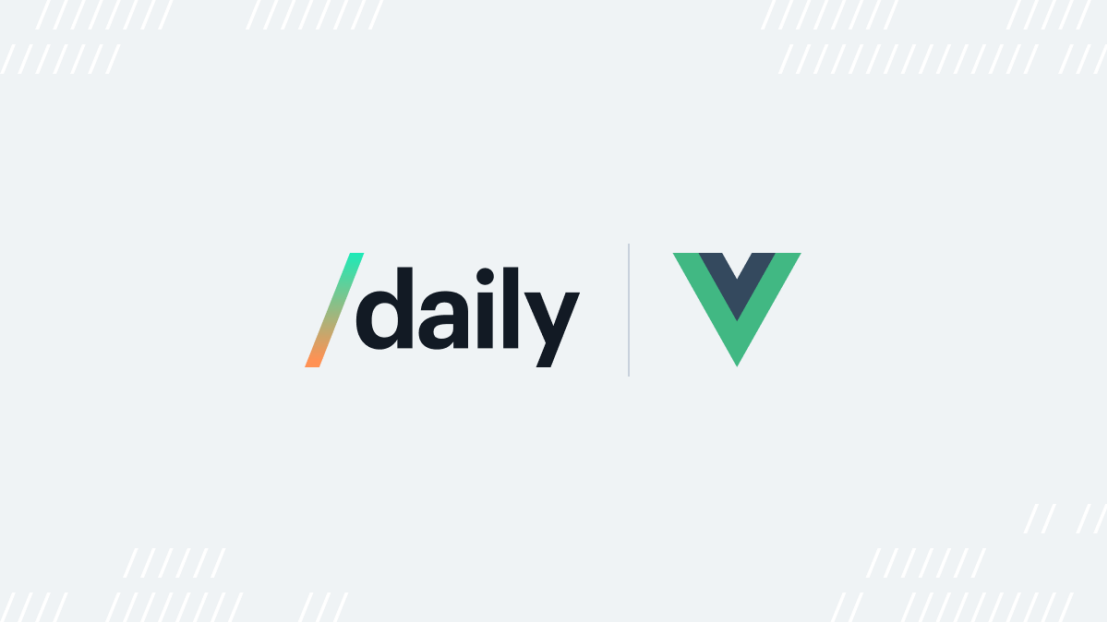
Code & Tutorials

Build a video chat app with Vue and Daily Prebuilt

Embed Daily Prebuilt and add custom video call controls to any Vue app



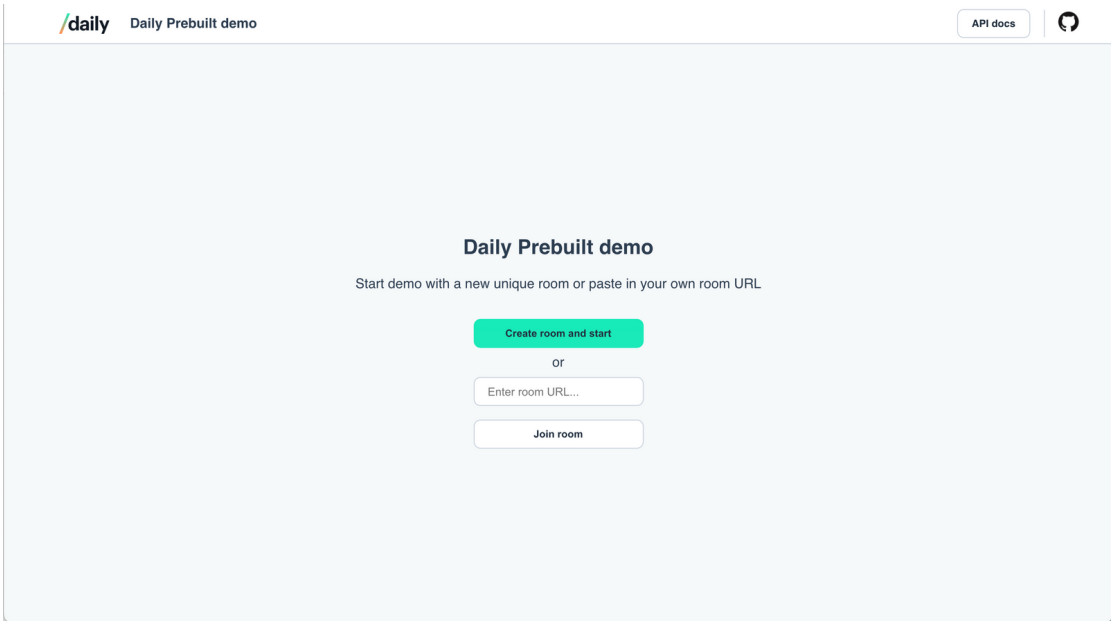
Jess Mitchell
Published: 30 Aug 2021 (Updated: 14 Oct 2021) • 12 min read



