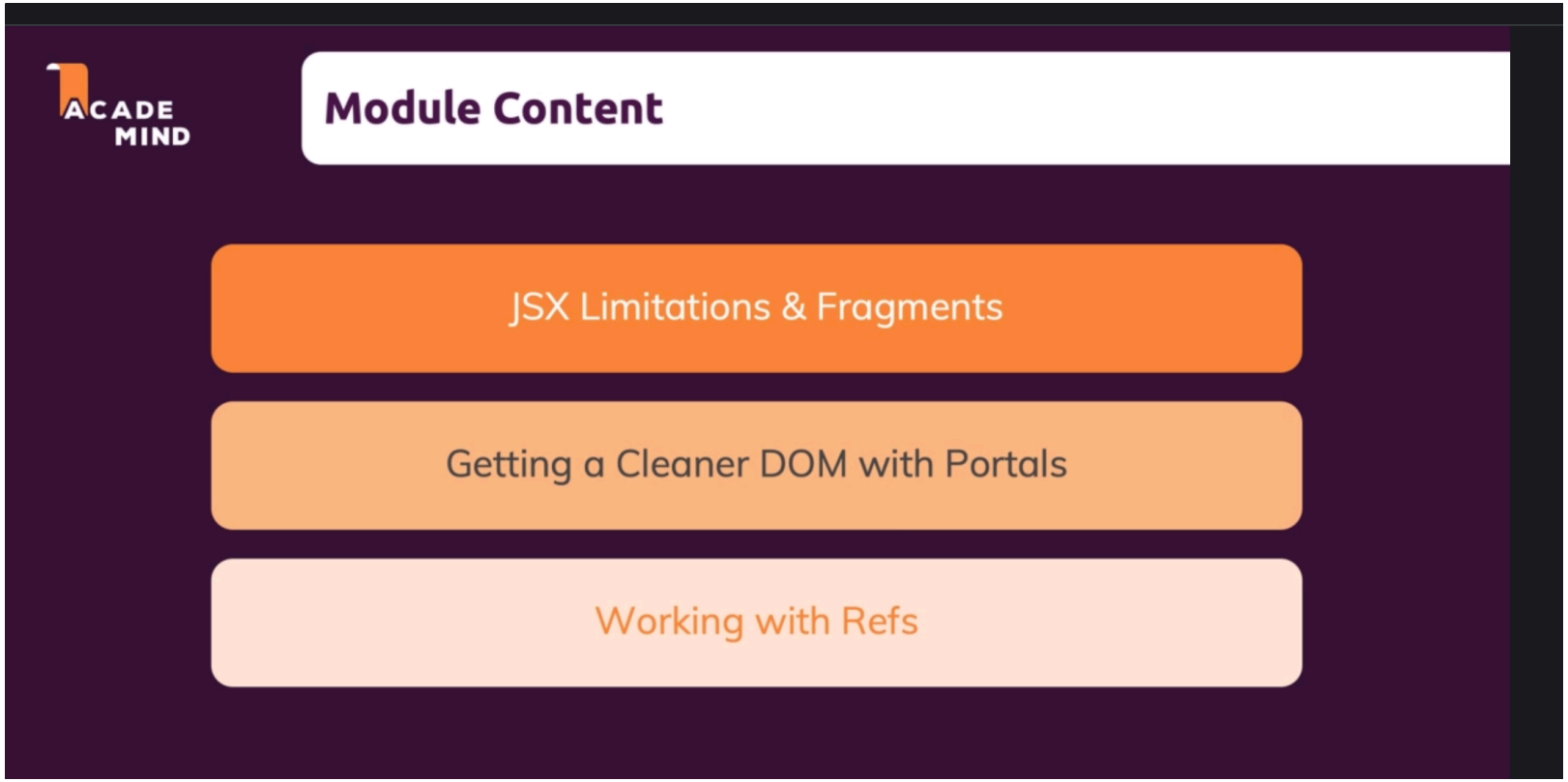
Front End

Class 10

September 8, 2021

Agenda

- Kahoot
- Code walkthrough with slides
- Studio


ACADEMIND
Module Content
JSX Limitations & Fragments
Getting a Cleaner DOM with Portals
Working with Refs

ACADEMIND

JSX Limitations

```
return (
  <h2>Hi there!</h2>
  <p>This does not work :-(</p>
);
```

You **can't return more than one "root" JSX element** (you also can't store more than one "root" JSX element in a variable).

Because this also isn't valid JavaScript

```
return (
  React.createElement('h2', {}, 'Hi there!')
  React.createElement('p', {}, 'This does not work :-(')
);
```

The Solution: Always Wrap Adjacent Elements

```
return (
  <div>
    <h2>Hi there!</h2>
    <p>This does not work :-(</p>
  </div>
);
```

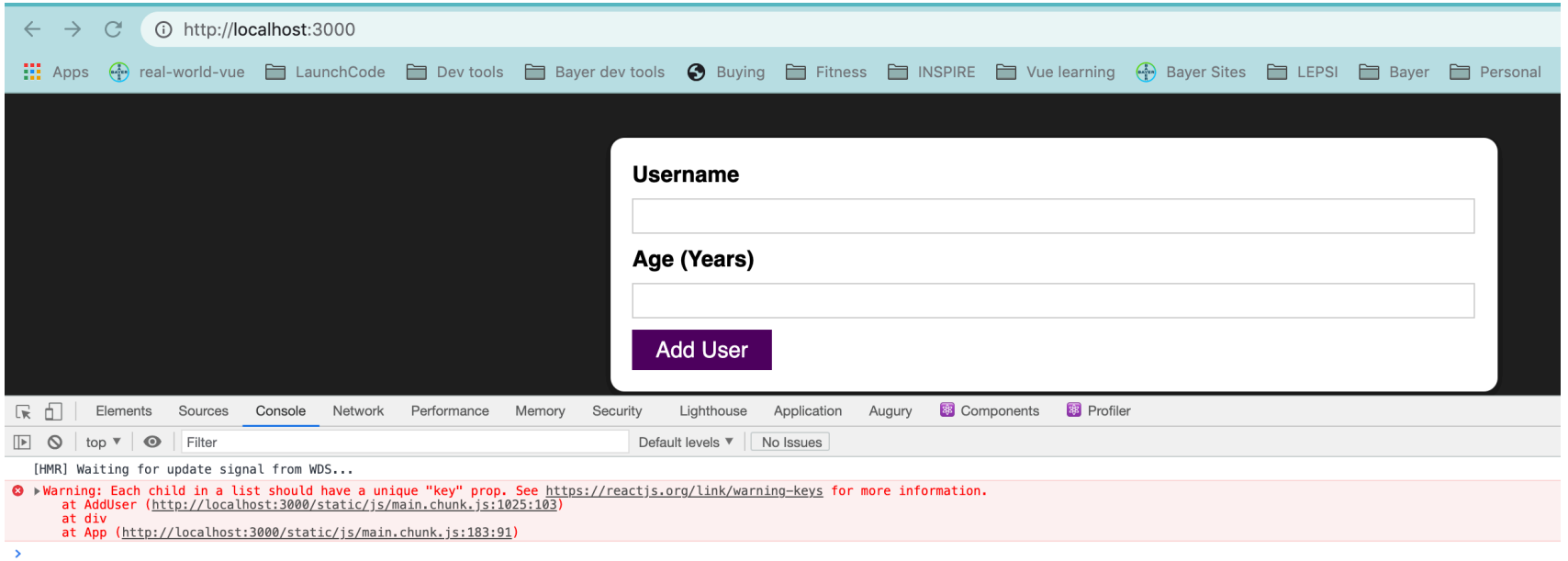
Array of JSX elements.

- We could return instead an Array or JSX elements, but we get a warning to supply keys for the elements.

```
return [
  error && (
    <ErrorModal
      title={error.title}
      message={error.message}
      onConfirm={errorHandler}
    />
  ),
  <Card className={classes.input}>
    <form onSubmit={addUserHandler}>
      <label htmlFor="username">Username</label>
      <input
        id="username"
        type="text"
        value={enteredUsername}
        onChange={usernameChangeHandler}
      />
      <label htmlFor="age">Age (Years)</label>
      <input
        id="age"
        type="number"
        value={enteredAge}
        onChange={ageChangeHandler}
      />
      <Button type="submit">Add User</Button>
    </form>
  </Card>,
];
```

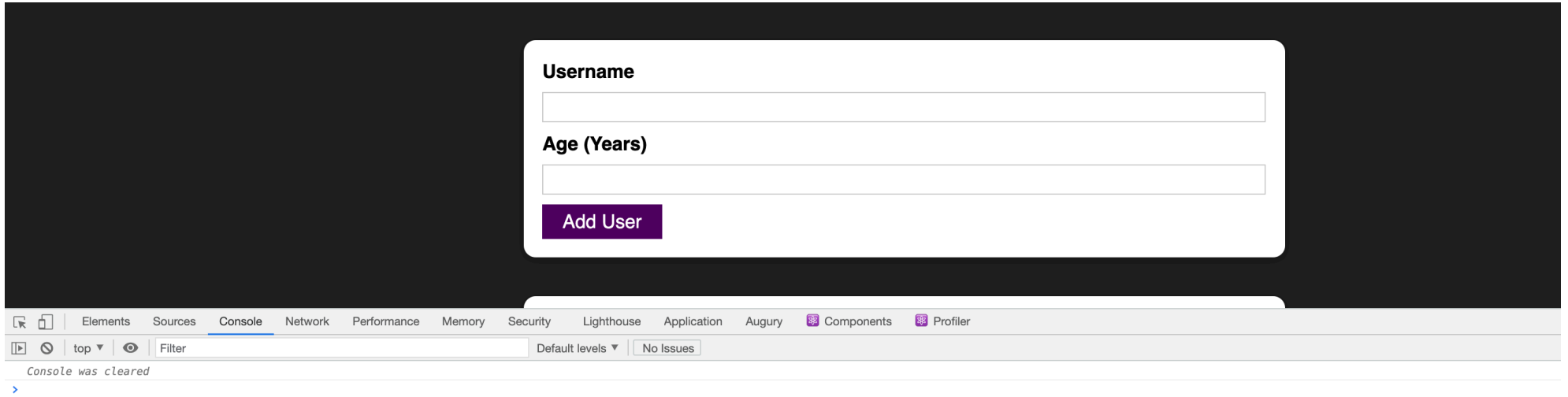
Array of JSX elements.

- We could return instead an Array or JSX elements, but we get a warning to supply keys for the elements.



Can add keys.

```
return [
  error && (
    <ErrorModal
      key="error-modal"
      title={error.title}
      message={error.message}
      onConfirm={errorHandler}
    />
  ),
  <Card key="add-user-card" className={classes.input}>
    <form onSubmit={addUserHandler}>
      <label htmlFor="username">Username</label>
      <input
        id="username"
        type="text"
        value={enteredUsername}
        onChange={usernameChangeHandler}
      />
      <label htmlFor="age">Age (Years)</label>
      <input
        id="age"
        type="number"
        value={enteredAge}
        onChange={ageChangeHandler}
      />
      <Button type="submit">Add User</Button>
    </form>
  </Card>,
];
```



This seems like additional work. Adding a wrapper element works.

