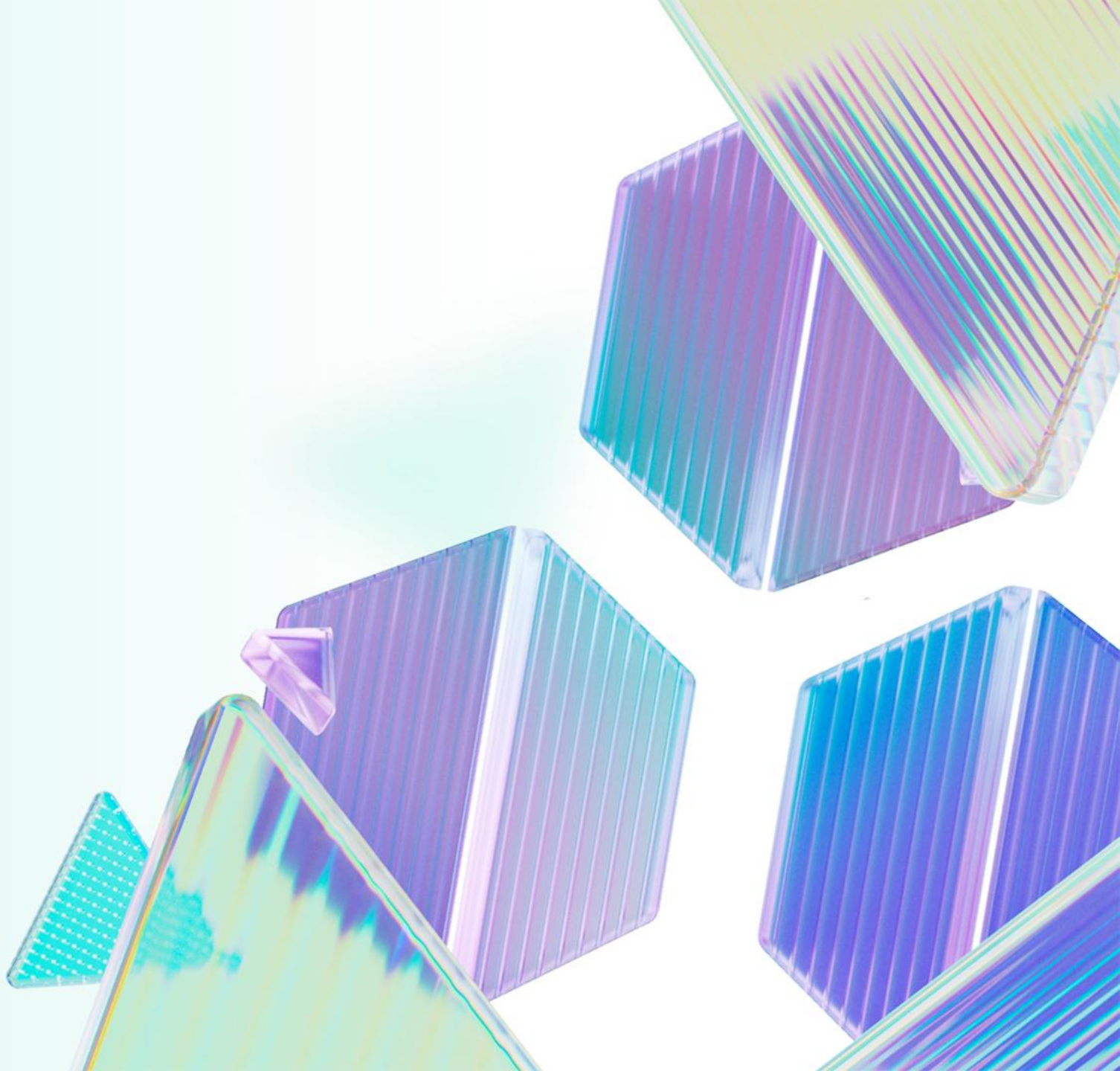
Microsoft

MCAPS
Tech Connect



Microsoft

THR505 - Integrating and branding Copilot Studio with web chat (text and audio)

Speakers

Jose Del Castillo
Architect

Godwin Tenzing
Architect

Agenda

- Citizen Advice Web Chat Agent Demo
- Web Chat Integration Approaches using Direct Line
- M365 Agents SDK Framework Architecture Overview
- Web Chat Integration Approaches
- Web Chat Branding & Styling
- Explore Adaptive Cards for Intent Recognition
- Discuss ambiguous user inputs & intent recognition
- Interactive Voice Response Options & Demo
- Telephony IVR with LiveHub AudioCodes Integration & Demo

Citizen Advice CoPilot Web Chat Demo

What is Direct Line?

Direct Line is a **communication channel** provided by the Microsoft's Bot Framework.

By leveraging Direct Line API, developers can embed bots into various platforms (websites, mobile apps, or even custom apps, while maintaining full control over the user experience.

Secure & Scalable Communication

- **Authentication & Token Generation:** Direct Line API requires a token to establish a secure connection
- The backend server to generate the token using key vault and provide it to the web chat client.
- Direct Line API supports messages and event-based triggers on web chat client.

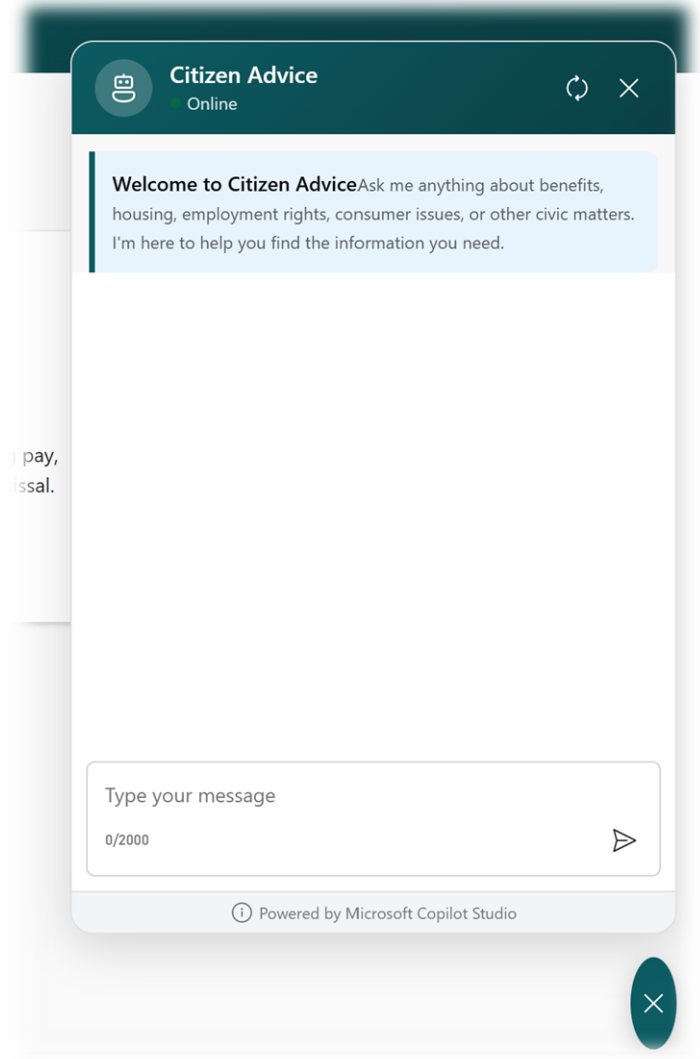
Connecting Web Chat to Direct Line

- The Web Chat Client abstracts Direct Line API Communication
- The Web Chat client uses the token to create a secure Direct Line session.
- Messages are exchanged between the user and the bot via Direct Line API.
- The bot processes user messages with text, adaptive cards or multi-media content.
- Direct Line pulls bot responses.

Copilot Studio Bot Integration in a Web Chat

Web Chat Integration Approaches

1. Web Chat via CDN and Direct Line API (Simplest Approach)
2. Embedding Web Chat with simple ReactWebChat Control
3. Embedding ReactWebChat With Middleware and Custom Styling
4. Embedding Web Chat with React RenderWebChat Control for Advanced Customizations



Anonymous Copilot WebChat Demo using Bot Framework (Legacy)

Comparing Web Chat Integration Approaches

CS Web Chat Integration Options (Legacy – Anonymous DL + Token Endpoint)

- **CDN:** Use <script src="webchat.js"> - No build step required, perfect for quick prototypes
- **Simple ReactWebChat:** Import ReactWebChat from botframework-webchat - Best for basic React apps
- **Advanced ReactWebChat:** Use Composer + BasicWebChat for full customization with middleware
- **RenderWebChat:** Call renderWebChat() function directly for non-React or custom DOM scenarios
- All approaches support styleOptions for theming and directLine for bot connection

Feature	Web Chat using CDN Script tag embed	Simple ReactWebChat npm package	★ RECOMMENDED Advanced ReactWebChat Full component control	RenderWebChat Function Vanilla JS render
Ease of Use	✓ Easiest	✓ Easy	✗ Complex	✗ Complex
Customization	✓	~	✓ Medium	✓ Full
Middleware Support	✓	~	✓	✓
DOM Control	✓	✗	✗	✓
Performance	✓ Fastest	✓ Lightweight	✗ Heavy	✗ Heaviest
Styling Options	✓ Advanced with Style Options	✓	✓ Advanced with Style Options	✓ Full
Security	✓	✓	✓	✓ Full Control
Build Tools Required	✓ None	~ npm/webpack	~ npm/webpack	✓ Optional
TypeScript Support	✗	✓	✓	✓
Component Composition	✗	~	✓	✓
Ideal Use Case	Fully featured Web Chat with CDN simplicity	Simple Bot integration	Advanced Web Chat Bot Customization	Enterprise-level customization & branding

✓ Fully Supported ~ Partial / Depends ✗ Not Available / Manual ★ Recommended for most scenarios

M365 Agents SDK & Bot Framework Copilot Studio Integration Architecture overview

Bot Framework v4 → M365 Agents SDK

Azure Bot Framework SDK v4 (Legacy)

- Released 2018, mature but aging
- Azure Bot Service dependent
- Limited channel extensibility
- Separate SDK per language
- Bot Framework Emulator for testing
- Complex authentication setup
- Maintenance mode - no new features

Microsoft 365 Agents SDK

New

- Released 2025, actively developed
- Multi-channel: Teams, M365 Copilot, Web
- AI-agnostic orchestration
- Unified samples across .NET, JS, Python
- Teams Test Tool & Agents Toolkit
- Simplified Entra ID integration
- Copilot Studio native support

Why Migrate to M365 Agents SDK?

