Tutorial 30: Next.js ADK Integration - React Chat Interfaces

Difficulty: intermediate
Reading Time: 2 hours
Tags: ui, nextjs, react, copilotkit, chat-interface
Description: Build modern chat interfaces using Next.js and CopilotKit to create seamless React-based agent interactions with real-time features.

:::tip Working Implementation Available

A complete, tested implementation of this tutorial is available!

👉 View Implementation (./../../tutorial_implementation/tutorial30)

The implementation includes:

- ✔️ Python ADK agent with customer support tools
- ✔️ FastAPI backend with AG-UI integration
- ✔️ Next.js 15 frontend with CopilotKit
- ✔️ Comprehensive test suite (30+ tests passing)
- ✔️ Production-ready Makefile
- ✔️ Complete documentation

Quick Start:

```
cd tutorial_implementation/tutorial30
make setup
# Configure your API key in agent/.env
make dev
# Open http://localhost:3000
```

:::

Tutorial 30: Next.js 15 + ADK Integration (AG-UI Protocol)

Estimated Reading Time: 65-75 minutes
Difficulty Level: Intermediate
Prerequisites: Tutorial 29 (UI Integration Intro), Tutorial 1-3 (ADK Basics), Basic Next.js knowledge

Table of Contents

1. Overview (#overview)
2. Prerequisites & Setup (#prerequisites--setup)
3. Quick Start (10 Minutes) (#quick-start-10-minutes)
4. Understanding the Architecture (#understanding-the-architecture)
5. Building a Customer Support Agent (#building-a-customer-support-agent)
6. Advanced Features (#advanced-features)
7. Production Deployment (#production-deployment)
8. Troubleshooting (#troubleshooting)
9. Next Steps (#next-steps)

Overview

What You'll Build

In this tutorial, you'll build a **production-ready customer support chatbot** using:

- **Next.js 15** (App Router)
- **CopilotKit** (AG-UI Protocol)

- **Google ADK** (Agent backend)
- **Gemini 2.0 Flash** (LLM)

Final Result:

```
┌──────────────────────────────────────────────────────────┐
│  Customer Support Chatbot                                │
│  ├─ Real-time chat interface                             │
│  ├─ Tool-augmented responses (knowledge base search)     │
│  ├─ Streaming responses                                  │
│  ├─ Session persistence                                  │
│  ├─ Production deployment (Vercel + Cloud Run)           │
│  └─ 99.9% uptime capability                              │
└──────────────────────────────────────────────────────────┘
```

Why Next.js 15 + ADK?

Feature	Benefit
Next.js 15 App Router	Server Components, streaming, optimized routing
CopilotKit/AG-UI	Pre-built chat UI, type-safe integration
Google ADK	Powerful agent framework with tool calling
Gemini 2.0 Flash	Fast, cost-effective, state-of-the-art LLM
Vercel + Cloud Run	Scalable, global deployment

Prerequisites & Setup

System Requirements

```
# Node.js 18.17 or later
node --version  # Should be >= 18.17

# Python 3.9 or later
python --version  # Should be >= 3.9

# npm/pnpm/yarn
npm --version  # Any version
```

API Keys

1. Google AI API Key

Get your key from Google AI Studio (https://makersuite.google.com/app/apikey):

```

```

2. (Optional) Vercel Account

For deployment: Sign up at Vercel (https://vercel.com)

Quick Start (10 Minutes)

Option 1: Use CopilotKit CLI (Recommended)

The fastest way to get started:

```
# Create new project with ADK template
npx copilotkit@latest create -f adk

# Follow prompts:
# ✓ Project name: customer-support-bot
# ✓ Include ADK agent: Yes
# ✓ Include frontend: Yes (Next.js)

cd customer-support-bot

# Install dependencies (includes Python agent deps)
npm install

# Set API key

# Or create agent/.env:
echo "GOOGLE_API_KEY=your_api_key" > agent/.env

# Run both frontend and agent together!
npm run dev
```

Open http://localhost:3000 - Your agent is live! 🎉

What just happened?

- ✔️ Created Next.js 15 app with App Router
- ✔️ Installed CopilotKit frontend packages
- ✔️ Created Python ADK agent in `agent/` directory
- ✔️ Configured bidirectional communication (AG-UI Protocol)
- ✔️ Set up hot reloading for both frontend and backend

Option 2: Manual Setup (Full Control)

Want to understand every piece? Build from scratch:

Step 1: Create Next.js App

```
npx create-next-app@latest customer-support-bot
# ✓ TypeScript: Yes
# ✓ ESLint: Yes
# ✓ Tailwind CSS: Yes
# ✓ App Router: Yes
# ✓ import alias: No

cd customer-support-bot
```

Step 2: Install CopilotKit

```
npm install @copilotkit/react-core @copilotkit/react-ui
```

Step 3: Setup Project

Clone the tutorial implementation and install dependencies:

```
# Clone and navigate to tutorial
cd tutorial_implementation/tutorial30

# Install all dependencies (backend + frontend)
make setup

# Configure API key
cp agent/.env.example agent/.env
# Edit agent/.env and add your GOOGLE_API_KEY
```

Alternative Manual Setup:

```
# Backend setup
pip install -r requirements.txt
pip install -e .

# Frontend setup
cd nextjs_frontend
npm install
cd ..
```

Step 4: Create Agent

Create `agent/agent.py`:

```
"""Customer support ADK agent with AG-UI integration."""

import os
from typing import Dict
from dotenv import load_dotenv
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
import uvicorn

# AG-UI ADK integration imports
from ag_ui_adk import ADKAgent, add_adk_fastapi_endpoint

# Google ADK imports
from google.adk.agents import Agent

# Load environment variables
load_dotenv()

# Define knowledge base search tool
def search_knowledge_base(query: str) -> str:
    """
    Search the knowledge base for relevant information.

    Args:
        query: Search query to find relevant articles

    Returns:
        Formatted string with article title and content
    """
    # Mock knowledge base - replace with real database/vector store
    knowledge_base = {
        "refund policy": {
            "title": "Refund Policy",
            "content": "We offer full refunds within 30 days of purchase. " +
                       "Contact support@company.com to initiate a refund."
        },
        "shipping": {
            "title": "Shipping Information",
            "content": "Standard shipping takes 5-7 business days. " +
                       "Express shipping (2-3 days) available for $15 extra."
        },
        "warranty": {
            "title": "Warranty Coverage",
            "content": "All products include 1-year warranty covering " +
                       "manufacturing defects. Extended warranty available."
        },
```

```
        "account": {
            "title": "Account Management",
            "content": "Reset password at /account/reset. Update billing " +
                       "info at /account/billing. Cancel subscription anytime."
        }
    }

    # Simple keyword matching - use vector search in production
    query_lower = query.lower()
    for key, article in knowledge_base.items():
        if key in query_lower:
            return f"**{article['title']}**\n\n{article['content']}"

    # Default response
    return ("**General Support**\n\n"
            "Please contact our support team at support@company.com "
            "or call 1-800-SUPPORT for personalized assistance.")

def lookup_order_status(order_id: str) -> str:
    """
    Look up the status of a customer order.

    Args:
        order_id: The order ID to look up

    Returns:
        Order status information
    """
    # Mock order database - replace with real database
    orders = {
        "ORD-12345": "Shipped - Arriving tomorrow",
        "ORD-67890": "Processing - Ships in 2-3 days",
        "ORD-11111": "Delivered on Jan 15, 2024"
    }

    if order_id.upper() in orders:
        return f"Order {order_id}: {orders[order_id.upper()]}"
    return f"Order {order_id} not found. Please check the order ID and try aga

def create_support_ticket(issue_description: str, priority: str = "normal") ->
    """
    Create a support ticket for complex issues.

    Args:
        issue_description: Description of the customer's issue
        priority: Priority level (low, normal, high, urgent)
```

```
    Returns:
        Ticket confirmation with ticket ID
    """
    import uuid
    ticket_id = f"TICKET-{uuid.uuid4().hex[:8].upper()}"

    return (f"Support ticket created successfully!\n\n"
            f"**Ticket ID:** {ticket_id}\n"
            f"**Priority:** {priority}\n"
            f"**Issue:** {issue_description}\n\n"
            f"Our support team will contact you within 24 hours.")

def get_product_details(product_id: str) -> Dict[str, Any]:
    """
    Get product details from the database.

    Returns product information that can be displayed to the user.
    The frontend will handle rendering this as a ProductCard component.

    Args:
        product_id: The product ID to look up (format: PROD-XXX)

    Returns:
        Dict with status, report, and product details
    """
    # Mock product database - replace with real database in production
    products = {
        "PROD-001": {
            "name": "Widget Pro",
            "price": 99.99,
            "image": "https://placehold.co/400x400/6366f1/fff.png",
            "rating": 4.5,
            "inStock": True,
        },
        "PROD-002": {
            "name": "Gadget Plus",
            "price": 149.99,
            "image": "https://placehold.co/400x400/8b5cf6/fff.png",
            "rating": 4.8,
            "inStock": True,
        },
        "PROD-003": {
            "name": "Premium Kit",
            "price": 299.99,
            "image": "https://placehold.co/400x400/ec4899/fff.png",
            "rating": 4.9,
            "inStock": False,
```

```
        },
    }

    product_id_upper = product_id.upper()

    if product_id_upper in products:
        product = products[product_id_upper]
        return {
            "status": "success",
            "report": f"Here are the details for {product['name']}. "
                      "I'll display it as a product card for you.",
            "product": product,
        }
    else:
        return {
            "status": "error",
            "report": f"Product {product_id} not found",
            "error": "Please check the product ID and try again.",
        }

# Create ADK agent with tools
adk_agent = Agent(
    name="customer_support_agent",
    model="gemini-2.0-flash-exp",
    instruction="""You are a helpful customer support agent for an e-commerce

Your responsibilities:
- Answer customer questions clearly and concisely
- Search the knowledge base when needed using search_knowledge_base()
- Look up order status using lookup_order_status() when customers ask about
  their orders
- Create support tickets using create_support_ticket() for complex issues
- Get product details using get_product_details() when customers ask about pro
- Be empathetic and professional
- Escalate complex issues to human support when appropriate
- Never make up information - if unsure, say so

IMPORTANT - Advanced Features:

1. **Product Information (Generative UI)**:
   - When users ask about products, follow this two-step process:
     a) First call get_product_details(product_id) to fetch product data
     b) Then call render_product_card(name, price, image, rating, inStock)
        with the product details
   - Example: "Show me product PROD-001"
     → call get_product_details("PROD-001")
     → extract the product data from the result
```

```
     → call render_product_card(name="Widget Pro", price=99.99, image="...",
        rating=4.5, inStock=True)
   - The frontend will render a beautiful interactive ProductCard component
   - IMPORTANT: Do NOT include the JSON data in your response. Just say someth
     "Here's the product information for [product name]" or "I've displayed th
   - Let the visual card speak for itself - don't repeat the data in text form

2. **Refunds (Human-in-the-Loop)**:
   - When users request refunds, call process_refund(order_id, amount, reason)
   - This is a FRONTEND action that requires user approval
   - An approval dialog will appear asking the user to confirm or cancel
   - The dialog shows: Order ID, Amount, and Reason
   - Wait for the user's decision before proceeding
   - If approved: Acknowledge "Refund processed successfully"
   - If cancelled: Acknowledge "Refund cancelled by user"
   - IMPORTANT: You must gather all three parameters (order_id, amount, reason

Guidelines:
- Greet customers warmly
- Use the appropriate tool for each type of query
- Offer next steps after answering
- Keep responses under 3 paragraphs unless more detail is requested
- Use a friendly but professional tone
- Format responses with markdown for better readability""",
    tools=[
        search_knowledge_base,
        lookup_order_status,
        create_support_ticket,
        get_product_details,
        # Note: process_refund is ONLY available as a frontend action (not bac
        # This ensures the HITL approval dialog is shown before processing
    ],
)

# Wrap ADK agent with AG-UI middleware
agent = ADKAgent(
    adk_agent=adk_agent,
    app_name="customer_support_app",
    user_id="demo_user",
    session_timeout_seconds=3600,
    use_in_memory_services=True,
)

# Create FastAPI app
app = FastAPI(title="Customer Support Agent API")

# Add CORS middleware for frontend
```

```
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:3000", "http://localhost:5173"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Add ADK endpoint for CopilotKit
add_adk_fastapi_endpoint(app, agent, path="/api/copilotkit")

# Health check endpoint
@app.get("/health")
def health_check():
    """Health check endpoint."""
    return {"status": "healthy", "agent": "customer_support_agent"}

# Run with: uvicorn agent:app --reload --port 8000
if __name__ == "__main__":
    port = int(os.getenv("PORT", "8000"))
    uvicorn.run(
        "agent:app",
        host="0.0.0.0",
        port=port,
        reload=True
    )
```

