

Remix Party

You're invited.



Remix is a full-stack **web** framework based on web fundamentals

The logo consists of the word "Remix" in a bold, sans-serif font. Each letter is filled with a different color from a rainbow gradient: blue, green, yellow, and red. The letters are slightly overlapping, creating a sense of depth. The background is a dark, solid color.

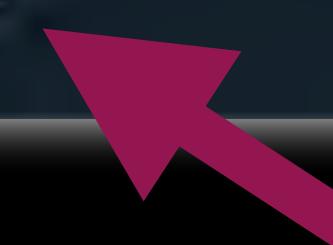
“Remix is a framework to build overengineered WEBSITES”



A guy on the internet
a_guy_on_the_internet

@dave_bitter Remix is a framework to build overengineered WEBSITES. You appear to not know the difference between a website (a place on the internet with a URL) and an app (a standalone program for a mobile device).

Twitter Web App



“Remix counters overengineering by going back to web fundamentals”



Dave Bitter 🚀
@dave_bitter

@a_guy_on_the_internet Hi A guy on the internet,
sorry you feel that way. I believe the power of web
apps is the omni platform nature of the web
making useful tools for people accessible through
their browser. You actually might find Remix
interesting as it counters over engineering by going
back to web fundamentals.

"Web apps will inevitably fail at being a native app"



A guy on the internet
a_guy_on_the_internet

@dave_bitter You are still talking about "web apps". There is no such thing. Websites have an URL. Apps are standalone programs for mobile devices. You can try mashing them together but that will inevitably fail being good at one or the other.

“The web offers functionalities that we sometimes forget to leverage”



Dave Bitter 🚀

@dave_bitter

@a_guy_on_the_internet Yes, web apps, as in Progressive Web Apps that you can run standalone on a device [web.dev/progressive-we...](https://web.dev/progressive-web-apps/). It has its pros and cons, as do native apps have. I believe there is a place where either of them make more sense

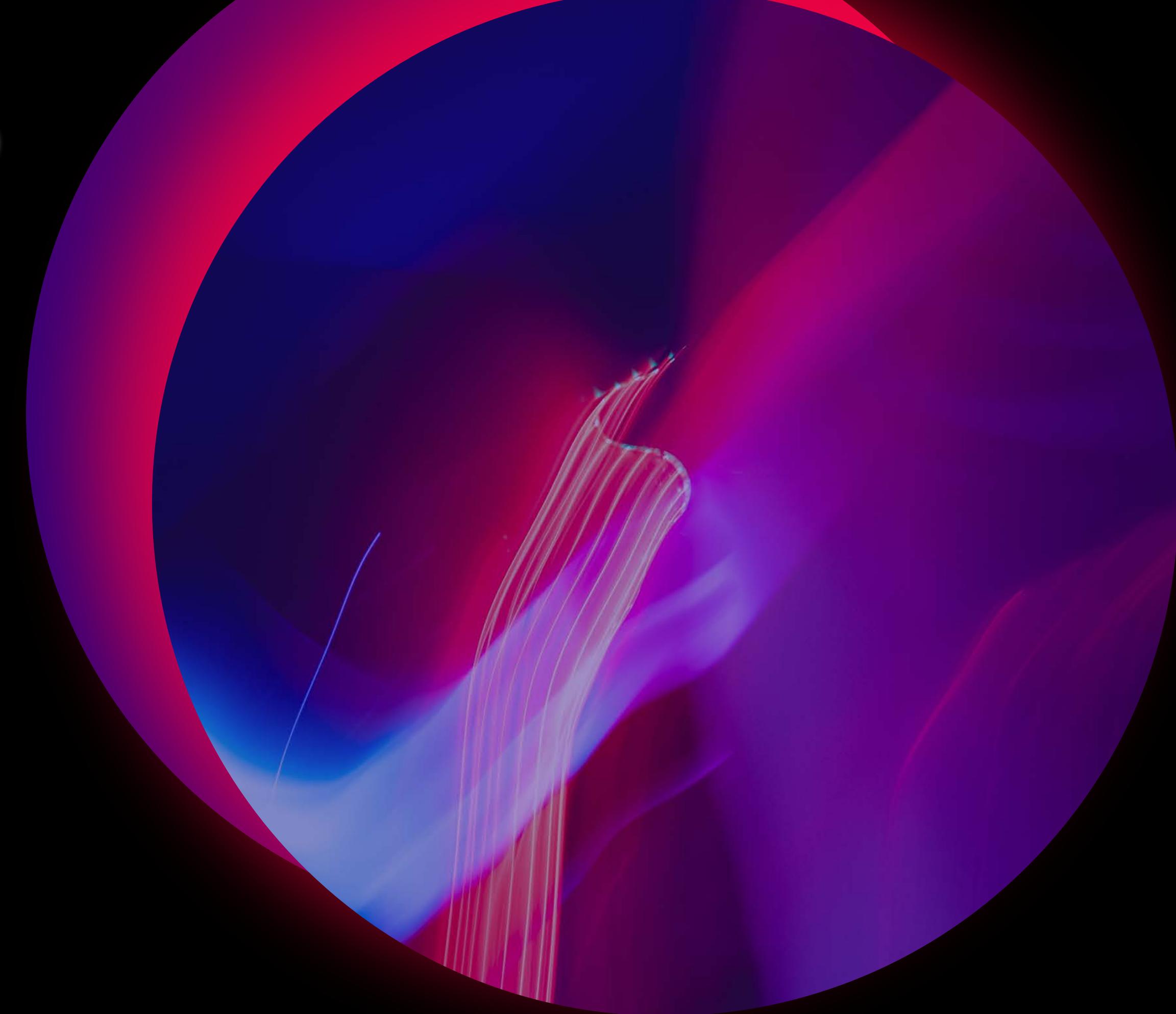
**TELL ME YOU'RE
FRUSTRATED BY ALL
THESE FRAMEWORKS**

without telling me you're frustrated by all these frameworks

“You should totally come to the party!”

|

- “You should see the lineup!”
- “This person is going to be there!”
- “It’s just good vibes!”



FOMO

: fear of missing out : fear of not being included in something (such as an interesting or enjoyable activity) that others are experiencing

