Front End

Class 16

October 13, 2021

Agenda

- Kahoot
- Review – Forms and User Input
- Studio Review plus Studio

ACADEMIND

Handling Forms & User Input

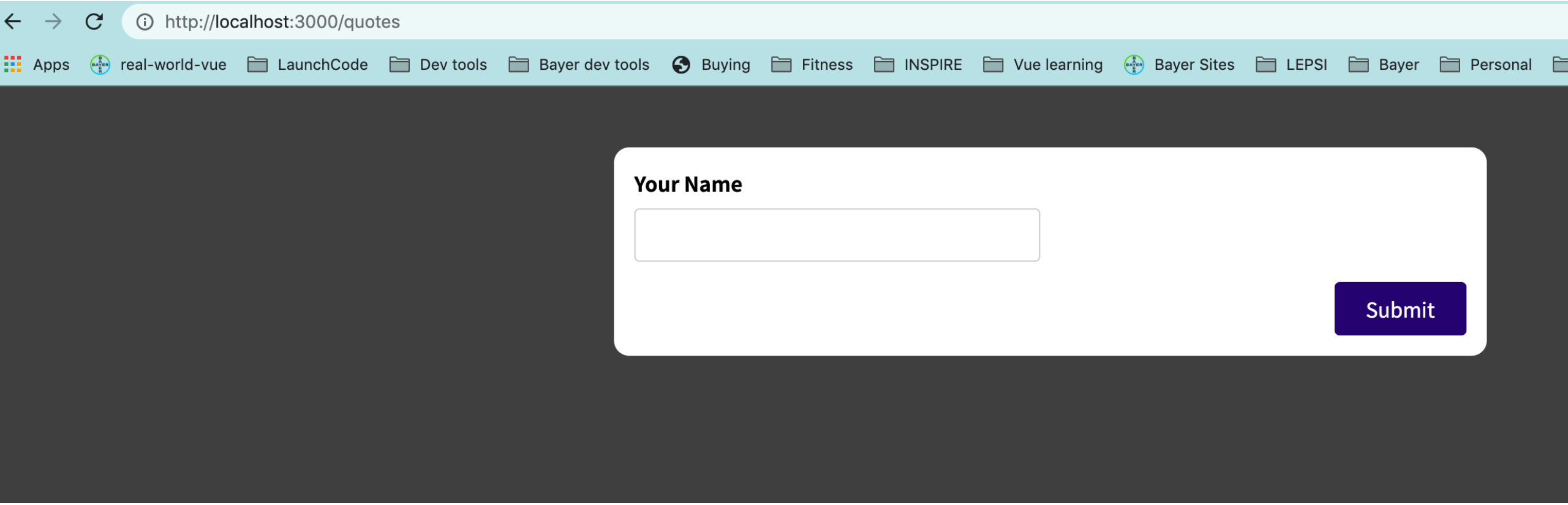
Working with Values, Validation & State

ACADEMIND

Module Content

- What's Complex About Forms?
- Handling Inputs & Forms with React
- Simplifications

```jsx
const SimpleInput = (props) => {
  return (
    <form>
      <div className='form-control'>
        <label htmlFor='name'>Your Name</label>
        <input type='text' id='name' />
      </div>
      <div className="form-actions">
        <button>Submit</button>
      </div>
    </form>
  );
};

export default SimpleInput;
```


http://localhost:3000/quotes
Apps
real-world-vue
LaunchCode
Dev tools
Bayer dev tools
Buying
Fitness
INSPIRE
Vue learning
Bayer Sites
LEPSI
Bayer
Personal
Your Name
Submit

What's Complex About Forms?

Forms and inputs can assume different states

One or more inputs are invalid

Output input-specific error messages & highlight problematic inputs

Ensure form can't be submitted / saved

All inputs are valid

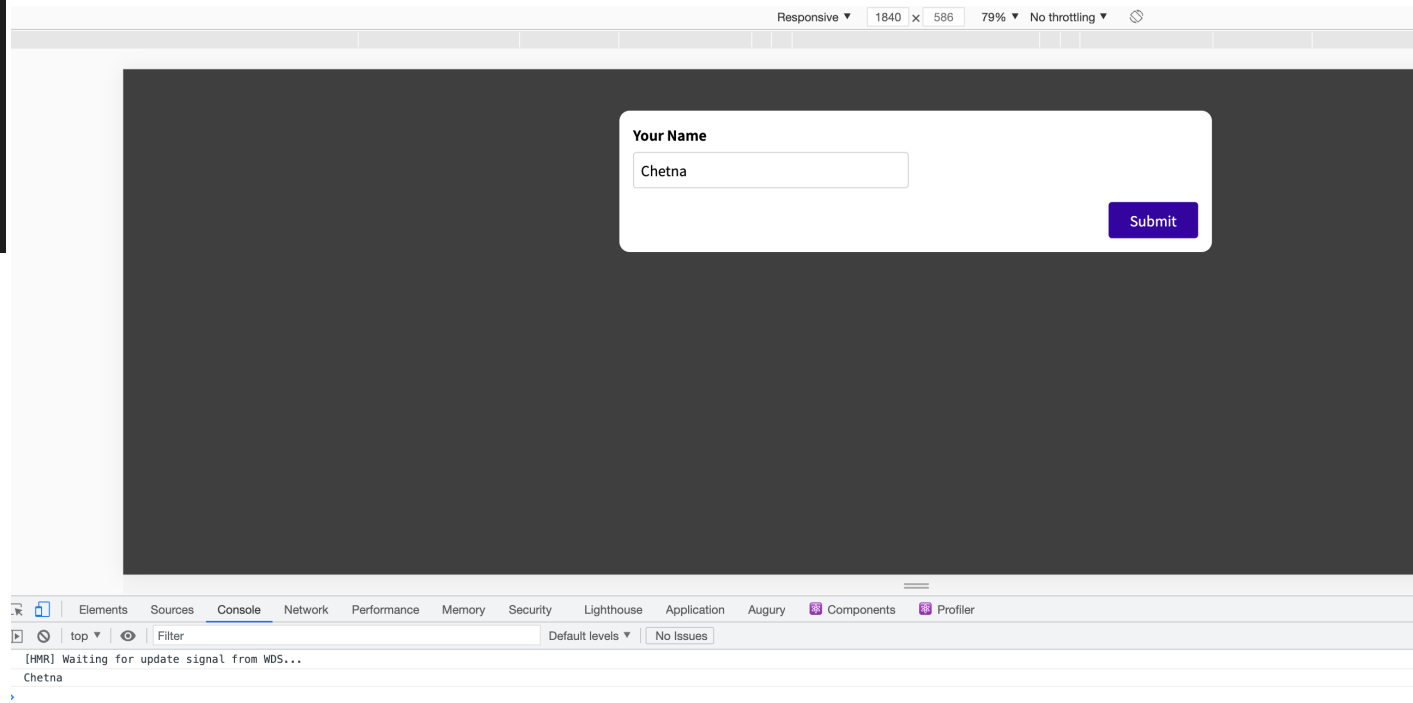
Allow form to be submitted / saved

ACADEMIND

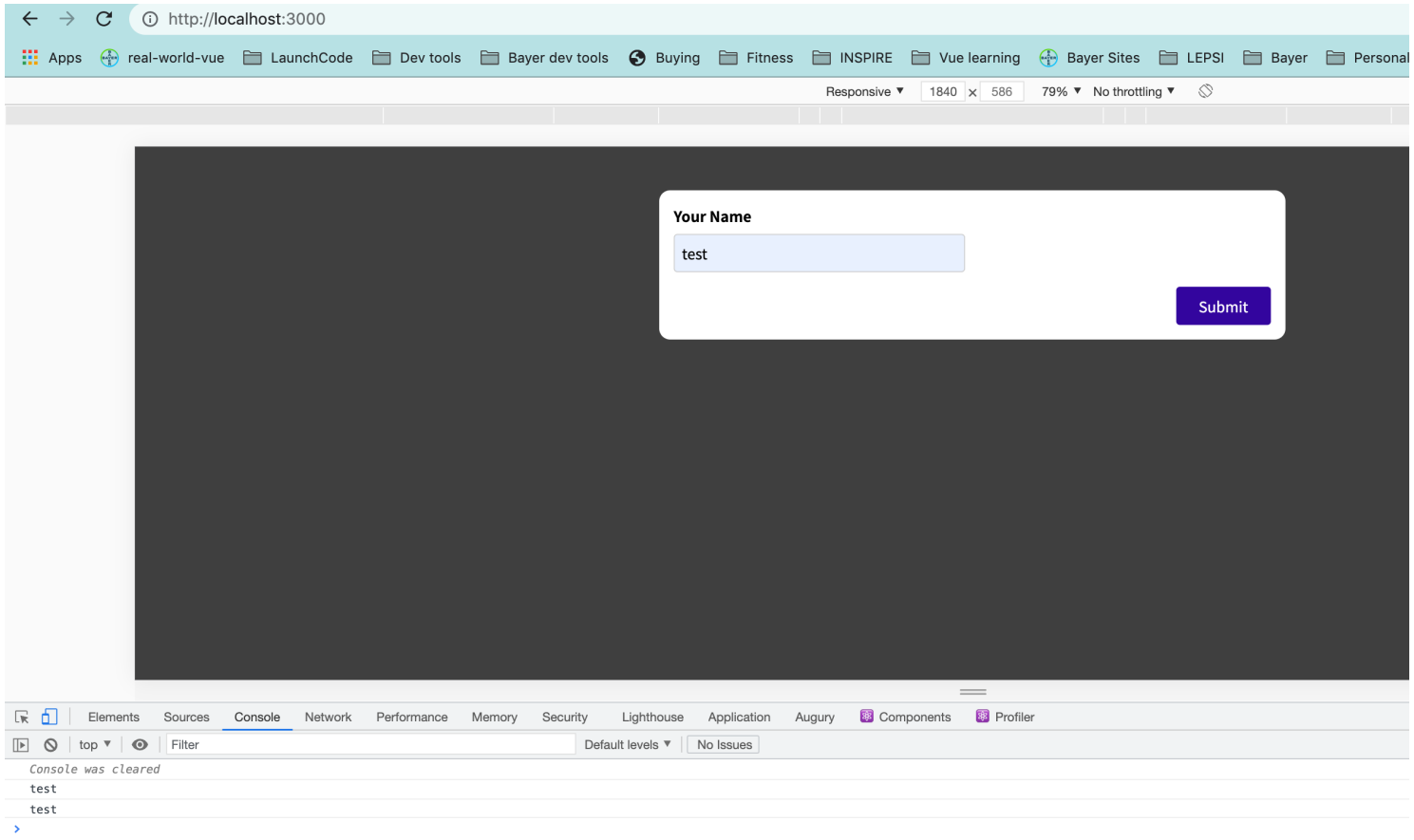
When To Validate?

