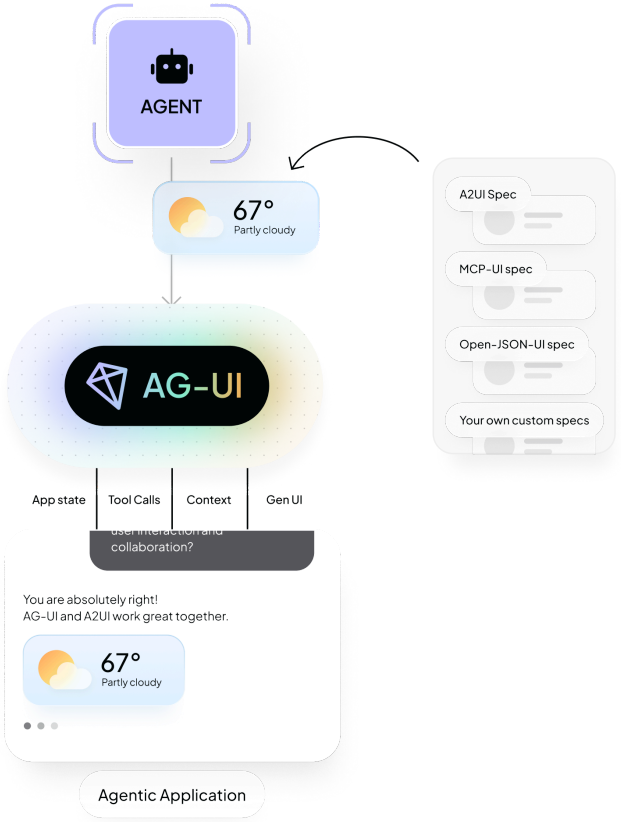
CopilotKit

What Is Generative UI?

Generative UI refers to any user interface that is **partially or fully produced by an AI agent**, rather than authored exclusively by human designers and developers. Instead of the UI being hand-crafted in advance, the agent plays a role in determining *what* appears on the screen, *how* information is structured, and in some cases even *how* the layout is composed.



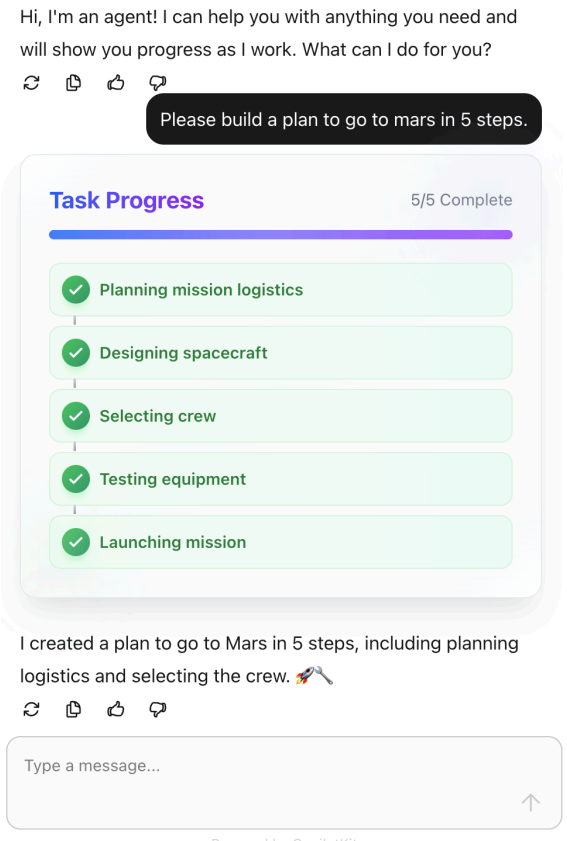
The core idea is simple: as agents become more capable, an agentic application's UI itself becomes more of a dynamic output of the system — able to adapt, reorganize, and respond to user intent and application context. This can be done in very different ways, each with its own tradeoffs.

In the following sections, we will cover:

- **Application Surfaces** - where Generative UI shows up within an agentic application.
- **Attributes** - how different Generative UI types and uses vary and why.
- **Types** - the prominent types of Generative UI, their uses and tradeoffs.

Application Surfaces for Generative UI

Generative UI can surface in different parts of an application depending on how users interact with the agent and how much the application mediates that interaction. These surfaces shape the UX, developer responsibilities, and where generative UI appears.



