Below is a step-by-step **Node.js embedding guide for ChatKit**, written for people who **already built an agent workflow in OpenAI Agent Builder** and want to **embed that published workflow into a Node/Express website** (the "Nikon demo" pattern).

This guide assumes you downloaded the demo from GitHub: https://github.com/BitBloom-HQ/openai_agent_builder_nikondemo (the repo structure matches your ZIP), and that you already have a **published Workflow ID** from Agent Builder.

---

0) Who this guide is for

This guide is **not** about building the agent workflow logic in Agent Builder. It's for the next step: **embedding your existing agent workflow into a Node.js site using ChatKit**, with a secure server-side session endpoint and a minimal browser-side script.

---

1) What you get in this demo project (folder map)

From the ZIP (and the GitHub repo), the important files are:

- server.js
  Express server that exposes **POST /api/chatkit/session** and creates a ChatKit session **server-side** (so your API key never goes to the browser).
- package.json
  Contains the libraries you install (Express, EJS, dotenv, axios).
- views/index.ejs
  Main page template. Loads ChatKit from the CDN and includes the chat container partial.
- views/partials/chat-container.ejs
  The floating chat panel markup (container + "Connecting…" status).
- public/app-chatkit.js
  Browser logic: fetches a short-lived client secret from your server and mounts <openai-chatkit>.
- public/chat.css
  Styling for the floating chat window.
- public/styles.css (+ optional page CSS)
  Site styling (not required for ChatKit to work).

---

2) Prerequisites from Agent Builder (you already did this)

A) You must publish your workflow

Publishing gives you the **Workflow ID** you'll use in the Node server.

B) You must have credits / billing set up

If your account has no funds, the UI may look fine but responses will fail; the real clue is in Logs ("quota / billing").

C) Domain allowlist + public key (important, even for localhost)

In your OpenAI dashboard, whitelist the domain under **Security → Domain allowlist**, then use the **Public Key** for that domain. For local dev you can use a temporary domain (example mentioned: ngrok).

---

3) Install dependencies (from package.json)

From the extracted folder:

npm install

This project installs (from package.json in your ZIP):

- express
- ejs
- dotenv
- axios

---

4) Create your .env file (keys live on the server)

Create a .env file in the project root (next to server.js).

You need **three** variables:

1. OPENAI_API_KEY
   Your secret API key. **Server only, never in the browser.**

2. WORKFLOW_ID
   Your published workflow id from Agent Builder.
3. CHATKIT_DOMAIN_PK
   The **public key** you got from **Security → Domain allowlist** (used by the ChatKit CDN script in views/index.ejs).

Example:

OPENAI_API_KEY=sk-...

WORKFLOW_ID=wf_...

CHATKIT_DOMAIN_PK=pk_...

PORT=3000

Why this matters: the demo's "golden rule" is exactly this: don't ship your OpenAI key to the browser.

---

5) Run the demo site

Start it:

npm start

(or node server.js, same thing)

Open:

- http://localhost:3000

Then click **"Chat with the Agent"** and you should see the floating chat window and a **"Connecting to agent…"** status line while it fetches a client secret.

---

6) How the integration works (the real wiring)

A) The server endpoint: POST /api/chatkit/session

Your browser never talks to OpenAI directly for sessions. Instead:

browser → your server → OpenAI → client_secret → browser

That's the handshake pattern.

In this codebase, server.js uses **axios** (not the OpenAI SDK) to call:

- POST https://api.openai.com/v1/chatkit/sessions
- With headers:
    - Authorization: Bearer ${OPENAI_API_KEY}
    - OpenAI-Beta: chatkit_beta=v1
- And body:
    - workflow: { id: WORKFLOW_ID }
    - user: <some identifier>

The endpoint responds to the browser with:

- client_secret
- expires_at

It also normalizes common error types (quota, rate limit, unauthorized) so the frontend can show a friendly message.

B) The client script: public/app-chatkit.js

This file does 3 core jobs:

1. Calls your server:

- fetch('/api/chatkit/session', { method: 'POST' })

2. Creates and mounts the web component:

- document.createElement('openai-chatkit')

3. Injects configuration via:

- el.setOptions({ api: { getClientSecret() { ... } }, ...buildOptions() })

So the ChatKit widget gets its session secret **on demand** from your backend.

C) The HTML: ChatKit CDN + mount point

In views/index.ejs, ChatKit is loaded with the public key:

- https://cdn.platform.openai.com/deployments/chatkit/chatkit.js?pk=<%= process.env.CHATKIT_DOMAIN_PK %>

And the chat UI mounts into the container from:

- views/partials/chat-container.ejs (the target + status line)

---

7) Customizing the experience (colors + greeting + placeholder)

You have two layers of customization:

A) ChatKit theme + text (in public/app-chatkit.js)

The demo's buildOptions() returns:

- theme.colorScheme: 'dark'
- theme.color.accent.primary: '#FFD400' (Nikon yellow)
- composer.placeholder: 'Ask about the Nikon D700…'
- startScreen.greeting: 'What can I help you with?'
- history.enabled: true
- locale: 'en-US'

To customize:

- Change accent color: edit theme.color.accent.primary
- Change the input placeholder: edit composer.placeholder
- Change the opening greeting: edit startScreen.greeting

B) Widget container styling (in public/chat.css)

This controls the floating panel itself:

- size (width/height)
- position (bottom/right)
- border, shadow, radius
- responsive behavior for mobile

If you want a bigger widget, update:

- #chat-container { width: 360px; height: 640px; }

---

8) Common issues (quick fixes)

“It shows the chat but doesn’t answer”

Most common causes:

- Wrong WORKFLOW_ID
- Wrong OPENAI_API_KEY
- Your endpoint path isn’t exactly /api/chatkit/session
- No billing/credits (check logs: quota/billing issue).

“Forbidden / domain issues”

Make sure you added your domain to **Security → Domain allowlist** and used the matching public key, even for local testing (use a temporary domain if needed).

---

9) Production safety rules (do not skip)

- Keep OPENAI_API_KEY on the server only.
- Treat the ChatKit client_secret as short-lived and fetch fresh when needed.
- Domain allowlist your production domain and use the correct public key.