Rapport Technique SmartLearn MERN

Date: 7 Janvier 2026
Version: 1.0
Stack: MongoDB + Express.js + React + Node.js

---

1. Vue d'Ensemble

Description

SmartLearn est une application web full-stack d'apprentissage en ligne spécialisée dans l'enseignement des mathématiques. L'application permet aux utilisateurs de créer des cours, générer des exercices, soumettre des réponses et suivre leur progression.

Architecture

```
Frontend (React + Vite)
Port: 5173

         HTTP/REST


Backend (Express.js)
Port: 3000

         Mongoose


Database (MongoDB)
Port: 27017
```

---

2. Backend - Express.js

2.1 Structure

```
backend/
  config/          # Configuration DB
  controllers/     # Logique métier (9 contrôleurs)
  middleware/      # Auth + Error handling
  models/          # Schémas MongoDB (8 modèles)
  routes/          # Endpoints API (10 routes)
```

```
utils/          # Utilitaires
server.js       # Point d'entrée
```

2.2 Technologies

- **Node.js** - Runtime JavaScript
- **Express.js 5.2** - Framework web
- **Mongoose 9.1** - ODM MongoDB
- **JWT** - Authentification
- **bcryptjs** - Hashage mots de passe
- **cors** - Gestion CORS
- **axios** - Client HTTP

2.3 Modèles MongoDB

User

```
{
  username: String (unique),
  email: String (unique),
  password: String (hashé),
  createdAt: Date,
  updatedAt: Date
}
```

UserProfile

```
{
  user: ObjectId → User,
  firstName: String,
  lastName: String,
  bio: String,
  avatar: String (base64),
  phone: String,
  address: { street, city, country, postalCode },
  preferences: { language, difficulty, notifications }
}
```

Course

```
{
  title: String,
  description: String,
  creator: ObjectId → User,
  level: String (beginner|intermediate|advanced),
  imageUrl: String,
  published: Boolean
}
```

Exercise

```
{
  course: ObjectId → Course,
  title: String,
  description: String,
  type: String (mcq|short-answer|essay|coding|math),
  difficulty: String,
  points: Number,
  question: String,
  options: [String],
  correctAnswer: Mixed,
  solution: String
}
```

Submission

```
{
  student: ObjectId → User,
  exercise: ObjectId → Exercise,
  studentAnswer: Mixed,
  status: String (pending|graded|reviewed),
  score: Number,
  feedback: String,
  feedbackSentiment: { label, score }
}
```

Enrollment

```
{
  user: ObjectId → User,
  course: ObjectId → Course,
  progress: Number (0-100),
  completedExercises: [ObjectId],
  enrolledAt: Date
}
```

ChatSession

```
{
  user: ObjectId → User,
  messages: [{ role, content, timestamp }]
}
```

ActivityLog

```
{
  user: ObjectId → User,
```

```
  action: String,
  details: Mixed,
  ipAddress: String,
  timestamp: Date
}
```

2.4 API Endpoints

Méthode	Endpoint	Description
Authentication		
POST	`/api/auth/register`	Inscription
POST	`/api/auth/login`	Connexion
GET	`/api/auth/me`	Utilisateur actuel
Cours		
GET	`/api/courses`	Liste des cours
POST	`/api/courses`	Créer un cours
GET	`/api/courses/:id`	Détails d'un cours
PUT	`/api/courses/:id`	Modifier un cours
DELETE	`/api/courses/:id`	Supprimer un cours
Exercices		
GET	`/api/exercises`	Liste des exercices
POST	`/api/exercises`	Créer un exercice
GET	`/api/exercises/:id`	Détails d'un exercice
PUT	`/api/exercises/:id`	Modifier un exercice
DELETE	`/api/exercises/:id`	Supprimer un exercice
Soumissions		
GET	`/api/submissions`	Liste des soumissions
POST	`/api/submissions`	Soumettre une réponse
PUT	`/api/submissions/:id`	Noter/Feedback
GET	`/api/submissions/professor/submissions`	Tous les soumissions
Inscriptions		
GET	`/api/enrollments`	Mes inscriptions
POST	`/api/enrollments`	S'inscrire à un cours
GET	`/api/enrollments/course/:courseId`	Inscrits d'un cours
PUT	`/api/enrollments/:id/progress`	Mettre à jour progression
Profils		
GET	`/api/profiles/:userId`	Obtenir profil
POST	`/api/profiles`	Créer profil
PUT	`/api/profiles/:userId`	Modifier profil
Chat		
GET	`/api/chat-sessions`	Mes sessions
POST	`/api/chat-sessions`	Créer session
POST	`/api/chat-sessions/:id/messages`	Envoyer message
AI		
POST	`/api/ai/chat`	Chatbot IA