At Daily, we've spent a lot of time making sure our video and audio-only APIs can be used with any frontend framework, or no framework at all. 🍦 It's important to us to make flexible APIs that can be incorporated into any app looking to add audio and video chat. We've created several demos for our customers using plain JavaScript, React, Next.js, React Native, and more to help cover as many use cases as possible.

Recently, we decided to expand our demo coverage even more with one of our favourite frameworks: Vue!

In today's tutorial, we'll cover how to incorporate Daily Prebuilt into your Vue app, as well as how to programmatically manage Daily Prebuilt controls via your app's UI with our latest demo app.



Vue Daily Prebuilt demo home page

If you are interested in building a custom video chat app with Vue, fear not; we have an upcoming tutorial series on how to do just that. Stay tuned! 👀

Tutorial requirements

Before we get started, be sure to sign up for a Daily account. Once you're logged in, you can either create a room through the Dashboard or through the REST API.

For this tutorial, you can clone the Daily Prebuilt Vue demo repo and run it locally, or start from scratch and follow along as we build our Vue components.

To run the Daily Prebuilt Vue demo app locally, clone it and run the following in your terminal:

```
npm install
```

```
npm run serve
```

To view the app, open `http://localhost:8080` in the browser of your choice.

Create a new Vue app

If you prefer to create your own Vue app to add Daily Prebuilt to, start by installing the Vue CLI globally to your machine.

```
npm install -g @vue/cli
```

Once installed, we can create a new Vue app to add Daily Prebuilt to using the Vue CLI.

In your terminal, run:

```
vue create daily-prebuilt-demo
```

Once the project is created, go to the project's root directory and add `daily-js` as a dependency.

```
npm install @daily-co/daily-js
```

Then, following the same instructions as above for the demo app, start the server:

```
npm run serve
```

Demo project overview

The Daily Prebuilt Vue demo only has four components:

1. `App.vue` , the parent component for every other component

included in the app.

2. `Header.vue`, a completely optional component we included for the app's title and project links.
3. `Home.vue`, the main component which is where Daily Prebuilt is embedded and the control panel is added when in a Daily call.
4. `Controls.vue`, the control panel for programmatically controlling Daily Prebuilt. This is also optional but useful for understanding how to interact with `daily-js` to customize your app's usage of Daily Prebuilt.

We won't go into the details of what's happening in the `Header` since it's static content, but what's important to know is that the `App` component imports the `Header` and `Home` component, and both are displayed at all times.

```
<template>
 <Header />
 <Home />
</template>

<script>
import Home from "./components/Home.vue";
import Header from "./components/Header.vue";

export default {
 name: "App",
 components: {
   Home,
   Header,
 },
};
</script>
```

Ready to go Home: Importing Daily Prebuilt to your Vue app

The `Home` component is the most important one in this demo because it loads all the main content, including the Daily call and control panel.

The default view of the Home component will include two buttons and

an input:

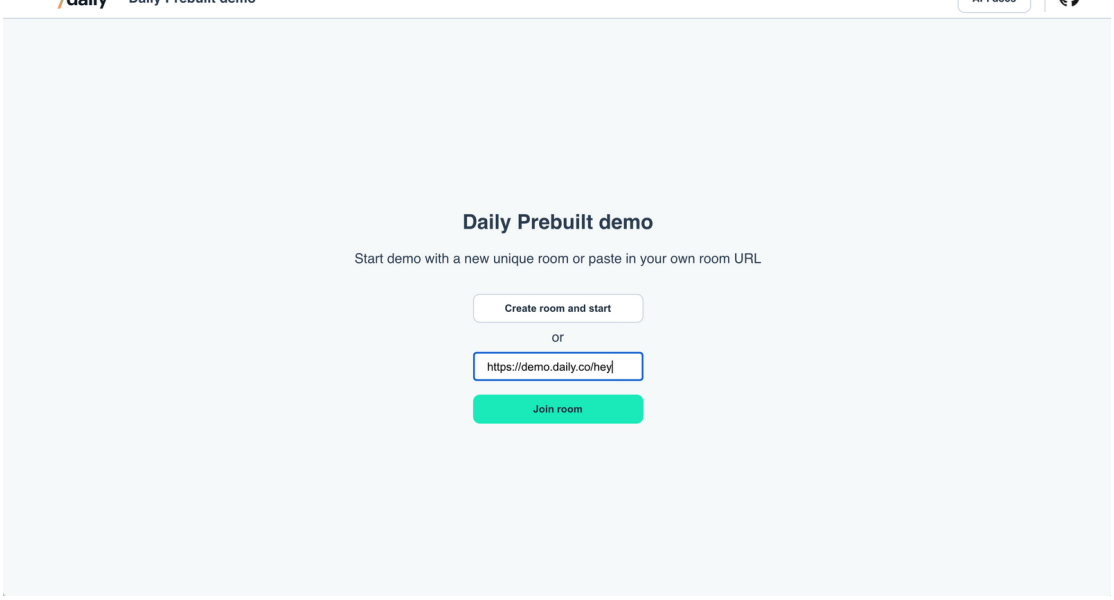
1. The first button is only used if you've deployed the app via Netlify, so we'll skip that for now. (Check out the project's README for more information.)
2. The input and second button are used to submit the Daily room URL you'll be joining (i.e. from the Daily room created above). The format of this URL is `https://YOUR_DAILY_DOMAIN.daily.co/ROOM_NAME`.

The container for this default home view is conditionally rendered depending on the `status` value in the component's data option.

```
<div class="home" v-if="status === 'home'">
 …
</div>
```

The status can be `home`, `lobby`, or `call`. `home` refers to the default view, before a call has been started, and `lobby` refers to when a call has been started but not joined yet. (We call this the "hair check" view sometimes too, so you can view yourself and set up your devices before joining a call.) Lastly, `call` refers to when you are live in a Daily call. We'll look at how the `status` value gets updated in a bit.

There is also a call container `div` that is included in the `Home` component, which is conditionally displayed depending on the app's current status. This means it is in the DOM in the default view but only visible to the user once a call has been started.



The join button enables when a valid Daily room URL is entered

Let's look at the Vue template for how this is set up:

```
<template>
 <main class="wrapper">
   <div class="home" v-if="status === 'home'">
     <h2>Daily Prebuilt demo</h2>
     <p>Start demo with a new unique room or paste in your own r
     <div class="start-call-container">
       <button @click="createAndJoinRoom" :disabled="runningLoca
         Create room and start
       </button>
       <p v-if="roomError" class="error">Room could not be creat
       <p class="subtext">or</p>
       <!-- Daily room URL is entered here -->
       <input
         type="text"
         placeholder="Enter room URL..."
         v-model="roomUrl"
         pattern="^(https:\/\/)?[\w.-]+(\.(daily\.(co)))+[\/\/]+
         @input="validateInput"
       />
       <!-- button to submit URL and join call -->
       <button @click="submitJoinRoom" :disabled="!validRoomURL"
         Join room
       </button>
     </div>
   </div>

   <div class="call-container" :class="{ hidden: status === 'hom
     <!-- The Daily Prebuilt iframe is embedded in the div below
     <div id="call" ref="callRef"></div>
     <!-- Only show the control panel if a call is live -->
```

```
      <controls
        v-if="status === 'call'"
        :roomUrl="roomUrl"
        :callFrame="callFrame"
      />
    </div>
  </main>
</template>
```