THESE GUYS

Complex state

Client-side only

Outdated data

The same, but different

Non-standards

THIS IS WHERE
REMIX COMES IN



FEATURES

01

CREATE & JOIN
SESSIONS



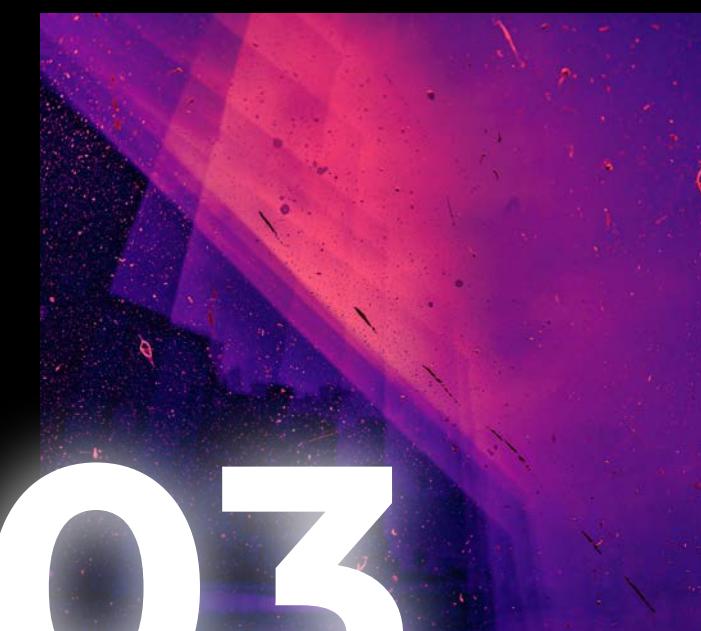
02

CAST A VOTE



03

TOGGLE
VISIBILITY



04

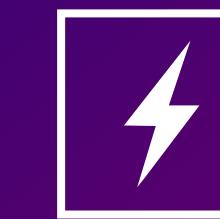
CLEAR ROUND



**LET'S GET THE
PARTY STARTED!**



— THE LINEUP —



CLUB REMIX



MC_{ss} TAILWIND



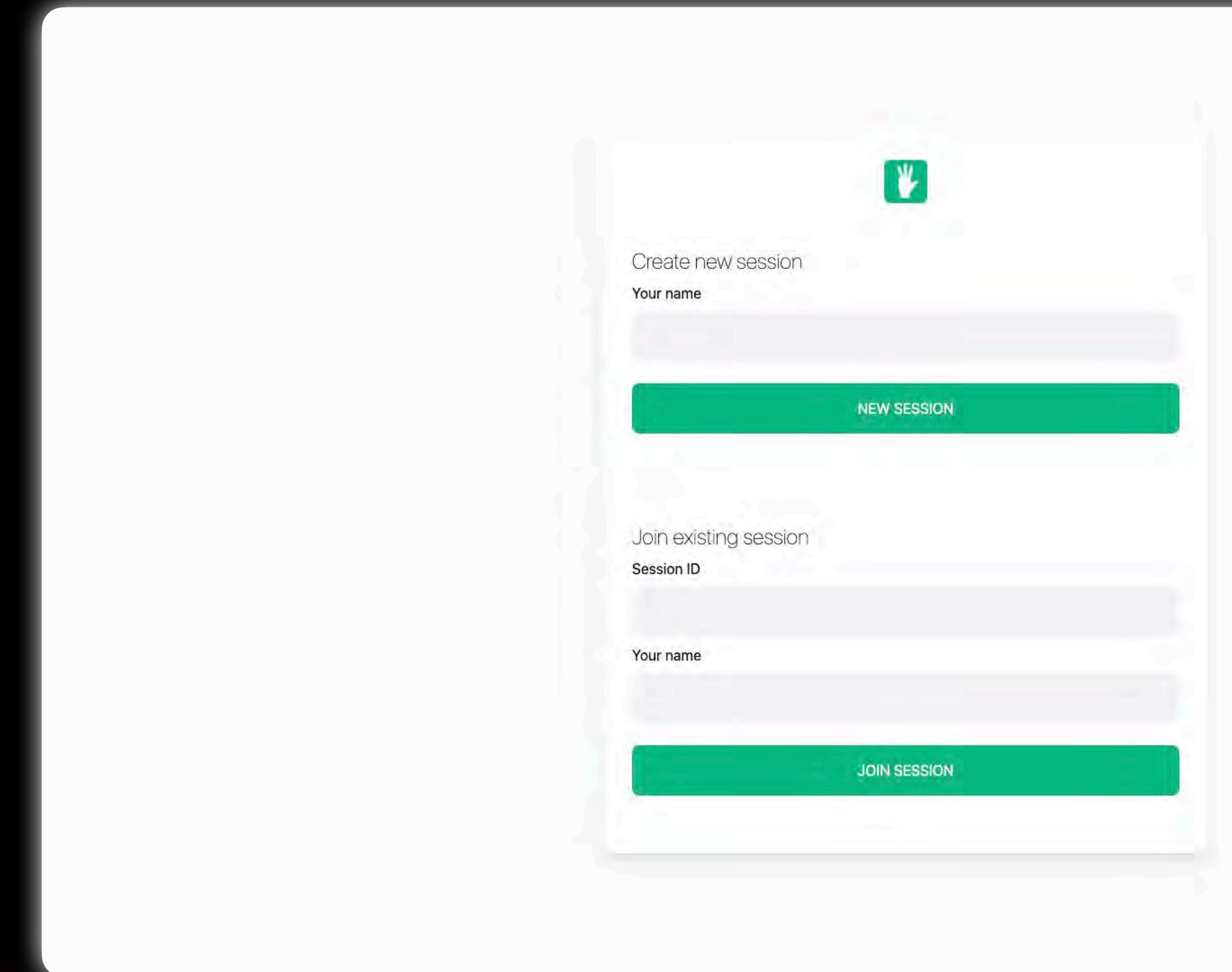
DJ SUPABASE

FRAMEWORK OF CHOICE

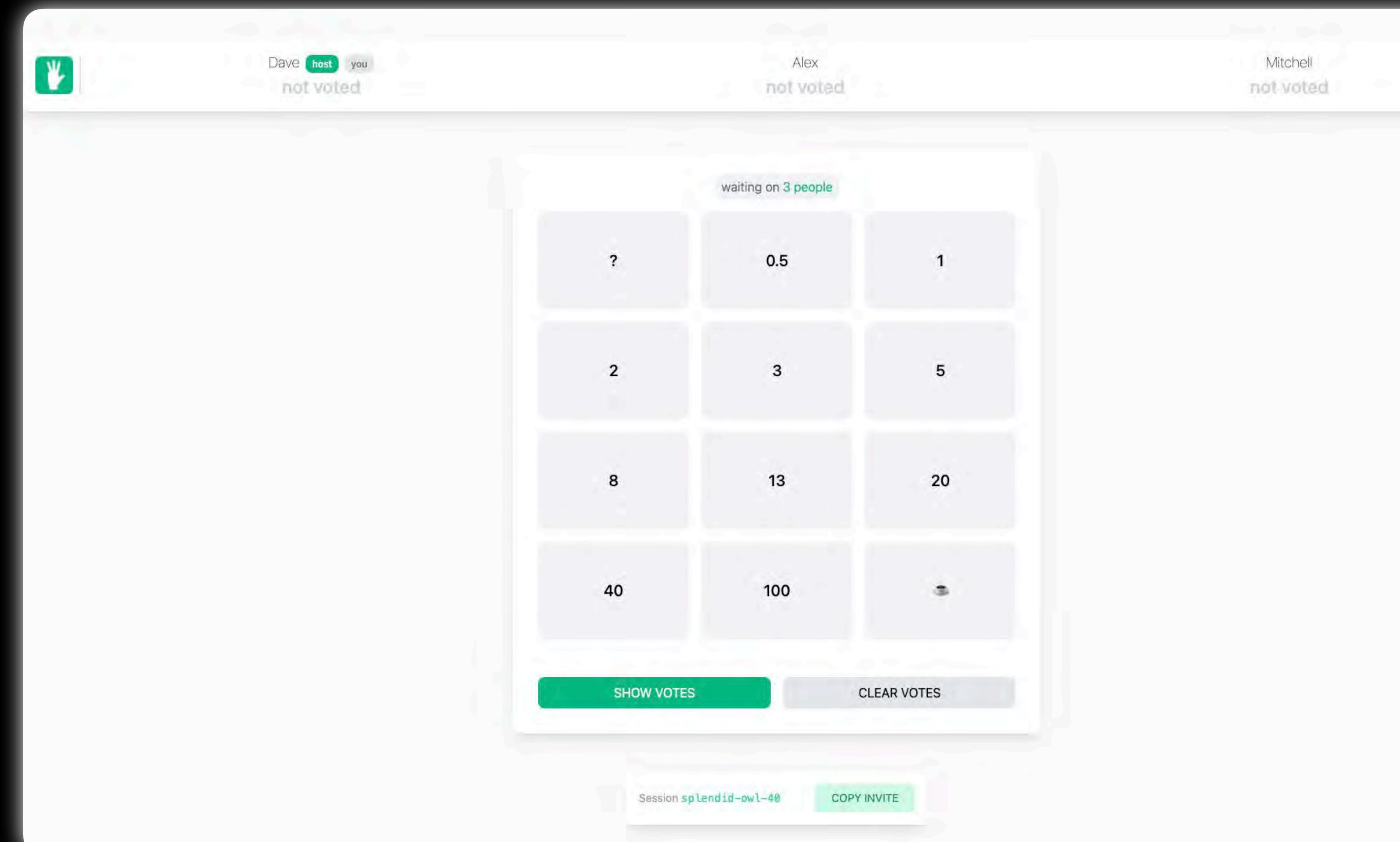
DATABASE OF CHOICE

STYLING FRAMEWORK OF CHOICE

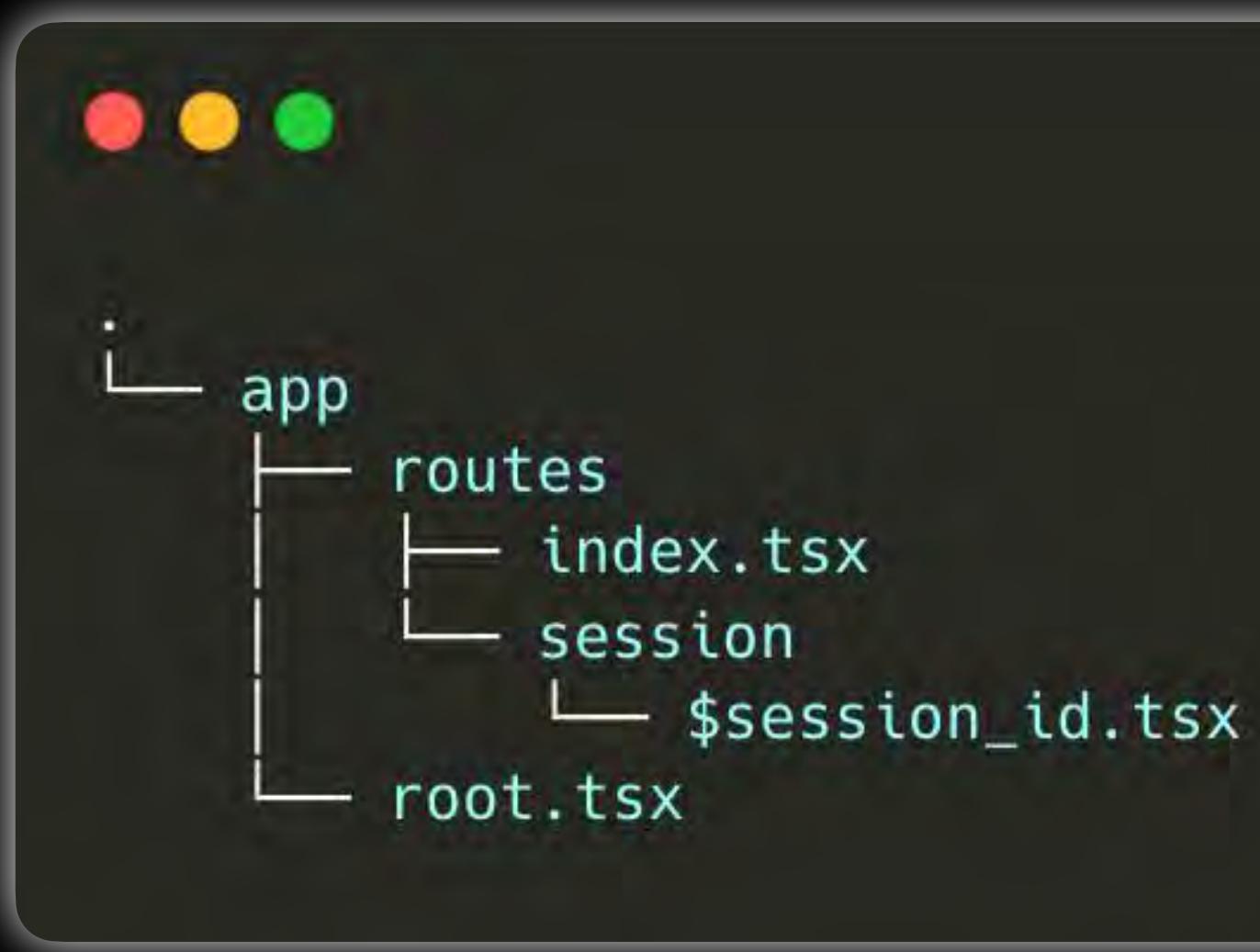
Create or join session page



Dynamic session page



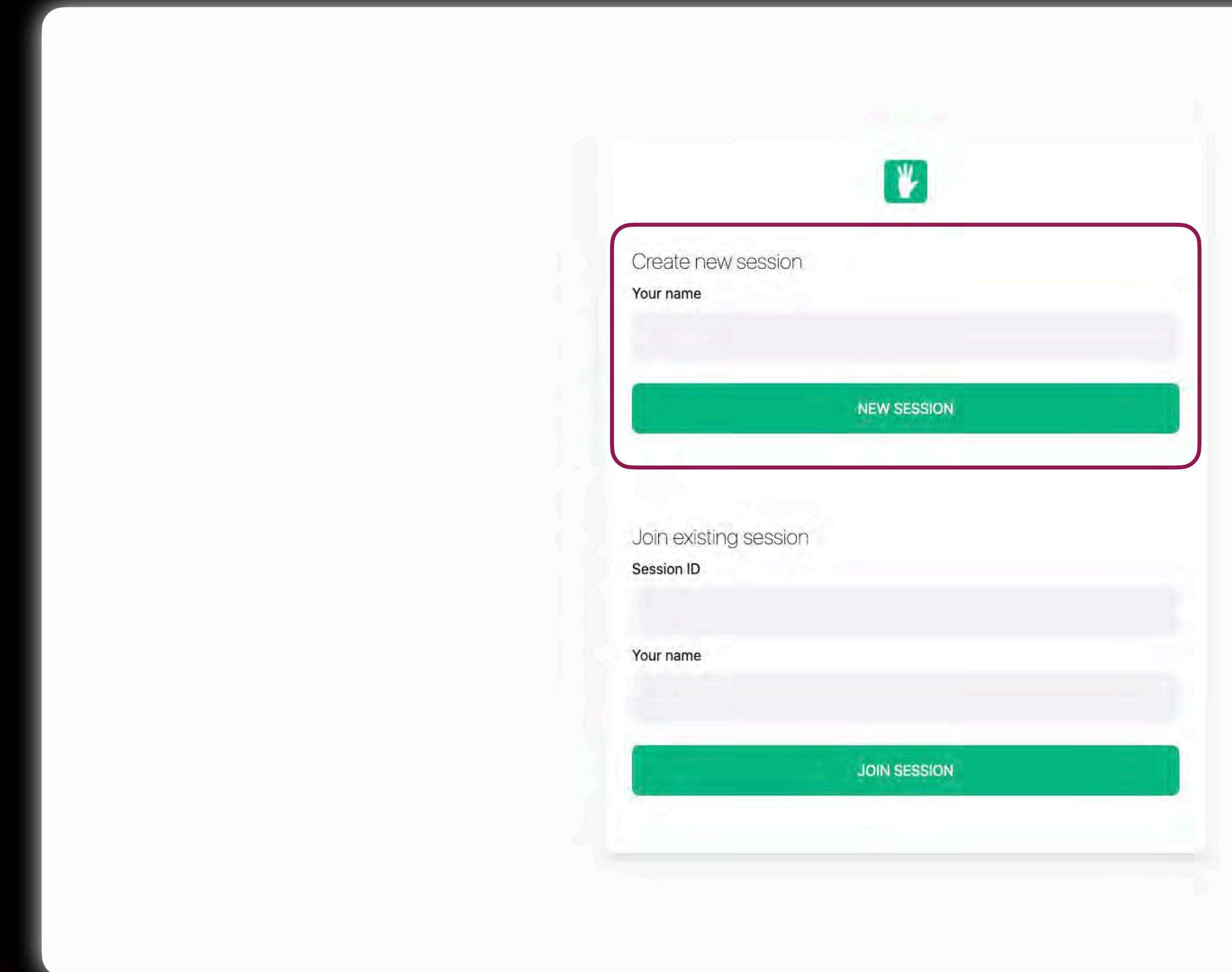
File based routing



A screenshot of a Mac OS X desktop showing a file tree in a terminal window. The window has red, yellow, and green close buttons at the top. The file structure is as follows:

```
└── app
    ├── routes
    │   ├── index.tsx
    │   └── session
    │       └── $session_id.tsx
    └── root.tsx
```

Create or join session page



Creating a basic form for the web

```
import { Form } from 'remix';

export default () => {
  return (
    <main>
      <h2>Create new session</h2>

      <Form method='post'>
        <label htmlFor='username'>Your name</label>
        <input id='username' name='username' required />
        <button type='submit'>New session</button>
      </Form>
    </main>
  )
}
```

Handling a form post through an action

```
import { Form, redirect, useActionData } from 'remix';
import { hri } from 'human-readable-ids';

export const action = async ({ request }) => {
  const formData = await request.formData();
  const username = formData.get("username");

  const newSession_id = hri.random();

  const { error } = await supabaseClient
    .from('sessions')
    .insert({ session_id: newSession_id })
    .single()

  return error ? { error: JSON.stringify(error) } :
  redirect(`/session/${newSession_id}`);
}

export default () => {
  const actionData = useActionData(); /* Contains any potential data returned
  from action function */

  return (
    /* Your render logic */
  )
}
```

