Code for building website

Absolutely—let's rebuild this as a **clean, modern, AI-powered, gamified learning platform** with:

- email login
- a good-looking UI (Tailwind)
- YouTube video player with **AI summaries of the video you're watching**
- **AI-generated quizzes** from the video transcript
- **XP, levels, badges**, streaks
- **personalized recommendations** driven by clickstream + quiz history
- full **clickstream tracking** (page views, video events, quiz attempts, AI actions)
- **Firebase Hosting** deploy + **Firestore** + **Cloud Functions** (secure OpenAI calls)
- **Git/GitHub** from the start

Below is everything from scratch for **VSCode + Windows PowerShell**.

I explain every line **inside the code as comments**, so you can read code + explanation together.

---

0) Install & check tools (one-time)

In a fresh PowerShell window:

```
node -v          # should print v20+ (install Node LTS from nodejs.org if missing)
npm -v           # should print 9+
git --version    # install from git-scm.com if missing
npm i -g firebase-tools
firebase --version
firebase login   # opens browser; pick your Google account
```

---

1) Create the project (Vite + React + Tailwind)

```
# Create a workspace folder
mkdir ai-gamified-learning
cd ai-gamified-learning

# Scaffold a React app with Vite
npm create vite@latest frontend -- --template react

# Move into the app and install deps
cd frontend
npm install

# UI + app deps
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p

# Firebase client + router + fetch helper
npm install firebase react-router-dom axios

# Start dev server once to verify
npm run dev
```

Open the URL it shows (e.g., http://localhost:5173). You should see the Vite splash page.

---

2) Initialize Git & GitHub

```
git init
git add .
git commit -m "chore: bootstrap vite react app"
# Create a new GitHub repo at https://github.com/new named ai-gamified-l
earning
git remote add origin https://github.com/<your-username>/ai-gamified-lear
ning.git
```

```
git branch -M main
git push -u origin main
```

3) Tailwind setup (make it pretty fast)

Edit **tailwind.config.js** (replace content):

```
// tailwind.config.js
/** @type {import('tailwindcss').Config} */
export default {
  // Tell Tailwind where to scan for class names
  content: ["./index.html", "./src/**/*.{js,jsx}"],
  theme: {
    extend: {
      // A few custom tokens for polish (optional)
      boxShadow: {
        soft: "0 10px 25px rgba(0,0,0,.06)"
      }
    },
  },
  plugins: [],
};
```

Edit **src/index.css** (replace content):

```
/* src/index.css */

/* Tailwind's base styles, components, and utilities */
@tailwind base;
@tailwind components;
@tailwind utilities;

/* A tiny reset + nice font */
:root {
  --bg: #0b1020;
  --card: #121833;
  --muted: #97a0b5;
  --brand: #7c5cff;
}
html, body, #root {
  height: 100%;
}
body {
  background: linear-gradient(180deg, #0c1225, #0b0f1e 50%, #0a0e1b);
  color: #e6e9f2;
  font-family: ui-sans-serif, system-ui, -apple-system, Segoe UI, Roboto, H
elvetica, Arial, Apple Color Emoji, Segoe UI Emoji;
}
.card {
  background: var(--card);
  border-radius: 1rem;
  box-shadow: var(--shadow, 0 10px 25px rgba(0,0,0,.15));
}
.btn {
  @apply inline-flex items-center px-4 py-2 rounded-xl font-medium transiti
on;
  background: var(--brand);
}
.btn:hover { filter: brightness(1.1); }
.badge {
  @apply text-xs px-2 py-1 rounded-full;
  background: rgba(255,255,255,.08);
}
```

Edit **src/main.jsx** (replace content):

```
// src/main.jsx
// Import React and ReactDOM for rendering the app
import React from "react";
import ReactDOM from "react-dom/client";
// Import global CSS (Tailwind + custom tokens)
import "./index.css";
// Import the root App component
import App from "./App.jsx";
```

```
// Render <App /> into #root in index.html
ReactDOM.createRoot(document.getElementById("root")).render(
  // StrictMode helps catch problems in dev; safe to keep
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

---

4) Firebase project + local init

1. Create a Firebase project at https://console.firebase.google.com (e.g., `ai-gamified-learning-<yourname>`).

   In the project:

- Enable **Authentication → Email/Password**
- Enable **Firestore (Native)**
- Enable **Hosting**
- (We'll add Functions in a minute)

1. Back in PowerShell (inside `frontend`):

```
firebase init
```

Choose:

- Firestore, Functions, Hosting (spacebar to select)
- Use existing project → pick the one you created
- Hosting public directory: `dist`
- Single-page app: `Yes`
- Functions language: **JavaScript**
- Use ESLint: your choice
- Install deps now: **Yes**

This creates `firebase.json`, `.firebaserc`, and a **functions/** folder.

---

5) App structure (files you'll add now)

```
frontend/
  src/
    lib/
      firebase.js
      tracker.js
      points.js
    components/
      Navbar.jsx
      VideoPlayer.jsx
      Quiz.jsx
      XPBar.jsx
    pages/
      Home.jsx
      Lesson.jsx
      Profile.jsx
      Admin.jsx
    App.jsx
```

We'll paste each file with inline comments.

5.1 Firebase client init

Create **src/lib/firebase.js**:

