Complex Form Handling

Prerequisites:

- Next.js Version: $14.2.3$ or Later
- Shadcn/ui Library

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

Well-designed HTML forms are:

- Well-structured and semantically correct
- Easy to use and navigate (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

Shadcn Form Component (Base)

In here we will use Shadcn Form Component. 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 message
- 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
- **Schema library** using `zod`

Anatomy

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

Example

```
const form = useForm()

<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.</
            <FormMessage/>
        </FormItem>
    )}
/>
```

To add this Schema and Annotation, we need to install Shadcn but also wie need to add the form, also using a Command:

```
npx shadcn@latest add form
```

Schritt-für-Schritt zur ersten Form

Create a form schema

Define a shape of your form using a **Zod** schema. With Zod you declare a validator once and Zod will automatically infer the static TypeScript type. In this example we are going to create an object Schema, describing our Form.

```
"use client"

import { z } from "zod"

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

Now we have here an Object Form Schema containing an String (username), which takes in a String with a minimum of 2 characters and a maximum of 50 characters.

Define a form

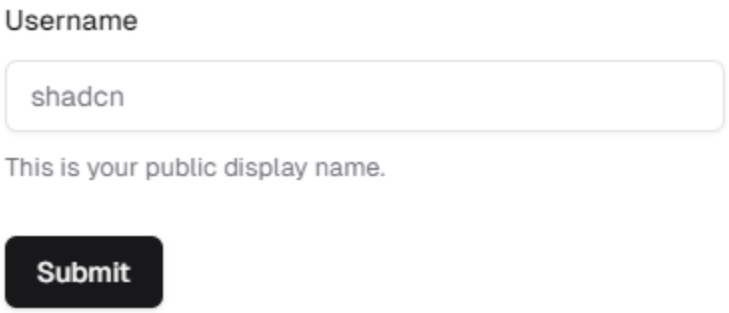
Use the `useForm` hook from `react-hook-form` to create a form.

```
"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 our form the Type will be above in formSchema
    const form = useForm<z.infer<typeof formSchema>>({
        resolver: zodResolver(formSchema),
        defaultValues: {
            username: "",
        },
    })

    // 2. Define a sumbit handler
    function onSubmit(values: z.infer<typeof formSchema>) {
        // Do something with the form values
        console.log(values)
    }

}
```

Build your form

We can now use the `<Form/>` components (Shadcn) to build our form

```
//..... Code

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



Abstract Form Schema

We can abstract the Form Schema so it is a export function.

1. **Create a `validation.ts` inside the `lib` Folder**

   In here we can import **Zod** and define the Schemas

   ```
   // ./lib/validation.ts

   export const UserFormValidation = z.object({
       name: z.string().min(2, "Name must be at least 2 characte
       email: z.string().email("Invalid email address")
       phone: z.string().refine((phone) => /^\+\d{10,15}$/.test(
   });
   ```

   After defining the Schema and declaring it as a Export, we can use it we Define our Form

2. **Define our Form**

```
// e.x ./components/forms/PatientForm.tsx
import { zodResolver } from "@hookform/resolvers/zod";
import { useForm } from "react-hook-form";
import { z } from "zod";
import { UserFormValidation } from "@/lib/validation"; // Ou

const PatientForm = () => {

    const form = useForm<z.infer<typeof UserFormValidation>>(
        resolver: zodResolver(UserFormValidation),
        defaultValues: {
            name: "",
            email: "",
            phone: "",
        },
    })

    async function onSubmit({name,email,phone}: z.infer<typeo
        // Code
    }

    return (
        ...
    )
}
```

Custom Form Fields

We can also abstract the usage of FormControls (Inputs), so in large forms, we don’t have that much code. With Custom Form Fields we can abstract it and make it reusable.

Here we have two functions:

1. **RenderField** (this determines, which Type of Input we will have (e.g Date, Checkbox etc.))
2. **CustomFormField** (which takes the RenderField and builds the basic Components around it)

1. **Define Props of Custom Form Fields**

```
interface CustomProps {
    control: Control<any>,                                  //
    fieldType: FormFieldType,                               //
    name: string,                                           //
    label?: string,                                         //
    placeholder?: string,                                   //
    iconSrc?: string,                                       //
    iconAlt?: string,                                       //
    disabled?: boolean,                                     //
    dateFormat?: string,                                    //
    showTimeSelect?: boolean,                               //
    children?: React.ReactNode,                             //
    renderSkeleton?: (field: any) => React.ReactNode,       //
}
```