Create `agent/.env`:

```
GOOGLE_API_KEY=your_gemini_api_key_here
```

Step 5: Create Frontend

First, create a theme toggle component. Create `components/ThemeToggle.tsx`:

```
"use client";

  const [theme, setTheme] = useState<"light" | "dark">("light");

  useEffect(() => {
    // Check system preference and localStorage on mount
    const savedTheme = localStorage.getItem("theme") as "light" | "dark" | nul
    const systemTheme = window.matchMedia("(prefers-color-scheme: dark)")
      .matches
      ? "dark"
      : "light";
    const initialTheme = savedTheme || systemTheme;

    setTheme(initialTheme);
    document.documentElement.classList.toggle("dark", initialTheme === "dark")
  }, []);

  const toggleTheme = () => {
    const newTheme = theme === "light" ? "dark" : "light";
    setTheme(newTheme);
    localStorage.setItem("theme", newTheme);
    document.documentElement.classList.toggle("dark", newTheme === "dark");
  };

  return (
    <button
      onClick={toggleTheme}
      className="flex items-center justify-center w-9 h-9 rounded-lg border bo
      aria-label="Toggle theme"
    >
      {theme === "light" ? (
        <svg
          className="w-5 h-5 text-foreground"
          fill="none"
          stroke="currentColor"
          viewBox="0 0 24 24"
        >
          <path
            strokeLinecap="round"
            strokeLinejoin="round"
            strokeWidth={2}
            d="M20.354 15.354A9 9 0 018.646 3.646 9.003 9.003 0 0012 21a9.003
          />
        </svg>
      ) : (
        <svg
```

```
            className="w-5 h-5 text-foreground"
            fill="none"
            stroke="currentColor"
            viewBox="0 0 24 24"
          >
            <path
              strokeLinecap="round"
              strokeLinejoin="round"
              strokeWidth={2}
              d="M12 3v1m0 16v1m9-9h-1M4 12H3m15.364 6.364l-.707-.707M6.343 6.34
            />
          </svg>
        )}
      </button>
    );
  }
```

Update `app/globals.css` with minimal, clean styles:

```
@import "tailwindcss";

@layer base {
  :root {
    --background: 0 0% 100%;
    --foreground: 222.2 84% 4.9%;
    --card: 0 0% 100%;
    --card-foreground: 222.2 84% 4.9%;
    --popover: 0 0% 100%;
    --popover-foreground: 222.2 84% 4.9%;
    --primary: 221.2 83.2% 53.3%;
    --primary-foreground: 210 40% 98%;
    --secondary: 210 40% 96.1%;
    --secondary-foreground: 222.2 47.4% 11.2%;
    --muted: 210 40% 96.1%;
    --muted-foreground: 215.4 16.3% 46.9%;
    --accent: 210 40% 96.1%;
    --accent-foreground: 222.2 47.4% 11.2%;
    --destructive: 0 84.2% 60.2%;
    --destructive-foreground: 210 40% 98%;
    --border: 214.3 31.8% 91.4%;
    --input: 214.3 31.8% 91.4%;
    --ring: 221.2 83.2% 53.3%;
    --radius: 0.5rem;
  }

  .dark {
    --background: 222.2 84% 4.9%;
    --foreground: 210 40% 98%;
    --card: 222.2 84% 4.9%;
    --card-foreground: 210 40% 98%;
    --popover: 222.2 84% 4.9%;
    --popover-foreground: 210 40% 98%;
    --primary: 217.2 91.2% 59.8%;
    --primary-foreground: 222.2 47.4% 11.2%;
    --secondary: 217.2 32.6% 17.5%;
    --secondary-foreground: 210 40% 98%;
    --muted: 217.2 32.6% 17.5%;
    --muted-foreground: 215 20.2% 65.1%;
    --accent: 217.2 32.6% 17.5%;
    --accent-foreground: 210 40% 98%;
    --destructive: 0 62.8% 30.6%;
    --destructive-foreground: 210 40% 98%;
    --border: 217.2 32.6% 17.5%;
    --input: 217.2 32.6% 17.5%;
    --ring: 224.3 76.3% 48%;
```

```
  }
}

@layer base {
  * {
    border-color: hsl(var(--border));
  }

  body {
    background: hsl(var(--background));
    color: hsl(var(--foreground));
    font-feature-settings: "rlig" 1, "calt" 1;
  }
}
```

Update `app/layout.tsx`:

```
import "./globals.css";

const inter = Inter({ subsets: ["latin"] });

  title: "Customer Support Chat",
  description: "AI-powered customer support powered by Google ADK",
};

  children,
}: Readonly<{
  children: React.ReactNode;
}>) {
  return (
    <html lang="en">
      <body className={inter.className}>{children}</body>
    </html>
  );
}
```

