Presentation

Modern React Technologies

Nikos Kleidis
@nikoskleidis

Zacharias Fragkiadakis
@ZacFragkiadakis

Stavros Bastakis
@sstauross

Presentation

Contents

Modern Technologies

- Styled Components
- React Hook Form
- React Query
- Framer Motion

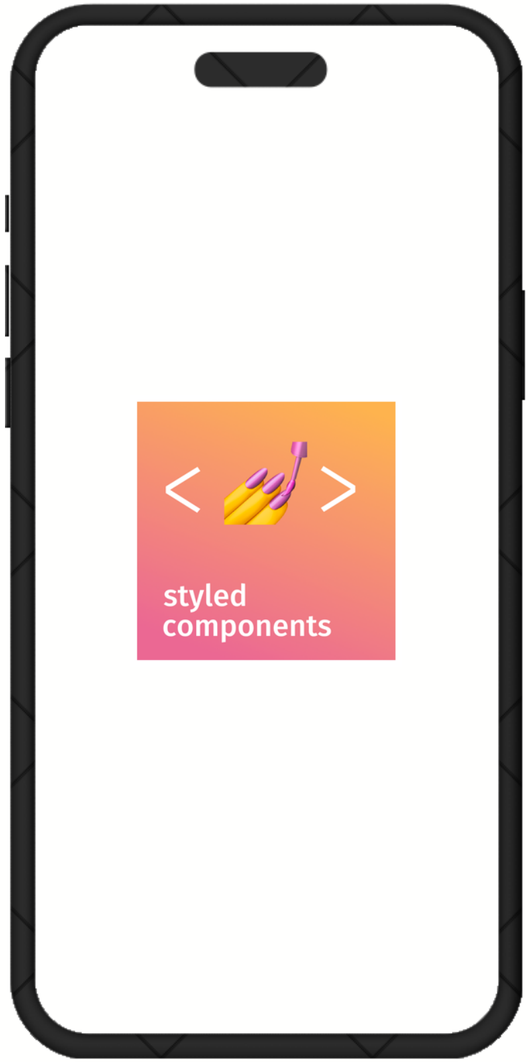
DevStaff Draw App

- Page Structure
- Home Page
- Participate Page
- Listing Page
- The Draw
- Deployment

CSS

Styled Components

Styled Components is a popular React library that allows developers to create and style components using CSS-in-JS. It simplifies the process of styling components by allowing CSS to be written directly in JavaScript code



Dynamic styling

Allows developers to use JavaScript expressions to create dynamic styles, making it easy to apply different styles based on the state of the components

Scoped styling

Allowing CSS to be written directly in JavaScript, ensuring that styles are only applied to the specific component and not to other parts of the application

Theming & Reusability

Enables developers to create reusable styles that can be applied to multiple components, reducing code duplication and improving maintainability. Also, it provides built-in support for theming

CSS

Dynamic styling

With styled-components, you can use props to pass dynamic values to your styles. This means that you can easily create variations of a component without having to write additional CSS classes

```
import styled from 'styled-components';
import Colors from './Colors';

const Button = styled.button`
  color: ${Colors.Black};
  background-color: ${(props) ⇒ (props.disabled ? 'gray' : 'green')};
  padding: 10px;
  border: none;
`;

<Button disabled>Click me!</Button>;
```

CSS

Scoped styling

The styles are scoped to the component that are define on. We don't have to worry about naming collisions with other CSS classes or styles. This is accomplished automatically by styled-components with the **dynamic generation** of classes.

```
<html>
  <head>...</head>
  <body>
    <button class="sc-173f745d-0">
      Click me!
    </button>
  </body>
</html>
```

CSS

Theming & Reusability

Styled-components make it easy to create a consistent design system by defining a set of global variables for colors, fonts, and other visual properties that can be accessed throughout the application.

```jsx
import { ThemeProvider } from 'styled-components';

const theme = {
  colors: {
    primary: '#0077cc',
    secondary: '#f3f3f3',
  }
};

const App = () => (
  <ThemeProvider theme={theme}>
    <div>
      <Button>Click me</Button>
    </div>
  </ThemeProvider>
);
```

Button.jsx

```jsx
import styled from 'styled-components';

export const Button = styled.button`
  color: ${props => props.theme.colors.primary};
  ...
`;
```

Forms

React Hook Form

React Hook Form is a lightweight and easy-to-use library that helps manage forms in React projects. It prioritizes performance and it's API is designed to handle form inputs, validation, and submission, providing a flexible and simple way to manage forms.

Simple & Lightweight

React Hook Form has a simple and easy-to-use API. Also, it doesn't have many dependencies making it lightweight and easy to add to projects.

Performance

Minimizes the number of re-renders, minimizes validate computation, and faster mounting.

Flexibility

React Hook Form supports a wide range of form inputs, including standard HTML inputs, custom inputs

Forms

register

The **register** function returned from the **useForm** hook allows us to bind an input field into React Hook Form so that it is available for the validation, and its value can be tracked for changes.

```
import { useForm } from "react-hook-form";

export default function App() {
  const { register } = useForm();

  return (
    <form onSubmit={...}>
      <input {...register("firstName")} />
      <input {...register("lastName")} />

      <input type="submit" />
    </form>
  );
}
```

Forms

errors

- **Adding validation** is as simple as adding an object as second parameter to the register function.
- **Built-in validations** based on the HTML standard.
  For example: required, maxLength, minLength, min, max etc. and the ability to create our own validator.
- **formState** object provides, among others, the **error** object with the fields' errors

```
import { useForm } from "react-hook-form";

export default function App() {
  const { register, formState: { errors } } = useForm();

  return (
    <form>
      <input {...register("firstName", {
        required: "Name is required",
        maxLength: {value: 20, message: "Name should be up to 20 characters" }})}
      />
      {errors.firstName ? <span>{errors.firstName.message}</span> : null}
      ...
    </form>
  );
}
```

Forms

handleSubmit

- **Validates** all the inputs in the form based on the rules defined in the register function. If any input is invalid, it will prevent the default form submission

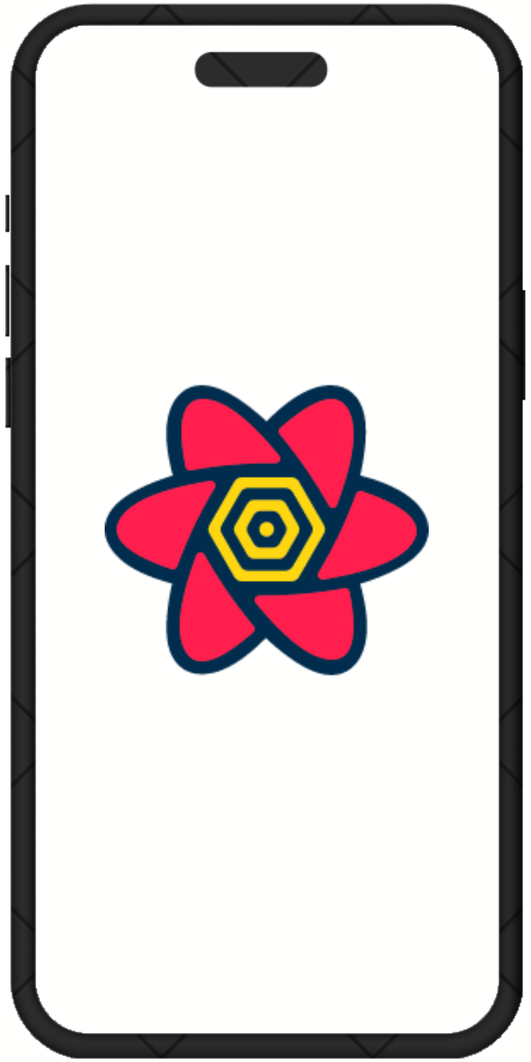
- **Executes the callback function** passed if all the inputs are valid

```jsx
import { useForm } from "react-hook-form";

export default function App() {
  const { register, formState: { errors }, handleSubmit } = useForm();

  const onSubmit = data => console.log(data); //{firstName: "", lastName:""}

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <input {...register("firstName", {
        required: "Name is required",
        maxLength: {value: 20, message: "Name should be up to 20 characters" }})}
      />
      {errors.firstName ? <span>{errors.firstName.message}</span> : null}
      <input {...register("lastName")} />

      <input type="submit" />
    </form>
  );
}
```