Microsoft 365 Copilot Integration
Build agents that extend M365 Copilot

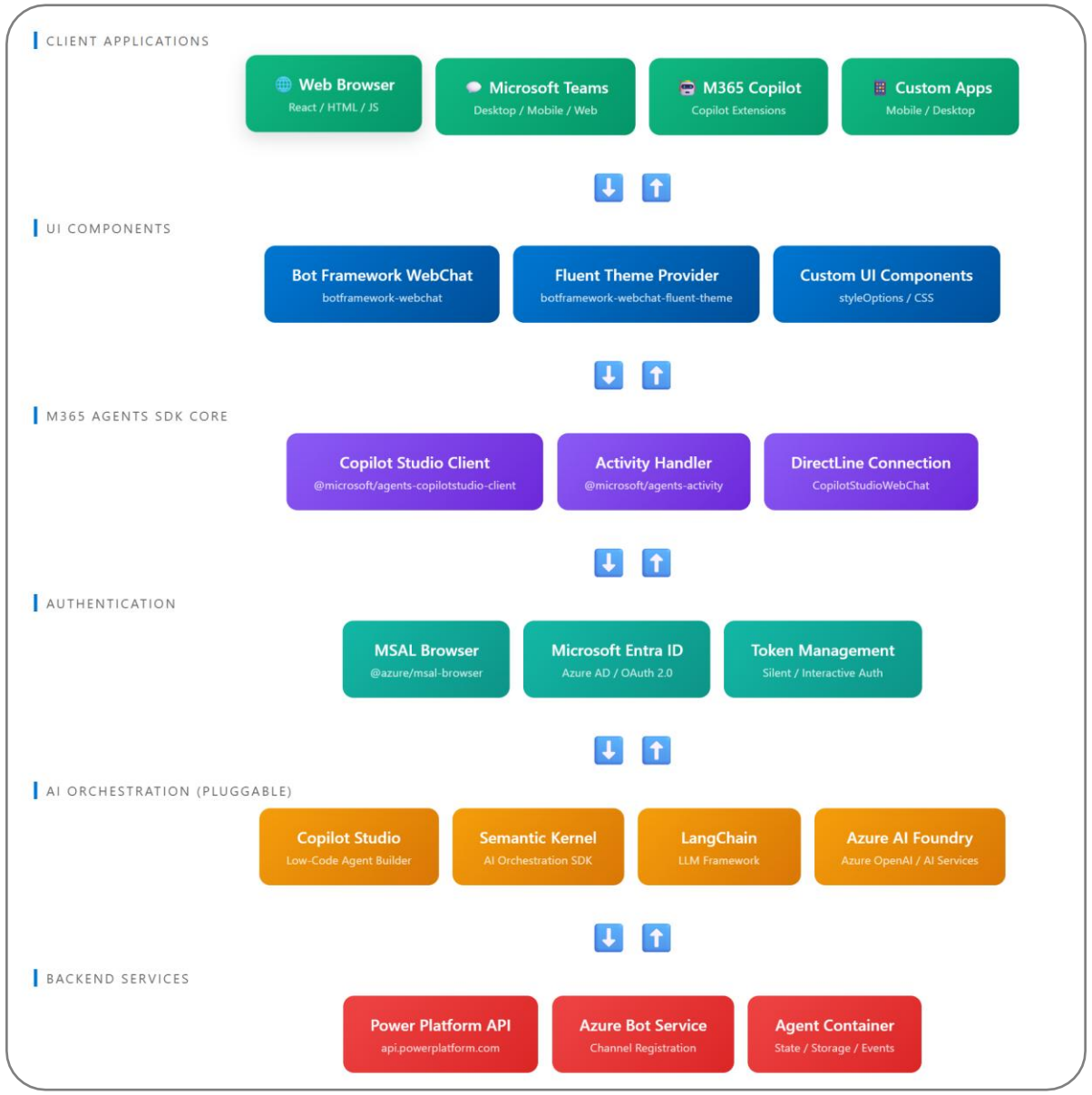
Future-Proof Architecture
Designed for AI-first scenarios

Simplified Development
Less boilerplate, more productivity

Copilot Studio Synergy
Low-code + Pro-code in one ecosystem

Modern Authentication
Native Entra ID support with MSAL

Streamlined Deployment
Azure Bot Service or standalone hosting



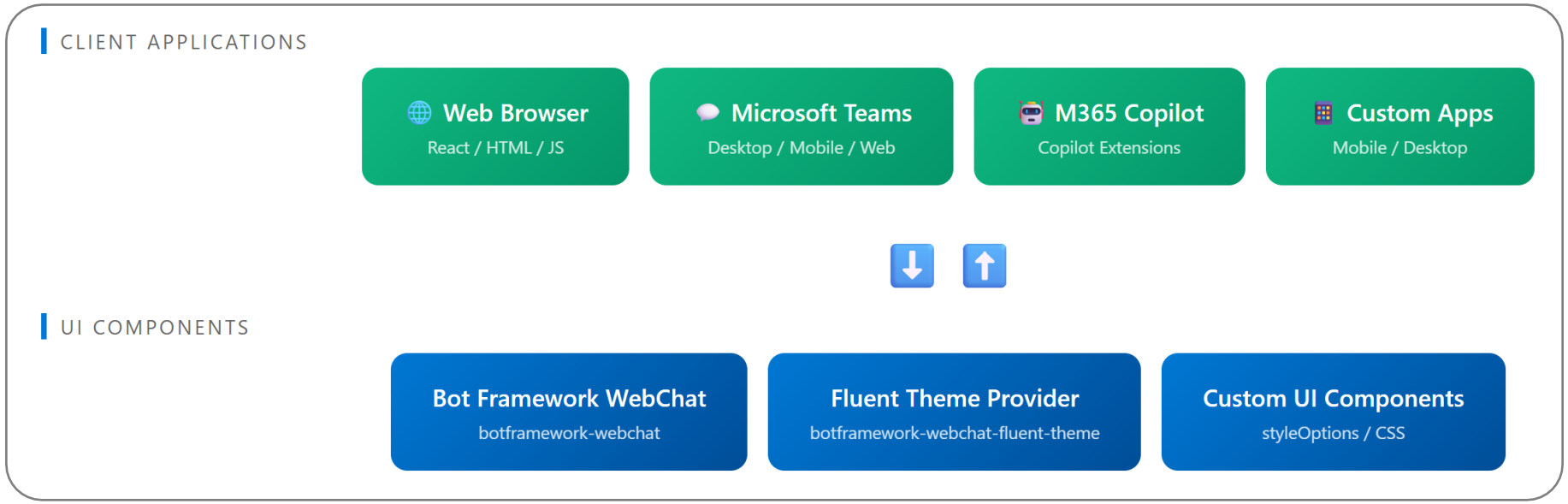
M365 Agents SDK Client & UI Components

Client Applications

- **Multi-channel reach:** Single agent, multiple surfaces (Web, Teams, Copilot, Mobile)
- **No client SDK required:** Standard web technologies (React, HTML, JavaScript)
- **Consistent experience:** Same agent logic across all channels

UI Components

- **Bot Framework WebChat:** Production-ready chat component with streaming support
- **Fluent Theme Provider:** Instant Microsoft 365 look and feel
- **styleOptions:** 50+ customization points for brand alignment (colors, avatars, bubbles)



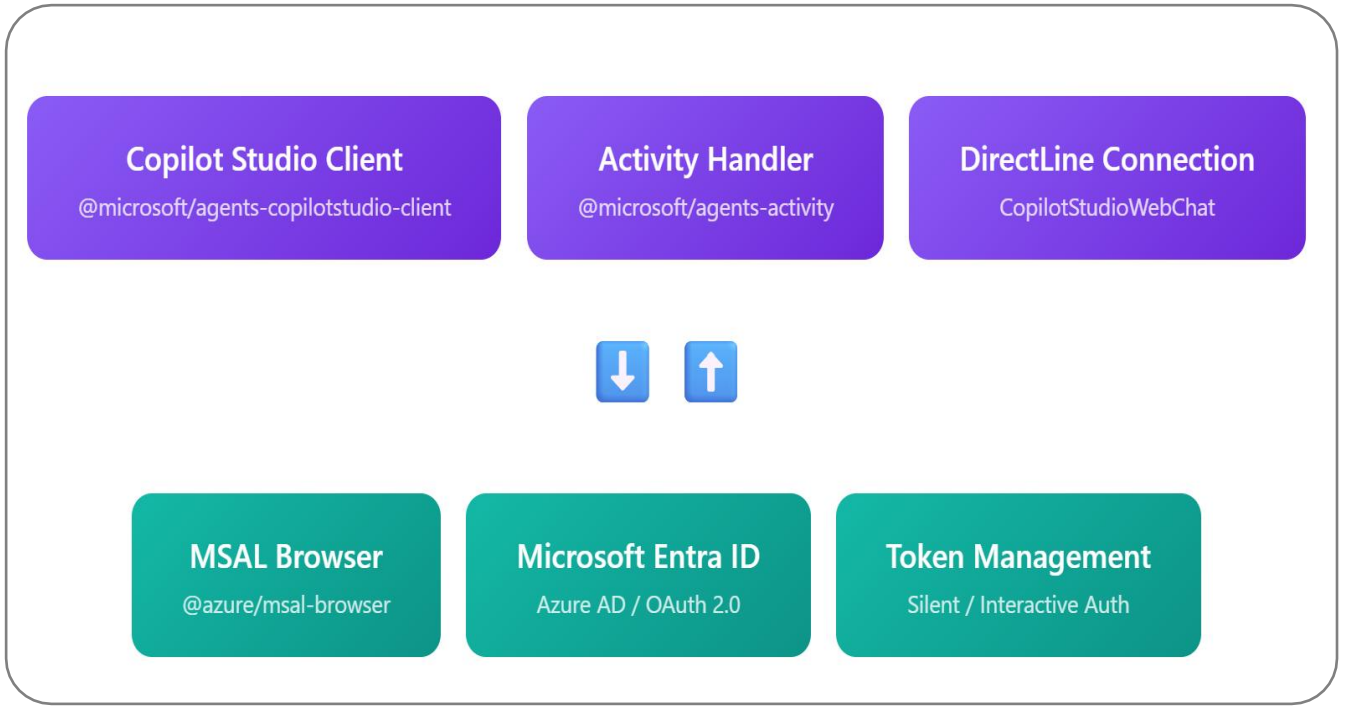
M365 Agents SDK & Auth Components

M365 Agents SDK Core

- **CopilotStudioClient:** Single API for all Copilot Studio interactions
- **Activity Handler:** Standardized message format (text, cards, events, attachments)
- **DirectLine Connection:** Real-time bidirectional streaming via WebSocket

Authentication

- **MSAL Integration:** Microsoft's official authentication library
- **Microsoft Entra ID:** Enterprise-grade identity (Azure AD)
- **Smart Token Flow:** Silent refresh, when possible, interactive only when needed