ACADEMIND

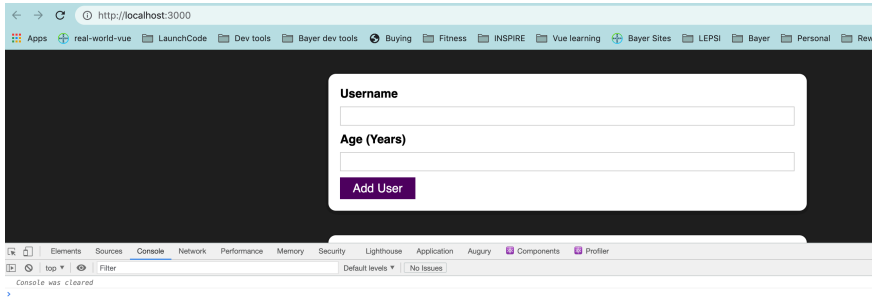
A New Problem: " Soup"

```
<div>
  <div>
    <div>
      <div>
        <h2>Some content - yeah, this can really happen.</h2>
      </div>
    </div>
  </div>
</div>
```

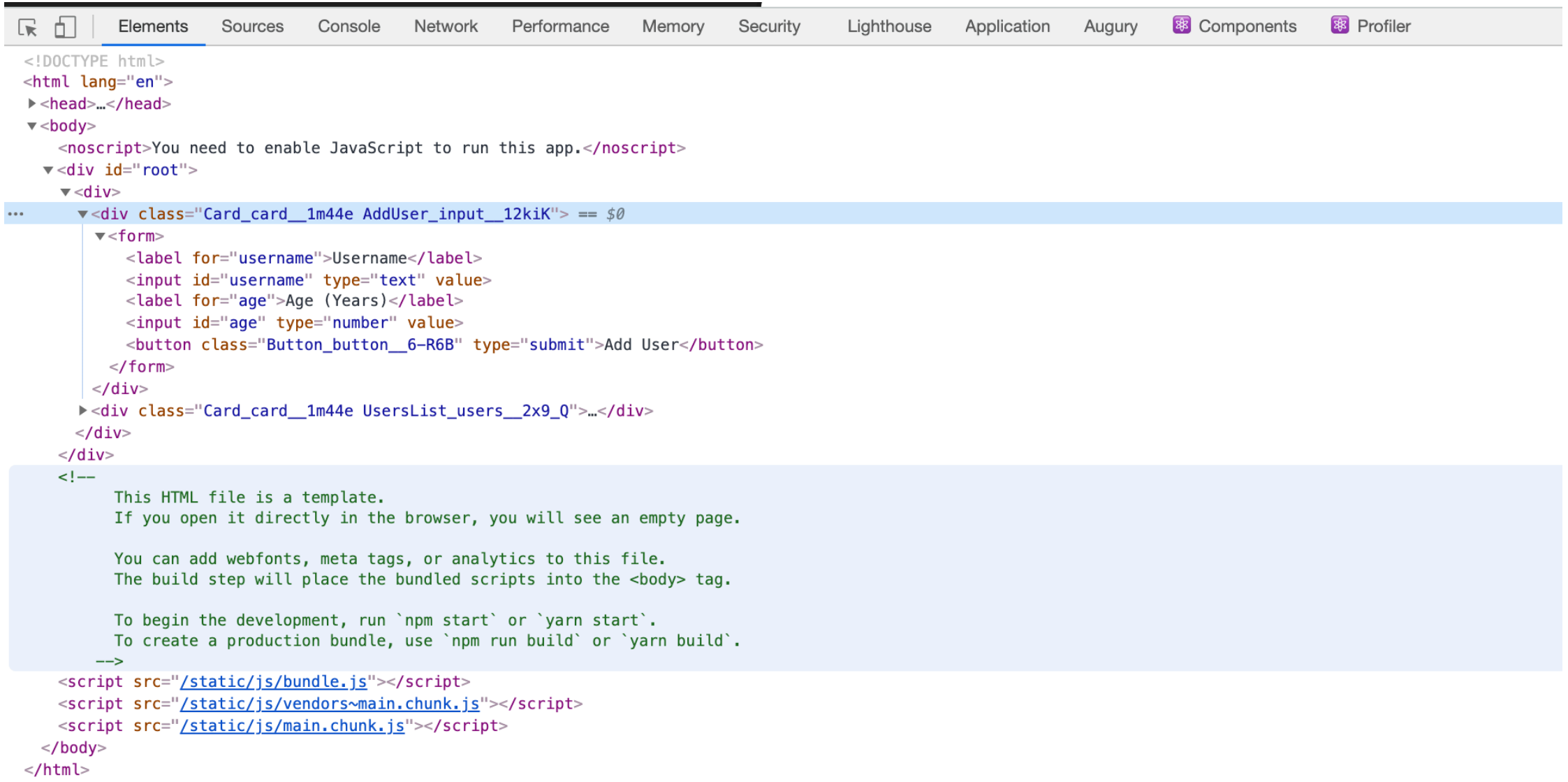
In bigger apps, you can easily end up with **tons of unnecessary **s (or other elements) which add **no semantic meaning or structure** to the page but **are only there because of React's/ JSX' requirement**.

Wrapper Component.

```jsx
const Wrapper = (props) => {
  return props.children;
};
export default Wrapper;
```




```jsx
return (
  <Wrapper>
    {error && (
      <ErrorModal
        title={error.title}
        message={error.message}
        onConfirm={errorHandler}
      />
    )}
    <Card className={classes.input}>
      <form onSubmit={addUserHandler}>
        <label htmlFor="username">Username</label>
        <input
          id="username"
          type="text"
          value={enteredUsername}
          onChange={usernameChangeHandler}
        />
        <label htmlFor="age">Age (Years)</label>
        <input
          id="age"
          type="number"
          value={enteredAge}
          onChange={ageChangeHandler}
        />
        <Button type="submit">Add User</Button>
      </form>
    </Card>
  </Wrapper>
);
```

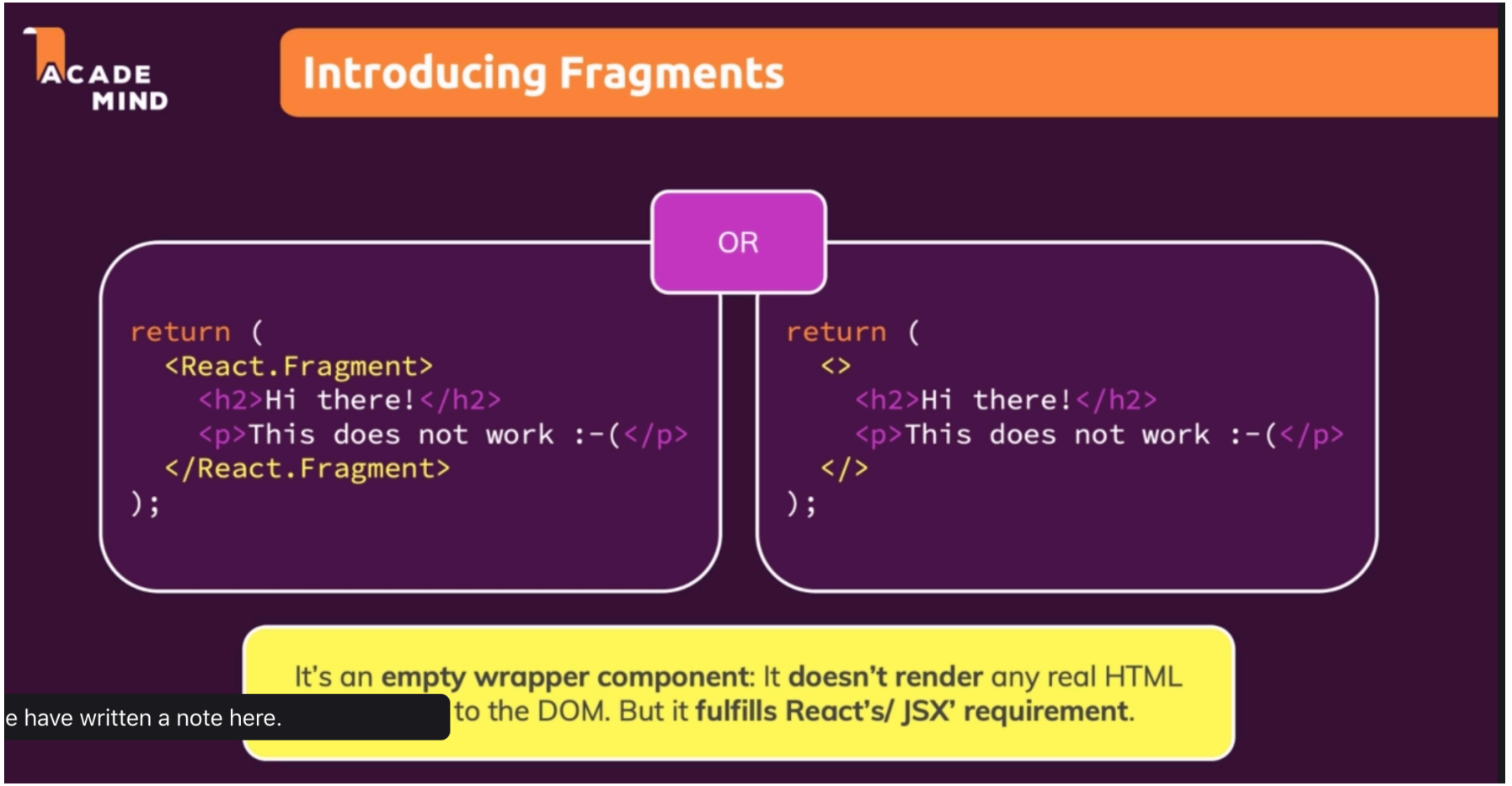

Elements
Sources
Console
Network
Performance
Memory
Security
Lighthouse
Application
Augury
Components
Profiler
<!DOCTYPE html>
<html lang="en">
<head>…</head>
<body>
<noscript>You need to enable JavaScript to run this app.</noscript>

 == $0
<form>
<label for="username">Username</label>
<input id="username" type="text" value>
<label for="age">Age (Years)</label>
<input id="age" type="number" value>
<button class="Button_button__6-R6B" type="submit">Add User</button>
</form>

…

<!--
This HTML file is a template.
If you open it directly in the browser, you will see an empty page.
You can add webfonts, meta tags, or analytics to this file.
The build step will place the bundled scripts into the <body> tag.
To begin the development, run `npm start` or `yarn start`.
To create a production bundle, use `npm run build` or `yarn build`.
-->
<script src="/static/js/bundle.js"></script>
<script src="/static/js/vendors~main.chunk.js"></script>
<script src="/static/js/main.chunk.js"></script>
</body>
</html>

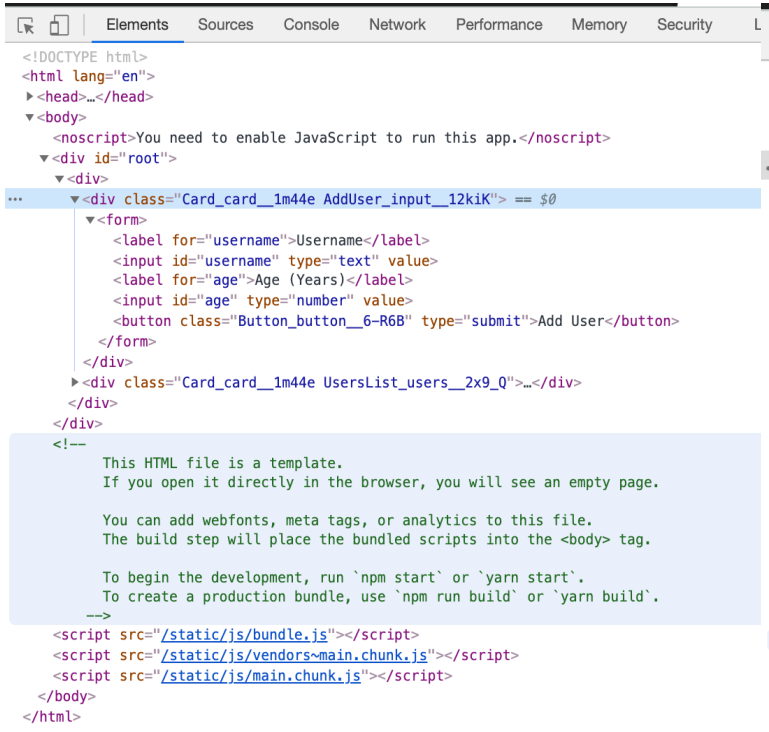
React Fragments – Built in wrapper.

