index.css

```
html,
body {
 display: flex;
 background-color: #333333;
 margin: 0 auto;
 padding: 30px;
 font-family: monospace;
 font-size: 16px;
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8" />
   <link rel="icon" type="image/svg+xml" href="/vite.svg" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>Vite + React + TS</title>
 </head>
 <body>
   <div id="root"></div>
   <script type="module" src="/src/main.tsx"></script>
 </body>
</html>
```

App.tsx

```
import React from "react";
import FormRef from "./components/FormRef";
import FormState from "./components/FormState";
import HookForm from "./components/HookForm";
import Validation from "./components/Validation";
import ZodValidation from "./components/Zod";
import styled from "styled-components";

const Container = styled.div`
 padding: 30px;
 background: cyan;
```

```
 border-radius: 10px;
 width: 375px;
`;

const InnerContainer = styled.div`
 height: 300px;
 font-size: 18px;
 font-weight: bold;
`;

const Describe = styled.p`
 font-size: 13px;
 margin: 0;
 color: purple;
`;

const App = () => {
 return (
   <Container>
     <InnerContainer>
       <Describe>(For with useRef)</Describe>
       <FormRef />
     </InnerContainer>

     <InnerContainer>
       <Describe>(Form with useState)</Describe>
       <FormState />
     </InnerContainer>

     <InnerContainer>
       <Describe>(Form with HookForm)</Describe>
       <HookForm />
     </InnerContainer>

     <InnerContainer>
       <Describe>(Form with HookForm and Validation)</Describe>
       <Validation />
     </InnerContainer>

     <InnerContainer>
       <Describe>(Form with HookForm and Zod)</Describe>
       <ZodValidation />
     </InnerContainer>
   </Container>
 );
};

export default App;
```

FormRef.tsx

```
import React, { FormEvent, useRef } from "react";
import styled from "styled-components";

/*
* In the submit:
   - must prevent the default behavior of posting to the server by:
       event.preventDefault()

* to create the form element with Bootstrap shortcuts, inside the <form>,
       we typed and tabbed for auto:
    div.mb-3>label.form-label+input.form-control
    this created the div element below via bootstrap
    - div with a class of mb-3 (margin-bottom: 3px)
    - label with a class of form-label
    - input with a class of form-control

* set the htmlFor to the id of the input so that when selected, it autofocuses

* second form field: div.mb-3>label.form-label+input[type=number].form-control
    - sets the type of input to number

* the button: button.btn.btn-primary[type=submit]
    - two classes, and type is submit

* useRef is a built-in hook in React that allows you to create a mutable object
that persists for the lifetime of the component. This object can hold a
reference
to a DOM element or any other value that needs to be persisted across renders.
   - When using useRef with TypeScript, you'll typically want to define the type
   of the value that the ref will hold. For example, if you're using useRef to
   hold a reference to a DOM element, you might define the type as
   React.RefObject<HTMLDivElement>. This tells TypeScript that the ref will hold
   a reference to a div element.
   - Here's an example of how you might use useRef in a TypeScript component:

           import { useRef } from 'react';

           function MyComponent() {
           const inputRef = useRef<HTMLInputElement>(null);

           const handleClick = () => {
               if (inputRef.current) {
               inputRef.current.focus();
               }
```

```
            };

            return (
                <div>
                <input type="text" ref={inputRef} />
                <button onClick={handleClick}>Focus Input</button>
                </div>
            );
            }

   - We're then using the current property of the ref to check if the input
element
   exists, and if so, calling its focus method to bring it into focus.
   - HTMLINputElement tells UseRef that the the kind of input to expect. This is
   standard to all browsers.
   - other elements that can be referenced like lists, etc.
   - every ref object has to be initialized as null
  */

const FormRef = () => {
 const nameRef = useRef<HTMLInputElement>(null);
 const ageRef = useRef<HTMLInputElement>(null);
 const person = { name: "", age: 0 };

 const handleSubmit = (event: FormEvent) => {
   event.preventDefault();
   if (nameRef.current !== null) person.name = nameRef.current.value;
   if (ageRef.current !== null) person.age = parseInt(ageRef.current.value);
   console.log(person);
 };

 return (
   <form onSubmit={handleSubmit}>
     <div className="mb-3">
       <label htmlFor="name" className="form-label">
         Name:
       </label>
       <input ref={nameRef} id="name" type="text" className="form-control" />
     </div>
     <div className="mb-3">
       <label htmlFor="age" className="form-label">
         Age:
       </label>
       <input ref={ageRef} id="age" type="number" className="form-control" />
     </div>
     <button className="btn btn-primary" type="submit">
       Submit
     </button>
```

```
    </form>
  );
};

export default FormRef;
```