M365 Agents SDK Orchestration & Backend Services

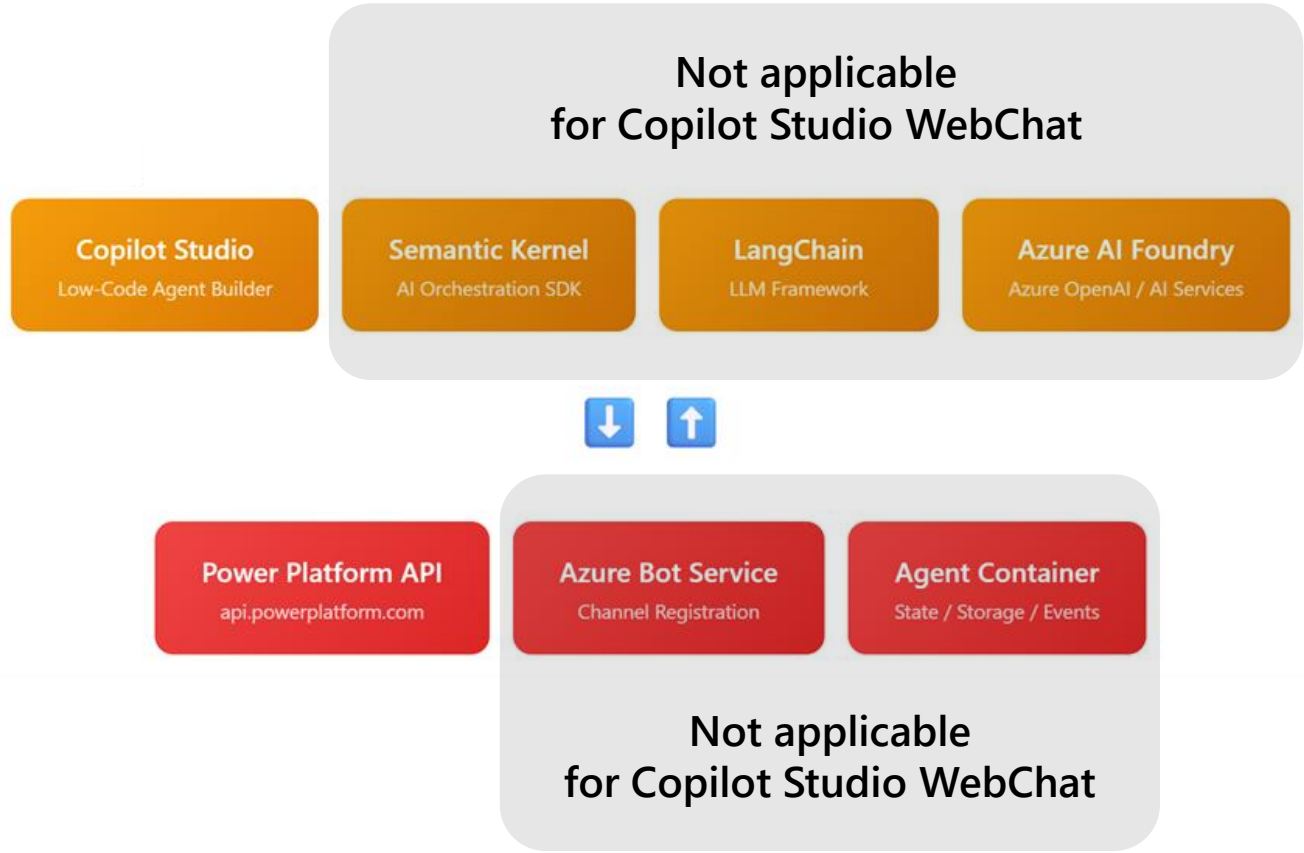
AI Orchestration

(Not applicable for CoPilot Studio)

- **AI Agnostic:** Bring your own AI service or model
- **Copilot Studio:** Low-code agent building with enterprise governance
- **Semantic Kernel / LangChain:** Pro-code orchestration frameworks
- **Azure AI Foundry:** Direct access to Azure OpenAI and AI services

Backend Services

- **Power Platform API:** Managed infrastructure, no servers to maintain
- **Azure Bot Service:** Optional channel registration for Teams/Outlook
- **Agent Container:** Built-in state management, storage, and event handling



M365 Agents SDK supported channels

Microsoft Teams

Native integration via Teams Toolkit

Microsoft 365 Copilot

Extend Copilot with custom agents

Web Applications

Embed via WebChat component

Copilot Studio

Bidirectional: consume agents & expose as skills

Custom Channels

DirectLine protocol for any client

Mobile Apps

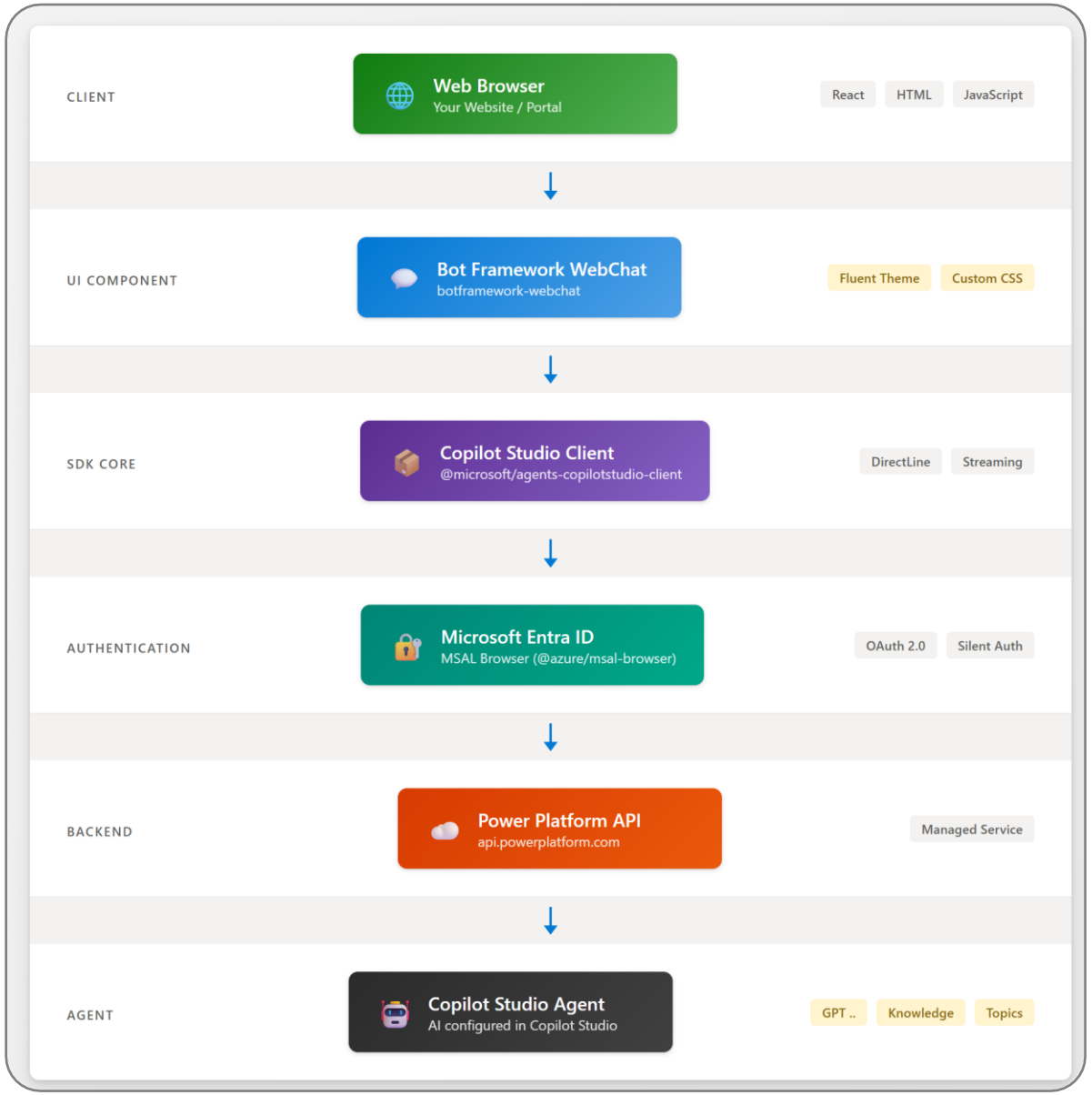
React Native compatible WebChat

Copilot Studio WebChat Architecture

- **Browser → WebChat → SDK → Auth → API → Agent**
- WebChat UI with Fluent styling in your website
- Copilot Studio Client handles DirectLine streaming
- MSAL authenticates via Microsoft Entra ID
- Power Platform API routes to your Copilot Studio agent
- AI/Knowledge configured in Copilot Studio

REQUIRED NPM PACKAGES

```
{
  "@microsoft/agents-copilotstudio-client": "^1.1.0",
  "botframework-webchat": "^4.18.0",
  "botframework-webchat-fluent-theme": "^0.3.0",
  "@azure/msal-browser": "^4.13.1"
}
```



Connecting WebChat to Copilot Studio (M365 Agents SDK)

CopilotStudioClient abstracts DirectLine, eliminating backend complexity

Browser-Based Authentication

- Browser-only authentication via MSAL - no server-side token generation
- MSAL acquires user token via Entra ID
- Silent refresh or interactive popup
- No backend server or Key Vault requirement

SDK Connection Layer

- CopilotStudioClient creates DirectLine connection
- CreateConnection() bridges WebChat to Copilot Studio
- WebSocket support for real-time messages

```
// Browser-based connection - No backend required!
const token = await acquireToken(settings)  // MSAL → Entra ID
const client = new CopilotStudioClient(settings, token)
const connection = CopilotStudioWebChat.createConnection(client)

// Render WebChat
<Composer directLine={connection}>
  <BasicWebChat />
</Composer>
```

Handling User Interactions

- Agent Responds with text, Cards, multimedia
- Streaming responses via sendActivityStreaming()
- Typing indicators, suggested actions, & rich attachments.

What changed

	Old Bot Framework SDK v4	New M365 Agents SDK	Same Package?
UI Components	*botframework-webchat*	*botframework-webchat*	✅ YES
Styling	styleOptions	styleOptions OR FluentThemeProvider	✅ YES
Middleware	Composer Middleware prop	Composer middleware prop	✅ YES
Connection	CreateDirectLine({Secret})	**CopilotStudioWebChat.createConnection(client)**	❌ CHANGED
Auth	Backend token server	**MSAL Browser Auth**	❌ CHANGED

```
// ❌ DEPRECATED connection method
const directLine = window.WebChat.createDirectLine({
  secret: 'YOUR_DIRECTLINE_SECRET'  // From Azure Bot Service
})

window.WebChat.renderWebChat({ directLine }, document.getElementById('webchat'))
```

```
// ✅ CURRENT connection method
const token = await acquireToken(settings)  // MSAL
const client = new CopilotStudioClient(settings, token)
const directLine = CopilotStudioWebChat.createConnection(client)

window.WebChat.renderWebChat({ directLine }, document.getElementById('webchat'))
```

Migration from Bot Framework v4

1. **Assess Current Bot**
   Document channels, features, integrations
2. **Review Agents SDK Samples**
   Find equivalent patterns
3. **Update Authentication**
   Migrate to MSAL-based approach
4. **Refactor Channel Logic**
   Use new adapter patterns
5. **Update WebChat Integration**
   Switch to Copilot Studio client
6. **Test Thoroughly**
   Use Teams Test Tool and WebChat testing
7. **Plan Deployment**
   Azure Bot Service or standalone hosting

M365 Agents SDK Copilot Studio Agent Webchat integration

Godwin Tenzing & Jose Del Castillo

CDN + renderWebChat (Vanilla JS)

Embed WebChat without build tools. Load via <script> tags, authenticate with MSAL, customize with styleOptions. Ideal for CMS, WordPress, SharePoint.

