ChatGPT

TechZone LMS Project Blueprint

Tech Stack

- **Frontend:** Build the UI with React bundled by Vite for fast development and build times. Vite is known for near-instant reloads and efficient builds, making React development smoother [1] [2]. Use **Tailwind CSS** for utility-first styling; it integrates seamlessly with Vite via the official plugin [3]. The Tailwind/Vite combo enables quick setup and fast CSS builds [3].
- **Backend:** Use **Node.js** with **Express.js**. Express is a minimal and flexible Node.js web framework that provides robust routing and middleware to quickly build REST APIs [4]. Its minimalist core makes creating RESTful routes and handling middleware straightforward, which suits our LMS API needs [4].
- **Database:** Use **PostgreSQL** for its relational strengths. PostgreSQL is a powerful open-source relational database known for reliability, robustness, and performance [5] [6]. Its support for complex queries and data integrity (foreign keys, transactions) makes it a strong fit for course/student data.
- **Deployment (free-tier):** Host the backend on Render's free Hobby plan or Railway's Hobby (these offer free usage for small projects). For example, Railway provides free Postgres hosting, and Render offers a zero-cost tier for Node apps [7]. Deploy the React frontend on Vercel's free tier (optimized for React sites) [8]. In practice, developers often use Railway's free credits to prototype the database/backend and Vercel for frontend, or Render for both APIs and static content, according to project needs [7] [8].

User Roles

- **Admin:** Top-level superuser. Admins have full power over the platform configuration, user management, and course assignments. They can enroll any user, assign roles, and oversee all courses [9]. (Think of the LMS administrator who maintains courses and users without necessarily managing course content day-to-day.)
- **Teacher:** Course instructors. Teachers can create and manage *their* courses: adding lessons, uploading videos, and setting descriptions. They view analytics for their own classes and manage student progress within their courses [10]. A teacher only sees data for courses they teach.
- **Student:** End-users/learners. Students can enroll in available courses, watch videos (live or recorded), and track their own progress. Their dashboard shows courses they are enrolled in and progress/completion status [11]. Students have no rights to modify courses or access other users' data.

Each role should have a tailored dashboard page: for example, admins see site-wide stats and user lists, teachers see course management tools and class reports, and students see enrolled courses and progress bars.

Features & Pages

- **Role-Specific Dashboards:** Each role has a different main dashboard. For instance, learners see a personal dashboard listing their enrolled courses and progress [11]. Teachers get a dashboard to create courses and view analytics (enrollments, completion) for their classes [10]. Admins see

overall platform insights and management tools [9] . Design separate React routes and components for each dashboard view.