Create `app/page.tsx`:

```
"use client";

import {
  CopilotKit,
  useCopilotReadable,
  useCopilotAction,
} from "@copilotkit/react-core";

import "@copilotkit/react-ui/styles.css";

/**
 * ChatInterface component with advanced features:
 * 1. Generative UI - Product cards rendered from agent responses
 * 2. Human-in-the-Loop - User approval for refunds
 * 3. Shared State - User context accessible to agent
 */
function ChatInterface() {
  // Feature 3: Shared State - User context that agent can read
  const [userData] = useState({
    name: "John Doe",
    email: "john@example.com",
    accountType: "Premium",
    orders: ["ORD-12345", "ORD-67890"],
    memberSince: "2023-01-15",
  });

  // Feature 1: Generative UI - State to hold product data for rendering
  const [currentProduct, setCurrentProduct] = useState<{
    name: string;
    price: number;
    image: string;
    rating: number;
    inStock: boolean;
  } | null>(null);

  // Make user data readable by agent
  useCopilotReadable({
    description: "Current user's account information and order history",
    value: userData,
  });

  // Feature 1: Generative UI - Frontend action that agent can call to render
  useCopilotAction({
    name: "render_product_card",
    available: "remote",
    description:
```

```
    "Render a product card in the chat interface with product details",
  parameters: [
    {
      name: "name",
      type: "string",
      description: "Product name",
      required: true,
    },
    {
      name: "price",
      type: "number",
      description: "Product price in USD",
      required: true,
    },
    {
      name: "image",
      type: "string",
      description: "Product image URL",
      required: true,
    },
    {
      name: "rating",
      type: "number",
      description: "Product rating (0-5)",
      required: true,
    },
    {
      name: "inStock",
      type: "boolean",
      description: "Product availability",
      required: true,
    },
  ],
  handler: async ({ name, price, image, rating, inStock }) => {
    // Update state to show the product card
    setCurrentProduct({ name, price, image, rating, inStock });

    return `Product card displayed successfully for ${name}`;
  },
  render: ({ args, status }) => {
    if (status !== "complete") {
      return (
        <div className="p-4 border rounded-lg animate-pulse bg-card">
          <div className="h-48 bg-muted rounded mb-4"></div>
          <div className="h-4 bg-muted rounded w-3/4 mb-2"></div>
          <div className="h-4 bg-muted rounded w-1/2"></div>
        </div>
```

```
        );
      }

      return (
        <div className="my-4">
          <ProductCard
            name={args.name}
            price={args.price}
            image={args.image}
            rating={args.rating}
            inStock={args.inStock}
          />
        </div>
      );
    },
  });

  // Feature 2: Human-in-the-Loop - Refund approval
  const [refundRequest, setRefundRequest] = useState<{
    order_id: string;
    amount: number;
    reason: string;
  } | null>(null);

  // Frontend-only action that shows approval dialog
  useCopilotAction({
    name: "process_refund",
    available: "remote",
    description: "Process a refund after user approval",
    parameters: [
      {
        name: "order_id",
        type: "string",
        description: "Order ID to refund",
        required: true,
      },
      {
        name: "amount",
        type: "number",
        description: "Refund amount",
        required: true,
      },
      {
        name: "reason",
        type: "string",
        description: "Refund reason",
        required: true,
```

```
      },
    ],
    handler: async ({ order_id, amount, reason }) => {
      setRefundRequest({ order_id, amount, reason });

      // Return a promise that resolves when user approves/cancels
      return new Promise((resolve) => {
        (window as any).__refundPromiseResolve = resolve;
      });
    },
    render: ({ args, status }) => {
      if (status !== "complete") {
        return (
          <div className="p-5 border-2 border-yellow-300 dark:border-yellow-70
            <div className="flex items-center gap-3">
              <div className="w-10 h-10 bg-yellow-500 rounded-full flex items-
                <svg
                  className="w-6 h-6 text-white"
                  fill="none"
                  stroke="currentColor"
                  viewBox="0 0 24 24"
                >
                  <path
                    strokeLinecap="round"
                    strokeLinejoin="round"
                    strokeWidth={2.5}
                    d="M12 8v4l3 3m6-3a9 9 0 11-18 0 9 9 0 0118 0z"
                  />
                </svg>
              </div>
              <div>
                <h4 className="font-bold text-lg text-yellow-900 dark:text-yel
                  Awaiting Your Approval
                </h4>
                <p className="text-sm text-yellow-700 dark:text-yellow-300">
                  Please review the modal dialog above
                </p>
              </div>
            </div>
          </div>
        );
      }

      return (
        <div className="p-4 border-2 border-green-300 dark:border-green-700 ro
          <div className="w-10 h-10 bg-green-500 rounded-full flex items-cente
            <svg
```

```
              className="w-5 h-5 text-white"
              fill="none"
              stroke="currentColor"
              viewBox="0 0 24 24"
            >
              <path
                strokeLinecap="round"
                strokeLinejoin="round"
                strokeWidth={2.5}
                d="M5 13l4 4L19 7"
              />
            </svg>
          </div>
          <div>
            <p className="font-semibold text-green-900 dark:text-green-100">
              Decision Recorded
            </p>
            <p className="text-sm text-green-700 dark:text-green-300">
              Processing your choice...
            </p>
          </div>
        </div>
      );
    },
  });

  // Render approval dialog when refundRequest is set
  const handleRefundApproval = async (approved: boolean) => {
    const resolve = (window as any).__refundPromiseResolve;
    if (resolve && refundRequest) {
      if (approved) {
        resolve({
          approved: true,
          message: `Refund processed successfully for order ${refundRequest.or
        });
      } else {
        resolve({
          approved: false,
          message: "Refund cancelled by user",
        });
      }
    }

    setRefundRequest(null);
    delete (window as any).__refundPromiseResolve;
  };
```

```
  // Keyboard support for modal
  useEffect(() => {
    const handleKeyDown = (e: KeyboardEvent) => {
      if (refundRequest) {
        if (e.key === "Escape") {
          e.preventDefault();
          handleRefundApproval(false);
        } else if (e.key === "Enter" && !e.shiftKey) {
          e.preventDefault();
          handleRefundApproval(true);
        }
      }
    };

    window.addEventListener("keydown", handleKeyDown);
    return () => window.removeEventListener("keydown", handleKeyDown);
  }, [refundRequest]);

  return (
    <div className="flex flex-col min-h-screen">
      {/* HITL Approval Dialog */}
      {refundRequest && (
        <div
          className="fixed inset-0 bg-black/80 flex items-center justify-cente
          onClick={(e) => {
            if (e.target === e.currentTarget) {
              handleRefundApproval(false);
            }
          }}
        >
          <div className="bg-white dark:bg-gray-900 border border-gray-200 dar
            <div className="flex items-start gap-4 mb-6">
              <div className="w-14 h-14 bg-yellow-400 rounded-full flex items-
                <svg
                  className="w-8 h-8 text-gray-900"
                  fill="none"
                  stroke="currentColor"
                  viewBox="0 0 24 24"
                >
                  <path
                    strokeLinecap="round"
                    strokeLinejoin="round"
                    strokeWidth={2.5}
                    d="M12 9v2m0 4h.01m-6.938 4h13.856c1.54 0 2.502-1.667 1.73
                  />
                </svg>
              </div>
```

```
            <div className="flex-1">
              <h2 className="text-2xl font-bold text-gray-900 dark:text-gray
                Refund Approval Required
              </h2>
              <p className="text-sm text-gray-600 dark:text-gray-400">
                Please review the details below carefully
              </p>
            </div>
          </div>

          <div className="space-y-3 bg-gray-50 dark:bg-gray-800 rounded-lg p
            <div className="flex justify-between items-center py-2 border-b
              <span className="text-sm font-medium text-gray-600 dark:text-g
                Order ID
              </span>
              <span className="text-sm font-mono font-semibold text-gray-900
                {refundRequest.order_id}
              </span>
            </div>
            <div className="flex justify-between items-center py-2 border-b
              <span className="text-sm font-medium text-gray-600 dark:text-g
                Refund Amount
              </span>
              <span className="text-2xl font-bold text-gray-900 dark:text-gr
                ${refundRequest.amount.toFixed(2)}
              </span>
            </div>
            <div className="pt-2">
              <span className="text-sm font-medium text-gray-600 dark:text-g
                Reason
              </span>
              <div className="text-sm text-gray-900 dark:text-gray-100 bg-wh
                {refundRequest.reason}
              </div>
            </div>
          </div>

          <div className="flex items-start gap-3 mb-6 p-4 bg-yellow-50 dark:
            <svg
              className="w-5 h-5 text-yellow-600 dark:text-yellow-400 flex-s
              fill="currentColor"
              viewBox="0 0 20 20"
            >
              <path
                fillRule="evenodd"
                d="M18 10a8 8 0 11-16 0 8 8 0 0116 0zm-7-4a1 1 0 11-2 0 1 1
                clipRule="evenodd"
```

```
                />
              </svg>
              <p className="text-sm text-yellow-900 dark:text-yellow-100 font-
                This action cannot be undone. Approving will process the refun
                immediately.
              </p>
            </div>

            <div className="flex gap-4">
              <button
                onClick={() => handleRefundApproval(false)}
                className="flex-1 px-6 py-3.5 bg-gray-200 hover:bg-gray-300 da
              >
                Cancel
              </button>
              <button
                onClick={() => handleRefundApproval(true)}
                className="flex-1 px-6 py-3.5 bg-green-600 hover:bg-green-700
              >
                Approve Refund
              </button>
            </div>

            <p className="text-xs text-center text-gray-500 dark:text-gray-400
              Press{" "}
              <kbd className="px-2 py-1 bg-gray-100 dark:bg-gray-800 border bo
                ESC
              </kbd>{" "}
              to cancel
            </p>
          </div>
        </div>
      )}

      {/* Header */}
      <header className="border-b">
        <div className="container mx-auto px-4 py-4">
          <div className="flex items-center justify-between">
            <div className="flex items-center gap-3">
              <div className="flex items-center justify-center w-10 h-10 bg-pr
                <svg
                  className="w-5 h-5 text-primary-foreground"
                  fill="none"
                  stroke="currentColor"
                  viewBox="0 0 24 24"
                >
                  <path
```

```
                  strokeLinecap="round"
                  strokeLinejoin="round"
                  strokeWidth={2}
                  d="M8 10h.01M12 10h.01M16 10h.01M9 16H5a2 2 0 01-2-2V6a2 2
                />
              </svg>
            </div>
            <div>
              <h1 className="text-lg font-semibold">Support Assistant</h1>
              <p className="text-xs text-muted-foreground">
                AI-Powered Help • Logged in as {userData.name}
              </p>
            </div>
          </div>
          <div className="flex items-center gap-3">
            <ThemeToggle />
          </div>
        </div>
      </div>
    </header>

    {/* Main Content */}
    <main className="flex-1">
      <div className="container mx-auto px-4 py-6 h-[600px]">
        <div className="h-full border rounded-lg bg-card">
          <CopilotChat
            instructions="You are a friendly and professional customer suppo
            labels={{
              title: "Support Chat",
              initial:
                "👋 Hi! I'm your AI support assistant.\n\n" +
                "**Try these example prompts:**\n\n" +
                "🎨 **Generative UI**\n" +
                '• "Show me product PROD-001"\n' +
                '• "Display product PROD-002"\n\n' +
                "🔐 **Human-in-the-Loop**\n" +
                '• "I want a refund for order ORD-12345"\n' +
                '• "Process a refund for my purchase"\n\n' +
                "👤 **Shared State**\n" +
                '• "What\'s my account status?"\n' +
                '• "Show me my recent orders"\n\n' +
                "📦 **General Support**\n" +
                '• "What is your refund policy?"\n' +
                '• "Track my order ORD-67890"\n' +
                '• "I need help with a billing issue"\n\n' +
                "💡 *Scroll down to see interactive demos of all features!*"
            }}
```

```
              className="h-full"
            />
          </div>
        </div>
      </main>
    </div>
  );
}

  return (
    <div className="min-h-screen bg-background">
      <CopilotKit runtimeUrl="/api/copilotkit" agent="customer_support_agent">
        <ChatInterface />
      </CopilotKit>
    </div>
  );
}
```

Step 6: Run Everything

```
# Start both backend and frontend servers
make dev

# Or run separately:
# Terminal 1: Backend
make dev-backend

# Terminal 2: Frontend
make dev-frontend
```

Open http://localhost:3000 - Your custom support agent is live! 🚀

Understanding the Architecture

Component Diagram

USER'S BROWSER
Next.js 15 App (Port 3000)
├─ app/page.tsx
│ └─ <CopilotKit> provider
│   └─ <CopilotChat> component
└─ @copilotkit/react-core (TypeScript SDK)
   ├─ WebSocket connection
   ├─ Message streaming
   └─ State management
AG-UI Protocol (WebSocket/SSE)
BACKEND SERVER (Port 8000)
ag_ui_adk (AG-UI Middleware)
├─ FastAPI app
├─ /api/copilotkit endpoint
├─ AG-UI protocol adapter
└─ Session management
ADKAgent (wrapper)
├─ app_name: "customer_support_app"
├─ user_id & session management
└─ Wraps LlmAgent
Google ADK LlmAgent
├─ model: "gemini-2.5-flash"
├─ instruction: System prompt
└─ tools: [search_knowledge_base, lookup_order,
create_support_ticket]
Gemini API

```
|              GEMINI 2.0 FLASH                                    |
|  ├─ Text generation                                              |
|  ├─ Function calling                                             |
|  └─ Streaming responses                                          |
└──────────────────────────────────────────────────────────────────┘
```

Request Flow

1. User sends message: "What's your refund policy?"

2. Frontend (`<CopilotChat>`):

```
// Message sent via WebSocket
{
  type: "textMessage",
  content: "What's your refund policy?",
  sessionId: "user-123"
}
```

3. AG-UI Middleware (ag_ui_adk):

```
# ADKAgent wraps your LlmAgent
# Translates AG-UI Protocol → ADK format
# Manages sessions with timeout
# Handles tool execution
# add_adk_fastapi_endpoint() creates /api/copilotkit endpoint
```

4. ADK Agent:

```
# Agent processes message
# Decides to call search_knowledge_base tool
# Executes tool with query="refund policy"
# Generates response with knowledge base result
```

5. Gemini 2.0 Flash:

```
System: You are a customer support agent...
User: What's your refund policy?
Function Call: search_knowledge_base(query="refund policy")
Function Result: {"title": "Refund Policy", "content": "We offer..."}
Agent: "Our refund policy is...
```

6. Response streams back:

```
// Frontend receives chunks
{
  type: "textMessageChunk",
  content: "Our refund policy"
}
{
  type: "textMessageChunk",
  content: " is very customer-friendly..."
}
```

