Below is a **step-by-step, beginner-friendly** guide for someone who has:

- **A PHP website (server-side PHP)**
- **JavaScript on the client side**
- **Access to their website files (FTP / cPanel / file manager)**
- A **published workflow** in OpenAI Agent Builder (they already built it)

This guide shows how to reuse your GitHub demo files (same as your ZIP), **without building a new widget**.

ChatKit + OpenAI Agent Builder on a PHP Website (Using the Nikon Demo Files)

Who this guide is for

This guide is for people who **already created and published an agent workflow** in OpenAI Agent Builder, and now want to **embed that agent into a website using ChatKit**.

The Nikon demo shows the idea in Node.js, but the key concept is the same everywhere:

- The **browser** shows the chat UI.
- The **server** (PHP in your case) creates a **ChatKit session** and returns a **short-lived client secret**.
- Your **OpenAI API key never goes into the browser**.

What you will copy from the GitHub repo

From the repo/ZIP, we will reuse these files:

1. **public/app-chatkit.js**
   This is the main browser JavaScript file that mounts ChatKit and calls the server endpoint.
2. **public/chat.css**
   This styles the chat container.
3. **views/partials/chat-container.ejs**
   This contains the HTML markup for the chat container. We will copy/paste its content into a PHP page.

We will **NOT** use the Node server (server.js) or EJS templates as-is. Instead, we will create a small **PHP endpoint** that behaves like the Node endpoint.

Step 1: Get the 3 things you need from OpenAI

You need:

A) Your Workflow ID

This is the wf_... ID you get after you **Publish** your workflow in Agent Builder.

B) Add your domain to the Domain allowlist, and get the Public Key

In OpenAI dashboard:

- Go to **Security**
- Under **Domain allowlist**, add your site domain
- You will get a **Public Key** for that domain (used in the ChatKit script tag).

C) Your OpenAI API key

You will store it **only on the server**, never in JavaScript.

Step 2: Copy the frontend files into your PHP website

On your PHP server, create these folders (if you don't already have them):

- /public/ (or /assets/ or /static/ – any public folder is fine)
- Inside it create:
    - /public/js/
    - /public/css/

Now copy these files from the repo/ZIP:

- Copy **public/app-chatkit.js** into:
  your-website/public/js/app-chatkit.js
- Copy **public/chat.css** into:
  your-website/public/css/chat.css

(If you prefer different folders, that's okay. You just must use the correct paths in your HTML later.)

Step 3: Add the chat HTML container into your PHP page

Open the file in the repo:

- views/partials/chat-container.ejs

Copy its entire content. It looks like this:

```
<div id="chat-container">
  <div id="my-chat" style="width:100%; height:100%;"></div>
  <div id="chat-status">
    <span id="chat-status-text">Connecting to agent…</span>
    <button id="chat-close" aria-label="Close chat">✕</button>
  </div>
</div>
```

Now paste that HTML into the PHP page where you want the chat to exist, for example:

- index.php
- support.php
- camera-help.php

✅ Paste it near the bottom of <body>.

Step 4: Add a button that opens the chat

Your public/app-chatkit.js listens for clicks on any element with class:

- .open-chat

So you must add a button somewhere on your page like this:

```
<button class="open-chat">Chat with the Agent</button>
```

When someone clicks that button, the chat appears.

Step 5: Add the CSS + scripts to your PHP page

Inside your PHP page:

A) In the <head>, add the CSS

<link rel="stylesheet" href="/public/css/chat.css">

B) Load ChatKit with your domain public key

Still in <head> (or before your JS file), add:

<script

src="https://cdn.platform.openai.com/deployments/chatkit/chatkit.js?pk=YOUR_DOMAIN_PUBLIC_KEY"

async

></script>

This is the exact pattern used in the demo template (it loads ChatKit “async”).

C) At the bottom of <body>, load app-chatkit.js

<script src="/public/js/app-chatkit.js" defer></script>

At this point:

- The chat UI can load
- But it will not connect yet (because we still need the PHP server endpoint)

Step 6: Create the PHP endpoint /api/chatkit/session

In the demo, the browser calls this route:

- POST /api/chatkit/session

We will create the same route in PHP.

Option A (Easiest): Use a real file path like /api/chatkit/session.php

If you cannot do “pretty URLs”, do this:

1. Create a folder:
   your-website/api/chatkit/
2. Create this file:
   your-website/api/chatkit/session.php

Then you must update **one line** inside public/app-chatkit.js:

Find:

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

Change it to:

fetch('/api/chatkit/session.php', { method: 'POST' });

✅ This works on almost any PHP hosting.

Option B (Nice URLs): Keep /api/chatkit/session (no .php)

If your server supports URL rewriting (common on Apache), you can keep the exact URL the demo expects.

Example .htaccess (Apache) inside /api/chatkit/ folder:

RewriteEngine On

RewriteRule ^session$ session.php [L]

Then your browser can call:

- /api/chatkit/session
  and Apache will run:
- session.php

If you don't know which option you have, use **Option A**. It's simpler.

Step 7: Store your keys securely (simple beginner method)

Important rule

- Your **OpenAI API key must stay on the server**, not in JavaScript.

Beginner-friendly secure approach (works on most shared hosting)

Create a folder **outside** the public web root if possible.