Data Fetching

React Query

React Query is a powerful caching library for React applications that simplifies the process of retrieving and managing data from APIs.



Efficient caching

Automatically caches API responses, reducing network requests and improving application performance

Automatic background updates

Performs background updates of cached data, ensuring that the application always displays the latest data.

Simplified data management

Provides a simple API for managing queries, including the ability to refetch data on demand, invalidate cached data, and synchronize data between multiple components in your application

React Query

useQuery

Promised based

- **queryFn** should return a promise
- **data** is *undefined* until the promise resolves

Data lifecycle hooks

- **isLoading** is true while data is fetched
- If **queryFn** throws an error, the **error** variable will contain the error

```
function UserDetails({ userId }) {
  const { isLoading, error, data } = useQuery({
    queryKey: ['user', userId],
    queryFn: () =>
      fetch(`http://example.com/users/${userId}`).then(
        (res) => res.json(),
      ),
  })
  if (isLoading) return 'Loading...'

  if (error) return 'An error has occurred: ' + error.message

  return (
    <div>
      <span>{data.name}</span>
      <span>{data.email}</span>
    </div>
  )
}
```

React Query

Caching Mechanism

Adding to cache

- Data returned by the **queryFn** is stored under the **queryKey** in the cache
- Different **queryKeys** create different entries in the cache

Retrieving from cache

- If data is found under the **queryKey** it is retrieved immediately
- A new request is initiated in the background to fetch the latest data

```
function useUserQuery(userId, options = {}) {
  return useQuery({
    queryKey: ['user', userId],
    queryFn: () =>
      fetch(`http://example.com/users/${userId}`).then(
        (res) => res.json()
      ),
    ...options
  })
}
```

```
function EditUserDetails({ userId }) {
  const { data } = useUserQuery(userId)
  // Loading data handle
  // error data handle
  return (
    <>
      <Input defaultValue={data.name} label="Name" />
      <Input defaultValue={data.email} label="email" />
    </>
  )
}
```

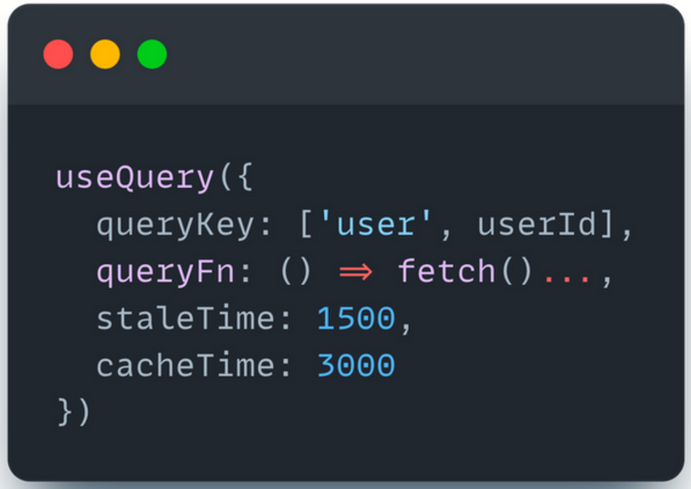
React Query

Caching Mechanism

Invalidation

- **staleTime**: duration in ms during which our data is deemed up-to-date. During this period, no background fetching of the data will occur as it is still considered fresh.

- **cacheTime**: time in ms that unused data will remain stored in the cache. Once this time period elapses, the data will be removed from the cache through garbage collection.



React Query

useMutation

- Update the client-side cache in the **onMutate** function with the optimistic results of the mutation, before waiting for the server's response.
- We can rollback the optimistic update on the **onError** function if something goes wrong
- **Invalidate** the queries that this mutation might affect in the **onSuccess** function. This will cause those all active queries to refetch and fresh queries to be marked as stale

```
const { mutate, isLoading, isSuccess } = useMutation({
  mutationFn: () => fetch('/user', { method: 'POST' }),
  onMutate: (variables) => {
    // A mutation is about to happen!
  },
  onError: (error, variables, context) => {
    // An error happened!
  },
  onSuccess: (data, variables, context) => {
    // Boom baby!
    queryClient.invalidateQueries({ queryKey: ['users'] })
  },
  onSettled: (data, error, variables, context) => {
    // Error or success... doesn't matter!
  },
})
```

React Query

DevTools

- View the cache data and query keys
- View **fresh**, **fetching**, **stale,** and **inactive** queries
- Trigger refetch or remove of existing queries

Animations

Framer Motion

Framer Motion is a React library that provides easy-to-write production-ready animations, providing a range of powerful features that simplify the animation process and ensure excellent performance.

Simple and elegant animations

Provides a straightforward declarative API for creating smooth and visually appealing animations

Built-in gesture recognition

Includes support for a variety of gestures, allowing to create interactive elements that respond to user input

Performance

Optimizes animation performance by utilizing hardware acceleration, allowing animations to run smoothly and efficiently by leveraging the power of the user's device.

Framer Motion

Declarative Syntax

"A simple declarative syntax means you write less code. Less code means your codebase is easier to read and maintain."