When form is **submitted**	When a input is **losing focus**	On **every keystroke**
Allows the user to enter a valid value before warning him / her	Allows the user to enter a valid value before warning him / her	Warns user before he / she had a chance of entering valid values
Avoid unnecessary warnings but maybe present feedback ”too late”	Very useful for untouched forms	If applied only on invalid inputs, has the potential of providing more direct feedback

```jsx
import { useState } from 'react';
const SimpleInput = (props) => {
  const [enteredName, setEnteredName] = useState('');

  const nameInputChangeHandler = (event) => {
    setEnteredName(event.target.value);
  };

  const formSubmissionHandler = (event) => {
    event.preventDefault();
    console.log(enteredName);
  };
  return (
    <form onSubmit={formSubmissionHandler}>
      <div className="form-control">
        <label htmlFor="name">Your Name</label>
        <input type="text" id="name" onChange={nameInputChangeHandler} />
      </div>
      <div className="form-actions">
        <button>Submit</button>
      </div>
    </form>
  );
};

export default SimpleInput;
```

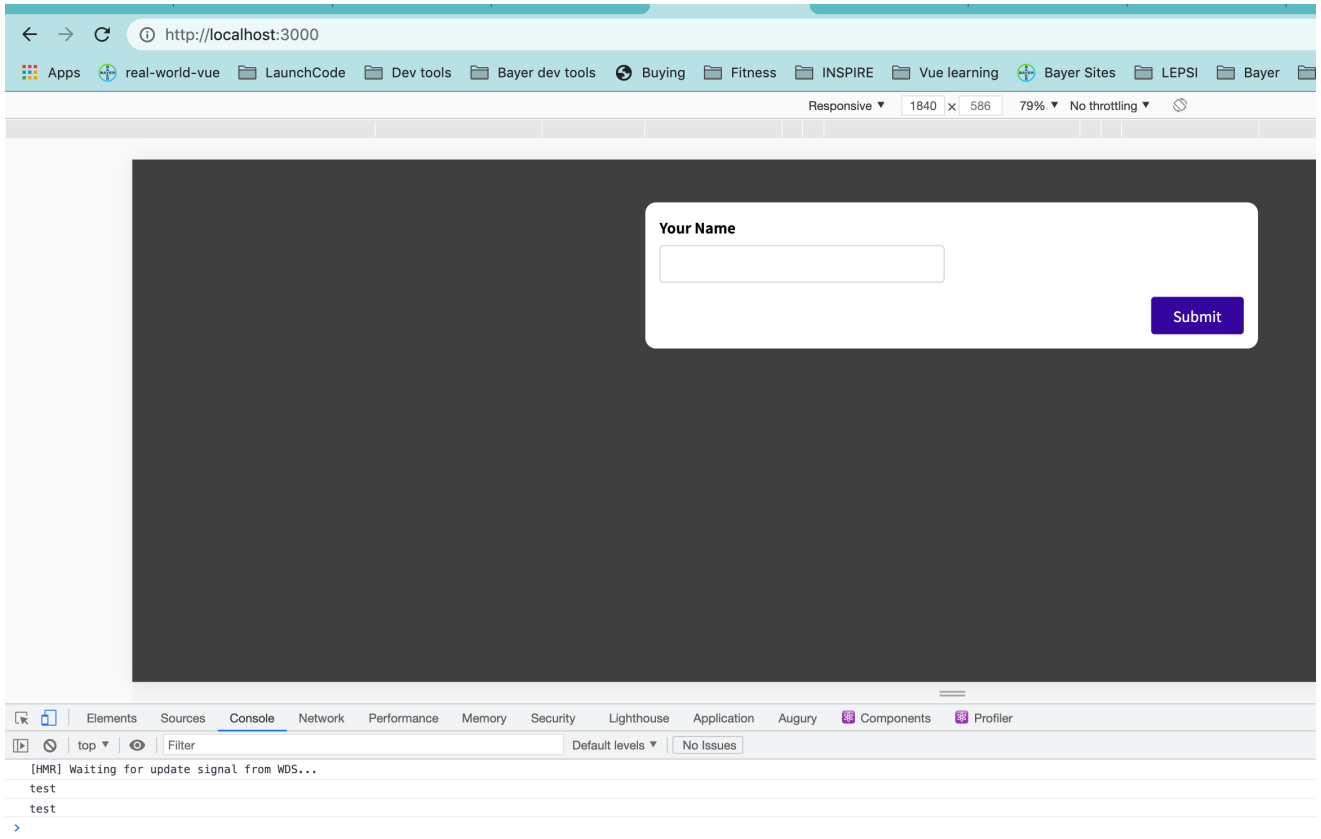


```jsx
import { useRef, useState } from 'react';
const SimpleInput = (props) => {
  const nameInputRef = useRef();
  const [enteredName, setEnteredName] = useState('');

  const nameInputChangeHandler = (event) => {
    setEnteredName(event.target.value);
  };

  const formSubmissionHandler = (event) => {
    event.preventDefault();
    console.log(enteredName);
    const enteredValue = nameInputRef.current.value;
    console.log(enteredValue);
  };
  return (
    <form onSubmit={formSubmissionHandler}>
      <div className="form-control">
        <label htmlFor="name">Your Name</label>
        <input
          ref={nameInputRef}
          type="text"
          id="name"
          onChange={nameInputChangeHandler}
        />
      </div>
      <div className="form-actions">
        <button>Submit</button>
      </div>
    </form>
  );
};

export default SimpleInput;
```



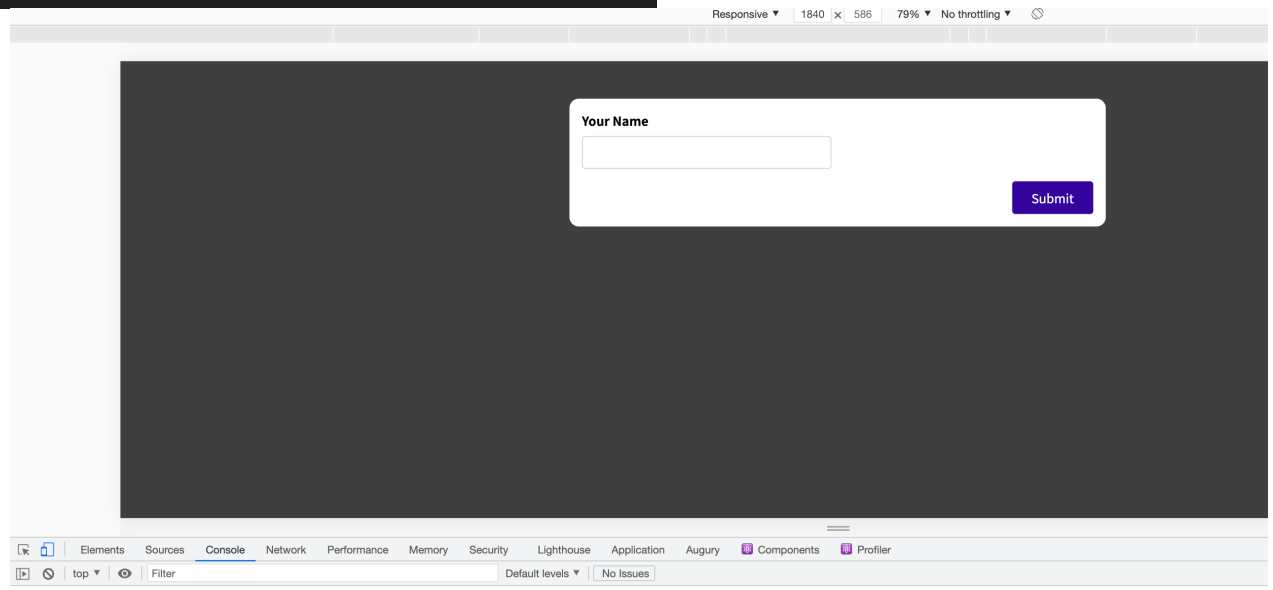
Ref is better if value only wanted once.
Value if needed with every stroke for validation on keystroke, then use the state way. Also, to reset value to empty string, so we can bind the entered value through the value prop.
DOM manipulation via ref isn't recommended.

```jsx
import { useRef, useState } from 'react';
const SimpleInput = (props) => {
  const nameInputRef = useRef();
  const [enteredName, setEnteredName] = useState('');

  const nameInputChangeHandler = (event) => {
    setEnteredName(event.target.value);
  };

  const formSubmissionHandler = (event) => {
    event.preventDefault();
    console.log(enteredName);
    const enteredValue = nameInputRef.current.value;
    console.log(enteredValue);
    setEnteredName('');
  };
  return (
    <form onSubmit={formSubmissionHandler}>
      <div className="form-control">
        <label htmlFor="name">Your Name</label>
        <input
          ref={nameInputRef}
          type="text"
          id="name"
          onChange={nameInputChangeHandler}
          value={enteredName}
        />
      </div>
      <div className="form-actions">
        <button>Submit</button>
      </div>
    </form>
  );
};

export default SimpleInput;
```



