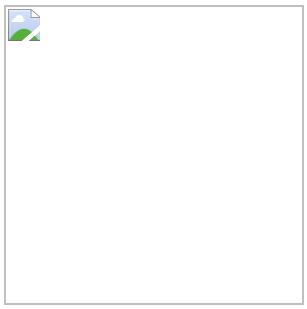
tambo_ai



Tambo AI

Generative UI for React

Build apps that adapt to your users.

Documentation • Discord

What is Tambo?

Tambo is a generative UI SDK for React. Register your components, and the AI decides which ones to render based on natural language conversations.

https://github.com/user-attachments/assets/8381d607-b878-4823-8b24-ecb8053bef23

Why We Built This

Most software is built around a one-size-fits-all mental model that doesn't fit every user.

Users shouldn't have to learn your app. Generative UI shows the right components based on what someone is trying to do. First-time users and power users see different things.

Users shouldn't have to click through your workflows. "Show me sales from last quarter grouped by region" should just work. The AI translates what users want into the right interface.

```
const components: TamboComponent[] = [{
  name: "Graph",
```

```
  description: "Displays data as charts",
  component: Graph,
  propsSchema: z.object({ data: z.array( ... ), type: z.enum(["line", "bar",
"pie"]) })
}];
```

Get Started

```
npx tambo create-app my-tambo-app
cd my-tambo-app
npx tambo init      # choose cloud or self-hosted
npm run dev
```

Tambo Cloud is a free hosted backend. **Self-hosted** runs on your own infrastructure.

Check out the pre-built component library for ready-made primitives, or fork a template:

Template	Description
AI Chat with Generative UI	Chat interface with dynamic component generation
AI Analytics Dashboard	Analytics dashboard with AI-powered visualization

https://github.com/user-attachments/assets/6cbc103b-9cc7-40f5-9746-12e04c976dff

How It Works

Tambo supports two kinds of components.

Generative components render once in response to a message. Charts, summaries, data visualizations.

https://github.com/user-attachments/assets/3bd340e7-e226-4151-ae40-aab9b3660d8b

Interactable components persist and update as users refine requests. Shopping carts, spreadsheets, task boards.

https://github.com/user-attachments/assets/12d957cd-97f1-488e-911f-0ff900ef4062

Registering Components

Tell the AI which components it can use. Zod schemas define the props.

```
// Generative: AI creates on-demand
const components: TamboComponent[] = [
```

```
  {
    name: "Graph",
    description: "Displays data as charts using Recharts library",
    component: Graph,
    propsSchema: z.object({
      data: z.array(z.object({ name: z.string(), value: z.number() })),
      type: z.enum(["line", "bar", "pie"]),
    }),
  },
];

// Interactable: persists and updates by ID
const InteractableNote = withInteractable(Note, {
  componentName: "Note",
  description: "A note supporting title, content, and color modifications",
  propsSchema: z.object({
    title: z.string(),
    content: z.string(),
    color: z.enum(["white", "yellow", "blue", "green"]).optional(),
  }),
});
```

Docs: generative components, interactable components

The Provider

Wrap your app with `TamboProvider`.

```
<TamboProvider
  apiKey={process.env.NEXT_PUBLIC_TAMBO_API_KEY!}
  components={components}
>
  <Chat />
  <InteractableNote id="note-1" title="My Note" content="Start writing..." />
</TamboProvider>
```

For apps with signed-in users, pass a per-user `userToken` (OAuth access token) to `TamboProvider` to enable per-user auth and connect Tambo to your app's end-user identity. See User Authentication for details.