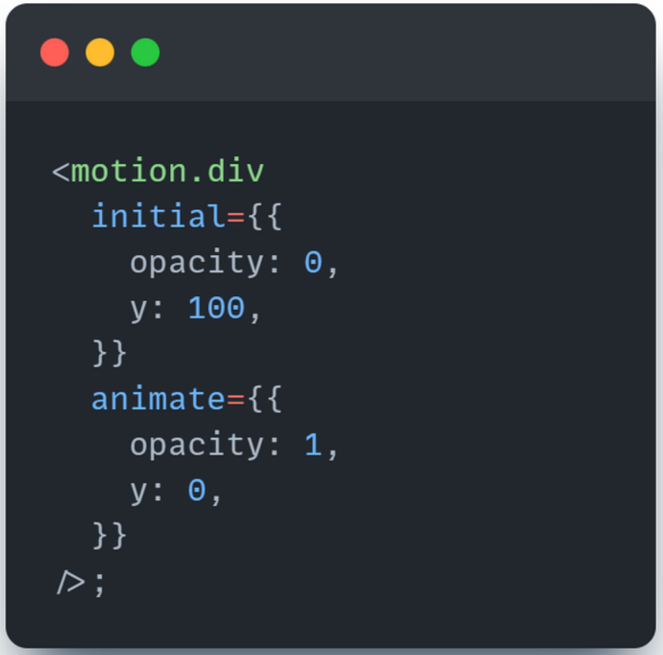
— Framer Motion docs

Framer Motion

Declarative Syntax

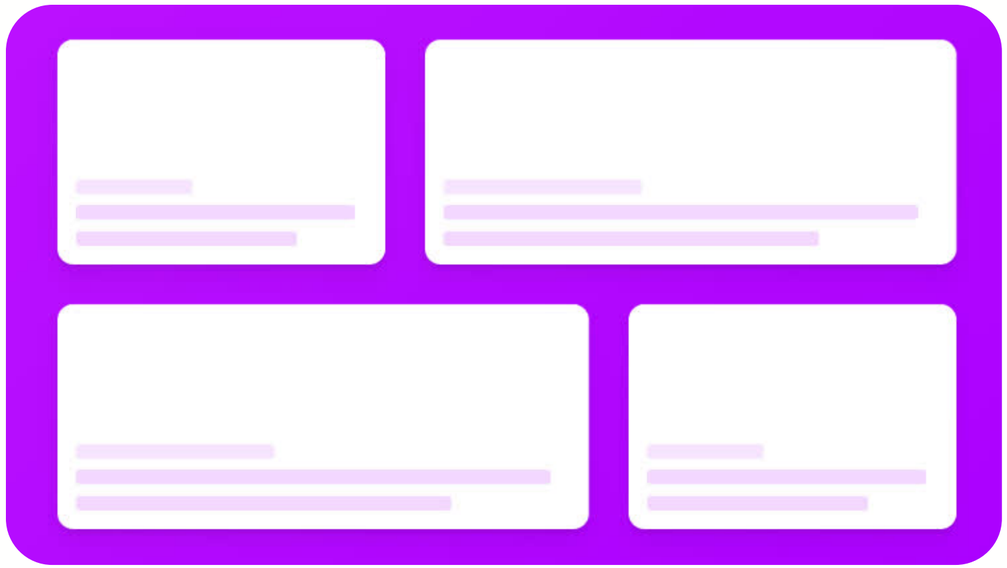
"A simple declarative syntax means you write less code. Less code means your codebase is easier to read and maintain."

— Framer Motion docs



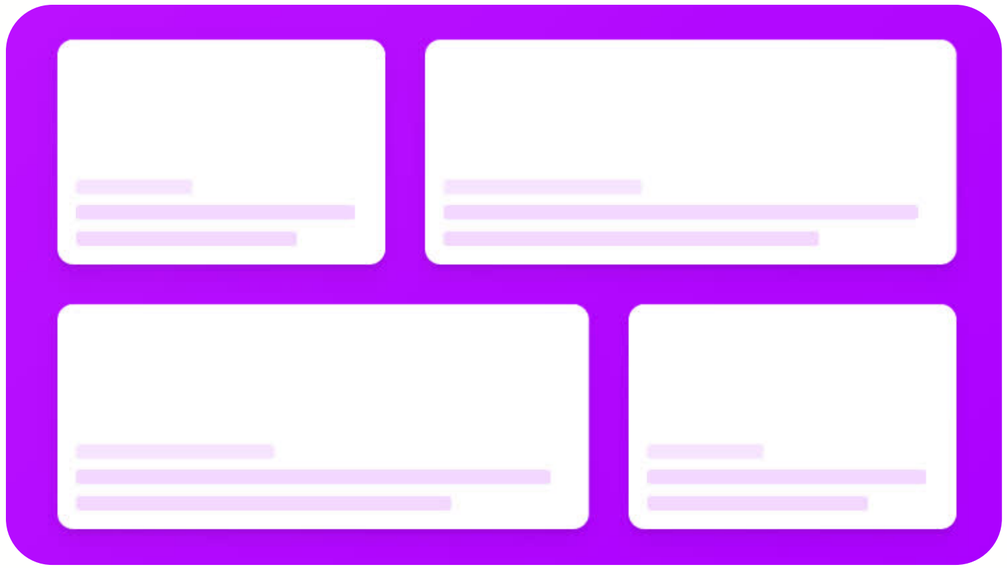
Framer Motion

Layout Animations

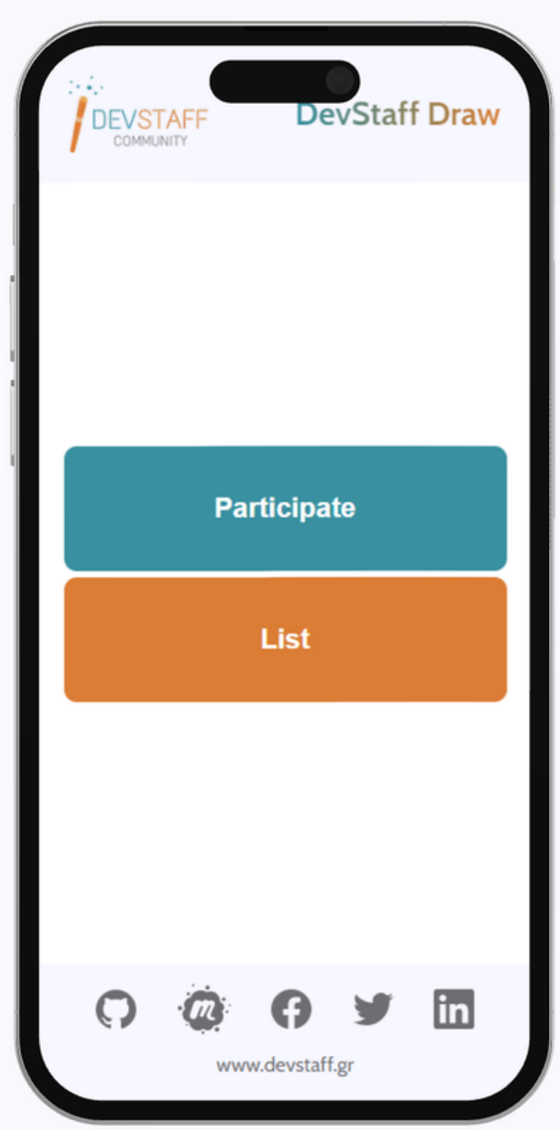


Framer Motion

Layout Animations



```
const [selectedId, setSelectedId] = useState(null)

{items.map(item => (
  <motion.div layoutId={item.id} onClick={() => setSelectedId(item.id)}>
    ...
  </motion.div>
))}

{selectedId && (
  <motion.div layoutId={selectedId}>
    ...
  </motion.div>
)}
```