- **Course Creation & Management (Teacher):** Teachers can **create** new courses with fields like title and description. They can **edit** courses, add video lessons (by uploading videos), and delete courses. Each course is composed of one or more modules/lessons. Model a Course table with (id, title, description, teacher_id, etc.), and a Lessons (or Modules) table linking to each course. (See *Database Schema* below.) Provide UI forms for teachers to CRUD (create/read/update/delete) courses and upload content.
- **Video Hosting:** Store actual videos on an external free video platform to save bandwidth. For example, embed videos hosted on **Streamable** (free plan) or **Cloudflare Stream** (video-on-demand service) instead of hosting in our server. Cloudflare Stream specifically "encodes, stores, and delivers optimized video" and can be used to handle different devices and bandwidths [12] . Teachers upload videos via the app, but those files go to the external service; the LMS stores only the video URL in the database. This ensures our app can stream videos publicly without managing raw video storage.
- **Student Enrollment & Course Viewing:** Students can browse or search courses and **enroll**. Once enrolled, they can access the course page which lists all video lessons. Support both **live** and **recorded** video content – the LMS can schedule live streams (e.g. via embed or external live tools) or serve recorded videos on-demand. (By design, LMS content "can be live or recorded so individuals in different locations... can easily access it" [13] .) Ensure access control so only enrolled students see course details.
- **Skip Detection in Video Playback:** To prevent students from jumping ahead without watching content, implement a simple check in the video player. For example, track the maximum `currentTime` watched and listen to the HTML5 video `seeking` event. If the user tries to seek past the last watched point, reset the video back. This basic logic (see cited code) effectively blocks skipping ahead [14] [15] .
- **Auto-Exit on Inactivity (Live Sessions):** In live video sessions, automatically remove inactive users. Use a front-end idle timer: listen for user events (mouse moves, keypresses, clicks, etc.) and reset a timer on activity. If no interaction occurs for a set period (e.g. 5 minutes), trigger an exit from the session [16] . (As described in live app blogs, this uses "inactivity timer" and "reset on activity" logic [16] . Optionally show a "Still here?" prompt before leaving.)
- **Completion Tracking:** Mark lessons and courses complete based on video watching. For example, require the student to watch each video fully (or at least a high percentage like 95%) to count as done [17] . Track per-user per-lesson watch percentage on the front end, and when it exceeds the threshold, mark that lesson completed. Compute course completion when all lessons are done.
- **Certificate Generation:** When a student completes **all lessons in a course**, automatically issue a completion certificate. Define this trigger as "all modules finished." Many LMS systems automate cert issuance on such criteria [18] [19] . Implement logic that, upon detecting 100% course completion, generates a certificate (e.g. PDF with student name/course name) and records it in a `certificates` table. (There's no quiz; just full video completion triggers it.)
- **Teacher Analytics:** Provide basic course reports for teachers: e.g., total enrollments per course, overall course completion rate, and average video watch percentage. Many LMS dashboards include "enrollment trends" and "course completion rates" as key metrics [20] [21] . For example, show *% of content viewed* or *number of completed lessons* per student [21] . This helps teachers see how well students engage with their content.

Authentication

- **Email/Password:** Allow users to register with email and password. Use a secure password-hashing library (e.g. bcrypt) to store hashes [22] . For example, use Passport.js or a similar auth

middleware on Express; the popular *bcrypt* algorithm is widely recommended for secure password storage [22].

- **Google OAuth:** Also support "Login with Google". You can plug in Passport.js's Google OAuth strategy for this. The passport-google-oauth2 module integrates Google login into Node/Express apps [23]. Users clicking "Login with Google" will be redirected through Google OAuth and your backend will create or find their user record. (This complements email login by simplifying sign-up.)

Database Schema

Design relational tables roughly like this:
- **users:** id (UUID), email (unique), password_hash, first_name, last_name, role (enum: admin/teacher/student) [24].
- **courses:** id, instructor_id (FK→users), title, description, etc. [25].
- **lessons (modules):** id, course_id (FK→courses), title, video_url (the hosted video link), etc. [26].
- **enrollments:** id, user_id (FK→users), course_id (FK→courses), enrolled_at, completed_at, progress_pct [27].
- **lesson_progress (optional):** Track per-lesson view progress (FKs to enrollment and lesson, plus watched duration, last_position, completed boolean) [28].
- **certificates:** id, user_id, course_id, issued_at (when certificate was granted).
This schema mirrors typical LMS data models (see example schema) [24] [25]. Adjust fields (timestamps, metadata, etc.) as needed.

API Structure

Implement REST endpoints along these lines:

- **Auth endpoints:**
- `POST /auth/signup` – create new user with email/password.
- `POST /auth/login` – verify credentials and return a token or session.
- `GET /auth/google` – start Google OAuth flow (handled by Passport).
- `GET /auth/google/callback` – Google OAuth callback endpoint.

- **User endpoints:**

- `GET /users/me` – get current user profile.
- `PUT /users/me` – update profile (name, etc.).
- (Admin only) `GET /users` – list users; `PATCH /users/:id` – change user role; etc.

- **Course endpoints:**

- `GET /courses` – list all courses (or search).
- `GET /courses/:id` – course details and lessons list.
- `POST /courses` – create a new course (teacher only).
- `PATCH /courses/:id` – update course info (teacher only).
- `DELETE /courses/:id` – delete course (teacher/admin).

- **Lesson endpoints:**

- `POST /courses/:courseId/lessons` – add a new lesson to a course.
- `PATCH /lessons/:id` – update a lesson.
- `DELETE /lessons/:id` – remove a lesson.