7. User sees response progressively rendering in real-time!

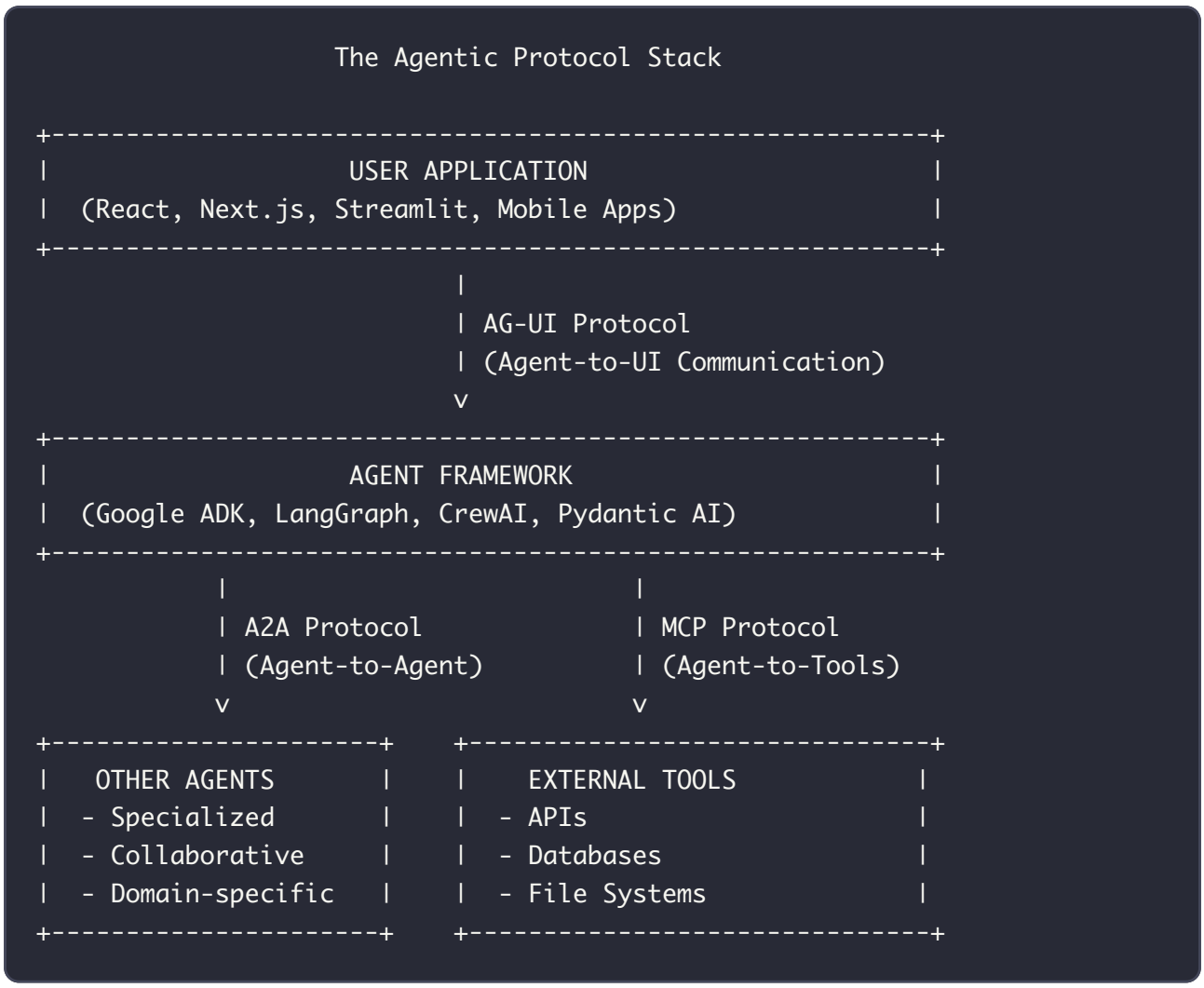
Understanding AG-UI Protocol

AG-UI (Agent-User Interaction Protocol) is an open, lightweight, event-based protocol that standardizes how AI agents connect to user-facing applications.

What is AG-UI?

AG-UI is complementary to other agentic protocols in the ecosystem:

- **MCP** (Model Context Protocol) - Gives agents tools
- **A2A** (Agent2Agent) - Allows agents to communicate with other agents
- **AG-UI** - Brings agents into user-facing applications

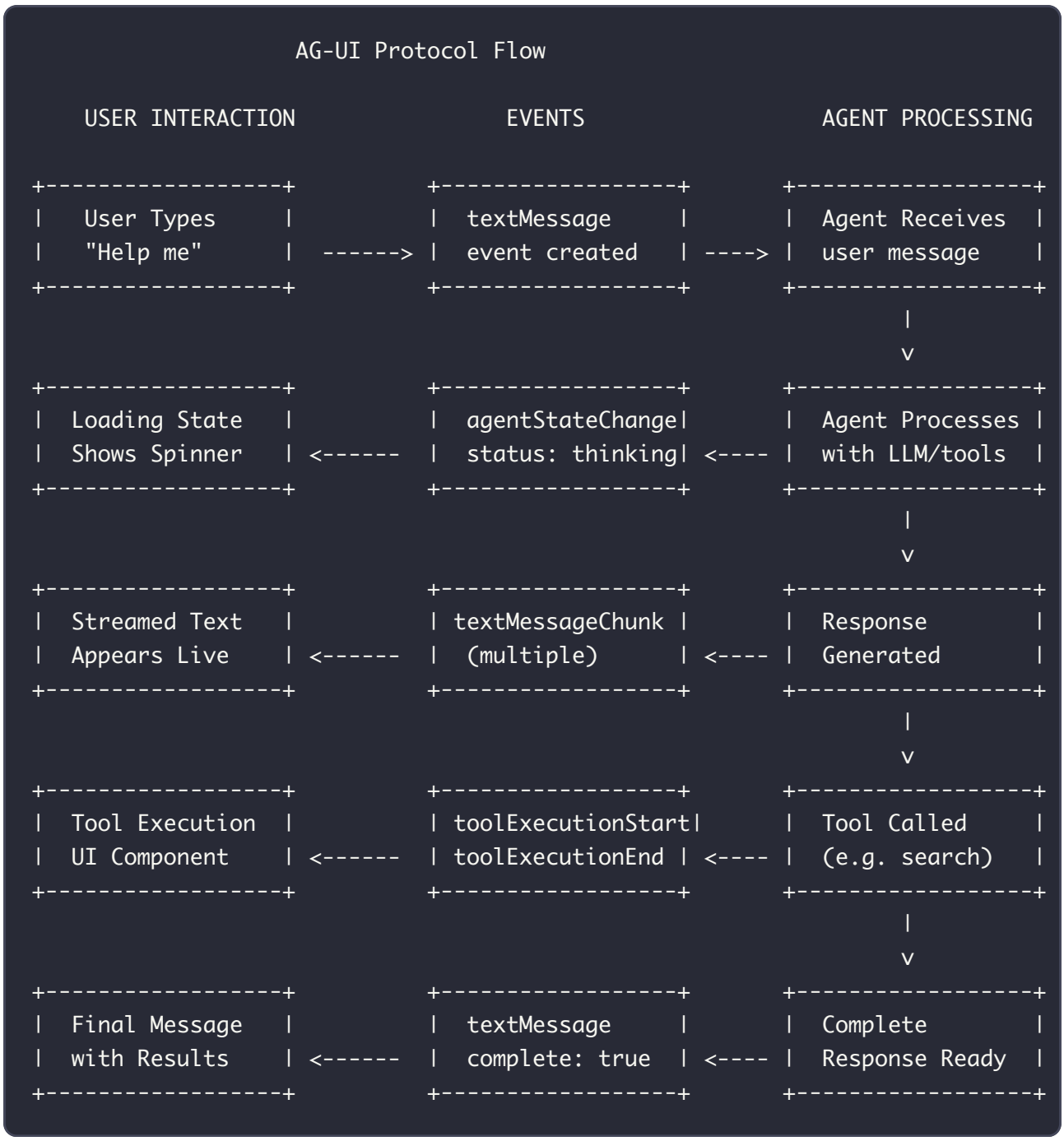


Key Features

- 💬 **Real-time Communication**: Streaming responses via WebSocket/SSE
- 🔄 **Bi-directional State**: Sync state between agent and frontend
- 🧩 **Generative UI**: Render custom React components from agent responses
- 🧠 **Context Enrichment**: Share application state with agents in real-time
- 🛠️ **Frontend Tools**: Execute frontend actions from agent workflows
- 🧑‍💻 **Human-in-the-Loop**: Built-in approval flows for sensitive actions

How It Works

1. **Agent Backend** emits events compatible with AG-UI's ~16 standard event types
2. **Middleware Layer** translates between agent framework (ADK) and frontend
3. **Frontend SDK** receives events and updates UI in real-time
4. **Transport Agnostic**: Works with WebSocket, SSE, or webhooks



Framework Support

AG-UI supports 15+ agent frameworks with official partnerships:

Framework	Status	Type
Google ADK	✔️ Supported	Partnership
LangGraph	✔️ Supported	Partnership
CrewAI	✔️ Supported	Partnership
Pydantic AI	✔️ Supported	1st party
Mastra	✔️ Supported	1st party
LlamaIndex	✔️ Supported	1st party
AG2	✔️ Supported	1st party
Vercel AI SDK	🛠️ In Progress	Community
OpenAI Agent SDK	🛠️ In Progress	Community

View all supported frameworks → (https://docs.ag-ui.com/introduction#supported-frameworks)

Licensing

- **AG-UI Protocol**: MIT License (https://github.com/ag-ui-protocol/ag-ui/blob/main/LICENSE) - Open source, free for commercial use
- **CopilotKit**: MIT License (https://github.com/CopilotKit/CopilotKit/blob/main/LICENSE) - Open source, free for commercial use
- **Google ADK**: Apache 2.0 License (https://github.com/google/adk-python/blob/main/LICENSE) - Open source, free for commercial use

All components in this tutorial are **fully open source** with permissive licenses suitable for commercial applications.

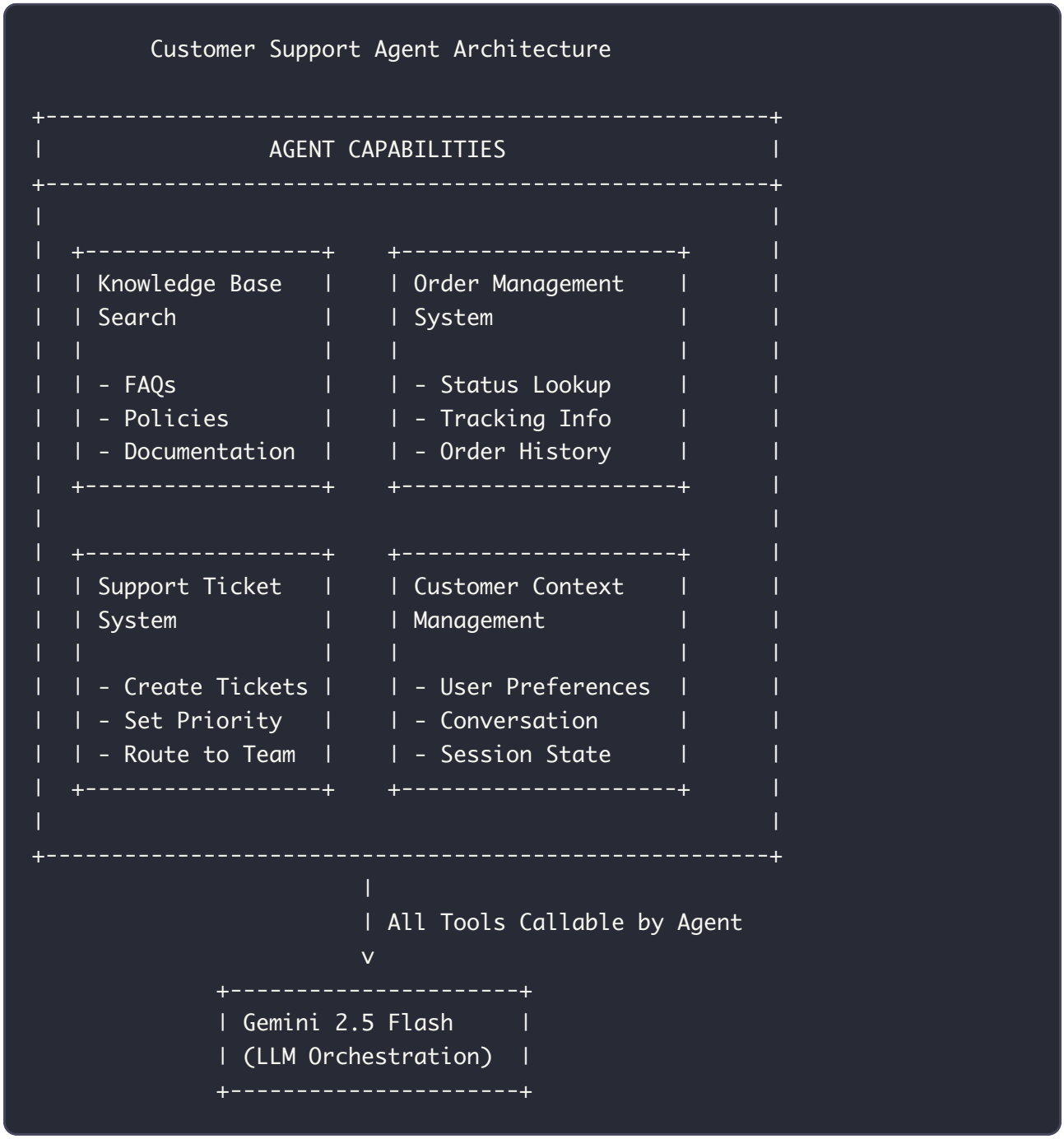
Learn More

- AG-UI Official Documentation (https://ag-ui.com/)
- AG-UI GitHub Repository (https://github.com/ag-ui-protocol/ag-ui)
- AG-UI Dojo (Interactive Examples) (https://dojo.ag-ui.com/)
- CopilotKit Documentation (https://docs.copilotkit.ai/)

Building a Customer Support Agent

Enhancing the Agent

Let's add more realistic features to our support agent.