1. Chat (Threaded Interaction)

A Slack-like conversational interface where the app brokers each turn. Generative UI appears inline as cards, blocks, or tool responses.

Key traits:

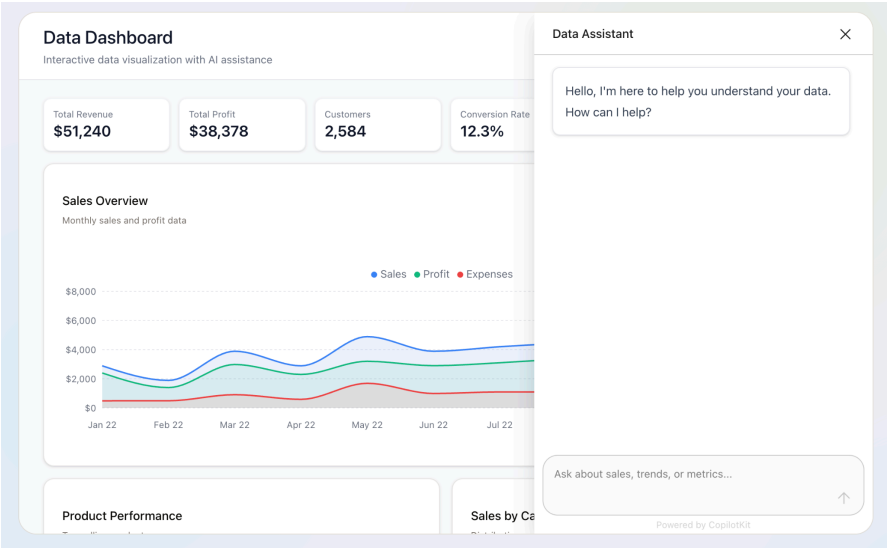
- Turn-based, message-driven flow.
- App mediates all agent communication.
- Great for support, Q&A, debugging, and guided workflows.
- **Examples:** Slack bots, Discord bots, Intercom AI Agent, Zendesk AI, GitHub Copilot Chat, Notion AI Chat.

2. Chat+ (Co-Creator Workspace)

A side-by-side or multi-pane layout: chat in one pane, a dynamic canvas in another. The canvas becomes a shared working space where agent-generated UI appears and evolves.

Key traits:

- Chat remains present but secondary.
- Canvas displays structured outputs and previews.
- Generative UI can appear in the canvas or chat space.
- Ideal for creation, planning, editing, and multi-step tasks.
- **Examples:** Figma AI, Notion AI workspace, Google Workspace Duet side-panel, Replit Ghostwriter paired editor.