Docs: provider options

Hooks

Send messages with `useTamboThreadInput`. `useTamboThread` handles streaming, including props for generated components and tool calls.

```
const { value, setValue, submit, isPending } = useTamboThreadInput();

<input value={value} onChange={(e) => setValue(e.target.value)} />
<button onClick={() => submit()} disabled={isPending}>Send</button>
```

```
const { thread } = useTamboThread();

{
  thread.messages.map((message) => (
    <div key={message.id}>
      {Array.isArray(message.content) ? (
        message.content.map((part, i) =>
          part.type === "text" ? <p key={i}>{part.text}</p> : null,
        )
      ) : (
        <p>{String(message.content)}</p>
      )}
      {message.renderedComponent}
    </div>
  ));
}
```

Track streaming status if you want progressive loading:

```
const { streamStatus, propStatus } = useTamboStreamStatus();

if (!streamStatus.isSuccess) return <Spinner />;
{
  propStatus["title"]?.isSuccess && <h3>{title}</h3>;
}
```

Docs: threads and messages, streaming status

Full tutorial

Features

MCP Integrations

Connect to Linear, Slack, databases, or your own MCP servers. Tambo supports the full MCP protocol: tools, prompts, elicitations, and sampling.

```
import { MCPTransport } from "@tambo-ai/react/mcp";

const mcpServers = [
  {
    name: "filesystem",
    url: "http://localhost:8261/mcp",
    transport: MCPTransport.HTTP,
  },
];

<TamboProvider components={components} mcpServers={mcpServers}>
  <App />
</TamboProvider>;
```

https://github.com/user-attachments/assets/c7a13915-8fed-4758-be1b-30a60fad0cda

Supports the full MCP protocol: tools, prompts, elicitations, and sampling.

Docs: MCP integration

Local Tools

Sometimes you need functions that run in the browser. DOM manipulation, authenticated fetches, accessing React state. Define them as tools and the AI can call them.

```
const tools: TamboTool[] = [
  {
    name: "getWeather",
    description: "Fetches weather for a location",
    tool: async (params: { location: string }) =>
      fetch(`/api/weather?q=${encodeURIComponent(params.location)}`).then((r)
=>
        r.json(),
      ),
    inputSchema: z.object({
      location: z.string(),
    }),
    outputSchema: z.object({
      temperature: z.number(),
      condition: z.string(),
      location: z.string(),
    }),
  },
];
```

```
<TamboProvider tools={tools} components={components}>
  <App />
</TamboProvider>;
```

Docs: local tools

Context, Auth, and Suggestions

Additional context lets you pass metadata to give the AI better responses. User state, app settings, current page. **User authentication** passes tokens from your auth provider. **Suggestions** generates prompts users can click based on what they're doing.

```
<TamboProvider
  userToken={userToken}
  contextHelpers={{
    selectedItems: () => ({
      key: "selectedItems",
      value: selectedItems.map((i) => i.name).join(", "),
    }),
    currentPage: () => ({ key: "page", value: window.location.pathname }),
  }}
/>
```

```
const { suggestions, accept } = useTamboSuggestions({ maxSuggestions: 3 });

suggestions.map((s) => (
  <button key={s.id} onClick={() => accept(s)}>
    {s.title}
  </button>
));
```

Docs: additional context, user authentication, suggestions

Supported LLM Providers

OpenAI, Anthropic, Cerebras, Google Gemini, Mistral, and any OpenAI-compatible provider. Full list. Missing one? Let us know.

How Tambo Compares

Feature	Tambo	Vercel AI SDK	CopilotKit	Assistant UI
Component selection	AI decides which components to render	Manual tool-to-component mapping	Via agent frameworks (LangGraph)	Chat-focused tool UI
MCP integration	Built-in	Experimental (v4.2+)	Recently added	Requires AI SDK v5
Persistent stateful components	Yes	No	Shared state patterns	No
Client-side tool execution	Declarative, automatic	Manual via onToolCall	Agent-side only	No
Self-hostable	MIT (SDK + backend)	Apache 2.0 (SDK only)	MIT	MIT
Hosted option	Tambo Cloud	No	CopilotKit Cloud	Assistant Cloud
Best for	Full app UI control	Streaming and tool abstractions	Multi-agent workflows	Chat interfaces

Full documentation

Pricing

Self-Hosted

Free forever. MIT licensed. 5-minute Docker setup.

```
npx tambo init
# Select "Self-hosted"
```

Tambo Cloud

Free tier, then pay as you grow.

- **Free**: 10,000 messages/month
- **Growth**: $25/mo for 200k messages + email support
- **Enterprise**: Custom volume, SLA, SOC 2, HIPAA

Pricing details

Repository Structure

This Turborepo hosts the React SDK ecosystem and Tambo Cloud platform.