FormState.tsx

```
import React, { FormEvent, useState } from "react";
import styled from "styled-components";

const FormState = () => {
  const [person, setPerson] = useState({
    name: "",
    age: "",
  });

  const handleSubmit = (event: FormEvent) => {
    event.preventDefault();
    console.log(person);
  };

  return (
    <form onSubmit={handleSubmit}>
      <div className="mb-3">
        <label htmlFor="name" className="form-label">
          Name:
        </label>
        <input
          onChange={(event) =>
            setPerson({ ...person, name: event.target.value })
          }
          value={person.name}
          id="name"
          type="text"
          className="form-control"
        />
      </div>
      <div className="mb-3">
        <label htmlFor="age" className="form-label">
          Age:
        </label>
        <input
          onChange={(event) =>
            setPerson({ ...person, age: event.target.value })
          }
          value={person.age}
```

```
        id="age"
        type="number"
        className="form-control"
      />
    </div>
    <button className="btn btn-primary" type="submit">
      Submit
    </button>
  </form>
 );
};

export default FormState;
```

HookForm.tsx

```
/* Hook Form - Form Attributes:
- use console.log(form); to get info on the methods
   * useForm(): This is the main function provided by React Hook Form that
initializes the form and returns a set of methods and properties that can be
used to manage the form state. It takes an optional configuration object as a
parameter, which can be used to customize the behavior of the form.

   * register(): This function is used to register an input field with the form.
It takes a name attribute as a parameter and returns an object with several
properties that can be used to bind the input field to the form state.

   * handleSubmit(): This function is used to handle the submission of the form.
It takes a callback function as a parameter, which is called when the form is
submitted.

   * errors: This is a property that contains the validation errors for each
input field in the form. It is an object where the keys are the names of the
input fields and the values are the error messages.

   * watch(): This function is used to watch for changes to specific input
fields in the form. It takes an optional parameter that specifies the name of
the input field to watch and returns the current value of the input field.

   * setError(): This function is used to manually set validation errors for an
input field. It takes the name of the input field and the error message as
parameters.

   * clearErrors(): This function is used to clear the validation errors for an
   input field. It takes the name of the input field as a parameter.
*/
```

```
import React, { FormEvent, useState } from "react";
import { FieldValues, useForm } from "react-hook-form";
import styled from "styled-components";

const HookForm = () => {
 const form = useForm();
 const { register, handleSubmit } = useForm();

 console.log(register("name"));

 const onSubmit = (data: FieldValues) => console.log(data);

 return (
   <form onSubmit={handleSubmit(onSubmit)}>
     <div className="mb-3">
       <label htmlFor="name" className="form-label">
         Name:
       </label>
       <input
         {...register("name")}
         id="name"
         type="text"
         className="form-control"
       />
     </div>
     <div className="mb-3">
       <label htmlFor="age" className="form-label">
         Age:
       </label>
       <input
         {...register("age")}
         id="age"
         type="number"
         className="form-control"
       />
     </div>
     <button className="btn btn-primary" type="submit">
       Submit
     </button>
   </form>
 );
};

export default HookForm;
```