Méthode	Endpoint	Description
POST	`/api/ai/generate-exercises`	Générer exercices
POST	`/api/ai/validate-answer`	Valider réponse

2.5 Middleware

authMiddleware.js

- Vérifie le token JWT dans les headers
- Decode le token et récupère l'utilisateur
- Ajoute `req.user` pour les routes protégées

errorHandler.js

- Capture toutes les erreurs Express
- Formate la réponse en JSON
- Masque les détails sensibles en production

2.6 Sécurité

JWT Authentication - Token généré lors du login - Expiration: 30 jours - Secret stocké dans variables d'environnement

Hashage Passwords - bcryptjs avec 10 salt rounds - Comparaison sécurisée lors du login

Validation - express-validator pour les entrées - Vérification des types et formats - Protection contre injection

---

3. Frontend - React + Vite

3.1 Structure

```
Frontend/src/
  api/              # Configuration Axios
  components/       # Composants réutilisables
  context/          # Context API (Auth)
  pages/            # Pages de l'application (11)
  App.jsx           # Composant racine
  main.jsx          # Point d'entrée
```

3.2 Technologies

- **React 19.2** - Bibliothèque UI
- **Vite 7.2** - Build tool
- **React Router 7.11** - Routing

- **Axios 1.13** - Client HTTP
- **Context API** - State management

3.3 Pages

Page	Route	Description
Home	`/`	Page d'accueil
Login	`/login`	Connexion
Register	`/register`	Inscription
Courses	`/courses`	Catalogue des cours
CourseDetails	`/courses/:id`	Détails d'un cours
MyEnrollments	`/my-enrollments`	Mes cours suivis
ProfessorCourses	`/professor-courses`	Gestion des cours créés
StudentSubmissions	`/student-submissions`	Révision des soumissions
ExerciseGenerator	`/exercise-generator`	Générateur d'exercices IA
Chatbot	`/chatbot`	Assistant IA
Profile	`/profile`	Profil utilisateur

3.4 Composants Clés

Navbar.jsx

- Navigation principale
- Menu contextuel (authentifié/non-authentifié)
- Liens: Courses, My Learning, Practice, AI Tutor, Profile
- Logout button

CourseCard.jsx

```
Props: { course, onEnroll, onEdit, showActions }
Affiche: Titre, description, niveau, image, boutons
```

ProtectedRoute.jsx

```
Vérifie l'authentification avant d'accéder à une route
Redirige vers /login si non authentifié
```

3.5 State Management

AuthContext.jsx

```
État global:
- user: Utilisateur connecté
- loading: État de chargement
- login(email, password)
- register(username, email, password)
```

- `logout()`

`Persistance via localStorage`

3.6 API Client (axios.js)

Configuration

```
baseURL: process.env.VITE_API_URL || 'http://localhost:3000'
timeout: 300000 (5 minutes pour IA)
```

Intercepteurs - Request: Ajoute token JWT automatiquement - Response: Gestion globale des erreurs

---

4. Fonctionnalités Principales

4.1 Authentification

- Inscription avec username, email, password
- Login avec génération de JWT
- Session persistante (localStorage)
- Logout avec suppression du token

4.2 Gestion des Cours

- **Créer**: Titre, description, niveau, image
- **Modifier**: Tous les champs éditables
- **Publier/Dépublier**: Contrôle de visibilité
- **Supprimer**: Avec confirmation
- **Statistiques**: Nombre d'inscrits, soumissions, avis

4.3 Système d'Exercices

- **Types supportés**: MCQ, Réponses courtes, Essais, Coding, Math
- **Création manuelle**: Formulaire détaillé
- **Génération IA**: Via service externe
- **Association**: Lien avec les cours

4.4 Soumissions et Notation

- **Soumettre**: Réponse à un exercice
- **Dashboard**: Vue d'ensemble des soumissions
- **Révision**: Attribution de note (0-100)
- **Feedback**: Commentaires détaillés
- **Analyse sentiment**: Positif/Négatif/Neutre

4.5 Inscriptions

- **Enroll**: S'inscrire en un clic
- **Progression**: Tracking automatique (%)
- **Historique**: Liste des cours suivis
- **Accès contenu**: Exercices du cours