Many hosts have this structure:

- /public_html/ (public website files)
- /home/username/ (private)

If you have public_html, do this:

1. Create a private config file here (NOT inside public_html):

- /home/username/private/openai-config.php

2. Put this inside:

```
<?php
// /home/username/private/openai-config.php
return [
  'OPENAI_API_KEY' => 'paste_your_openai_api_key_here',
  'WORKFLOW_ID'    => 'wf_your_workflow_id_here',
];
```

✅ This file should NOT be reachable from the internet.

Step 8: Add the PHP code that creates a ChatKit session

Create (or edit) the file:

- /api/chatkit/session.php

Paste this code:

```
<?php
// /api/chatkit/session.php
header('Content-Type: application/json');

if ($_SERVER['REQUEST_METHOD'] !== 'POST') {
  http_response_code(405);
```

```
  echo json_encode(['error' => ['code' => 'method_not_allowed', 'message' => 'POST only']]);
  exit;
}

// Load secrets from your private config file:
$config = require '/home/username/private/openai-config.php';

$OPENAI_API_KEY = $config['OPENAI_API_KEY'] ?? '';
$WORKFLOW_ID    = $config['WORKFLOW_ID'] ?? '';

if (!$OPENAI_API_KEY || !$WORKFLOW_ID) {
  http_response_code(500);
  echo json_encode(['error' => ['code' => 'server_misconfig', 'message' => 'Missing API key or workflow id']]);
  exit;
}

// Identify the user (simple)
$userId = $_SERVER['REMOTE_ADDR'] ?? 'php-demo';

// Call OpenAI to create a ChatKit session (same idea as the demo server.js)
$payload = json_encode([
  'user' => $userId,
  'workflow' => ['id' => $WORKFLOW_ID],
]);
```

```
$ch = curl_init('https://api.openai.com/v1/chatkit/sessions');
curl_setopt_array($ch, [
  CURLOPT_POST => true,
  CURLOPT_RETURNTRANSFER => true,
  CURLOPT_HTTPHEADER => [
    'Content-Type: application/json',
    'OpenAI-Beta: chatkit_beta=v1',
    'Authorization: Bearer ' . $OPENAI_API_KEY,
  ],
  CURLOPT_POSTFIELDS => $payload,
  CURLOPT_TIMEOUT => 15,
]);

$response = curl_exec($ch);
$httpCode = curl_getinfo($ch, CURLINFO_HTTP_CODE);
$errorMsg = curl_error($ch);
curl_close($ch);

if ($response === false) {
  http_response_code(500);
  echo json_encode(['error' => ['code' => 'curl_error', 'message' => $errorMsg]]);
  exit;
}

$data = json_decode($response, true);
```

```
// Success: return only the short-lived client_secret (what the frontend needs)
if ($httpCode >= 200 && $httpCode < 300) {
  echo json_encode([
    'client_secret' => $data['client_secret'] ?? null,
    'expires_at'    => $data['expires_at'] ?? null,
  ]);
  exit;
}

// Error: return a safe message
http_response_code($httpCode ?: 500);
echo json_encode([
  'error' => [
    'code' => $data['error']['code'] ?? 'openai_error',
    'message' => $data['error']['message'] ?? 'OpenAI error',
  ]
]);
```

This matches the same “handshake” described in the script:

- Browser → your server endpoint → OpenAI → returns a short-lived secret.

Step 9: Customize the chat (colors, greeting, placeholder)

Open this file you copied:

- public/app-chatkit.js

Find the function:

- buildOptions()

In your repo it contains options like:

- theme.colorScheme
- theme.color.accent.primary
- composer.placeholder
- startScreen.greeting

You can edit it like this:

```
function buildOptions() {
  return {
    theme: {
      colorScheme: 'dark',
      color: { accent: { primary: '#FFD400', level: 2 } },
      radius: 'round',
      density: 'compact',
      typography: { fontFamily: "'Inter', system-ui, sans-serif" },
    },
    composer: {
      placeholder: 'Ask about the Nikon D700…',
    },
    startScreen: {
      greeting: 'What can I help you with?',
    },
    history: { enabled: true },
    locale: 'en-US',
  };
}
```

Easy customization examples

- Change accent color:

    - primary: '#00AEEF' (blue)
- Change greeting:
    - greeting: 'Hi! Ask me anything about our product.'
- Change placeholder:
    - placeholder: 'Type your message here…'

If you break something: don’t panic. Just undo your last change and refresh.

Step 10: Test it (simple checklist)

1. Open your website page in the browser.
2. Click the button: **“Chat with the Agent”**
3. You should see:
    - “Connecting to agent…” (loading state)
4. If it connects, the status disappears and the chat is live.

If it does NOT connect (most common causes)

Your script calls out the top 3 causes:

- Wrong API key
- Wrong workflow ID
- Wrong session route URL (must match exactly)

So check these:

- Did you paste the right API key in openai-config.php?
- Did you paste the right wf_... workflow ID?
- Does the JS call the correct URL (/api/chatkit/session.php if you chose Option A)?
- Did you add your domain in Domain allowlist and use the correct pk=... public key?

Minimal security (very easy and worth doing)

Even for beginners: add one simple protection so random sites can’t call your session endpoint.

In session.php, before calling OpenAI, add:

```
$allowedOrigins = [
  'https://yourdomain.com',
  'https://www.yourdomain.com'
];

$origin = $_SERVER['HTTP_ORIGIN'] ?? '';
if ($origin && !in_array($origin, $allowedOrigins, true)) {
  http_response_code(403);
  echo json_encode(['error' => ['code' => 'forbidden', 'message' => 'Invalid origin']]);
  exit;
}
```

This is not “military security”, but it blocks a lot of lazy abuse.