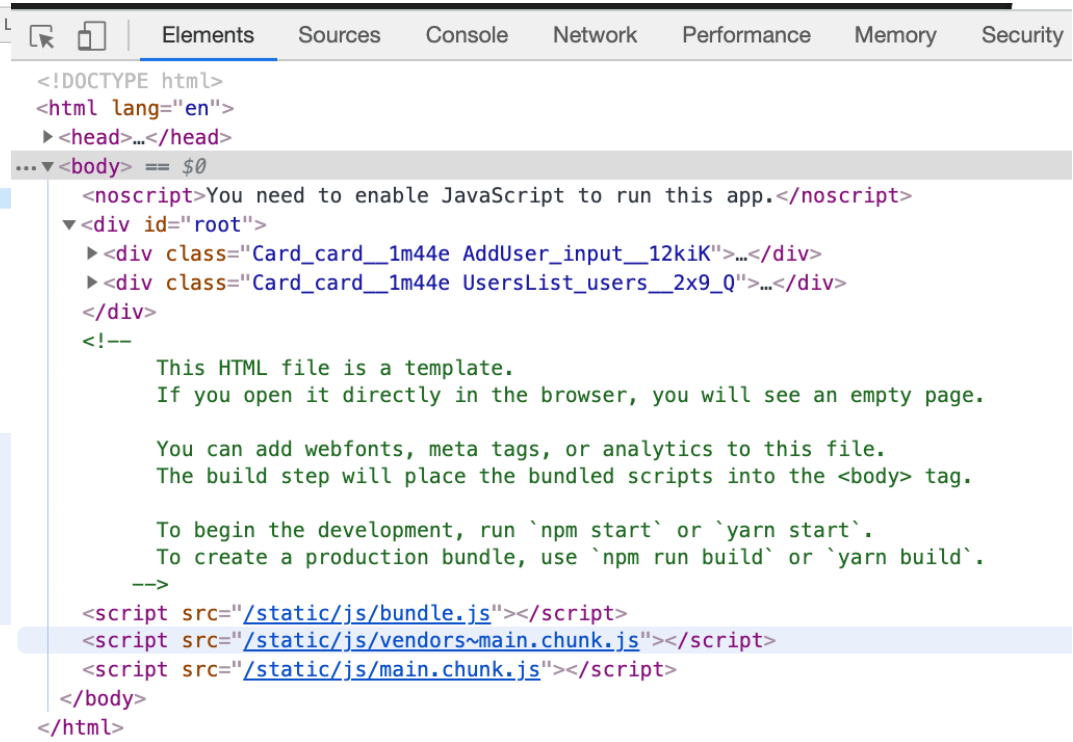


React Fragments – Built in wrapper.

```jsx
import React, { useState } from 'react';

import AddUser from './components/Users/AddUser';
import UsersList from './components/Users/UsersList';

function App() {
  const [usersList, setUsersList] = useState([]);

  const addUserHandler = (uName, uAge) => {
    setUsersList((prevUsersList) => {
      return [
        ...prevUsersList,
        { name: uName, age: uAge, id: Math.random().toString() },
      ];
    });
  };

  return (
    <>
      <AddUser onAddUser={addUserHandler} />
      <UsersList users={usersList} />
    </>
  );
}

export default App;
```

React Fragments – Built in wrapper.





React Fragments – Alternate Syntax.

```jsx
import React, { useState } from 'react';

import AddUser from './components/Users/AddUser';
import UsersList from './components/Users/UsersList';

function App() {
  const [usersList, setUsersList] = useState([]);

  const addUserHandler = (uName, uAge) => {
    setUsersList((prevUsersList) => {
      return [
        ...prevUsersList,
        { name: uName, age: uAge, id: Math.random().toString() },
      ];
    });
  };

  return (
    <React.Fragment>
      <AddUser onAddUser={addUserHandler} />
      <UsersList users={usersList} />
    </React.Fragment>
  );
}

function App(): JSX.Element

export default App;
```

React Fragments.

```jsx
import React, { useState, Fragment } from 'react';

import AddUser from './components/Users/AddUser';
import UsersList from './components/Users/UsersList';

function App() {
  const [usersList, setUsersList] = useState([]);

  const addUserHandler = (uName, uAge) => {
    setUsersList((prevUsersList) => {
      return [
        ...prevUsersList,
        { name: uName, age: uAge, id: Math.random().toString() },
      ];
    });
  };

  return (
    <Fragment>
      <AddUser onAddUser={addUserHandler} />
      <UsersList users={usersList} />
    </Fragment>
  );
}

export default App;
```

ACADEMIND

Understanding React Portals

It's a bit like styling a like a <button> and adding an event listener to it: It'll work, but it's not a good practice.

```
<div onClick={clickHandler}>Click me, I'm a bad button</div>
```

ACADEMIND

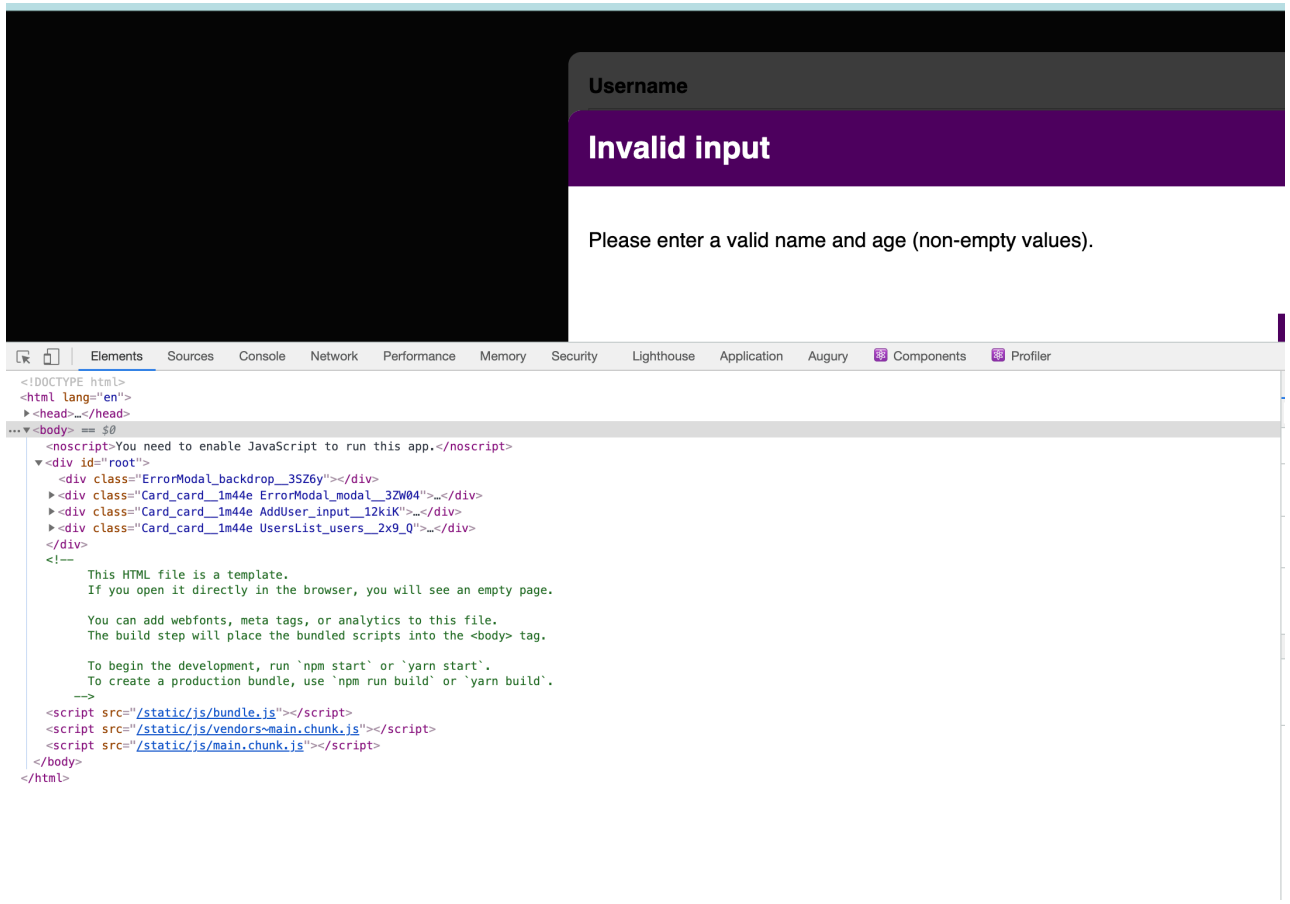
Understanding React Portals

```
return (
  <React.Fragment>
    <MyModal />
    <MyInputForm />
  </React.Fragment>
);
```

Real DOM

```
<div class="my-modal">
  <h2>A Modal Title!</h2>
</div>
<section>
  <h2>Some other content … </h2>
  <form>
    <label>Username</label>
    <input type="text" />
  </form>
</section>
```

React Portals.

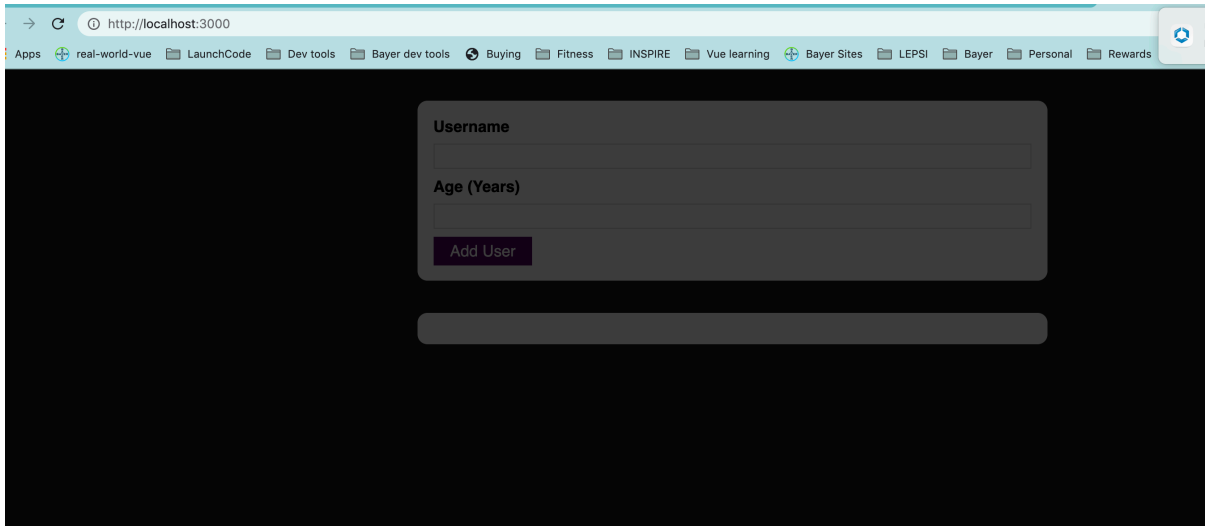


React Portals.