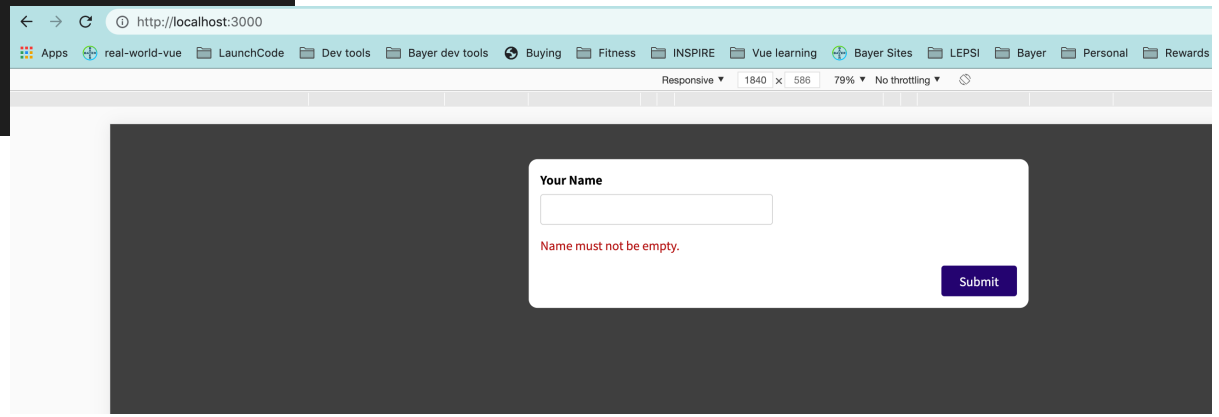
Adding Basic Validation

```
const formSubmissionHandler = (event) => {
  event.preventDefault();
  if (enteredName.trim() === '') {
    return;
  }
  console.log(enteredName);
  const enteredValue = nameInputRef.current.value;
  console.log(enteredValue);
  setEnteredName('');
};
```

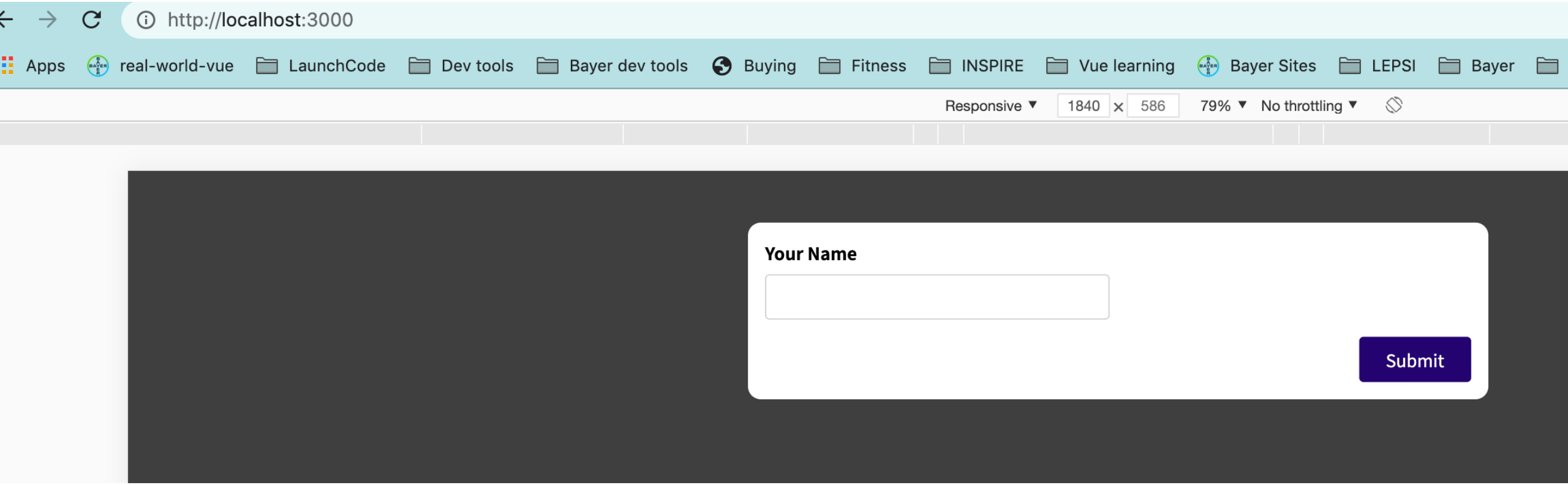


Provide user feedback.

```jsx
import { useRef, useState } from 'react';
const SimpleInput = (props) => {
  const nameInputRef = useRef();
  const [enteredName, setEnteredName] = useState('');
  const [enteredNameIsValid, setEnteredNameIsValid] = useState(false);

  const nameInputChangeHandler = (event) => {
    setEnteredName(event.target.value);
  };

  const formSubmissionHandler = (event) => {
    event.preventDefault();
    if (enteredName.trim() === '') {
      setEnteredNameIsValid(false);
      return;
    }
    setEnteredNameIsValid(true);
    console.log(enteredName);
    const enteredValue = nameInputRef.current.value;
    console.log(enteredValue);
    setEnteredName('');
  };
```

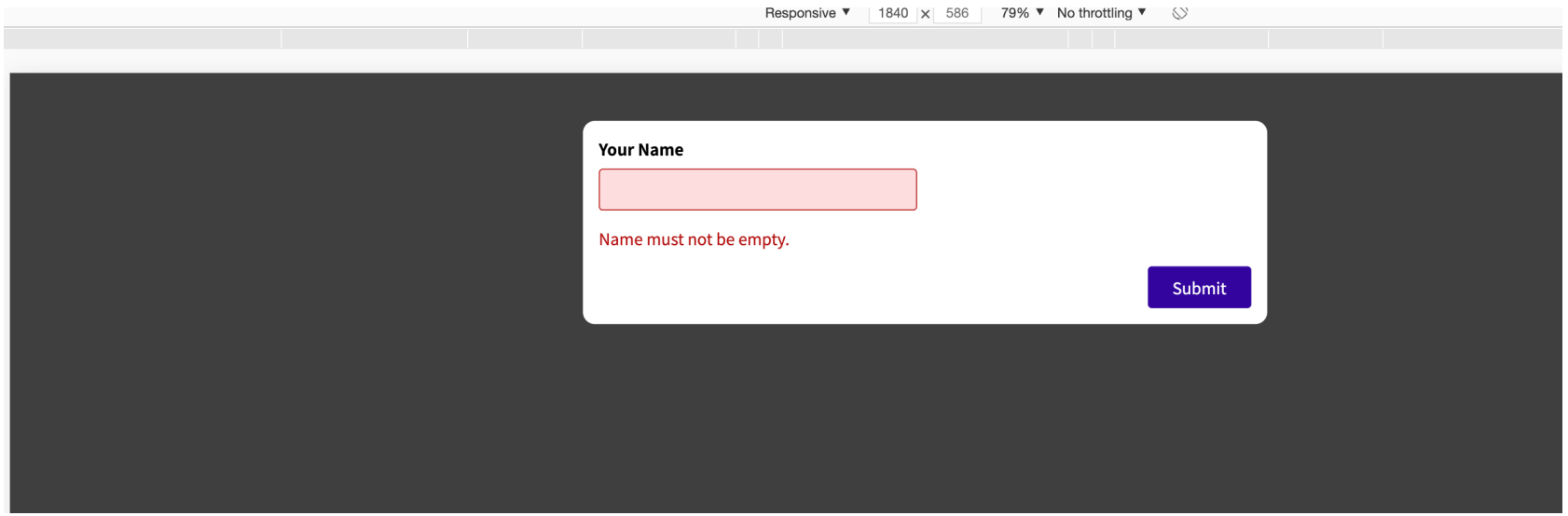


Change to set enteredNameIsValid to true initially



```jsx
const nameInputClasses = enteredNameIsValid
  ? 'form-control'
  : 'form-control invalid';
return (
  <form onSubmit={formSubmissionHandler}>
    <div className={nameInputClasses}>
      <label htmlFor="name">Your Name</label>
      <input
        ref={nameInputRef}
        type="text"
        id="name"
        onChange={nameInputChangeHandler}
        value={enteredName}
      />
      {!enteredNameIsValid && (
        <p className="error-text">Name must not be empty.</p>
      )}
    </div>
    <div className="form-actions">
      <button>Submit</button>
    </div>
  </form>
);
```