Feature 1: Order Status Lookup

Update `agent/agent.py`:

```python
def lookup_order_status(order_id: str) -> Dict[str, str]:
    """
    Look up the status of an order.

    Args:
        order_id: The order ID to look up (format: ORD-XXXXX)

    Returns:
        Dict with order status details
    """
    # Mock order database - replace with real database
    orders = {
        "ORD-12345": {
            "status": "Shipped",
            "tracking": "1Z999AA10123456784",
            "estimated_delivery": "2025-10-12",
            "items": "2x Widget Pro, 1x Gadget Plus"
        },
        "ORD-67890": {
            "status": "Processing",
            "tracking": None,
            "estimated_delivery": "2025-10-15",
            "items": "1x Premium Kit"
        }
    }

    order_id_upper = order_id.upper()

    if order_id_upper in orders:
        return orders[order_id_upper]
    else:
        return {
            "status": "Not Found",
            "message": f"Order {order_id} not found. Please check the order ID
        }

# Add to agent tools - note: for testing purposes, showing function reference
# In actual implementation, tools are added to Agent constructor
from google.adk.agents import Agent

agent = Agent(
    model="gemini-2.0-flash-exp",
    name="customer_support_agent",
    instruction="""...""",  # Same as before
    tools=[lookup_order_status]  # Add function directly
)
```

```
# If using genai.Tool for testing:
# Tool(
#     function_declarations=[
#         # ... search_knowledge_base (as before)
                FunctionDeclaration(
                    name="lookup_order_status",
                    description="Look up the status and tracking information f
                    parameters={
                        "type": "object",
                        "properties": {
                            "order_id": {
                                "type": "string",
                                "description": "The order ID in format ORD-XXX
                            }
                        },
                        "required": ["order_id"]
                    }
                )
            ]
        )
    ],
    tool_config={"function_calling_config": {"mode": "AUTO"}}
)

# Update runtime tools
app = create_copilotkit_runtime(
    agent=agent,
    tools={
        "search_knowledge_base": search_knowledge_base,
        "lookup_order_status": lookup_order_status
    }
)
```

Test it:

User: "What's the status of my order ORD-12345?"

Agent: "Your order ORD-12345 has been shipped! Here are the details:

- Status: Shipped
- Tracking: 1Z999AA10123456784
- Estimated Delivery: October 12, 2025
- Items: 2x Widget Pro, 1x Gadget Plus

You can track your package using the tracking number above. Is there anything else I can help you with?"

Feature 2: Create Support Ticket

Add escalation capability:

```
import uuid
from datetime import datetime

def create_support_ticket(
    issue_type: str,
    description: str,
    priority: str = "normal"
) -> Dict[str, str]:
    """
    Create a support ticket for issues that need human attention.

    Args:
        issue_type: Type of issue (billing, technical, account, other)
        description: Detailed description of the issue
        priority: Priority level (low, normal, high, urgent)

    Returns:
        Dict with ticket ID and estimated response time
    """
    ticket_id = f"TKT-{uuid.uuid4().hex[:8].upper()}"

    # Mock ticket creation - replace with real ticketing system API
    response_times = {
        "urgent": "1-2 hours",
        "high": "4-6 hours",
        "normal": "12-24 hours",
        "low": "24-48 hours"
    }

    return {
        "ticket_id": ticket_id,
        "status": "Created",
        "priority": priority,
        "estimated_response": response_times.get(priority, "24 hours"),
        "created_at": datetime.now().isoformat(),
        "message": f"Ticket {ticket_id} created successfully. Our support team
    }

# Add to tools
FunctionDeclaration(
    name="create_support_ticket",
    description="Create a support ticket for complex issues that need human ag
    parameters={
        "type": "object",
        "properties": {
            "issue_type": {
```

```
                "type": "string",
                "description": "Type of issue",
                "enum": ["billing", "technical", "account", "other"]
            },
            "description": {
                "type": "string",
                "description": "Detailed description of the issue"
            },
            "priority": {
                "type": "string",
                "description": "Priority level",
                "enum": ["low", "normal", "high", "urgent"],
                "default": "normal"
            }
        },
        "required": ["issue_type", "description"]
    }
)

# Update runtime
app = create_copilotkit_runtime(
    agent=agent,
    tools={
        "search_knowledge_base": search_knowledge_base,
        "lookup_order_status": lookup_order_status,
        "create_support_ticket": create_support_ticket
    }
)
```

Test it:

User: "My product stopped working after 2 months and warranty doesn't seem to cover it"

Agent: "I understand how frustrating that must be. Let me create a support ticket for our specialist team to review your warranty coverage.

Creates ticket TKT-A1B2C3D4

I've created ticket TKT-A1B2C3D4 for you with high priority. Our specialized support team will reach out within 4-6 hours to review your case and warranty details.

In the meantime, have you tried:

- Checking if firmware updates are available
- Performing a factory reset (if applicable)

Is there anything else I can help you with while you wait?"

Adding Personality & Context

Make your agent more engaging:

```
from google.adk.agents import Agent

agent = Agent(
    model="gemini-2.0-flash-exp",
    name="customer_support_agent",
    instruction="""You are Jamie, a friendly and knowledgeable customer suppor

Your personality:
- Warm and empathetic, but professional
- Patient and understanding with frustrated customers
- Enthusiastic about helping solve problems
- Use occasional (appropriate) emojis to be friendly 😊
- Remember context from the conversation

Your responsibilities:
1. Answer product and policy questions using the knowledge base
2. Look up order status when customers provide order IDs
3. Create support tickets for complex issues
4. Escalate urgent problems immediately
5. Never make up information - if unsure, check knowledge base or create ticke

Guidelines:
- Greet returning customers warmly
- Acknowledge frustration with empathy
- Offer proactive solutions
- End with "Is there anything else I can help with?"
- Keep responses concise but complete
- Use bullet points for clarity

Company values:
- Customer satisfaction is our top priority
- We stand behind our products
- Transparency in all communications

Remember: You represent TechCo's commitment to excellent customer service!""",
    tools=[...],  # Same tools as before
    tool_config={"function_calling_config": {"mode": "AUTO"}}
)
```

Advanced Features

:::tip Complete Implementation Available

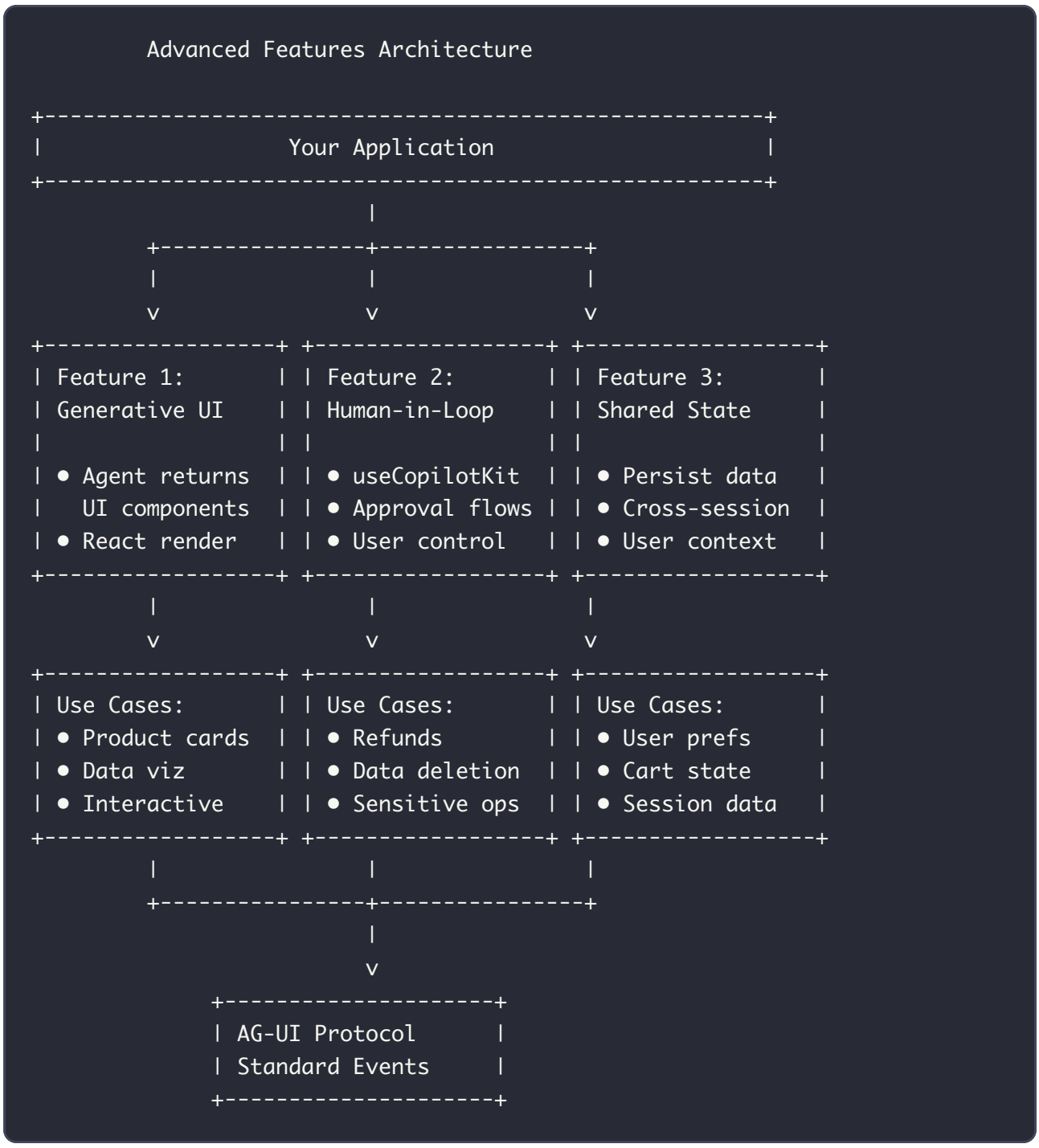
All three advanced features are **fully implemented** in the working example at `/tutorial_implementation/tutorial30/nextjs_frontend/app/page.tsx`.

Try them now:

```
cd tutorial_implementation/tutorial30
make dev
# Open http://localhost:3001
```

- 🎨 **Generative UI**: "Show me product PROD-001" → Beautiful product card renders
- 🔐 **Human-in-the-Loop**: "I want a refund for ORD-12345" → Approval modal appears
- 👤 **Shared State**: "What's my account status?" → Agent knows you're John Doe

:::



Feature 1: Generative UI

:::success Fully Implemented in Tutorial 30

The working Generative UI implementation renders beautiful product cards:

- ✔️ **ProductCard component** with responsive design
- ✔️ **useCopilotAction** registration with proper render function
- ✔️ **Dynamic content** with product images, pricing, ratings

- ✔️ **Dark mode support** with Tailwind classes

Try it:

```
cd tutorial_implementation/tutorial30
make dev
# Chat: "Show me product PROD-001"
# Beautiful product card renders inline! 🎨
```

Implementation: `nextjs_frontend/app/page.tsx` (lines 45-89), `components/ProductCard.tsx`

:::

Render custom React components directly from agent responses.

