Svelte

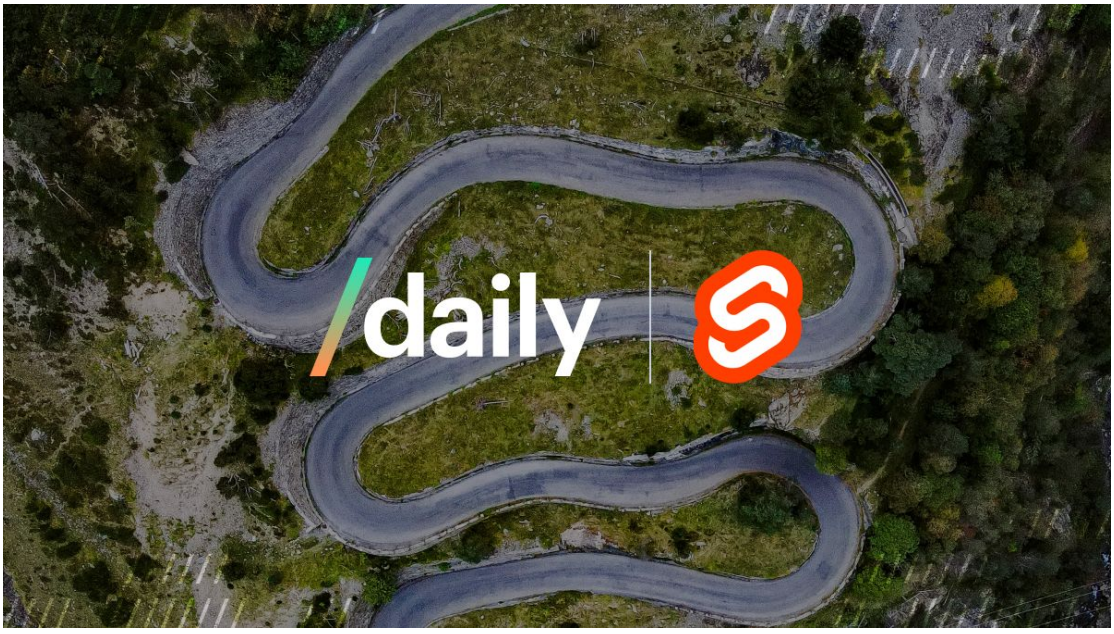
Build a custom video call experience with SvelteKit and Daily (Part 1)

Use SvelteKit's filesystem-based routing to create new Daily rooms on the fly



Jess Mitchell

Published: 10 Feb 2022 (Updated: 10 Feb 2022) • 9 min read



> *This is post one in a tutorial series on how to build a custom Daily app with SvelteKit.*

At Daily, we know how important it is to build flexible APIs. We intentionally build our APIs to allow for a variety of video call customizations.

In one of our previous blog posts, we looked at how to embed Daily Prebuilt – Daily's video chat interface – into a basic Svelte app.

In cases where you're looking to build a more custom video experience, using Daily's call object can provide much more flexibility.

In today's tutorial, we'll kick off a new Daily SvelteKit series by setting up our routing in a SvelteKit demo app.

By the end of this post, you'll be able to submit a form from the home screen to create and join a Daily room. Once the room is created from the home screen, we'll navigate to a path that corresponds with the Daily room name.

Svelte vs. SvelteKit: What's the difference?

Svelte is an open-source front-end component framework that can be used as an alternative to other front-end frameworks, like React or Vue.

SvelteKit, on the other hand, is a full-stack app framework (or "meta framework") built on top of Svelte. It is built to handle application routing and custom API routes, in addition to the front-end components offered by your standard Svelte project.

💡 ***Note: If you're familiar with React, Svelte is to SvelteKit what React is to Next.js.***

For this project, we are using SvelteKit to take advantage of two features that will simplify the demo app we're building:

1. The file-based routing system to navigate between our home screen and Daily call screen.
2. The custom route-based API endpoints we can build to create rooms on the fly prior to navigating away from the home screen.

Building our “create room” endpoint directly in this project means we won’t need to build a separate custom API to be used in our Svelte components, which is great!