- **Enrollment endpoints:**

- `POST /courses/:id/enroll` – enroll current student in course.
- `GET /users/me/enrollments` – list current user's enrollments and progress.

- **Progress tracking:**

- `POST /lessons/:id/progress` – record video watch progress (e.g. send currentTime or completion).
- `GET /courses/:courseId/completion` – (optional) check if current user has finished course.

- **Certificate issuance:**

- `GET /courses/:courseId/certificate` – when called by a completed student, return the certificate (generate or fetch).
- `GET /users/me/certificates` – list issued certificates for the student.

- **Admin analytics endpoints (optional):** For simplicity, many analytics can be computed client-side from enrollments data. You might add `GET /courses/:id/analytics` for teacher dashboards if needed.

Deployment Plan

- **Backend Hosting:** Deploy the Node/Express API to a free tier like **Render** (Hobby Web Service) or **Railway** (Hobby). These support Node apps and can stay up continuously on free or low-cost plans. For example, render.com has a free plan with automatic sleeps when idle (ok for testing) [7].
- **Frontend Hosting:** Deploy the React app on **Vercel**'s free tier. Vercel provides instant React deployment with zero config and global CDN [8]. Connect your Git repository and Vercel auto-detects React, builds, and deploys. It gives a `.vercel.app` URL by default [8].
- **Database Hosting:** Use a free hosted Postgres service. **Railway** offers a free PostgreSQL database up to a credit limit, or **Supabase** can be used as a free Postgres backend [29]. Alternatively, Render's "Postgres" add-on also has a free tier. Configure the database URL as an environment variable for the backend.
- **Env & CI/CD:** Store secrets (DB credentials, Google OAuth keys) in the host's environment variables (never commit to repo). Use Git-based deployment: pushing to GitHub can trigger Vercel (frontend) and Render (backend) to rebuild automatically.

Development Roadmap (Example Sprints)

1. **Week 1:** Set up repo and dev environment. Scaffold React app (Vite+Tailwind) and Express server (Node project). Implement database models in Postgres (create tables for users, courses, lessons).
2. **Week 2:** Build user authentication. Implement email/password sign-up/login (with bcrypt) and Google OAuth via Passport.js. Create React forms for auth.
3. **Week 3:** Develop core CRUD APIs: user profile endpoints, course CRUD (teachers), lesson CRUD. Build corresponding React pages/forms for course/lesson management (teacher UI).
4. **Week 4:** Implement student workflow: course listing page, enrollment action, and course viewing. Embed videos using a dummy host (Streamable links). Track video playback progress on client side (with "timeupdate" events).
5. **Week 5:** Add extra features: disable skip logic (video seeking), inactivity timer for live sessions (JS idle detection) [14] [16]. Implement completion logic: mark lesson done when watched end.
6. **Week 6:** Certificate logic: upon full course completion, generate a PDF/record in `certificates` table. Teacher analytics: simple counts from enrollments (show on dashboard). Finalize UI polish.
7. **Week 7:** Testing and deployment: fix bugs, write basic tests if possible. Configure Vercel and Render deployments, migrate database. Launch a demo build.

(Adjust the timeline as needed; this is a guideline for an MVP approach.)