```
// src/lib/firebase.js

// Initialize Firebase client SDK for Auth + Firestore
import { initializeApp } from "firebase/app";          // core initializer
import { getAuth } from "firebase/auth";               // auth (login/logout)
import { getFirestore, serverTimestamp } from "firebase/firestore"; // db + s
erver time

// ↓ Replace these with your Firebase Web App config from Console → Proj
ect settings → Web app
const firebaseConfig = {
  apiKey: "REPLACE",
  authDomain: "REPLACE.firebaseapp.com",
  projectId: "REPLACE",
  storageBucket: "REPLACE.appspot.com",
  messagingSenderId: "REPLACE",
  appId: "REPLACE"
};

// Create the app instance (must be done before using auth/db)
const app = initializeApp(firebaseConfig);

// Export Auth + Firestore handles for use across the app
export const auth = getAuth(app);
export const db = getFirestore(app);
// Export server timestamp helper so all events use backend time
export const ts = serverTimestamp;
```

> In Firebase Console → Project settings → "Add app" → "Web" to get the config values.

5.2 Clickstream tracker (client-side writes to Firestore)

Create **src/lib/tracker.js**:

```
// src/lib/tracker.js

// Firestore helpers for writing documents
import { collection, addDoc } from "firebase/firestore";
// Our db instance + server timestamp helper
import { db, ts } from "./firebase";

/**
 * Get or create a lightweight session ID stored in localStorage.
 * This lets us stitch events together within a browser session.
 */
function getSessionId() {
  let s = localStorage.getItem("sessionId");
  if (!s) {
    s = "sess_" + Math.random().toString(36).slice(2);
    localStorage.setItem("sessionId", s);
  }
  return s;
}

/**
 * Track an event into Firestore /events collection.
 * @param {string} eventType - e.g., 'page_view', 'video_play', 'quiz_submi
t'
 * @param {object} data - free-form metadata to attach
 */
export async function trackEvent(eventType, data = {}) {
  // Build the payload with consistent fields
  const payload = {
    sessionId: getSessionId(),        // link events within a session
    eventType,                        // what happened
    pageUrl: window.location.pathname, // where it happened
    metadata: data,                   // any additional details
    timestamp: ts()                   // server-side event time
  };
  try {
    // Write to Firestore "events" as a new doc
    await addDoc(collection(db, "events"), payload);
  } catch (err) {
    // For demo, log if something fails
    console.error("trackEvent failed:", err);
  }
}
```

5.3 Gamification helpers (XP/levels/badges)

Create **src/lib/points.js**:

```
// src/lib/points.js

// Configurable constants for XP economy
```

```
export const XP_PER_MINUTE_WATCHED = 5;  // each minute watched yiel
ds XP
export const XP_PER_CORRECT_ANSWER = 20; // each correct quiz answe
r yields XP
export const LEVEL_XP = 200;            // XP threshold per level (simple linea
r)

// Compute level from total XP (simple linear model)
export function levelFromXP(xp) {
  return Math.floor(xp / LEVEL_XP) + 1;
}

// Award a badge when a rule matches (simple demo rules)
export function badgesFor({ totalMinutes, totalCorrect }) {
  const badges = [];
  if (totalMinutes >= 30) badges.push("Half-Hour Hero");
  if (totalMinutes >= 60) badges.push("One-Hour Scholar");
  if (totalCorrect >= 10) badges.push("Quiz Apprentice");
  if (totalCorrect >= 25) badges.push("Quiz Ninja");
  return badges;
}
```

5.4 Fancy Navbar

Create **src/components/Navbar.jsx**:

```
// src/components/Navbar.jsx
import React from "react";
import { Link } from "react-router-dom";

/**
 * Top navigation with brand + quick links.
 * Tailwind utility classes keep it stylish and compact.
 */
export default function Navbar({ right }) {
  return (
    <header className="sticky top-0 z-40 backdrop-blur bg-black/30">
      <div className="max-w-6xl mx-auto px-4 py-3 flex items-center justif
y-between">
        {/* Brand */}
        <Link to="/" className="text-lg font-semibold tracking-wide">
          <span className="text-white">Health</span>
          <span className="text-[var(--brand)] ml-1">Coach AI</span>
        </Link>

        {/* Primary nav */}
        <nav className="hidden md:flex gap-6 text-sm text-[var(--mute
d)]">
          <Link to="/lesson/lesson1" className="hover:text-white">Lesson 1
</Link>
          <Link to="/profile" className="hover:text-white">Profile</Link>
          <Link to="/admin" className="hover:text-white">Admin</Link>
        </nav>

        {/* Right side (Auth status or buttons) */}
        <div className="flex items-center gap-2">{right}</div>
      </div>
    </header>
  );
}
```

5.5 YouTube Player component (tracks play/pause/seek/end)

Create **src/components/VideoPlayer.jsx**:

```
// src/components/VideoPlayer.jsx
import React, { useEffect, useRef } from "react";
import { trackEvent } from "../lib/tracker";

/**
 * YouTube IFrame API wrapper that fires tracking events.
 * Props:
 *   videoId: string (YouTube ID)
 *   onSecondsWatched: (secs) => void (to award XP)
 */
```

```
export default function VideoPlayer({ videoId, onSecondsWatched }) {
  const playerRef = useRef(null);   // ref to the <div> container
  const ytPlayer = useRef(null);    // store the YT player instance
  const lastTime = useRef(0);       // track playtime deltas
  const intervalId = useRef(null);  // store interval for watch time ticks

