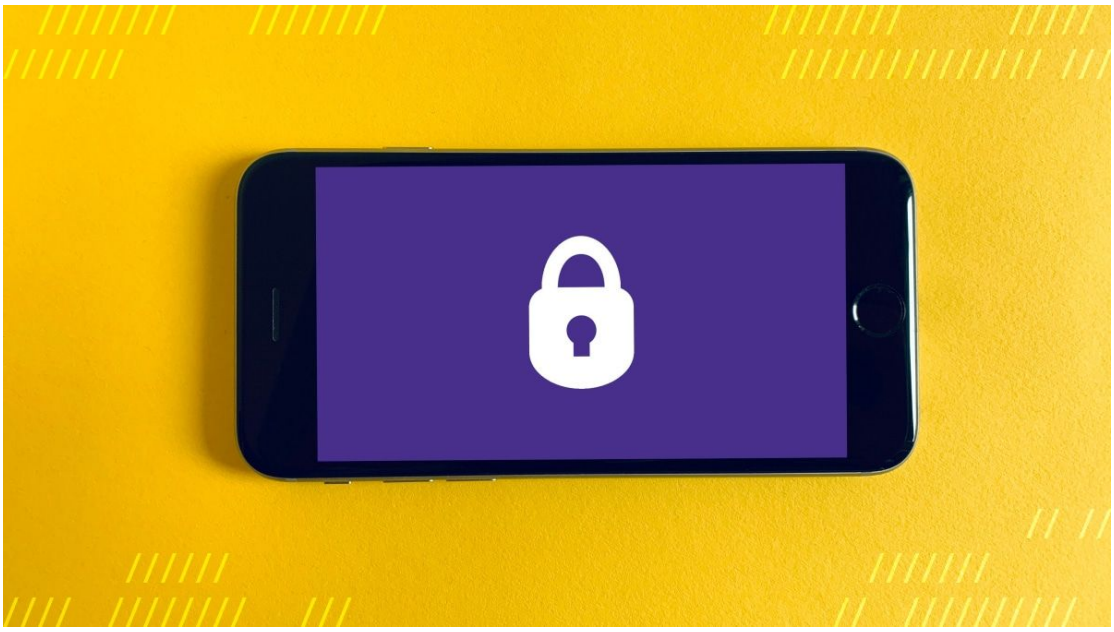
API Tutorial

Create admins in React apps with Daily meeting tokens

Validating hosts in Daily's React webinar app



Jess Mitchell
Published: 19 Mar 2021 (Updated: 4 Mar 2022) • 9 min read



> *This post is part of Daily's webinar blog series on how to build your own webinar platform using Daily Prebuilt and a custom chat widget.*

Daily's APIs are built to handle a variety of features and use cases related to hosting video calls. Managing who can join a call, as well as in-call permissions, are often major considerations for building a video

integration. At Daily, we understand privacy can be crucial for many meeting app requirements.

If you've been following our webinar series, you'll know that webinars are a great example of a type of video call that relies on managing privacy and permissions. For example, webinars will often have two levels of access for call participants: hosts and attendees. The host(s) can turn on their camera/microphone and speak to attendees. They can often share their screen to present important information, as well. Attendees, on the other hand, typically can watch the host's presentation but are not active participants; they usually can't turn on their cameras or microphones and mostly observe.

In the first two posts of our webinar series on how to build your own webinar app, we decided on our MVP feature set and embedded a Daily call in our React app using Daily's prebuilt UI.

Even after these first two steps, we're not quite at "webinar" status yet because we're not yet distinguishing between hosts (or "admins") and attendees.

In this tutorial, we'll address how to differentiate between these participant types using only Daily meeting tokens. That's right— no formal authentication required!



Who will find this tutorial useful?

- Anyone who has been following our webinar series up until this point will find adding meeting token validation helpful. This is required to get the full webinar experience (unless you'd rather add your own authentication, of course!)
- Anyone who wants to add meeting token validation to their app even if it's not a webinar. You could give validated participants access to more features or customize their UI experience, for example.
- Anyone looking to learn how to access query params in their React app to trigger state changes in their UI.

Dependency versions

The specifics of this tutorial are for a React app using React Router to handle routing.

We'll be using the following versions of `daily-js`, React and React Router. If you're behind or ahead of these versions, you may find some implementation details have changed.

```
"dependencies": {
  "@daily-co/daily-js": "^0.9.998",
```

```
  "react": "^17.0.1",
  "react-dom": "^17.0.1",
  "react-router-dom": "^5.2.0",
  ...
},
```

The webinar app's (simplified) dependency list

Let's make a plan

To add meeting token validation to our webinar app, we'll need to:

- Create meeting tokens to be used by our hosts. We'll use a cURL command for now and revisit simplifying token creation in a future tutorial.
- Include a valid meeting token as a query parameter when entering the app's URL.
- Access the token in our `WebinarCall` component and validate it to ensure it's a real Daily token for the Daily call being accessed.
- Set the participant as a host if they have a valid token and otherwise fall back to the "attendee" status.