`apps/` has the web dashboard (Next.js), the API (NestJS), and MCP services.

`packages/` has shared code. Database schema (Drizzle), LLM helpers, pure utilities, and tooling configs.

The root holds framework packages: `react-sdk/`, `cli/`, `showcase/`, `docs/`, `create-tambo-app/`.

Development

You'll need Node.js 22+, npm 11+, and optionally Docker.

```
git clone https://github.com/tambo-ai/tambo.git
cd tambo
npm install
npm run dev        # apps/web + apps/api
```

Useful commands:

```
npm run build        # Build everything
npm run lint         # Lint (lint:fix to autofix)
npm run check-types  # Type check
npm test             # Run tests
```

Database (requires Docker):

```
npm run db:generate  # Generate migrations
npm run db:migrate   # Apply migrations
npm run db:studio    # Open Drizzle Studio
```

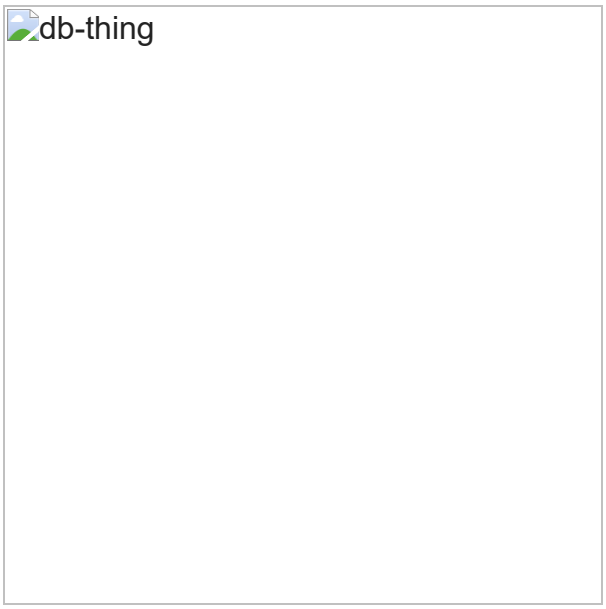
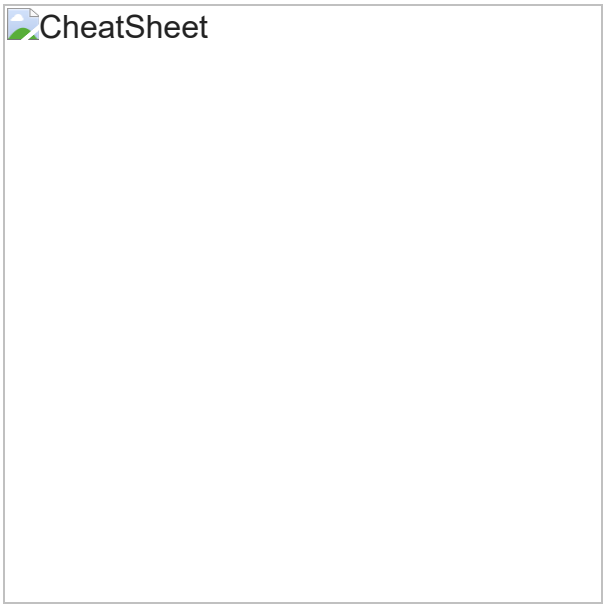
Docker workflow lives in `scripts/cloud/`. See README.DOCKER.md for details.

Contributing Guide

Community

Discord for help and discussion. GitHub to contribute. @tambo_ai for updates.

Built with Tambo

Project	Preview	Description	Links
db-thing by @akinloluwami		Database design through conversation. Create schemas, generate ERDs, get optimization tips, export SQL.	GitHub • Demo
CheatSheet by @michaelmagan		Spreadsheet editor with natural language. Edit cells, create charts, connect external data via MCP.	GitHub • Demo

Built something? Open a PR or share it in Discord.

License

Unless otherwise noted in a workspace (app or package), code in this repo is licensed under MIT (see the root LICENSE).

Some workspaces are licensed under Apache-2.0; see the accompanying `LICENSE` and `NOTICE` files in those workspaces.

For AI/LLM agents: docs.tambo.co/llms.txt