System Design Document: Coding Learning Platform

Project: Coding Learning Platform (YouTube-Embedded Courses + Quiz Engine)

Tech Stack (Selected): - Frontend: React + Tailwind CSS - Backend: FastAPI (Python) — recommended (Node.js alternative available) - Database: MongoDB (recommended) — Firebase considered as alternative - Auth: JWT (or Firebase Auth if using Firebase) - Hosting: Vercel/Netlify for frontend, Render/Heroku/AWS/GCP for backend, MongoDB Atlas for DB

Chapter 1: Introduction

1.1 Purpose of the document

This System Design Document (SDD) describes the high-level architecture, object-oriented design, functional and behavioral models, and interaction models for the Coding Learning Platform — a web application that aggregates free coding content (primarily YouTube), organizes it into courses/modules, and provides a quiz/assessment system with Student and Admin panels.

The SDD is intended for developers, project managers, reviewers, and stakeholders who will implement, evaluate, or maintain the system. It provides the information required to implement software components, APIs, and the database schema.

1.2 Scope of the Project

The platform provides the following core capabilities: - Student registration, authentication, and profile management. - Browsing and searching coding courses categorized by topics (e.g., Frontend, Backend, Data Structures). - Course pages that embed playlists/video lessons (YouTube embeds + optional API metadata via YouTube Data API). - Quiz engine per course/lesson with MCQs and attempt tracking. - Admin panel to add/edit/delete courses, lessons, and quiz questions; manage students; view analytics. - Progress tracking and a simple recommendation engine (e.g., suggest next course based on progress).

Out of scope (v1): paid content, in-browser code execution (code runner), hands-on submit grading, advanced proctoring.

1.3 Definitions and Abbreviations

- LMS: Learning Management System
- API: Application Programming Interface
- JWT: JSON Web Token
- CRUD: Create, Read, Update, Delete
- DB: Database
- UI: User Interface

1.4 References

- React documentation
- FastAPI documentation
- MongoDB Atlas docs
- YouTube Data API v3 docs

1.5 Overview of remainder

Chapters 2–5 detail the Object-Oriented Design, Functional Modeling (DFDs), Behavioral Modeling (State Machines), and Interaction Modeling (Use Cases and Sequence Diagrams). Additionally, deployment, security, and API endpoint lists are provided.

Chapter 2: Object-Oriented Design

2.1 Class Diagram (textual description)

The central classes and entities are: - User (student/admin) - Course - Module / Lesson - VideoResource (embed link + metadata) - Quiz - Question - Attempt / QuizAttempt - Enrollment - Notification (optional)

(PlantUML / Mermaid definitions for diagrams are included in the appendix so you can render the images in any UML tool.)

2.2 Data Dictionary

Users - `user_id` (ObjectId, PK) - `name` (string) - `email` (string, unique) - `password_hash` (string) — or OAuth/Firebase token - `role` (enum: student, admin) - `created_at`, `updated_at`

Courses - `course_id` (ObjectId, PK) - `title` (string) - `description` (text) - `category` (string) - `thumbnail_url` (string) - `created_by` (user_id) - `published` (bool) - `created_at`, `updated_at`

Lessons - `lesson_id` (ObjectId, PK) - `course_id` (FK) - `title` (string) - `sequence_no` (int) - `resource` (VideoResource reference) - `content_summary` (text)

VideoResource - `video_id` (string - YouTube ID) - `title`, `description`, `url`, `thumbnail` - `duration` (optional)

Quiz - `quiz_id` (ObjectId) - `lesson_id` (FK) or `course_id` - `title`, `passing_score` - `time_limit` (optional)

Question - `question_id` (ObjectId) - `quiz_id` (FK) - `text` (string) - `options` (array of strings) - `correct_option_index` (int) - `explanation` (string)

Attempt / QuizAttempt - `attempt_id` (ObjectId) - `quiz_id`, `user_id` - `answers` (array of {question_id, selected_index}) - `score` (int) - `started_at`, `completed_at`

Enrollments - `enrollment_id` (ObjectId) - `user_id`, `course_id` - `enrolled_at`, `progress` (percentage)

Chapter 3: Functional Modeling (DFD)

3.1 DFD Level 0 (Context)

External Entities: Student, Admin, YouTube (external API) **System:** Coding Learning Platform **Primary Data Flows:** Registration/Login, Browse Courses, Watch Video, Attempt Quiz, Admin Manage Content

(Level 0 diagram shows Student & Admin interacting with the system; system interacts with YouTube API and MongoDB.)