Home.vue

Now that we know how the `Home` component is structured, let's look at the JavaScript code that gives it functionality:

```
<script>
import DailyIframe from "@daily-co/daily-js";
import Controls from "./Controls.vue";
import api from "../api.js";

export default {
 components: { Controls },
 name: "Home",
 data() {
   return {
     roomUrl: "",
     status: "home",
     callFrame: null,
     validRoomURL: false,
     roomError: false,
     runningLocally: false,
   };
 },
 created() {
   if (window?.location?.origin.includes("localhost")) {
     this.runningLocally = true;
   }
 },
 methods: {
   createAndJoinRoom() {
     api
       .createRoom()
       .then((room) => {
         this.roomUrl = room.url;
         this.joinRoom(room.url);
       })
       .catch((e) => {
         console.log(e);
         this.roomError = true;
```

```
    });
  },
  // Daily callframe created and joined below
  joinRoom(url) {
    if (this.callFrame) {
      this.callFrame.destroy();
    }

    // Daily event callbacks
    const logEvent = (ev) => console.log(ev);
    const goToLobby = () => (this.status = "lobby");
    const goToCall = () => (this.status = "call");
    const leaveCall = () => {
      if (this.callFrame) {
        this.status = "home";
        this.callFrame.destroy();
      }
    };
    // DailyIframe container element
    const callWrapper = this.$refs.callRef;

    // Create Daily call
    const callFrame = DailyIframe.createFrame(callWrapper, {
      iframeStyle: {
        height: "auto",
        width: "100%",
        aspectRatio: 16 / 9,
        minWidth: "400px",
        maxWidth: "920px",
        border: "1px solid var(--grey)",
        borderRadius: "4px",
      },
      showLeaveButton: true,
    });
    this.callFrame = callFrame;

    // Add event listeners and join call
    callFrame
      .on("loaded", logEvent)
      .on("started-camera", logEvent)
      .on("camera-error", logEvent)
      .on("joining-meeting", goToLobby)
      .on("joined-meeting", goToCall)
      .on("left-meeting", leaveCall);

    callFrame.join({ url });
  },
  submitJoinRoom() {
    this.joinRoom(this.roomUrl);
  },
```

```
    validateInput(e) {
      this.validRoomURL = !!this.roomUrl && e.target.checkValidit
    },
  },
};
</script>
```

Let's start by focusing on the `joinRoom` method, which is where all the Daily video call ✨ magic ✨ happens.

```
joinRoom(url) {
  if (this.callFrame) {
    this.callFrame.destroy();
  }
  ...
```

In joinRoom, start by cleaning up any previous calls

First, if there is already a `callFrame` (i.e. the video call iframe), we destroy it to avoid multiple calls being loaded unintentionally. Defensive coding FTW. 🎉

```
// Daily event callbacks
const logEvent = (ev) => console.log(ev);
const goToLobby = () => (this.status = "lobby");
const goToCall = () => (this.status = "call");
const leaveCall = () => {
   if (this.callFrame) {
      this.status = "home";
      this.callFrame.destroy();
   }
};
```

Next, set the callbacks used by daily-js

Next, we set up the callbacks that will be used by `daily-js` whenever an event happens in the call that will affect our app's UI. This can be moved outside the `joinRoom` function too, but we won't worry about optimizing for now.

These callbacks are where we update our data options' `status` value

to know what stage of the call we're in.

```
const callWrapper = this.$refs.callRef;
```

Next, we select the `div` container that we'll instruct `daily-js` to embed the video call iframe into (the `DailyIframe` instance).

```
<div id="call" ref="callRef"></div>
```

If we look back at the DOM structure, there was a `div` included with a ref added to it to simplify selecting that `div` in our `joinRoom` method. This is what we're targeting with `const callWrapper = this.$refs.callRef;`

```
// Create Daily call
const callFrame = DailyIframe.createFrame(callWrapper, {
  iframeStyle: {
      height: "auto",
      width: "100%",
      aspectRatio: 16 / 9,
      minWidth: "400px",
      maxWidth: "920px",
      border: "1px solid var(--grey)",
      borderRadius: "4px",
  },
  showLeaveButton: true,
});

this.callFrame = callFrame;
```

Then, create the Daily iframe and set it in the data options for later use

Getting back to `joinRoom`, we then actually create the `DailyIframe` that will host our video call and assign it to the variable `callFrame`. This variable then gets assigned to our data option so it can be referenced later. (If you were using a state management library, you would add it to your app's state at this point.)