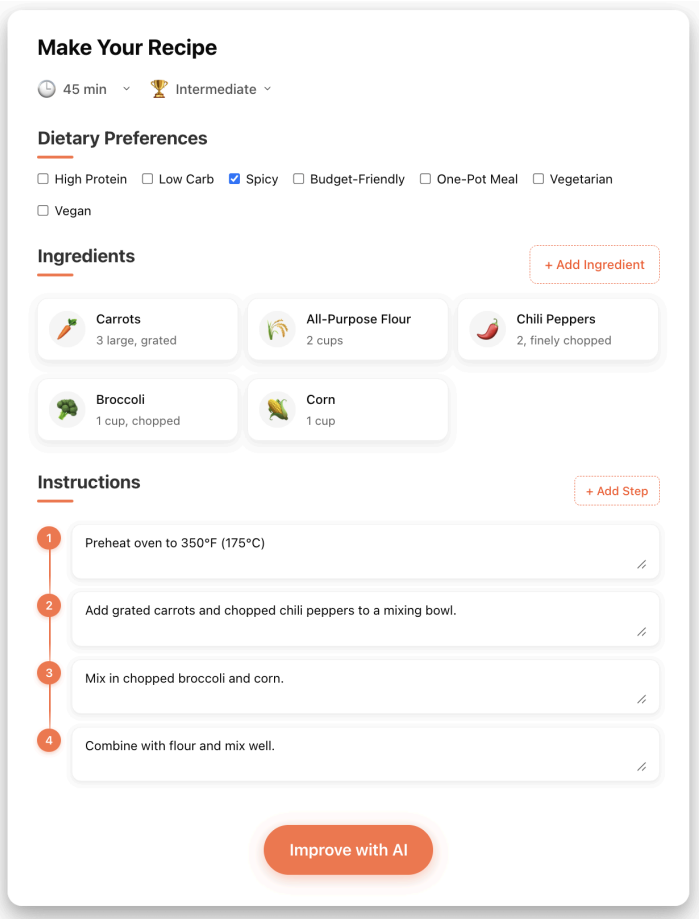


3. Chatless (Generative UI integrated into application UI)

The agent doesn't talk directly to the user. Instead, it communicates with the application through APIs, and the app renders generative UI from the agent as part of its native interface.

Key traits:

- No chat surface at all.
- App decides when and where generative UI appears.
- Feels like a built-in product feature, rather than a conversation.
- Ideal for dashboards, suggestions, and autonomous task helpers.
- **Examples:** Microsoft 365 Copilot (inline editing), Linear Insights, Superhuman AI triage, HubSpot AI Assist, Datadog Notebooks AI panels.



Attributes of Generative UI

Types of Generative UI, and even individual uses vary greatly in terms of two attributes: freedom, and control.

Attribute: Freedom

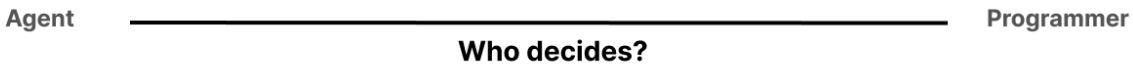
The generative UI types are highly differentiated by what they can represent - their visual "freedom". On the fixed end of the spectrum, static generative UI's return only predefined components. On the other end of this axis, open-ended UI can include arbitrary HTML, making any kind of interaction possible, in theory. Declarative UI sits in the middle, with a wider, but still constrained visual vocabulary form which both the programmer and the agent can choose.

Attribute: Who has Control?

A more subtle, and tricker to manage, attribute of generative UI lies in who decides on the representation: the Agent (LLM) or the Programmer (Application Developer).

To take open-ended generative UI as an example, the agent has the ability to present arbitrary HTML.
But where does the HTML come from? It can be predefined in code which is returned by the agent, or it can be fully generated by an LLM. In many cases the application developer would want to define what the agent can deliver in order to feel native to the app, even though the programmer might want to use HTML for its richness.

Even with static generative UI, there are control choices to be made. You can hardcode the agent to present a specific generative UI when something specific happens, or let an LLM choose to surface it from scratch.



Types of Generative UI

Generative UI approaches fall into three broad categories, each with distinct tradeoffs in developer experience, UI freedom, adaptability, and long-term maintainability.

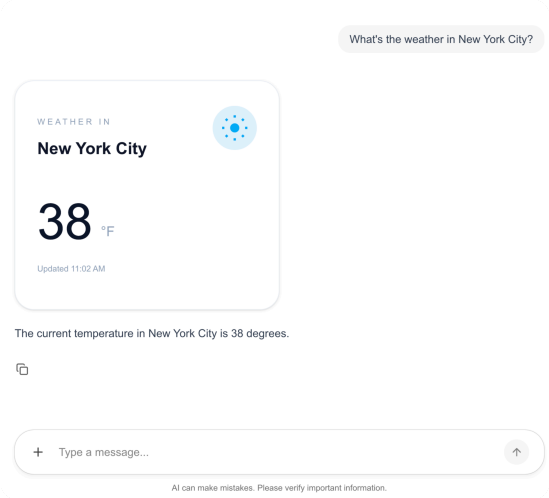
The three patterns can be summarized at a glance:

- **Static** — UI is chosen from a fixed set of hand-built components.
- **Open-Ended** — Arbitrary UI (HTML, iframes, free-form content) is passed between agent and frontend.
- **Declarative** — A structured UI specification (cards, lists, forms, widgets) is used between agent and frontend.

As described above, these types are differentiated by their freedom/vocabulary of UI expression, but any of the types can be controlled by the application programmer, or left up to the agent to define.

1. Static Generative UI

Static generative UI allows engineers to hand-craft specific visual components, and agents simply decide **which** of those components to use. The agent does not generate arbitrary UI; instead, it maps generated data to existing UI components.