By the end of this tutorial, you'll be able to set any participants trying to join your Daily call as either a host or an attendee.

Handling Daily API requests

Our webinar app is using Netlify to proxy Daily API requests for us, which means our webinar app can work without a formal back-end. To learn more about how to set this up, check out this previous post on managing Daily API requests or instantly deploy your own back-end server. Our full webinar demo app code is available to see exactly how we set it up too.

If you prefer to build your own back-end server, you can swap out our example endpoints for your own, as well.

Meeting tokens? What are you token

about?

Daily meeting tokens can be used in several ways but their overall purpose is to identify a call participant as "known" or "permitted". They can also be used to set a call participant as an `owner` in the call, which gives them extra access like being able to mute other call participants.

Tokens can be used to set a participant's username before joining the call, or to identify the participant as "validated". For example, in private Daily calls, users with meeting tokens do not need to "knock" to join the call. They can go right in! This makes tokens a useful and flexible tool for having more control over user experience in a video call you're hosting.

Creating Daily meeting tokens

The quickest way to create a meeting token is to open up a new terminal window and run a cURL command. As a first step, though, make sure you've signed up for a (free) Daily account, create a new room, and grab your developer API key.

Once you have a Daily room created, be sure to make note of the room's name. You can create a token for it by running the following command. To make sure this works, don't forget to add your (secret) developer API key where it says `<API KEY>` below. You'll also want to use the name of Daily room you just created and set the `user_name` to whatever you want.

For the token to work in this example, `is_owner` will also need to be set to `true`. This is because we're only giving webinar hosting access to room "owners".

```
curl --request POST \
  --url https://api.daily.co/v1/meeting-tokens \
  --header 'Authorization: Bearer <API KEY>' \
  --header 'Content-Type: application/json' \
  --data '{"properties":{"room_name":"webinar","user_name":"Admi
```

If all went as planned, you should see a meeting token like the one

below returned in the response. This is what we'll use in our webinar app to make someone a webinar host.

```
{
"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJvIjp0cnVlLCJ1I
}
```

Accessing query params in React

At this point we've created a basic React app and embedded the Daily prebuilt UI in it. We also have a Daily room name and a meeting token for that room we'll be using in our webinar.

Next we need to add React Router to our app. In your terminal from your app's root directory, run:

```
yarn add react-router-dom
```

Once React Router has been added as a dependency, we'll need to add routing to our `App` component. To keep things simple we'll have two routes:

- A route for the home screen if a Daily room isn't specified.
- A route for the webinar call view if the room is specified.

It should look something like this:

```
import React from "react";
import styled from "styled-components";
import { BrowserRouter as Router, Switch, Route } from "react-ro
import WebinarCall from "../views/WebinarCall";
import Home from "../views/Home";
import Header from "./Header";

const App = () => {
  return (
    <Router>
        <Body>
          <Header />
          <Switch>
            <Route path="/:roomName">
```

```
            <WebinarCall />
          </Route>
          <Route path="/">
            <Home />
          </Route>
        </Switch>
      </Body>
    </Router>
  );
};
```

App component with routing

Above, we're wrapping our App component contents in the `<Router>` component provided by React Router. Then to specify our specific app routes, we can use the `<Route>` component with the `path` prop. Any route that matches the path `/` will go to our `Home` component (or home page). Any path other than `/` will be assumed to be a Daily room name and will go to our `WebinarCall` component.

Let's say we're running our webinar app locally (`yarn start`) on port `3000` . If we visit `http://localhost:3000/` , it will go to our homepage. If we go to `http://localhost:3000/test-room` , it will route to our webinar call page and assume `test-room` is the Daily room we're trying to use.

Note: We'll take a deeper look at validating the room name. For now, let's assume the room name provided in the URL is a valid Daily room on your account.

Using our token as a query parameter

Now that we have our routing sorted out, we can start using the token created above.

As a reminder, regular attendees (non-hosts) will not provide a token and will visit the default URL (`http://localhost:3000/test-room` or `www.your-webinar-domain.com/room-name` if it's deployed).