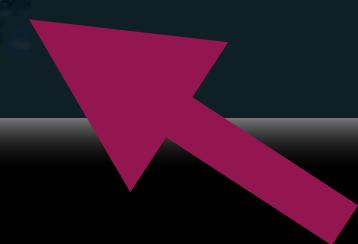
PROGRESSIVELY ENHANCE WITH JAVASCRIPT



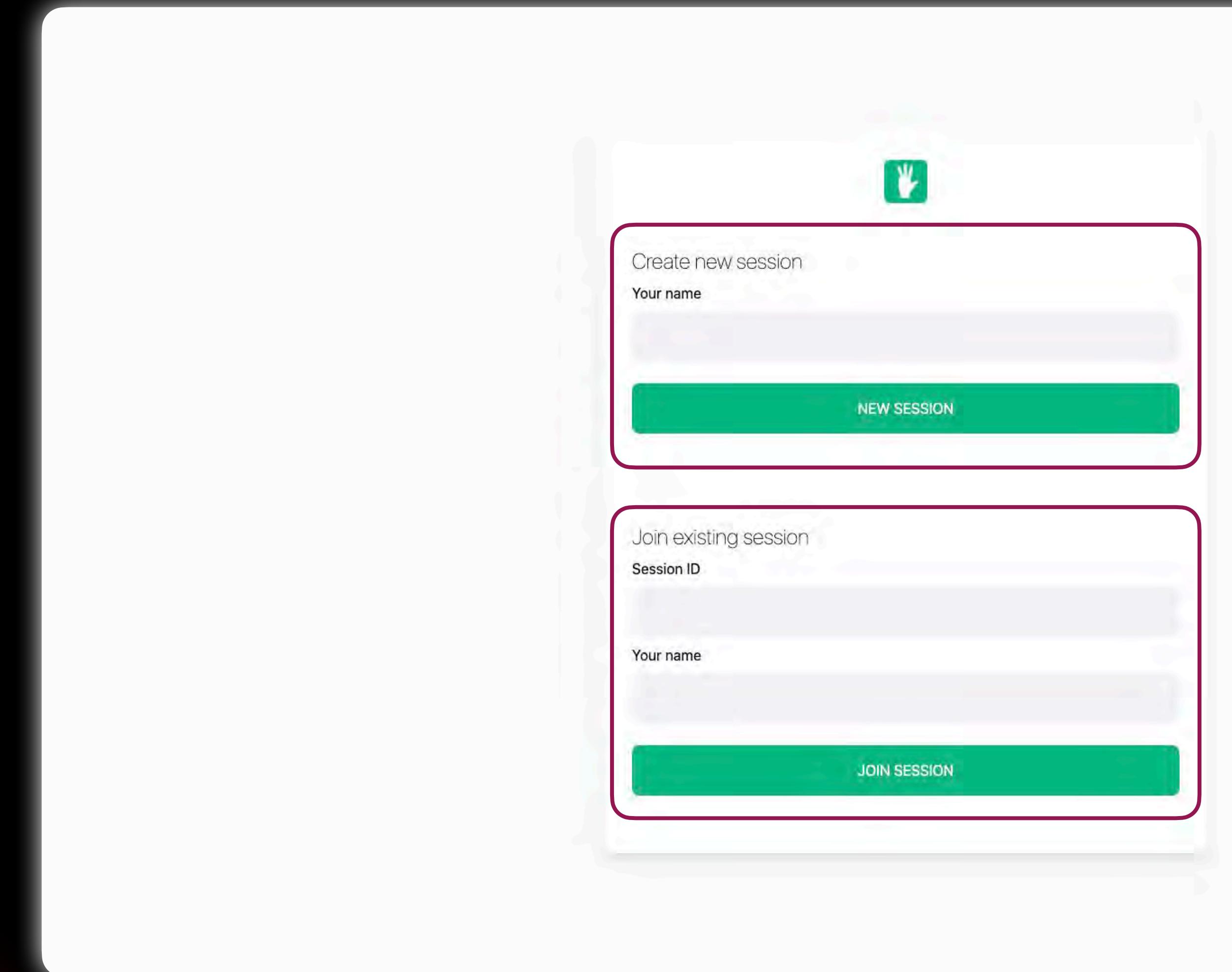
Jake Archibald
@jaffathecake

"We don't have any non-JavaScript users" No, all
your users are non-JS while they're downloading
your JS

3:09 PM · May 28, 2012



Create or join session page



Adding a second form with unique identifier

```
import { Form } from 'remix';

export default () => {
  return (
    <main>
      <h2>Join existing session</h2>

      <Form method='post'>
        <input
          name='form_type'
          defaultValue='join_session'
          required
          hidden
        />
        <label htmlFor='session_id'>Session ID</label>
        <input id='session_id' name='session_id' required />
        <label htmlFor='username'>Your name</label>
        <input id='username' name='username' required />
        <button type='submit'>Join session</button>
      </Form>
    </main>
  )
}
```

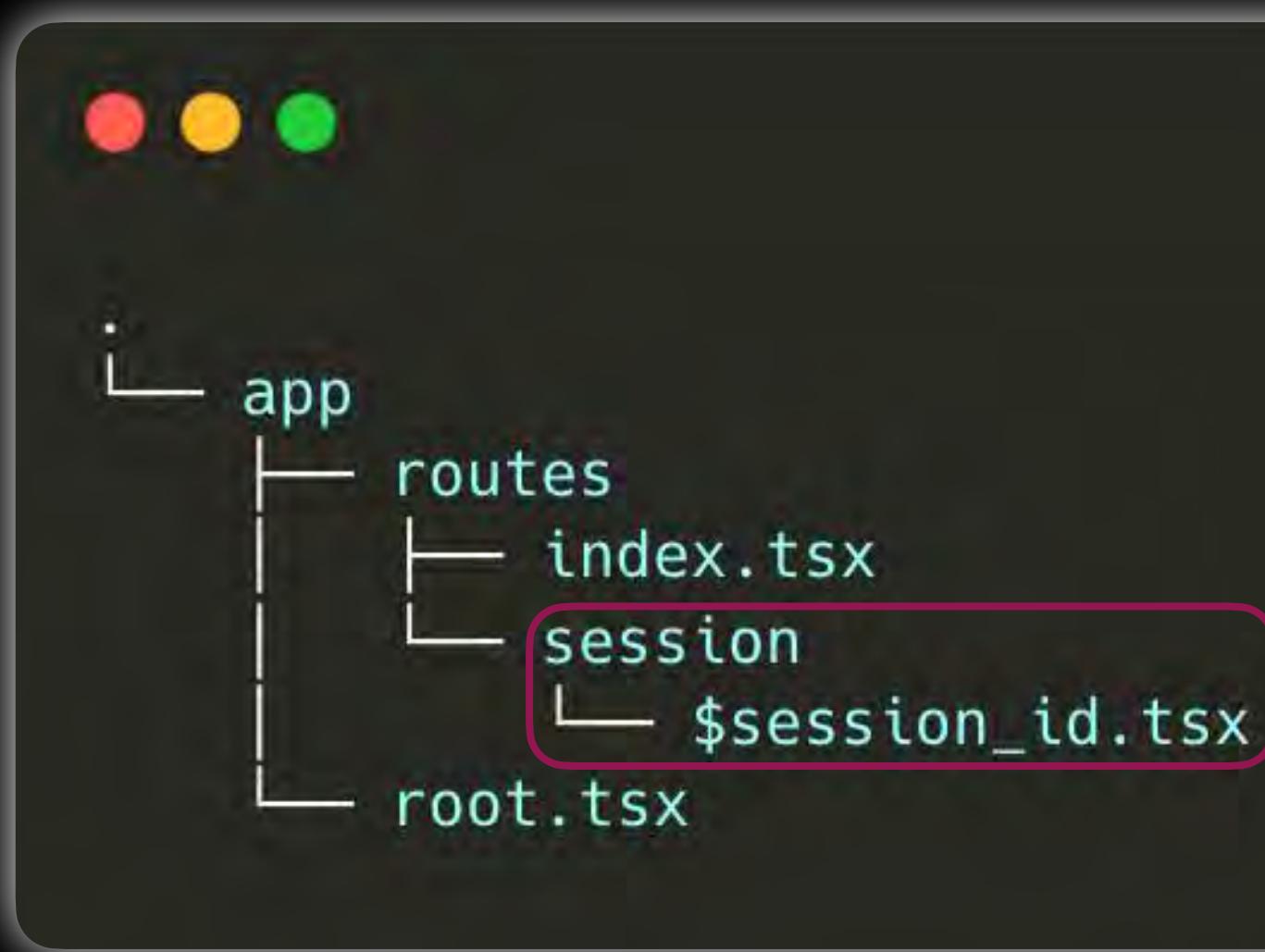
Handling multiple forms in single action

```
import { Form } from 'remix';

export const action = async ({ request }) => {
  const formData = await request.formData();
  const form_type = formData.get('form_type');

  switch (form_type) {
    case 'create_session':
      /* handle your logic */
      break;
    case 'join_session':
      /* handle your logic */
      break;
    default:
      break;
  }
}
```

Adding the dynamic page



```
└── app
    ├── routes
    │   └── index.tsx
    └── session
        └── $session_id.tsx
    └── root.tsx
```

Dynamic session page

The screenshot displays a dynamic session page with a red header bar. The header contains three user profiles: Dave (host), Alex, and Mitchell, all labeled "not voted". Below the header is a large input field for voting, which includes a placeholder message "waiting on 3 people". The input field is a grid of 12 boxes arranged in four rows of three. The visible values are: Row 1: ?, 0.5, 1; Row 2: 2, 3, 5; Row 3: 8, 13, 20; Row 4: 40, 100, (empty). At the bottom of the input field are two buttons: "SHOW VOTES" (green) and "CLEAR VOTES" (gray). At the very bottom of the page is a footer bar with the text "Session **splendid-owl-40**" and a "COPY INVITE" button.

Dave host you
not voted

Alex
not voted

Mitchell
not voted

waiting on 3 people

?

