potentielles ont pu être corrigées dès la phase de développement. Par exemple, des règles de validation plus strictes ont été ajoutées sur les champs de saisie utilisateur, et l'authentification a été renforcée pour éviter le vol de sessions. De plus, l'utilisation de Stripe a permis de déléguer la gestion sensible des données bancaires à un service certifié et fiable.

## **Conclusion**

La veille technologique et sécuritaire a joué un rôle essentiel dans la conception de Knowledge Learning. Elle a permis non seulement d'anticiper les vulnérabilités propres aux technologies employées, mais aussi d'appliquer des correctifs adaptés pour assurer un haut niveau de protection des utilisateurs et de leurs données. Ce travail constitue un socle solide pour la mise en production future de l'application et garantit sa conformité aux standards actuels de sécurité.

## Conclusion

Ce projet m'a permis de mettre en pratique l'ensemble des compétences acquises au cours de ma formation, notamment en développement front-end et back-end, gestion de bases de données et intégration d'outils externes. J'ai pu concevoir et développer une application fonctionnelle répondant aux besoins définis dans le cahier des charges, tout en respectant les contraintes techniques et les bonnes pratiques de développement.

Au-delà des aspects techniques, ce projet m'a également permis de renforcer mes compétences en organisation et en gestion de projet. J'ai appris à anticiper les difficultés, à proposer des solutions adaptées et à optimiser le workflow de développement pour gagner en efficacité.

Pour l'avenir, ce projet ouvre la voie à plusieurs axes d'évolution : l'amélioration des fonctionnalités existantes, l'optimisation de la performance de l'application, et l'exploration de nouvelles technologies pour enrichir l'expérience utilisateur. Il constitue ainsi une base solide sur laquelle je pourrai continuer à développer mes compétences et contribuer à des projets plus complexes.

## Annexes

### Documentation

La documentation de l'API a été faite à l'aide de Swagger.

#### Cursus

GET /cursus/ get all cursus

return all cursus

Parameters

No parameters

Responses

Code	Description	Links
200	Cursus list found successfully	No links
401	Unauthorized - Invalid or missing token	No links
500	server internal error	No links

GET /cursus/{id} Get a specific cursus by ID

return a specific cursus by ID

Parameters

Name Description

<b>id</b> * required	Cursus ID
string (path)	id

Responses

Code	Description	Links
200	Cursus found successfully	No links
401	Unauthorized - Invalid or missing token	No links
404	Cursus not found	No links
500	Server internal error	No links

## Projet Knowledge Learning - Gwenaëlle Boussès

**PUT** /cursus/{id} Update a cursus

Update an existing cursus. Requires admin permissions.

**Parameters**

Name	Description
<b>id</b> * required string (path)	Cursus ID id

**Request body** required

application/json

Example Value | Schema

```
{  
  "name": "Cursus Mis à jour",  
  "description": "Nouvelle description du cursus"  
}
```

**Responses**

Code	Description	Links
200	Cursus updated successfully	No links
400	Invalid data	No links
401	Unauthorized - Invalid or missing token	No links
403	Unauthorized - Admin permission required	No links
404	Cursus not found	No links
500	Server internal error	No links

**DELETE** /cursus/{id} Delete a cursus

Delete a specific cursus by ID. Requires admin permissions.

**Parameters**

Name	Description
<b>id</b> * required string (path)	Cursus ID id

**Responses**

Code	Description	Links
200	Cursus deleted successfully	No links
401	Unauthorized - Invalid or missing token	No links
403	Unauthorized - Admin permission required	No links
404	Cursus not found	No links
500	Server internal error	No links

## Projet Knowledge Learning - Gwenaëlle Boussès

**POST** /cursus/add Add a new cursus

Create a new cursus. Requires admin permissions.

**Parameters**

No parameters

**Request body** required

application/json

Example Value | Schema

```
{ "name": "Informatique", "description": "Formation avancée en informatique" }
```

**Responses**

Code	Description	Links
201	Cursus created successfully	No links
400	Invalid data	No links
401	Unauthorized - Invalid or missing token	No links
403	Unauthorized - Admin permission required	No links
500	Server internal error	No links

## Lessons

**GET** /lesson/ Get all lessons

Return all lessons. Requires authentication via JWT.

