# Projet E-learning: ****MySmartCode****

## I. Objectif du projet

Créer un site e-learning nommé **MySmartCode**, simple et fonctionnel, avec deux rôles principaux :

* ‍ **Étudiant** :
  + Inscription
  + Connexion
  + Consultation des cours
  + Déconnexion
* **Administrateur** :
  + Connexion (identifiants prédéfinis)
    - **Email** : admin@gmail.com
    - **Mot de passe** : admin123
  + **Téléversement** de cours (PDF, Word, PowerPoint, HTML, fichiers web, etc.)
  + Suppression de cours
  + Ajout / suppression d'étudiants
  + Déconnexion

## II. Technologies utilisées

* **Frontend** : HTML, CSS, JavaScript, React
* **Backend** : Node.js + Express
* **Base de données** : MongoDB + Mongoose

**III. Fonctionnement de l’application**

L'application **MySmartCode** s’articule autour de **six pages principales**, chacune correspondant à un rôle ou une fonctionnalité bien définie :

**1. Page de Connexion / Accueil**

* **En-tête (haut gauche)** : logo **MySmartCode**
* **Contenu central** :
  + Champ **Adresse email**
  + Champ **Mot de passe**
  + Bouton **Se connecter**
  + Bouton **S’inscrire**

**Comportement attendu** :

* Si l’utilisateur est un **étudiant déjà inscrit** :
  + Il saisit ses identifiants, puis clique sur **Se connecter**.
* Si l’utilisateur est un **nouvel étudiant** :
  + Il clique sur **S’inscrire** pour être redirigé vers la page d’inscription.
* Si l’utilisateur est l’**administrateur** :
  + Il utilise directement les **identifiants prédéfinis** (pas d’inscription requise).

**2. Page d’Inscription (étudiants uniquement)**

* **En-tête (haut gauche)** : logo **MySmartCode**
* **Contenu central** :
  + Champ **Nom et Prénom**
  + Champ **Adresse email**
  + Champ **Mot de passe**
  + Bouton **S’inscrire**

**3. Interface Administrateur (accessible après connexion)**

* **En-tête (haut gauche)** : logo **MySmartCode**
* **Barre de navigation** avec 3 boutons :
  + **Gestion des cours**
  + **Gestion des étudiants**
  + **Déconnexion**

**Fonctionnalités** :

* **Téléversement de cours** (formats acceptés : PDF, Word, PowerPoint, HTML, etc.)
* **Suppression de cours**
* **Suppression d’étudiants**

**4. Interface Étudiant (accessible après connexion)**

* **En-tête (haut gauche)** : logo **MySmartCode**
* **Barre de navigation** avec 2 boutons :
  + **Consultation des cours**
  + **Déconnexion**

**Fonctionnalité principale** :

* L’étudiant peut consulter la liste des cours mis en ligne par l’administrateur.

**5. Comportement de l’interface – Espace Administrateur**

**a. Affichage initial**

* À l’ouverture de l’espace admin, la section **Gestion des cours** est affichée par défaut.
  + Le bouton **Gestion des cours** est mis en évidence (couleur verte).
  + Les autres boutons (**Gestion des étudiants**, **Déconnexion**) apparaissent en bleu.

**Actions possibles** :

* **Ajouter** un cours (formats : PDF, Word, PowerPoint, HTML, etc.)
* **Supprimer** des cours existants

**b. Navigation dans l’interface**

* Si l’administrateur clique sur **Gestion des étudiants** :
  + Ce bouton devient actif (vert) tandis que les autres redeviennent bleus.

**Actions possibles** :

* Visualiser la **liste des étudiants inscrits**
* **Supprimer** un étudiant

**6. Comportement de l’interface – Espace Étudiant**

**a. Affichage initial**

* Lors de la connexion, la section **Consultation des cours** est affichée par défaut.
  + Le bouton actif (**Consultation des cours**) apparaît en vert.
  + Le bouton **Déconnexion** reste bleu.

**b. Fonctionnalité principale**

* L’étudiant a accès à la **liste des cours disponibles**, téléversés par l’administrateur.