Adding invalid css



Downside to adding invalid css

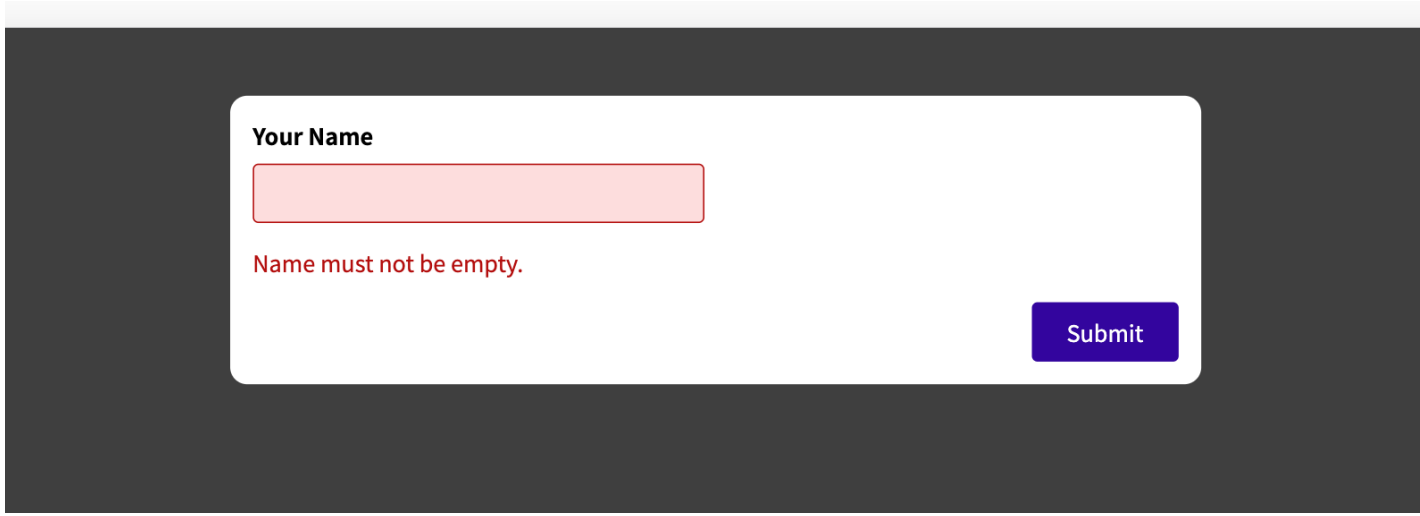
Setting enteredNameIsValid to true initially is cheating and untrue
Why could this be a problem?

If there was a useEffect when enteredNameIsValid is changed.
For example, logging the value to, and calling a service for example.
It can make a redundant call and cause problems.

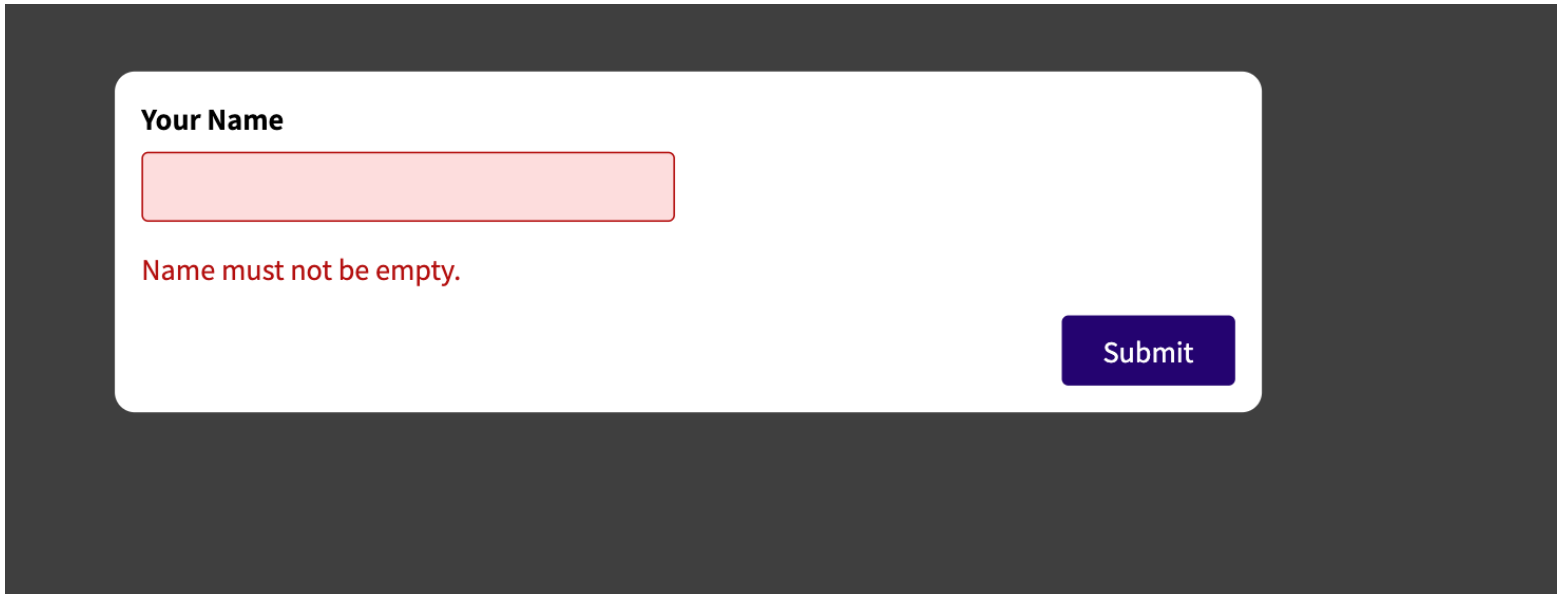
It makes more sense to set it to false initially.

We can make a third state called enteredNameTouched. Initialize to false.

```jsx
import { useRef, useState } from 'react';
const SimpleInput = (props) => {
  const nameInputRef = useRef();
  const [enteredName, setEnteredName] = useState('');
  const [enteredNameIsValid, setEnteredNameIsValid] = useState(true);
  const [enteredNameTouched, setEnteredNameTouched] = useState(false);

  const nameInputChangeHandler = (event) => {
    setEnteredName(event.target.value);
  };

  const formSubmissionHandler = (event) => {
    event.preventDefault();
    setEnteredNameTouched(true);
    if (enteredName.trim() === '') {
      setEnteredNameIsValid(false);
      return;
    }
    setEnteredNameIsValid(true);
    console.log(enteredName);
    const enteredValue = nameInputRef.current.value;
    console.log(enteredValue);
    setEnteredName('');
  };

  const nameInputIsInvalid = !enteredNameIsValid && enteredNameTouched;

  const nameInputClasses = nameInputIsInvalid
    ? 'form-control invalid'
    : 'form-control';
```



```
const nameInputBlurHandler = (event) => {
  setEnteredNameTouched(true);
  if (enteredName.trim() === '') {
    setEnteredNameIsValid(false);
    return;
  }
};

const formSubmissionHandler = (event) => {
  event.preventDefault();
  setEnteredNameTouched(true);
  if (enteredName.trim() === '') {
    setEnteredNameIsValid(false);
    return;
  }
  setEnteredNameIsValid(true);
  console.log(enteredName);
  const enteredValue = nameInputRef.current.value;
  console.log(enteredValue);
  setEnteredName('');
};

const nameInputIsInvalid = !enteredNameIsValid && enteredNameTouched;

const nameInputClasses = nameInputIsInvalid
  ? 'form-control invalid'
  : 'form-control';
return (
  <form onSubmit={formSubmissionHandler}>
    <div className={nameInputClasses}>
      <label htmlFor="name">Your Name</label>
      <input
        ref={nameInputRef}
        type="text"
        id="name"
        onChange={nameInputChangeHandler}
        onBlur={nameInputBlurHandler}
        value={enteredName}
      />
```