**Parameters**

No parameters

**Responses**

Code	Description	Links
200	Lessons list found successfully	No links
401	Unauthorized - Invalid or missing token	No links
500	Server internal error	No links

**GET** /lesson/{id} Get a specific lesson by ID

return a specific lesson by ID. Requires authentication via JWT.

**Parameters**

Name Description

**id** \* required ID de la leçon à récupérer  
string  
(path)

Try it out

**Responses**

Code	Description	Links
200	Lesson found successfully	No links
401	Unauthorized - Invalid or missing token	No links
404	Lesson not found	No links
500	Server internal error	No links

## Projet Knowledge Learning - Gwenaëlle Boussès

**PUT** /lesson/{id} Update a lesson  

Update an existing lesson. Requires authentication via JWT and admin rights.

**Parameters** 

Name	Description
<b>id</b> * required string (path)	Lesson ID to update id

**Request body** required 

Example Value | Schema

```
{ "title": "Leçon mise à jour" }
```

**Responses**

Code	Description	Links
200	Lesson updated successfully	No links
400	Invalid data	No links
401	Unauthorized - Invalid or missing token	No links
403	Unauthorized - Admin rights required	No links
404	Lesson not found	No links
500	Server internal error	No links

**DELETE** /lesson/{id} Delete a lesson  

Delete an existing lesson. Requires authentication via JWT and admin rights.

**Parameters** 

Name	Description
<b>id</b> * required string (path)	Lesson ID to delete id

**Responses**

Code	Description	Links
200	Lesson deleted successfully	No links
401	Unauthorized - Invalid or missing token	No links
403	Unauthorized - Admin rights required	No links
404	Lesson not found	No links
500	Server internal error	No links

## Projet Knowledge Learning - Gwenaëlle Boussès

**POST** /lesson/add Add a new lesson

Create a new lesson. Requires authentication via JWT and admin rights.

**Parameters**

No parameters

**Request body** required

application/json

Example Value | Schema

```
[{"title": "Nouvelle leçon"}]
```

**Responses**

Code	Description	Links
201	Lesson created successfully	No links
400	Invalid data	No links
401	Unauthorized - Invalid or missing token	No links
403	Unauthorized - Admin rights required	No links
500	Server internal error	No links

**PATCH** /users/lessons/{lessonId}/complete Mark a lesson as completed

Mark a specific lesson as completed for the authenticated user.

**Parameters**

Name Description

**lessonId** \* required Lesson ID to mark as completed  
string (path) lessonId

**Responses**

Code	Description	Links
200	Lesson marked as completed successfully	No links
401	Unauthorized - Invalid or missing token	No links
404	Lesson not found	No links
500	Server internal error	No links

## Orders

**POST** /orders/ Create a new order

create a new order for the authenticated user.

**Parameters**

No parameters

**Request body** required

application/json

Example Value | Schema

```
{
  "items": [
    {
      "productId": "123",
      "quantity": 2
    }
  ]
}
```

**Responses**

Code	Description	Links
201	Order created successfully	No links
400	Invalid data	No links
401	Unauthorized - Invalid or missing token	No links
500	Server internal error	No links

**GET** /orders/user/{userId} Get all orders by user

Return all orders for a specific user.

**Parameters**

Try it out

Name	Description
userId * required	User ID to retrieve orders string (path)

**Responses**

Code	Description	Links
200	Orders found successfully	No links
400	Invalid user ID	No links
404	No orders found	No links
500	Server internal error	No links

**GET** /orders/{orderId} Get a specific order by ID

Return a specific order by ID.

**Parameters**

Try it out

Name	Description
orderId * required	Order ID to retrieve string (path)

**Responses**

Code	Description	Links
200	Order found successfully	No links
400	Invalid order ID	No links
404	Order not found	No links
500	Server internal error	No links

## Projet Knowledge Learning - Gwenaëlle Boussès

PUT /orders/{orderId} Update an existing order status

Update an existing order status.