## IV. Ce que je demande

1. **Architecture complète et détaillée du projet avec Create React App**

**avec des commentaires**

* + Organisation des dossiers **frontend** et **backend**
  + Commentaires pédagogiques

1. **Code source complet**
   * Fichiers React (**frontend**)
   * Fichiers Node.js / Express (**backend**)
   * Modèles Mongoose (**MongoDB**)
2. **Guide d’installation et d’exécution pour débutants**
   * Installation de Node.js, MongoDB, etc.
   * Configuration des fichiers .env
   * Commandes pour lancer le backend et le frontend

**Solution**

## I- Architecture Complète du Projet

MySmartCode/

│

├── backend/ # Dossier backend : serveur Node.js avec Express

│ ├── controllers/ # Contrôleurs : logique métier pour gérer les requêtes

│ │ ├── authController.js # Gestion des connexions, inscriptions, authentification

│ │ ├── courseController.js # Gestion des opérations liées aux cours (upload, suppression, liste)

│ │ └── studentController.js # Gestion des étudiants (liste, suppression)

│ │

│ ├── models/ # Modèles Mongoose définissant la structure des documents MongoDB

│ │ ├── Admin.js # Modèle Admin (email, mot de passe) — administrateur unique

│ │ ├── Course.js # Modèle Course (titre, description, fichier, date upload)

│ │ └── Student.js # Modèle Student (nom complet, email, mot de passe)

│ │

│ ├── node\_modules/ # Est créé automatiquement par npm

│ │

│ ├── routes/ # Définition des routes API REST avec Express Router

│ │ ├── authRoutes.js # Routes pour authentification : login, inscription

│ │ ├── courseRoutes.js # Routes pour gestion des cours (upload, liste, suppression)

│ │ └── studentRoutes.js # Routes pour gestion des étudiants (liste, suppression)

│ │

│ ├── uploads/ # Dossier pour stocker les fichiers de cours uploadés par l’admin

│ │ # Accessible en lecture via une route statique

│ │

│ ├── middleware/ # Middlewares Express personnalisés

│ │ ├── authMiddleware.js # Middleware pour vérifier l’authentification admin (simplifié)

│ │ └── uploadMiddleware.js # Middleware Multer pour gérer l’upload des fichiers

│ │

│ ├── .env # Variables d’environnement pour le serveur Node.js (ex : MONGODB\_URI, PORT)

│ ├── server.js # Point d’entrée principal du serveur Express

│ │ # Configuration, connexion MongoDB, lancement serveur

│ └── package.json # Dépendances backend et scripts npm

│ # est créé automatiquement par la commande : npm init –y

├── frontend/ # Frontend React (créé avec Create React App)

│ ├── public/ # Fichiers statiques publics

│ │ └── index.html # Fichier HTML principal injectant l’application React

│ ├── src/ # Code source React

│ │ ├── components/ # Composants React réutilisables (ex : Header, NavBar, Formulaires)

│ │ │

│ │ ├── pages/ # Pages principales React

│ │ │ ├── LoginPage.js // Page pour connexion utilisateur

│ │ │ ├── RegisterPage.js // Page pour inscription étudiante

│ │ │ ├── AdminDashboard.js // Tableau de bord admin (upload, suppression)

│ │ │ ├── StudentDashboard.js // Tableau de bord étudiant (consultation cours)

│ │ │ └── Layout.js

│ │ ├── services/ # Fonctions pour appels API au backend (fetch ou axios)

│ │ ├── App.js # Composant principal gérant la navigation conditionnelle par rôle

│ │ ├── index.js # Point d’entrée React (render dans DOM)

│ │ └── styles.css # Styles globaux CSS pour l’application

│ ├── package.json # (optionnel) Dépendances frontend et scripts npm (start, build)

│ └──.env # Variables d’environnement pour l'application React (ex : REACT\_APP\_API\_URL)

│

├── README.md # Documentation du projet (objectif, installation, usage)

└── .gitignore # Fichiers et dossiers ignorés par Git (node\_modules, uploads, .env)

## II. Guide d’installation (débutants)

### 1. Prérequis

* Node.js installé : [https://nodejs.org](https://nodejs.org/)
* MongoDB installé localement ou MongoDB Atlas (base de données en ligne (cloud))

### 2. Créer le dossier principal

|  |
| --- |
| mkdir MySmartCode  cd MySmartCode |

### 3. Installation du Backend

### a. Créer le dossier backend dans ton projet

|  |
| --- |
| mkdir backend  cd backend |

### b. Initialiser un projet Node.js dans le dossier backend

|  |
| --- |
| npm init -y |

Cela va créer un fichier package.json qui contiendra les infos du projet backend (nom, dépendances, scripts...).