Refactoring and Deriving States

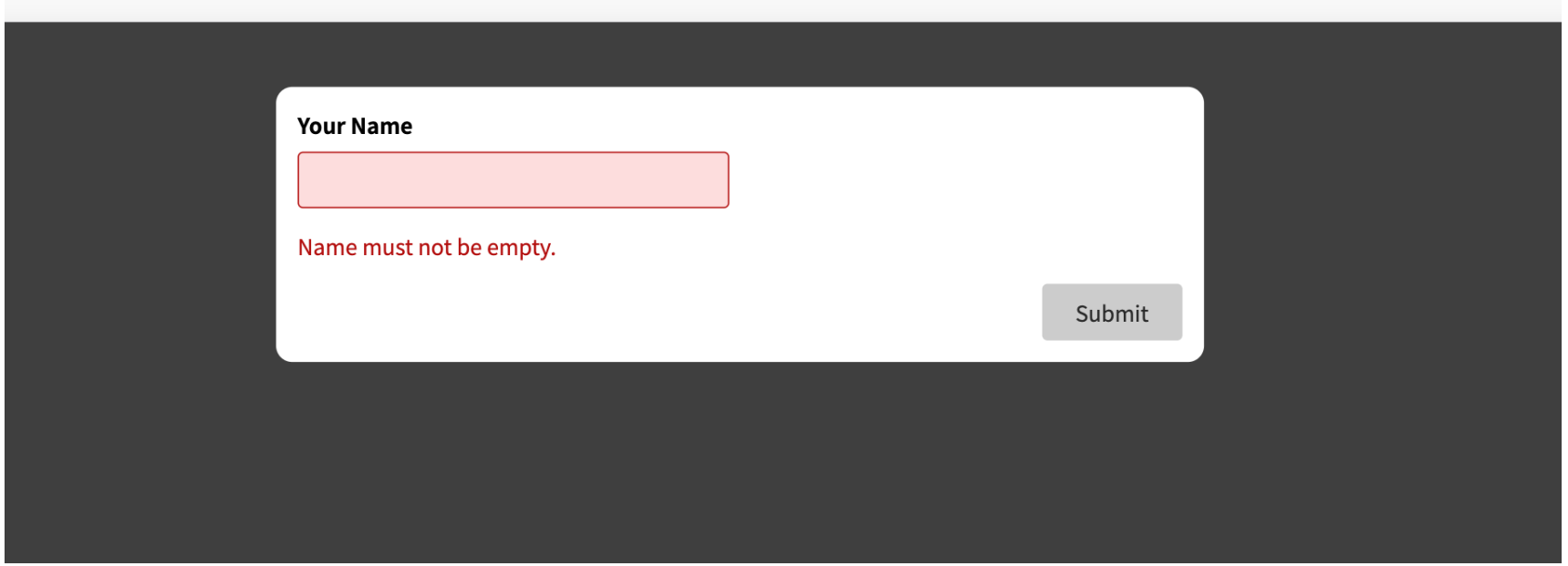
Remove useRef.
Clean up code.
Analyze behavior and refactor flow if needed.

Overall Form Validity

```
import { useEffect, useState } from 'react';
import '../index.css';
const SimpleInput = (props) => {
  const [enteredName, setEnteredName] = useState('');
  const [enteredNameIsValid, setEnteredNameIsValid] = useState(true);
  const [enteredNameTouched, setEnteredNameTouched] = useState(false);
  const [formIsValid, setFormIsValid] = useState(false);

  useEffect(() => {
    if (setEnteredNameIsValid === true) {
      setFormIsValid(true);
    } else {
      setFormIsValid(false);
    }
  }, [enteredNameIsValid]);

  const nameInputChangeHandler = (event) => {
    setEnteredName(event.target.value);
  };

  const nameInputBlurHandler = (event) => {
    setEnteredNameTouched(true);
    if (event.target.value.trim() === '') {
      setEnteredNameIsValid(false);
      return;
    }
  };

  const formSubmissionHandler = (event) => {
    event.preventDefault();
    setEnteredNameTouched(true);
    if (event.target.value.trim() === '') {
      setEnteredNameIsValid(false);
      return;
    }
    setEnteredNameIsValid(true);
    console.log(enteredName);
    setEnteredName('');
  };

  const nameInputIsInvalid = !enteredNameIsValid && enteredNameTouched;

  const nameInputClasses = nameInputIsInvalid
    ? 'form-control invalid'
    : 'form-control';
```

Overall Form Validity

Inputs all need to valid for Form to be valid.
Update the formIsValid to false when any of the inputs is invalid.



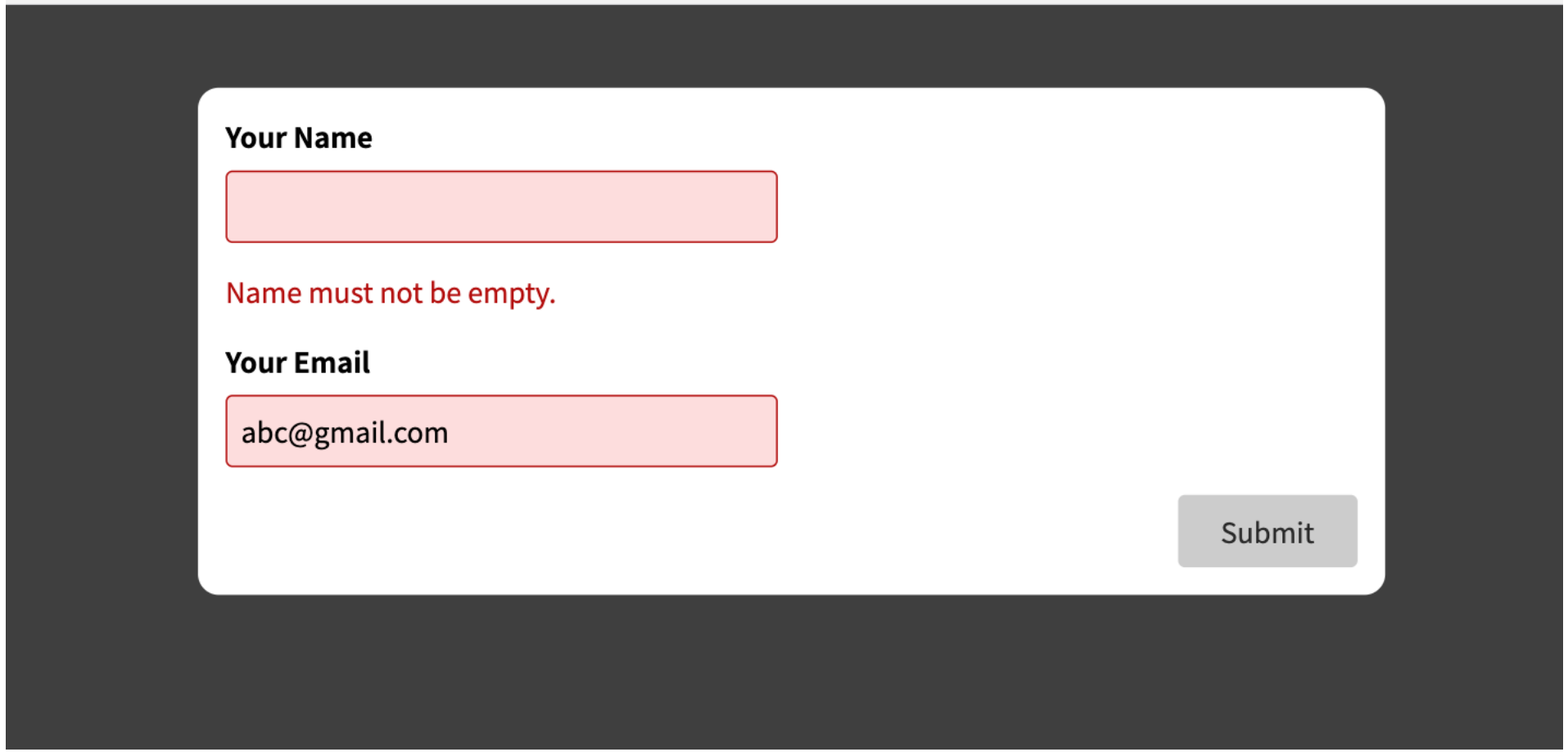
Can choose to enable or disable button when input is invalid.

