~~~
# Vite (Javascript)

> Learn how to create a web application that enables voice conversations with ElevenLabs AI
agents

This tutorial will guide you through creating a web client that can interact with a
Conversational AI agent. You'll learn how to implement real-time voice conversations,
allowing users to speak with an AI agent that can listen, understand, and respond naturally
using voice synthesis.

<Note>
  Looking to build with React/Next.js? Check out our [Next.js
  guide](/docs/conversational-ai/guides/quickstarts/next-js)
</Note>

## What You'll Need

1. An ElevenLabs agent created following [this guide](/docs/conversational-ai/quickstart)
2. `npm` installed on your local system
3. Basic knowledge of JavaScript

<Note>
  Looking for a complete example? Check out our [Vanilla JS demo on
  GitHub](https://github.com/elevenlabs/elevenlabs-
examples/tree/main/examples/conversational-ai/javascript).
</Note>

## Project Setup

<Steps>
  <Step title="Create a Project Directory">
    Open a terminal and create a new directory for your project:

    ```bash
    mkdir elevenlabs-conversational-ai
    cd elevenlabs-conversational-ai
    ```
  </Step>

  <Step title="Initialize npm and Install Dependencies">
    Initialize a new npm project and install the required packages:

    ```bash
    npm init -y
    npm install vite @11labs/client
    ```
  </Step>

  <Step title="Set up Basic Project Structure">
    Add this to your `package.json`:

    ```json package.json {4}
    {
        "scripts": {
            ...
            "dev:frontend": "vite"
        }
    }
    ```

    Create the following file structure:

    ```shell {2,3}
    elevenlabs-conversational-ai/
~~~

~~~
    â”œâ”€â”€ index.html
    â”œâ”€â”€ script.js
    â”œâ”€â”€ package-lock.json
    â”œâ”€â”€ package.json
    â””â”€â”€ node_modules
    ```
  </Step>
</Steps>

## Implementing the Voice Chat Interface

<Steps>
  <Step title="Create the HTML Interface">
    In `index.html`, set up a simple user interface:

    <Frame background="subtle">
      ![](file:a7a4d92e-757f-4e05-9c66-1698f75b87c4)
    </Frame>

    ```html index.html
    <!DOCTYPE html>
    <html lang="en">
        <head>
            <meta charset="UTF-8" />
            <meta name="viewport" content="width=device-width, initial-scale=1.0" />
            <title>ElevenLabs Conversational AI</title>
        </head>
        <body style="font-family: Arial, sans-serif; text-align: center; padding: 50px;">
            <h1>ElevenLabs Conversational AI</h1>
            <div style="margin-bottom: 20px;">
                <button id="startButton" style="padding: 10px 20px; margin: 5px;">Start
Conversation</button>
                <button id="stopButton" style="padding: 10px 20px; margin: 5px;"
disabled>Stop Conversation</button>
            </div>
            <div style="font-size: 18px;">
                <p>Status: <span id="connectionStatus">Disconnected</span></p>
                <p>Agent is <span id="agentStatus">listening</span></p>
            </div>
            <script type="module" src="../images/script.js"></script>
        </body>
    </html>
    ```
  </Step>

  <Step title="Implement the Conversation Logic">
    In `script.js`, implement the functionality:

    ```javascript script.js
    import { Conversation } from '@11labs/client';

    const startButton = document.getElementById('startButton');
    const stopButton = document.getElementById('stopButton');
    const connectionStatus = document.getElementById('connectionStatus');
    const agentStatus = document.getElementById('agentStatus');

    let conversation;

    async function startConversation() {
        try {
            // Request microphone permission
            await navigator.mediaDevices.getUserMedia({ audio: true });

            // Start the conversation
            conversation = await Conversation.startSession({
~~~

```
                agentId: 'YOUR_AGENT_ID', // Replace with your agent ID
                onConnect: () => {
                    connectionStatus.textContent = 'Connected';
                    startButton.disabled = true;
                    stopButton.disabled = false;
                },
                onDisconnect: () => {
                    connectionStatus.textContent = 'Disconnected';
                    startButton.disabled = false;
                    stopButton.disabled = true;
                },
                onError: (error) => {
                    console.error('Error:', error);
                },
                onModeChange: (mode) => {
                    agentStatus.textContent = mode.mode === 'speaking' ? 'speaking' :
'listening';
                },
            });
        } catch (error) {
            console.error('Failed to start conversation:', error);
        }
    }

    async function stopConversation() {
        if (conversation) {
            await conversation.endSession();
            conversation = null;
        }
    }

    startButton.addEventListener('click', startConversation);
    stopButton.addEventListener('click', stopConversation);
    ```
  </Step>

  <Step title="Start the frontend server">
    ```shell
    npm run dev:frontend
    ```
  </Step>
</Steps>

<Note>
  Make sure to replace

  `'YOUR_AGENT_ID'`

   with your actual agent ID from ElevenLabs.
</Note>

<Accordion title="(Optional) Authenticate with a Signed URL">
  <Note>
    This authentication step is only required for private agents. If you're using a public
agent, you can skip this section and directly use the `agentId` in the `startSession` call.
  </Note>

  <Steps>
    <Step title="Create Environment Variables">
      Create a `.env` file in your project root:

      ```env .env
      ELEVENLABS_API_KEY=your-api-key-here
      AGENT_ID=your-agent-id-here
      ```
```
```