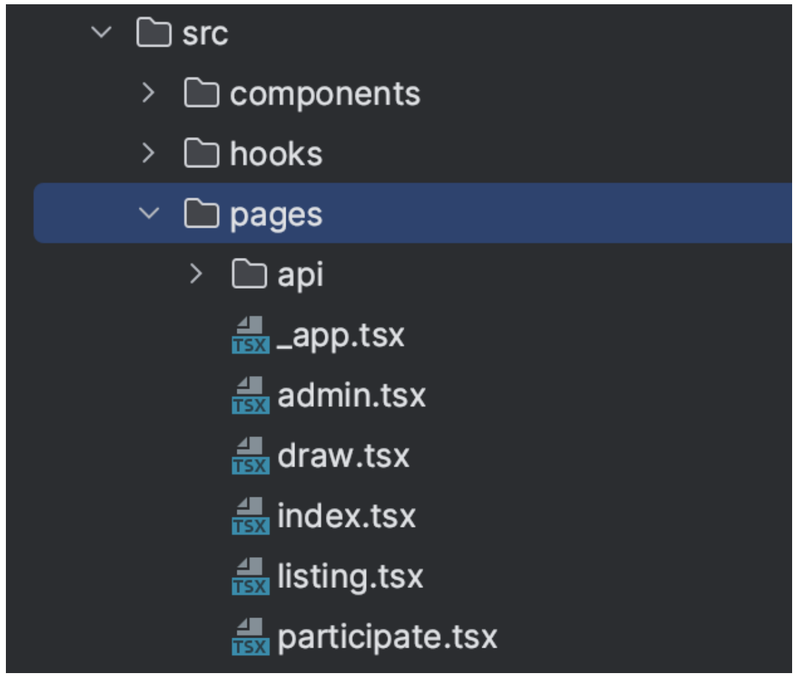


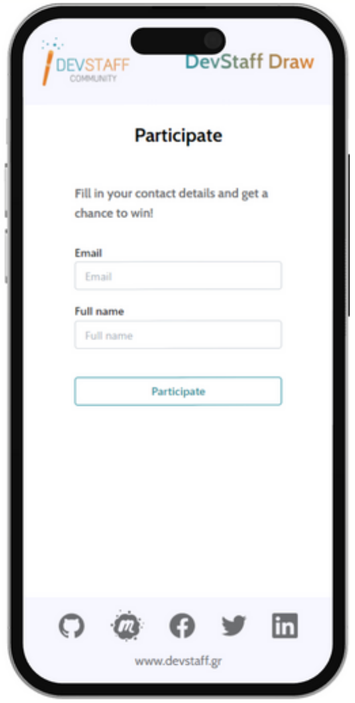
APPLICATION DEMO

DevStaff Draw App

Application

ROUTES





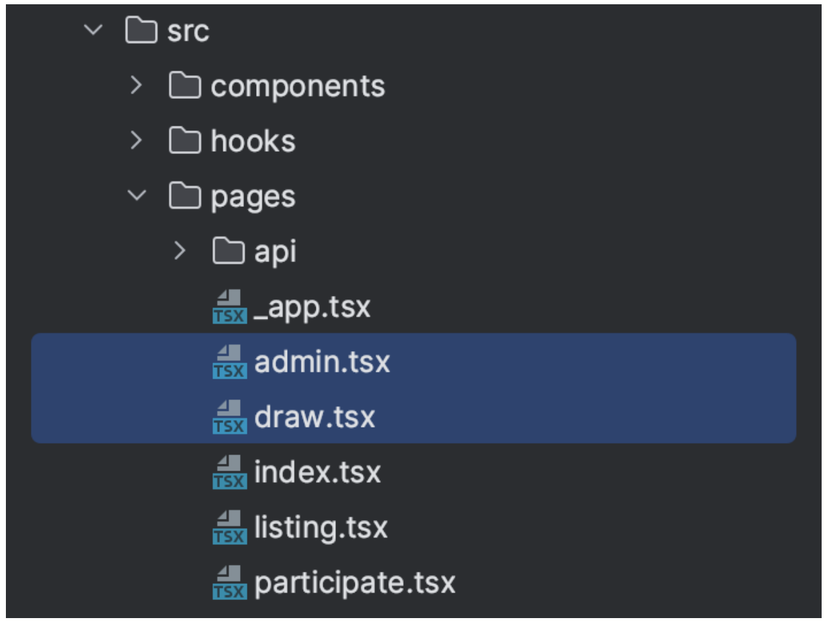
https://{host}/**participate**

https://{host}/

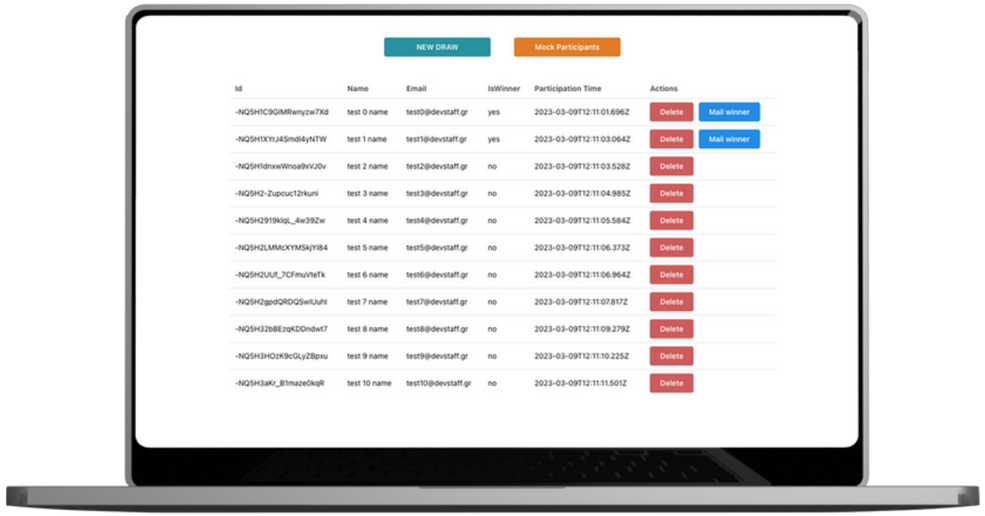
https://{host}/**listing**

Application

ADMIN ROUTES



https://{host}**/draw**



https://{host}**/admin**

Home page

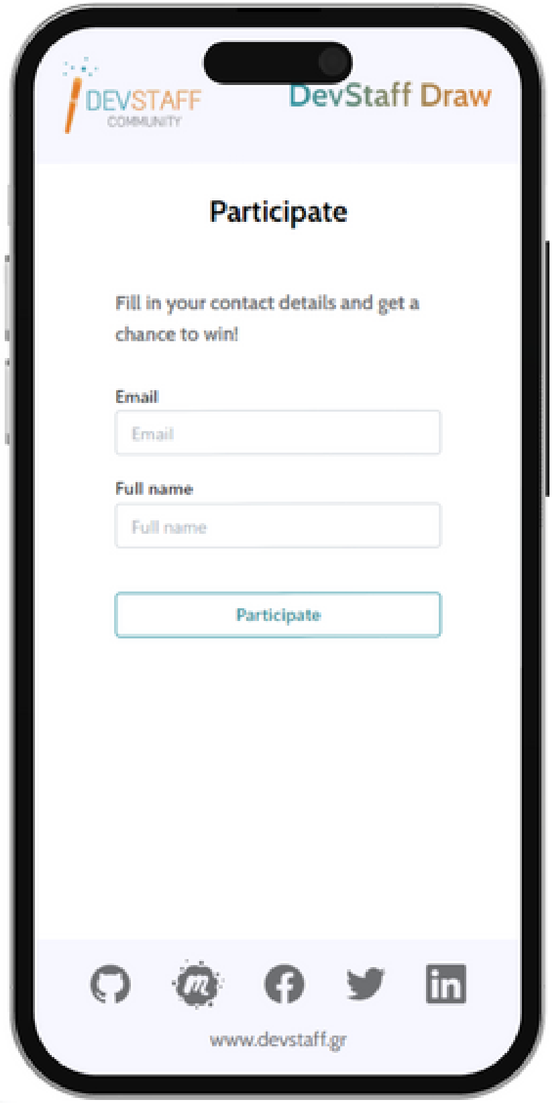
STYLED-COMPONENTS

```tsx
import styled, { css } from 'styled-components';
import { Colors } from '@/src/constants';

const Button = styled.button<{
  buttonType: 'primary' | 'secondary'
}>`
  ...
  background-color: ${({ buttonType }) =>
    buttonType === 'primary' ?
      Colors.colorPrimary :
      Colors.colorSecondary
    };
`;

const Home = () => {
  return (
    <Layout>
      <Wrapper>
        <Link href="/participate">
          <Button buttonType="primary">Participate</Button>
        </Link>
        <Link href="/listing">
          <Button buttonType="secondary">List</Button>
        </Link>
      </Wrapper>
    </Layout>
  );
};
```