In this model, the front end defines every detail of the experience — the layouts, the styles, the interaction patterns, and the constraints. The backend or agent contributes information and intent, but the rendering ultimately comes from a predefined set of components.

Why teams use it:

- Guarantees high visual polish and consistency.
- Ideal for high-traffic, mission-critical surfaces where predictability matters.

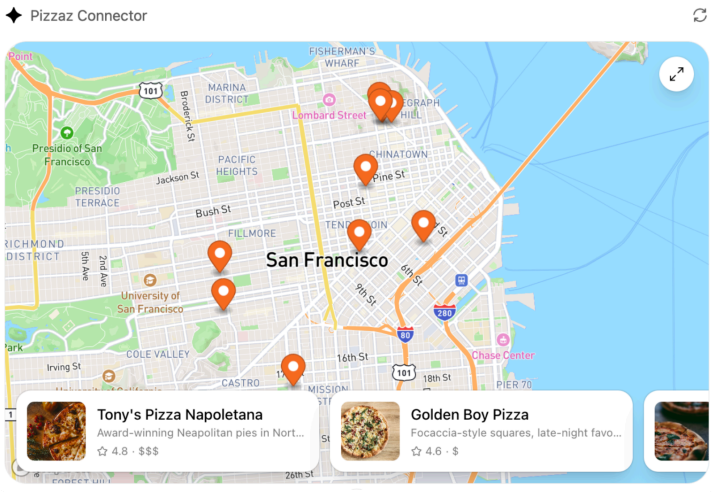
Tradeoffs:

- The more use cases, the more components you must build and maintain.
- The frontend codebase grows proportionally to the number of agent capabilities.

---

2. Open-Ended Generative UI

Open-ended generative UI represents the opposite end of the spectrum. Here, agents generate **complete** UI surfaces — often as HTML, iframes, or free-form markup. Instead of choosing from predefined components, the agent can respond with an entire UI payload that the frontend simply displays.



This approach provides unparalleled flexibility of expression. Agents can render a calendar, a custom table, an animated visualization, or an interactive HTML widget, either generated by the LLM, or predefined by an application developer. The frontend primarily acts as a container.

Why teams use it:

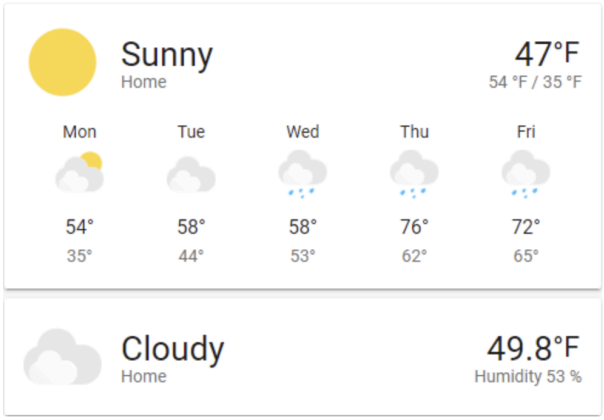
- Any type of UI can be part of an agent response, whether predefined by the programmer or generated by the agent..
- Minimal coupling between frontend code and agent behavior.
- Supports rapid prototyping and complex workflows without frontend engineering cycles.

Tradeoffs:

- Security and performance considerations when rendering arbitrary content.
- Typically web-first and difficult to port to native environments.
- Styling consistency and brand alignment become challenging.

---

3. Declarative Generative UI



Declarative generative UI balances structure and flexibility by having agents return **a structured specification** rather than arbitrary UI code. Instead of free-form HTML, agents emit a well-defined schema — such as a collection of cards, lists, forms, or widgets defined by a declarative standard.

This approach preserves consistency while giving agents far greater expressive power than purely static component libraries. It creates a middle ground where UI is not handcrafted for each use case, but is also not fully free-form.

Why teams use it:

- Supports a wide range of use cases without requiring custom components for each.
- Developers can render the same spec across multiple frameworks (React, mobile, desktop, etc.).
- Cleaner separation between application logic and presentation.

Tradeoffs:

- Custom UI patterns may not be possible.
- Visual differences can still occur if specs are interpreted differently.

Ecosystem Mapping

Several recently announced Generative UI Specifications have added richness (and some confusion) to declarative generative UIs. These include MCP-UI, Open JSON UI, and the impending A2UI.