Validation.tsx

```
import React, { FormEvent, useState } from "react";
import { FieldValues, useForm } from "react-hook-form";
import styled from "styled-components";

const Text = styled.p`
 font-size: 13px;
 color: deeppink;
 margin: 5px;
`;

interface FormData {
 name: string;
 age: number;
}

const Validation = () => {
 // nested destructuring to access errors from formState
 const {
   register,
   handleSubmit,
   formState: { errors },
 } = useForm();

 console.log(register("name"));

 const onSubmit = (data: FieldValues) => console.log(data);

 return (
   <form onSubmit={handleSubmit(onSubmit)}>
     <div className="mb-3">
       <label htmlFor="name" className="form-label">
         Name:
       </label>
       <input
         {...register("name", { required: true, minLength: 3 })}
         id="name"
         type="text"
         className="form-control"
       />
       {/* if you get an error for the name field, and type = required, print
this */}
       {errors.name?.type === "required" && <Text>Name required.</Text>}
       {/* if you get an error for the name field, and type = minLength, print
this */}
       {errors.name?.type === "minLength" && (
         <Text>Name must have at least 3 characters.</Text>
```

```
      )}
    </div>
    <div className="mb-3">
      <label htmlFor="age" className="form-label">
        Age:
      </label>
      <input
        {...register("age")}
        id="age"
        type="number"
        className="form-control"
      />
    </div>
    <button className="btn btn-primary" type="submit">
      Submit
    </button>
  </form>
 );
};

export default Validation;
```

Zod.tsx

```
/* ZOD FORM VALIDATION: Zod.dev
-  defines a schema object using zod. The schema defines the shape of the form
data and the validation requirements for each field
-  z.object defines an object schema.
-  z.string() defines a string schema.
-  z.number() defines a number schema.
-  min sets a minimum length or value for the field.
-  message sets a custom error message for each field.
-  invalid_type_error sets a custom error message for the number schema if the
value is not a number.
- z.infer to define a FormData type:
- This type will be used in the useForm hook to infer the type of the form data
- Inside ZodValidation, the code uses the useForm hook to register form fields
and validate form data
- register is a function that registers a form field with react-hook-form.
- handleSubmit is a function that handles form submission.
- formState is an object that contains the form state and errors.
- resolver is a hook from @hookform/resolvers/zod that uses the zod schema to
validate form data
*/
```

```
import React, { FormEvent, useState } from "react";
import { FieldValues, useForm } from "react-hook-form";
import { z } from "zod";
import { zodResolver } from "@hookform/resolvers/zod";
import styled from "styled-components";

const Text = styled.p`
 font-size: 13px;
 color: deeppink;
 margin: 5px;
`;

/* returns an object that stores the shape and validation requirements
of the form
- defining a custom error message rather than the default
- notice the differences in the error message objects passed for name vs. age */

const schema = z.object({
 name: z.string().min(3, { message: "Name must be at least 3 characters." }),
 age: z
   .number({ invalid_type_error: "Age field is required." })
   .min(18, { message: "Age must be at least 18." }),
});

/* do not have to define the types in an interface, instead
can use z.infer and pass it to the types declared in the schema */
type FormData = z.infer<typeof schema>;

const ZodValidation = () => {
 // pass the resolver when declaring the useForm hook and pass schema object
 const {
   register,
   handleSubmit,
   formState: { errors, isValid },
 } = useForm<FormData>({ resolver: zodResolver(schema) });

 console.log(register("name"));

 const onSubmit = (data: FieldValues) => console.log(data);

 return (
   <form onSubmit={handleSubmit(onSubmit)}>
     <div className="mb-3">
       <label htmlFor="name" className="form-label">
         Name:
       </label>
       <input
         {...register("name")}
```

```
          id="name"
          type="text"
          className="form-control"
        />
        {errors.name && <Text>{errors.name.message}</Text>}
      </div>
      <div className="mb-3">
        <label htmlFor="age" className="form-label">
          Age:
        </label>
        {/* rather than parseInt on age, we use the unique attribute
valueAsNumber */}
        <input
          {...register("age", { valueAsNumber: true })}
          id="age"
          type="number"
          className="form-control"
        />
        {errors.age && <Text>{errors.age.message}</Text>}
      </div>
      <button disabled={!isValid} className="btn btn-primary" type="submit">
        Submit
      </button>
    </form>
  );
};

export default ZodValidation;
```