3.2 DFD Level 1

Break the system into major processes: 1. Authentication Service 2. Course Management Service 3. Content Delivery (Lesson Viewer) 4. Quiz Engine 5. Analytics & Progress Tracker 6. Notification Service

Data Stores: Users DB, Courses DB, Quizzes DB, Attempts DB

3.3 DFD Level 2

Drill into Quiz Engine and Course Management. - Quiz Engine: Load quiz → Validate answers → Store Attempt → Compute score → Return result - Course Management: Admin uploads course metadata → store in Courses DB → generate course page

(Include diagrams using your favorite DFD tool; placeholders exist in the appendix.)

Chapter 4: Behavioral Modeling (State Transition Diagram)

4.1 Student Learning Flow (State Machine)

States: - Visitor → Registered → Enrolled → In Progress → Completed → Certified (optional)

Events / Transitions: - Register/Login → Enroll Course → Start Lesson → Attempt Quiz → Pass/Fail → Update Progress

4.2 Quiz Attempt State Machine

States: Not Started → In Progress → Submitted → Graded → Finished Transitions triggered by user actions and timeouts.

Chapter 5: Interaction Modeling

5.1 Use Case Diagram (Key Use Cases)

Actors: Student, Admin **Use Cases (Student):** Register/Login, Browse Courses, View Lesson, Attempt Quiz, Track Progress **Use Cases (Admin):** Login, Create Course, Edit Course, Delete Course, Create Quiz, Manage Users

5.2 Sequence Diagram (Example: Student Attempts a Quiz)

Sequence: 1. Student requests Quiz via UI 2. Frontend calls `GET /api/quizzes/{id}` 3. Backend validates user enrollment and returns quiz data 4. Student submits answers via `POST /api/quizzes/{id}/attempts` 5. Backend grades and stores attempt, returns score and feedback 6. Frontend updates UI and progress

---

Appendix: API Endpoints (Suggested)

Auth - `POST /api/auth/register` — Register - `POST /api/auth/login` — Login (returns JWT)

Courses - `GET /api/courses` — List - `GET /api/courses/{id}` — Details - `POST /api/courses` — Admin create - `PUT /api/courses/{id}` — Admin update - `DELETE /api/courses/{id}` — Admin remove

Lessons - `POST /api/courses/{id}/lessons` — Create lesson - `GET /api/lessons/{id}` — Get lesson

Quizzes - `GET /api/quizzes/{id}` - `POST /api/quizzes/{id}/attempts`

Admin - `GET /api/admin/users` - `PUT /api/admin/users/{id}`

Database Schema (ER Summary)

- Users (1) — (N) Enrollments
- Courses (1) — (N) Lessons
- Lessons (1) — (N) Quizzes
- Quizzes (1) — (N) Questions
- Users (1) — (N) QuizAttempts

Technology Choices & Recommendation

Backend: FastAPI (recommended) - Pros: Fast development with Python, async support, excellent docs, native pydantic models, easy to write tests. - Cons: If whole team prefers JavaScript, Node.js + Express/NestJS is an alternative.

Database: MongoDB (recommended) - Pros: Flexible schema for lessons, quizzes, and resources; great with JSON-like documents; easy scale; MongoDB Atlas provides hosted DB. - Firebase alternative: Provides Auth, Realtime DB/Firestore, hosting, and easier client SDKs; but vendor lock-in and query complexity for some operations.

Why not Firebase? - Firebase can speed up MVP (auth + hosting + Firestore) but may limit complex queries and server-side logic; for this project, FastAPI + MongoDB gives more control and server-side validation for quizzes.

Security & Deployment Considerations

- Use HTTPS everywhere; secure JWT tokens with short expiry and refresh tokens.
- Hash & salt passwords (bcrypt) if handling auth locally.
- Protect admin endpoints with role-based access control.
- Rate-limit quiz endpoints to prevent abuse.
- Use CORS properly for React frontend origin.

Deployment: - Frontend: Vercel / Netlify - Backend: Render / Heroku / AWS ECS / DigitalOcean App Platform - DB: MongoDB Atlas

---

Resume / LinkedIn Ready Points (Suggested)

- "Designed and implemented a full-stack Coding Learning Platform using React, Tailwind, FastAPI, and MongoDB. Implemented course aggregation from YouTube and an MCQ quiz engine with progress tracking and admin content management."

Next Steps (What I will produce if you want me to continue)

1. Produce PlantUML / Mermaid source for Class Diagram, Use Case, Sequence, State, and DFD diagrams so you can render them into images.
2. Provide ready-to-copy content for a Word document (including placeholders where you will paste diagram screenshots).
3. Optionally generate JSON schema / Mongoose models and FastAPI Pydantic models + sample endpoints.

End of document