- Portal need a place to port the component.
- Let the component know that it should have a portal to that place.
- Make a change in the index.html in the public folder.

```
    -->
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="backdrop-root"></div>
    <div id="overlay-root"></div>
    <div id="root"></div>
```

```jsx
import Button from './Button';
import classes from './ErrorModal.module.css';

const Backdrop = (props) => {
  return <div className={classes.backdrop} onClick={props.onConfirm} />;
};

const ModalOverlay = (props) => {
  return (
    <Card className={classes.modal}>
      <header className={classes.header}>
        <h2>{props.title}</h2>
      </header>
      <div className={classes.content}>
        <p>{props.message}</p>
      </div>
      <footer className={classes.actions}>
        <Button onClick={props.onConfirm}>Okay</Button>
      </footer>
    </Card>
  );
};

const ErrorModal = (props) => {
  return <>
    {}
  </>;
};

export default ErrorModal;
```

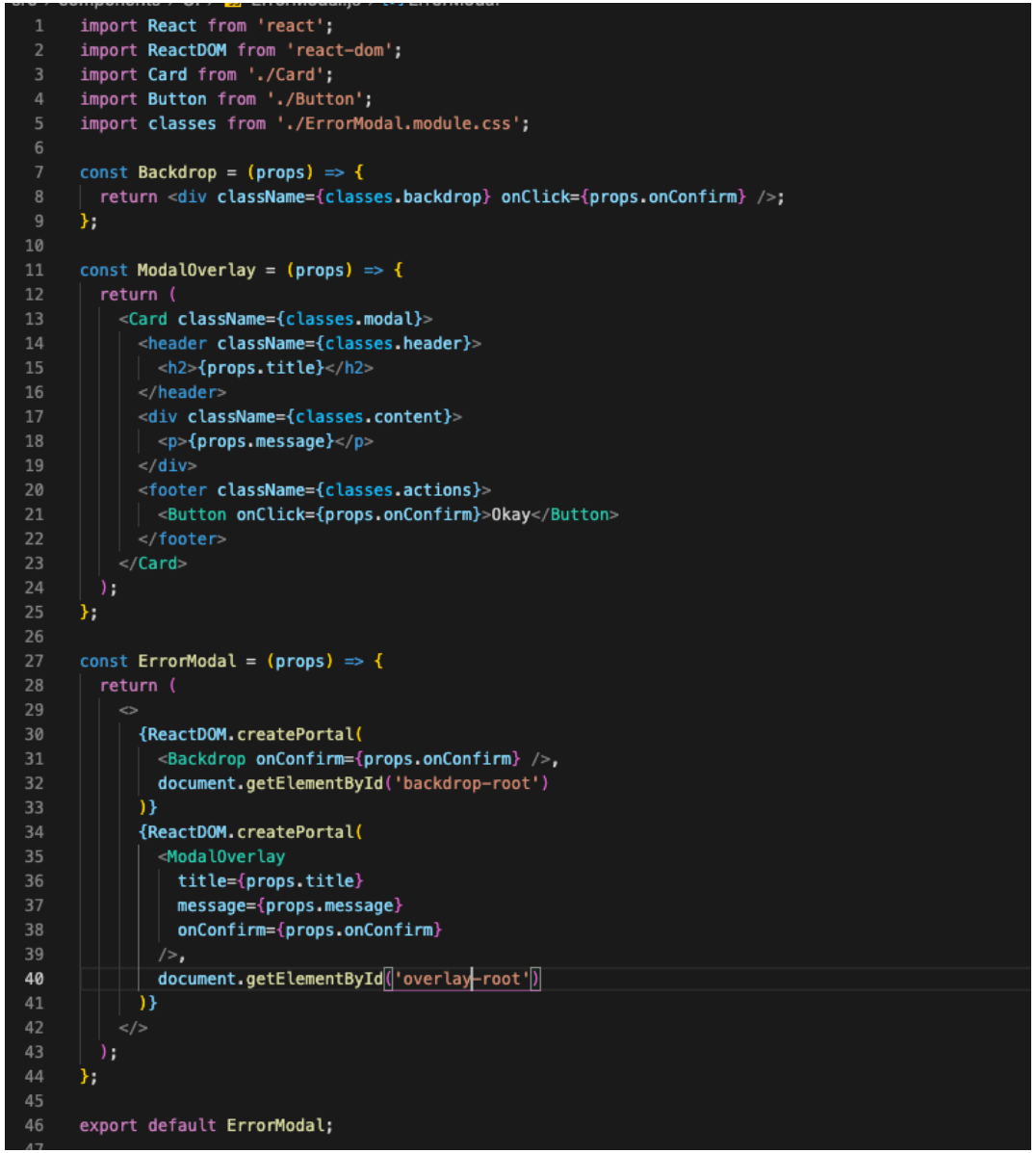

http://localhost:3000
Apps
real-world-vue
LaunchCode
Dev tools
Bayer dev tools
Buying
Fitness
INSPIRE
Vue learning
Bayer Sites
LEPSI
Bayer
Personal
Rewards
Username
Age (Years)
Add User

Username

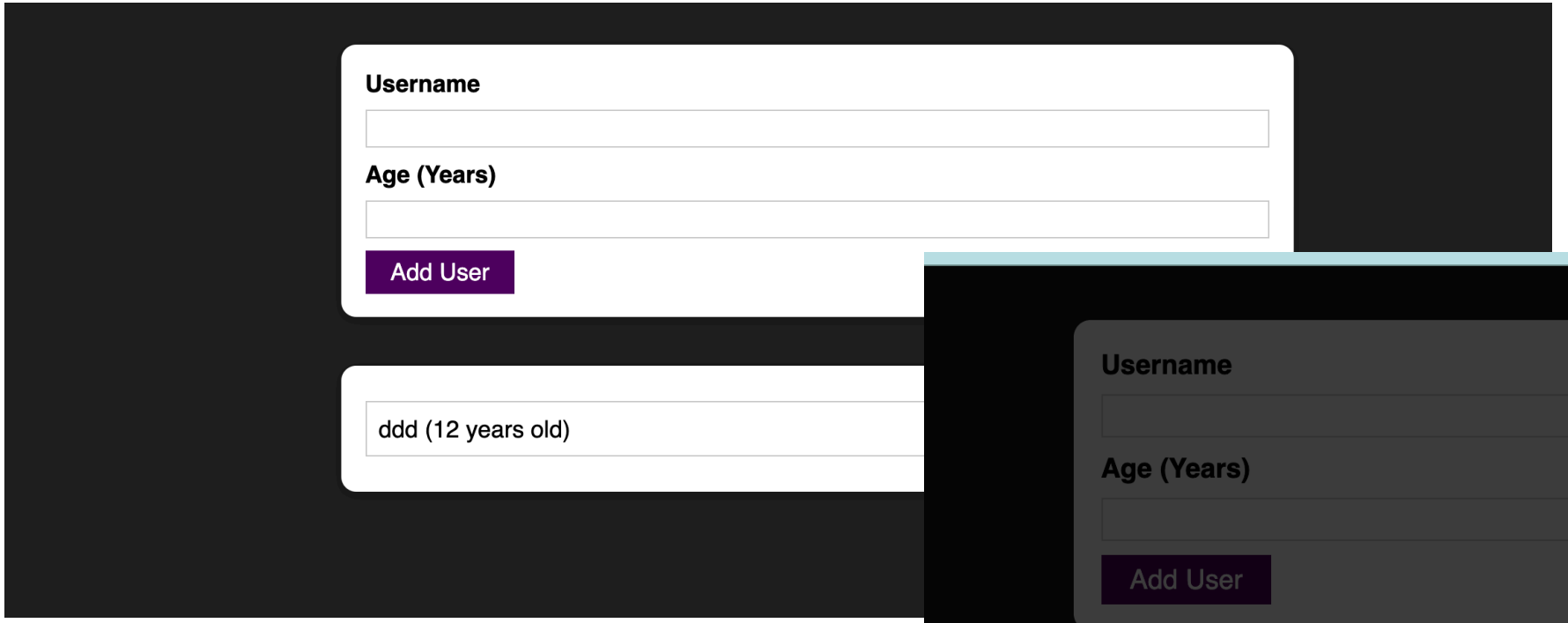
Age (Years)

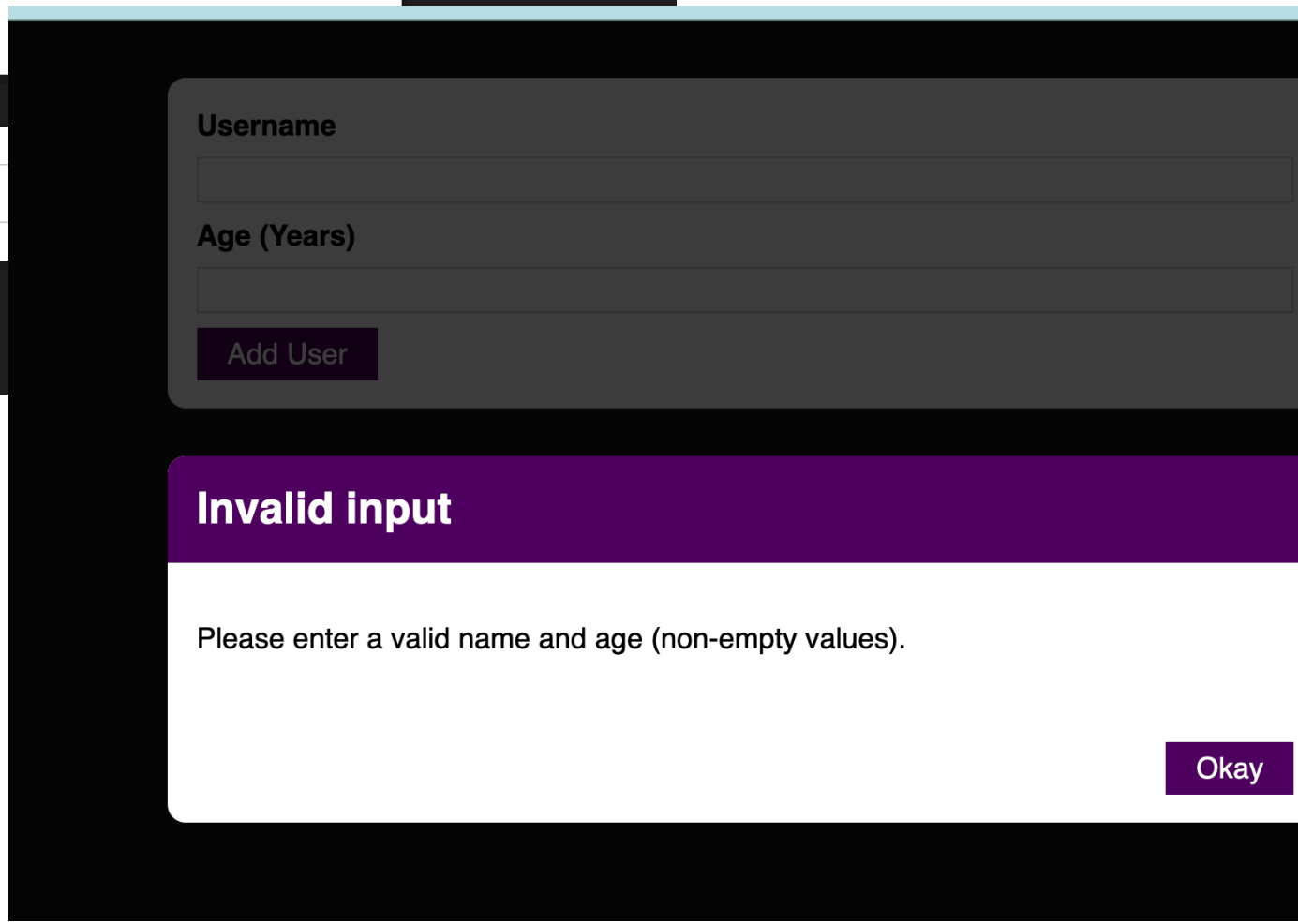
Add User

ddd (12 years old)



```jsx
import React from 'react';
import ReactDOM from 'react-dom';
import Card from './Card';
import Button from './Button';
import classes from './ErrorModal.module.css';

const Backdrop = (props) => {
  return <div className={classes.backdrop} onClick={props.onConfirm} />;
};

const ModalOverlay = (props) => {
  return (
    <Card className={classes.modal}>
      <header className={classes.header}>
        <h2>{props.title}</h2>
      </header>
      <div className={classes.content}>
        <p>{props.message}</p>
      </div>
      <footer className={classes.actions}>
        <Button onClick={props.onConfirm}>Okay</Button>
      </footer>
    </Card>
  );
};

const ErrorModal = (props) => {
  return (
    <>
      {ReactDOM.createPortal(
        <Backdrop onConfirm={props.onConfirm} />,
        document.getElementById('backdrop-root')
      )}
      {ReactDOM.createPortal(
        <ModalOverlay
          title={props.title}
          message={props.message}
          onConfirm={props.onConfirm}
        />,
        document.getElementById('overlay-root')
      )}
    </>
  );
};

export default ErrorModal;
```


Username
Age (Years)
Add User
ddd (12 years old)


Username
Age (Years)
Add User
Invalid input
Please enter a valid name and age (non-empty values).
Okay

Working with refs.