Frontend Implementation (`app/page.tsx`):

```
"use client";

function ChatInterface() {
  // State to store product data when agent calls action
  const [currentProduct, setCurrentProduct] = useState<any>(null);

  // Register action that agent can call to render product cards
  useCopilotAction({
    name: "render_product_card",
    available: "remote", // Agent calls this from backend
    description: "Render a product card UI component",
    parameters: [
      { name: "product_id", type: "string", description: "Product ID" },
      { name: "name", type: "string", description: "Product name" },
      { name: "price", type: "number", description: "Product price" },
      { name: "image", type: "string", description: "Image URL" },
      { name: "rating", type: "number", description: "Rating 0-5" },
      { name: "in_stock", type: "boolean", description: "Stock status" },
    ],
    handler: async ({ product_id, name, price, image, rating, in_stock }) => {
      // Store product data to trigger render
      setCurrentProduct({ product_id, name, price, image, rating, in_stock });

      return `Product card rendered for ${name}`;
    },
    // Render function shows the UI in chat
    render: ({ status, result }) => (
      <div className="my-4 animate-fade-in">
        {status === "executing" && (
          <div className="flex items-center gap-2">
            <div className="w-5 h-5 border-2 border-blue-500 border-t-transpar
            <span>Loading product...</span>
          </div>
        )}
        {status === "complete" && currentProduct && (
          <ProductCard {...currentProduct} />
        )}
      </div>
    ),
  });

  return <CopilotChat />;
}
```

Product Component (`components/ProductCard.tsx`):

```
interface ProductCardProps {
  name: string;
  price: number;
  image: string;
  rating: number;
  in_stock: boolean;
}

  name,
  price,
  image,
  rating,
  in_stock,
}: ProductCardProps) {
  return (
    <div className="border border-gray-200 dark:border-gray-700 rounded-xl p-5
      <div className="relative w-full h-48 mb-4 rounded-lg overflow-hidden">
        <Image src={image} alt={name} fill className="object-cover" />
      </div>

      <h3 className="font-bold text-xl mb-2">{name}</h3>

      <div className="flex items-center justify-between mb-3">
        <span className="text-3xl font-bold text-green-600 dark:text-green-500
          ${price.toFixed(2)}
        </span>
        <span className="text-yellow-500 flex items-center gap-1">
          ⭐ {rating.toFixed(1)}
        </span>
      </div>

      {in_stock ? (
        <span className="inline-block px-4 py-2 bg-green-100 dark:bg-green-900
          ✓ In Stock
        </span>
      ) : (
        <span className="inline-block px-4 py-2 bg-red-100 dark:bg-red-900 tex
          ✗ Out of Stock
        </span>
      )}
    </div>
  );
}
```

Backend Agent (`agent/agent.py`):

```
# Agent uses the action but doesn't define it
# The action is frontend-only, just like process_refund

# When user asks about products, agent calls:
# get_product_details(product_id) to fetch data
# Then render_product_card(name, price, image, rating, inStock) to display

# Beautiful ProductCard component appears in chat! 🎨
```

How It Works:

1. User: "Show me product PROD-001"
2. Agent calls `get_product_details("PROD-001")` to fetch product data
3. Agent extracts product details from response
4. Agent calls `render_product_card(name, price, image, rating, inStock)`
5. Frontend handler receives data, stores in `currentProduct` state
6. Render function displays `<ProductCard>` component inline in chat
7. User sees beautiful, interactive product card with image, price, rating

Now when agent mentions products, gorgeous cards render inline! 🎨

Feature 2: Human-in-the-Loop (HITL)

:::success Fully Implemented in Tutorial 30

The working HITL implementation includes:

- ✔️ **Professional modal dialog** with solid design
- ✔️ **Keyboard shortcuts** (ESC to cancel, Enter to approve)
- ✔️ **Promise-based flow** that blocks agent until user decides
- ✔️ **Click-outside-to-close** functionality
- ✔️ **Full dark mode support**

See it in action:

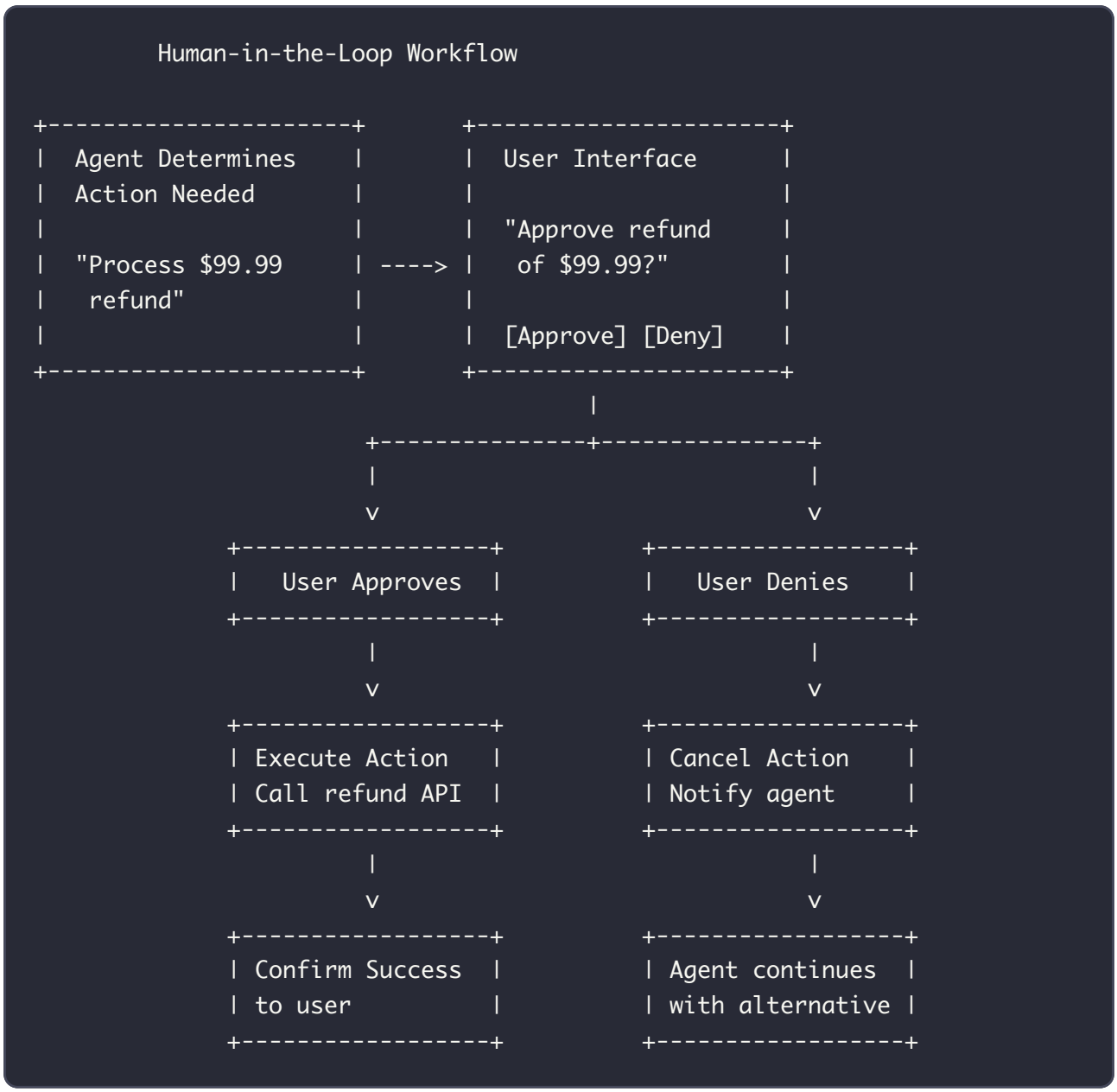
```
cd tutorial_implementation/tutorial30
make dev
# Chat: "I want a refund for ORD-12345"
# Provide: Amount "100", Reason "Items arrived broken"
# Beautiful modal appears for approval! 🎉
```

Implementation details:

- Frontend: `nextjs_frontend/app/page.tsx` (lines 99-279)
- Backend: Agent does NOT have `process_refund` tool (frontend-only action)
- Pattern: `available: "remote"` + Promise + React state + modal overlay

:::

Let users approve sensitive actions with a professional approval modal:



Key Implementation Pattern:

The HITL implementation uses a **frontend-only action** pattern:

1. **Backend** (`agent/agent.py`): Does NOT include `process_refund` in tools list
2. **Frontend** (`app/page.tsx`): Implements `process_refund` with `available: "remote"`
3. **Flow**: Agent calls action → Frontend handler → Modal shows → User decides → Promise resolves → Agent continues