When to Use:

- React apps with simple needs & Quick Integration
- Default styling is acceptable
- No need for FluentThemeProvider

Pros:

- Simplest React Option and Single Import
- All Props in one place

Cons:

- Can't insert components between Composer and UI
- Can’t use FluentThemeProvider
- Less Flexible

```
// No build tools required - just include via <script> tags
const connection = await CopilotStudioWebChat.createConnection({
  botId: 'YOUR_BOT_ID',
  environmentId: 'YOUR_ENV_ID',
  getToken: () => msalInstance.acquireTokenSilent(tokenRequest)
});

window.WebChat.renderWebChat(
  {
    directLine: connection,
    styleOptions: {
      bubbleBackground: '#E0E0E0',
      bubbleFromUserBackground: '#0078D4',
      botAvatarImage: '/bot-avatar.png'
    }
  },
  document.getElementById('webchat')
);
```

Simple ReactWebChat

Simplest React integration. Single component, pass directLine connection and styleOptions. Quick setup for basic React apps.

When to Use:

- React apps with simple needs
- Default styling is acceptable
- No need for FluentThemeProvider
- Quick integration

Pros:

- Simplest React Option
- Simple Import

Cons:

- Can't insert components between Composer and UI
- Can’t use FluentThemeProvider
- Less Flexible

```jsx
import ReactWebChat from 'botframework-webchat';
import { CopilotStudioClient } from '@microsoft/agents-copilotstudio-client';

function ChatBot() {
  const [connection, setConnection] = useState(null);

  useEffect(() => {
    const client = new CopilotStudioClient({
      botId: 'YOUR_BOT_ID',
      environmentId: 'YOUR_ENV_ID',
      getToken: () => msalInstance.acquireTokenSilent(tokenRequest)
    });
    setConnection(client.createDirectLine());
  }, []);

  return connection ? (
    <ReactWebChat
      directLine={connection}
      styleOptions={{ bubbleFromUserBackground: '#0078D4' }}
    />
  ) : <p>Loading...</p>;
}
```

Flexible React (Compose + BasicWebChat)

Production-ready with Microsoft styling. Separates context provider (Composer) from UI (BasicWebChat), enabling FluentThemeProvider for Teams/M365 look

When to Use:

- Need Microsoft styling (FluentThemeProvider)
- Want to add custom components alongside chat
- Need access to hooks outside BasicWebChat
- Building complex integrations

Pros:

- Full Flexibility
- Support FluentThemeProvider

Cons:

- More Verbose
- Requires understanding component structure

```
import { Composer, BasicWebChat } from 'botframework-webchat';
import { FluentThemeProvider } from 'botframework-webchat-fluent-theme';
import { CopilotStudioClient } from '@microsoft/agents-copilotstudio-client';

function ChatBot() {
  const [connection, setConnection] = useState(null);

  useEffect(() => {
    const client = new CopilotStudioClient({ /* config */ });
    setConnection(client.createDirectLine());
  }, []);

  return connection ? (
    <Composer directLine={connection}>
      <FluentThemeProvider>
        <BasicWebChat />
      </FluentThemeProvider>
    </Composer>
  ) : <p>Loading...</p>;
}
```

ReactWebChat with Middleware

Intercept and modify messages. Use createStore with middleware to log, filter, or transform incoming/outgoing activities before they reach the UI.

```
import ReactWebChat, { createStore } from 'botframework-webchat';

function ChatWithCustomCards() {
  const [connection, setConnection] = useState(null);

  // Custom attachment middleware for rendering cards
  const attachmentMiddleware = () => next => ({ attachment, activity }) => {
    if (attachment.contentType === 'application/vnd.mycompany.product-card') {
      return <ProductCard data={attachment.content} />;
    }
    return next({ attachment, activity });
  };

  return connection ? (
    <ReactWebChat
      directLine={connection}
      attachmentMiddleware={attachmentMiddleware}
    />
  ) : <p>Loading...</p>;
}

// Custom component for your product cards
const ProductCard = ({ data }) => (
  <div className="product-card">
    <img src={data.image} alt={data.name} />
    <h3>{data.name}</h3>
    <p>${data.price}</p>
    <button onClick={() => addToCart(data)}>Add to Cart</button>
  </div>
);
```

Render custom cards. Intercept specific contentType attachments and return your own React components (e.g., ProductCard, BookingCard) instead of default rendering.


```
import ReactWebChat, { createStore } from 'botframework-webchat';

function ChatWithCustomCards() {
  const [connection, setConnection] = useState(null);

  // Custom attachment middleware for rendering cards
  const attachmentMiddleware = () => next => ({ attachment, activity }) => {
    if (attachment.contentType === 'application/vnd.mycompany.product-card') {
      return <ProductCard data={attachment.content} />;
    }
    return next({ attachment, activity });
  };

  return connection ? (
    <ReactWebChat
      directLine={connection}
      attachmentMiddleware={attachmentMiddleware}
    />
  ) : <p>Loading...</p>;
}

// Custom component for your product cards
const ProductCard = ({ data }) => (
  <div className="product-card">
    <img src={data.image} alt={data.name} />
    <h3>{data.name}</h3>
    <p>${data.price}</p>
    <button onClick={() => addToCart(data)}>Add to Cart</button>
  </div>
);
```

Individual Components (Full Custom Layout)

Build fully custom chat layouts. Use BasicTranscript and BasicSendBox separately. Insert custom headers, toolbars, or quick-reply buttons between components.

When to Use:

- Building a completely custom chat UI
- Only need specific pieces (transcript, send box)
- Replacing default components entirely
- Available Components

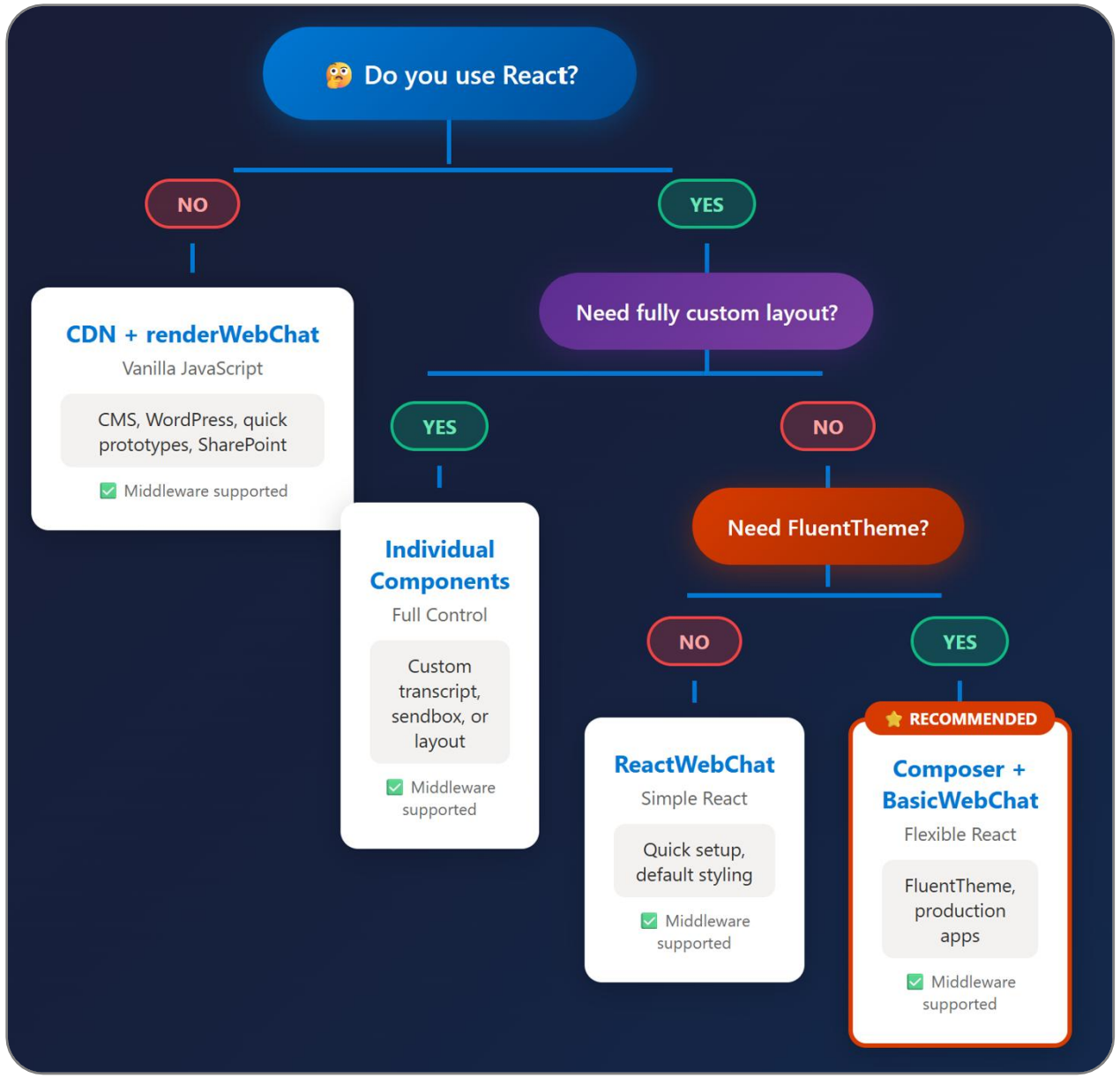
```jsx
import { Composer, BasicTranscript, BasicSendBox } from 'botframework-webchat';
import { CopilotStudioClient } from '@microsoft/agents-copilotstudio-client';

function CustomChatLayout() {
  const [connection, setConnection] = useState(null);

  useEffect(() => {
    const client = new CopilotStudioClient({ /* config */ });
    setConnection(client.createDirectLine());
  }, []);

  return connection ? (
    <Composer directLine={connection}>
      <div className="my-custom-chat">
        <header>AI Assistant</header>
        <BasicTranscript className="my-transcript" />
        <div className="custom-toolbar">
          <button>Quick Reply 1</button>
          <button>Quick Reply 2</button>
        </div>
        <BasicSendBox className="my-sendbox" />
      </div>
    </Composer>
  ) : <p>Loading...</p>;
}
```

Component	What It Renders
BasicTranscript	Message history
BasicSendBox	Text input + send button
BasicSendBoxToolbar	Toolbar above send box
BasicToaster	Notifications/toasts
BasicConnectivityStatus	Connection status banner
SuggestedActions	Quick reply buttons
MicrophoneButton	Speech input button
UploadButton	File attachment button
SendButton	Send message button
Avatar	Bot/user avatar
Bubble	Message bubble wrapper
Timestamp	Message timestamp

When to use what?