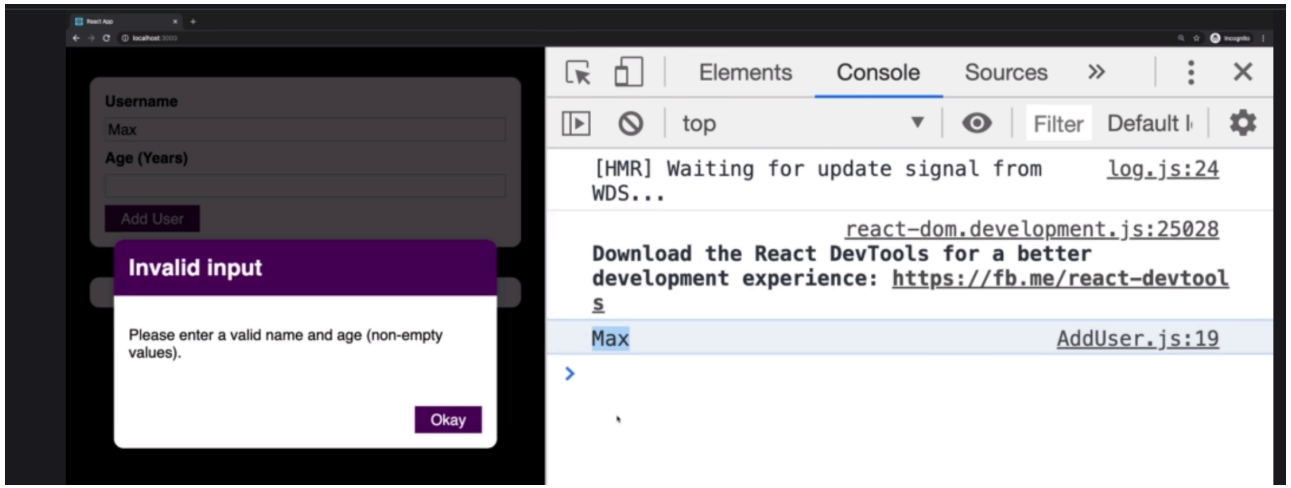
- Powerful – allow us to get access to other DOM elements and use them.
- useRef is a hook that is only accessible inside of a functional component.
- useRef takes a default value to initialize itself to, and it returns a value that let's us work with the element it connects to later.

```
import React, { useState, useRef } from 'react';

import Card from '../UI/Card';
import Button from '../UI/Button';
import ErrorModal from '../UI/ErrorModal';
import classes from './AddUser.module.css';
import Wrapper from '../Helpers/Wrapper';
const AddUser = (props) => {
  const nameInputRef = useRef();
  const ageInputRef = useRef();
  const [enteredUsername, setEnteredUsername] = useState('');
  const [enteredAge, setEnteredAge] = useState('');
  const [error, setError] = useState();
```

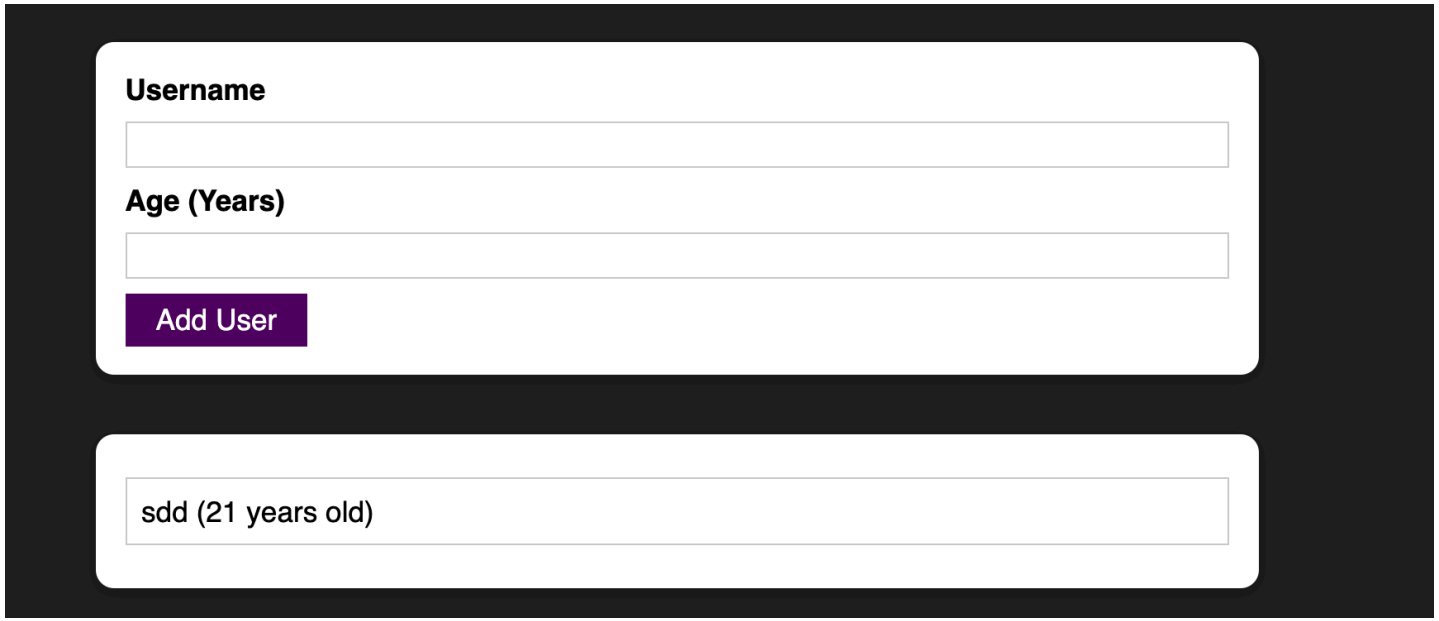
```
<input
  id="username"
  type="text"
  value={enteredUsername}
  onChange={usernameChangeHandler}
  ref={nameInputRef}
/>
<label htmlFor="age">Age (Years)</label>
<input
  id="age"
  type="number"
  value={enteredAge}
  onChange={ageChangeHandler}
  ref={ageInputRef}
/>
```

Working with refs.

- It holds as an object that has a current value.
- DOM node being stored.
- Should not be manipulated. React should manipulate the DOM. However, you can read data. nameInputRef.current.value
- Don’t need state.



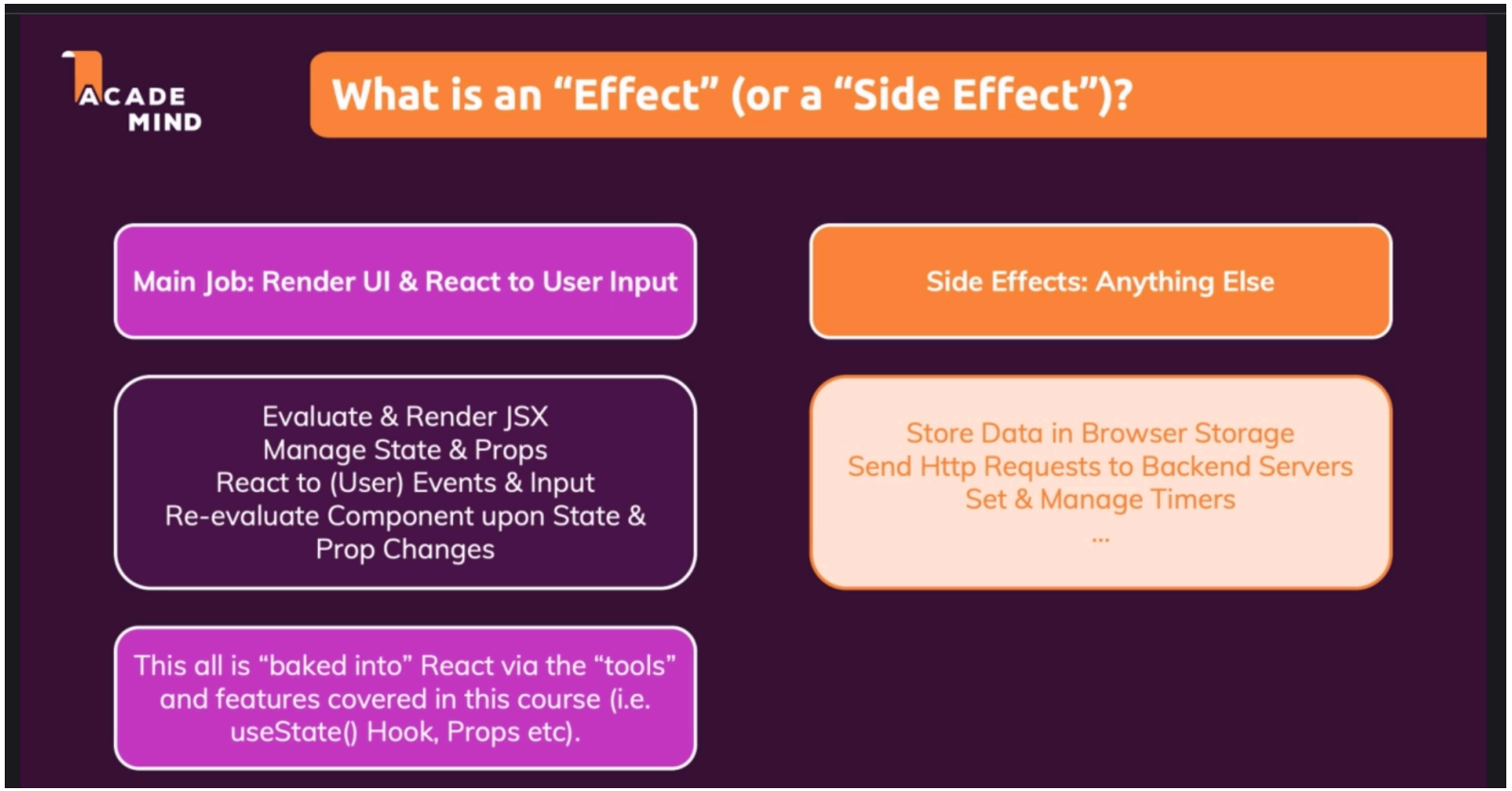
```jsx
import React, { useState, useRef } from 'react';

import Card from '../UI/Card';
import Button from '../UI/Button';
import ErrorModal from '../UI/ErrorModal';
import classes from './AddUser.module.css';
import Wrapper from '../Helpers/Wrapper';
const AddUser = (props) => {
  const nameInputRef = useRef();
  const ageInputRef = useRef();
  const [error, setError] = useState();

  const addUserHandler = (event) => {
    event.preventDefault();
    const enteredName = nameInputRef.current.value;
    const enteredUserAge = ageInputRef.current.value;
    if (enteredName.trim().length === 0 || enteredUserAge.trim().length === 0) {
      setError({
        title: 'Invalid input',
        message: 'Please enter a valid name and age (non-empty values).',
      });
      return;
    }
    if (+enteredUserAge < 1) {
      setError({
        title: 'Invalid age',
        message: 'Please enter a valid age (> 0).',
      });
      return;
    }
    props.onAddUser(enteredName, enteredUserAge);
    nameInputRef.current.value = '';
    ageInputRef.current.value = '';
  };
  const errorHandler = () => {
    setError(null);
  };

  return (
    <Wrapper>
      {error && (
        <ErrorModal
          title={error.title}
          message={error.message}
          onConfirm={errorHandler}
        />
      )}
      <Card className={classes.input}>
        <form onSubmit={addUserHandler}>
          <label htmlFor="username">Username</label>
          <input id="username" type="text" ref={nameInputRef} />
          <label htmlFor="age">Age (Years)</label>
          <input id="age" type="number" ref={ageInputRef} />
          <Button type="submit">Add User</Button>
        </form>
      </Card>
    </Wrapper>
  );
};
```



Controlled vs Uncontrolled Components.

