Netcentric: Step■by■Step Build Documentation

A browser-based tutoring platform with real-time live class updates via Firebase Firestore

Date: August 12, 2025

Project Summary

Netcentric is a lightweight web application consisting of two main pages: **admin.html** (administration) and **index.html** (user-facing). Administrators paste a YouTube link or ID in the admin panel; the app extracts a clean YouTube video ID and saves it to Firestore. The user page listens for real-time updates to that Firestore document and embeds the latest live class video automatically, without refresh.

Goals

1. Enable non-technical admins to update the live class YouTube video easily.
2. Propagate video updates in real-time to all connected users (no page refresh).
3. Keep the project simple enough to run on localhost with minimal setup.

Technology Stack

- **Frontend:** Vanilla HTML, CSS, and JavaScript
- **Backend/DB:** Firebase Firestore (client SDK)
- **Realtime:** Firestore onSnapshot listeners

High-Level Architecture

Firestore stores a single configuration document at **config/liveClass** with field **videoId**. The admin page writes this value; the user page reads it via a real-time listener and updates the YouTube *iframe* to https://www.youtube.com/embed/<videoId>.

Step 1 — Firebase Project Setup

1. Create or open a Firebase project in the Firebase Console.
2. Enable Firestore: Firestore Database → Create Database (use test mode for local development).
3. Copy your web app configuration: Project Settings → General → Your Apps → Firebase SDK snippet.

Example Firebase Config Snippet

```
const firebaseConfig = {
  apiKey: "YOUR_API_KEY",
  authDomain: "YOUR_PROJECT.firebaseapp.com",
  projectId: "YOUR_PROJECT",
  storageBucket: "YOUR_PROJECT.appspot.com",
  messagingSenderId: "SENDER_ID",
  appId: "APP_ID"
};

firebase.initializeApp(firebaseConfig);
const db = firebase.firestore();
```

Step 2 — Firestore Rules (Local Development)

For localhost development, it's convenient to allow reads/writes to the configuration document. Use permissive rules during development only, then restrict before production.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /config/{docId} {
      allow read, write: if true; // DEV ONLY
    }
  }
}
```

Step 3 — Admin Panel Implementation (admin.html)

1 Provide an input field for a YouTube URL or a raw video ID.

2 On submit, normalize the input to extract a clean video ID (supports watch/shorts/embed/youtu.be formats).

3 Write the clean ID to Firestore: collection config, document liveClass, field videoId.

YouTube ID Extraction Helper (used before saving):

```
function extractYouTubeId(input) {
  if (!input) return null;
  input = input.trim();

  // Accept plain IDs if they look like YouTube IDs
  const cleanIdMatch = input.match(/^[a-zA-Z0-9_-]{6,}$/);
  if (cleanIdMatch) return cleanIdMatch[0];

  try {
    const url = new URL(input);
    const host = url.hostname.replace(/^www\./, "");

    if (host === "youtu.be") {
      const seg = url.pathname.split("/").filter(Boolean)[0];
      if (seg) return seg;
    }

    if (host === "youtube.com" || host === "m.youtube.com" || host === "music.youtube.com") {
      const v = url.searchParams.get("v");
      if (v) return v;

      const shortsMatch = url.pathname.match(/\/shorts\/([a-zA-Z0-9_-]+)/);
      if (shortsMatch) return shortsMatch[1];

      const embedMatch = url.pathname.match(/\/embed\/([a-zA-Z0-9_-]+)/);
      if (embedMatch) return embedMatch[1];
    }
  } catch (e) {}

  const idFromQuery = input.match(/[?&]v=([a-zA-Z0-9_-]+)/);
  if (idFromQuery) return idFromQuery[1];

  return null;
}
```

Step 4 — User Page Implementation (index.html)

1 Initialize Firebase using the same configuration as the admin page.

2 Register an onSnapshot listener on the document config/liveClass.

3 When videoId changes, normalize it defensively and update the iframe source to the embed URL.

```
db.collection("config").doc("liveClass").onSnapshot((doc) => {
  if (!doc.exists) return;
  let videoId = doc.data().videoId;
  videoId = extractYouTubeId(videoId); // defensive normalization

  const iframe = document.getElementById("liveClassVideo");
  if (videoId && /^[a-zA-Z0-9_-]{6,}$/.test(videoId)) {
    iframe.src = `https://www.youtube.com/embed/${videoId}`;
  }
});
```

Step 5 — Running Locally (Localhost)

1 Use a local web server (XAMPP, VS Code Live Server, or Node http-server). Avoid opening files via file://.

2 Place the Netcentric folder under your server root (e.g., XAMPP htdocs).

3 Visit the admin page and user page via http://localhost/...

Troubleshooting

- **Video not updating:** Verify Firebase config in both pages; confirm Firestore rules permit read/write.
- **Broken embed:** Ensure only a valid YouTube ID is stored; avoid storing full URLs in Firestore.
- **No real-time updates:** Confirm the document path is config/liveClass and that onSnapshot is attached.
- **Local file access:** Always use http://localhost rather than file:// to avoid SDK restrictions.

Step 6 — Hardening for Production

1 Restrict Firestore writes to authenticated admins only.

2 Validate inputs server-side if you later add a backend; keep client checks as a UX layer.

3 Consider adding an admin-side preview to verify the video before saving.

Example Production■ready Rule:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /config/{docId} {
      allow read: if true;
      allow write: if request.auth != null && request.auth.token.admin == true;
    }
  }
}
```

Appendix — File Map

```
Netcentric/
  admin.html          # Admin panel (paste YouTube link/ID, saves to Firestore)
  index.html          # User page (reads Firestore and embeds video)
  assets/
    logo/
      De-briliant-logo.jpg
```

End of Document.