WebChat Control Guide:

- Use CDN for non-React sites.
- For React apps
  - choose ReactWebChat for simplicity
  - Composer+BasicWebChat for FluentTheme/production
  - Individual Components for custom layouts.
- Middleware works with all.



Comparing Web Chat Integration Approaches

Copilot Studio Web Chat Integration Options — M365 Agents SDK

CDN + VanillaJS

Zero build setup. Include scripts via unpkg.com. Best for quick POCs or embedding in CMS platforms. Limited styling options.

React + Fluent Theme

Production-ready with Microsoft's Fluent UI styling. Use FluentThemeProvider wrapper. Matches Teams/M365 look and feel automatically.

React + Custom Styling

Full control via styleOptions prop (50+ options). Remove FluentThemeProvider and apply your brand colors, avatars, fonts, and bubble styles.

Feature	CDN + Vanilla JS copilotstudio-webclient	★ RECOMMENDED React + Fluent Theme copilotstudio-webchat-react	React + Custom Styling styleOptions + CSS
Ease of Use	✓	✓	✗
Build Tools Required	✓ None	~ esbuild	~ esbuild/webpack
Customization	~ styleOptions	~ Fluent Theme	✓
Brand Styling	~ styleOptions	~ Limited	✓
Microsoft UX	✗	✓	✗ Custom
TypeScript Support	✗	✓	✓
Component Composition	✗	✓	✓
Middleware Support	✓	✓	✓
Streaming Support	✓	✓	✓
Performance	✓ Smallest	✓	✓
Security (MSAL)	✓	✓	✓
Ideal Use Case	Quick prototypes, simple demos	Production apps, Copilot-style UX	Branded experiences, custom design systems
SDK Sample	`samples/nodejs/copilotstudio-webclient`	`samples/nodejs/copilotstudio-webchat-react`	`Same as React + remove FluentThemeProvider`

✓ Fully Supported ~ Partial / Depends ✗ Not Available / Manual ★ Recommended for most scenarios

Authentication

Godwin Tenzing & Jose Del Castillo

Authentication prerequisites

Step 1

Create App Registration in Azure Portal

- Single-page application (SPA) type
- Redirect URI: http://localhost for development

Step 2

Configure API Permissions

- Add Power Platform API permission
- Grant CopilotStudio.Copilots.Invoke delegated permission

Step 3

Configure MSAL in Your App

- Use @azure/msal-browser package
- Scope: https://api.powerplatform.com/.default

Token Acquisition with MSAL

Browser-based MSAL authentication for Copilot Studio. Tries silent token first, falls back to popup. Uses Power Platform API scope —no backend token service required.

Usage of Auth within CopilotStudioClient

```typescript
import { CopilotStudioClient } from '@microsoft/agents-copilotstudio-client';
import { acquireToken } from './acquireToken';

const settings = {
  appClientId: 'YOUR_ENTRA_APP_CLIENT_ID',
  tenantId: 'YOUR_TENANT_ID',
  botId: 'YOUR_COPILOT_STUDIO_BOT_ID',
  environmentId: 'YOUR_POWER_PLATFORM_ENV_ID',
};

const client = new CopilotStudioClient({
  ...settings,
  getToken: () => acquireToken(settings),
});

const directLine = client.createDirectLine();
```

Setup & Token Logic

```typescript
import { PublicClientApplication, InteractionRequiredAuthError }
  from '@azure/msal-browser';
import { ConnectionSettings } from '@microsoft/agents-copilotstudio-client';

export async function acquireToken(settings: ConnectionSettings) {

  // 1. Create MSAL instance with Entra ID configuration
  const msalInstance = new PublicClientApplication({
    auth: {
      clientId: settings.appClientId,
      authority: `https://login.microsoftonline.com/${settings.tenantId}`,
    },
  });

  await msalInstance.initialize();

  // 2. Define token request with Power Platform scope
  const loginRequest = {
    scopes: ['https://api.powerplatform.com/.default'],
    redirectUri: window.location.origin,
  };

  // 3. Try silent token acquisition first (cached session)
  try {
    const accounts = await msalInstance.getAllAccounts();
    if (accounts.length > 0) {
      const response = await msalInstance.acquireTokenSilent({
        ...loginRequest,
        account: accounts[0],
      });
      return response.accessToken;
    }
  } catch (e) {
    if (!(e instanceof InteractionRequiredAuthError)) {
      throw e;
    }
  }

  // 4. Fallback to interactive popup login
  const response = await msalInstance.loginPopup(loginRequest);
  return response.accessToken;
}
```

WebChat branding & customization

Godwin Tenzing & Jose Del Castillo

Customization options

FluentTheme Provider

Microsoft Fluent UI styling out-of-the-box

styleOptions Prop

Fine-grained control over WebChat appearance

- Colors: accent, background, text colors
- Typography: font family, sizes
- Avatars: bot and user avatars
- Bubbles: message bubble styling

Custom CSS Prop

Override default styles with your own CSS

Custom Components

Replace WebChat components entirely

Fluent UI Theme System

The Fluent UI Theme System brings a native Microsoft Copilot-style user experience to Web Chat. It's a complete design system overhaul using Microsoft's Fluent UI library.

Key Features

- **Copilot Variant:** Built-in styling that matches Microsoft Copilot
- **Icon Customization:** CSS variable-based icon system (no JS overrides needed)
- **Activity Decorators:** Pluggable visual enhancements (animated borders, etc.)
- **Code Block Highlighting:** Syntax highlighting with copy buttons using Shiki
- **Starter Prompts:** Pre-chat suggested prompts for onboarding
- Chain-of-Thought Grouping:Collapsible reasoning flows for AI responses
- **Shadow DOM Support:** Style encapsulation for Web Components

```
// Package all customizations in a reusable theme
import { MyCustomThemeProvider } from 'my-custom-webchat-theme';

<MyCustomThemeProvider>
  <ReactWebChat />
</MyCustomThemeProvider>
```

WebChat styleOptions Customizations

StyleOptions in the Index.html or Index.js

```
<ReactWebChat
  directLine={connection}
  styleOptions={styleOptions}
/>
```

```
const styleOptions = {
  // Accent colors
  accent: '#0078D4',                          // Microsoft Blue
  backgroundColor: '#F5F5F5',

  // Typography
  primaryFont: "'Segoe UI', sans-serif",

  // Bot avatar
  botAvatarBackgroundColor: '#0078D4',
  botAvatarInitials: 'AI',

  // User avatar
  userAvatarBackgroundColor: '#107C10',
  userAvatarInitials: 'ME',

  // Message bubbles
  bubbleBackground: '#FFFFFF',
  bubbleFromUserBackground: '#E1DFDD',
  bubbleBorderRadius: 8,

  // Suggested actions
  suggestedActionBackground: '#0078D4',
  suggestedActionTextColor: '#FFFFFF',
};
```

Minimizable Web Chat Widget

- React-based approach using create-react-app with two main components
- A floating chat widget pattern where Web Chat is not the main app but an accessory. Users can minimize/maximize the chat via a button (typically positioned at the bottom-right corner of the page).

Key Features includes:

- State preservation: Conversation persists when minimized
- Toggle visibility: Show/hide without destroying the component
- Side switching: Can move between left/right of screen
- Customizable appearance: Full control over widget styling

MinimizableWebChat.tsx (Shows Chat Icon & Hides Chat Container)

```tsx
import ReactWebChat, { createStore } from 'botframework-webchat';
import { CopilotStudioClient, CopilotStudioWebChat } from '@microsoft/agents-copilotstudio-client';
import { acquireToken } from './acquireToken';

const WebChat = ({ store, styleOptions }) => {
  const [connection, setConnection] = useState(null);

  useEffect(() => {
    (async () => {
      const token = await acquireToken(settings);
      const client = new CopilotStudioClient(settings, token);
      setConnection(CopilotStudioWebChat.createConnection(client));
    })();
  }, []);

  return connection ? (
    <ReactWebChat directLine={connection} store={store} styleOptions={styleOptions} />
  ) : (
    <div className="connect-spinner">Connecting...</div>
  );
};
```

WebChat.tsx (Chat Component)

```tsx
import { createStore } from 'botframework-webchat';

const MinimizableWebChat = () => {
  const [minimized, setMinimized] = useState(true);
  const store = useMemo(() => createStore(), []);

  return minimized ? (
    <button onClick={() => setMinimized(false)}>💬 Chat</button>
  ) : (
    <div className="chat-container">
      <header>
        <button onClick={() => setMinimized(true)}>Minimize</button>
      </header>
      <WebChat store={store} />
    </div>
  );
};
```

Web Chat Branding & Styling

Idiosyncratic Manual Styling - StyleSet

- **Idiosyncratic Manual Styling** offers granular control over the chat interface's appearance, enabling developers to tailor the user experience to specific requirements.
- **styleOptions** provides a simply & effective option to style Web Chat component, but it may not cover advanced UI customization needs.
- **styleSet** offers more advanced styling but may require updates with the new Web Chat releases.
- **createStyleSet** function is used to define base styles, such as background colors, text colors, font family, font size, and border radii for chat bubbles
- The **textContent** component is further customized to have bold text and justified alignment.

```
(async function () {
  // 1. Authenticate with MSAL
  const msalInstance = new msal.PublicClientApplication({
    auth: {
      clientId: 'YOUR_ENTRA_APP_CLIENT_ID',
      authority: 'https://login.microsoftonline.com/YOUR_TENANT_ID'
    }
  });
  await msalInstance.initialize();
  const tokenResponse = await msalInstance.loginPopup({
    scopes: ['https://api.powerplatform.com/.default']
  });

  // 2. Create Copilot Studio connection
  const client = new CopilotStudioClient({
    environmentId: 'YOUR_ENVIRONMENT_ID',
    botId: 'YOUR_BOT_ID'
  }, tokenResponse.accessToken);
  const connection = CopilotStudioWebChat.createConnection(client);

  // 3. Create styleSet for advanced customization
  const styleSet = window.WebChat.createStyleSet({
    backgroundColor: '#F0F0F0',
    bubbleBackground: '#FFFFFF',
    bubbleTextColor: '#000000',
    bubbleFromUserBackground: '#0178D4',
    bubbleFromUserTextColor: '#FFFFFF',
    fontFamily: "'Segoe UI', sans-serif",
    fontSize: '14px',
    bubbleBorderRadius: 8,
    bubbleFromUserBorderRadius: 8
  });

  // 4. Override textContent for advanced styling
  styleSet.textContent = {
    ...styleSet.textContent,
    fontWeight: 'bold',
    textAlign: 'justify'
  };

  // 5. Render WebChat
  window.WebChat.renderWebChat({
    directLine: connection,
    styleSet: styleSet
  }, document.getElementById('webchat'));
})();
```

Web Chat branding & styling

When to use stylSet over stylOptions?

