Freestyle prompt app

The way it works is it’s going to give you a random prompt as to how to guide your freestyle round.

For example if the prompt “bounce” comes up, you will focus on your bounce for the round

Another example would be “waves” focusing on your waving wether that be arm waves. Along with the prompt will be a tips button which will help you better understand and execute the concept, a more advanced version would be sending you youtube links to help you visualise

Another feature of the app would have random music generator, you also have the choice of the genres so its not completely random.

How do I actually learn this and not just rely on AI

Remember to use AI as a boost not a reliable drug to keep my thinking sedated

Different between typescript and javascript

APIs and how to implement them

what foundations do I need to learn

Sometimes I hate AI because it just pops up this app I described in seconds which would’ve taken me weeks to learn and execute

A screenshot of a computer

AI-generated content may be incorrect.

This is the goal, now lets reverse engineer learning it

**Freestyle Prompt App — Study Path (Beginner → Ship)**

**Milestone 0 — Setup & “Hello Vite”**

**You’ll learn:** Node/npm, VS Code, Git, Vite project.  
**Do:**

1. Install Node LTS + VS Code.
2. npm create vite@latest freestyle-prompt -- --template react → cd → npm i → npm run dev.
3. Git init + first commit; create GitHub repo and push.  
   **Checkpoint:** App shows the Vite React starter in the browser.

**Mini-drills (10–15 min each):**

* Change the page title and main heading.
* Add a new <button> that console.logs “clicked”.

**Milestone 1 — Web Fundamentals (just enough)**

**You’ll learn:** HTML structure, CSS basics, JS essentials (let/const, arrays, objects, loops, functions), DOM events.  
**Do:**

* In a new folder (not React), make a plain HTML page with a <div id="prompt"> and a “Spin” button.
* Write spin() that picks a random word from ["Bounce","Waves","Groove"] and updates the div.  
  **Checkpoint:** Button changes the prompt every click.

**Mini-drills:**

* Add a second button: “Show Tips” → toggles a hidden <ul> of tips.
* Add a “Genre” chip bar (House, Hip Hop…) that toggles an “active” class.

**Milestone 2 — React Fundamentals**

**You’ll learn:** components, props, useState, rendering lists, conditional UI.  
**Do (in your Vite app):**

* Replace starter with a small component: PromptCard with title, “Tips” toggle, and a “Spin Prompt” button using useState.
* Render a list of tips from an array with map, each tip keyed by index.  
  **Checkpoint:** You can spin between 3 prompts and toggle tips without page reloads.

**Mini-drills:**

* Add a ResourceLink list that only shows when “Resources” is toggled.
* Create a GenreChips component with internal selected state (array of strings).

**Milestone 3 — Styling with Tailwind (to match the UI you saw)**

**You’ll learn:** Tailwind setup in Vite, utility classes, responsive spacing, rounded corners, shadows.  
**Do:**

1. Install Tailwind + PostCSS + config in your Vite app (standard steps from Tailwind docs).
2. Replace inline styles with Tailwind classes for cards, buttons, and layout.  
   **Checkpoint:** Your UI looks like simple cards with rounded corners, spacing, and hover styles.

**Mini-drills:**

* Make buttons have an active state (pressed feel).
* Create a responsive container (max-w-3xl mx-auto p-6).

**Milestone 4 — App State & Persistence**

**You’ll learn:** composing components, lifting state up, localStorage.  
**Do:**

* Add “Source” selector: Curated | YouTube | Spotify.
* When a track is rolled, save {prompt, source, title, url, time} to localStorage["sessions"].
* Add a tiny “History (last 5)” under the player.  
  **Checkpoint:** Refreshing the page keeps your last 5 rolls.

**Mini-drills:**

* Add “Clear history” button.
* Add a “Combo” toggle to spin 2 prompts at once.

**Milestone 5 — Fetching External APIs (safe & simple)**

**You’ll learn:** fetch, async/await, errors, environment variables.  
**Do (YouTube first):**

* Create a “Roll Track” function that, when source=YouTube, calls the **YouTube Data API v3 Search** endpoint with query like "<genre> instrumental <prompt>".
* Use Vite env keys: VITE\_YT\_API\_KEY (stored in .env, never committed).
* Show loading and error states; if error, fall back to curated links.  
  **Checkpoint:** You can roll tracks from YouTube (new tab link), and errors don’t crash the app.

**Mini-drills:**

* Always filter for videoEmbeddable=true.
* Randomise between top 10 results.

**Milestone 6 — Spotify Integration (concept → working)**

**You’ll learn:** auth flows, **why secrets don’t belong in the browser**, serverless function basics.  
**Two steps:**

1. **Dev-only (quick & dirty):** use Client Credentials from the browser to understand the flow (what you already have in the canvas code).
2. **Production-ready:** move token exchange to a serverless function (Netlify/Cloudflare). Frontend calls /api/spotify-token → gets short-lived token → calls Spotify Search.

**Do:**

* Implement dev-only approach to prove it works (roll Spotify tracks, show title + artist, preview if available).
* Then migrate to a serverless function and remove secrets from the client.  
  **Checkpoint:** Spotify option works with a serverless token, no secrets in client code.

**Mini-drills:**

* Add country market=GB.
* Prefer instrumentals: add q="genre:house NOT vocals" (imperfect but okay).

**Milestone 7 — Motion & Polish**

**You’ll learn:** framer-motion basics, micro-interactions, simple accessibility checks.  
**Do:**

* Install framer-motion and animate prompt card on spin and button taps.
* Add focus styles for keyboard users; check button labels for screen readers.  
  **Checkpoint:** The app feels snappy; tabbing around is usable.

**Mini-drills:**

* Animate genre chip selection (scale/opacity).
* Add a 60s round timer with a start/stop button and small beep (HTML5 audio).

**Milestone 8 — Ship It**

**You’ll learn:** deploy choices + envs.  
**Option A (easiest with APIs):** **Netlify**

* Connect your GitHub repo.
* Add environment variables in Netlify Dashboard (YT key, and Spotify vars used by your serverless function).
* Put your function in netlify/functions/spotify-token.js.  
  **Option B:** GitHub Pages (fine for the curated-only version or if you build locally with envs baked in).

**Checkpoint:** Live URL you can share, working with YouTube and Spotify (via serverless).

**Stretch Goals (pick one)**

* **PWA installable:** manifest + service worker → “Install app” on mobile.
* **Session analytics:** per-prompt frequency; ensure you rotate concepts evenly.
* **Teacher mode:** preload class-friendly clean tracks; QR code to share the round.

**What to send me at each checkpoint (so I can coach you)**

* **0:** Screenshot of Vite running locally.
* **1:** Small GIF/video of the vanilla “Spin Prompt” page.
* **2:** Repo link with React PromptCard.
* **3:** Screenshot of Tailwind-styled card/buttons.
* **4:** JSON dump from localStorage["sessions"].
* **5:** Short clip rolling a YouTube track (show loading → success/fallback).
* **6:** Netlify functions folder + live URL showing Spotify working.
* **7:** 3-second clip of the animated prompt card.
* **8:** Public URL + a 30-sec demo video.

**Quick reference (what to look up as you go)**

* “Array methods: map/filter/sort”, “DOM events vs React events”, “React useState”, “Vite env variables (import.meta.env)”, “Tailwind install Vite”, “fetch + async/await + try/catch”, “YouTube Data API v3 Search”, “Spotify Client Credentials”, “Netlify functions”.

Want me to tailor the pacing? **One quick question:** how many hours per week can you realistically put into this (5, 8, or 12+)?  
Tell me that, and I’ll turn this into a week-by-week plan with exact tasks per session.