DeltaForceOS

CODEX ROUTE / BEGINNER MODULE 08

Voice Agent Basics: Codex Route

BUILD OUTCOME

Understand the voice pipeline and build a tiny local demo before touching real customer calls.

BEST FOR

Use this PDF when you want exact prompts and clicks for this route instead of general theory.

MODEL TIER

Mid for live conversation; light for transcript cleanup; heavy only for post-call analysis.

FIRST INSTRUCTION

```
Inspect this repo for the Voice Agent Basics build.
Explain the files a beginner needs: README.md, package.json, src, public, scripts, .env.local, and any
Supabase files.
Do not edit yet. Give me the smallest safe build plan.
```

Before using the agent, make the workspace boring and clear.

01 Open the project folder. If you do not have one yet, create a new folder on Desktop and name it after the lesson.

02 Install Node.js if npm is not available. Beginners can check by opening Terminal and typing npm -v.

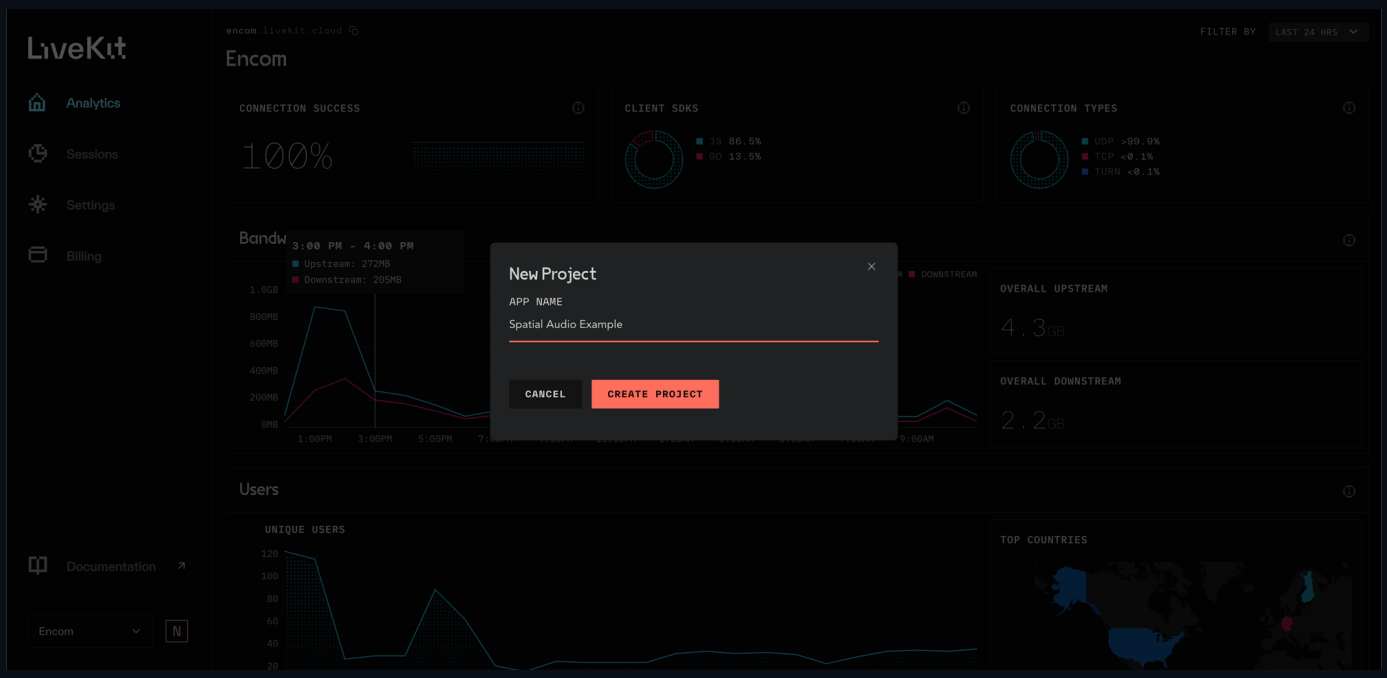
03 Create .env.local in the project root and add only the keys this lesson needs.

