Fitness use case

Fitness app use case: Adding an “incognito mode” for participants

Create more privacy for participants by letting them toggle who can see them



Jess Mitchell

Published: 2 May 2022 (Updated: 2 May 2022) • 8 min read



In our previous fitness use case post, we looked at how to record only the instructor of a class to create a better sense of privacy for class participants. One important factor in growing app usage is ensuring your app users feel comfortable joining a call because – let’s be honest – participating in fitness classes online can make anyone feel a bit vulnerable.

In today's post, we'll look at how to add this sense of privacy to your participants' in-class experience, too. We'll add a toggle input in the fitness demo app UI for students to turn on "incognito mode", which means only the instructor will be able to see their video. You could change this so that ***no one*** can see their video, but we find instructors usually need to be able to see what class participants are doing.

Since this post is part of our fitness use case series, we'll be working with the same demo app as previous posts; ***however*** we'll try to discuss this a bit more generally than what is literally happening in the code. Since this feature is more of an app-level feature than a Daily feature, the specifics of your own code will depend a lot on how you've implemented your app. Because of this, we find it's better to understand ***what*** we're trying to do than worry too much about ***how*** we're doing it.

In the sample code below, we'll be working with a custom useSharedState hook described in a previous post about building client-side persistent chat in a Daily app. We need newcomers to the class to know all other participants' incognito status, so this information needs to persist beyond just their local component state. The `useSharedState` hook allows us to do just that, but sharing a data object with everyone in the call whenever it is updated or someone new joins.

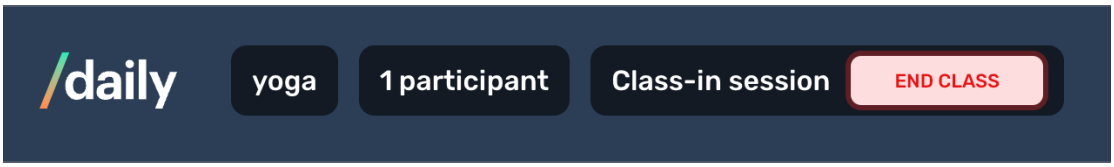
Working with the fitness demo app

Since the sample code is from an app built with React (specifically, Next.js) and TypeScript, some background knowledge of working with those will be useful. That said, this functionality could be added to any front-end set up using Daily.

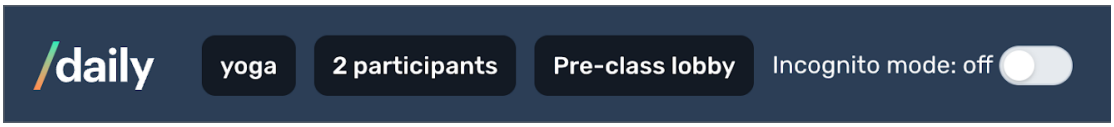
As a final note before getting started, if you want to try running the fitness demo locally, check out our local set up instructions from the previous post in this series. The demo app README is also a great resource for getting set up.

Defining our feature

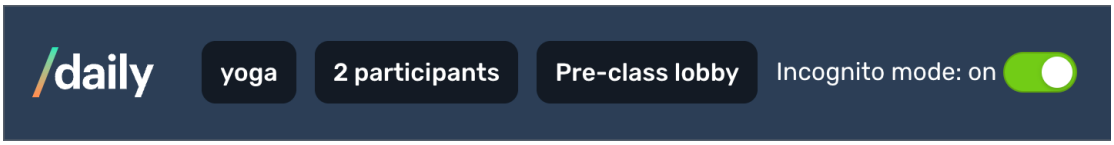
As mentioned, we want to add an input that class participants can toggle to turn incognito mode on and off. We'll only provide it if the participant is not the instructor (i.e. not the meeting "owner", in Daily terms.)