The generative UI styles map cleanly onto the existing ecosystem of tools and these standards:

Approach	Examples	Strengths	Weaknesses
Static	AG-UI, CopilotChat, useAgent	Fidelity, reliability, brand control	Engineering intensive, linear growth
Open-Ended	MCP-UI, ChatGPT Apps	Unlimited creativity	Security, Web-first
Declarative	Open-JSON-UI, A2UI	Balanced, scalable, multi-renderer	Limited full customization

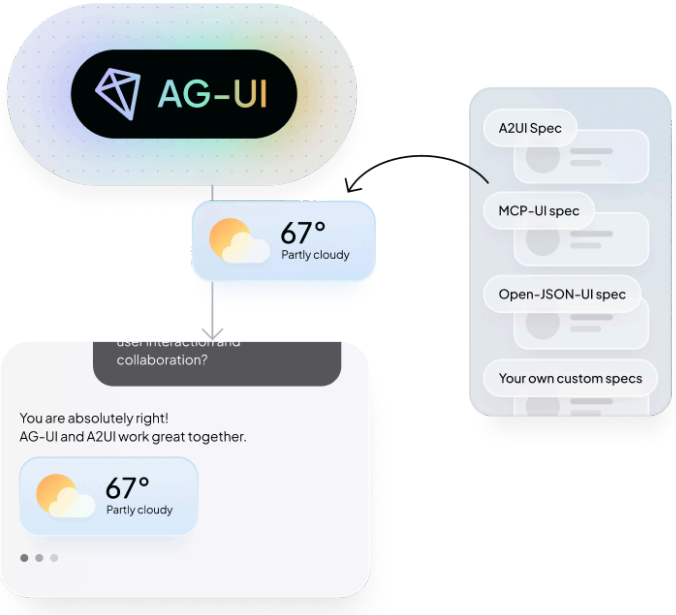
This mapping highlights that no single approach is superior — the best choice depends on your application's priorities, surfaces, and UX philosophy.

AG-UI is Gen UI Agnostic

AG-UI is designed to **support the full spectrum** of generative UI techniques while adding important capabilities that unify them.

AG-UI **integrates seamlessly** with all types: static, declarative, and open-ended generative UI approaches. Whether teams prefer handcrafted components, structured schemas, or agent-authored surfaces, AG-UI can support the workflow.

But AG-UI **adds shared primitives** — interaction models, context synchronization, event handling, a common state framework — that standardize how