Assignment: Practice Forms

Add a second input as email address.
Validate the email address – JS email validation. Check for @ symbol.
Managing the state for email and its validity
Ultimately form can be submitted only both inputs are valid.

```jsx
import { useEffect, useState } from 'react';
import '../index.css';
const SimpleInput = (props) => {
  const [enteredName, setEnteredName] = useState('');
  const [enteredNameIsValid, setEnteredNameIsValid] = useState(true);
  const [enteredNameTouched, setEnteredNameTouched] = useState(false);
  const [enteredEmail, setEnteredEmail] = useState('');
  const [enteredEmailIsValid, setEnteredEmailIsValid] = useState(true);
  const [enteredEmailTouched, setEnteredEmailTouched] = useState(false);
  const [formIsValid, setFormIsValid] = useState(false);

  useEffect(() => {
    if (enteredNameIsValid === true && enteredEmailIsValid) {
      setFormIsValid(true);
    } else {
      setFormIsValid(false);
    }
  }, [enteredNameIsValid, enteredEmailIsValid]);

  const nameInputChangeHandler = (event) => {
    setEnteredName(event.target.value);
  };

  const nameInputBlurHandler = (event) => {
    setEnteredNameTouched(true);
    if (event.target.value.trim() === '') {
      setEnteredNameIsValid(false);
      return;
    }
  };

  const emailInputChangeHandler = (event) => {
    setEnteredEmail(event.target.value);
  };

  const emailInputBlurHandler = (event) => {
    setEnteredEmailTouched(true);
    if (
      event.target.value.trim() === '' ||
      event.target.value.indexOf('@') === -1
    ) {
      setEnteredEmailIsValid(false);
      return;
    }
  };
```

```jsx
  const nameInputIsInvalid = !enteredNameIsValid && enteredNameTouched;
  const emailInputIsInvalid = !enteredEmailIsValid && enteredEmailTouched;
  const nameInputClasses = nameInputIsInvalid
    ? 'form-control invalid'
    : 'form-control';
  return (
    <form onSubmit={formSubmissionHandler}>
      <div className={nameInputClasses}>
        <label htmlFor="name">Your Name</label>
        <input
          type="text"
          id="name"
          onChange={nameInputChangeHandler}
          onBlur={nameInputBlurHandler}
          value={enteredName}
        />
        {nameInputIsInvalid && (
          <p className="error-text">Name must not be empty.</p>
        )}
        <label htmlFor="email">Your Email</label>
        <input
          type="email"
          id="email"
          value={enteredEmail}
          onChange={emailInputChangeHandler}
          onBlur={emailInputBlurHandler}
        />
        {emailInputIsInvalid && (
          <p className="error-text">Not a valid email.</p>
        )}
      </div>
      <div className="form-actions">
        <button disabled={!formIsValid}>Submit</button>
      </div>
    </form>
  );
};

export default SimpleInput;
```


Your Name
Name must not be empty.
Your Email
abc@gmail.com
Submit

Your Name

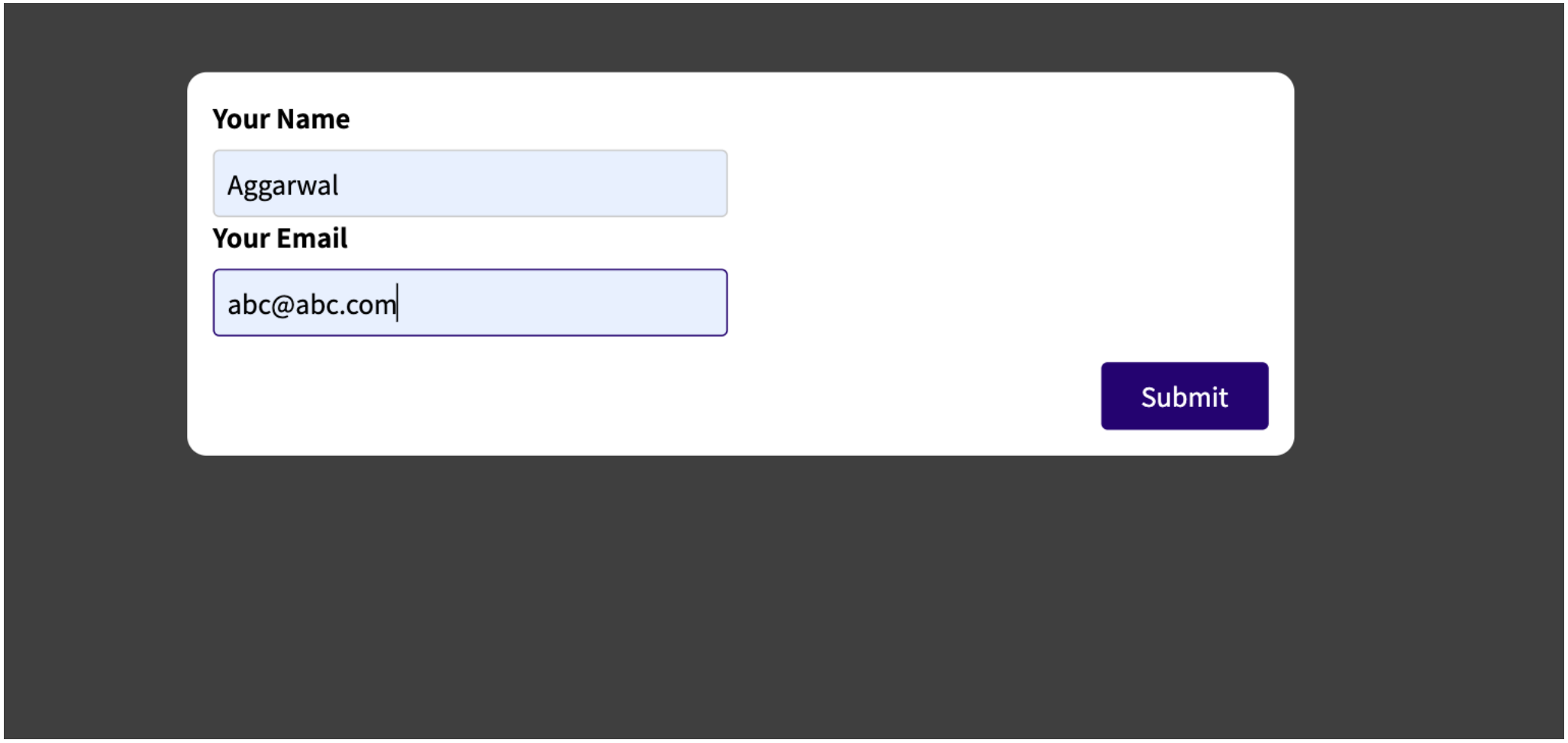
Chetna

Your Email

abc

Not a valid email.

Submit


Your Name
Aggarwal
Your Email
abc@abc.com
Submit

Since adding code for each input field makes the js very repetitive and crowded, we can outsource logic somewhere

Input Component with logic to validate inputs etc

Managing form validity might be tricky in that case

It can be done via props.

There is another approach called custom hook.

Create hooks folder and create use-input.js file as custom hook.

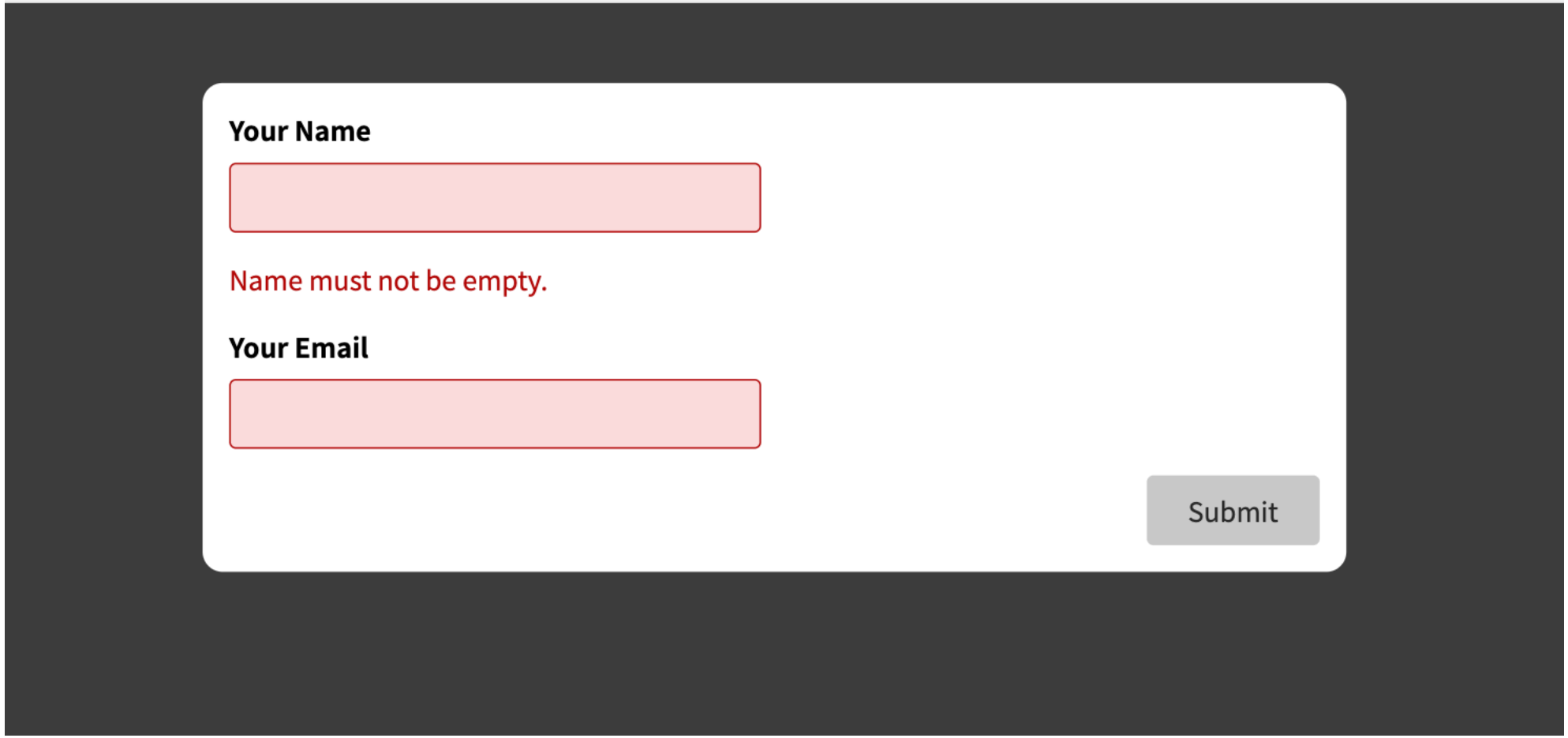
Hook receives a validate function as a param.