Participate Page

REACT-HOOK-FORM

```
import { registerNewParticipant } from '@/src/api';
import { useForm } from 'react-hook-form';
...

const Participate = () => {
  const { register, handleSubmit, formState: { errors }} = useForm();

  return (
    ...
    <Heading>Participate</Heading>
    <Paragraph>Fill in your contact details and get a chance to win!</Paragraph>
    <Form id="participate-form" onSubmit={handleSubmit(formData =>
        registerParticipant({
          name: formData.fullName,
          email: formData.email
        })
    )}>
      <TextInput type="text" label="Email"
        {...register('email', {
          required: 'This field is required',
          pattern: { value: emailRegex(),
          message: 'Please provide a valid email address' }
        })}/>
      {errors.email ? <ErrorMessage>{errors.email.message}</ErrorMessage> : null}

      <TextInput type="text" label="Full name"
        {...register('fullName', { required: 'This field is required' })}
      />
      {errors.fullName ? <ErrorMessage>{errors.fullName.message}</ErrorMessage> : null}

      <Button type="submit">Participate</Button>
    </Form>
  );
};
```



Listing Page

REACT-QUERY

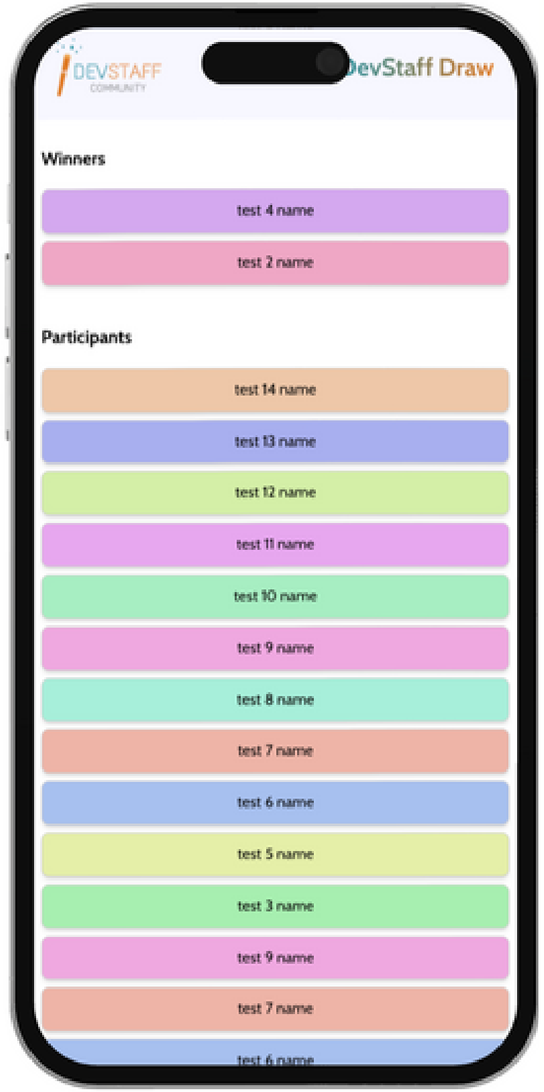
```
const { data } = useQuery(['participants'], getParticipants, {
    refetchInterval: 1500,
});
```

Listing Page

REACT-QUERY

Modify react-query response

```
const { data } = useQuery(['participants'], getParticipants, {
    refetchInterval: 1500,
    select: data => ({
      winners: data.filter(participant => participant.isWinner),
      participants: data.filter(participant => !participant.isWinner)
    })
});
```



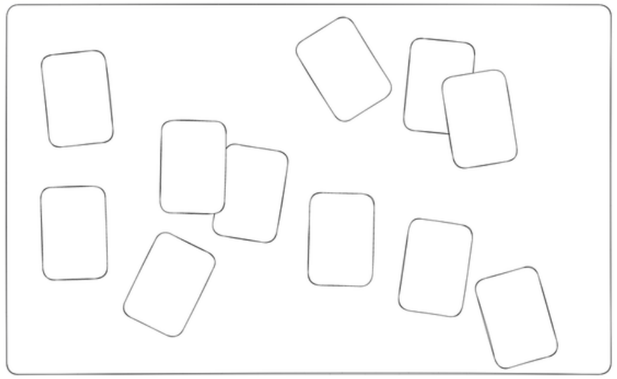
The Draw

Requirements

- Select winner count
- Initiate the draw
- Display winners

The Draw

STORYBOARD



Animation Steps

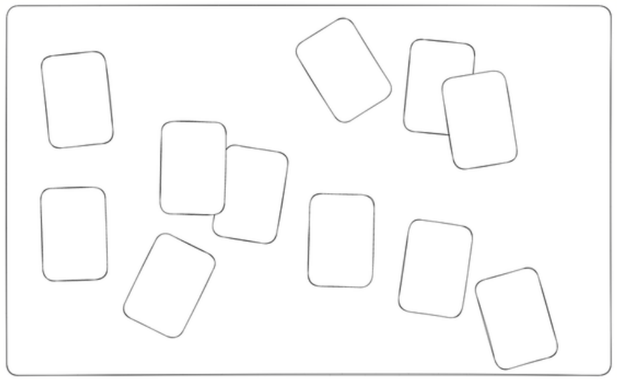
- Randomly spread cards around

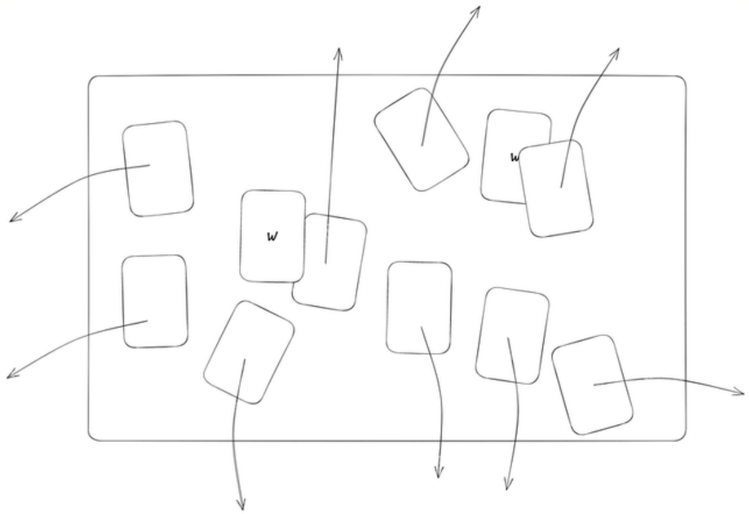
The Draw

STORYBOARD

Animation Steps

- Randomly spread cards around
- **(Draw is triggered)**
- Sequentially remove losing cards