### c. Installer les dépendances nécessaires

|  |
| --- |
| npm install express mongoose dotenv cors jsonwebtoken bcryptjs |

Voici à quoi sert chaque dépendance :

|  |  |
| --- | --- |
| **Paquet** | **Utilité** |
| express | Créer facilement un serveur HTTP / API |
| mongoose | Communiquer avec MongoDB |
| dotenv | Gérer les variables d’environnement |
| cors | Autoriser les échanges frontend-backend |
| jsonwebtoken | Créer et vérifier les tokens JWT pour l’auth |
| bcryptjs | Hasher les mots de passe |

NB : Le dossier **node\_modules** est automatiquement **créé par npm**(Node Package Manager)

Donc, **tu ne dois jamais le créer toi-même.**

### 4. Frontend

### a- Installer le frontend

|  |
| --- |
| cd ../frontend # Revenir au dossier racine puis aller dans frontend  npx create-react-app . # crée un projet React complet dans le dossier actuel avec tous les fichiers  # et un package.json prêt à l’emploi.  npm install axios react-router-dom # react-router-dom: Pour gérer la navigation (Login, Admin, etc.)  # axios : Pour faire des requêtes HTTP vers ton backend |

## 5- Les fichiers de base complets

L**es fichiers de base complets** que tu dois **ajouter tout de suite** pour pouvoir **lancer le projet**

**e-learning MySmartCode**, même si tu n’as pas encore développé toutes les fonctionnalités.

## a. ****Fichiers backend (Node.js + Express)****

### i- backend/models/Admin.js

const mongoose = require('mongoose');

const adminSchema = new mongoose.Schema({

email: String,

password: String

});

module.exports = mongoose.model('Admin', adminSchema);

### ii- backend/routes/authRoutes.js

const express = require('express');

const router = express.Router();

// Authentification de l'administrateur (admin uniquement)

router.post('/login', (req, res) => {

const { email, password } = req.body;

if (email === 'admin@gmail.com' && password === 'admin123') {

return res.status(200).json({ message: 'Admin connecté', role: 'admin' });

}

res.status(401).json({ message: 'Identifiants invalides' });

});

module.exports = router;

### iii- backend/server.js

require('dotenv').config();

const express = require('express');

const mongoose = require('mongoose');

const cors = require('cors');

const bcrypt = require('bcryptjs');

const path = require('path');

const Admin = require('./models/Admin');

const authRoutes = require('./routes/authRoutes');

const app = express();

// Middlewares

app.use(cors());

app.use(express.json());

// Dossier statique pour les cours (à créer manuellement)

app.use('/uploads', express.static(path.join(\_\_dirname, 'uploads')));

// Routes API

app.use('/api/auth', authRoutes);

// Connexion MongoDB

mongoose.connect(process.env.MONGODB\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true

})

.then(() => {

console.log('✅ MongoDB connecté');

createAdminIfNotExists();

})

.catch((err) => console.error('❌ Erreur MongoDB :', err));

// Créer automatiquement l’admin

async function createAdminIfNotExists() {

const adminEmail = 'admin@gmail.com';

const adminPassword = 'admin123';

const adminExist = await Admin.findOne({ email: adminEmail });

if (adminExist) return console.log('ℹ️ Admin déjà existant');

const hashed = await bcrypt.hash(adminPassword, 10);

const admin = new Admin({ email: adminEmail, password: hashed });

await admin.save();

console.log('✅ Admin créé automatiquement');

}

// Lancer le serveur

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {

console.log(`🚀 Serveur backend lancé sur http://localhost:${PORT}`);

});

### iv- backend/.env

PORT=5000

MONGODB\_URI=mongodb://127.0.0.1:27017/mysmartcode

JWT\_SECRET=secretkey123

## b. ****Fichiers frontend (React)****

### i- frontend/src/App.js

import React from 'react';

import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';

import LoginPage from './pages/LoginPage';

function App() {

return (

<Router>

<Routes>

<Route path="/" element={<LoginPage />} />

</Routes>

</Router>

);

}

export default App;

### ii- frontend/src/pages/LoginPage.js

import React, { useState } from 'react';

import axios from 'axios';

function LoginPage() {

const [email, setEmail] = useState('');

const [password, setPassword] = useState('');

const [message, setMessage] = useState('');

const handleLogin = async (e) => {

e.preventDefault();

try {

const res = await axios.post(`${process.env.REACT\_APP\_API\_URL}/api/auth/login`, { email, password });

setMessage(res.data.message);

} catch (err) {

setMessage('Erreur de connexion');

}

};

return (

<div style={{ padding: '20px' }}>

<h2>Connexion - MySmartCode</h2>

<form onSubmit={handleLogin}>

<input type="email" placeholder="Email" onChange={(e) => setEmail(e.target.value)} required /><br />

<input type="password" placeholder="Mot de passe" onChange={(e) => setPassword(e.target.value)} required /><br />

<button type="submit">Se connecter</button>

</form>

<p>{message}</p>

</div>

);

}

export default LoginPage;

### iii- frontend/.env

REACT\_APP\_API\_URL=http://localhost:5000

### iv- frontend/src/index.js

import React from 'react';

import ReactDOM from 'react-dom/client';

import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<React.StrictMode>

<App />

</React.StrictMode>

);

### v-frontend/public/index.html (déjà généré par create-react-app)

Assure-toi qu’il contient cette balise :

<div id="root"></div>

**6 - Exécution du projet e-learning**

|  |  |
| --- | --- |
| Question | Réponse |
| Pourquoi en lance le **backend** et le **frontend** maintenant même avant de terminer le reste  des codes ? | S’assurer que l’environnement est prêt  et fonctionnel avant d’écrire la suite du code. |

Solution : Lancer les 3 services dans 3 terminaux différents.