0.5

1

2

3

5

8

13

20

40

100

SHOW VOTES

CLEAR VOTES

Session **splendid-owl-40**

COPY INVITE

Copy to invite link

domain.com/?join_session_id=splendid-owl-40

Session **splendid-owl-40**

COPY INVITE

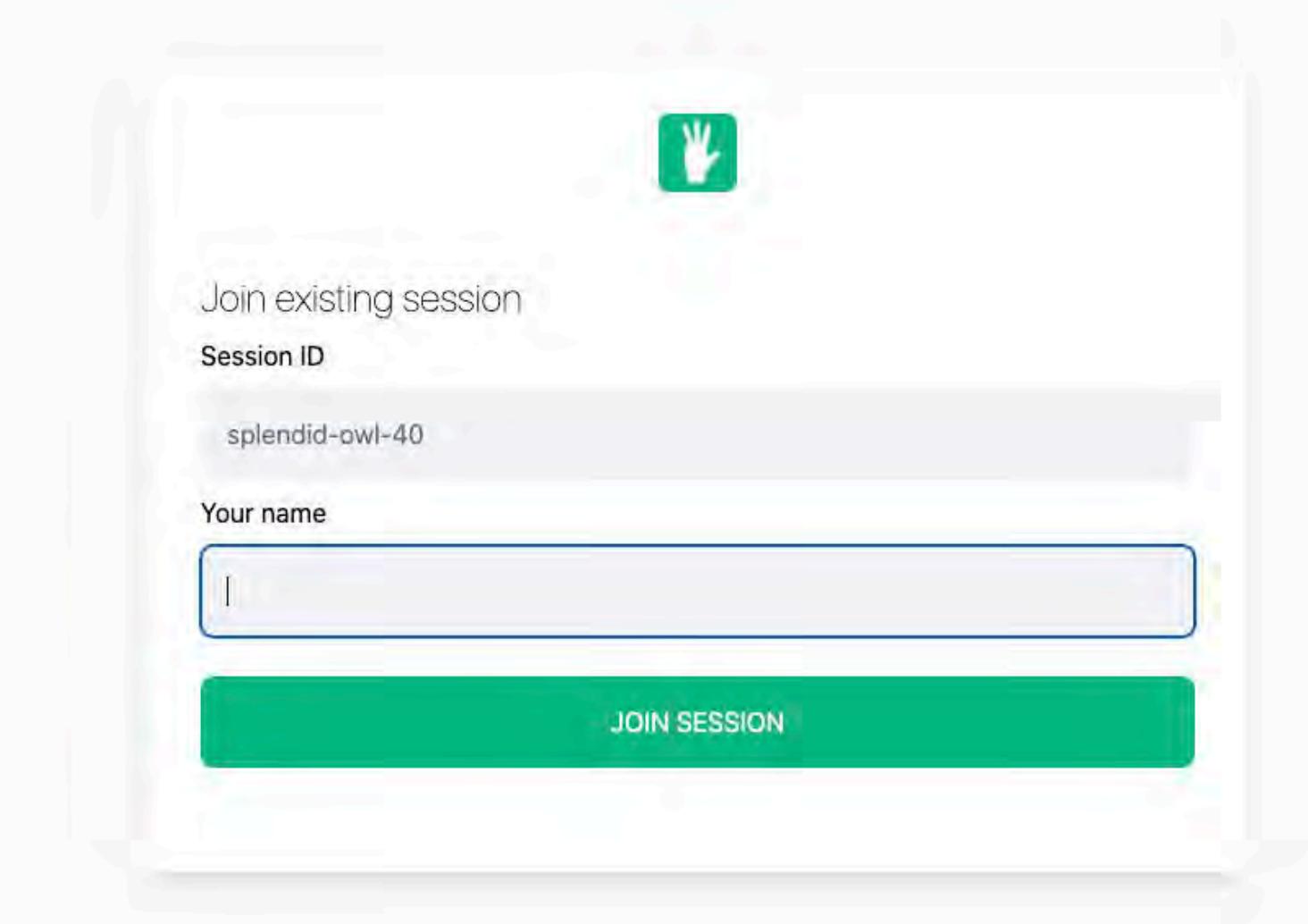
Some smart (SS) rendering for a better UX

```
import { Form, useSearchParams } from 'remix';

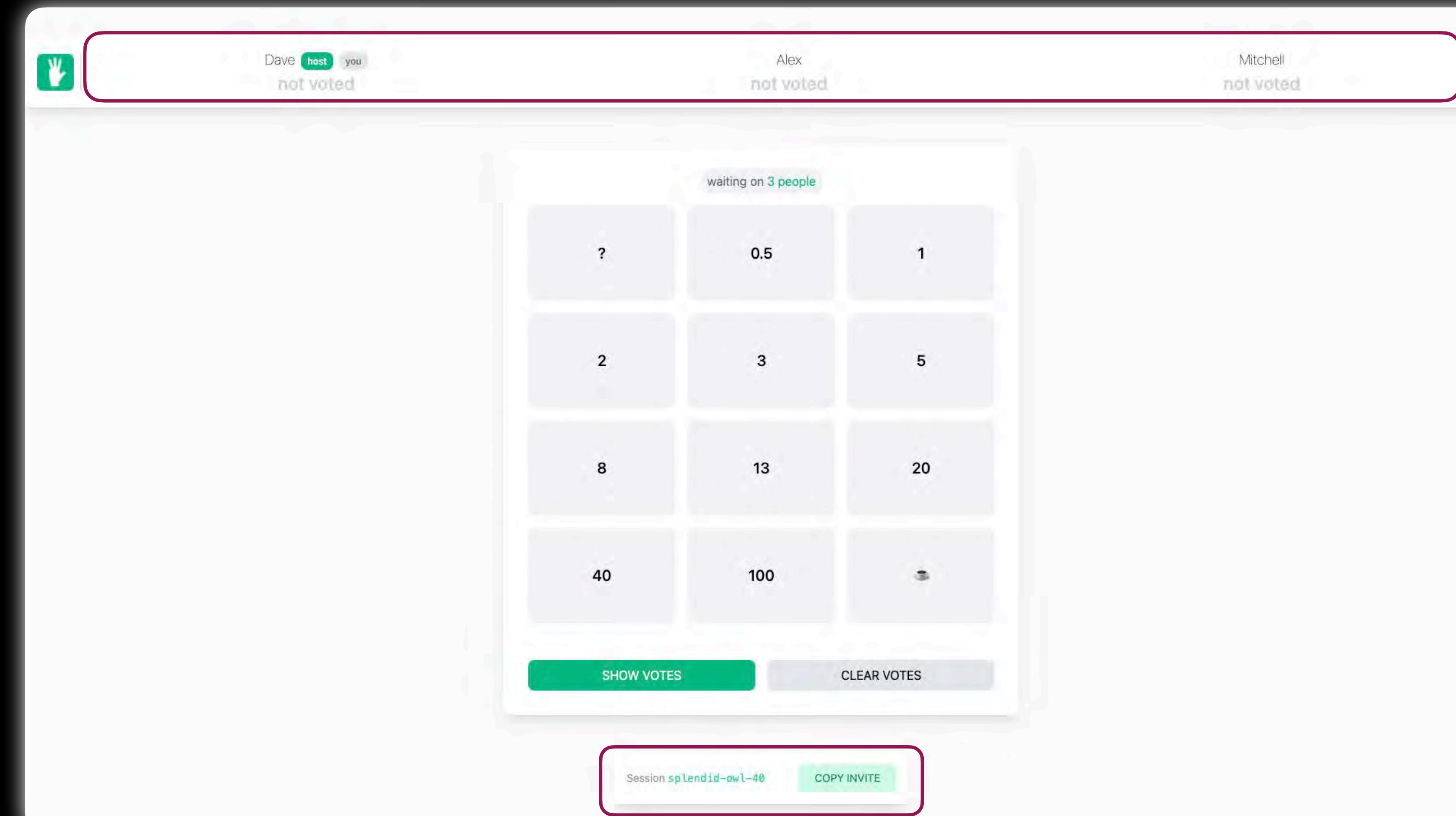
export default () => {
  let [ searchParams ] = useSearchParams();
  const join_session_id = searchParams.get('join_session_id');

  return (
    <main>
      {!join_session_id && <>
        <h2>Create new session</h2>
        <Form method='post'>
          <input name='form_type' defaultValue='create_session' required hidden />
          <label htmlFor='username'>Your name</label>
          <input id='username' name='username' autoFocus={!join_session_id} required />
          <button type='submit'>New session</button>
        </Form>
      </>}
    <h2>Join existing session</h2>
    <Form method='post'>
      <input name='form_type' defaultValue='join_session' required hidden />
      <label htmlFor='session_id'>Session ID</label>
      <input defaultValue={join_session_id} id='session_id' name='session_id' readOnly={!join_session_id} required />
      <label htmlFor='username'>Your name</label>
      <input id='username' name='username' autoFocus={!!join_session_id} required />
      <button type='submit'>Join session</button>
    </Form>
  </main >
);
}
```