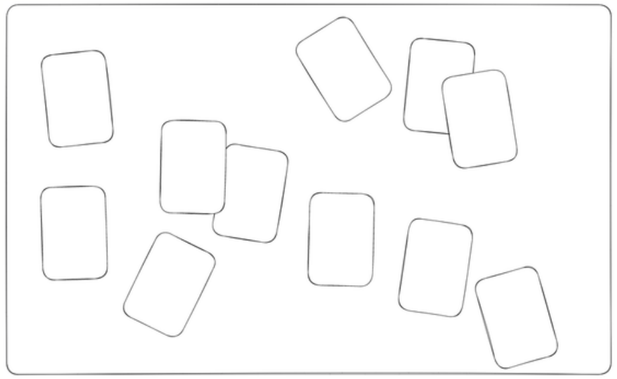


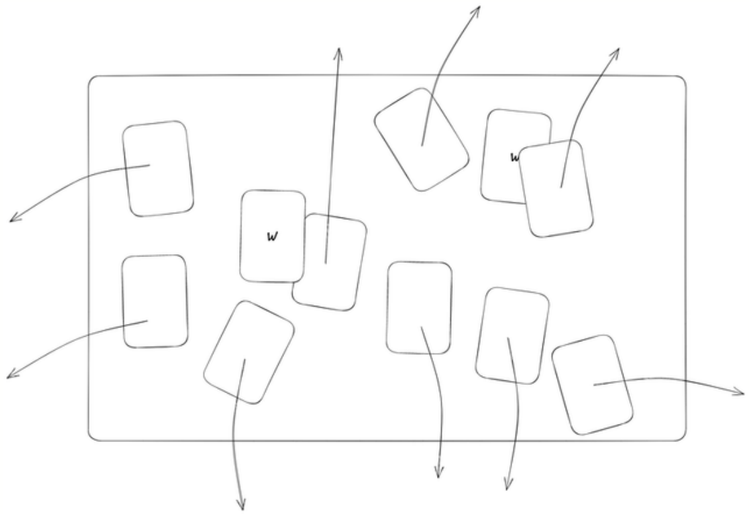
The Draw

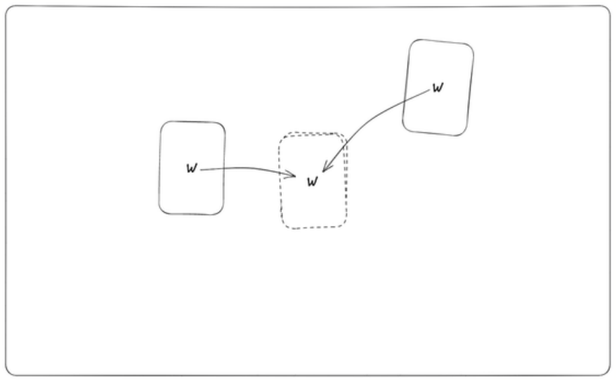
STORYBOARD

Animation Steps

- Randomly spread cards around
- **(Draw is triggered)**
- Sequentially remove losing cards
- Move winning cards to the center





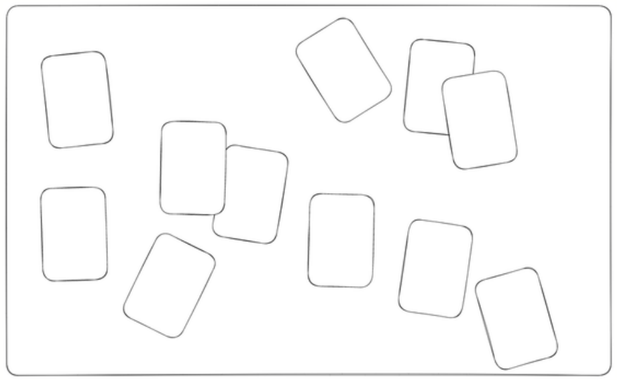


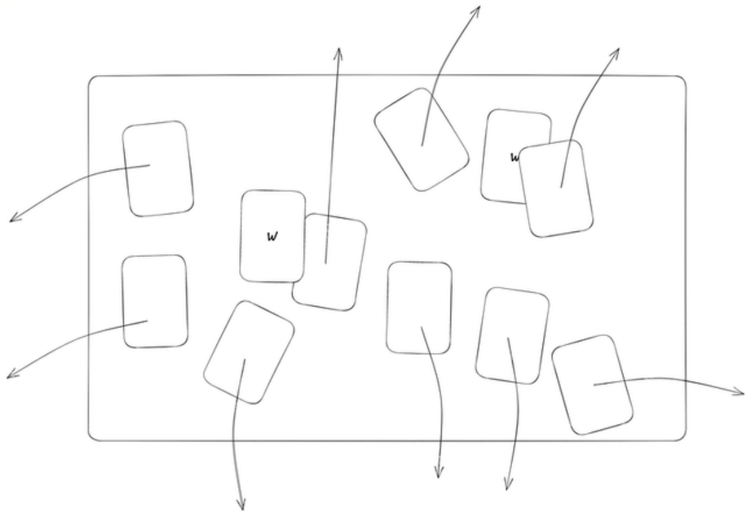
The Draw

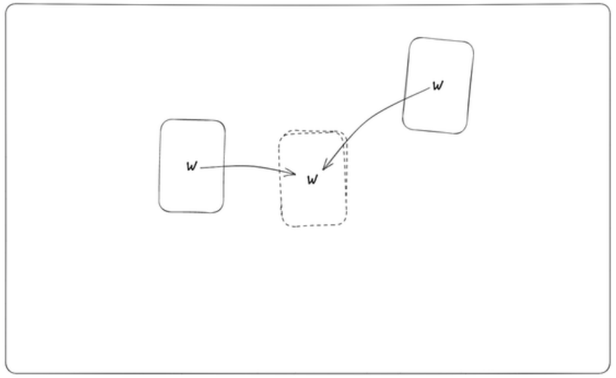
STORYBOARD

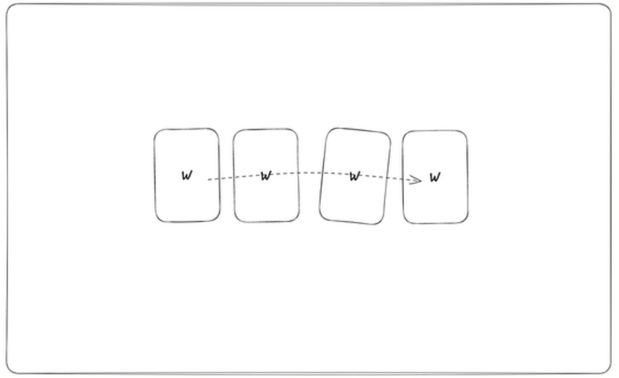
Animation Steps

- Randomly spread cards around
- **(Draw is triggered)**
- Sequentially remove losing cards
- Move winning cards to the center
- Fan winning cards