Owner view (no incognito toggle)



Participant view (incognito toggle off)



Participant view (incognito toggle on)

Now for the code:

```
{!localParticipant?.isOwner && (
  <div className="incognito-button">
    <label htmlFor="incognitoMode">
      Incognito mode: {isIncognito ? 'on' : 'off'}
    </label>
    <BooleanInput
      id="incognitoMode"
      value={isIncognito}
      onChange={toggleIncognito}
    />
  </div>
)}
```

We're adding this to Header.js

In the fitness demo's `Header` component, we can add an input that is

conditionally shown if the local participant is ***not*** the owner.

Note: If you're using Daily React Hooks, there's a handy `localParticipant` value available from the useLocalParticipant hook that gives you access the the local participant's call information.✨

You'll notice we are setting the value of the `BooleanInput` component to `isIncognito` and updating it `onChange` with the function `toggleIncognito`. (In case you're wondering, `BooleanInput` is a React component that is basically just an `<input>` element with `type=checkbox` and some fancy styling).

So, how do `isIncognito` and `toggleIncognito` work? That's where things get interesting, so let's take a look!

Creating a shared state with the `useSharedState` hook

The `useSharedState` hook is a custom React hook that can share local state among call participants. When someone new joins a call, they'll immediately get existing shared state data from another participant. Additionally, if the shared state updates, everyone will get that new shared state data value.

We originally wrote this hook to handle persistent chat messaging in Daily Prebuilt so that we wouldn't have to add a backend solution for chat history. This custom hook can be used for sharing any state, though – not just chat messages.

Note: For an overview on how the `useSharedState` hook works, check out our persistent chat history post. In short, it uses Daily's `sendAppMessage()` method to share data among call participants.

Since we're already using `useSharedState` in our `ClassStateProvider`, we can start by updating the `initialValues`

object to include an empty array for participants have incognito mode enabled. The idea here is that any time someone turns on incognito mode, we'll add them to this array, which will get shared with everyone in the call.

```
import { useSharedState } from '@custom/shared/hooks/useSharedSt
…
export const ClassStateProvider = ({ children }) => {
…
const { sharedState, setSharedState } = useSharedState({
   initialValues: { type: PRE_CLASS_LOBBY, incognitoIds: [] },
 });
…
 return (
   <ClassStateContext.Provider
     value={{ classType, setClassType, sharedState, setSharedSta
   >
     {children}
   </ClassStateContext.Provider>
 );
};
```

ClassStateProvider.js

We'll also make the `sharedState` and `setSharedState` values available to the `children` components, which includes the `Header` component we've already added our toggle input to.

Let's go back to our `Header` component now. We'll import these two new values from the `ClassStateProvider` , like so:

```
import {
 useClassState,
…
} from '../../contexts/ClassStateProvider';

export const Header = () => {
…
const { sharedState, setSharedState } = useClassState();
```

We're adding this to Header.js

Next, we can determine if the local participant is incognito if their

session ID is in the `sharedState.incognitoIds` array.

```
const isIncognito = useMemo(() => {
   return sharedState
       .incognitoIds?
       .indexOf(localParticipant?.sessionId) > 0;
 }, [sharedState.incognitoIds, localParticipant?.sessionId]);
```

Next, we can write a `toggleIncognito` function to toggle the `isIncognito` value by either adding or removing the local participant's session ID to the `sharedSate.incognitoIds` array.

First, let's write a little helper function to determine if the ID is present in the array:

```
const addOrRemove = (arr, item) =>
 arr.includes(item) ? arr.filter((i) => i !== item) : [...arr, i
```