## a. Ouvrir la 1er Terminal pour démarrer MongoDB

MongoDB est nécessaire pour que ton backend fonctionne et puisse enregistrer les utilisateurs

et les cours.

|  |
| --- |
| mongod |

- Cette commande démarre le serveur MongoDB local sur mongodb://localhost:27017.

- Laisse ce terminal ouvert : **il doit rester actif pour que le backend fonctionne**.

## b. Ouvrir la 2éme Terminal pour le backend

|  |  |
| --- | --- |
| Méthode 1 : Sans nodemon | Méthode 2 : Avec nodemon (recommander) |
| npm install  node server.js | npm install  npm run dev |

Si tout est bien configuré, tu verras :

Serveur backend sur http://localhost:5000

MongoDB connecté

## i- Quand utiliser ou ne pas utiliser npm install ?

|  |  |
| --- | --- |
| **Quand utiliser npm install ?** | **Quand ne pas l’utiliser ?** |
| - Après avoir cloné ou créé un projet | - Si les dépendances sont déjà installées |
| - Après modification du fichier package.json | - Si package.json n’a pas changé depuis |

Si le projet est fonctionnelle utiliser seulement :

|  |
| --- |
| node server.js ou npm run dev |

**ii- Les étapes pour installer et utiliser nodemon**

|  |  |  |
| --- | --- | --- |
| **Étape** | **Commande / Action** | **Description** |
| 1. Installer nodemon | npm install --save-dev nodemon | Installe nodemon en dépendance  de développement |
| 2. Modifier package.json | Ajoute dans ton package.json un script "dev" qui utilise nodemon :  "scripts": {  "dev": "nodemon server.js"  } | Configure un script pour lancer nodemon facilement |

## ****iii- Importance de****nodemon

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Commande** | **Avantages** | **Inconvénients** |
| Méthode 1  Sans nodemon | node server.js | - Simple à utiliser  - Pas d’installation supplémentaire nécessaire | - Pas de redémarrage automatique  - Relancer manuellement après  chaque modif |
| Méthode 2  Avec nodemon | npm run dev | - Redémarrage automatique à chaque changement  - Gain de temps  en développement | - Nécessite d’installer nodemon  - Usage surtout pour dev, pas prod |

## c. Ouvrir la 3éme Terminal pour le frontend

|  |
| --- |
| npm install suivant le cas  npm start |

Si tout est bien configuré, tu verras :

webpack compiled successfully

et

Cela ouvrira automatiquement [http://localhost:3000](http://localhost:3000/) dans ton navigateur.

Remarque :

|  |  |
| --- | --- |
| **Quand utiliser npm install ?** | **Quand ne pas l’utiliser ?** |
| - Après avoir cloné ou créé un projet | - Si les dépendances sont déjà installées |
| - Après modification du fichier package.json | - Si package.json n’a pas changé depuis |

Si le projet est fonctionnelle utiliser seulement :

|  |
| --- |
| npm start |

**d- Résumé**

|  |
| --- |
| On utilise 3 terminaux actifs :   1. **Terminal 1** : pour MongoDB (mongod) 2. **Terminal 2** : pour le backend (npm run dev) 3. **Terminal 3** : pour le frontend (npm start) |

**Résultat attendu :**

|  |  |  |
| --- | --- | --- |
| **Élément** | **Port** | **Adresse** |
| MongoDB | --- | mongodb://localhost:27017 |
| Frontend (React) | 3000 | [http://localhost:3000](http://localhost:3000/) |
| Backend (API) | 5000 | <http://localhost:5000/api> |

### 7. Tester le projet

Connecte-toi avec :

|  |  |  |  |
| --- | --- | --- | --- |
| **Rôle** | Nom | **Email** | **Mot de passe** |
| Administrateur |  | [admin@gmail.com](mailto:admin@gmail.com) | admin123 |
| Étudiant | etudiant | [etudiant@gmail.com](mailto:etudiant@gmail.com) | etudiant123 |
| Étudiant | Etudiant1 | [etudiant1@gmail.com](mailto:etudiant1@gmail.com) | Etudiant1123 |
| Étudiant | Etudiant2 | [etudiant2@gmail.com](mailto:etudiant2@gmail.com) | Etudiant2123 |

## Les étapes pour crée un projet MERN

## 1- Crée le diagramme de classe.