Parameters

Name	Description
orderId <small>required</small>	Order ID to update
string	(path)
orderId	

Request body required

application/json

Example Value | Schema

```
{ "status": "completed" }
```

Responses

Code	Description	Links
200	Order status updated successfully	No links
400	Invalid data	No links
404	Order not found	No links
500	Server internal error	No links

## Themes

GET /theme/ Get all themes

Return all themes. Requires authentication via JWT.

Parameters

No parameters	
---------------	--

Responses

Code	Description	Links
200	Themes list found successfully	No links
500	Server internal error	No links

GET /theme/{id} Get a specific theme by ID

Return a specific theme by ID. Requires authentication via JWT.

Parameters

Name	Description
id <small>required</small>	Theme ID to retrieve
string	(path)
id	

Responses

Code	Description	Links
200	Theme found successfully	No links
401	Unauthorized - Invalid or missing token	No links
404	Theme not found	No links
500	Server internal error	No links

## Projet Knowledge Learning - Gwenaëlle Boussès

PUT /theme/{id} Update a theme

Update an existing theme. Requires authentication via JWT and admin rights.

Parameters

Name	Description
<b>id</b> * required	Theme ID to update
string (path)	id

Request body required

application/json

Example Value | Schema

```
{ "name": "Développement Mobile" }
```

Responses

Code	Description	Links
200	Theme updated successfully	No links
400	Invalid data	No links
401	Unauthorized - Invalid or missing token	No links
403	Unauthorized - Admin rights required	No links
404	Theme not found	No links
500	Server internal error	No links

DELETE /theme/{id} Delete a theme

Delete an existing theme. Requires authentication via JWT and admin rights.

Parameters

Name	Description
<b>id</b> * required	Theme ID to delete
string (path)	id

Responses

Code	Description	Links
200	Theme deleted successfully	No links
401	Unauthorized - Invalid or missing token	No links
403	Unauthorized - Admin rights required	No links
404	Theme not found	No links
500	Server internal error	No links

## Projet Knowledge Learning - Gwenaëlle Boussès

POST /theme/add Add a new theme

Create a new theme. Requires authentication via JWT and admin rights.

**Parameters**

No parameters

**Request body required**

application/json

Example Value | Schema

```
{ "name": "Développement Web" }
```

**Responses**

Code	Description	Links
201	Theme created successfully	No links
400	Invalid data	No links
401	Unauthorized - Invalid or missing token	No links
403	Unauthorized - Admin rights required	No links
500	Server internal error	No links

## Users

GET /users/{id} Get a specific user by ID

Return a specific user by ID. Requires authentication.

**Parameters**

Name Description

**id \* required** User ID to retrieve  
string (path) id

**Responses**

Code	Description	Links
200	User found successfully	No links
401	Unauthorized - Invalid or missing token	No links
404	User not found	No links
500	Server internal error	No links

GET /users/ Get all users

Return all users. Requires authentication

**Parameters**

No parameters

**Responses**

Code	Description	Links
200	Users list found successfully	No links
401	Unauthorized - Invalid or missing token	No links
500	Server internal error	No links

## Projet Knowledge Learning - Gwenaëlle Boussès

POST /users/add Add a new user

create a new user. Requires authentication and admin rights.

**Parameters**

No parameters

**Request body required**

application/json

```
{ "name": "John Doe", "email": "john@example.com", "password": "123456" }
```

**Responses**

Code	Description	Links
201	User created successfully	No links
400	Invalid data	No links
401	Unauthorized - Invalid or missing token	No links
403	Unauthorized - Admin rights required	No links
500	Server internal error	No links

PATCH /users/update/{id} Update a user

Update an existing user. Requires authentication.

**Parameters**

Try it out

Name Description

**id** \* required User ID to update  
string (path) id

**Request body required**

application/json

```
{ "name": "Jane Doe", "email": "jane@example.com" }
```

**Responses**

Code	Description	Links
200	User updated successfully	No links
400	Invalid data	No links
401	Unauthorized - Invalid or missing token	No links
404	User not found	No links
500	Server internal error	No links

## Projet Knowledge Learning - Gwenaëlle Boussès

**DELETE** /users/delete/{id} Delete a user

Delete an existing user. Requires authentication and admin rights.