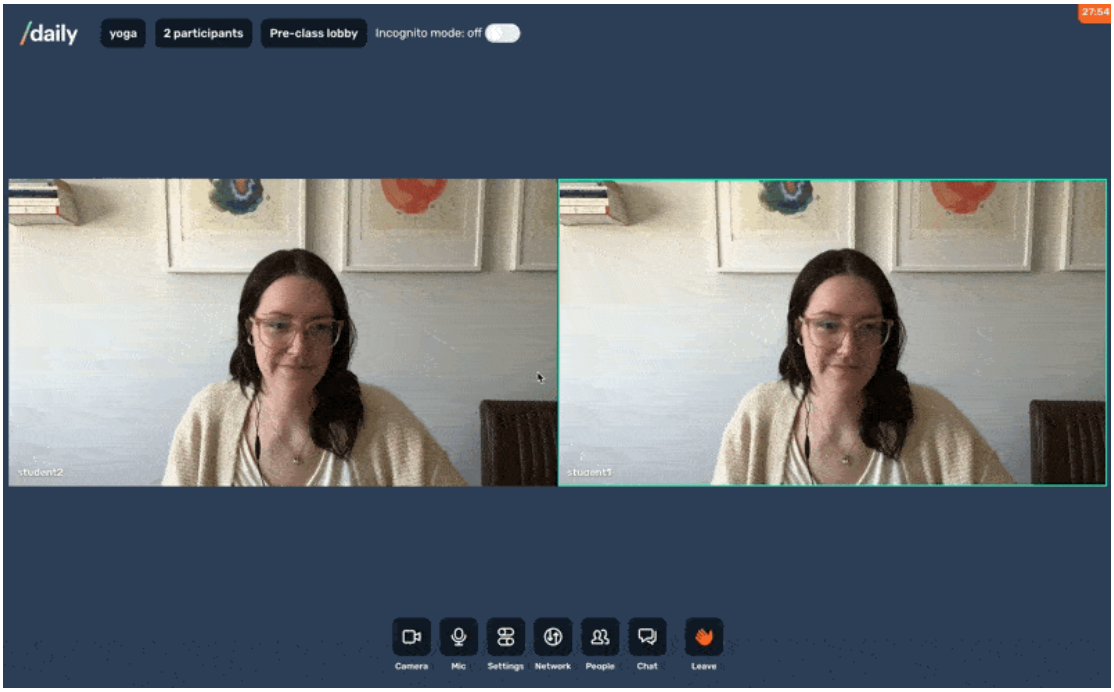
Then, using this helper function, we update our `sharedState` values using `setSharedState` in our new `toggleIncognito` function:

```
const toggleIncognito = useCallback(
  () => {
   setSharedState((values) => {
    // add if it's not in the list, remove if it is
    const newArr = addOrRemove(
      values.incognitoIds,
      localParticipant.sessionId
    );
    return {
      ...values,
      incognitoIds: newArr,
    };
  });

  },
  [localParticipant, setSharedState]
);
```

Now we have both values we needed to figure out for our toggle input. 💫

```
<BooleanInput
    id="incognitoMode"
    value={isIncognito}
    onChange={toggleIncognito}
/>
```



Toggling incognito mode on and off

Updating our video display

A toggle input turning off and on isn't very exciting if it doesn't actually, well, do anything. In the gif above we can see the input toggling but our video grid stays the same. 🙈

Now we need to actually update our grid based on the

`sharedState.incognitoIds` array.

This is the part of the tutorial that will completely depend on how your app is set up and whether you have a grid view, speaker view, or any other type of video layout.

Our goal here – generally speaking – is to write a condition that uses the following logic:

- If the local participant is the class instructor, the incognito state will not affect their view since we've decided instructors should see everyone.
- If the local participant is ***not*** the class instructor, they will ***not*** see video tiles for participants who have incognito mode turned on. You may decide that local participants should be able to see themselves even if they're incognito but we won't do this for now. The UX design gets a little more complicated because we want them to be certain no one else can see them and that's easier to show from a UX perspective by just removing their tile.

Let's see how we can accomplish this in the fitness demo.

Updating our pre-class grid view

Before a class begins in the fitness demo app, everyone is visible in grid view (`GridView`).