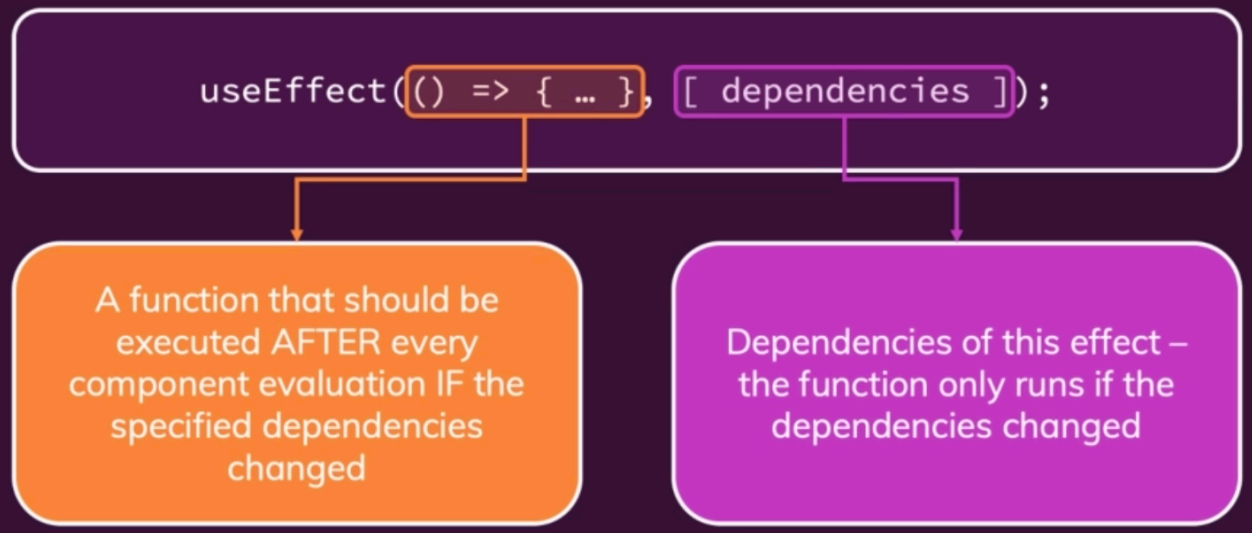
- Input elements internal state not controlled by React. Uncontrolled Components
- Internal State – when we work with component state. Controlled Components. Internal state is controlled by React.


ACADEMIND
Module Content
Working with (Side) Effects
Managing more Complex State with Reducers
Managing App-Wide or Component-Wide State with Context

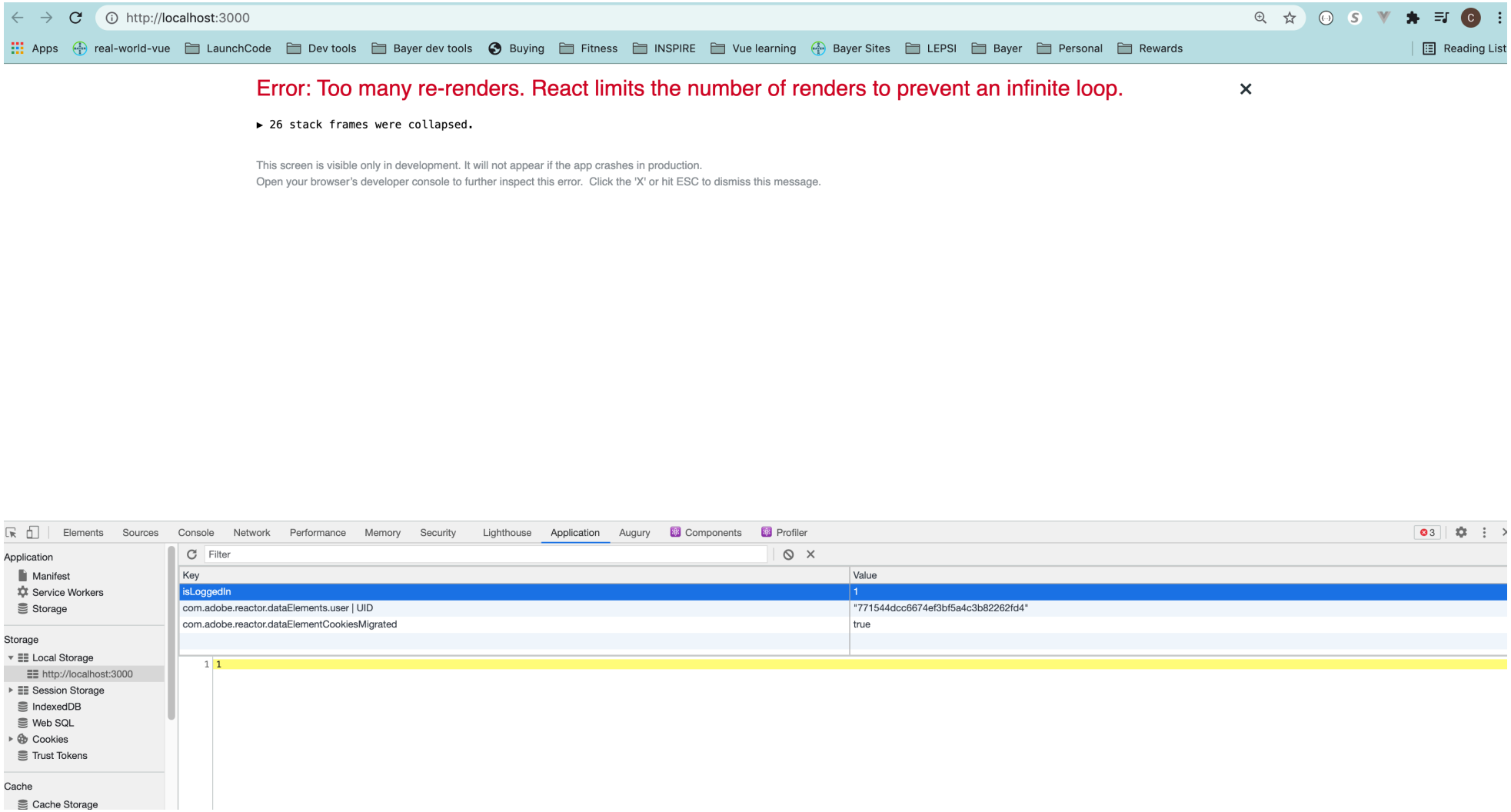

ACADEMIND
What is an “Effect” (or a “Side Effect”)?
Main Job: Render UI & React to User Input
Evaluate & Render JSX
Manage State & Props
React to (User) Events & Input
Re-evaluate Component upon State & Prop Changes
This all is “baked into” React via the “tools” and features covered in this course (i.e. useState() Hook, Props etc).
Side Effects: Anything Else
Store Data in Browser Storage
Send Http Requests to Backend Servers
Set & Manage Timers
...

ACADEMIND

Handling Side Effects with the useEffect() Hook



```jsx
import React, { useState } from 'react';

import Login from './components/Login/Login';
import Home from './components/Home/Home';
import MainHeader from './components/MainHeader/MainHeader';

function App() {
  const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');

  const [isLoggedIn, setIsLoggedIn] = useState(false);

  if (storedUserLoggedInInformation === '1') {
    setIsLoggedIn(true);
  }

  const loginHandler = (email, password) => {
    // We should of course check email and password
    // But it's just a dummy/ demo anyways
    localStorage.setItem('isLoggedIn', '1');
    setIsLoggedIn(true);
  };

  const logoutHandler = () => {
    setIsLoggedIn(false);
  };

  return (
    <React.Fragment>
      <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />
      <main>
        {!isLoggedIn && <Login onLogin={loginHandler} />}
        {isLoggedIn && <Home onLogout={logoutHandler} />}
      </main>
    </React.Fragment>
  );
}

export default App;

```

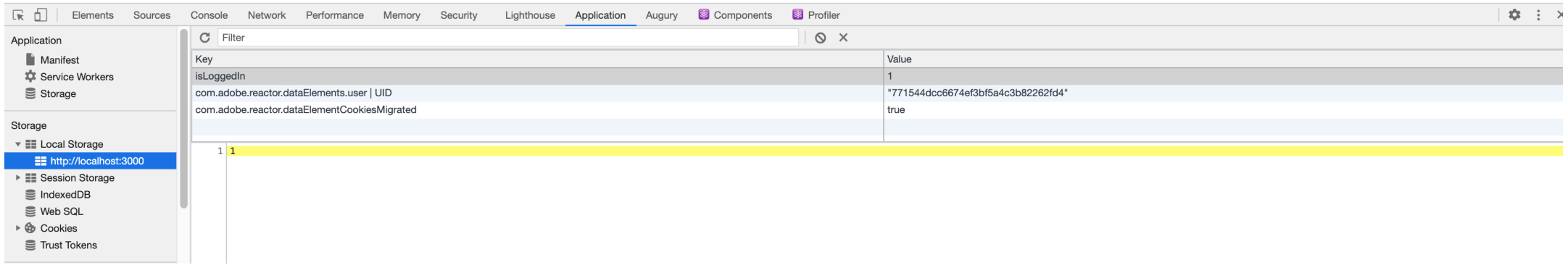

http://localhost:3000
Apps
real-world-vue
LaunchCode
Dev tools
Bayer dev tools
Buying
Fitness
INSPIRE
Vue learning
Bayer Sites
LEPSI
Bayer
Personal
Rewards
Reading List
Error: Too many re-renders. React limits the number of renders to prevent an infinite loop.
▶ 26 stack frames were collapsed.
This screen is visible only in development. It will not appear if the app crashes in production.
Open your browser's developer console to further inspect this error. Click the 'X' or hit ESC to dismiss this message.
Elements
Sources
Console
Network
Performance
Memory
Security
Lighthouse
Application
Augury
Components
Profiler
Application
Manifest
Service Workers
Storage
Storage
Local Storage
http://localhost:3000
Session Storage
IndexedDB
Web SQL
Cookies
Trust Tokens
Cache
Cache Storage
Filter
Key
Value
isLoggedIn
1
com.adobe.reactor.dataElements.user | UID
"771544dcc6674ef3bf5a4c3b82262fd4"
com.adobe.reactor.dataElementCookiesMigrated
true
1
1

```jsx
import React, { useState, useEffect } from 'react';

import Login from './components/Login/Login';
import Home from './components/Home/Home';
import MainHeader from './components/MainHeader/MainHeader';

function App() {
  const [isLoggedIn, setIsLoggedIn] = useState(false);

  //run this code only once when the app starts up.
  useEffect(() => {
    const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');
    if (storedUserLoggedInInformation === '1') {
      setIsLoggedIn(true);
    }
  }, []);

  const loginHandler = (email, password) => {
    localStorage.setItem('isLoggedIn', '1');
    setIsLoggedIn(true);
  };

  const logoutHandler = () => {
    setIsLoggedIn(false);
  };

  return (
    <React.Fragment>
      <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />
      <main>
        {!isLoggedIn && <Login onLogin={loginHandler} />}
        {isLoggedIn && <Home onLogout={logoutHandler} />}
      </main>
    </React.Fragment>
  );
}

export default App;

```

A Typical Page

Users Admin Logout

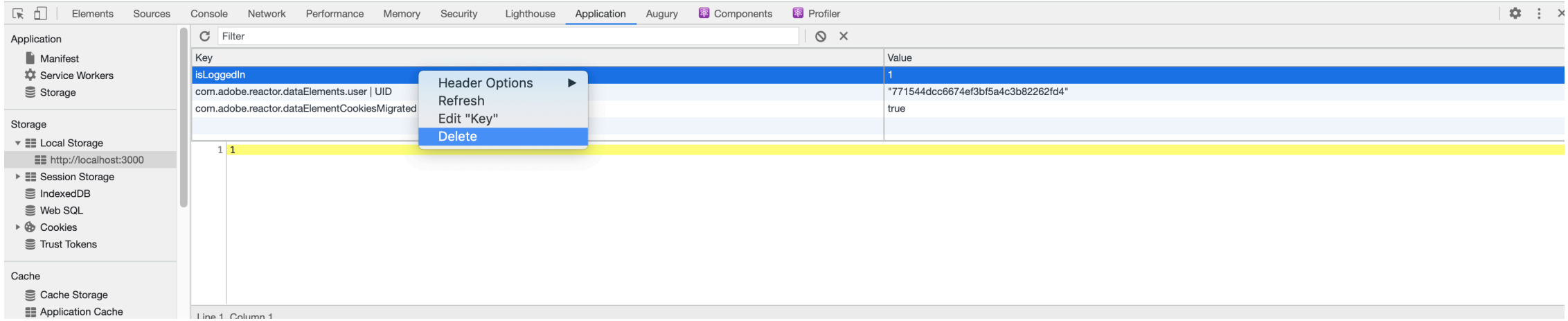
Welcome back!