**Parameters**

Name	Description
<b>id</b> * required	User ID to delete
string (path)	id

**Responses**

Code	Description	Links
200	User deleted successfully	No links
401	Unauthorized - Invalid or missing token	No links
404	User not found	No links
500	Server internal error	No links

## Authentification

**GET** /users/confirm/{token} Confirm registration

Verify the registration token and confirm the user registration.

**Parameters**

Name	Description
<b>token</b> * required	Registration token to verify
string (path)	token

**Responses**

Code	Description	Links
200	Registration confirmed successfully	No links
400	Invalid token	No links
500	Server internal error	No links

**POST** /users/authenticate Authenticate a user

Verify user credentials and return a JWT token.

**Parameters**

No parameters

**Request body** required

application/json

Example Value : Schema

```
{  
  "email": "user@example.com",  
  "password": "password123"  
}
```

**Responses**

Code	Description	Links
200	Authentication successful, JWT token returned	No links
400	Invalid data	No links
500	Server internal error	No links

**Lien vers le GitHub du projet**

<https://github.com/cgwena/Knowledge-Learning>

**ReadMe du projet**

Knowledge Learning

**Description**

Knowledge Learning est une application full-stack conçue pour l'entreprise fictive Knowledge Learning qui est une plateforme e-learning.

Le backend est développé avec Express.js et le frontend avec Vue.js 3.

**Structure du Projet**

Le projet est divisé en deux parties principales :

1. Backend (knowledge-learning) : Serveur Node.js avec Express.
2. Frontend (kl-frontend) : Application Vue.js.

**Prérequis**

- Node.js (v14 ou supérieur)
- npm ou yarn
- MongoDB (local ou cloud)
- MailDev (pour tester les emails en dev)

**Installation**

Backend

1. Cloner le dépôt

```
git clone <URL_DU_DEPOT>
```

```
cd knowledge-learning
```

2. Installer les dépendances

```
npm install
```

3. Configuration de l'environnement

Créer un fichier .env à partir du modèle :

Exemple de configuration .env

```
# Serveur
```

```
PORT=3000
```

```
HOST=localhost
```

```
# Base de données
```

Projet Knowledge Learning - Gwenaëlle Boussès

MONGODB\_URI=mongodb://localhost:27017/knowledge\_learning

# Authentification

JWT\_SECRET=your\_jwt\_secret\_key

# Stripe

STRIPE\_SECRET\_KEY=your\_stripe\_secret\_key

STRIPE\_WEBHOOK\_SECRET=your\_stripe\_webhook\_secret

# Mailer

EMAIL\_HOST=smtp.example.com

EMAIL\_PORT=587

EMAIL\_USER=your\_email@example.com

EMAIL\_PASS=your\_email\_password

EMAIL\_FROM=Knowledge Learning <no-reply@example.com>

# Sécurité

CSRF\_SECRET=your\_csrf\_secret\_key

4. Démarrer le serveur

•En développement :

npm start

•En production :

npm run prod

Frontend

1. Aller dans le dossier frontend

cd kl-frontend

2. Installer les dépendances

npm install

3. Démarrer l'application

npm run serve

Accès : <http://localhost:8080>

Tests (Backend)

Le projet utilise Mocha, Chai, Sinon et Supertest pour les tests :

npm run test

## Technologies principales

### Backend

- Express.js
- MongoDB + Mongoose
- JWT
- Stripe
- Swagger
- Nodemailer
- MailDev

### Frontend

- Vue 3
- Vuex
- Vue Router
- Axios
- Vue3 Toastify

## Tester les emails en local (MailDev)

MailDev permet d'intercepter et de visualiser les emails envoyés en local.

Installation (globale ou locale)

npm install -g maildev

Lancer MailDev

maildev

Par défaut :

- Interface : <http://localhost:1080>
- SMTP : localhost:1025

Exemple de configuration .env pour MailDev

EMAIL\_HOST=localhost

EMAIL\_PORT=1025

EMAIL\_USER=unused

EMAIL\_PASS=unused

## ?Documentation API

L'API est documentée avec Swagger :

<http://localhost:3000/api-docs>