Note: The options passed to `createFrame`, like `iframeStyle`, are optional.

```
// Add event listeners and join call
callFrame
  .on("loaded", logEvent)
  .on("started-camera", logEvent)
  .on("camera-error", logEvent)
  .on("joining-meeting", goToLobby)
  .on("joined-meeting", goToCall)
  .on("left-meeting", leaveCall);

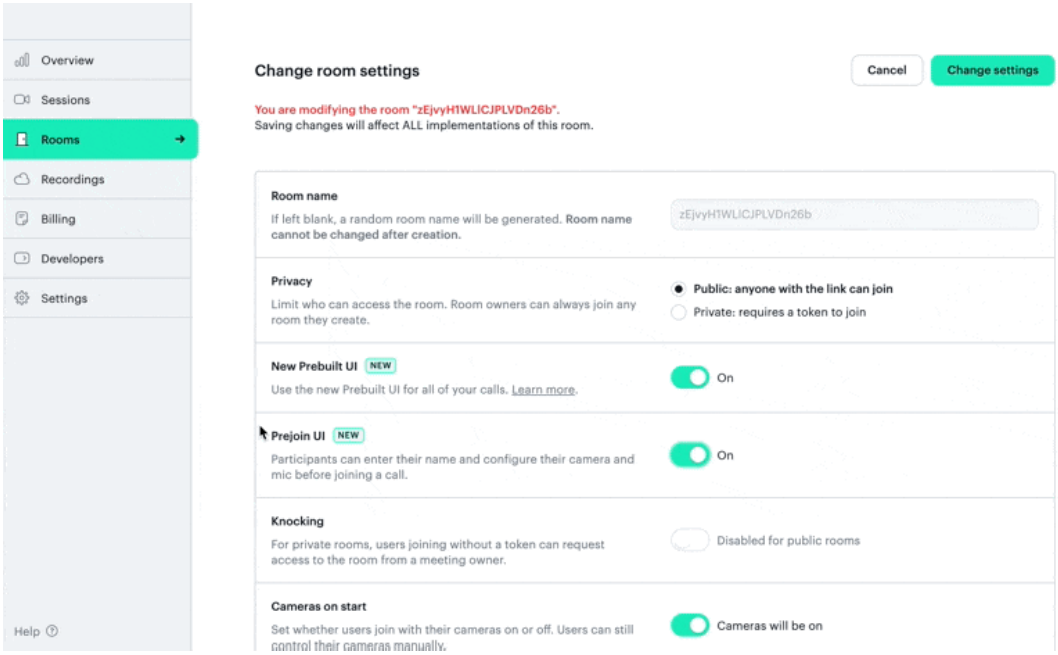


callFrame.join({ url });
```

Attach the callbacks created earlier to the call frame and join the call

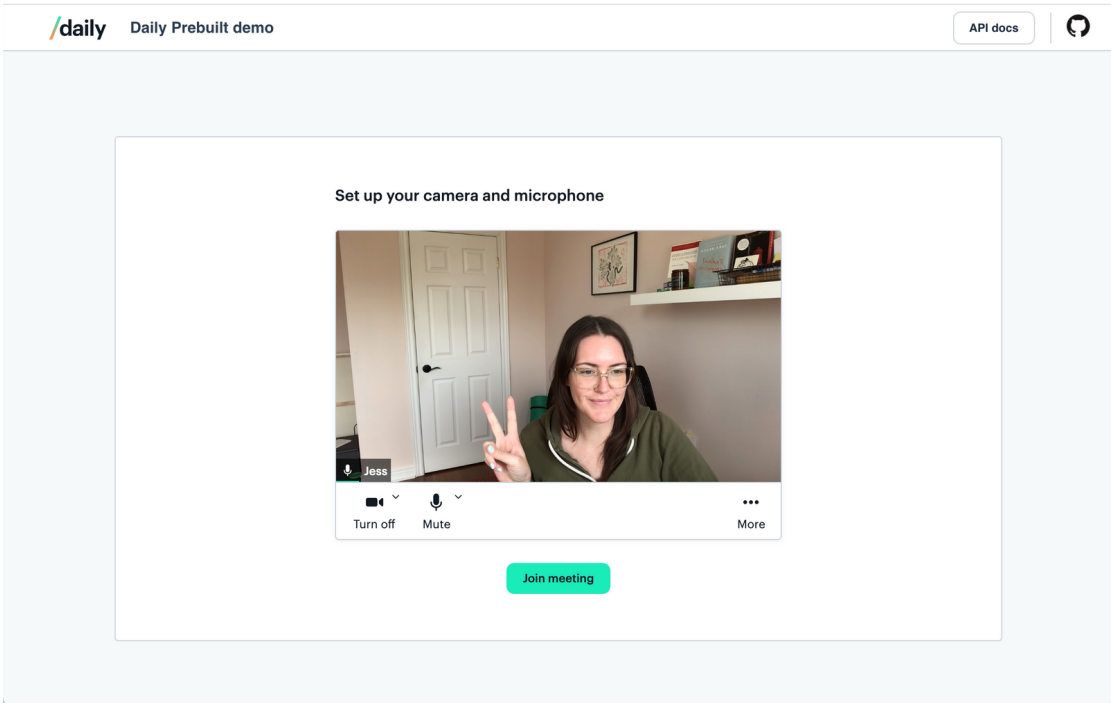
Once the `callFrame` exists, we can attach all the Daily event listeners to it with our callbacks created earlier, and join the call. To join, make sure you pass the Daily room URL, which is the value the user entered into the input.