4.6 Profil Utilisateur

- **Informations**: Nom, bio, avatar, téléphone
- **Adresse**: Complète (rue, ville, pays, code postal)
- **Préférences**: Langue, difficulté, notifications
- **Upload avatar**: Image base64 (max 500KB)

4.7 Chatbot IA

- **Interface chat**: Style messagerie
- **Questions/Réponses**: Mathématiques
- **Historique**: Sauvegarde des conversations
- **Sessions**: Gestion multi-sessions

4.8 Générateur d'Exercices

- **Paramètres**: Sujet, type, difficulté, nombre
- **Génération**: Appel service IA
- **Affichage**: Liste des exercices générés
- **Export**: Vers un cours

---

5. Base de Données

5.1 Relations

```
USER (1:1) → USERPROFILE
USER (1:N) → COURSE (créateur)
USER (1:N) → SUBMISSION (étudiant)
USER (1:N) → CHATSESSION
USER (N:M) → COURSE (via ENROLLMENT)
COURSE (1:N) → EXERCISE
EXERCISE (1:N) → SUBMISSION
```

5.2 Indexes

```
users: { email: 1 }, { username: 1 }
courses: { creator: 1 }, { published: 1 }
exercises: { course: 1 }
submissions: { student: 1 }, { exercise: 1 }
enrollments: { user: 1, course: 1 }
```

---

6. Configuration

6.1 Variables d'Environnement

Backend (.env)

```
MONGO_URI=mongodb://localhost:27017/smartlearn
JWT_SECRET=your_secret_key
PORT=3000
AI_CHATBOT_URL=http://localhost:5001
AI_EXERCISE_URL=http://localhost:5002
```

Frontend (.env)

```
VITE_API_URL=http://localhost:3000
```

6.2 Installation

Backend

```
cd SmartLearn/backend
npm install
npm start
```

Frontend

```
cd SmartLearn/Frontend
npm install
npm run dev
```

---

7. Points Forts

Architecture MVC claire - Séparation des responsabilités
API RESTful complète - CRUD sur toutes les entités
Authentification JWT - Sécurisé et stateless
Interface moderne - React 19 avec Vite
Routing avancé - React Router avec routes protégées
State management - Context API simple et efficace
Validation - Côté serveur et client
Error handling - Middleware global
Relations DB - Références Mongoose bien définies
Responsive - Interface adaptative

8. Flux Utilisateur

8.1 Inscription et Login

```
1. Utilisateur accède à /register
2. Remplit formulaire (username, email, password)
3. Frontend envoie POST /api/auth/register
4. Backend hash le password (bcrypt)
5. Création User + UserProfile
6. Génération JWT token
7. Retour token au frontend
8. Stockage dans localStorage
9. Redirection vers /courses
```

8.2 Création de Cours

```
1. Utilisateur clique "Créer un cours"
2. Remplit formulaire (titre, description, niveau)
3. Frontend envoie POST /api/courses (avec JWT token)
4. Backend vérifie token (authMiddleware)
5. Création du cours avec creator = userId
6. Retour du cours créé
7. Mise à jour UI
```

8.3 Soumission d'Exercice

```
1. Utilisateur répond à un exercice
2. Clique "Soumettre"
3. Frontend envoie POST /api/submissions
4. Backend crée Submission (status: pending)
5. Retour confirmation
6. Affichage message succès
```

8.4 Notation par Créateur

```
1. Créateur accède à /student-submissions
2. Voit liste des soumissions
3. Clique sur une soumission
4. Attribue note (0-100) + feedback
5. Frontend envoie PUT /api/submissions/:id
6. Backend met à jour submission (status: graded)
7. Analyse sentiment du feedback
8. Retour soumission mise à jour
```

---

9. Conclusion

SmartLearn MERN est une application full-stack complète et fonctionnelle qui démontre:

- **Maîtrise du stack MERN** complet
- **Architecture professionnelle** scalable
- **Sécurité** avec JWT et hashage
- **UX moderne** avec React 19
- **API RESTful** bien structurée
- **Gestion d'état** efficace

L'application est prête pour la production avec quelques améliorations possibles: - Tests automatisés (Jest, Cypress) - Cache Redis - Upload d'images via service cloud - Websockets pour notifications temps réel - Docker pour conteneurisation

---

Stack: MongoDB 9.1 + Express 5.2 + React 19.2 + Node.js
Ports: Backend 3000, Frontend 5173, MongoDB 27017
Authentification: JWT avec expiration 30 jours
Database: 8 collections avec relations