Hook can return an object

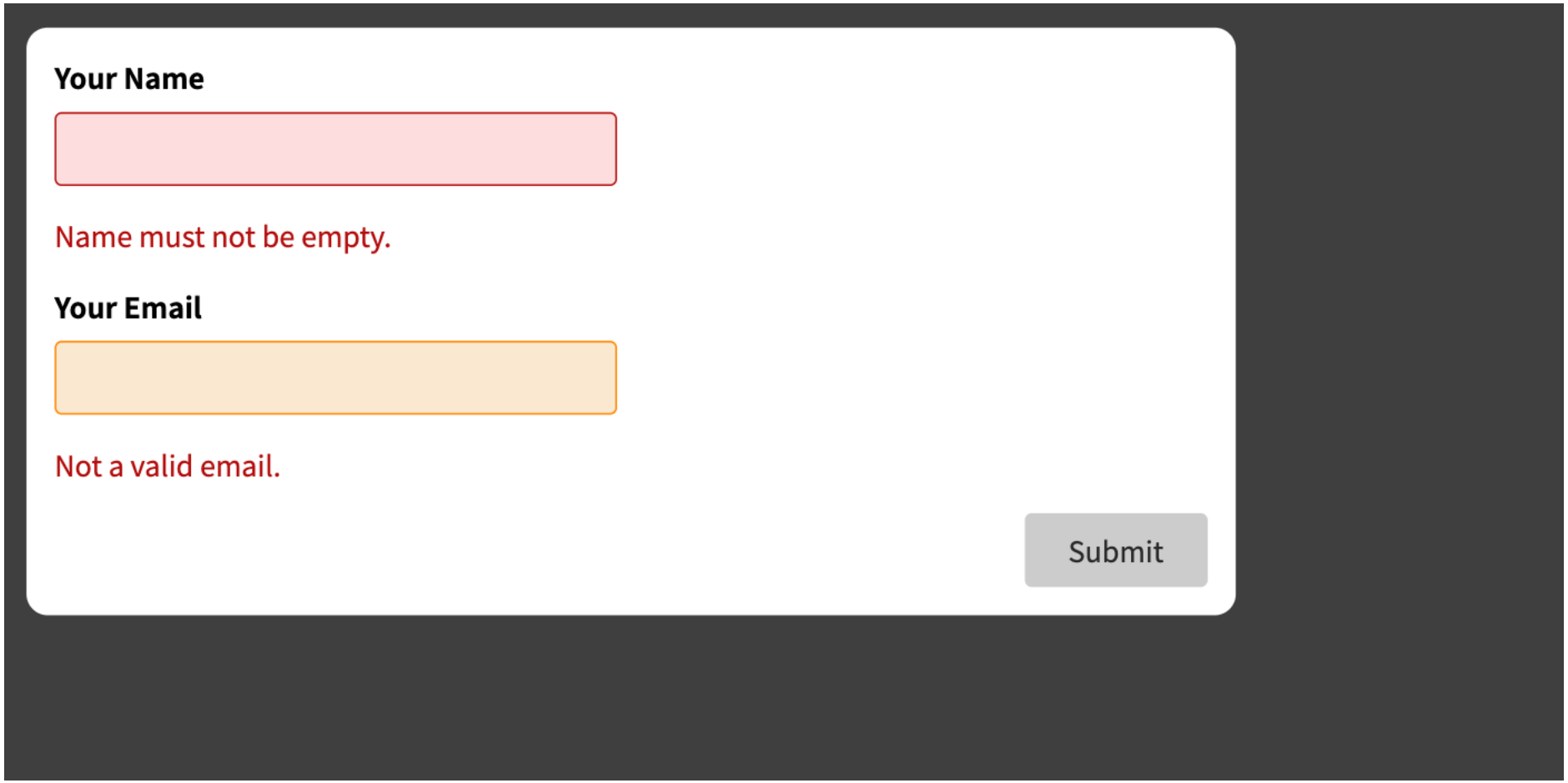
```jsx
import { useState } from 'react';
const useInput = (validateValue) => {
  const [enteredValue, setEnteredValue] = useState('');
  const [isTouched, setIsTouched] = useState(false);

  const valueIsValid = validateValue(enteredValue);
  const hasError = !valueIsValid && isTouched;

  const valueChangeHandler = (event) => {
    setEnteredValue(event.target.value);
  };

  const inputBlurHandler = (event) => {
    setIsTouched(true);
  };

  const reset = () => {
    setEnteredValue('');
    setIsTouched(false);
  };
  return {
    value: enteredValue,
    isValid: valueIsValid,
    hasError,
    valueChangeHandler,
    inputBlurHandler,
    reset,
  };
};
export default useInput;
```

```jsx
import { useEffect, useState } from 'react';
import '../index.css';
import useInput from '../hooks/use-input';
const SimpleInput = (props) => {
  const {
    value: enteredName,
    isValid: enteredNameIsValid,
    hasError: nameInputHasError,
    valueChangeHandler: nameChangeHandler,
    inputBlurHandler: nameBlurHandler,
    reset: resetNameInput,
  } = useInput((value) => value.trim() !== '');

  const [enteredEmail, setEnteredEmail] = useState('');
  const [enteredEmailIsValid, setEnteredEmailIsValid] = useState(true);
  const [enteredEmailTouched, setEnteredEmailTouched] = useState(false);
  const [formIsValid, setFormIsValid] = useState(false);

  useEffect(() => {
    if (enteredNameIsValid === true && enteredEmailIsValid) {
      setFormIsValid(true);
    } else {
      setFormIsValid(false);
    }
  }, [enteredNameIsValid, enteredEmailIsValid]);

  const emailInputChangeHandler = (event) => {
    setEnteredEmail(event.target.value);
  };

  const emailInputBlurHandler = (event) => {
    setEnteredEmailTouched(true);
    if (
      event.target.value.trim() === '' ||
      event.target.value.indexOf('@') === -1
    ) {
      setEnteredEmailIsValid(false);
      return;
    }
  };
```

```jsx
const formSubmissionHandler = (event) => {
  event.preventDefault();
  resetNameInput();
};

const emailInputIsInvalid = !enteredEmailIsValid && enteredEmailTouched;
const nameInputClasses = nameInputHasError
  ? 'form-control invalid'
  : 'form-control';
return (
  <form onSubmit={formSubmissionHandler}>
    <div className={nameInputClasses}>
      <label htmlFor="name">Your Name</label>
      <input
        type="text"
        id="name"
        onChange={nameChangeHandler}
        onBlur={nameBlurHandler}
        value={enteredName}
      />
      {nameInputHasError && (
        <p className="error-text">Name must not be empty.</p>
      )}
      <label htmlFor="email">Your Email</label>
      <input
        type="email"
        id="email"
        value={enteredEmail}
        onChange={emailInputChangeHandler}
        onBlur={emailInputBlurHandler}
      />
      {emailInputIsInvalid && (
        <p className="error-text">Not a valid email.</p>
      )}
    </div>
    <div className="form-actions">
      <button disabled={!formIsValid}>Submit</button>
    </div>
  </form>
);
};

export default SimpleInput;
```


Your Name
Name must not be empty.
Your Email
Submit

```
const {
  value: enteredEmail,
  isValid: enteredEmailIsValid,
  hasError: emailInputHasError,
  valueChangeHandler: emailChangeHandler,
  inputBlurHandler: emailBlurHandler,
  reset: resetEmailInput,
} = useInput((value) => value.trim() !== '' && value.indexOf('@') !== -1);
```


Your Name
Name must not be empty.
Your Email
Not a valid email.
Submit

```jsx
import { useEffect, useState } from 'react';
import '../index.css';
import useInput from '../hooks/use-input';
const BasicForm = (props) => {
  const {
    value: enteredName,
    isValid: enteredNameIsValid,
    hasError: nameInputHasError,
    valueChangeHandler: nameChangeHandler,
    inputBlurHandler: nameBlurHandler,
    reset: resetNameInput,
  } = useInput((value) => value.trim() !== '');

  const [formIsValid, setFormIsValid] = useState(false);

  useEffect(() => {
    if (enteredNameIsValid === true) {
      setFormIsValid(true);
    } else {
      setFormIsValid(false);
    }
  }, [enteredNameIsValid]);

  const nameInputClasses = nameInputHasError
    ? 'form-control invalid'
    : 'form-control';

  const formSubmissionHandler = (event) => {
    event.preventDefault();
    resetNameInput();
  };
  return (
    <form onSubmit={formSubmissionHandler}>
      <div className={nameInputClasses}>
        <div className="form-control">
          <label htmlFor="name">First Name</label>
          <input
            type="text"
            id="name"
            onChange={nameChangeHandler}
            onBlur={nameBlurHandler}
            value={enteredName}
          />
          {nameInputHasError && (
            <p className="error-text">Name must not be empty.</p>
          )}
        </div>
        <div className="form-control">
          <label htmlFor="name">Last Name</label>
          <input type="text" id="name" />
        </div>
      </div>
      <div className="form-control">
        <label htmlFor="name">E-Mail Address</label>
        <input type="text" id="name" />
      </div>
      <div className="form-actions">
        <button disabled={!formIsValid}>Submit</button>
      </div>
    </form>
  );
};
```