04 Confirm .env.local is listed in .gitignore so secrets do not go to GitHub.

05 Open the official dashboard for the first two tools in this lesson and create one tiny test project.

06 Decide where proof will live: Supabase row, screenshot, live URL, or Skool post.

LiveKit project: click path for this route.



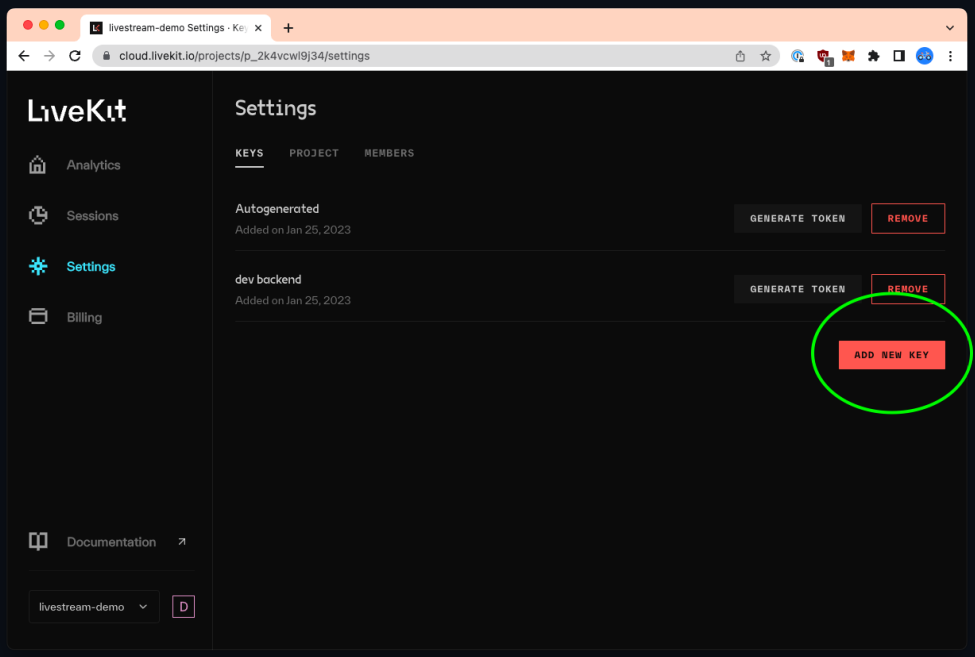
WHERE TO CLICK

Go to cloud.livekit.io. Sign in. Click the project switcher or New Project. Type a simple name like voice-agent-demo. Click Create Project.

ENV NAMES

```
No env value yet. This step creates the project that will give you the URL and keys.
```

LiveKit keys: click path for this route.



WHERE TO CLICK

Inside the project, click Settings in the left sidebar. Stay on the Keys tab. Click Add New Key. Copy the API key and secret.

ENV NAMES

`LIVEKIT_API_KEY, LIVEKIT_API_SECRET`

Copy this box exactly, then let the agent ask clarifying questions only if needed.

INSPECT PROMPT

```
Inspect this repo for the Voice Agent Basics build.
Explain the files a beginner needs: README.md, package.json, src, public, scripts, .env.local, and any
Supabase files.
Do not edit yet. Give me the smallest safe build plan.
```

BEGINNER CHECK

If the agent proposes a huge rebuild, ask it to shrink the scope to the smallest artifact that proves this lesson.

DO NOT PASTE

Do not paste service-role keys, payment secrets, signing keys, or private client data into the chat.

Copy this box exactly, then let the agent ask clarifying questions only if needed.

BUILD PROMPT

```
Implement the smallest useful version of Voice Agent Basics.
Outcome: Understand the voice pipeline and build a tiny local demo before touching real customer calls.
Tools allowed: LiveKit, Deepgram, ElevenLabs, OpenAI, Claude, Twilio, Bland.ai, Supabase
Keep secrets server-side. Add code comments only where a beginner would get lost.
```

BEGINNER CHECK

If the agent proposes a huge rebuild, ask it to shrink the scope to the smallest artifact that proves this lesson.

DO NOT PASTE

Do not paste service-role keys, payment secrets, signing keys, or private client data into the chat.