Feature	styleOptions	styleSet
Change bubble background Color	✓	✓
Change font size & Color globally	✓	✓
Customize timestamps	✗	✓
Modify suggestion action alignment	✗	✓
Style the typing indicator	✗	✓
Add custom border styles to bubbles	✗	✓
Modify scrollbar appearance	✗	✓

Adaptive cards & intent recognition

Godwin Tenzing & Jose Del Castillo

Adaptive Cards in Web Chat

Adaptive Cards & Intent Recognition

Adaptive Cards are lightweight, UI-rich components that dynamically present information and collect user inputs. These cards allow:

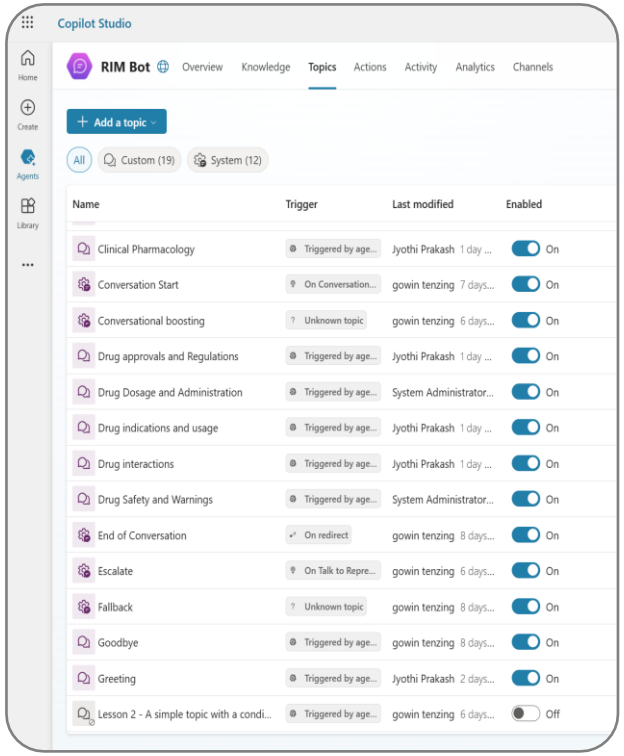
- Dynamic content rendering without modifying Web Chat UI.
- Interactive elements like buttons, forms, and input fields.
- Context-aware actions to guide user flow.
- Seamless integration with Copilot Studio and Direct Line API.

How do I create, configure & use Adaptive Cards?

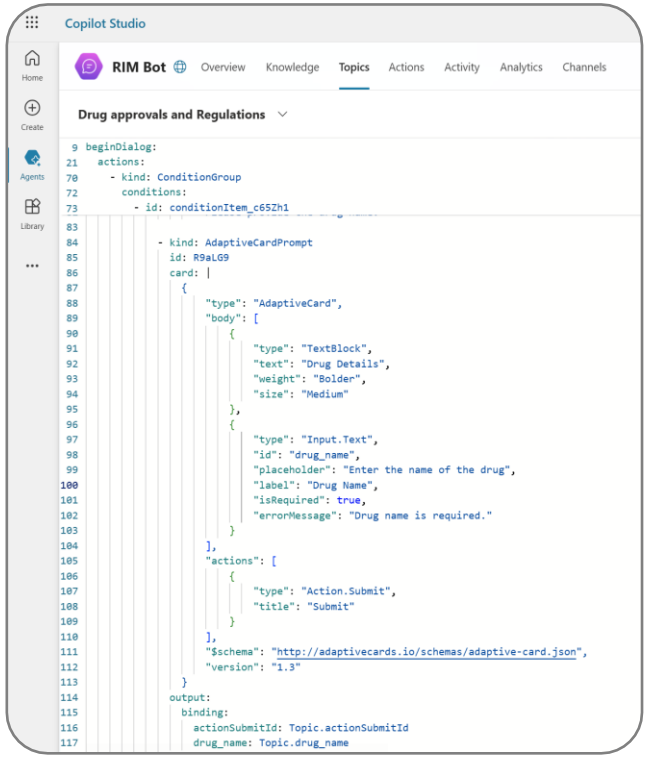
Adaptive Cards & Intent Recognition

Adaptive Card Designer - https://www.adaptivecards.io/designer/

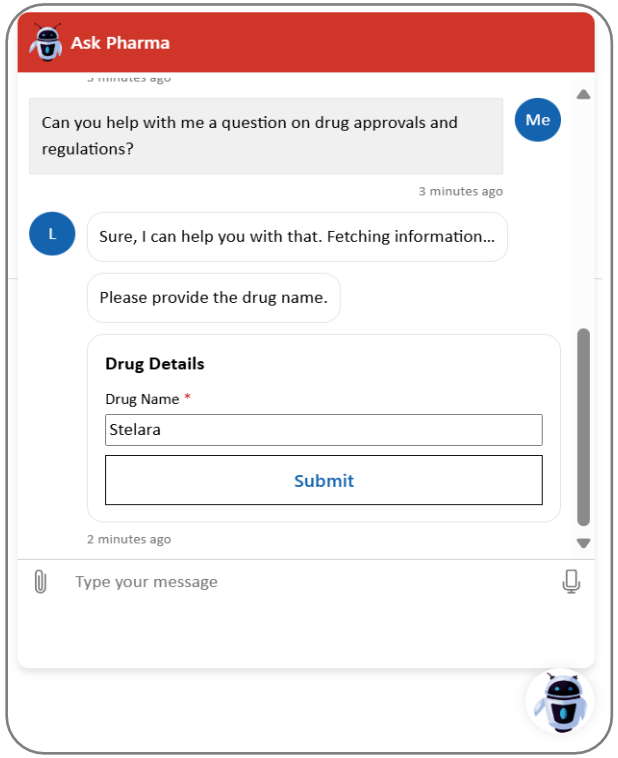
Create or Update an Existing Topic



Design using Adaptive Card Designer & Include it in the CoPilot Studio Agent Topic



RenderWebChat control using Direct Line renders the Adaptive Card



Intent Recognition & Handling Ambiguous Inputs

Adaptive Cards & Intent Recognition

- GenAI-powered natural language processing (NLP) instead of classic Topics, which are manually defining trigger phrases such as “can you help with drug dosage guidance”, “how much do I owe”, “I want to pay my bill” etc.
- Classic or Dynamic (AI-Powered) Intent, when an ambiguous intent is detected, you can use Adaptive Cards to Clarify the intent.
- When the agent is unable to help, design it to handover to a human agent, if enabled.

Voice-Based Interaction In Web Chat & Interactive Voice Response (IVR) via Telephony System

Godwin Tenzing & Jose Del Castillo

Voice Based Interaction and IVR

Voice Based Interaction

- Allows users to communicate using speech instead of typing.
- Enhances user accessibility and engagement.

Interactive Voice Based Response

- Automated telephony system that interacts with humans through voice.
- Enables users to navigate systems using voice commands or keypad inputs.

Web-Based Voice Chat & IVR

Web Chat with **Direct Line** and **Azure Speech Services** enables **voice-based interactions** with Microsoft Copilot Studio Bot. It allows users to speak directly to the chatbot using Microsoft Cognitive Services' speech recognition and synthesis capabilities.

Benefits

- Enhances Accessibility
- Improves User Engagement
- Optimized for Web & Devices without additional plugins.
- Multi-lingual support.

Features

- **Barge-In –** Users can interrupt the Bot speech
- **Latency Message**
- **Silence Detection & Timeout**
- **Speech Synthesis Markup Language (SSML)** - Control how your agent's voice sounds and behaves with users.
- **DTMF (Dual Tone Multi-Frequency) –** User can input via a Phone keyboard (Not via Web)

Use-Cases

- Voice Assistants
- Smart displays with embedded browsers (Amazon Echo Show, Google Nest Hub)
- KIOSK-based interactions
- Call Center Automated 1st line support operations
- Progressive Web Apps (PWA) with voice

Speech Integration (Direct Line Speech)

Copilot Studio Web Chat Voice Integration Options

Overview

- Single API Call & Unified Channel for text + speech
- Unified Credential (One token for bot connection & speech
- Uses Azure Cognitive Services Speech SDK
- Low-latency real-time voice interaction

Prerequisites

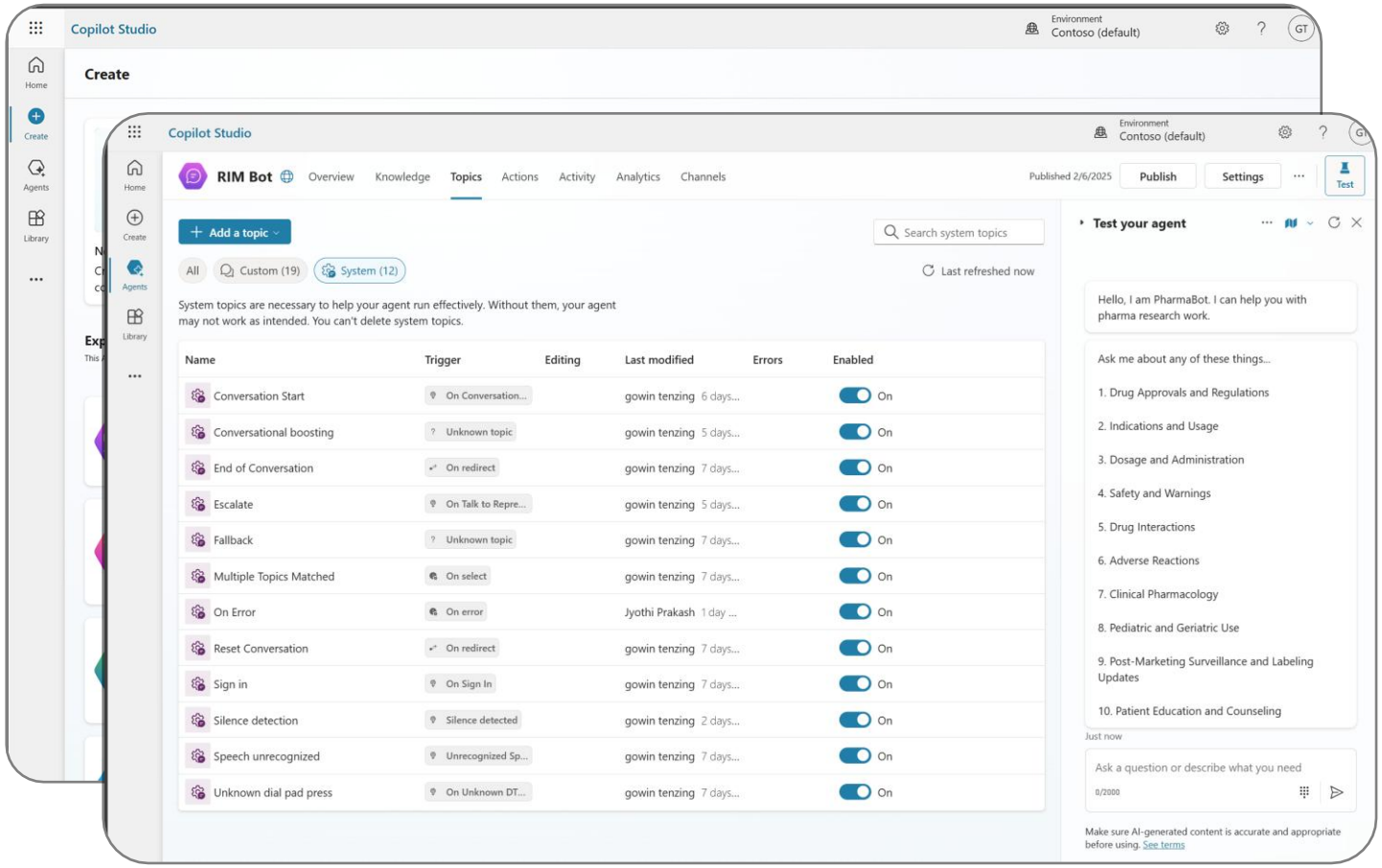
- Create Azure Speech Service resource
- Create Azure AD App Registration for the bot
- Deploy Proxy Bot to forward to Copilot Studio
- Set up token server endpoint

```
// Fetch Speech token for DLS
console.log('🔑 Fetching Speech token for DLS...');
const speechToken = await fetchSpeechToken();

// Create Direct Line Speech adapters
// Single WebSocket for both audio and messaging
const dlsAdapters = await createAdapters({
  fetchCredentials: async () => ({
    authorizationToken: speechToken.token,
    region: speechToken.region,
  }),
  speechRecognitionLanguage: speechToken.locale || 'en-GB',
  enableTelemetry: false,
});

// Ready for WebChat
setAdapters(dlsAdapters);
setConnectionStatus('connected');
```