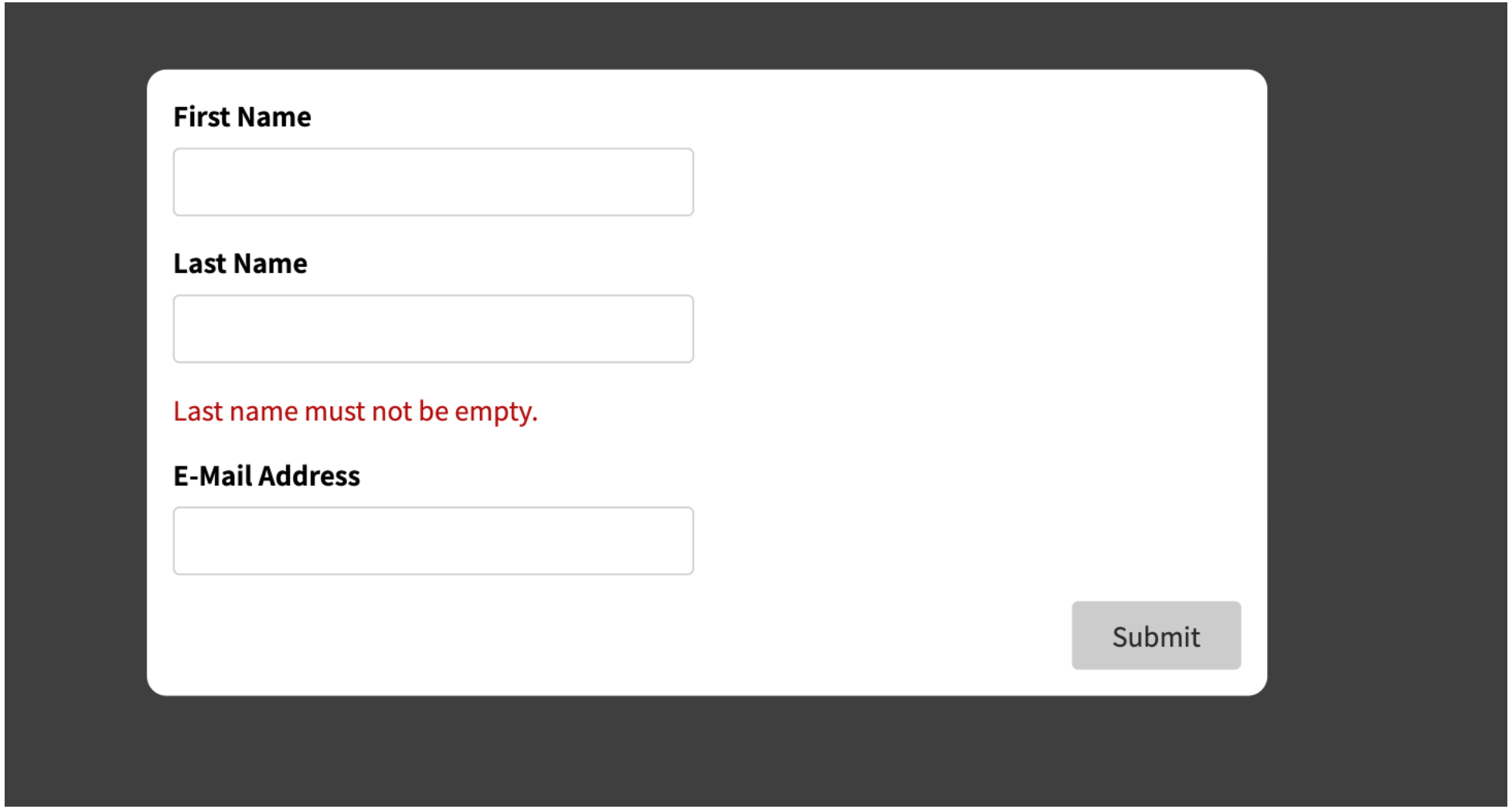
First Name

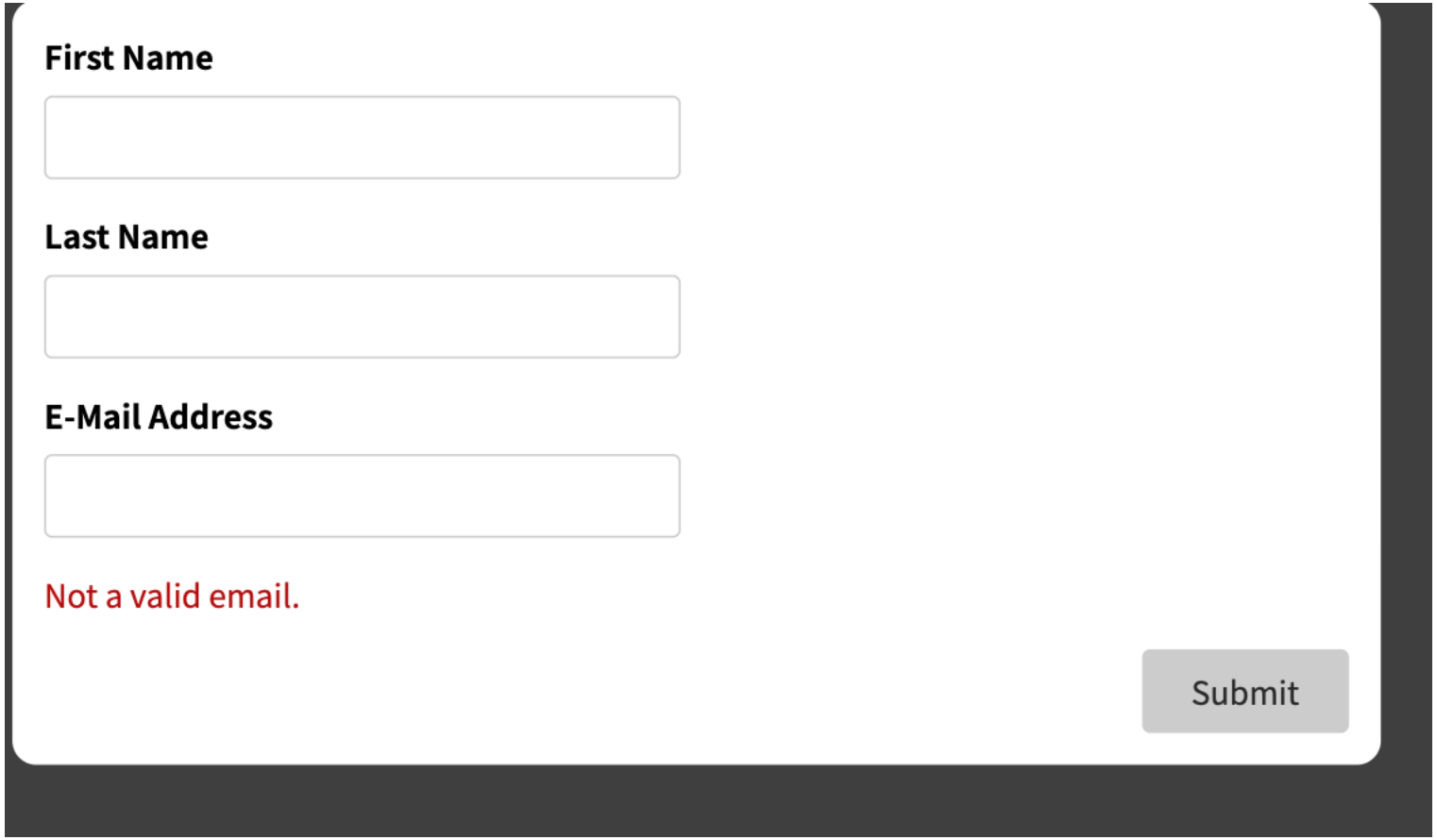
Name must not be empty.

Last Name

E-Mail Address

Submit


First Name
Last Name
Last name must not be empty.
E-Mail Address
Submit


First Name
Last Name
E-Mail Address
Not a valid email.
Submit

Refactoring

Form components
Refactor hook for form that can send readily available input fields.
Code can be less verbose and reduce code
duplication with the custom hooks.
Can develop more elaborate custom hooks.
https://academind.com/tutorials/reactjs-a-custom-useform-hook
Formik is a third party library to build forms.

ACADEMIND

useState() vs useReducer()

Generally, you'll know when you need useReducer() (➔ when using useState() becomes cumbersome or you're getting a lot of bugs/ unintended behaviors)

useState()	useReducer()
The main state management "tool"	Great if you need "more power"
Great for independent pieces of state/ data	Should be considered if you have related pieces of state/ data
Great if state updates are easy and limited to a few kinds of updates	Can be helpful if you have more complex state updates