SkillForge – AI-Powered Learning Management System

1. Project Title

SkillForge – AI-Powered Learning Management System

2. Problem Statement

In the current digital learning landscape, learners often struggle to identify suitable courses, track their progress, and receive personalized recommendations. Traditional e-learning platforms lack intelligent features that adapt to a user's skill level and goals. SkillForge aims to bridge this gap by providing an AI-driven, user-friendly learning management system (LMS) that helps students and professionals discover courses, monitor progress, and receive smart recommendations powered by AI.

3. System Architecture

Architecture Flow:
`Frontend (React.js) → Backend (Node.js + Express API) → Database (MongoDB)`

Architecture Components:

- Frontend:
  Developed using React.js and React Router for seamless navigation between pages. Axios is used for API communication.

- Backend:
  Built using Node.js and Express.js to provide RESTful APIs for authentication, course management, and AI integration.

- Database:
  MongoDB (hosted on MongoDB Atlas) is used for data persistence, including user data, courses, and progress tracking.

- Authentication:
  Implemented using JWT (JSON Web Tokens) for secure login and role-based access.

- Hosting:
    - Frontend: Vercel
    - Backend: Render
    - Database: MongoDB Atlas

4. Key Features

Category	Features
Authentication & Authorization	User registration, login, logout, and role-based access (Admin, Instructor, Student)
Course Management (CRUD)	Create, view, update, and delete courses and lessons
AI Integration	Personalized course recommendations using the OpenAI API
Frontend Routing	Home, Login, Dashboard, Courses, Profile, and Admin Panel pages
Search & Filter	Search courses by category, level, or keyword
Progress Tracking	Track course progress and completion percentage
Responsive UI	Built using TailwindCSS for modern, mobile-friendly design
Hosting	Fully deployed on Vercel (Frontend) and Render (Backend) for live access

5. Tech Stack

Layer	Technologies
Frontend	React.js, React Router, Axios, TailwindCSS
Backend	Node.js, Express.js
Database	MongoDB (Mongoose ORM)

Authentication	JWT (JSON Web Token)
AI Integration	OpenAI API
Hosting	Vercel (Frontend), Render (Backend), MongoDB Atlas (Database)

6. API Overview

Endpoint	Method	Description	Access
`/api/auth/signup`	POST	Register a new user	Public
`/api/auth/login`	POST	Authenticate user and issue JWT token	Public
`/api/courses`	GET	Fetch all available courses	Authenticated
`/api/courses/:id`	PUT	Update existing course	Instructor/Admin
`/api/courses/:id`	DELETE	Delete a course	Admin only
`/api/recommend`	POST	Generate AI-powered course recommendations	Authenticated

7. Expected Outcome

- A fully functional, AI-powered LMS accessible via the web.
- Users can register, explore courses, and get AI-based personalized suggestions.
- Admins can manage courses, instructors, and platform users efficiently.
- AI integration enhances user engagement and learning experience.

8. Searching, Sorting, Filtering, and Pagination

Objective: Enhance user experience and scalability by implementing efficient data retrieval and management for course

1. Searching: Users can search for courses using keywords such as “Python,” “AI,” or “Data Science.”
Implementation:
- Backend: Implement a MongoDB text index on fields like title, category, and description.

Example:
courseSchema.index({ title: 'text', description: 'text', category: 'text' });
Frontend: Search bar with real-time results (debouncing).

2. Filtering: Users can filter courses by category, difficulty level, instructor, or duration.
Implementation:
Use query parameters such as /api/courses?category=AI&level=Beginner for compound queries.

3. Sorting: Courses can be sorted by name (A–Z, Z–A), rating, date added, or popularity.
Implementation:
```
const sortField = req.query.sortBy || 'createdAt';
const sortOrder = req.query.order === 'desc' ? -1 : 1;
const courses = await Course.find(filters).sort({ [sortField]: sortOrder });
```

4. Pagination: Prevents API overload by fetching limited records per page.
Implementation:
```
const page = parseInt(req.query.page) || 1;
const limit = parseInt(req.query.limit) || 10;
const skip = (page - 1) * limit;
const courses = await Course.find(filters).skip(skip).limit(limit);
```

Expected Outcome:
- Faster and more relevant search results.
- Improved performance for large datasets.
- Enhanced user experience with efficient course discovery.

By, Divyansh Bhartia 2401010158