~~~
      <Warning>
        Make sure to add `.env` to your `.gitignore` file to prevent accidentally committing
sensitive credentials.
      </Warning>
    </Step>

    <Step title="Setup the Backend">
      1. Install additional dependencies:

      ```bash
      npm install express cors dotenv
      ~~~

      2. Create a new folder called `backend`:

      ```shell {2}
      elevenlabs-conversational-ai/
      â”œâ”€â”€ backend
      ...
      ```
    </Step>

    <Step title="Create the Server">
      ```javascript backend/server.js
      require("dotenv").config();

      const express = require("express");
      const cors = require("cors");

      const app = express();
      app.use(cors());
      app.use(express.json());

      const PORT = process.env.PORT || 3001;

      app.get("/api/get-signed-url", async (req, res) => {
          try {
              const response = await fetch(
                  `https://api.elevenlabs.io/v1/convai/conversation/get_signed_url?
agent_id=${process.env.AGENT_ID}`,
                  {
                      headers: {
                          "xi-api-key": process.env.ELEVENLABS_API_KEY,
                      },
                  }
              );

              if (!response.ok) {
                  throw new Error("Failed to get signed URL");
              }

              const data = await response.json();
              res.json({ signedUrl: data.signed_url });
          } catch (error) {
              console.error("Error:", error);
              res.status(500).json({ error: "Failed to generate signed URL" });
          }
      });

      app.listen(PORT, () => {
          console.log(`Server running on http://localhost:${PORT}`);
      });
      ```
    </Step>
```
```

~~~
    <Step title="Update the Client Code">
      Modify your `script.js` to fetch and use the signed URL:

      ```javascript script.js {2-10,16,19,20}
      // ... existing imports and variables ...

      async function getSignedUrl() {
          const response = await fetch('http://localhost:3001/api/get-signed-url');
          if (!response.ok) {
              throw new Error(`Failed to get signed url: ${response.statusText}`);
          }
          const { signedUrl } = await response.json();
          return signedUrl;
      }

      async function startConversation() {
          try {
              await navigator.mediaDevices.getUserMedia({ audio: true });

              const signedUrl = await getSignedUrl();

              conversation = await Conversation.startSession({
                  signedUrl,
                  // agentId has been removed...
                  onConnect: () => {
                      connectionStatus.textContent = 'Connected';
                      startButton.disabled = true;
                      stopButton.disabled = false;
                  },
                  onDisconnect: () => {
                      connectionStatus.textContent = 'Disconnected';
                      startButton.disabled = false;
                      stopButton.disabled = true;
                  },
                  onError: (error) => {
                      console.error('Error:', error);
                  },
                  onModeChange: (mode) => {
                      agentStatus.textContent = mode.mode === 'speaking' ? 'speaking' :
'listening';
                  },
              });
          } catch (error) {
              console.error('Failed to start conversation:', error);
          }
      }

      // ... rest of the code ...
      ```

      <Warning>
        Signed URLs expire after a short period. However, any conversations initiated before
expiration will continue uninterrupted. In a production environment, implement proper error
handling and URL refresh logic for starting new conversations.
      </Warning>
    </Step>

    <Step title="Update the package.json">
      ```json package.json {4,5}
      {
          "scripts": {
              ...
              "dev:backend": "node backend/server.js",
              "dev": "npm run dev:frontend & npm run dev:backend"
~~~

~~~
          }
        }
        ```
      </Step>

      <Step title="Run the Application">
        Start the application with:

        ```bash
        npm run dev
        ```
      </Step>
    </Steps>
  </Accordion>

  ## Next Steps

  Now that you have a basic implementation, you can:

  1. Add visual feedback for voice activity
  2. Implement error handling and retry logic
  3. Add a chat history display
  4. Customize the UI to match your brand

  <Info>
    For more advanced features and customization options, check out the
    [@11labs/client](https://www.npmjs.com/package/@11labs/client) package.
  </Info>
~~~