TYPE THIS INTO FILES

```
# README.md lesson block
Lesson: Voice Agent Basics
Outcome: Understand the voice pipeline and build a tiny local demo before touching real customer calls.
Tools: LiveKit, Deepgram, ElevenLabs, OpenAI, Claude, Twilio, Bland.ai, Supabase

# .env.local placeholders
NEXT_PUBLIC_SUPABASE_URL=your_value_here
NEXT_PUBLIC_SUPABASE_ANON_KEY=your_value_here
OPENAI_API_KEY=server_only
ANTHROPIC_API_KEY=server_only

# Verify
npm run build
```

WHERE IT GOES

README.md gets instructions. .env.local gets real local values. Vercel gets production values. Source files get implementation.

Basic English. Click by click. Do not skip ahead.

01 Open chrome and go to livekit.io. Click Sign up. Choose Continue with Google.

02 Click Create project. Name it voice-test. Pick the closest region. Click Create.

03 On the project dashboard, copy three values: WebSocket URL, API Key, API Secret.

04 Open .env.local. Add four lines: LIVEKIT_URL, LIVEKIT_API_KEY, LIVEKIT_API_SECRET, and NEXT_PUBLIC_LIVEKIT_URL (set to the same WebSocket URL).

05 Open chrome and go to deepgram.com. Sign up. Copy your API key. Add DEEPGRAM_API_KEY to .env.local.

06 Open chrome and go to elevenlabs.io. Sign up. Click your profile, then Profile + API key. Copy the key. Add ELEVENLABS_API_KEY to .env.local.

Finish the build. The last step always posts proof in Skool.

01 Create a file in your project root called call-script.md. Paste: I am a friendly receptionist for [BUSINESS]. I greet, qualify, and book a call. I never give legal, medical, or financial advice. If asked something I cannot answer, I say I will have a human follow up.

02 In Terminal, type npm install livekit-client livekit-server-sdk and press Enter.

03 Build a tiny page at src/app/voice/page.tsx that joins a LiveKit room. Copy the starter snippet from docs.livekit.io.

04 Run npm run dev. Open chrome to http://localhost:3000/voice. Allow microphone access in the chrome popup.

05 Speak into your mic. The page should show your audio level moving. If silent, check chrome address bar for a blocked mic icon and click Allow.

06 Wire the audio through Deepgram for transcription, Claude for the reply, and ElevenLabs for speech. Run five test calls. Save each transcript to a Supabase calls table. Listen back and note where the agent gets confused.

The build is not finished until verification is boring.

01 Can the caller interrupt and still be understood?

02 Does the agent know when to stop and transfer?

03 Can you replay the transcript and find every decision?

04 Run npm run build or the focused test command.

05 Download or open the produced artifact and inspect it visually.

06 Write the failure and fix in README.md or the Skool proof post.

01 Commit safe files to GitHub after reviewing the diff.

02 Open Vercel and deploy a preview first.

03 Add missing env vars under Project Settings -> Environment Variables if the build fails.

04 Run the workflow with fake or test data in preview.

05 Promote or deploy production only after the smoke test passes.

06 Save the live URL and proof artifact in Skool.

AGENT DEPLOY PROMPT

```
Verify the preview build, list any missing env vars, and give me the exact smoke test before production.
```

If something goes wrong, ask for a small repair instead of a total rewrite.

01 Letting the agent answer legal, medical, or financial questions.

02 No handoff path when the caller is angry or confused.

03 Judging quality from one demo call instead of repeated QA.

04 Ask the agent to explain the failing layer: local app, env var, database, external API, prompt, deploy, or auth.

05 Ask for one fix, one test, and one rollback step.

06 Do not let the agent change unrelated files to hide the failure.

SKOOL PROOF

Design a five-minute voice-agent QA script with ten test calls and clear pass/fail criteria.

WHAT TO POST

Artifact, screenshot or URL, what worked, what broke, exact prompt used, and the next target.

SOURCES

Codex: https://developers.openai.com/codex

LiveKit: https://docs.livekit.io/intro/overview/

Deepgram: https://developers.deepgram.com/docs

ElevenLabs: https://elevenlabs.io/docs

OpenAI: https://platform.openai.com/docs/quickstart

Claude: https://docs.anthropic.com/en/docs/overview