Frontend Implementation (Professional Modal):

```
"use client";

function ChatInterface() {
  // State to manage the approval dialog
  const [refundRequest, setRefundRequest] = useState<{
    order_id: string;
    amount: number;
    reason: string;
  } | null>(null);

  // Frontend-only action that agent can call
  useCopilotAction({
    name: "process_refund",
    available: "remote", // Frontend-only, not a backend tool
    description: "Process a refund after user approval",
    parameters: [
      { name: "order_id", type: "string", description: "Order ID" },
      { name: "amount", type: "number", description: "Refund amount" },
      { name: "reason", type: "string", description: "Refund reason" },
    ],
    handler: async ({ order_id, amount, reason }) => {
      console.log("🔍 HITL handler called with:", { order_id, amount, reason }

      // Store the refund request to show in the dialog
      setRefundRequest({ order_id, amount, reason });

      // Return a promise that resolves when user approves/cancels
      return new Promise((resolve) => {
        // We'll resolve this in the dialog buttons
        (window as any).__refundPromiseResolve = resolve;
      });
    },
    render: ({ args, status }) => {
      console.log("🔍 HITL render - Status:", status, "Args:", args);

      if (status !== "complete") {
        // Show loading while waiting for user decision
        return (
          <div className="p-5 border-2 border-yellow-300 dark:border-yellow-70
            <div className="flex items-center gap-3">
              <div className="w-10 h-10 bg-yellow-500 rounded-full flex items-
                <svg
                  className="w-6 h-6 text-white"
                  fill="none"
                  stroke="currentColor"
                  viewBox="0 0 24 24"
```

```
              >
                <path
                  strokeLinecap="round"
                  strokeLinejoin="round"
                  strokeWidth={2}
                  d="M12 8v4l3 3m6-3a9 9 0 11-18 0 9 9 0 0118 0z"
                />
              </svg>
            </div>
            <div>
              <h4 className="font-bold text-lg text-yellow-900 dark:text-yel
                Awaiting Your Approval
              </h4>
              <p className="text-sm text-yellow-700 dark:text-yellow-300">
                Please review the modal dialog above
              </p>
            </div>
          </div>
          <div className="pl-13 space-y-1">
            <div className="flex items-center gap-2 text-sm text-yellow-800
              <div className="w-2 h-2 bg-yellow-500 rounded-full animate-pul
              <span>
                Order: <strong>{args.order_id}</strong>
              </span>
            </div>
            <div className="flex items-center gap-2 text-sm text-yellow-800
              <div
                className="w-2 h-2 bg-yellow-500 rounded-full animate-pulse"
                style={{ animationDelay: "0.2s" }}
              ></div>
              <span>
                Amount: <strong>${args.amount}</strong>
              </span>
            </div>
          </div>
        </div>
      );
    }

    return (
      <div className="p-4 border-2 border-green-300 dark:border-green-700 ro
        <div className="w-10 h-10 bg-green-500 rounded-full flex items-cente
          <svg
            className="w-6 h-6 text-white"
            fill="none"
            stroke="currentColor"
            viewBox="0 0 24 24"
```

```
            >
              <path
                strokeLinecap="round"
                strokeLinejoin="round"
                strokeWidth={2.5}
                d="M5 13l4 4L19 7"
              />
            </svg>
          </div>
          <div>
            <p className="font-semibold text-green-900 dark:text-green-100">
              Decision Recorded
            </p>
            <p className="text-sm text-green-700 dark:text-green-300">
              Processing your choice...
            </p>
          </div>
        </div>
      );
    },
  });

  // Render approval dialog when refundRequest is set
  const handleRefundApproval = async (approved: boolean) => {
    console.log("🔍 User decision:", approved ? "APPROVED" : "CANCELLED");

    const resolve = (window as any).__refundPromiseResolve;
    if (resolve && refundRequest) {
      if (approved) {
        // Call backend API to actually process the refund
        try {
          const response = await fetch("http://localhost:8000/api/copilotkit",
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({
              action: "process_refund_backend",
              params: refundRequest,
            }),
          });
          const result = await response.json();
          resolve({
            approved: true,
            message: `Refund processed successfully for order ${refundRequest.
          });
        } catch (error) {
          resolve({
            approved: true,
```

```
            message: `Refund approved for order ${refundRequest.order_id} - $$
          });
        }
      } else {
        resolve({
          approved: false,
          message: "Refund cancelled by user",
        });
      }
    }

    setRefundRequest(null);
    delete (window as any).__refundPromiseResolve;
  };

  // Keyboard support for modal (ESC to cancel, Enter to approve)
  useEffect(() => {
    const handleKeyDown = (e: KeyboardEvent) => {
      if (refundRequest) {
        if (e.key === "Escape") {
          e.preventDefault();
          handleRefundApproval(false);
        } else if (e.key === "Enter" && !e.shiftKey) {
          e.preventDefault();
          handleRefundApproval(true);
        }
      }
    };

    window.addEventListener("keydown", handleKeyDown);
    return () => window.removeEventListener("keydown", handleKeyDown);
  }, [refundRequest]);

  return (
    <div>
      {/* HITL Approval Dialog - Enhanced UX Modal */}
      {refundRequest && (
        <div
          className="fixed inset-0 bg-black/80 flex items-center justify-cente
          onClick={(e) => {
            // Close modal if clicking backdrop
            if (e.target === e.currentTarget) {
              handleRefundApproval(false);
            }
          }}
        >
          <div className="bg-white dark:bg-gray-900 border border-gray-200 dar
```

```
            {/* Header with icon */}
            <div className="flex items-start gap-4 mb-6">
              <div className="flex-shrink-0 w-14 h-14 bg-yellow-400 dark:bg-ye
                <svg
                  className="w-8 h-8 text-gray-900 dark:text-gray-900"
                  fill="none"
                  stroke="currentColor"
                  viewBox="0 0 24 24"
                >
                  <path
                    strokeLinecap="round"
                    strokeLinejoin="round"
                    strokeWidth={2.5}
                    d="M12 9v2m0 4h.01m-6.938 4h13.856c1.54 0 2.502-1.667 1.73
                  />
                </svg>
              </div>
              <div className="flex-1">
                <h2 className="text-2xl font-bold text-gray-900 dark:text-gray
                  Refund Approval Required
                </h2>
                <p className="text-sm text-gray-600 dark:text-gray-400">
                  Please review the details below carefully
                </p>
              </div>
            </div>

            {/* Refund details card */}
            <div className="space-y-3 bg-gray-50 dark:bg-gray-800 rounded-lg p
              <div className="flex justify-between items-center py-2 border-b
                <span className="text-sm font-medium text-gray-600 dark:text-g
                  Order ID
                </span>
                <span className="text-sm font-mono font-semibold text-gray-900
                  {refundRequest.order_id}
                </span>
              </div>
              <div className="flex justify-between items-center py-2 border-b
                <span className="text-sm font-medium text-gray-600 dark:text-g
                  Refund Amount
                </span>
                <span className="text-2xl font-bold text-gray-900 dark:text-gr
                  ${refundRequest.amount.toFixed(2)}
                </span>
              </div>
              <div className="pt-2">
                <span className="text-sm font-medium text-gray-600 dark:text-g
```

```
            Reason
          </span>
          <div className="text-sm text-gray-900 dark:text-gray-100 bg-wh
            {refundRequest.reason}
          </div>
        </div>
      </div>

      {/* Warning message */}
      <div className="flex items-start gap-3 mb-6 p-4 bg-yellow-50 dark:
        <svg
          className="w-5 h-5 text-yellow-600 dark:text-yellow-400 flex-s
          fill="currentColor"
          viewBox="0 0 20 20"
        >
          <path
            fillRule="evenodd"
            d="M18 10a8 8 0 11-16 0 8 8 0 0116 0zm-7-4a1 1 0 11-2 0 1 1
            clipRule="evenodd"
          />
        </svg>
        <p className="text-sm text-yellow-900 dark:text-yellow-100 font-
          This action cannot be undone. Approving will process the refun
          immediately.
        </p>
      </div>

      {/* Action buttons */}
      <div className="flex gap-4">
        <button
          onClick={() => handleRefundApproval(false)}
          className="flex-1 px-6 py-3.5 bg-gray-200 hover:bg-gray-300 da
        >
          <svg
            className="w-5 h-5"
            fill="none"
            stroke="currentColor"
            viewBox="0 0 24 24"
          >
            <path
              strokeLinecap="round"
              strokeLinejoin="round"
              strokeWidth={2.5}
              d="M6 18L18 6M6 6l12 12"
            />
          </svg>
          Cancel
```

```
              </button>
              <button
                onClick={() => handleRefundApproval(true)}
                className="flex-1 px-6 py-3.5 bg-green-600 hover:bg-green-700
              >
                <svg
                  className="w-5 h-5"
                  fill="none"
                  stroke="currentColor"
                  viewBox="0 0 24 24"
                >
                  <path
                    strokeLinecap="round"
                    strokeLinejoin="round"
                    strokeWidth={2.5}
                    d="M5 13l4 4L19 7"
                  />
                </svg>
                Approve Refund
              </button>
            </div>

            {/* ESC hint */}
            <p className="text-xs text-center text-gray-500 dark:text-gray-400
              Press{" "}
              <kbd className="px-2 py-1 bg-gray-100 dark:bg-gray-800 border bo
                ESC
              </kbd>{" "}
              to cancel
            </p>
          </div>
        </div>
      )}

      {/* Your chat interface */}
      <CopilotChat />
    </div>
  );
}
```

Why This Pattern Works:

1. **No Backend Tool Collision**: Backend doesn't have `process_refund`, so agent can't bypass approval
2. **Promise Blocks Agent**: Agent waits for Promise to resolve before continuing

3. **Professional UX**: Modal with proper styling, animations, and keyboard shortcuts
4. **Type-Safe**: TypeScript ensures correct parameters
5. **Accessible**: Keyboard navigation, ARIA labels, high contrast

User Experience:

User: "I want a refund for order ORD-12345"
Agent: "I can help with that. What's the amount and reason?"
User: "100, items arrived broken"
→ **Beautiful modal appears** with all details
→ User can approve (Enter) or cancel (ESC)
→ Agent receives decision and responds accordingly

Feature 3: Shared State

:::success Fully Implemented in Tutorial 30

Shared state works seamlessly with `useCopilotReadable`:

- ✔️ **User context** automatically available to agent
- ✔️ **Real-time sync** when state changes
- ✔️ **No manual passing** of data required

Try it:

```
cd tutorial_implementation/tutorial30
make dev
# Chat: "What's my account status?"
# Agent knows you're John Doe with Premium account! 👤
```

Implementation: `nextjs_frontend/app/page.tsx` (lines 18-26, 40-43)
:::

Sync application state with the agent automatically using `useCopilotReadable`:

```
"use client";

  // Application state (could come from auth, database, etc.)
  const [userData, setUserData] = useState({
    name: "John Doe",
    email: "john@example.com",
    accountType: "Premium",
    orders: ["ORD-12345", "ORD-67890"],
  });

  // Make state readable by agent - that's it!
  useCopilotReadable({
    description: "Current user's account information and order history",
    value: userData,
  });

  return (
    <CopilotKit runtimeUrl="http://localhost:8000/copilotkit">
      <CopilotChat />
      {/* Agent automatically knows user context without manual passing! */}
    </CopilotKit>
  );
}
```

How It Works:

1. **Define State**: Create React state with user/app data
2. **Make Readable**: Call `useCopilotReadable` with description and value
3. **Agent Accesses**: Agent automatically receives context in every request

Example Interaction:

```
User: "What's my account status?"

Agent Response: "Hi John! You have a Premium account with email
john@example.com. I see you have 2 orders: ORD-12345 and ORD-67890.
Would you like to check on any of them?"
```

The agent knows ALL this without you explicitly telling it! 🎯

Advanced: Multiple Readable States

```
// User profile
useCopilotReadable({
  description: "User profile information",
  value: userProfile,
});

// Shopping cart
useCopilotReadable({
  description: "Current shopping cart contents",
  value: cart,
});

// App preferences
useCopilotReadable({
  description: "User preferences and settings",
  value: preferences,
});

// Agent now has access to all three contexts automatically!
```

Real-Time Updates:

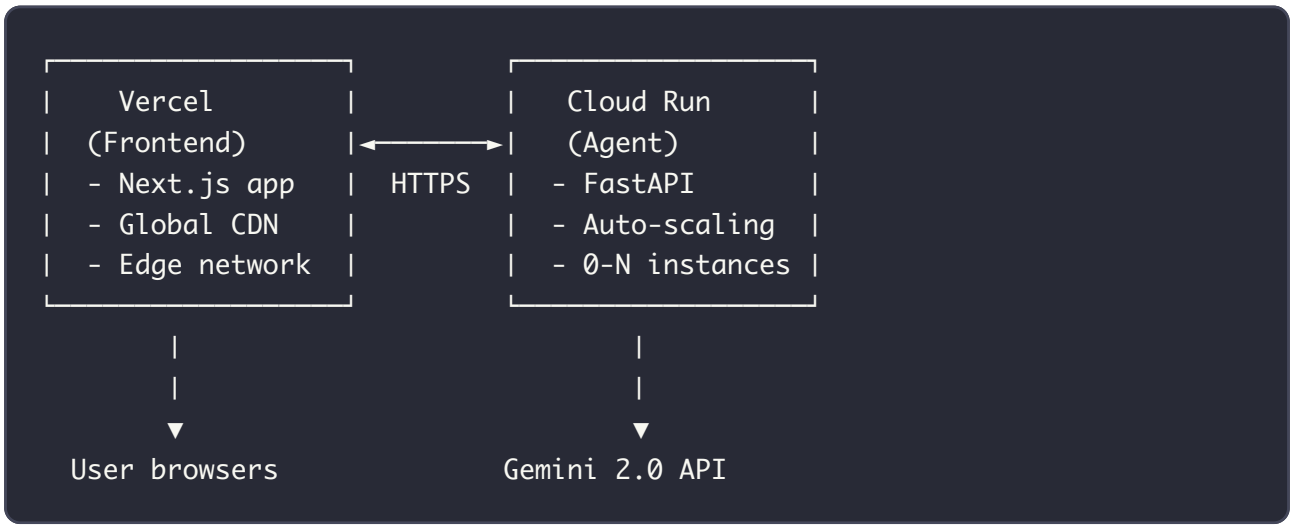
When state changes, agent automatically gets updated context:

```
// User adds item to cart
const addToCart = (item: Product) => {
  setCart([...cart, item]);
  // Agent immediately knows about new cart state!
};
```