Hosts, on the other hand, will pass their token as a query parameter like

this:

`http://localhost:3000/test-room?t=<token>`

HBwIiwibyI6dHJ1ZSwiZCI6IjQzZDVlYWI4LWY0YjctNGY1MS05ZDY1LTk2ODdlMjhiZGI0ZSIsImlhdCI6MT

Reacting to tokens

In our `WebinarCall` component, which is rendered when a room name is included in the path, we now need to access the token from the URL path. To do this, we can use the `useLocation` function provided by React Router in our `WebinarCall` component.

(To see the full example, refer to our `WebinarCall` component.)

```
import React, { useState, useEffect } from "react";
import DailyIframe from "@daily-co/daily-js";
import { useLocation } from "react-router-dom";

const roomName = 'test-room'; // we'll make this dynamic later

const WebinarCall = () => {
 ...
 const [roomInfo, setRoomInfo] = useState(null);
 const { search } = useLocation();

 useEffect(() => {
   if (search && search.match(/^[?t=*+]/)) {
     // remove the first few characters to isolate the token
     const token = search.replace("?t=", "");

     // validate the token from the URL if supplied
     fetch(`https://webinar-demo.netlify.app//api/meeting-tokens
       .then((res) => res.json())
       .then((res) => {
         if (res.is_owner && res.room_name === roomName) {
           // add admin settings
           setRoomInfo({
             token,
             username: res.user_name,
             accountType: 'admin',
           });
           return;
```

```
          }
          // else: handle the error
        })
        .catch((err) => { /* handle the error */ });
    } else {
      // set the participant to a regular attendee
      setRoomInfo({
        token: null,
        username: null,
        accountType: "participant",
      });
    }
  }, [ /* Add dependencies based on full code*/]);
```

Let's step through what we're doing in the code above:

1. Include the necessary dependencies with special attention to `useLocation` from `react-dom-router`.
2. For now, assume the room name in our path is `test-room` (e.g. `http://localhost:3000/test-room?t=<token>`)
3. In our component, access the query parameters from our local URL using `const { search } = useLocation();`
4. In the `useEffect`, check the value of `search`. If it includes a `t` (or token) param, it means someone is trying to join as a host and we will need to validate the token. If there is no `t` (or token) param, the person is trying to join as a regular attendee.
5. To validate, use the endpoint hosted on Netlify mentioned above and pass the token included in the URL. We then use the response to make sure `is_owner` is true and the room name matches the same room we're trying to join before actually setting the participant to be a host.

Psst... If using the Netlify endpoint is at all confusing, you can simplify the fetch request with the code below. Just be sure not to include your API key in any commits! 🙈

```
fetch('https://api.daily.co/v1/meeting-tokens/${token}, {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    Authorization: 'Bearer ' + API_KEY,
```

```
  },
})
.then((res) => res.json())
.then((res) => /* continue with the example above */)
```

Once we know if the token is valid or not, we can update our UI to reflect the level of access the participant should have. In our webinar app, we let hosts/admins go straight into the Daily call by updating the `currentView` state value from `waiting` to `call`. That means as soon as they've been validated they're in the call.

For regular attendees, we show them a form to enter their name before joining. This user name will be needed in a future feature: live chat.

Status: Validated 🔑

Now that we can access tokens provided by participants and validate them before letting the participant join, we have officially reached Webinar status! 🎉 There are still a lot of features we can (and will!) add but, for now, we have our basics down.

Not to toot our own horn, but we also managed to distinguish between hosts and attendees without having to build a back-end or any formal authentication, and that's pretty neat!



Token it to the next level

In upcoming posts for this webinar series, we'll continue to build on what we've created so far.

This will include:

- Validating the room name in the URL path, which we currently assume is valid.
- Adding a chat box that lets hosts send direct messages to single participants or broadcast messages to everyone.
- Building a simple admin form to create tokens so you don't have to use cURL commands for the rest of your webinar-hosting career.
- And more!

To learn more about meeting tokens, be sure to read our previous posts.

And, as always, let us know how we can help! 🙌

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 API Tutorial

Implementing cursor sharing with Daily's video call API

24 Oct 2022 - 11 min read

Add hand-raising to your custom video app with the Daily user data API

12 Sep 2022 - 12 min read

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

30 Aug 2022 - 5 min read

See all 82 posts →



API Tutorial

Update call participants' displayed names with the Daily API

We're noticing a pretty popular trend here at Daily: people are joining more and more calls online with people they haven't met yet, attending webinars, joining audio communities,



Kimberlee Johnson
23 Mar 2021 – 5 min read



Tools & Tips

How to load an external HTML file in an iOS app

Trying to load an external HTML file in an iOS app? We explain the easiest way to do it so that you can use getUserMedia in WKWebView.



Phil Miller
18 Mar 2021 – 2 min read