The Draw

STEP 1 - SPREAD THE CARDS AROUND


```
{participants.map(participant ⇒ (
  <motion.div
    initial={{
      x: getRandomPointX(),
      y: getRandomPointY(),
      rotate: getRandomAngle(15)
    }}
  >
    {participant.name}
  </motion.div>
))}
```

The Draw

STEP 2 - REMOVE NON-WINNING CARDS


```jsx
const controls = useAnimationControls();

const handleDraw = (winnerIds) => {
  controls.start(i =>
    !winnerIds.includes(participants[i].id)
    ? {
        x: getRandomExternalPointX(),
        y: getRandomExternalPointY(),
      }
    : {}
  );
}

{participants.map((participant, i) => (
  <motion.div
    custom={i}
    animate={controls}
    initial={{
      x: getRandomPointX(),
      y: getRandomPointY(),
      rotate: getRandomAngle(15)
    }}
  >
    {participant.name}
  </motion.div>
))}
```

The Draw

STEP 2 - REMOVE NON-WINNING CARDS


```
const controls = useAnimationControls();

const handleDraw = (winnerIds) => {
  controls.start(i =>
    !winnerIds.includes(participants[i].id)
    ? {
        x: getRandomExternalPointX(),
        y: getRandomExternalPointY(),
        transition: {
          delay: (i / winnerIds.length) * 3,
          duration: 0.3
        }
      }
    : {}
  );
}

{participants.map((participant, i) => (
  <motion.div
    custom={i}
    animate={controls}
    initial={{
      x: getRandomPointX(),
      y: getRandomPointY(),
      rotate: getRandomAngle(15)
    }}
  >
    {participant.name}
  </motion.div>
))}
```

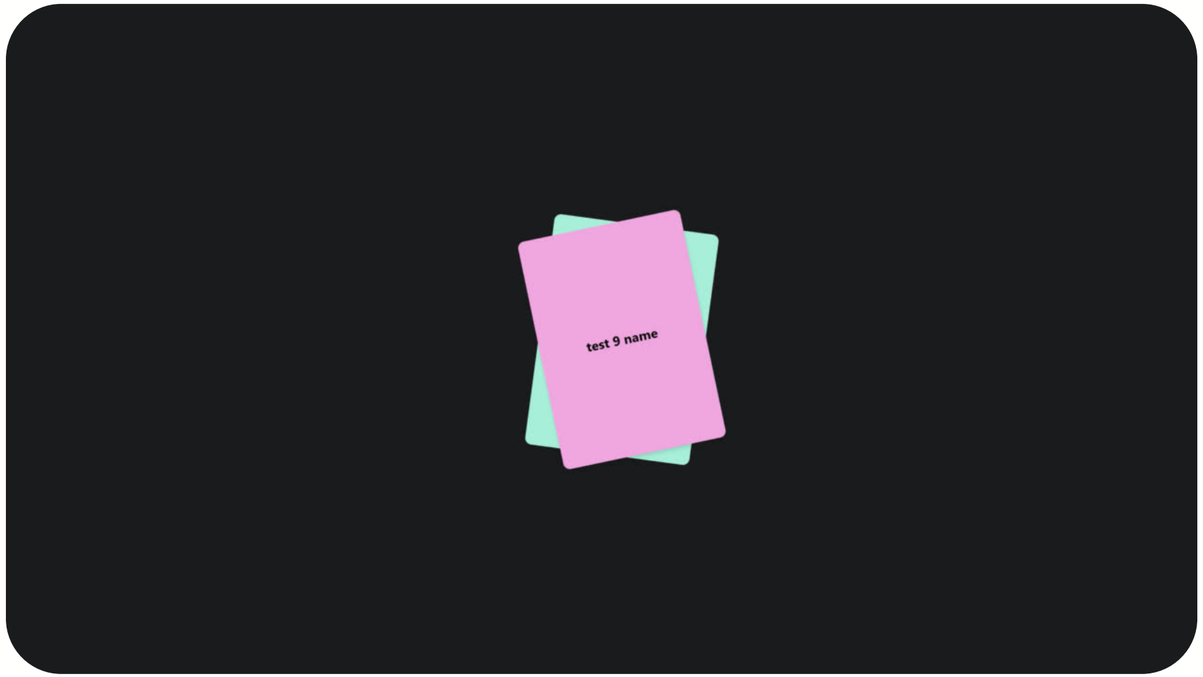
The Draw

STEP 3 - GATHER WINNERS TO THE CENTER


```
const handleDraw = async (winnerIds) => {
  // Remove non-winning cards
  await controls.start(...);

  controls.start(i =>
    winnerIds.includes(participants[i].id)
    ? { x: 0, y: 0 }
    : {}
  );
}

...
```

The Draw

STEP 4 - FAN CARDS




```
const handleDraw = async (winnerIds) => {
  // Remove non-winning cards
  await controls.start(...);

  // Gather winners to the center
  await controls.start(...);

  controls.start(i =>
    winnerIds.includes(participants[i].id)
    ? {
        x: getFinalPosition(i),
        rotate: getRandomAngle(5)
      }
    : {}
  );
}

...
```