Reusing the page in a smart way



Loading some data



Loading the current votes

```
import { useLoaderData, redirect } from 'remix';

export const loader = async ({ params }) => {
  const { sessionData } = await supabaseClient
    .from('sessions')
    .select('*')
    .eq('session_id', params.session_id)
    .single()

  if (!sessionData) {
    return redirect('/')
  }

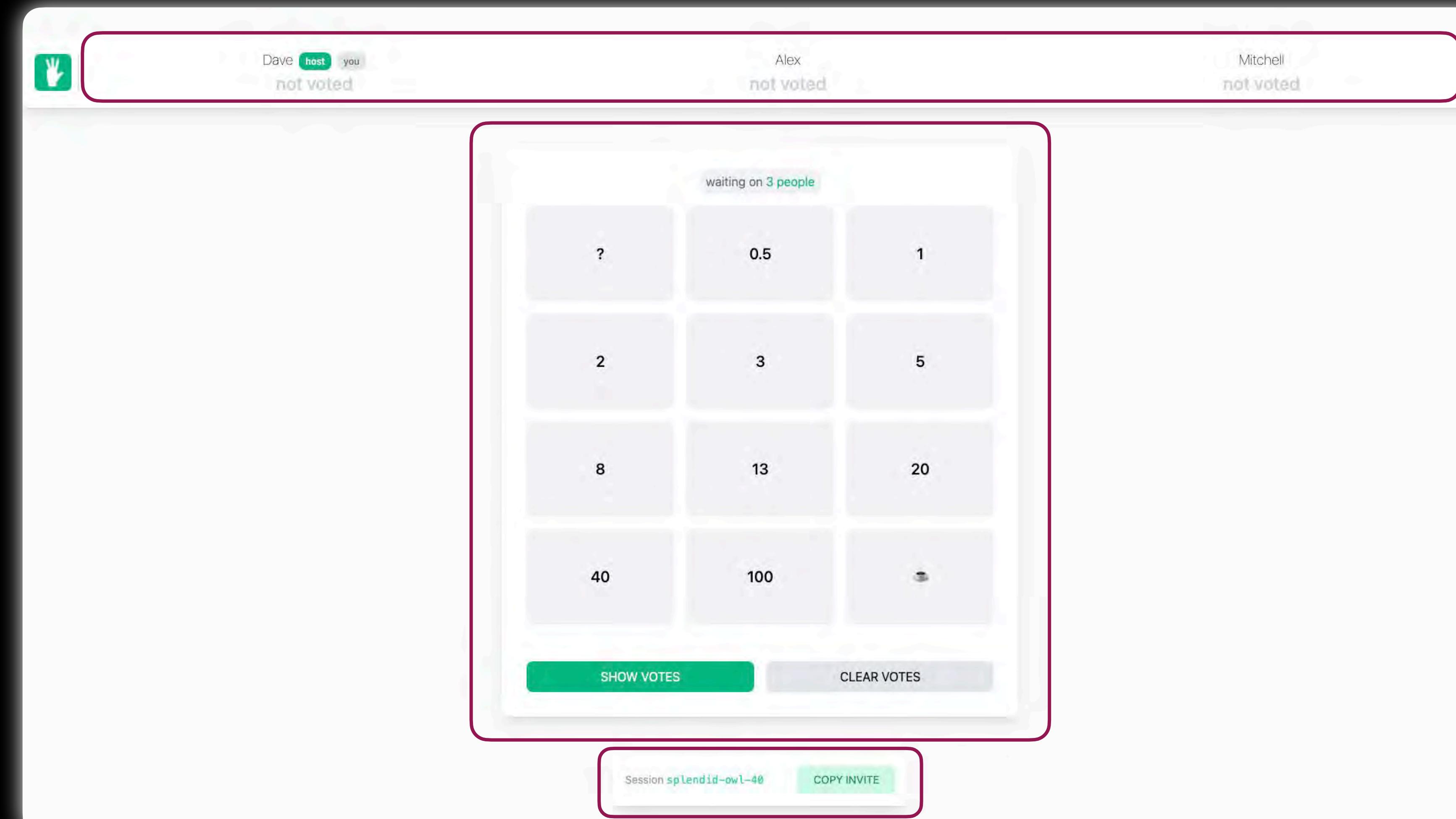
  const { data, error } = await supabaseClient
    .from('votes')
    .select('*')
    .eq('session_id', params.session_id);

  return {
    data,
    error
  }
}

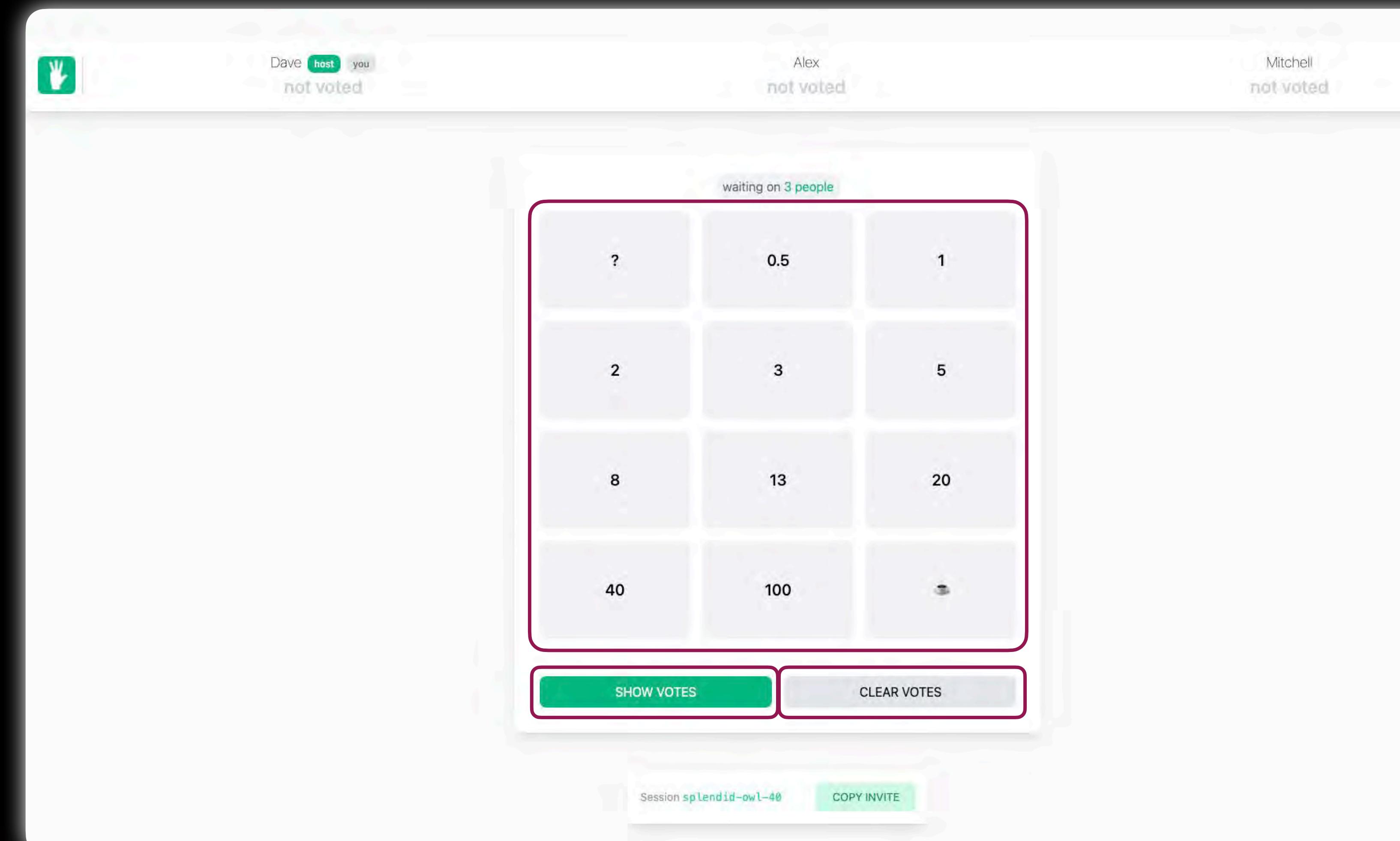
export default () => {
  const loaderData = useLoaderData();

  return (
    /* your render logic */
  )
}
```

Adding some interactivity



Multiple “micro-forms”



