🎯 PART 1 — Project Generation Instructions

You are a senior full-stack software engineer with over 10 years of industry experience, specializing in designing and delivering scalable, production-grade web applications.

Your task is to create a **complete, end-to-end Learning Management System (LMS)** that a computer science student can execute locally, deploy online, and present in a professional interview.

The project must meet **industry standards** and be **fully functional**, **testable**, and **deployable**.

PROJECT DETAILS

Project Title: SkillUp LMS
Description: A modern, full-stack Learning Management System where instructors can create courses, upload lessons and quizzes, and students can enroll, track progress, and receive certificates.

Primary Role: Full-Stack Developer
Objective: Build an end-to-end LMS web application that demonstrates practical understanding of backend, frontend, database, authentication, testing, and deployment.

TECH STACK REQUIREMENTS

- **Frontend:** React (Vite or Next.js), Tailwind CSS, Axios
- **Backend:** Node.js, Express.js
- **Database:** PostgreSQL (via Prisma ORM)
- **Authentication:** JWT + bcrypt
- **API Docs:** Swagger (OpenAPI Spec)
- **Containerization:** Docker + docker-compose
- **Testing:** Jest (backend), React Testing Library (frontend)
- **CI/CD:** GitHub Actions (build, test, deploy)
- **Deployment:** Render / Vercel / Heroku (you choose one)
- **Version Control:** Git + GitHub

DELIVERABLES

Generate and label all project files clearly. Include the following:

1. **Complete file tree** for the LMS project
   - `/backend`, `/frontend`, `/infra`, `.github`, etc.

2. **Backend Implementation**
   - Express server setup with REST APIs
   - Routes, controllers, middleware, models, Prisma schema
   - Authentication (JWT)
   - Course, Lesson, Enrollment, User modules (CRUD)
   - Validation with Joi/Zod
   - `.env.example`, Dockerfile, `docker-compose.yml`
   - Unit + integration tests
   - Swagger API documentation

3. **Frontend Implementation**
   - React components (Course list, Dashboard, Lesson view, etc.)
   - Auth flow (login/register)
   - Protected routes (JWT)
   - API service layer with Axios
   - Dockerfile for frontend
   - Test examples

4. **Infrastructure**
   - Docker Compose setup for backend, frontend, and PostgreSQL
   - GitHub Actions workflow for build + test + deploy
   - Makefile for common commands (e.g., `make install`, `make dev`, `make test`, `make deploy`)

5. **Documentation**
   - `README.md` with:
     - Tech stack summary
     - Architecture diagram (ASCII or markdown)
     - Exact commands for setup, migration, seeding, running, testing, deploying
     - Environment variable setup
     - Demo credentials
     - Troubleshooting guide
   - `presenter_notes.md` — how to demo project (step-by-step)
   - `interview_qna.md` — 20 technical questions + professional answers

6. **Deployment**
   - Include deployment instructions (e.g., Render, Vercel, or Heroku)
   - Explain how to connect backend and frontend in production
   - Provide .env.production example

7. **Quality Checks**
   - Use ESLint + Prettier
   - Minimum 80% backend test coverage
   - Docker-based local development works out of the box

EXECUTION REQUIREMENTS

After you generate the project:

- Every code block must specify the **file path** (e.g., `/backend/src/server.js`)
- Provide **step-by-step CLI commands** to:
  - Initialize the project
  - Run migrations
  - Seed database
  - Start development servers
  - Run tests
  - Deploy to chosen platform
- Ensure that the commands are 100% runnable (no missing steps)
- Provide a **checklist for GitHub upload**
- Explain how to verify successful deployment

FINAL SECTIONS TO INCLUDE

1. ✅ **Zero-to-Deploy Checklist**
   - Setup environment
   - Run app locally
   - Test endpoints
   - Build frontend
   - Deploy
   - Verify production deployment

2. 🧠 **Interview Q&A**
   Include 20 realistic interview questions & short expert-level answers covering:
   - Architecture & tech stack
   - Authentication flow
   - Database design
   - Scaling strategy
   - Security
   - CI/CD
   - Testing
   - Deployment process

3. 🧑‍💻 **Troubleshooting Section**
   - Common setup issues (DB, ports, auth)
   - Fixes for Docker / CI errors

Generate all of this **as one comprehensive output** — labeled, organized, and production-ready.
Do not leave placeholders like `<insert here>`.

Produce concrete code and examples (e.g., real course model, lesson endpoints, React components, etc.).

Your goal is to give a **computer science student a fully functional, deployable, end-to-end Learning Management System** that they can:
- Run locally,
- Push to GitHub,
- Deploy live, and
- Present confidently in an interview.

⚙️ PART 2 — Step-by-Step Procedure (How You'll Build & Run It)

Once ChatGPT gives you the full LMS code:

1. Create Folder & Initialize Git

```
mkdir SkillUp-LMS && cd SkillUp-LMS
git init
```

2. Copy Files Paste all generated files from ChatGPT into your folders (backend, frontend, etc.).

3. Setup Environment

```
cp backend/.env.example backend/.env
cp frontend/.env.example frontend/.env
```

4. Run Docker

```
docker-compose up --build
```

Visit http://localhost:3000 → App should load.

5. Run Tests

```
cd backend && npm run test
cd ../frontend && npm run test
```

6. Push to GitHub

git add .
git commit -m "Initial LMS project"
gh repo create <your-username>/SkillUp-LMS --public --source=. --remote=origin -y
git push -u origin main

7. Deploy

Backend → Render / Heroku

Frontend → Vercel

Add environment variables on both platforms

8. Test Production

Hit live backend API endpoint

Visit deployed frontend

Verify login, course creation, and enrollments work.