After the `join` method is called, you should see two possible views depending on your room's `prejoin UI` settings.



Daily dashboard: Room settings

If you have the `prejoin UI` option enabled, you will see the lobby view. The `joining-meeting` event will get triggered, which will call the `goToLobby` callback that we set above.



Daily prejoin UI (a.k.a. the lobby... a.k.a. hair check)

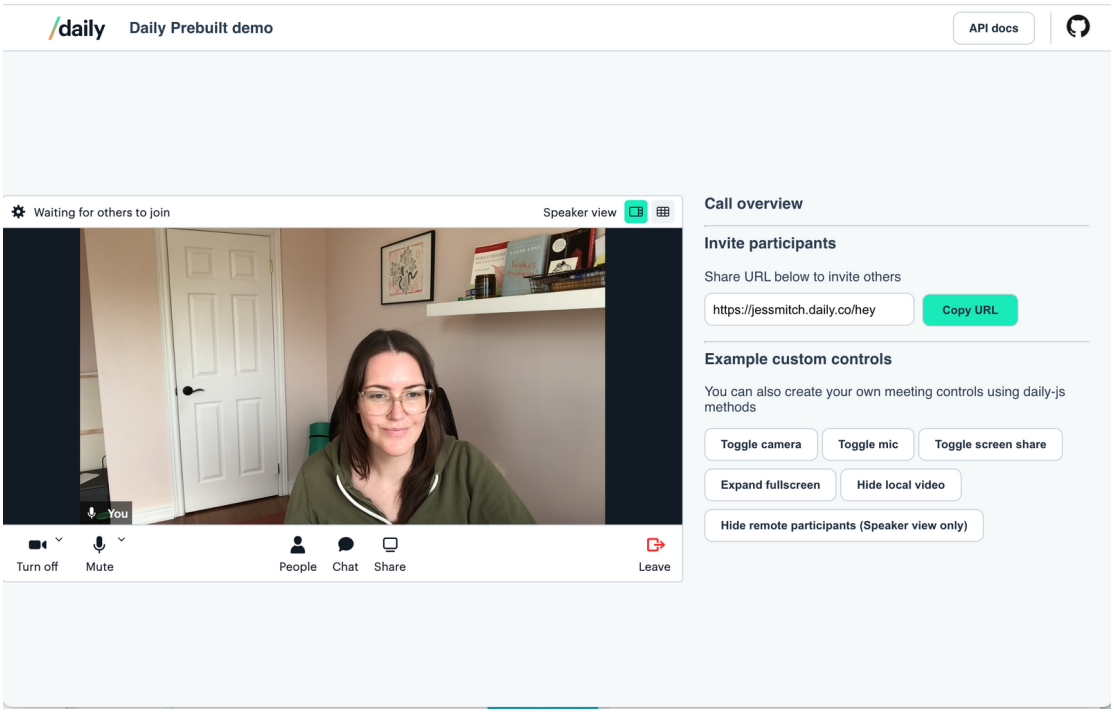
In the lobby view, you will no longer see the default view because the `status` value has changed to `lobby`. If we review our DOM elements, we can see the call container now shows because `status !== 'home'` (it equals `lobby` now). The controls do not show yet, though, because we're not officially in the call yet.

```
<div class="call-container" :class="{ hidden: status === 'home'
    <!-- The Daily Prebuilt iframe is embedded in the div below
    <div id="call" ref="callRef"></div>
    <!-- Only show the control panel if a call is live -->
    <controls
      v-if="status === 'call'"
      :roomUrl="roomUrl"
      :callFrame="callFrame"
    />
</div>
```

The second possible view, if you have the `prejoin UI` disabled for the room you're in, is seeing the call view. This means you are in the Daily call! 💪

The `joined-meeting` event would have been triggered, calling the

`goToCall` callback we set, which will update the `status` to be `call`. This status change will cause the controls to now show.