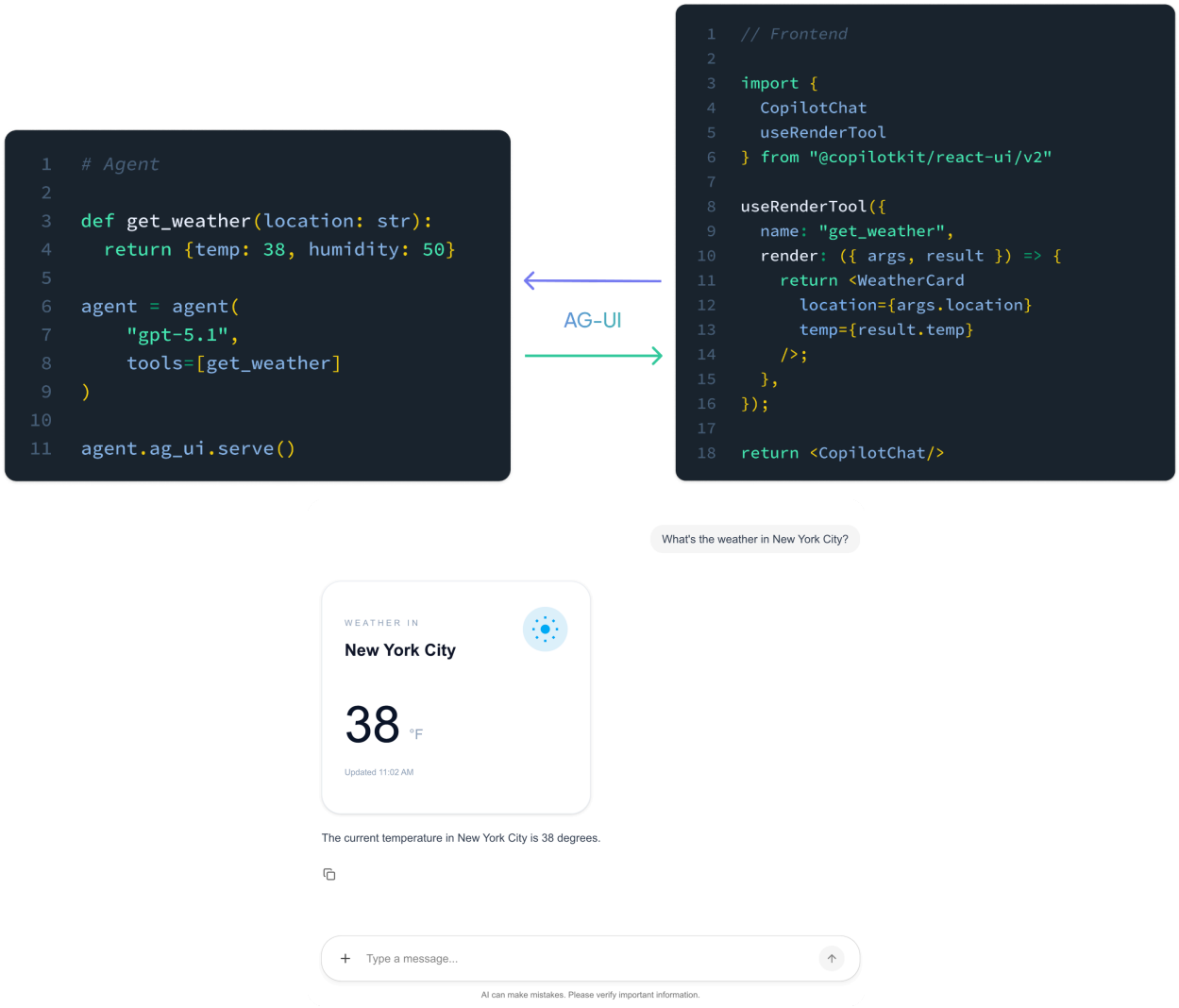


agents and UIs communicate across all surface types.

This creates a consistent mental model for developers while empowering agents to take advantage of the capabilities of any generative UI pattern.

Example Usage for each Generative UI Type

1. Static Gen UI Using CopilotChat

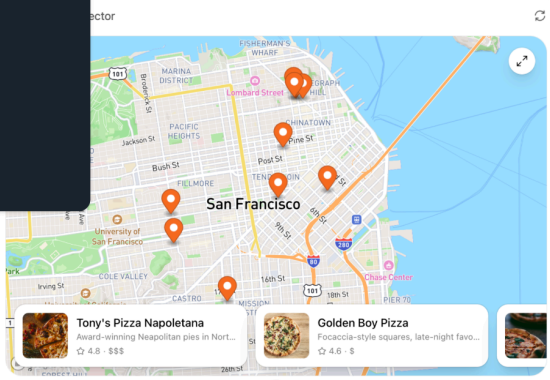



```python
# Agent

def get_weather(location: str):
  return {temp: 38, humidity: 50}

agent = agent(
    "gpt-5.1",
    tools=[get_weather]
)

agent.ag_ui.serve()
```

```tsx
// Frontend

import {
  CopilotChat
  useRenderTool
} from "@copilotkit/react-ui/v2"

useRenderTool({
  name: "get_weather",
  render: ({ args, result }) => {
    return <WeatherCard
      location={args.location}
      temp={result.temp}
    />;
  },
});

return <CopilotChat/>
```

2. Open-ended Gen UI - Using MCP-UI & ChatGPT Apps SDK

```html
<!-- ChatGPT UI -->
<html>
  <body>
    <!-- ChatGPT wraps your app in sandboxed iframes -->
    <iframe src="https://web-sandbox.oaiusercontent.com/...">
      <iframe src="https://your-app.example.com/">
        <!-- Your ChatGPT App SDK UI runs here -->
      </iframe>
    </iframe>
  </body>
</html>

```



3. Declarative Generative UI Using Open-JSON-UI

```
<Card>
  <Image src={airline.logo} size={24} />

  <Text value={`${airline.name} ${number}`} />

  <Text value={`${departure.city} → ${arrival.city}`} />
</Card>

```