Copy page

ChatKit

Build and customize an embeddable chat with ChatKit.

ChatKit is the best way to build agentic chat experiences. Whether you’re building an internal knowledge base assistant, HR onboarding helper, research companion, shopping or scheduling assistant, troubleshooting bot, financial planning advisor, or support agent, ChatKit provides a customizable chat embed to handle all user experience details.

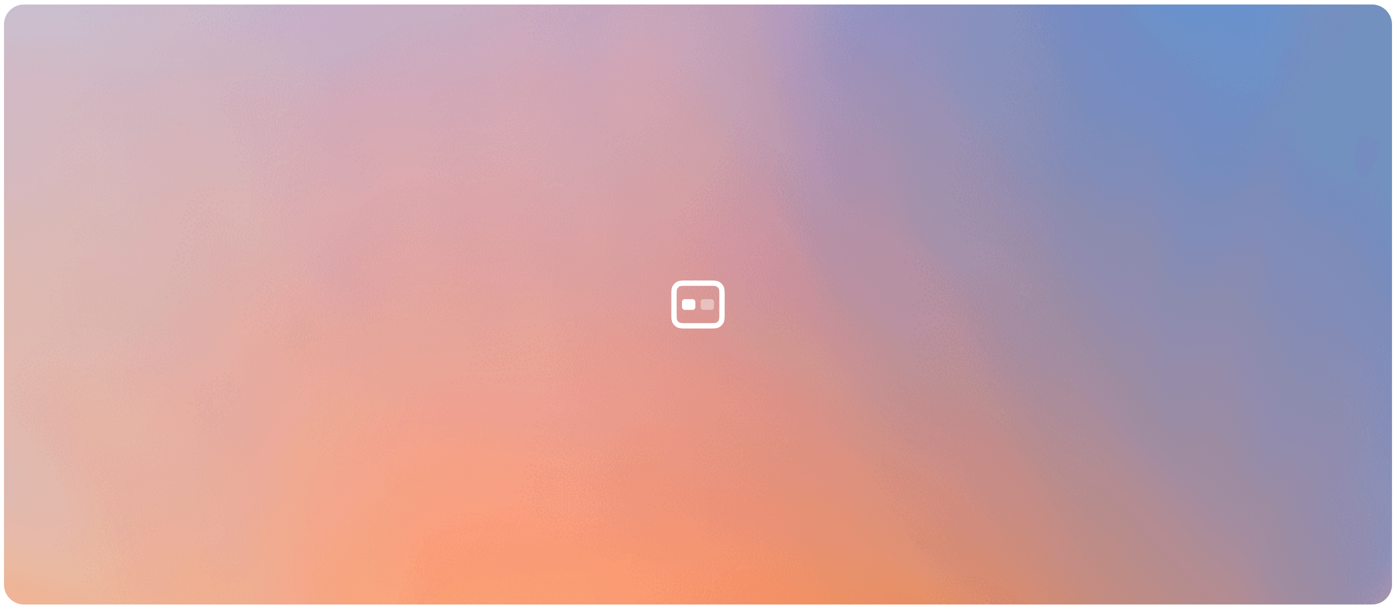
Use ChatKit's embeddable UI widgets, customizable prompts, tool-invocation support, file attachments, and chain-of-thought visualizations to build agents without reinventing the chat UI.

Overview

There are two ways to implement ChatKit:

- **Recommended integration**. Embed ChatKit in your frontend, customize its look and feel, let OpenAI host and scale the backend from Agent Builder. Requires a development server.
- **Advanced integration.** Run ChatKit on your own infrastructure. Use the ChatKit Python SDK and connect to any agentic backend. Use widgets to build the frontend.