Note these, will also be used for RenderField

2. **Define the Main Component → CustomFormField**

```
const CustomFormField = (props: CustomProps) => {

        // Destructering the props
        const {control, fieldType, name, label, placeholder,
```

```
    return(
        <FormField
            control={control}
            name={name}
            render={({field}) => (
                <FormItem className="flex-1">
                    // Only render the Label when it is not a
                    {fieldType !== FormFiledType.CHECKBOX &&
                        <FormLabel>{label}</FormLabel>
                    )}

                    // Here we the Input based on whih field
                    <RenderField field={field} props={props}/

                    <FormMessage className="shad-error"/>
                </FormItem>
            )}
        />
    )
}
```

3. **Define FormFieldType to determine which Types we have**

```
// Somewhere else in the Code
// It is easier to type in wrongly the Stringtext, then a sir
// Also a Variable if wrongly typed will indicate a SyntaxErr
export enum FormFieldType {
  INPUT = 'input',
  TEXTAREA = 'textarea',
  PHONE_INPUT = 'phoneInput',
  CHECKBOX = 'checkbox',
  DATE_PICKER = 'datePicker',
  SELECT = 'select',
```

```
  SKELETON = 'skeleton',
}
```

4. **Define our RenderField Input property based on specifc FieldType**

```
import { FormFieldType } from "./forms/PatientForm";

const RenderField = ({ field, props } : {field: any, props: C

  const {fieldType, iconSrc, iconAlt, placeholder, showTimeSe

  // Switch statement to determine the rendering of each fiel
  switch (props.fieldType) {
    case FormFieldType.INPUT: // FormFieldType is a enum defi
      return (
        <div className="flex rounded-md border border-dark-50
            {iconSrc && (
              <Image
                src={iconSrc}
                height={24}
                width={24}
                alt={iconAlt || 'icon'}
                className="ml-2"
              />
            )}
            <FormControl>
              <Input
                placeholder={placeholder}
                {...field}                          // Spread op
                className="shad-input border-0"
              />
            </FormControl>
        </div>
      )
```

```
    case FormFieldType.PHONE_INPUT:
      return (
        <FormControl>
          <PhoneInput
            defaultCountry="US"
            placeholder={placeholder}
            international
            withCountryCallingCode
            value={field.value}
            onChange={field.onChange}
            className="input-phone"
          />
        </FormControl>
      )
    case FormFieldType.DATE_PICKER:
      return (
        <div className="flex rounded-md border border-dark-5(
          <Image
            src="/assets/icons/calendar.svg"
            height={24}
            width={24}
            alt="calendar"
            className="ml-2"
          />
          <FormControl>
            <DatePicker
              selected={field.value}
              onChange={(date) => field.onChange(date)}
              dateFormat={dateFormat ?? 'MM/dd/yyyy'}
              showTimeSelect={showTimeSelect ?? false}
              timeInputLabel="Time:"
              wrapperClassName="date-picker"
            />
          </FormControl>
        </div>
      )
```

```
      case FormFieldType.SKELETON:
        return (
          renderSkeleton ? renderSkeleton(field) : null
        )
      case FormFieldType.SELECT:
        return (
          <FormControl>
            <Select onValueChange={field.onChange} defaultValue
              <FormControl>
                <SelectTrigger className="shad-select-trigger">
                  <SelectValue placeholder={placeholder}/>
                </SelectTrigger>
              </FormControl>
              <SelectContent className="shad-select-content">
                {props.children}
              </SelectContent>
            </Select>
          </FormControl>
        )
      case FormFieldType.TEXTAREA:
        return (
          <FormControl>
            <Textarea
              placeholder={placeholder}
              {...field}
              className="shad-textArea"
              disabled={props.disabled}
            />
          </FormControl>
        )
      case FormFieldType.CHECKBOX:
          return (
            <FormControl>
              <div className="flex items-center gap-4">
                <Checkbox
                  id={props.name}
```

```
                checked={field.value}
                onCheckedChange={field.onChange}
              />
              <label htmlFor={props.name} className="checkbo
                {props.label}
              </label>
            </div>
          </FormControl>
        )
    default:
      break;
  }
}
```

Now we can call a custom FormField in that way, where our Form is defined:

```
// Code...
return (
    <Form {...form}>
        <form onSubmit={form.handleSubmit(onSubmit)} className="
        <section className="mb-12 space-y-4">
          <h1 className="header">Hi there ✌️</h1>
          <p className="text-dark-700">Schedule your first appo
        </section>

            {/* Here we call our Abstraktion of FormFields */}
            <CustomFormField
          fieldType={FormFieldType.INPUT}
          control={form.control}
          name="name"
          label="Full name"
          placeholder="John Doe"
          iconSrc="/assets/icons/user.svg"
          iconAlt="user"
```

```
         />
         ...
        <CustomFormField
           fieldType={FormFieldType.PHONE_INPUT}
           control={form.control}
           name="phone"
           label="Phone number"
           placeholder="(555) 123-4567"
         />


                <SubmitButton isLoading={isLoading}>
           Get started
         </SubmitButton>
          </from>
   </Form>
)
```