  // Load the YouTube IFrame API if not present
  useEffect(() ⇒ {
    // If API already loaded globally, just init player
    if (window.YT && window.YT.Player) {
      create();
    } else {
      // Create script tag for YT API
      const tag = document.createElement("script");
      tag.src = "https://www.youtube.com/iframe_api";
      document.body.appendChild(tag);
      // When API ready, window.onYouTubeIframeAPIReady is called once
      window.onYouTubeIframeAPIReady = () ⇒ create();
    }

    // Clean up on unmount
    return () ⇒ {
      if (intervalId.current) clearInterval(intervalId.current);
      if (ytPlayer.current && ytPlayer.current.destroy) ytPlayer.current.destroy
();
    };

    // Create the player after API is loaded
    function create() {
      ytPlayer.current = new window.YT.Player(playerRef.current, {
        videoId,
        playerVars: {
          // Minimal UI; we track ourselves
          modestbranding: 1,
          rel: 0
        },
        events: {
          onReady: handleReady,
          onStateChange: handleStateChange
        }
      });
    }

    // When player is ready, mark a page/video_view event
    async function handleReady() {
      await trackEvent("video_ready", { videoId });
    }

    // Map YT states to our tracker and compute watch time
    async function handleStateChange(e) {
      const YTState = window.YT.PlayerState; // {UNSTARTED:-1, ENDED:0, P
LAYING:1, PAUSED:2, BUFFERING:3, CUED:5}
      const state = e.data;
      const currentTime = ytPlayer.current.getCurrentTime();

      if (state === YTState.PLAYING) {
        // Start ticking every second to accumulate watch time
        lastTime.current = currentTime;
        intervalId.current = setInterval(() => {
          try {
            const t = ytPlayer.current.getCurrentTime();
            const delta = Math.max(0, t - lastTime.current);
            lastTime.current = t;
            // Send watched seconds up for XP
            onSecondsWatched?.(delta);
          } catch {}
        }, 1000);
        await trackEvent("video_play", { videoId, currentTime });
      }

      if (state === YTState.PAUSED) {
        // Stop ticking
        if (intervalId.current) clearInterval(intervalId.current);
        await trackEvent("video_pause", { videoId, currentTime });
      }
      if (state === YTState.ENDED) {
        if (intervalId.current) clearInterval(intervalId.current);
        await trackEvent("video_complete", { videoId });
      }
    }
  }, [videoId, onSecondsWatched]);

  // Render the container div that YT replaces with an <iframe>
  return (
    <div className="rounded-2xl overflow-hidden shadow-soft">
      <div ref={playerRef} className="w-full aspect-video bg-black" />
    </div>
  );
}
```

5.6 XP bar (visual feedback)

Create **src/components/XPBar.jsx**:

```
// src/components/XPBar.jsx
import React from "react";
import { levelFromXP, LEVEL_XP } from "../lib/points";

/**
 * Visualize current XP and level.
 * Props: xp (number)
 */
export default function XPBar({ xp = 0 }) {
  const level = levelFromXP(xp);              // compute level from XP
  const progress = (xp % LEVEL_XP) / LEVEL_XP;// progress within current l
evel
  const pct = Math.round(progress * 100);     // 0-100%

  return (
    <div className="card p-4">
      <div className="flex items-baseline justify-between">
        <h3 className="font-semibold">Level {level}</h3>
        <span className="badge">{xp} XP</span>
      </div>
      <div className="mt-3 h-3 w-full rounded-full bg-white/10 overflow-hidden">
        <div className="h-full" style={{ width: `${pct}%`, background: "var(--brand)" }} />
      </div>
      <div className="mt-2 text-xs text-[var(--muted)]">{pct}% to next level</div>
    </div>
  );
}
```

5.7 Quiz (pretty + scored + tracked)

Create **src/components/Quiz.jsx**:

```
// src/components/Quiz.jsx
import React, { useState } from "react";
import { collection, addDoc } from "firebase/firestore";
import { db, ts } from "../lib/firebase";
import { trackEvent } from "../lib/tracker";

/**
 * Quiz renders MCQs and records attempts.
 * Props:
 *  - questions: [{ id, q, choices:[], answerIndex }]
 *  - videoId: string
 *  - onAwardXP: (xp) => void
 */
export default function Quiz({ questions = [], videoId, onAwardXP }) {
  const [answers, setAnswers] = useState({});
  const [submitted, setSubmitted] = useState(false);
  const [score, setScore] = useState(null);

  // Toggle one answer; track every answer click
  const choose = (qId, idx) => {
    setAnswers((s) => ({ ...s, [qId]: idx }));
    trackEvent("quiz_question_answer", { videoId, questionId: qId, selected: i
dx });
  };

  // Submit: score, persist attempt, award XP, and emit event
  const handleSubmit = async () ⇒ {
    if (!questions.length) return;

    let correct = 0;
    const details = [];

    for (const q of questions) {
      const sel = answers[q.id];
      const isCorrect = sel === q.answerIndex;
      if (isCorrect) correct++;
      details.push({ qId: q.id, selected: sel ?? null, correct: !!isCorrect });
    }

    const pct = Math.round((correct / questions.length) * 100);
    setScore(pct);
    setSubmitted(true);