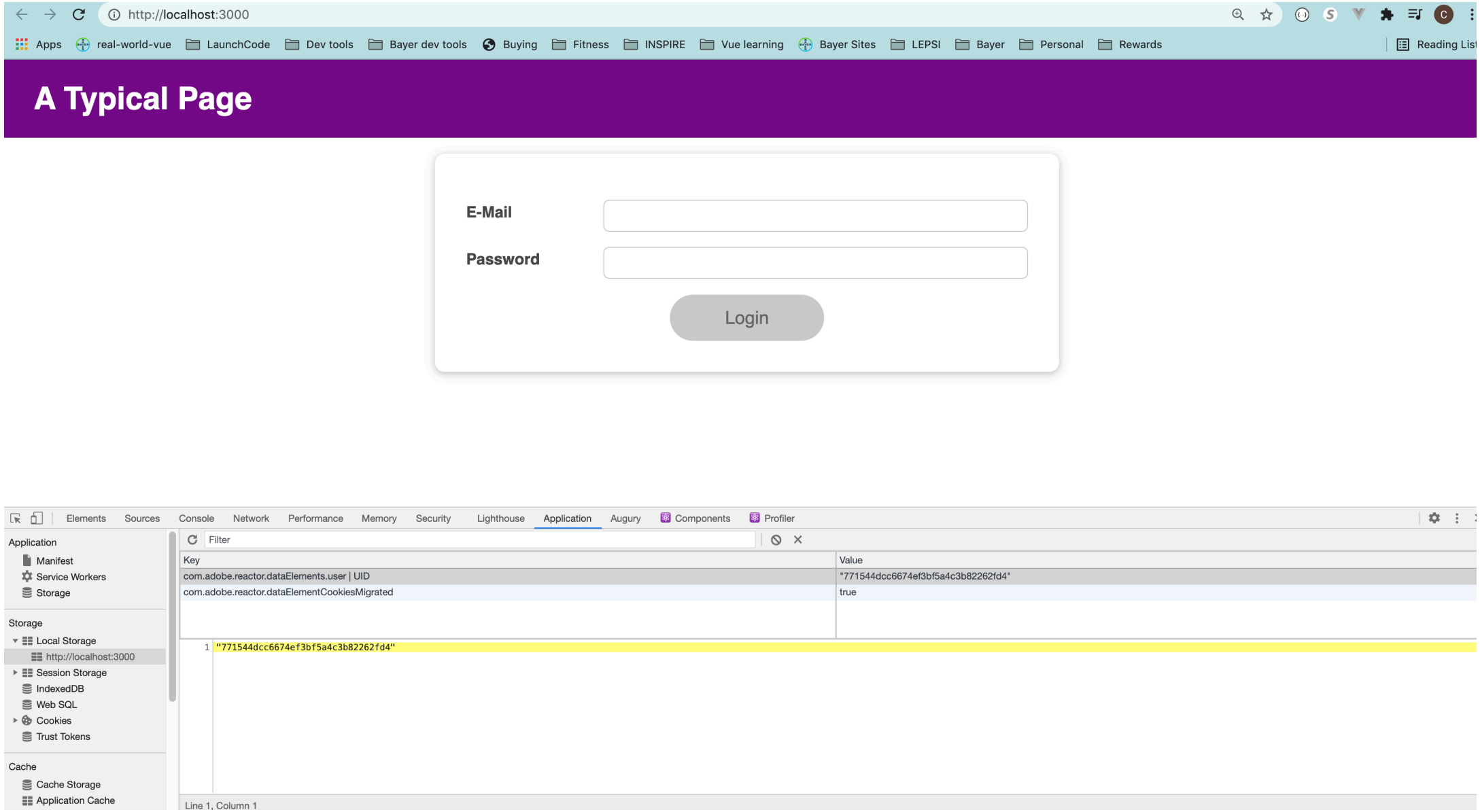


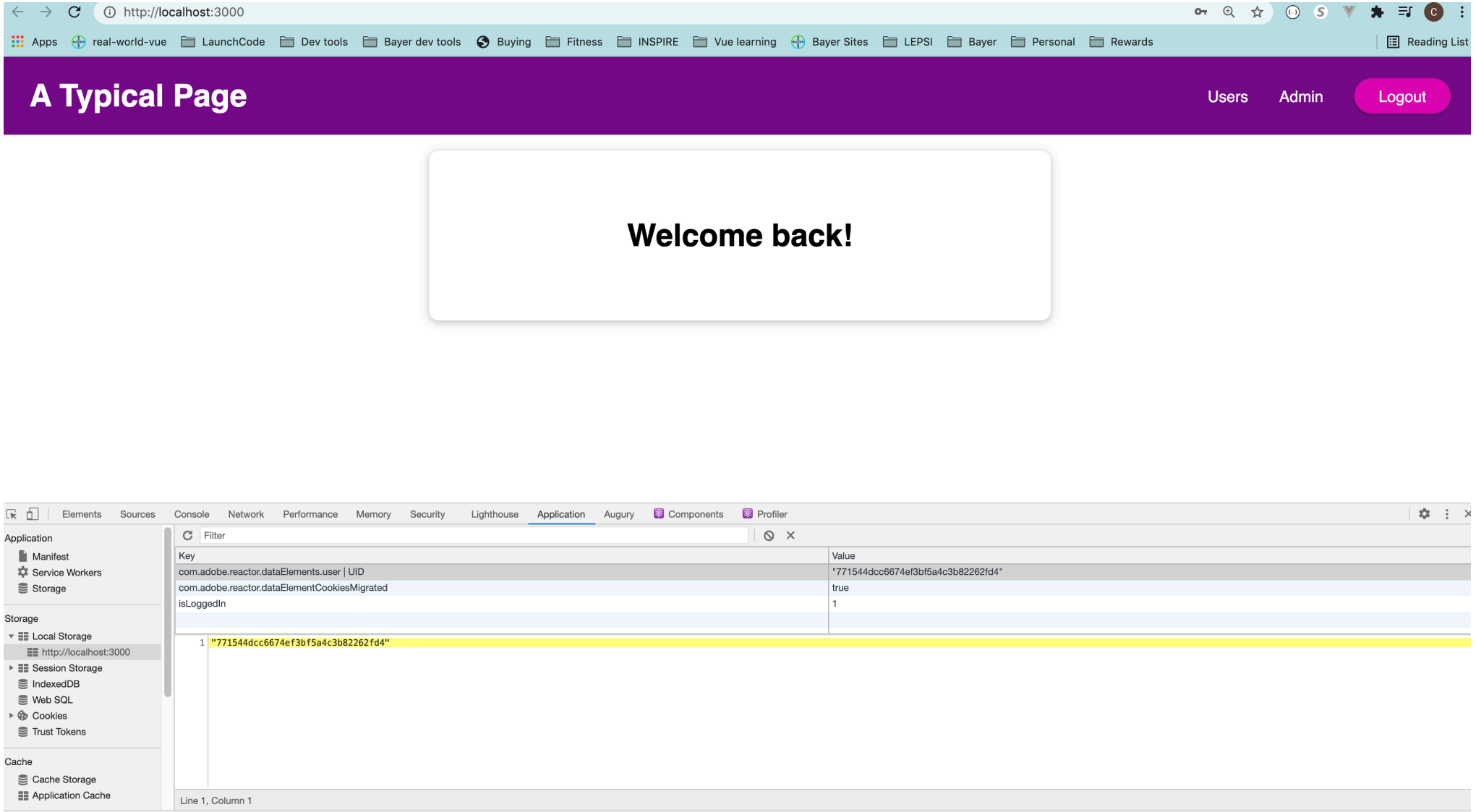
A Typical Page

Users Admin Logout

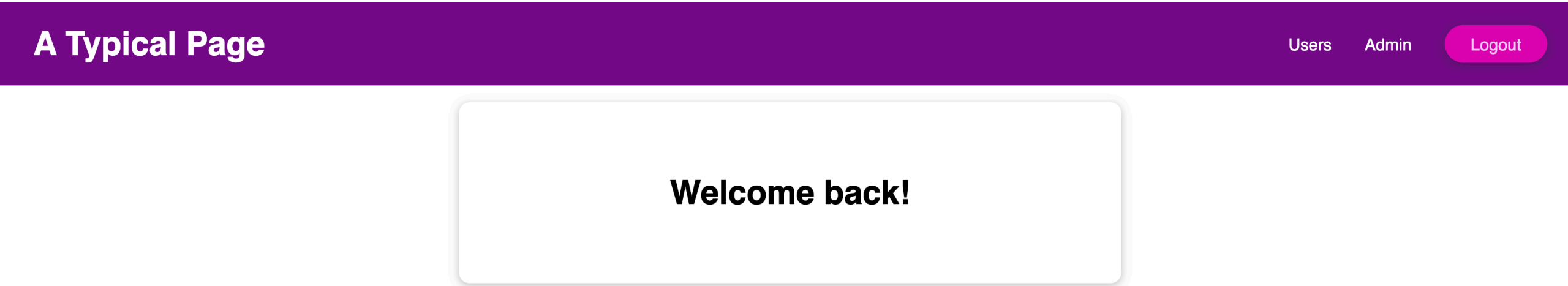
Welcome back!

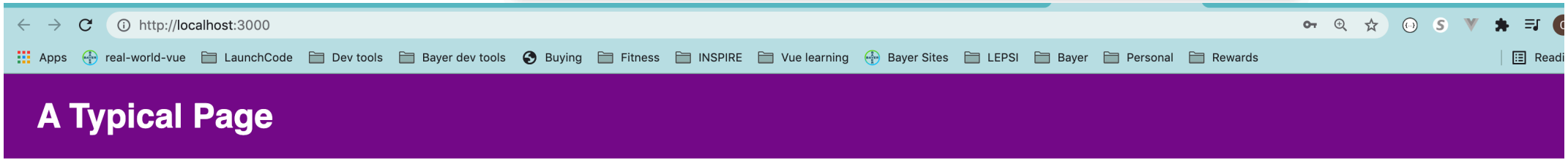


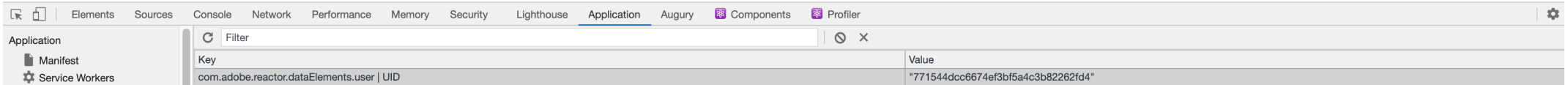

http://localhost:3000
Apps
real-world-vue
LaunchCode
Dev tools
Bayer dev tools
Buying
Fitness
INSPIRE
Vue learning
Bayer Sites
LEPSI
Bayer
Personal
Rewards
Reading List
A Typical Page
E-Mail
Password
Login
Elements
Sources
Console
Network
Performance
Memory
Security
Lighthouse
Application
Augury
Components
Profiler
Application
Manifest
Service Workers
Storage
Storage
Local Storage
http://localhost:3000
Session Storage
IndexedDB
Web SQL
Cookies
Trust Tokens
Cache
Cache Storage
Application Cache
Filter
Key
Value
com.adobe.reactor.dataElements.user | UID
"771544dcc6674ef3bf5a4c3b82262fd4"
com.adobe.reactor.dataElementCookiesMigrated
true
1 "771544dcc6674ef3bf5a4c3b82262fd4"
Line 1, Column 1


http://localhost:3000
Apps
real-world-vue
LaunchCode
Dev tools
Bayer dev tools
Buying
Fitness
INSPIRE
Vue learning
Bayer Sites
LEPSI
Bayer
Personal
Rewards
Reading List
A Typical Page
Users
Admin
Logout
Welcome back!
Elements
Sources
Console
Network
Performance
Memory
Security
Lighthouse
Application
Augury
Components
Profiler
Application
Manifest
Service Workers
Storage
Storage
Local Storage
http://localhost:3000
Session Storage
IndexedDB
Web SQL
Cookies
Trust Tokens
Cache
Cache Storage
Application Cache
Filter
Key
Value
com.adobe.reactor.dataElements.user | UID
"771544dcc6674ef3bf5a4c3b82262fd4"
com.adobe.reactor.dataElementCookiesMigrated
true
isLoggedIn
1
1 "771544dcc6674ef3bf5a4c3b82262fd4"
Line 1, Column 1

```jsx
import React, { useState, useEffect } from 'react';

import Login from './components/Login/Login';
import Home from './components/Home/Home';
import MainHeader from './components/MainHeader/MainHeader';

function App() {
  const [isLoggedIn, setIsLoggedIn] = useState(false);

  //run this code only once when the app starts up.
  useEffect(() => {
    const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');
    if (storedUserLoggedInInformation === '1') {
      setIsLoggedIn(true);
    }
  }, []);

  const loginHandler = (email, password) => {
    localStorage.setItem('isLoggedIn', '1');
    setIsLoggedIn(true);
  };

  const logoutHandler = () => {
    localStorage.removeItem('isLoggedIn');
    setIsLoggedIn(false);
  };

  return (
    <React.Fragment>
      <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />
      <main>
        {!isLoggedIn && <Login onLogin={loginHandler} />}
        {isLoggedIn && <Home onLogout={logoutHandler} />}
      </main>
    </React.Fragment>
  );
}

export default App;
```