Get started with ChatKit

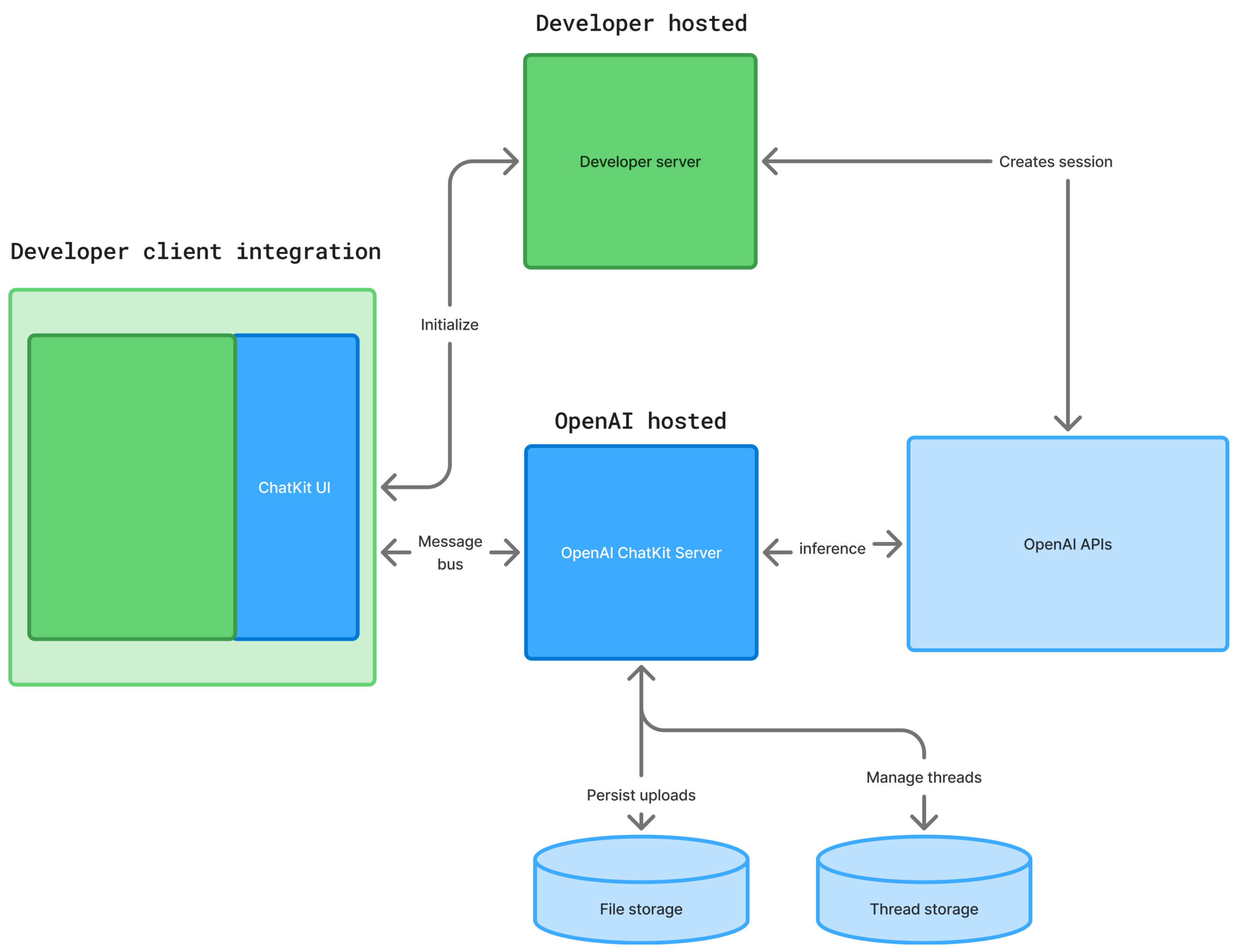


Embed ChatKit in your frontend
Embed a chat widget, customize its look and feel, and let OpenAI host and scale the backend

Advanced integration
Use any backend and the ChatKit SDKs to build your own custom ChatKit user experience

Embed ChatKit in your frontend

At a high level, setting up ChatKit is a three-step process. Create an agent workflow, hosted on OpenAI servers. Then set up ChatKit and add features to build your chat experience.