    // Save attempt to Firestore
    await addDoc(collection(db, "quizAttempts"), {
      videoId,
      answers: details,
      score: pct,
      createdAt: ts()
    });

    // Track submit
    trackEvent("quiz_submit", { videoId, score: pct });

    // Award XP for correct answers (20 per correct by default)
    onAwardXP?.(correct * 20);
  };

  if (!questions.length) {
    return <div className="text-sm text-[var(--muted)]">No quiz available
```

```
yet.</div>;
 }

 return (
   <div className="card p-5">
     <div className="flex items-center justify-between">
       <h3 className="font-semibold">Quiz</h3>
       {!submitted && (
         <button className="btn" onClick={handleSubmit}>Submit</button>
       )}
     </div>

     <div className="mt-4 space-y-5">
       {questions.map((q, i) => (
         <div key={q.id} className="p-4 rounded-xl bg-white/5">
           <p className="font-medium mb-2">{i + 1}. {q.q}</p>
           <div className="space-y-2">
             {q.choices.map((c, idx) => {
               const active = answers[q.id] === idx;
               return (
                 <button
                   key={idx}
                   onClick={() => choose(q.id, idx)}
                   className={`w-full text-left px-3 py-2 rounded-lg transition ${
                     active ? "bg-[var(--brand)]/20 border border-[var(--brand)]" :
"bg-white/5 hover:bg-white/10"
                   }`}
                 >
                   {String.fromCharCode(65 + idx)}. {c}
                 </button>
               );
             })}
           </div>
         </div>
       ))}
     </div>

     {submitted && (
       <div className="mt-4 text-center">
         <div className="text-lg font-semibold">Your Score: {score}%</div
>
         <div className="text-sm text-[var(--muted)]">Great work! Keep goi
ng.</div>
       </div>
     )}
   </div>
 );
}
```

5.8 Pages (Home / Profile / Admin / Lesson)

Home — dashboard with XP, streak, recommendations shell.

Create **src/pages/Home.jsx**:

```
// src/pages/Home.jsx
import React, { useEffect } from "react";
import { Link } from "react-router-dom";
import XPBar from "../components/XPBar";
import { trackEvent } from "../lib/tracker";

/**
 * Landing dashboard with quick start and XP overview.
 */
export default function Home({ xp = 0, badges = [] }) {
  useEffect(() => {
    trackEvent("page_view", { page: "home" });
  }, []);

  return (
    <div className="max-w-6xl mx-auto px-4 py-6 space-y-6">
      {/* Hero */}
      <div className="card p-6 md:p-8">
        <h1 className="text-2xl md:text-3xl font-semibold">Welcome to <sp
an className="text-[var(--brand)]">HealthCoach AI</span></h1>
        <p className="mt-2 text-[var(--muted)]">Watch lessons, get AI sum
maries, take quizzes, earn XP, unlock badges, and get a personalized plan.
</p>
        <div className="mt-4 flex gap-3">
          <Link to="/lesson/lesson1" className="btn">Start Lesson 1</Link>
          <Link to="/profile" className="btn" style={{ background:"#2dd4bf"
}}>View Profile</Link>
        </div>
      </div>

      {/* XP & Badges */}
      <div className="grid md:grid-cols-2 gap-6">
        <XPBar xp={xp} />
        <div className="card p-4">
          <h3 className="font-semibold">Badges</h3>
          <div className="mt-2 flex gap-2 flex-wrap">
            {badges.length ? badges.map((b, i) => (
              <span key={i} className="badge">{b}</span>
            )) : <span className="text-sm text-[var(--muted)]">No badges ye
t. Earn XP to unlock!</span>}
          </div>
        </div>
      </div>

      {/* Recommendations shell (we'll fill from a Function call later) */}
      <div className="card p-4">
        <h3 className="font-semibold">Recommended Next</h3>
        <p className="text-[var(--muted)] text-sm">Take Lesson 1 to unlock
personalized suggestions.</p>
      </div>
    </div>
  );
}
```

Profile — shows XP/badges (local state for demo; you can persist to Firestore later):

Create **src/pages/Profile.jsx**:

```
// src/pages/Profile.jsx
import React, { useEffect } from "react";
import XPBar from "../components/XPBar";
import { trackEvent } from "../lib/tracker";

/**
 * Simple profile page. You can expand to show history/progress saved in Fi
restore.
 */
export default function Profile({ xp = 0, badges = [] }) {
  useEffect(() => { trackEvent("page_view", { page: "profile" }); }, []);

  return (
    <div className="max-w-3xl mx-auto px-4 py-6 space-y-6">
      <XPBar xp={xp} />
      <div className="card p-4">
        <h3 className="font-semibold">Your Badges</h3>
        <div className="mt-2 flex gap-2 flex-wrap">
          {badges.length ? badges.map((b, i) => (
            <span key={i} className="badge">{b}</span>
          )) : <span className="text-sm text-[var(--muted)]">None yet — co
mplete a quiz and watch more minutes!</span>}
        </div>
      </div>
    </div>
  );
}
```

Admin — quick event peek (for demo):

Create **src/pages/Admin.jsx**:

```
// src/pages/Admin.jsx
import React, { useEffect, useState } from "react";
import { trackEvent } from "../lib/tracker";
import { db } from "../lib/firebase";
import { collection, query, orderBy, limit, getDocs } from "firebase/firestor
e";
```