Isn't it just a radio button group?

```
import { Form, useLoaderData, useSubmit } from 'remix';

export default () => {
  const submit = useSubmit();
  const loaderData = useLoaderData();
  const activeUserEffort = loaderData.votes[`${
    loaderData.user.username
  }`];

  return (
    <main>
      <Form method='post' onChange={e => submit(e.currentTarget)}>
        <input name='form_type' defaultValue='update_effort' required
hidden />

        <fieldset id='effort'>
          {[ '?', '0.5', '1', '2', '3', '5', '8', '13', '20', '40',
          '100', '∞' ].map((effort: string) => <div key={effort}>
            <input
              checked={effort === activeUserEffort}
              type='radio'
              value={effort}
              name='effort'
              required
            />
            <label htmlFor={`effort_${effort}`}>{effort}</label>
          </div>)
        </fieldset>
        <button type='submit'>Submit</button>
      </Form>
    </main>
  )
}
```

Being clever with CSS

```
@keyframes show {
  from {
    position: absolute;
    z-index: -1
  }
  to {
    position: static;
    z-index: 0;
  }
}

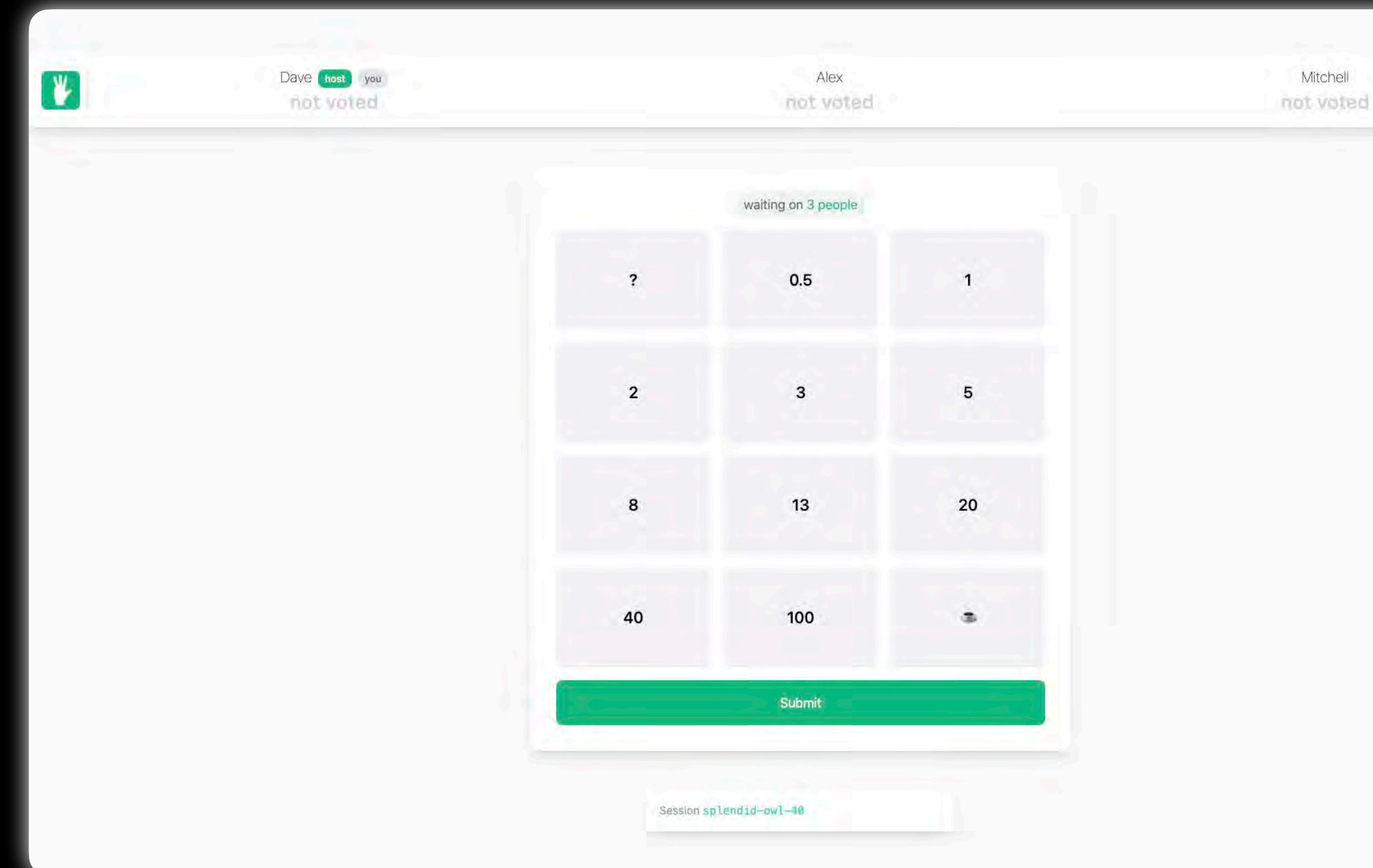
[data-has-js='true'] .no-js-show {
  display: none;
}

[data-has-js='false'] .no-js-show {
  position: absolute;
  z-index: -1;
  animation: show 0.01s ease-out forwards;
  /* Give JS a chance to load */
  animation-delay: 0.5s;
}

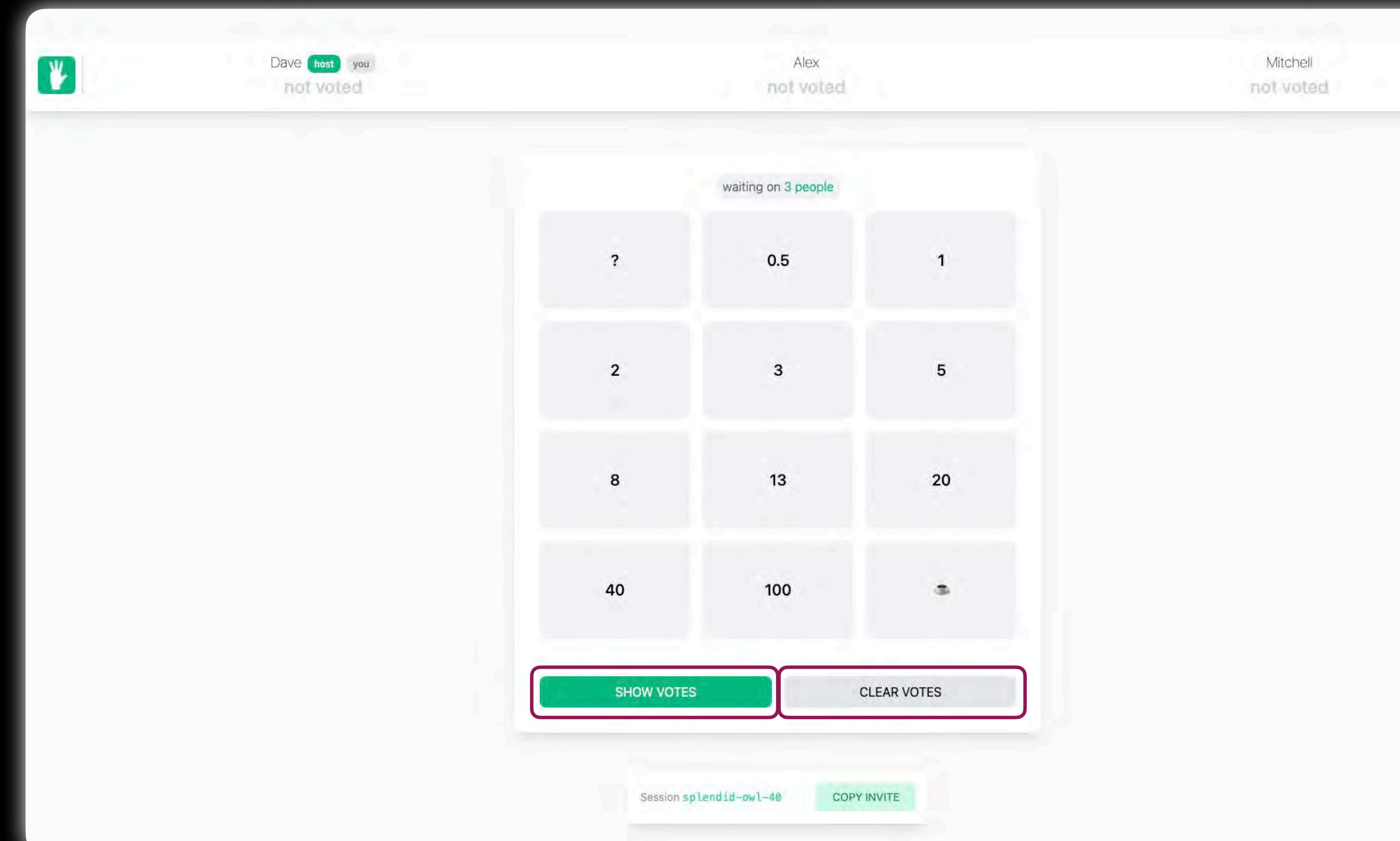
@keyframes hide {
  from {
    position: static;
    z-index: 0;
  }
  to {
    position: absolute;
    z-index: -1
  }
}

[data-has-js='false'] .no-js-hide {
  animation: hide 0.01s ease-out forwards;
  /* Give JS a chance to load */
  animation-delay: 0.5s;
}
```

Now we show a manual submit button for just the non-JS UI using CSS



Turning these “action-buttons” into “micro-forms”



Reusing the hidden form_type trick

```
import { Form, useLoaderData } from 'remix';

export default () => {
  const loaderData = useLoaderData();

  return (
    <main>
      <Form method='post'>
        <input
          name='form_type'
          defaultValue='toggle_effort'
          required
          hidden
        />
        <button type='submit'>
          {loaderData.votesVisible ? 'hide' : 'show'} votes
        </button>
      </Form>

      <Form method='post'>
        <input
          name='form_type'
          defaultValue='clear_effort'
          required
          hidden
        />
        <button type='submit'>Clear votes</button>
      </Form>
    </main>
  )
}
```

CAN YOU SHOW
US SOME ID?