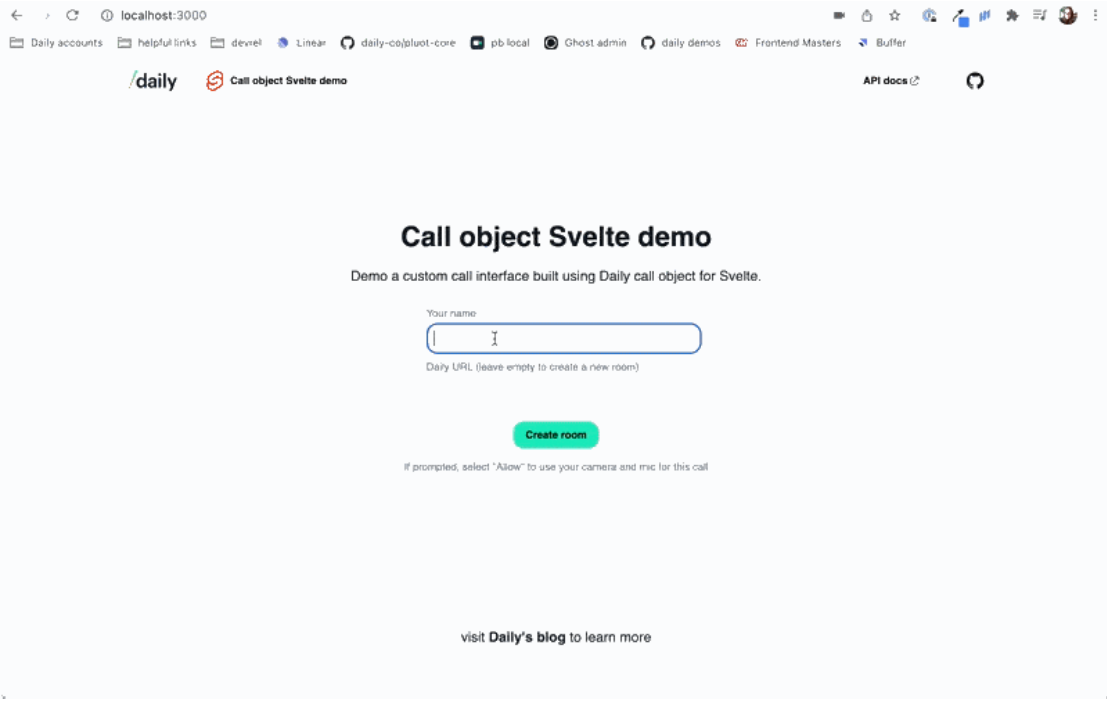
⚠️ ***Note: SvelteKit has not hit v1 yet, so keep that in mind while making framework decisions.***

Tutorial goals

In today’s tutorial, we’ll only be looking at how to use Daily’s REST API in a SvelteKit project to create new rooms.

We will:

- Create our home page with a form to create and join a call
- Create an in-call page that will eventually contain all our Daily video call UI
- Write a custom endpoint that creates the Daily room
- Set up our navigation to navigate to the new room’s URL once it’s been created



Submitting the room form from the home screen

Getting started

The completed version of this custom Daily call project is publicly available on Github. To run this project locally, first clone the repo.

Next, you will need to rename `env.example` to `.env.local` and add your Daily API key and Daily domain to it. (Visit the project's README for more information on where to retrieve these values.)

Lastly, from the project's root directory, run:

```
npm install
npm run dev
```

To view the project, open `http://localhost:3000` in the browser of your choice.

Planning our file structure

Where we add our files in this project is extremely important, given that the routing is based on our file structure in SvelteKit.

Let's first determine where we'll need to have files to cover both our app pages and custom endpoint.

```
Project_directory
|_  src
   |_  lib // UI components
      |_  call
      |_  forms
          |_ RoomForm.svelte
      |_  header
   |_  routes
      |_  room
         |_  [roomId].svelte // in-call page
         |_  index.json.js // room endpoints (Daily REST API use
      |_  index.svelte // home page
   |_  app.html // where our app will get injected
   |_ store.js
```

A couple things to note before we dive into this:

1. The `lib` directory is where we'll put all our Svelte UI components. The files in every subdirectory aren't listed since we're not discussing most of the UI components just yet.
2. There are many other files in this repo, but these are the ones we'll focus on today.

You may have noticed there are two types of file extensions in our `routes` directory: `.svelte` and `.json.js`. These represent the two types of routing files we'll use: our pages (`.svelte`) and our endpoints (`.json.js`).

Let's start by taking a look at the home page: `/routes/index.svelte`.

Building a home page view

When you visit the app's base URL (`http://localhost:3000` for now), it will render our `/routes` directory's `index.svelte` file.