```
/**
 * Admin screen lists recent events. (In rules we’ll restrict reads as needed.)
 */
export default function Admin() {
  const [events, setEvents] = useState([]);

  useEffect(() => {
    trackEvent("page_view", { page: "admin" });
    (async () => {
      try {
        const q = query(collection(db, "events"), orderBy("timestamp", "des
c"), limit(50));
        const snap = await getDocs(q);
        setEvents(snap.docs.map(d => ({ id: d.id, ...d.data() })));
      } catch (e) {
        console.warn("Admin cannot read events due to rules (that’s okay in pr
od).");
      }
    })();
  }, []);

  return (
    <div className="max-w-5xl mx-auto px-4 py-6">
      <h2 className="text-xl font-semibold mb-4">Recent Events</h2>
      <div className="card p-4 overflow-x-auto">
        <table className="w-full text-sm">
          <thead className="text-left text-[var(--muted)]">
            <tr><th>Type</th><th>Session</th><th>Page</th><th>Meta</th>
</tr>
          </thead>
          <tbody>
            {events.map(e => (
              <tr key={e.id} className="border-t border-white/10">
                <td className="py-2">{e.eventType}</td>
                <td>{e.sessionId}</td>
                <td>{e.pageUrl}</td>
                <td><pre className="max-w-[360px] whitespace-pre-wrap">{J
SON.stringify(e.metadata)}</pre></td>
              </tr>
            ))}
          </tbody>
        </table>
      </div>
    </div>
  );
}
```

Lesson — the star: YouTube player, AI summary, AI quiz, XP system, recommendations button.

Create **src/pages/Lesson.jsx**:

```
// src/pages/Lesson.jsx
import React, { useEffect, useMemo, useState } from "react";
import { useParams } from "react-router-dom";
import VideoPlayer from "../components/VideoPlayer";
import Quiz from "../components/Quiz";
import XPBar from "../components/XPBar";
import { trackEvent } from "../lib/tracker";
import axios from "axios";
import { XP_PER_MINUTE_WATCHED, badgesFor, levelFromXP } from "../li
b/points";

/**
 * Lesson page for the Public Health playlist.
 * For Lesson 1, we start with the first video from the provided playlist.
 * You can expand to a full playlist browser later.
 */
export default function Lesson() {
  const { id } = useParams(); // e.g., "lesson1"
  // Provided by you: Crash Course Public Health playlist, first video ID:
  const playlistId = "PL8dPuuaLjXtPjQj_LcJ0Zvj-VI3ssIJyF";
  const firstVideoId = "PjdJ19ugXzQ";

  const [videoId, setVideoId] = useState(firstVideoId);
  const [summary, setSummary] = useState("");
  const [quiz, setQuiz] = useState([]);
  const [xp, setXP] = useState(0);
  const [badges, setBadges] = useState([]);

  // Award XP helper and update badges on the fly
  const awardXP = (delta) ⇒ {
    setXP((x) ⇒ {
      const nx = Math.max(0, Math.floor(x + delta));
      setBadges(badgesFor({ totalMinutes: nx / XP_PER_MINUTE_WATCHED,
totalCorrect: 0 }));
      return nx;
    });
  };

  // Every second watched → XP (5 XP/minute = ~0.083 XP/sec)
  const handleSecondsWatched = (secs) ⇒ {
    const xpPerSec = XP_PER_MINUTE_WATCHED / 60;
    const gained = secs * xpPerSec;
    if (gained > 0) awardXP(gained);
  };

  // On mount: track page view
  useEffect(() ⇒ {
    trackEvent("page_view", { page: "lesson", lessonId: id, playlistId, videoId
});
  }, [id, playlistId, videoId]);

  // Call Cloud Function to summarize current video transcript
  const summarizeVideo = async () ⇒ {
    setSummary("Summarizing...");
    try {
      const res = await axios.post("/api/summarize-video", { videoId });
      setSummary(res.data.summary || "(No summary)");
      trackEvent("ai_summary_request", { videoId });
    } catch (e) {
      setSummary("Failed to summarize (maybe no transcript).");
    }
  };
```

```
 // Generate a quiz from transcript and render it
 const generateQuiz = async () => {
  try {
   const res = await axios.post("/api/generate-quiz", { videoId });
   setQuiz(res.data.questions || []);
   trackEvent("ai_quiz_generated", { videoId, count: res.data.questions?.length || 0 });
  } catch (e) {
   setQuiz([]);
   alert("Failed to generate quiz (try again or pick another video).");
  }
 };

 // XP for quiz correctness is awarded by <Quiz />
 const handleQuizXP = (xp) => awardXP(xp);

 const lvl = levelFromXP(xp);

 return (
  <div className="max-w-6xl mx-auto px-4 py-6 space-y-6">
   {/* Header row: title + controls */}
   <div className="flex items-start justify-between gap-4">
    <div>
     <h2 className="text-2xl font-semibold">Lesson 1 · Public Health</h2>
     <div className="text-[var(--muted)] text-sm">Crash Course playlist • Video {videoId}</div>
    </div>
    <div className="flex gap-2">
     <button className="btn" onClick={summarizeVideo}>AI Summarize</button>
     <button className="btn" style={{ background:"#22c55e" }} onClick={generateQuiz}>Generate Quiz</button>
    </div>
   </div>