We can update the `visibleParticipants` variable in our `GridView` component to also consider the `sharedState.incognitoIds` array. First, let's figure out who should be visible (i.e. not incognito):

```
const nonIncognitoParticipants = useMemo(
  () =>
    localParticipant.isOwner ? participants : participants.filte
      (p) => !sharedState.incognitoIds?.includes(p.sessionId)
    ),
  [sharedState.incognitoIds, participants, localParticipant]
);
```

Adding this to our GridView.js component

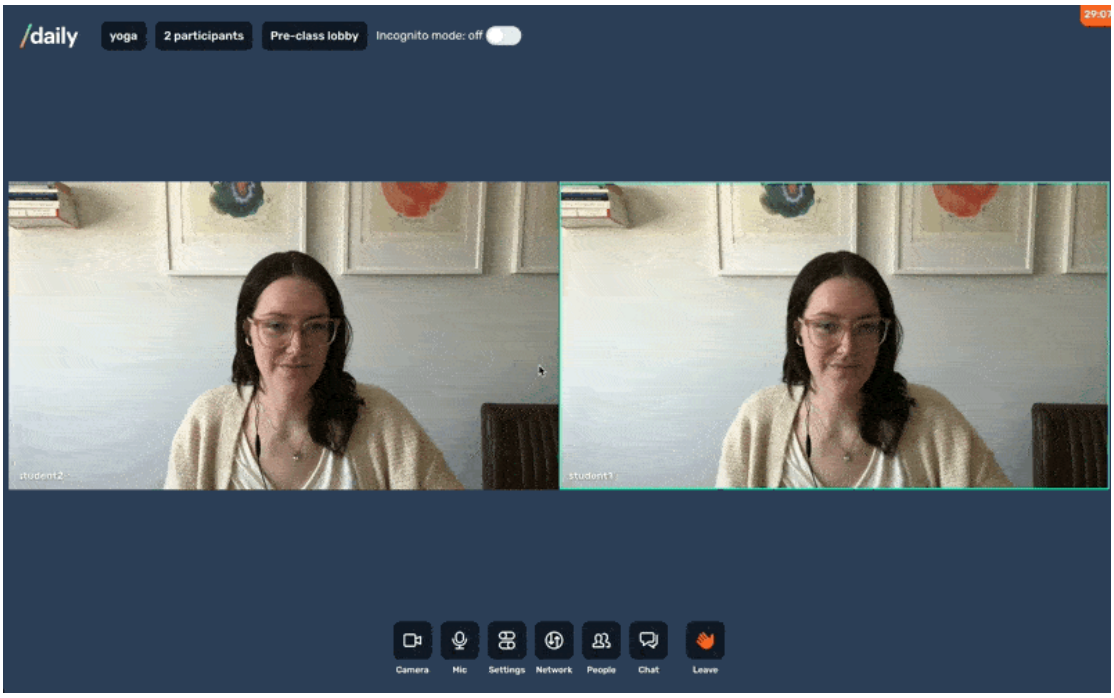
If the local participant is a class owner, they'll see everyone and if they aren't, they'll see a filtered list.

Next, since this demo app has pagination in grid view, the actual `visibleParticipants` will be based on the current page. We won't worry about this too much since – again – it's an app-level detail.

```
const visibleParticipants = useMemo(
  () =>
    participants.length - page * pageSize > 0
      ? nonIncognitoParticipants.slice((page - 1) * pageSize, pa
      : nonIncognitoParticipants.slice(-pageSize),
  [page, pageSize, nonIncognitoParticipants]
);
```

Updating this value in GridView.js to use `nonIncognitoParticipants`

Now that we've got our filtering applied, our incognito mode toggling will actually update our local view!