This file is actually quite concise:

```
<script>
   import RoomForm from '$lib/forms/RoomForm.svelte';
```

```
</script>

<svelte:head>
   <title>Daily Svelte demo</title>
</svelte:head>

<section>
   <h1>Call object Svelte demo</h1>
   <p>Demo a custom call interface built using Daily call object

   <!-- Provide a form to enter name and optional Daily URL -->
   <RoomForm />
   <p class="subtext">If prompted, select "Allow" to use your ca
</section>
```

index.svelte

In `/routes/index.svelte`, we import our `RoomForm` component from the `lib` directory (plus a header and some text).

Setting up our `RoomForm`

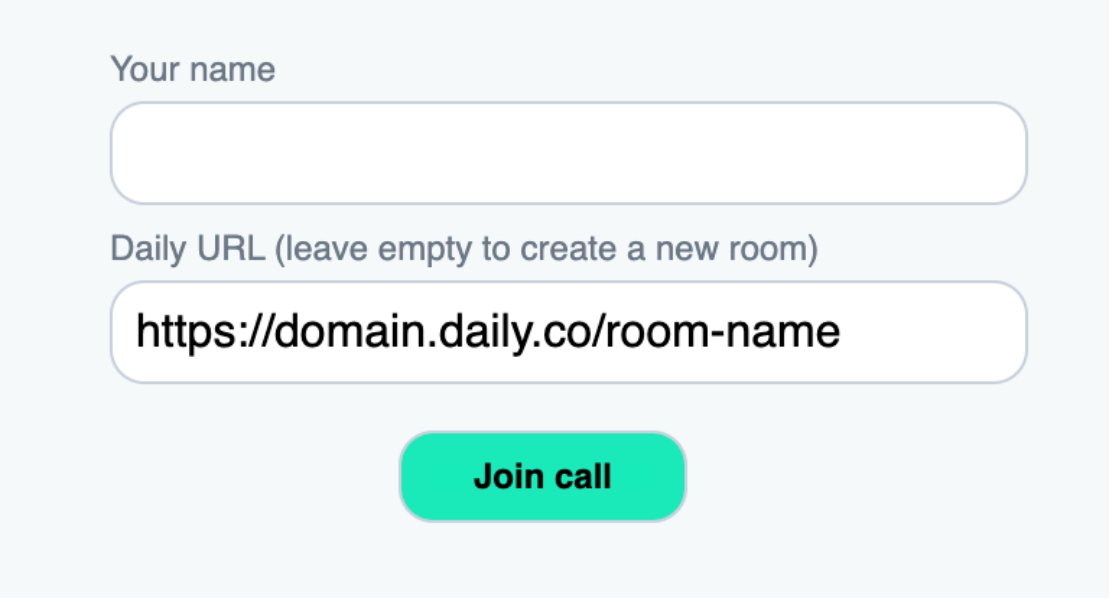
Before moving on to our custom endpoints, let's take a quick look at the `RoomForm` component so we know how it triggers a navigation change.

```
<script>
   import { username } from '../../store.js';
   import { goto } from '$app/navigation';

   …
   let dailyUrl = '';
   let dailyName = '';
   …
</script>

<form on:submit|preventDefault={submitForm}>
   <label for="name">Your name</label>
   <input id="name" type="text" bind:value={dailyName} required
   <label for="url">Daily URL (leave empty to create a new room)
   <input id="url" type="text" bind:value={dailyUrl} />
   <input type="submit" value={!dailyUrl ? 'Create room' : 'Join
</form>
```

RoomForm.svelte



Room form UI on home screen

The form itself has two inputs:

- The local user's name
- The Daily room URL they want to join

Each input has its value bound to a variable declared at the top of the file via Svelte's `bind` attribute (e.g. `bind:value={dailyName}` .) This means we can access the input values via the associated variables (`dailyName` and `dailyUrl`).

The form can handle two possible values for the URL input:

- An existing Daily room URL the local user wants to join
- An empty value. We'll assume this means they want to create and join a new Daily room

In other words, if the local user enters a URL, we'll use it. Otherwise, we'll need to create a new Daily room via the Daily REST API.

Submitting the `RoomForm`

Now that we have the form that lets us join a room, let's see what actually happens when you submit it.

As mentioned, there are two possible outcomes from submitting the

form, depending on whether the local user wants to create a new Daily room or not.

We'll start with if they already have a room they want to join:

```
async function submitForm() {
        // Set Daily name in store for future use
        username.set(dailyName);
        …

        /**
         * If a Daily URL has been included, use it.
         * (We're trusting it's valid at this point!)
         */
        if (dailyUrl) {
            …
            const roomName = dailyUrl.split('/').at(-1);
            goto(`/room/${roomName}`);
            return;
        }
        …
```

RoomForm.svelte

`onSubmit` gets called – unsurprisingly – when the form is submitted. In it, we set the name (`dailyName`) provided by the user in our app's store like so:

```
username.set(dailyName);
```

Note: Look at `store.js` to see how the `username` is initialized in our app's store.