   {/* Main grid: player + right rail */}
```

```
    <div className="grid md:grid-cols-3 gap-6">
      <div className="md:col-span-2 space-y-6">
        <VideoPlayer videoId={videoId} onSecondsWatched={handleSecond
sWatched} />

        {summary && (
          <div className="card p-4">
            <h3 className="font-semibold">AI Summary</h3>
            <pre className="mt-2 whitespace-pre-wrap">{summary}</pre>
          </div>
        )}

        {/* Quiz renders if available */}
        <Quiz questions={quiz} videoId={videoId} onAwardXP={handleQuizX
P} />
      </div>

      {/* Right rail: XP + quick plan (placeholder) */}
      <div className="space-y-6">
        <div className="card p-4">
          <h3 className="font-semibold">Your Progress</h3>
          <div className="text-sm text-[var(--muted)]">Level {lvl} • {Math.r
ound(xp)} XP</div>
        </div>

        <div className="card p-4">
          <h3 className="font-semibold">Suggested Plan</h3>
          <p className="text-sm text-[var(--muted)]">
            Watch the current video, read the summary, take the quiz.
            Get {XP_PER_MINUTE_WATCHED} XP/min for watching; bonus XP
for correct answers.
          </p>
          <p className="text-sm text-[var(--muted)] mt-2">
            After you finish, we'll recommend the next video based on your per
formance.
          </p>
        </div>
      </div>
    </div>
   </div>
  );
}
```

5.9 Root App + Auth form + Navbar

Create **src/App.jsx**:

```
// src/App.jsx
import React, { useEffect, useState } from "react";
import { BrowserRouter, Routes, Route, Link } from "react-router-dom";
import Navbar from "./components/Navbar";
import Home from "./pages/Home";
import Lesson from "./pages/Lesson";
import Profile from "./pages/Profile";
import Admin from "./pages/Admin";
import { auth } from "./lib/firebase";
import {
  createUserWithEmailAndPassword,
  signInWithEmailAndPassword,
  signOut
} from "firebase/auth";

/**
 * Root app: handles auth + routes + navbar.
 * We keep auth super simple (email/password) and show status in nav.
 */
export default function App() {
  const [user, setUser] = useState(null);     // current user object (or null)
  const [email, setEmail] = useState("");     // email input state
  const [password, setPassword] = useState("");// password input state
  const [mode, setMode] = useState("login");  // "login" or "register"

  // Subscribe to auth state changes (login/logout)
  useEffect(() => {
    return auth.onAuthStateChanged(setUser);
  }, []);
  // Register a new user with email+password
  const register = async (e) => {
    e.preventDefault();
    await createUserWithEmailAndPassword(auth, email, password);
  };

  // Login an existing user
  const login = async (e) => {
    e.preventDefault();
    await signInWithEmailAndPassword(auth, email, password);
  };

  // Logout
  const logout = async () => {
    await signOut(auth);
  };

  // Right side of navbar: auth status + actions
  const right = user ? (
    <div className="flex items-center gap-3">
      <span className="text-sm text-[var(--muted)] hidden md:inline">{use
r.email}</span>
      <button className="btn" onClick={logout}>Logout</button>
    </div>
  ) : (
    <form onSubmit={mode === "login" ? login : register} className="flex i
tems-center gap-2">
      <input
        className="px-3 py-2 rounded-xl bg-white/10 placeholder:text-whit
e/50"
        placeholder="Email"
        type="email" value={email} onChange={e=>setEmail(e.target.value)} r
equired
      />
      <input
        className="px-3 py-2 rounded-xl bg-white/10 placeholder:text-whit
e/50"
```

```
    placeholder="Password"
    type="password" value={password} onChange={e=>setPassword(e.ta
rget.value)} required
   />
   <button className="btn" type="submit">{mode === "login" ? "Sign I
n" : "Register"}</button>
   <button type="button" className="badge" onClick={()=>setMode(mod
e==="login"?"register":"login")}>
    {mode === "login" ? "Need an account?" : "Have an account?"}
   </button>
  </form>
 );

 return (
  <BrowserRouter>
   <Navbar right={right} />
   <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/lesson/:id" element={user ? <Lesson /> : <PleaseLogin
/>} />
    <Route path="/profile" element={user ? <Profile /> : <PleaseLogin />}
/>
    <Route path="/admin" element={<Admin />} />
   </Routes>
   <footer className="max-w-6xl mx-auto px-4 py-10 text-xs text-[var(--
muted)]">
    Built with Firebase + OpenAI • © You
   </footer>
  </BrowserRouter>
 );
}

// Small component shown when unauthenticated user tries to access prote
cted routes
function PleaseLogin() {
 return (
  <div className="max-w-xl mx-auto px-4 py-10 text-center card">
   <h3 className="font-semibold">Please sign in</h3>
```

```
    <p className="text-sm text-[var(--muted)] mt-1">Create an account o
r sign in from the top bar.</p>
   </div>
  );
}
```

6) Cloud Functions: secure AI (summarize transcript + generate quiz + recommend)

We'll implement these HTTPS endpoints:

- `POST /api/summarize-video` → fetch YouTube transcript (if available) → call OpenAI → return summary
- `POST /api/generate-quiz` → same transcript → return 5 MCQs (ID, question, 4 choices, correct index)
- (Optional) `POST /api/recommend` → read user's events and quiz attempts (server-side) and recommend next video(s)

6.1 Install function deps

Open a new PowerShell **in the `frontend/functions` folder**:

```
cd functions
npm install node-fetch@2 youtube-transcript
# (The init already installed firebase-admin, firebase-functions)
```

We'll set your OpenAI key later.

6.2 Configure Hosting rewrites (frontend → functions)

Open **firebase.json** (at `frontend/firebase.json`) and ensure the Hosting rewrites include the functions:

```
{
  "hosting": {
    "public": "dist",
    "rewrites": [
      { "source": "/api/**", "function": "api" },
      { "source": "**", "destination": "/index.html" }
    ]
  }
}
```

6.3 Functions code

Open **functions/index.js** and replace with the following:

```
// functions/index.js

// Firebase Functions runtime + Admin SDK
const functions = require("firebase-functions");
const admin = require("firebase-admin");

// Express to mount multiple endpoints under one function
const express = require("express");
const cors = require("cors");
// node-fetch v2 for HTTP calls to OpenAI
const fetch = require("node-fetch");
// Lightweight transcript fetcher (no API key; depends on captions availabili
ty)
const { YoutubeTranscript } = require("youtube-transcript");

admin.initializeApp(); // enables admin.auth(), admin.firestore(), etc.

const app = express();              // create an Express app
app.use(cors({ origin: true }));    // allow cross-origin from our site
app.use(express.json());            // parse JSON bodies

// Middleware: verify Firebase ID token (Authorization: Bearer <token>)
async function verifyToken(req, res, next) {
  try {
    const header = req.headers.authorization || "";
    const match = header.match(/^Bearer (.+)$/);
    if (!match) return res.status(401).json({ error: "Missing Authorization Bea
rer token" });
    const decoded = await admin.auth().verifyIdToken(match[1]);
    req.user = decoded; // attach decoded user info
    next();
  } catch (e) {
    return res.status(401).json({ error: "Invalid ID token" });
  }
}

// Get OpenAI key from Functions config (set via CLI)
const OPENAI_KEY = functions.config().openai?.key;
if (!OPENAI_KEY) {
  console.warn('OpenAI key missing. Set it with: firebase functions:config:s
et openai.key="sk-..."');
}

/**
 * Helper: fetch transcript for a given YouTube videoId.
 * If no transcript exists, throws.
 */
async function getTranscriptText(videoId) {
  // YoutubeTranscript.fetchTranscript returns an array of { text, duration, of
fset }
  const segments = await YoutubeTranscript.fetchTranscript(videoId);
  // Join all text segments into one long string
  return segments.map(s => s.text).join(" ");
}

/**
 * Helper: call OpenAI Chat Completions
 */
async function openaiChat(messages, max_tokens = 600, temperature = 0.
3) {
  const resp = await fetch("https://api.openai.com/v1/chat/completions", {
    method: "POST",
    headers: {
      "Authorization": `Bearer ${OPENAI_KEY}`,
      "Content-Type": "application/json"
    },
    body: JSON.stringify({
      model: "gpt-4o-mini",    // good balance of cost/quality
      messages,
      max_tokens,
      temperature
    })
  });
  if (!resp.ok) {
    const errText = await resp.text();
    throw new Error(`OpenAI error: ${resp.status} ${errText}`);
  }
  const data = await resp.json();
  return data.choices?.[0]?.message?.content || "";
}

/**
 * POST /summarize-video
 * Body: { videoId }
 * Returns: { summary: "bullet1\nbullet2\n..." }
 */
app.post("/summarize-video", verifyToken, async (req, res) => {
  try {
    const { videoId } = req.body || {};
    if (!videoId) return res.status(400).json({ error: "videoId required" });

    const text = await getTranscriptText(videoId);

    // Ask OpenAI for a tight, bulleted summary
    const content = await openaiChat([
      { role: "system", content: "You summarize educational videos. Output 5
-8 crisp bullet points separated by newline."},
      { role: "user", content: `Summarize the following transcript:\n\n${text}` }
    ], 500, 0.2);

    res.json({ summary: content.trim() });
  } catch (e) {
    console.error(e);
    res.status(500).json({ error: "Failed to summarize (no transcript or API er
ror?)" });
  }
});

/**
 * POST /generate-quiz
 * Body: { videoId }
 * Returns: { questions: [{ id, q, choices:[...], answerIndex }] }
 */
app.post("/generate-quiz", verifyToken, async (req, res) => {
  try {
    const { videoId } = req.body || {};
    if (!videoId) return res.status(400).json({ error: "videoId required" });

    const text = await getTranscriptText(videoId);

    // Ask OpenAI to output STRICT JSON for 5 MCQs
    const content = await openaiChat([
      { role: "system", content: "Create multiple-choice questions from transc
ript. Output ONLY JSON."},
      { role: "user", content:
`From the transcript below, generate 5 MCQs. Strict JSON:
{"questions":[
  {"id":"q1","q":"Question?","choices":["A","B","C","D"],"answerIndex":0},
  {"id":"q2","q":"Question?","choices":["A","B","C","D"],"answerIndex":2}
]}
Transcript:
${text}`
      }
    ], 800, 0.4);