Server session

```
import { createCookieSessionStorage, Session } from 'remix';

const { getSession, commitSession, destroySession } =
  createCookieSessionStorage({
    cookie: {
      name: '__session',
      path: '/'
    }
  });

const getSessionStorageInit: any = async (cookieSession: Session) => ({
  headers: {
    'Set-Cookie': await commitSession(cookieSession)
  }
})

export { getSession, getSessionStorageInit, destroySession };
```

On the create or join session page

```
import { redirect } from 'remix';
import { hri } from 'human-readable-ids';
import { v4 as uuidv4 } from 'uuid';

import { getSession, getSessionStorageInit } from '~/sessions';

export const action = async ({ request }) => {
  const formData = await request.formData();
  const session_id = formData.get('session_id');
  const cookieSession = await getSession(request.headers.get('Cookie'));
  const user_id = uuidv4();

  switch (form_type) {
    case 'create_session':
      const newSession_id = hri.random();
      cookieSession.set(newSession_id, { user_id, username });

      await supabaseClient
        .from('sessions')
        .insert({ session_id: newSession_id, host_id: user_id })
        .single()

      return redirect(`/session/${newSession_id}`, await getSessionStorageInit(cookieSession));
    case 'join_session':
      cookieSession.set(session_id, { user_id, username });

      return redirect(`/session/${session_id}`, await getSessionStorageInit(cookieSession));
    default:
      return {};
  }
}

export default () => {
  /* your render logic */
}
```

Use my user information from cookie to cast a vote

```
import { getSession } from '~/sessions';

export const action = async ({ request, params }) => {
    const session_id = params.session_id;
    const form = await request.formData();
    const form_type = form.get('form_type');
    const effort = form.get('effort');
    const session = await getSession(request.headers.get('Cookie'));

    const user = session.get(session_id);

    switch (form_type) {
        case 'update_effort':
            const { data: voteData } = await supabaseServerClient
                .from('votes')
                .select('*')
                .eq('user_id', user.user_id)
                .single();

            await supabaseServerClient
                .from('votes')
                .update({ ...voteData, effort })
                .eq('user_id', user.user_id)
                .break;
            /* You're other cases */
        default:
            break;
    }
}

export default () => {
    return (
        /* your render logic */
    )
}
```

Only expose selected data

```
import { json, LoaderFunction, redirect } from 'remix';
import { getSession, getSessionStorageInit } from '~/sessions';

export const loader = async ({ params, request }) => {
  const cookieSession = await getSession(request.headers.get('Cookie'));
  const user = cookieSession.get(params.session_id);

  if (!user) {
    return redirect(`/?join_session_id=${params.session_id}`);
  }

  const { data: sessionData } = await supabaseClient
    .from('sessions')
    .select('*')
    .eq('session_id', params.session_id)
    .single()

  user.isHost = sessionData.host_id === user.user_id

  const { data: votes } = await supabaseServerClient
    .from('votes')
    .select('*')
    .eq('session_id', params.session_id);

  let hostname;

  if (votes) {
    hostname = votes.find(({ user_id }) => user_id === sessionData.host_id)?.username;
    votes = votes.reduce((acc, { username, effort }) => ({ ...acc, [username]: effort }), {});
  }

  return json({
    session_id: params.session_id,
    votes_visible: sessionData.votes_visible,
    user,
    hostname,
    votes,
  }, await getSessionStorageInit(cookieSession));
};
```

MAKING IT
MORE SECURE

Adding some policies in Supabase

The screenshot shows the Supabase Authentication Policies interface. The left sidebar lists 'General', 'Users', 'Policies' (which is selected), 'Templates', 'Configuration', and 'Settings'. The main area displays two tables: 'sessions' and 'votes'.

sessions table:

- Restrict DELETE access to all users: false
- Restrict UPDATE access to all users: false
- Restrict INSERT access to all users: false
- Allow SELECT access to all users: true

votes table:

- Restrict INSERT access to all users: false
- Restrict DELETE access to all users: false
- Allow SELECT access to all users: true
- Restrict UPDATE access to all users: false

At the top right, there are 'Disable RLS' and 'New Policy' buttons. On the far right, there are 'DELETE', 'UPDATE', 'INSERT', 'SELECT' buttons for each table. A red arrow points from the bottom left towards the 'sessions' table.

Server only token



```
import { createClient } from '@supabase/supabase-js'
require('dotenv').config()

const supabaseUrl = process.env.SUPABASE_URL;
const supabaseSecretKey = process.env.SUPABASE_SECRET_KEY;

export const supabaseServerClient = createClient(supabaseUrl, supabaseSecretKey)
```

IT FEELS A BIT
SLOW

useTransition to the rescue for “optimistic UI”

```
import { useEffect, useState } from 'react';
import { useLoaderData, useTransition } from 'remix';

export default () => {
  const loaderData = useLoaderData();
  const transition = useTransition()
  const [optimisticVote, setOptimisticVote] = useState();
  const activeUserEffort = optimisticVote || loaderData.votes[`${
    loaderData.user.username}`];

  useEffect(() => {
    switch (transition.state) {
      case 'submitting':
        const effort = transition?.submission?.formData.get('effort');
        const form_type = transition?.submission?.formData.get('form_type');
        form_type === 'update_effort' && setOptimisticVote(effort);

        if (form_type === 'clear_effort') {
          setOptimisticVote(null);
          votesFormRef.current?.reset();
        }
        break;
      case 'idle':
        setOptimisticVote(null)
        break;
      default:
        break;
    }, [transition.state]);
  });

  return (
    /* your render logic */
  )
}
```

GOING REAL TIME

Remember, Remix isn't just forms and displaying data. We can enhance!

Supabase real-time hook

```
import { useEffect } from 'react';
import { createClient, SupabaseRealtimePayload } from '@supabase/supabase-js';

const useSupabaseSubscription = (
  SUPABASE_URL,
  SUPABASE_ANON_KEY,
  query,
  cb
) => {
  useEffect(() => {
    const subscription = createClient(SUPABASE_URL, SUPABASE_ANON_KEY)
      .from(query)
      .on('*', cb)
      .subscribe()

    return () => {
      subscription.unsubscribe()
    }
  }, []);
}

export default useSupabaseSubscription;
```

useFetcher to the rescue!

```
import { useFetcher, useLoaderData } from 'remix';

export default () => {
  const loaderData = useLoaderData();
  const fetcher = useFetcher();

  const votes = fetcher.data.votes || loaderData.votes;

  useSupabaseSubscription(
    loaderData.SUPABASE_URL,
    loaderData.SUPABASE_ANON_KEY,
    `sessions:session_id=eq.${loaderData.session_id}`,
    () => throttler(() => fetcher.load(window.location.pathname)));
  useSupabaseSubscription(
    loaderData.SUPABASE_URL,
    loaderData.SUPABASE_ANON_KEY,
    `votes:session_id=eq.${loaderData.session_id}`,
    () => throttler(() => fetcher.load(window.location.pathname)));

  return (
    /* your render logic */
  )
}
```

Going for a test run



REMIX ISN'T JUST SIMPLE FORMS

You can build entire real-time applications with it
and it will help you greatly!

MOVE THE STATE TO THE SERVER

It will make your application far easier to understand,
less error-prone and more robust

USE WEB FUNDAMENTALS

Great to use the web fundamentals
and use new tricks to have the UX up to par!

IT'S NOT ABOUT REMIX THE FRAMEWORK ITSELF

It's the thought behind it

THANKS

— L E T ' S C O N N E C T —



@DAVE_BITTER



DAVE BITTER



DAVEBITTER.COM | TECHHUB.IODIGITAL.COM