1. Create an agent workflow

Create an agent workflow with Agent Builder. Agent Builder is a visual canvas for designing multi-step agent workflows. You'll get a workflow ID.

The chat embedded in your frontend will point to the workflow you created as the backend.

2. Set up ChatKit in your product

To set up ChatKit, you'll create a ChatKit session and create a backend endpoint, pass in your workflow ID, exchange the client secret, add a script to embed ChatKit on your site.

1 On your server, generate a client token.

This snippet spins up a FastAPI service whose sole job is to create a new ChatKit session via the OpenAI Python SDK and hand back the session's client secret:

server.py python

```python
from fastapi import FastAPI
from pydantic import BaseModel
from openai import OpenAI
import os

app = FastAPI()
openai = OpenAI(api_key=os.environ["OPENAI_API_KEY"])

@app.post("/api/chatkit/session")
def create_chatkit_session():
    session = openai.chatkit.sessions.create({
      # ...
    })
    return { client_secret: session.client_secret }
```

2 In your server-side code, pass in your workflow ID and secret key to the session endpoint.

The client secret is the credential that your ChatKit frontend uses to open or refresh the chat session. You don't store it; you immediately hand it off to the ChatKit client library.

See the chatkit-js repo on GitHub.

chatkit.ts typescript

```typescript
export default async function getChatKitSessionToken(
deviceId: string
): Promise<string> {
const response = await fetch("https://api.openai.com/v1/chatkit/sessions'
    method: "POST",
    headers: {
    "Content-Type": "application/json",
    "OpenAI-Beta": "chatkit_beta=v1",
    Authorization: "Bearer " + process.env.VITE_OPENAI_API_SECRET_KEY,
    },
    body: JSON.stringify({
    workflow: { id: "wf_68df4b13b3588190a09d19288d4610ec0df388c3983f58d1'
    user: deviceId,
    }),
});

const { client_secret } = await response.json();

return client_secret;
}
```

3 In your project directory, install the ChatKit React bindings:

```
npm install @openai/chatkit-react
```

4 Add the ChatKit JS script to your page. Drop this snippet into your page's `<head>` or wherever you load scripts, and the browser will fetch and run ChatKit for you.

index.html — html

```html
<script
src="https://cdn.platform.openai.com/deployments/chatkit/chatkit.js"
async
></script>
```

5 Render ChatKit in your UI. This code fetches the client secret from your server and mounts a live chat widget, connected to your workflow as the backend.

Your frontend code — react

```
import { ChatKit, useChatKit } from '@openai/chatkit-react';

   export function MyChat() {
     const { control } = useChatKit({
       api: {
         async getClientSecret(existing) {
           if (existing) {
             // implement session refresh
           }

           const res = await fetch('/api/chatkit/session', {
             method: 'POST',
             headers: {
               'Content-Type': 'application/json',
             },
           });
           const { client_secret } = await res.json();
           return client_secret;
         },
       },
     });

     return <ChatKit control={control} className="h-[600px] w-[320px]" />
   }
```

3. Build and iterate

See the custom theming, widgets, and actions docs to learn more about how ChatKit works. Or explore the following resources to test your chat, iterate on prompts, and add widgets and tools.

Build your implementation

ChatKit docs on GitHub
Learn to handle authentication, add theming and customization, and more.

ChatKit Python SDK
Add server-side storage, access control, tools, and other backend functionality.

ChatKit JS SDK
Check out the ChatKit JS repo.

Explore ChatKit UI

chatkit.world

Play with an interactive demo of ChatKit.

Widget builder

Browse available widgets.

ChatKit playground

Play with an interactive demo to learn by doing.

See working examples

Samples on GitHub

See working examples of ChatKit and get inspired.

Starter app repo

Clone a repo to start with a fully working template.

Next steps

When you're happy with your ChatKit implementation, learn how to optimize it with evals. To run ChatKit on your own infrastructure, see the advanced integration docs.