Create a CoPilot Studio Agent to support Voice Capabilities

CoPilot Studio Agent

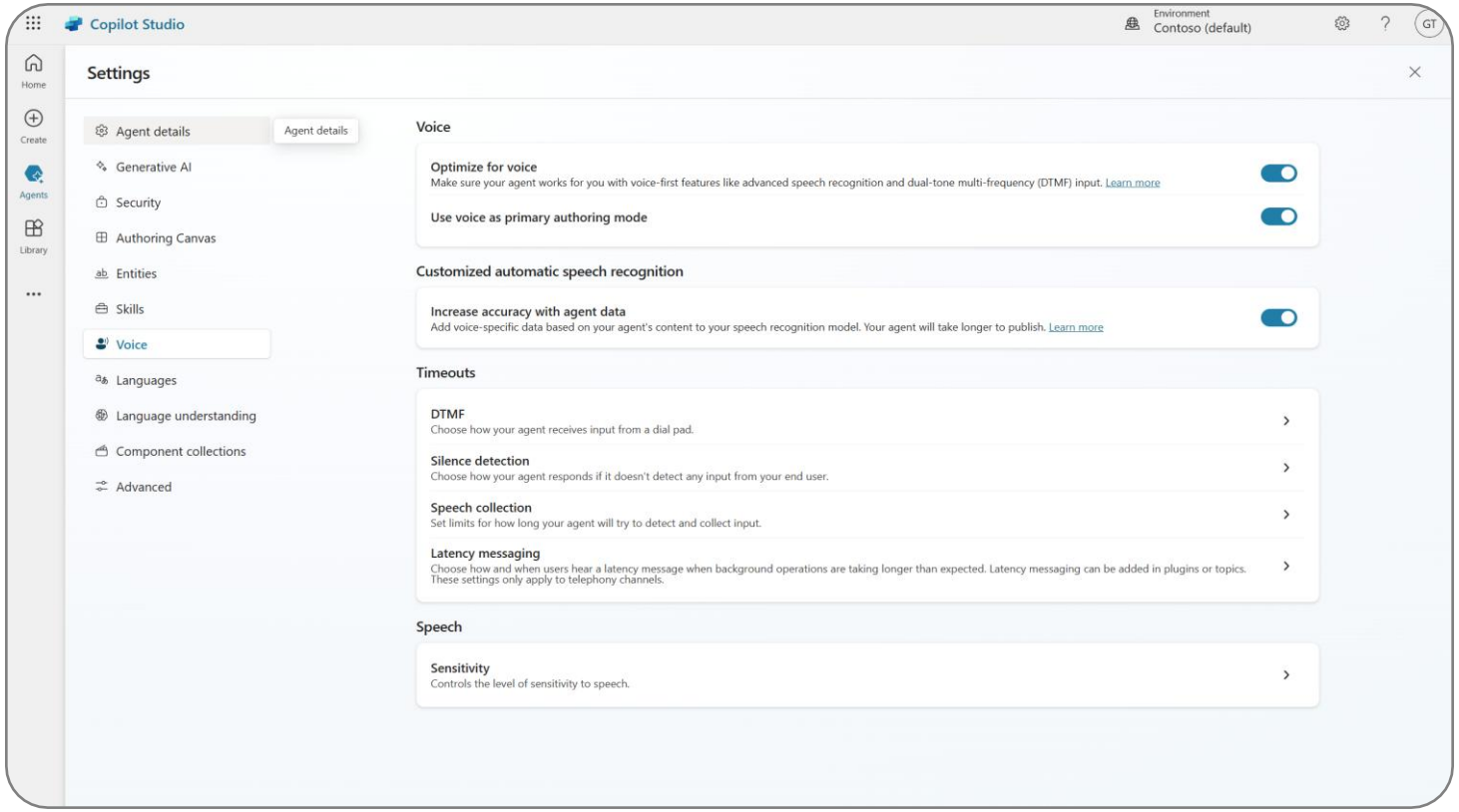
- Leverage an existing Voice Template to build a new agent.
- Voice Template includes topics and actions that are required to support voice capabilities.
- Voice Template provides:
  - Voice-optimized system topics & speech-friendly prompts
  - Default neural voice & locale settings
  - DTMF mappings, silence detection & latency messages
- Alternative: Enable voice on an existing agent via Settings → Channels → Direct Line Speech



Setting up Voice Based & IVR Interaction in Copilot Studio

CoPilot Studio Agent Configurations

- Enable Voice
- Set Primary Authoring Mode
- Configure DTMFs, Silence Detection, Latency Messaging and Speech Collection time-outs
- Configure Speech Sensitivity



Summary of Direct Line Speech Set up

Key Steps	Action	Where to Set Up?
Enable Voice for Agent	Enable & configure voice system topics	Copilot Studio
Direct Line Speech Channel	Enable DLS channel & link Speech resource	**Azure Bot Service** (Proxy Bot)
Create API Endpoint	/api/speechservices/token (returns authToken + region)	Solution Backend Code
Configure Web Chat	createAdapters (DLS SDK)	Web Chat (React/JS)
Deploy Web Chat	Host Web Chat UI	Azure Web App / Static Web App / Power Pages
Test & Debug	Verify Speech & IVR	Web Chat, Browser Console

Alternative approach

Speech Ponyfill

Copilot Studio Web Chat Voice Integration Options

Overview

- Two separate channels: Direct Line + Speech Services
- Add voice to existing text-based bots
- Full control over speech configuration
- Conversation history preserved
- Supports proactive messaging

Step 1: Fetch Both Credentials

```
// Step 1: Fetch both credentials in parallel
const [dlToken, speechCreds] = await Promise.all([
  fetchDirectLineToken(userId, userName),
  fetchPonyfillCredentials(locale, voice),
]);
```

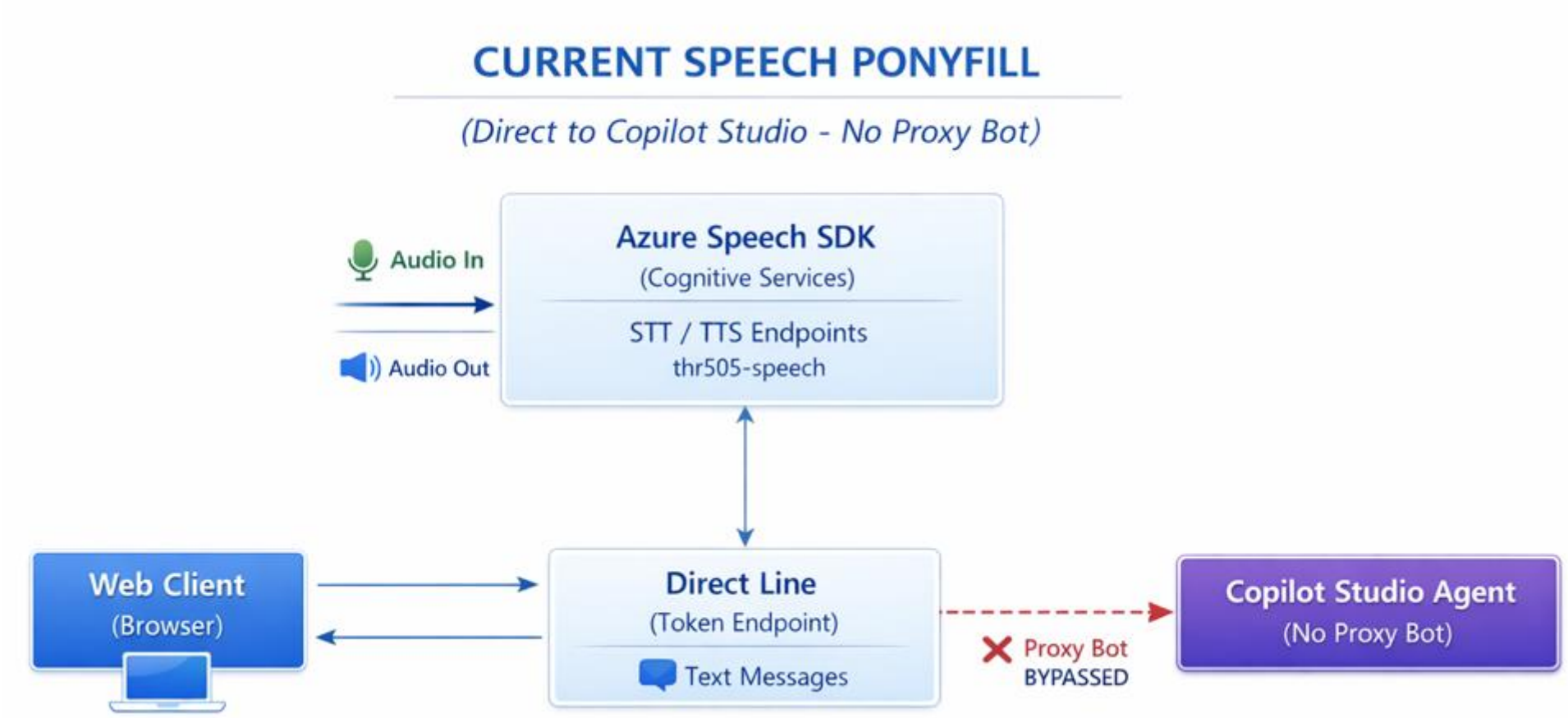
Step 2: Create Connections (Create Direct Line + Ponyfill

```
// Step 2: Create Direct Line + Speech Ponyfill
const directLine = createDirectLine({ token: dlToken.token });

const ponyfill = await createSpeechServicesPonyfill({
  credentials: {
    authorizationToken: speechCreds.token,
    region: speechCreds.region,
  },
  speechSynthesisOutputFormat: 'audio-24khz-48kbitrate-mono-mp3',
});
```