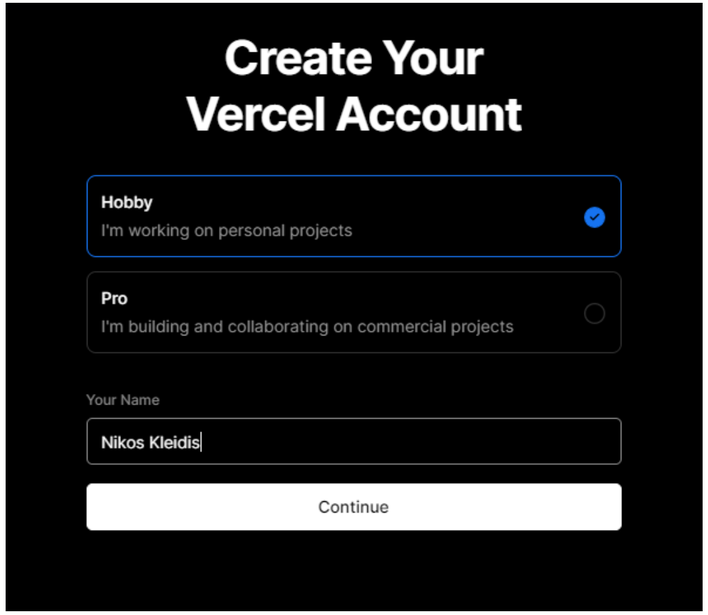
Deployment

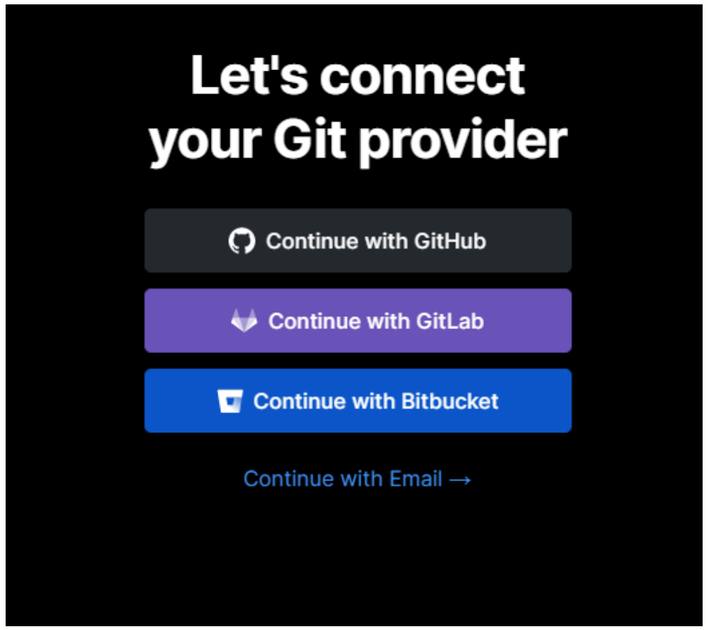
Deploying a Next.js app to Vercel

Deployment

Deploying a Next.js app to Vercel

- Create a Vercel account

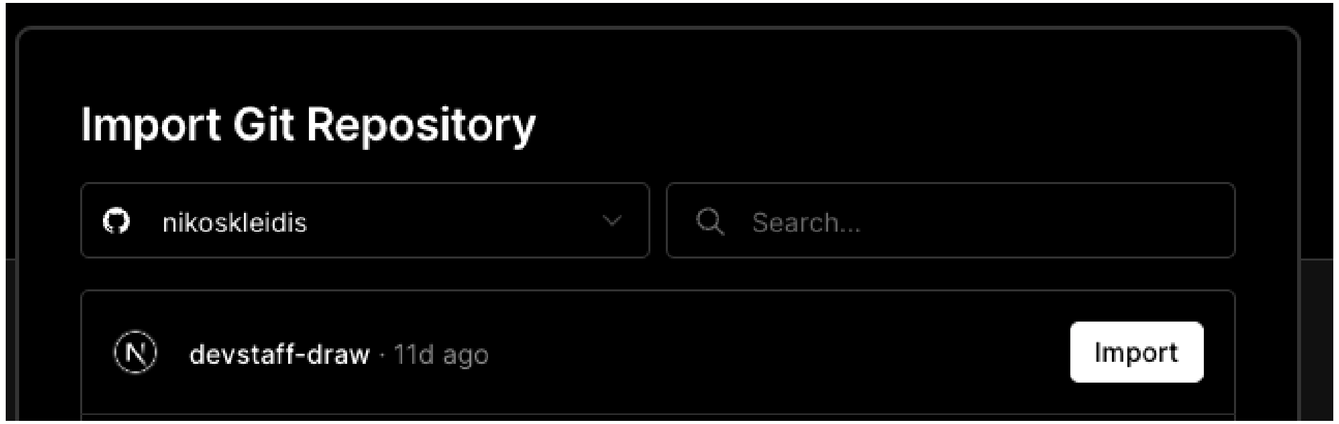




Deployment

Deploying a Next.js app to Vercel

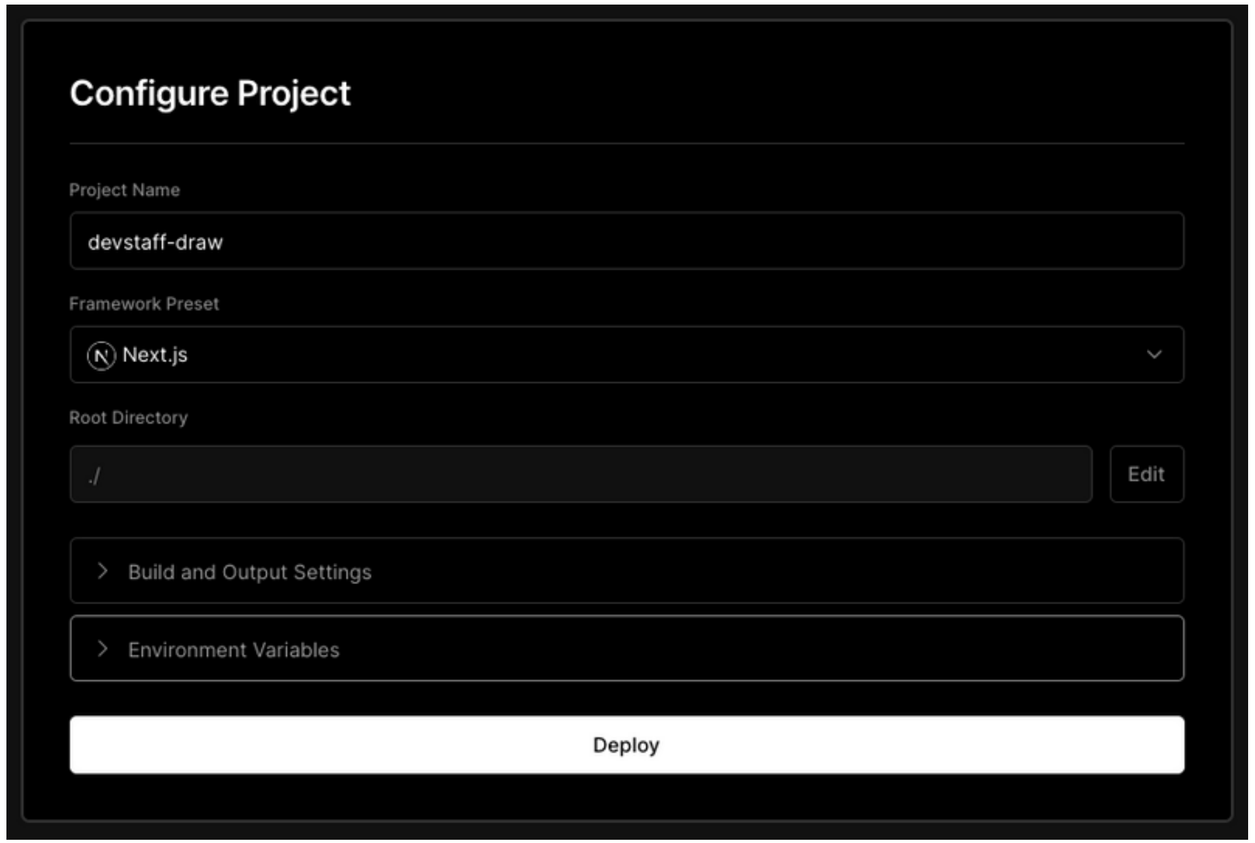
- Create a Vercel account
- Link the repository



Deployment

Deploying a Next.js app to Vercel

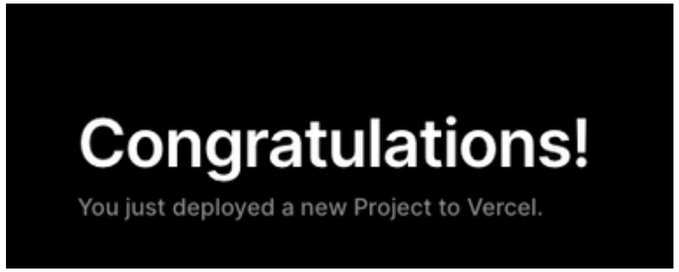
- Create a Vercel account
- Link the repository
- Configure the project



Deployment

Deploying a Next.js app to Vercel

- Create a Vercel account
- Link the repository
- Configure the project
- **Vercel** will now automatically build and deploy the app any time code is pushed to the relevant branch



Special Thanks!

George Detorakis

Questions?

Thank You!