AI-Ready Prompt

```
# AI Agent Prompt: TechZone LMS Code Generation

You are tasked with writing code for an LMS project named "TechZone" using
the following specifications:

- **Tech Stack:** React (with Vite and Tailwind CSS) on the frontend, Node.js
with Express on the backend, and PostgreSQL as the database.
- **Authentication:** Implement email/password sign-up/login (hash passwords
with bcrypt) and Google OAuth login (using Passport.js).
- **User Roles:** Three roles: Admin, Teacher, Student.
- **Features to implement:**
  - Role-specific dashboards: different UI for admin, teacher, student.
  - Teacher can create/edit/delete courses (title, description, video
lessons).
  - Store uploaded videos on a free host (e.g. Streamable or Cloudflare
Stream) and save their URLs.
  - Students can enroll in courses and watch video lessons (support live and
recorded).
  - Detect if a user tries to skip ahead in a video; prevent forward seeking
beyond what's watched.
  - In live sessions, auto-disconnect inactive users after a timeout.
  - Track lesson completion: mark a lesson done when video is fully watched
(>=95%).
  - Issue a certificate when a student completes all lessons in a course.
  - Teacher analytics: provide total enrollments, average completion rate,
and view percentage for their courses.
```

```
- **Database models:** Design tables for users, courses, lessons/videos,
enrollments, and certificates.
- **API Endpoints:**
  - Auth: /auth/signup, /auth/login, /auth/google, /auth/google/callback.
  - Users: /users (CRUD), /users/me.
  - Courses: /courses (CRUD).
  - Lessons: /courses/:id/lessons (CRUD).
  - Enrollments: /courses/:id/enroll.
  - Progress: /lessons/:id/progress to record watch time.
  - Certificates: /courses/:id/certificate (to fetch or download).
- **Deployment:** Plan to deploy on free services (e.g. front end on Vercel,
backend on Render, PostgreSQL on Railway). Use environment variables for
config.

Please write well-structured, modular code with clear component/API design,
reflecting the specs above. Focus on express routes, React components, and
database schema models that match the requirements. Use comments to clarify
each part.
```

---

1 2 React + Vite: why use? - DEV Community
https://dev.to/doccaio/react-vite-why-use-cg2

3 Installing Tailwind CSS with Vite - Tailwind CSS
https://tailwindcss.com/docs/installation/using-vite

4 Express - Node.js web application framework
https://expressjs.com/

5 6 PostgreSQL: The world's most advanced open source database
https://www.postgresql.org/

7 Discussion of Deploy Node.js Apps Like a Boss: Railway vs. Render vs. Heroku (Zero-Server Stress) - DEV Community
https://dev.to/alex_aslam/deploy-nodejs-apps-like-a-boss-railway-vs-render-vs-heroku-zero-server-stress-5p3/comments

8 How to Deploy a React Site with Vercel
https://vercel.com/guides/deploying-react-with-vercel

9 10 11 What Is LMS Administration? [2025 Overview]
https://www.educate-me.co/blog/lms-administration

12 Delivering Videos with Cloudflare · Cloudflare Fundamentals docs
https://developers.cloudflare.com/fundamentals/reference/policies-compliances/delivering-videos-with-cloudflare/

13 What is a learning management system? Plus top LMSs | HiBob
https://www.hibob.com/hr-glossary/learning-management-system/

14 15 html - How to disable seeking with HTML5 video tag ? - Stack Overflow
https://stackoverflow.com/questions/11903545/how-to-disable-seeking-with-html5-video-tag

16 How Web-Based Douyin Live Rooms Detect User Inactivity and Auto Exit | 陈涛のblog
https://www.nextdaddy.cn/article/2409b3f5-e6ae-80e7-8d44-c7764da5496f

17 Can I change Progress tracking to not count watching videos? - Educators - Open edX discussions
https://discuss.openedx.org/t/can-i-change-progress-tracking-to-not-count-watching-videos/13367

18 19 How To Create Automated Certification for Course Completion
https://learnpresslms.com/blog/automated-certification-for-course-completion/

20 Make Data-Driven Decisions with Tutor LMS Analytics | Tutor LMS
https://tutorlms.com/analytics/

21 11 LMS Reports You Need to Track and Optimize in 2025
https://www.educate-me.co/blog/lms-reporting

22 Password hashing in Node.js with bcrypt - LogRocket Blog
https://blog.logrocket.com/password-hashing-node-js-bcrypt/

23 passport-google-oauth2
https://www.passportjs.org/packages/passport-google-oauth2/

24 25 26 27 28 Building a Production-Ready LMS Platform: A Complete Guide to Modern EdTech Architecture - DEV Community
https://dev.to/nadim_ch0wdhury/building-a-production-ready-lms-platform-a-complete-guide-to-modern-edtech-architecture-16ek

29 How to deploy a Node/React/Postgres app for free and permanently? | by Isar | Medium
https://medium.com/@isar_/how-to-deploy-a-node-react-postgres-app-for-free-and-permanently-35042890be9c