Step 3: Render Web Chat

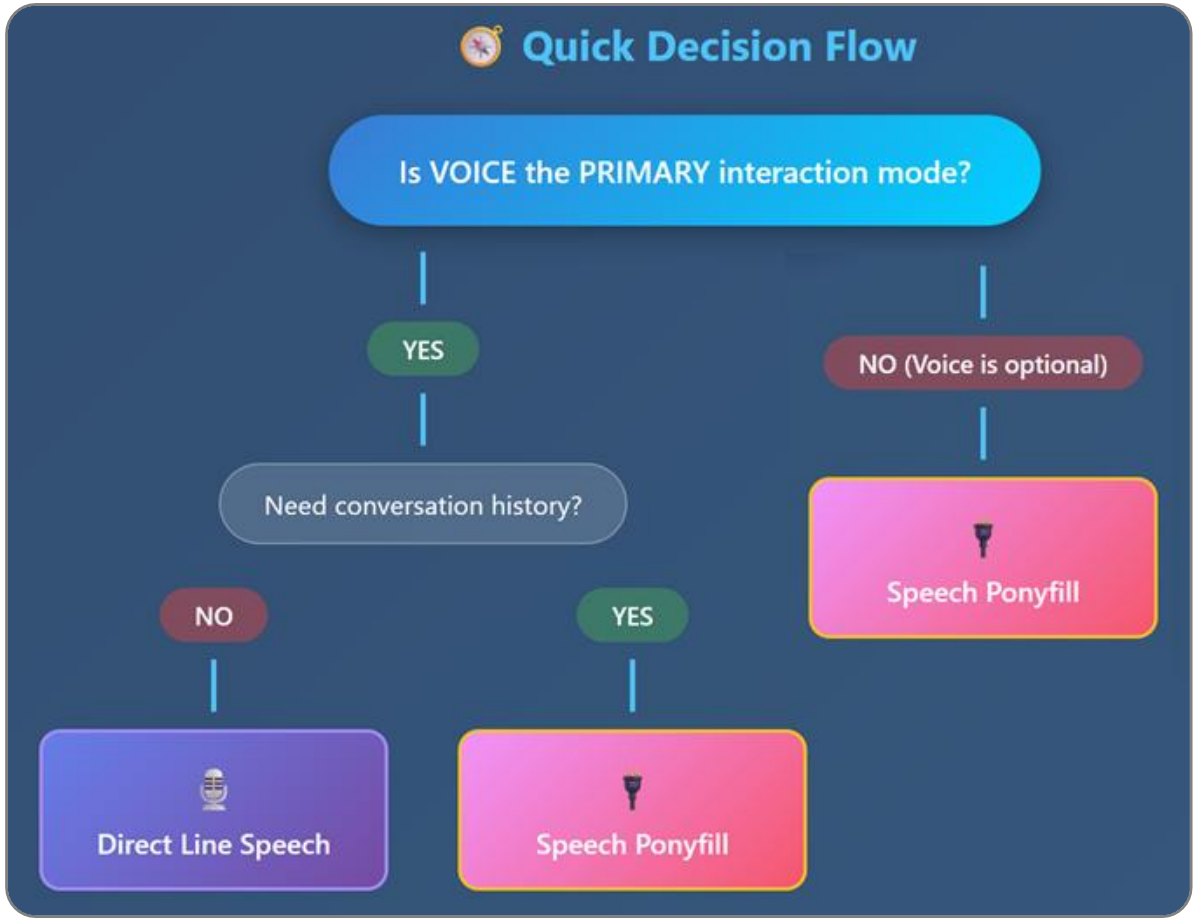
```
// Step 3: Render Web Chat with voice
<ReactWebChat
  directLine={directLine}
  webSpeechPonyfillFactory={ponyfillFactory}
/>
```

Speech Ponyfill approach



Speech Integration Decision Guide

Choose the Right Approach for Your Copilot Studio Web Chat



Choose Direct Line Speech

- ✅ Voice-first, new project
- ✅ Lowest latency required
- ✅ Native barge-in support
- ✅ Single WebSocket connection
- ⚠️ Requires Proxy Bot for Copilot Studio

Choose Speech Ponyfill

- ✅ Existing text-based bot
- ✅ Conversation history needed
- ✅ Voice is optional/add-on feature
- ✅ Custom speech configuration
- ✅ Works directly with Copilot Studio

Direct Line Speech & Ponyfill feature comparison

Feature	Direct Line Speech	Speech Ponyfill
Setup Complexity	◐ **Moderate** (Proxy Bot required)	✓ Simple
Credentials Required	✓ 1 (unified)	✗ 2 (separate)
Voice-First UX	✓ Optimized	◐ Possible
Conversation History	✗ Not persisted	✓ Persisted
Proactive Messages	✗ Not supported	✓ Supported
Custom Speech Models	◐ Limited	✓ Full control
Audio Format Control	✗ No	✓ Yes
Add to Existing Bot	✗ Requires Proxy Bot	✓ No changes needed
Latency	✓ Low	◐ Moderate

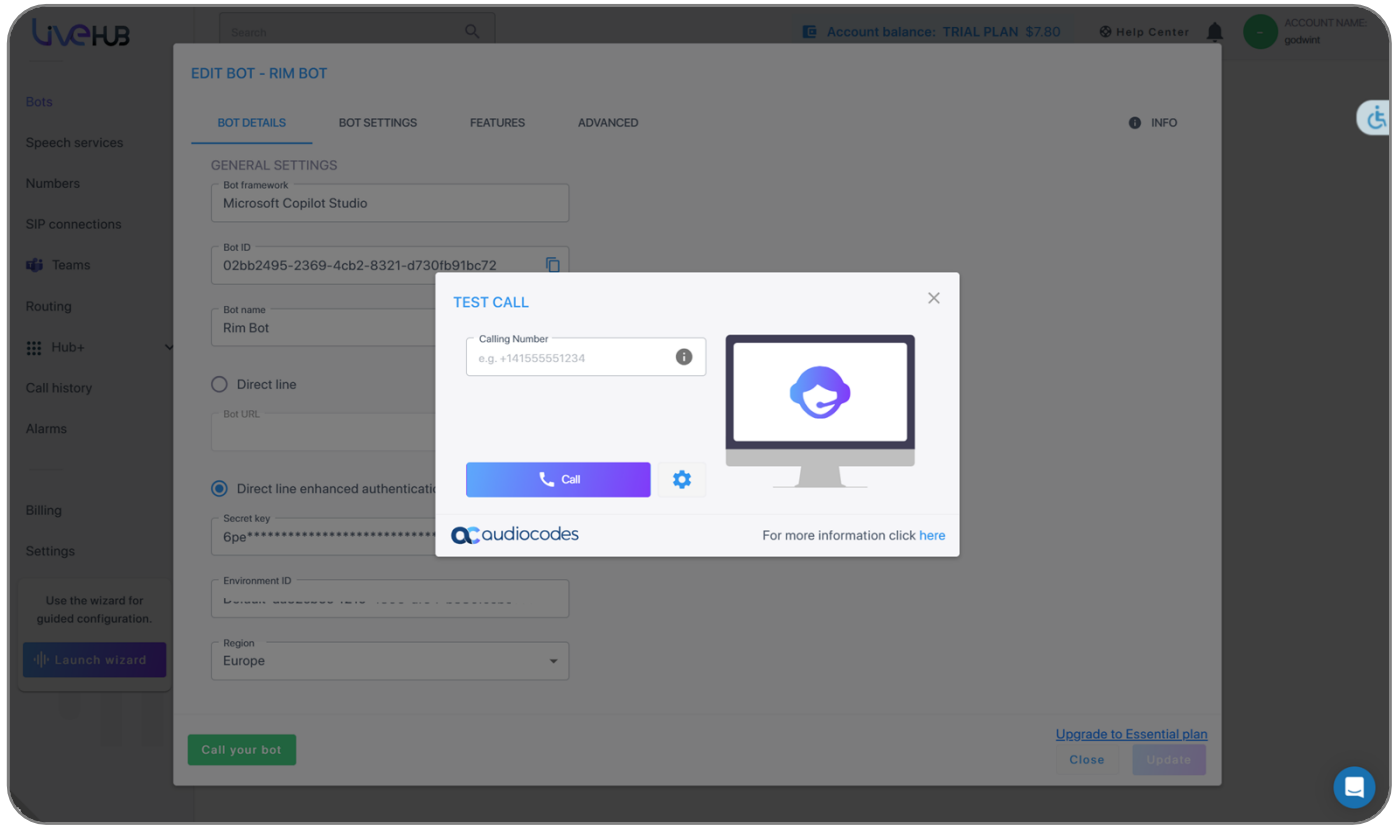
Note: For Copilot Studio, Direct Line Speech requires a Proxy Bot deployment since Copilot Studio doesn't support the DLS channel natively. This adds infrastructure complexity.

Setting up Telephony IVR using Partner/3rd Party Services

Setting up Telephony IVR using LiveHub

(Partner & 3rd Party Services)

- **LiveHub** is a platform for integrating **Microsoft Copilot Studio** bots with **Telephony IVR**.
- **AudioCodes** provides **voice connectivity,** allowing **phone-based** interactions with bots.
- Enables **voice-based IVR** functionality for **customer support, automated call routing, and conversational AI.**



Setting up Telephony IVR using LiveHub

(Partner & 3rd Party Services)

🤖 **Microsoft Copilot Studio Bot** - Ensure your bot is created and functional

🌐 **LiveHub Account** - Access the portal for bot configuration

📞 **AudioCodes Subscription** - For telephony/SIP integration

🔐 **Direct Line Enhanced Authentication** - For secure communication (/api/directline/livehubToken)

☎️ **Dedicated Phone Number** - For inbound and outbound calls

🔧 **Token Server Endpoint** *(optional to add)* - Backend API for secure token issuance

Telephony / IVR

Key takeaways

- Customizing Web Chat's appearance to align with your brand
- Integrating Copilot Studio with web chats using React Controls
- Leveraging voice-relevant takeaway for precise intent recognition and resolving ambiguities
- Enabling voice-based interactions and Telephony IVR to boost user engagement

Resources

- M365 Agents SDK Documentation
  - https://aka.ms/M365-Agents-SDK-Docs
- GitHub Repositories
  - github.com/microsoft/Agents (Main)
  - github.com/microsoft/Agents-for-net (.NET)
  - github.com/microsoft/Agents-for-js (JavaScript)
  - github.com/microsoft/Agents-for-python (Python)
- WebChat Customization
  - github.com/microsoft/BotFramework-WebChat/blob/main/docs/API.md
- Copilot Studio
  - copilotstudio.microsoft.com