    // Parse JSON (handle occasional extra text)
    let parsed;
    try {
      parsed = JSON.parse(content);
    } catch {
      const s = content.indexOf("{");
      const e = content.lastIndexOf("}");
      parsed = JSON.parse(content.slice(s, e + 1));
    }

    // Optionally store quiz for reuse: admin.firestore().collection("quizzes").
doc(videoId)...
    res.json({ questions: parsed.questions || [] });
  } catch (e) {
    console.error(e);
    res.status(500).json({ error: "Failed to generate quiz" });
  }
});

/**
 * (Optional) POST /recommend
 * Reads recent events/quizAttempts for the user and picks next video.
 * For demo, we recommend the next in playlist if score >=70, else rewatch
current.
 */
app.post("/recommend", verifyToken, async (req, res) => {
  try {
    const { lastVideoId, lastScore } = req.body || {};
    // Simple rule-based logic for demo
    const recommendation = lastScore >= 70
      ? { action: "next_in_playlist", reason: "Good score. Move forward." }
      : { action: "rewatch_and_retry", reason: "Score below 70. Rewatch sum
mary and retry quiz." };

    res.json({ recommendation });
  } catch (e) {
    console.error(e);
    res.status(500).json({ error: "Failed to recommend" });
  }
});

// Export a single HTTPS function named "api" that hosts all endpoints abo
ve
exports.api = functions.https.onRequest(app);
```

Set your OpenAI key (PowerShell inside `frontend`)

```
firebase functions:config:set openai.key="sk-REPLACE_ME"
```

> You can use OpenAI or any provider you prefer—this wiring is generic.

---

7) Secure Firestore rules (assignment-ready)

Open **firestore.rules** and paste:

```
rules_version = '2';
service cloud.firestore {
 match /databases/{database}/documents {

   // Public lesson content if you later store it here
   match /content/{id} {
    allow read: if true;
    allow write: if request.auth != null && request.auth.token.admin == true;
   }

   // Clickstream events: allow only creates by signed-in users
   match /events/{eventId} {
    allow create: if request.auth != null;
    allow read: if false; // keep raw events private (admin via server)
   }

   // Quiz attempts: user can create; read is limited
   match /quizAttempts/{attemptId} {
    allow create: if request.auth != null;
    allow read: if request.auth != null && (
     request.auth.token.admin == true ||
     request.auth.uid == resource.data.userId
    );
    allow update, delete: if false;
    }
  }
}
```

> For simplicity, our client writes events anonymously; if you want userId on attempts, add it in the client (auth.currentUser.uid) and adjust rules accordingly.

---

8) Wire Hosting → Functions (already done) & build/deploy

From **frontend**:

```
# Build the React app for production (outputs to dist/)
npm run build

# Deploy Functions + Hosting
firebase deploy --only functions,hosting
```

It prints a public URL like:

`https://<your-project-id>.web.app` ← **this is your live site**.

---

9) Seed Lesson 1 (Public Health) — minimal

We already hardcoded **Lesson 1** to use your playlist's **first video** (ID `PjdJ19ugXzQ`).

Just visit the deployed app → **Sign up** → click **Lesson 1** → press **AI Summarize** then **Generate Quiz**.

You should get:

- Summary from the real transcript (if captions are available)
- Fresh quiz questions

- XP ticking up as you watch; extra XP when you submit the quiz

> If a video has no transcript, the summarize/quiz calls will say “Failed.” Pick another video in that Crash Course playlist and set firstVideoId in Lesson.jsx to the new ID, or add a simple dropdown to switch.

---

10) Nice-to-haves you can add quickly

- **Playlist browser**: store an array of `{title, videoId}` and render as a sidebar list; clicking updates `videoId`.
- **Persist XP**: create a `profiles/{uid}` doc (`xp`, `badges[]`, `streak`) and read/write it when XP changes.
- **Streaks**: when user gets XP on a new date, increment streak.
- **Admin analytics:** create a Cloud Function cron to aggregate events (or query events server-side when requested).
- **Recommendations**: in `/recommend`, actually read last quiz score & watch time for that user and pick next video by index.

Recap of what you now have

- ✅ Modern UI (Tailwind) with a real navbar, cards, progress bar
- ✅ Auth (email/password in the top bar)
- ✅ YouTube player with clickstream tracking for play/pause/complete + watch-time → **XP**
- ✅ **AI Summary** of the exact video you’re watching (from transcript)
- ✅ **AI-generated quiz** with scoring and XP rewards
- ✅ **Gamification**: XP, levels, badges, visual XP bar
- ✅ **Clickstream** stored in Firestore (`events`, `quizAttempts`)
- ✅ **Deployed** on Firebase Hosting with secure Cloud Functions (OpenAI key never exposed)
- ✅ Clear, beginner-friendly project you can demo and extend

Troubleshooting quick tips

- If **/api/** calls 401, you're probably not logged in; sign in first so the browser includes a valid Firebase ID token.
- If **summarize/quiz** fails, that video might not have a transcript. Try another video ID or copy a transcript manually into the function to test.
- If deploy fails on functions, run inside `frontend/functions`:

  ```
  npm install
  ```

  and retry deploy.

If you'd like, I can add a **playlist sidebar** and **profile persistence** next (XP/badges saved per user in Firestore) so your progress survives page refreshes and shows up on the Profile page.