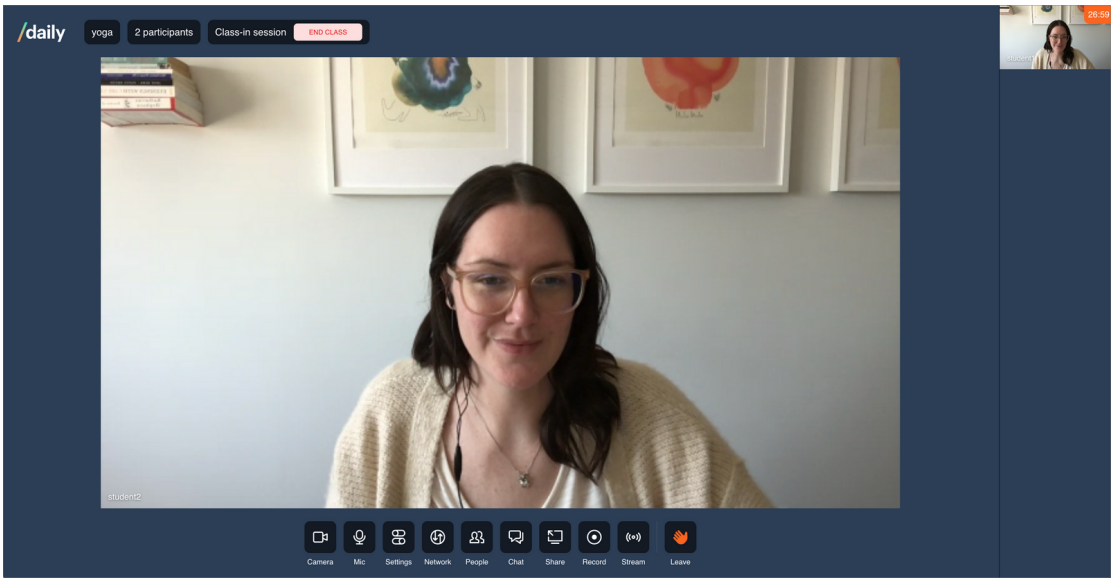


Incognito mode toggling the local participant's view for other students

When the incognito mode is on, there are only remote participants, and when it's off, the local participant is also visible.

Hold on, what about speaker view?

Okay, so there's another step before we wrap up. The in-class view actually uses a speaker view where the current speaker is placed in the large video tile. To properly apply incognito mode we'd need to update that view as well.



Fitness demo with speaker view on

However, the speaker view updates get a little more complex because there's a lot more logic related to the layout, including who gets placed where, and virtual scrolling for the participant bar on the right. We also probably don't want the current speaker in the large tile; instead, it would likely make more sense to keep the instructor in the large tile and update the participant bar (i.e. the class participants) based on who is incognito.

To avoid making this tutorial get overly complicated in those details, we're not going to review how to make all those updates. Ultimately, you need to apply the same logic we applied to grid view:

- If the local participant is an instructor (i.e. owner), show everyone.
- If they're not an instructor, filter the visible participants through the `incognitoIds` array.

After that, all the smaller UX details for how you'd like everyone placed in your layout is up to you! (As always, let us know if you have any questions, though.)

Wrapping up

We hope this introduction to one way of giving your video call participants more privacy helps get your creative juices flowing. Building a video call app is only half the battle – after that, you have to make sure people actually want to use it, and being empathetic to user experience is a big part of that.

To learn more about building a fitness class app, be sure to read our fitness series. To learn more about building more complex but still performant video call layouts, read our large meeting series.

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 Fitness use case

Auto-record calls when the instructor joins the class

31 Mar 2022 - 7 min read

Mute fitness class participants in a video chat app when the instructor joins

24 Mar 2022 - 6 min read

Daily use cases: Building a real-time video fitness class app with Daily

10 Mar 2022 - 4 min read

See all 3 posts →



Daily React Hooks

Build a custom video app with React and the Daily React Hooks library (Part 1)

Use daily-js and the Daily React Hooks library to create a custom video application from scratch



Nienke Dekker
9 May 2022 – 14 min read



Electron

Extending a Daily Prebuilt Electron app with custom virtual backgrounds

How to add a virtual backgrounds feature to an Electron Daily Prebuilt application



Liza Shulyayeva
28 Apr 2022 – 13 min read