This enables truly context-aware conversations without manual data passing! 🚀

Production Deployment

Architecture Overview



Step 1: Deploy Agent to Cloud Run

Create `agent/Dockerfile`:

```
FROM python:3.11-slim

WORKDIR /app

# Install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy agent code
COPY agent.py .
COPY .env .

# Expose port
EXPOSE 8000

# Run agent
CMD ["uvicorn", "agent:app", "--host", "0.0.0.0", "--port", "8000"]
```

Deploy to Cloud Run:

```
# Build and deploy
gcloud run deploy customer-support-agent \
  --source=./agent \
  --region=us-central1 \
  --allow-unauthenticated \
  --set-env-vars="GOOGLE_API_KEY=your_api_key"

# Output:
# Service URL: https://customer-support-agent-abc123.run.app
```

Step 2: Deploy Frontend to Vercel

Update `app/page.tsx` with production URL:

```
const AGENT_URL = process.env.NEXT_PUBLIC_AGENT_URL || "http://localhost:8000"

  return (
    <CopilotKit runtimeUrl={`${AGENT_URL}/copilotkit`}>
      <CopilotChat />
    </CopilotKit>
  );
}
```

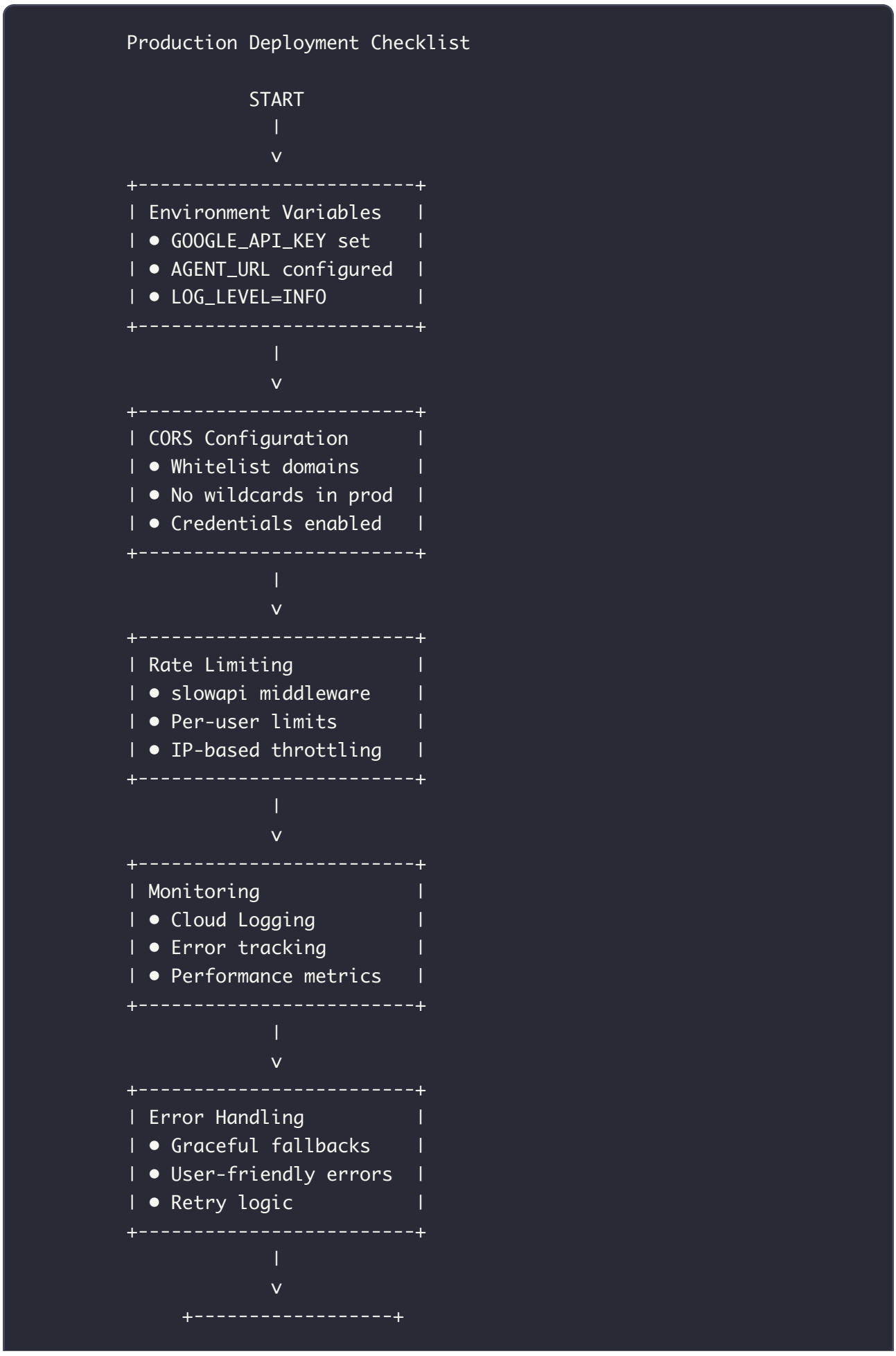
Deploy:

```
# Install Vercel CLI
npm i -g vercel

# Deploy
vercel

# Set environment variable
vercel env add NEXT_PUBLIC_AGENT_URL production
# Enter: https://customer-support-agent-abc123.run.app

# Deploy again with env
vercel --prod
```

Your app is live! 🚀

URL: `https://customer-support-bot.vercel.app`

Step 3: Production Best Practices


Production Deployment Checklist
START
Environment Variables
• GOOGLE_API_KEY set
• AGENT_URL configured
• LOG_LEVEL=INFO
CORS Configuration
• Whitelist domains
• No wildcards in prod
• Credentials enabled
Rate Limiting
• slowapi middleware
• Per-user limits
• IP-based throttling
Monitoring
• Cloud Logging
• Error tracking
• Performance metrics
Error Handling
• Graceful fallbacks
• User-friendly errors
• Retry logic

```
                | Production Ready |
                +------------------+
```

1. Environment Variables

```
# Vercel (Frontend)
NEXT_PUBLIC_AGENT_URL=https://agent.run.app

# Cloud Run (Agent)
GOOGLE_API_KEY=xxx
ENVIRONMENT=production
LOG_LEVEL=INFO
```

2. CORS Configuration

```
# agent/agent.py
from fastapi.middleware.cors import CORSMiddleware

app.add_middleware(
    CORSMiddleware,
    allow_origins=[
        "https://customer-support-bot.vercel.app",
        "https://*.vercel.app",  # Preview deployments
    ],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

3. Rate Limiting

```
from slowapi import Limiter
from slowapi.util import get_remote_address

limiter = Limiter(key_func=get_remote_address)

@app.post("/copilotkit")
@limiter.limit("100/hour")  # 100 requests per hour per IP
async def copilotkit_endpoint(...):
    ...
```

4. Monitoring

```
from opentelemetry import trace
from opentelemetry.exporter.cloud_trace import CloudTraceSpanExporter

# Set up Google Cloud Trace
tracer = trace.get_tracer(__name__)

@app.post("/copilotkit")
async def copilotkit_endpoint(...):
    with tracer.start_as_current_span("copilotkit_request"):
        # ... handle request
        pass
```

5. Error Handling

```
from fastapi import HTTPException, status

@app.exception_handler(Exception)
async def global_exception_handler(request, exc):
    logger.error(f"Unhandled error: {exc}", exc_info=True)
    return JSONResponse(
        status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
        content={"message": "Internal server error"}
    )
```

Troubleshooting

Common Issues

Issue 1: WebSocket Connection Failed

Symptoms:

- Chat doesn't load
- Console error: "WebSocket connection failed"

Solution:

```
// Check runtimeUrl is correct
<CopilotKit runtimeUrl="http://localhost:8000/copilotkit">  // ✔️ Correct
<CopilotKit runtimeUrl="http://localhost:8000">  // ❌ Missing /copilotkit
```

Issue 2: Agent Not Responding

Symptoms:

- Messages send but no response
- Loading spinner forever

Solution:

```
# Check agent is running
curl http://localhost:8000/health

# Check logs
# In agent terminal, look for errors

# Verify API key
echo $GOOGLE_API_KEY  # Should show your key
```

Issue 3: CORS Errors in Production

Symptoms:

- Works locally, fails in production
- Browser console: "CORS policy blocked"

Solution:

```
# agent/agent.py - Add your production domain
app.add_middleware(
    CORSMiddleware,
    allow_origins=[
        "https://your-app.vercel.app",  # Add this!
        "http://localhost:3000",  # Keep for local dev
    ],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Issue 4: Tools Not Working

Symptoms:

- Agent doesn't call functions
- Responses are generic

Solution:

```
# Verify tool registration
app = create_copilotkit_runtime(
    agent=agent,
    tools={
        "search_knowledge_base": search_knowledge_base,  # ✔️ Must match Funct
        "searchKnowledgeBase": search_knowledge_base,    # ❌ Wrong name
    }
)

# Check function signature
def search_knowledge_base(query: str) -> Dict[str, str]:  # ✔️ Return type hin
def search_knowledge_base(query):  # ❌ Missing type hint
```

Issue 5: Slow Responses

Symptoms:

- Agent takes 10+ seconds to respond

- Users complain about lag

Solution:

```
from google.adk.agents import Agent

# Use fast model and optimize instructions
agent = Agent(
    model="gemini-2.0-flash-exp",  # ✔️ Fast model
    # model="gemini-2.0-pro-exp",  # ❌ Slower, use only when needed
    name="customer_support_agent",
    instruction="Be concise. Answer in 2-3 sentences max."  # ✔️ Shorter is be
)

# ❌ Avoid: Very long instructions slow down responses
# instruction="You are an extremely detailed agent..." (5 paragraphs)

# Use caching for knowledge base
from functools import lru_cache

@lru_cache(maxsize=128)
def search_knowledge_base(query: str):
    # Cached for repeated queries
    ...
```

Next Steps

You've Mastered Next.js + ADK! 🎉

You now know how to:

✔️ Build production-ready Next.js 15 + ADK apps
✔️ Integrate CopilotKit/AG-UI Protocol
✔️ Create custom tools and agents
✔️ Add generative UI and HITL
✔️ Deploy to Vercel + Cloud Run
✔️ Monitor and troubleshoot

Continue Learning

Tutorial 31: React Vite + ADK Integration
Build a lightweight alternative with React Vite (same patterns, faster dev)

Tutorial 32: Streamlit + ADK Integration
Build data apps with Python-only stack (no frontend code!)

Tutorial 35: AG-UI Deep Dive
Master advanced features: multi-agent UI, custom protocols, enterprise patterns

Additional Resources

- CopilotKit Documentation (https://docs.copilotkit.ai/adk)
- Next.js 15 Documentation (https://nextjs.org/docs)
- ADK Documentation (https://google.github.io/adk-docs/)
- Example: gemini-fullstack (https://github.com/google/adk-samples/tree/main/gemini-fullstack)

🎉 **Tutorial 30 Complete!**

Next: Tutorial 31: React Vite + ADK Integration (./31_react_vite_adk_integration.md)

Questions or feedback? Open an issue on the ADK Training Repository (https://github.com/google/adk-training).

Generated on 2025-10-21 09:03:01 from 30_nextjs_adk_integration.md

Source: Google ADK Training Hub