Once the username is set in the store, we can access it from other components in the app. This will help when we're initializing the call with `daily-js` .

Next, we see if there is a Daily room URL provided by the form. If there is, we get the room name from the URL and navigate to that page via Svelte's `goto` function.

```
const roomName = dailyUrl.split('/').at(-1);
goto(`/room/${roomName}`);
```

Since Daily room URLs use the following format, we can assume the last value in the URL is the room name:
`https://<your-daily-domain>.daily.co/<room-name>`

Once the app navigation has updated, you will be viewing `http://localhost:3000/room/[room-name]`. This route maps to our `/routes` directory like so: `/routes/room/[roomId].svelte`.

Before we move on to that view, let's first cover what happens if a URL is ***not*** provided in the form.

Creating a new Daily room via a SvelteKit routing endpoint

If a room URL is not provided, we'll continue in `submitForm`:

```
async function submitForm() {
   …
    if (dailyUrl) {
        …
        return;
    }

    /**
     * If there isn't a Daily URL, we can create a new
     * room with a random name
     */
    const submit = await fetch('/room.json', {

        method: 'POST'
    });
    const data = await submit.json();

    if (data.success && data?.room?.name) {
        goto(`/room/${data.room.name}`);
        dailyErrorMessage.set('');
    } else if (data.status === '400') {
        dailyErrorMessage.set('bad request');
    } else if (data.status === '500') {
        dailyErrorMessage.set('server error :|');
    } else {
```

```
            dailyErrorMessage.set('Oops, something went wrong!');
        }

    }
```

Here, we make a `POST` request to `/room.json`. If the request is successful, we then navigate to that room using the response's `room.name` value via the `goto` navigation method again. If there was an issue creating the room, we instead show an error message.

That brings us to the next obvious question: what is the `/room.json` endpoint we're using? Let's look!

Using SvelteKit's route-based endpoints

If we go back to our project's file structure, we'll recall there are two types of file types in the `/routes` directory.

In `/routes/room`, there is `[roomId].svelte` (where `[roomId]` is a dynamic value), and `index.json.js`.

In our `POST` request above, `/room.json` will get mapped to `/routes/room/index.json.js`, which will handle any requests matching that path.

Looking at that file, there is one function in it called `post`. (You could add `get` or `delete`, etc., but we just need `post`.) When the submit form makes a `POST` request to `/rooms.json`, it will look for a `post` function in `/routes/room/index.json.js` to complete the request.

```
export async function post() {
    /**
     * Note: You must at your Daily API key to an .env file
     * for this request to work. Refer to the README for
     * further instructions. :)
     */
    const DAILY_API_KEY = import.meta.env.VITE_DAILY_API_KEY;

    // add 30min room expiration
    const exp = Math.round(Date.now() / 1000) + 60 * 30;
    const options = {
```

```
        properties: {
            exp
        }
    };
    try {
        const res = await fetch('https://api.daily.co/v1/rooms',
            method: 'POST',
            headers: {
                Authorization: `Bearer ${DAILY_API_KEY}`,
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(options)
        });
        if (res.ok) {
            const room = await res.json();
            return {
                status: 200,
                body: JSON.stringify({
                    success: true,
                    room
                })
            };
        } else {
            return {
                status: res.status,
                body: JSON.stringify({
                    success: false
                })
            };
        }
    } catch (error) {
        return {
            status: 500,
            body: JSON.stringify({
                success: false,
                message: 'something went wrong with the room subm
            })
        };
    }
}
```

The post function looks similar to how you would typically make a fetch request using a third party API.

- First, we get our Daily API key, which was set in the `.env.local` file while getting our local dev environment set up.
  `const DAILY_API_KEY =`

`import.meta.env.VITE_DAILY_API_KEY;`

- Next, we set our Daily room options. You can use whatever options you want but we've just set a 30 minute expiration to avoid long-lasting demo rooms.
- Then, we actually make our `POST` request to Daily's REST API using our API key to authenticate the request:

```
const res = await fetch('https://api.daily.co/v1/rooms', {
            method: 'POST',
            headers: {
                Authorization: `Bearer ${DAILY_API_KEY}`,
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(options)
        });
```

