React Hook Form

Building forms with React Hook Form and Zod.

Docs

Forms are tricky. They are one of the most common things you'll build in a web application, but also one of the most complex.

Well-designed HTML forms are:

- Well-structured and semantically correct.
- Easy to use and navigable (keyboard).
- Accessible with ARIA attributes and proper labels.
- Has support for client and server side validation.
- Well-styled and consistent with the rest of the application.

In this guide, we will take a look at building forms with `react-hook-form` and `zod`. We're going to use a `<FormField>` component to compose accessible forms using Radix UI components.

Features

The `<Form />` component is a wrapper around the `react-hook-form` library. It provides a few things:

- Composable components for building forms.
- A `<FormField />` component for building controlled form fields.
- Form validation using `zod`.
- Handles accessibility and error messages.
- Uses `React.useId()` for generating unique IDs.
- Applies the correct `aria` attributes to form fields based on states.
- Built to work with all Radix UI components.
- Bring your own schema library. We use `zod` but you can use anything you want.
- **You have full control over the markup and styling.**

Anatomy

```tsx
<Form>
  <FormField
    control={...}
    name="..."
    render={() => (
      <FormItem>
        <FormLabel />
        <FormControl>
          {/* Your form field */}
        </FormControl>
        <FormDescription />
        <FormMessage />
      </FormItem>
    )}
  />
</Form>
```

Example

```tsx
const form = useForm()

<Form{...form}>
  control={form.control}
  name="username"
  render={({ field }) => (
    <FormItem>
      <FormLabel>Username</FormLabel>
      <FormControl>
        <Input placeholder="shadcn" {...field} />
      </FormControl>
      <FormDescription>This is your public display name.
</FormDescription>
      <FormMessage />
    </FormItem>
  )}
/>
```

Installation

CLI Manual

1 **Command**

```
npm yarn bun

pnpm dlx shadcn@latest add form
```

Usage

1 **Create a form schema**

Define the shape of your form using a Zod schema. You can read more about using Zod in the Zod documentation.

```
"use client"

import { z } from "zod"

const formSchema = z.object({
  username: z.string().min(2).max(50),
})
```

2 **Define a form**

Use the `useForm` hook from `react-hook-form` to create a form.

```
"use client"

import { zodResolver } from "@hookform/resolvers/zod"
import { useForm } from "react-hook-form"
import { z } from "zod"

const formSchema = z.object({
  username: z.string().min(2, {
    message: "Username must be at least 2 characters.",
  }),
})

export function ProfileForm() {
  // 1. Define your form.
  const form = useForm<z.infer<typeof formSchema>>({
    resolver: zodResolver(formSchema),
    defaultValues: {
      username: "",
    },
  })

  // 2. Define a submit handler.
  function onSubmit(values: z.infer<typeof formSchema>) {
    // Do something with the form values.
    // ✅ This will be type-safe and validated.
    console.log(values)
  }
}
```

Since `FormField` is using a controlled component, you need to provide a default value for the field. See the React Hook Form docs to learn more about controlled components.

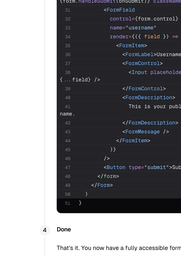
3 **Build your form**

We can now use the `<Form />` components to build our form.

```
"use client"

import { zodResolver } from "@hookform/resolvers/zod"
import { useForm } from "react-hook-form"
import { z } from "zod"

import { Button } from "@/components/ui/button"
import {
  Form,
  FormControl,
  FormDescription,
  FormField,
  FormItem,
  FormLabel,
  FormMessage,
} from "@/components/ui/form"
import { Input } from "@/components/ui/input"

const formSchema = z.object({
  username: z.string().min(2, {
    message: "Username must be at least 2 characters.",
  }),
})

export function ProfileForm() {
  // ...

  return (
    <Form {...form}>
      <form onSubmit={form.handleSubmit(onSubmit)} className="space-y-8">
        <FormField
          control={form.control}
          name="username"
          render={({ field }) => (
            <FormItem>
              <FormLabel>Username</FormLabel>
              <FormControl>
                <Input placeholder="shadcn" {...field} />
              </FormControl>
              <FormDescription>
                This is your public display name.
              </FormDescription>
              <FormMessage />
            </FormItem>
          )}
        />
        <Button type="submit">Submit</Button>
      </form>
    </Form>
  )
}
```

4 **Done**

That's it. You now have a fully accessible form that is type-safe with client-side validation.



Examples

See the following links for more examples on how to use the `<Form />` component with other components:

- Checkbox
- Date Picker
- Input
- Radio Group
- Select
- Switch
- Textarea
- Combobox