Web SDK

Integrate Vapi into your web application.

The Vapi Web SDK provides web developers a simple API for interacting with the real-time call functionality of Vapi.

Installation

Install the package:

```
$ yarn add @vapi-ai/web
```

or w/ npm:

```
$ npm install @vapi-ai/web
```

Importing

Import the package:

```
import Vapi from "@vapi-ai/web";
```

Then, create a new instance of the Vapi class, passing your **Public Key** as a parameter to the constructor:

```
const vapi = new Vapi("your-public-key");
```

You can find your public key in the Vapi Dashboard.

Usage

`.start()`

You can start a web call by calling the `.start()` function. The `start` function can either accept:

1. **a string**, representing an assistant ID
2. **an object**, representing a set of assistant configs (see Create Assistant)

The `start` function returns a promise that resolves to a call object. For example:

```
const call = await vapi.start(assistantId);
// { "id": "bd2184a1-bdea-4d4f-9503-b09ca8b185e6", "orgId": "6da6841c-0f
```

Passing an Assistant ID

If you already have an assistant that you created (either via the Dashboard or the API), you can start the call with the assistant's ID:

```
vapi.start("79f3XXXX-XXXX-XXXX-XXXX-XXXXXXXXce48");
```

Passing Assistant Configuration Inline

You can also specify configuration for your assistant inline.

This will not create a persistent assistant that is saved to your account, rather it will create an ephemeral assistant only used for this call specifically.

You can pass the assistant's configuration in an object (see Create Assistant for a list of acceptable fields):

```
vapi.start({
  transcriber: {
    provider: "deepgram",
    model: "nova-2",
    language: "en-US",
  },
  model: {
    provider: "openai",
    model: "gpt-3.5-turbo",
    messages: [
      {
        role: "system",
        content: "You are a helpful assistant.",
      },
    ],
  },
  voice: {
    provider: "playht",
    voiceId: "jennifer",
  },
  name: "My Inline Assistant",
  ...
});
```

Overriding Assistant Configurations

To override assistant settings or set template variables, you can pass `assistantOverrides` as the second argument.

For example, if the first message is "Hello `{{name}}`", set `assistantOverrides` to the following to replace `{{name}}` with `John`:

```
const assistantOverrides = {
  transcriber: {
    provider: "deepgram",
    model: "nova-2",
    language: "en-US",
  },
  recordingEnabled: false,
  variableValues: {
    name: "Alice",
  },
};

vapi.start("79f3XXXX-XXXX-XXXX-XXXX-XXXXXXXXce48", assistantOverrides);
```

`.send()`

During the call, you can send intermediate messages to the assistant (like background messages).

- `type` will always be `"add-message"`
- the `message` field will have 2 items, `role` and `content`.

```
vapi.send({
  type: "add-message",
  message: {
    role: "system",
    content: "The user has pressed the button, say peanuts",
  },
});
```

> Possible values for role are `system`, `user`, `assistant`, `tool` or `function`.

`.stop()`

You can stop the call session by calling the `stop` method:

```
vapi.stop();
```

This will stop the recording and close the connection.

`.isMuted()`

Check if the user's microphone is muted:

```
vapi.isMuted();
```

`.setMuted(muted: boolean)`

You can mute & unmute the user's microphone with `setMuted`:

```
vapi.isMuted(); // false
vapi.setMuted(true);
vapi.isMuted(); // true
```

`say(message: string, endCallAfterSpoken?: boolean)`

The `say` method can be used to invoke speech and gracefully terminate the call if needed

```
vapi.say("Our time's up, goodbye!", true)
```

Events

You can listen on the `vapi` instance for events. These events allow you to react to changes in the state of the call or user speech.

`speech-start`

Occurs when your AI assistant has started speaking.

```
vapi.on("speech-start", () => {
  console.log("Assistant speech has started.");
});
```

`speech-end`

Occurs when your AI assistant has finished speaking.

```
vapi.on("speech-end", () => {
  console.log("Assistant speech has ended.");
});
```

`call-start`

Occurs when the call has connected & begins.

```
vapi.on("call-start", () => {
  console.log("Call has started.");
});
```

`call-end`

Occurs when the call has disconnected & ended.

```
vapi.on("call-end", () => {
  console.log("Call has ended.");
});
```

`volume-level`

Realtime volume level updates for the assistant. A floating-point number between `0` & `1`.

```
vapi.on("volume-level", (volume) => {
  console.log(`Assistant volume level: ${volume}`);
});
```

`message`

Various assistant messages can be sent back to the client during the call. These are the same messages that your server would receive.

At assistant creation time, you can specify on the `clientMessages` field the set of messages you'd like the assistant to send back to the client.

Those messages will come back via the `message` event:

```
// Various assistant messages can come back (like function calls, transc
vapi.on("message", (message) => {
  console.log(message);
});
```

`error`

Handle errors that occur during the call.

```
vapi.on("error", (e) => {
  console.error(e);
});
```

Resources

NPM
View the package on NPM.

GitHub
View the package on GitHub.

Try Our Quickstart
Get up and running quickly with the Web SDK.