- Finally, we return the response from our request, either with a success message or an error message to be handled by our client-side code.

```
if (res.ok) {
            const room = await res.json();
            return {
                status: 200,
                body: JSON.stringify({
                    success: true,
                    room
                })
            };
        }
// error handling below
```

Once this response is received in our `RoomForm`'s `submitForm` function, it will navigate to the room's page or show an error, as mentioned before.

```
async function submitForm() {
    ...
    const submit = await fetch('/room.json', {
        method: 'POST'
    });
```

```
    const data = await submit.json();

    if (data.success && data?.room?.name) {
        goto(`/room/${data.room.name}`);
    ...
```

RoomForm.svelte

Navigating to our room page

Now that we've navigated to our room's page (`https://localhost:3000/room/[room-name]`), we're officially ready to build our video call with `daily-js` ! 🙌

`[roomId].svelte` is where we will build out our call in the next tutorial, including video, audio, chat, and screen sharing.

Wrapping up

We hope this helps you get started on building a custom Daily video chat app with SvelteKit. To get a head start on our next post, check out the completed demo app in Github. 🌟

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 Svelte

How to display device permission errors in a custom video or audio call app built with Daily

17 Mar 2022 - 6 min read

Build a custom text chat sidebar in a Svelte video app with Daily

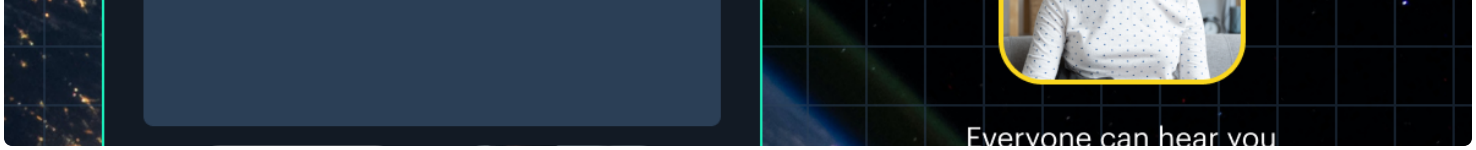
28 Feb 2022 - 9 min read

Build a custom Daily video chat app in SvelteKit (Part 2)

21 Feb 2022 - 12 min read

See all 4 posts →





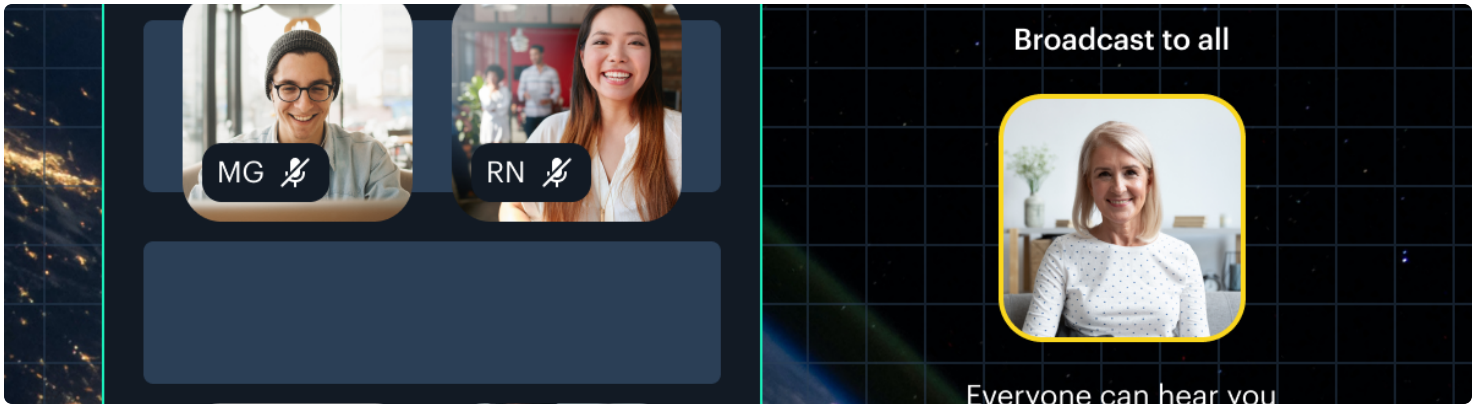
Spatialization

Running a 2D world for video call participants to traverse (Part 3)

User spawning, navigation, and remote user updates.



Liza Shulyayeva
14 Feb 2022 – 13 min read



Spatialization

Setting up the Daily call in our spatialization demo (Part 2)

Joining the Daily call and configuring our 2D world



Liza Shulyayeva
7 Feb 2022 – 13 min read