Controlling your in-call experience programmatically

One of the best things about Daily Prebuilt is that the hard parts of building video calls are done for you but there are still lots of options that can be configured or customized.

Once the `DailyIframe` instance (our video call iframe) has been created, you have access to dozens of instance methods to help you manage your call functionality.

For example, let's say you want to add a button to your app to leave a call. You can create a button that calls the `.leave()` instance method

on click.

To look at how some of these methods work, we can review how the `Controls` component is set up.

To start, let's see which props are passed to the `Controls` component where it's used in `Home`.

```
<controls
   v-if="status === 'call'"
   :roomUrl="roomUrl"
   :callFrame="callFrame"
/>
```

The `v-if` means the controls are only rendered if the `status` value is equal to `call`. This means it only shows when a person is live in a call in this demo.

The `roomUrl` prop is the URL the user submitted in the default home view.

The `callFrame` prop is the DailyIframe instance created for the call, which gives us access to all the instance methods.

Note: Not all instance methods are available for Daily Prebuilt. Refer to our documentation to know which ones can be used.

Now let's take a look at our `Controls` component and see how the HTML is structured:

```
<template>
 <div class="controls">
   <h2>Call overview</h2>
   <hr />
   <h3>Invite participants</h3>
   <label for="urlInput">Share URL below to invite others</label
   <div>
<!-- Room URL to copy and share -->
     <input type="text" id="urlInput" :value="roomUrl" />
     <button @click="copyUrl" class="teal">{{ copyButtonText }}<
```

```
    </div>
    <hr />
    <h3>Example custom controls</h3>
    <p>
      You can also create your own meeting controls using daily-j
    </p>
    <div>
      <button @click="toggleCamera">Toggle camera</button>
      <button @click="toggleMic">Toggle mic</button>
      <button @click="toggleScreenShare">Toggle screen share</but
      <button @click="expandFullscreen">Expand fullscreen</button
      <button @click="toggleLocalVideo">
        {{ localVideoText }} local video
      </button>
      <button @click="toggleRemoteParticipants">
        {{ remoteVideoText }} remote participants (Speaker view o
      </button>
      <button @click="leaveCall">
        Leave call
      </button>
    </div>
  </div>
</template>
```

We display the `roomUrl` prop in the input for the user to copy and share with others so they can join the call, too.

We also have eight buttons included in the control panel to programmatically interact with the `DailyIframe` instance. There interactions include:

- Turning the local camera on and off
- Turning the local microphone on an off
- Sharing the local call participant's screen
- Expanding Daily Prebuilt to be fullscreen
- Hiding and showing the local participant's tile in the call

- Hiding and showing the participants bar, which is where all the remote participants' tiles are locally while in speaker mode
- Leaving the call to go back to the default home view

Toggle your local camera programmatically

To understand how these work, let's review a couple, starting with

To understand how these work, let's review a couple, starting with toggling the local camera.

```
<button @click="toggleCamera">Toggle camera</button>
```

To turn the local camera on and off, the control panel button has the following click event attached to it:

```
toggleCamera() {
  this.callFrame.setLocalVideo(!this.callFrame.localVideo());
},
```

`this.callFrame` refers to the `callFrame` prop passed in the `Home` component, which gives us access to the `DailyIframe` instance. We can then call `.setLocalVideo()`, an instance method that accepts a boolean value.

The current status of the local camera can be accessed with the `.localVideo()` instance method, which will return whether the local camera is currently on or off. Since we want this method to toggle the current state, we can pass `.setLocalVideo()` whatever the inverse of the current state of the camera is with `!this.callFrame.localVideo()`.

So, if the camera is currently on, calling `this.callFrame.setLocalVideo(!this.callFrame.localVideo());` is the same as calling `this.callFrame.setLocalVideo(false);` to turn it off.

Go fullscreen with the click of a button ✨

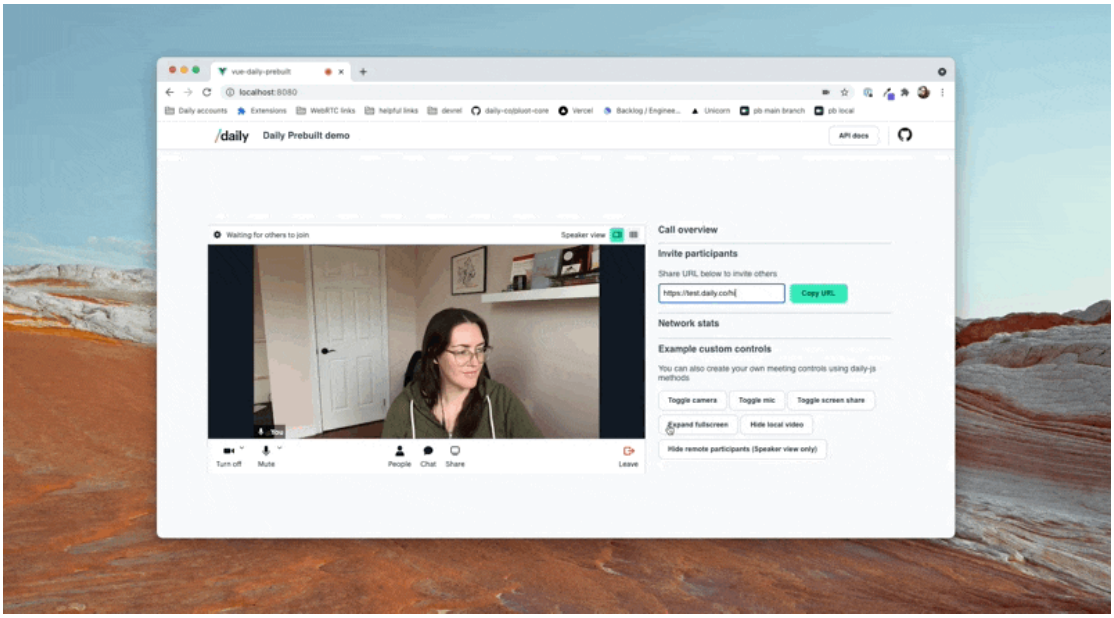
The other buttons in the control panel mostly work the same way. Let's take a look at one more example to see how to update your Daily Prebuilt calls programmatically.

The control panel includes a button to make the Daily Prebuilt iframe fullscreen:

```
<button @click="expandFullscreen">Expand
```

```
fullscreen</button>
```

The click handler on this button uses the `callFrame` prop to access the `DailyIframe` instance, which can then call the `requestFullscreen()` instance method.



Fullscreening Daily Prebuilt in the Vue demo app

And with one click, you're in fullscreen mode. It's really as simple as that! 🙌

Wrapping up

Now that you know how to embed Daily Prebuilt in a Vue app, you can add Daily video chat to any Vue projects you're building! Tag us on Twitter (@trydaily) to show us your projects. 😊

In terms of next steps, to learn how to customize your video apps even more, try updating your Daily Prebuilt colour theme.

Get updates from Daily

Don’t miss a thing! Subscribe to our monthly newsletter and we’ll keep you up-to-date with the latest product releases, tutorials, and more from Daily.

Email*

Country*

Join peerConnection - our video/WebRTC developer community.

By clicking submit below, you consent to allow Daily.co to store and process the personal information submitted above to provide you the content requested.

Submit

More in Code & Tutorials

Working with video call participants’ media tracks

10 Oct 2022 - 9 min read

Stream with HLS while recording to a custom S3 bucket using Daily

30 Aug 2022 - 5 min read

Tutorial: Integrate CometChat’s chat widget into a Daily video app for an enriched chat experience

22 Aug 2022 - 13 min read

See all 99 posts →



Collab Chrome Extension

Tutorial: Append Daily call transcription text with the Notion REST API

Creating a transcription feature in Daily Collab, our Daily video call Chrome extension demo



Jess Mitchell
2 Sep 2021 – 8 min read



Collab Chrome Extension

Tutorial: Build a custom Daily video call widget and embed it into any Notion page

How we added video and audio-only calls to the Daily Collab Chrome extension demo using Daily's custom APIs



Jess Mitchell
26 Aug 2021 – 11 min read