A Typical Page
Users
Admin
Logout
Welcome back!


http://localhost:3000
Apps
real-world-vue
LaunchCode
Dev tools
Bayer dev tools
Buying
Fitness
INSPIRE
Vue learning
Bayer Sites
LEPSI
Bayer
Personal
Rewards
A Typical Page
E-Mail
Password
Login


Elements
Sources
Console
Network
Performance
Memory
Security
Lighthouse
Application
Augury
Components
Profiler
Application
Manifest
Service Workers
Filter
Key
Value
com.adobe.reactor.dataElements.user | UID
"771544dcc6674ef3bf5a4c3b82262fd4"

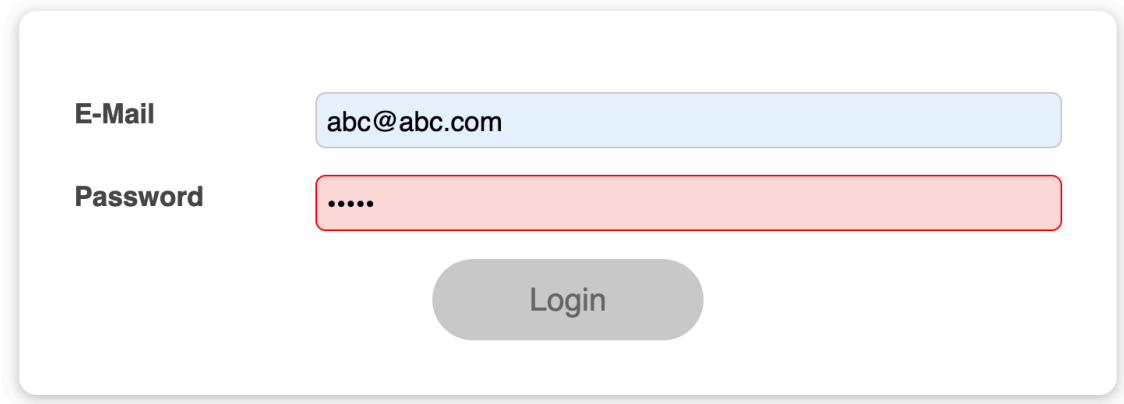
useEffect for Login.

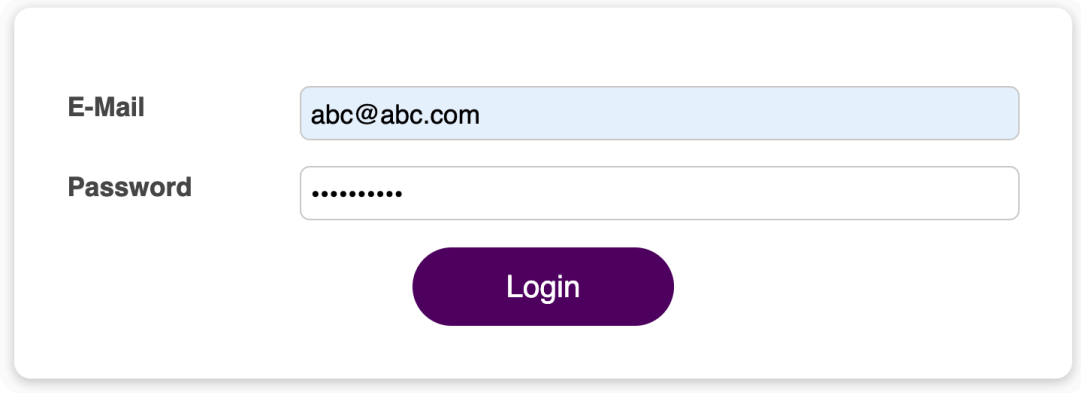
- Logic for checking validity of for (email address and password) can be combines into one useEffect.
- useEffect is used for side effects

c > components > Login > Login.js > Login > emailChangeHandler

```
import React, { useState, useEffect } from 'react';

import Card from '../UI/Card/Card';
import classes from './Login.module.css';
import Button from '../UI/Button/Button';

const Login = (props) => {
  const [enteredEmail, setEnteredEmail] = useState('');
  const [emailIsValid, setEmailIsValid] = useState();
  const [enteredPassword, setEnteredPassword] = useState('');
  const [passwordIsValid, setPasswordIsValid] = useState();
  const [formIsValid, setFormIsValid] = useState(false);

  useEffect(() => {
    setFormIsValid(
      enteredEmail.includes('@') && enteredPassword.trim().length > 6
    );
  }, [enteredEmail, enteredPassword]);

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

  const passwordChangeHandler = (event) => {
    setEnteredPassword(event.target.value);
  };

  const validateEmailHandler = () => {
    setEmailIsValid(enteredEmail.includes('@'));
  };

```

A Typical Page

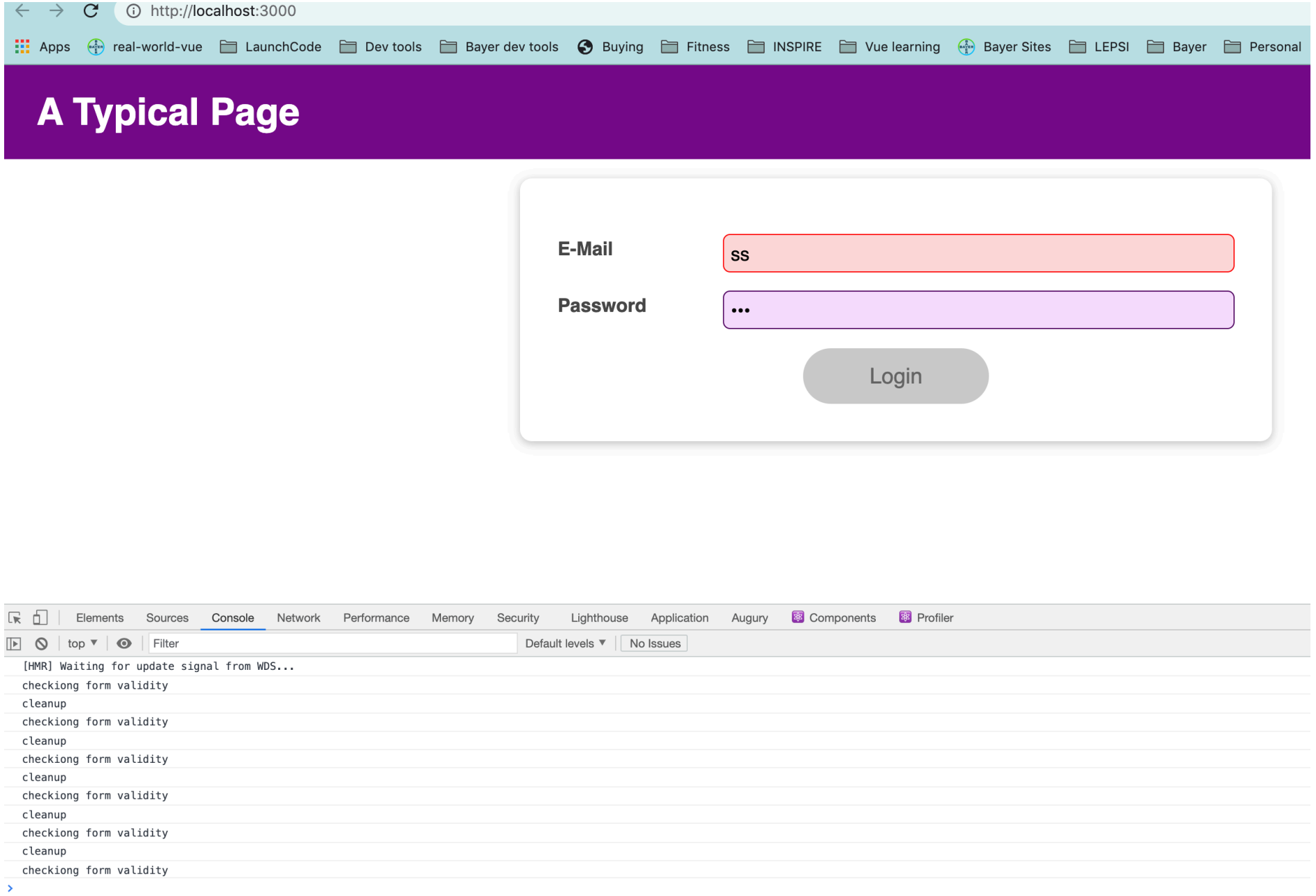




useEffect.

- Don’t want to do something with every keystroke.
- For example, when the user stops typing.
- This technique is called debouncing.
- useEffect can return an anonymous function. This is called a cleanup function.
- Cleanup function runs Before useEffect function runs the next time.
- Before component is removed(unmounted).
- It is a very important React hook.

```jsx
import React, { useState, useEffect } from 'react';

import Card from '../UI/Card/Card';
import classes from './Login.module.css';
import Button from '../UI/Button/Button';

const Login = (props) => {
  const [enteredEmail, setEnteredEmail] = useState('');
  const [emailIsValid, setEmailIsValid] = useState();
  const [enteredPassword, setEnteredPassword] = useState('');
  const [passwordIsValid, setPasswordIsValid] = useState();
  const [formIsValid, setFormIsValid] = useState(false);

  useEffect(() => {
    console.log('checkiong form validity');
    const identifier = setTimeout(() => {
      setFormIsValid(
        enteredEmail.includes('@') && enteredPassword.trim().length > 6
      );
    }, 500);
    return () => {
      console.log('cleanup');
      clearTimeout(identifier);
    };
  }, [enteredEmail, enteredPassword]);

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

  const passwordChangeHandler = (event) => {
    setEnteredPassword(event.target.value);
  };

  const validateEmailHandler = () => {
    setEmailIsValid(enteredEmail.includes('@'));
  };
```


http://localhost:3000
Apps
real-world-vue
LaunchCode
Dev tools
Bayer dev tools
Buying
Fitness
INSPIRE
Vue learning
Bayer Sites
LEPSI
Bayer
Personal
A Typical Page
E-Mail
ss
Password
...
Login
Elements
Sources
Console
Network
Performance
Memory
Security
Lighthouse
Application
Augury
Components
Profiler
top
Filter
Default levels
No Issues
[HMR] Waiting for update signal from WDS...
checkiong form validity
cleanup
checkiong form validity
cleanup
checkiong form validity
cleanup
checkiong form validity
cleanup
checkiong form validity
cleanup
checkiong form validity

```
useEffect(() => {
  console.log('EFFECT RUNNING');
});
```

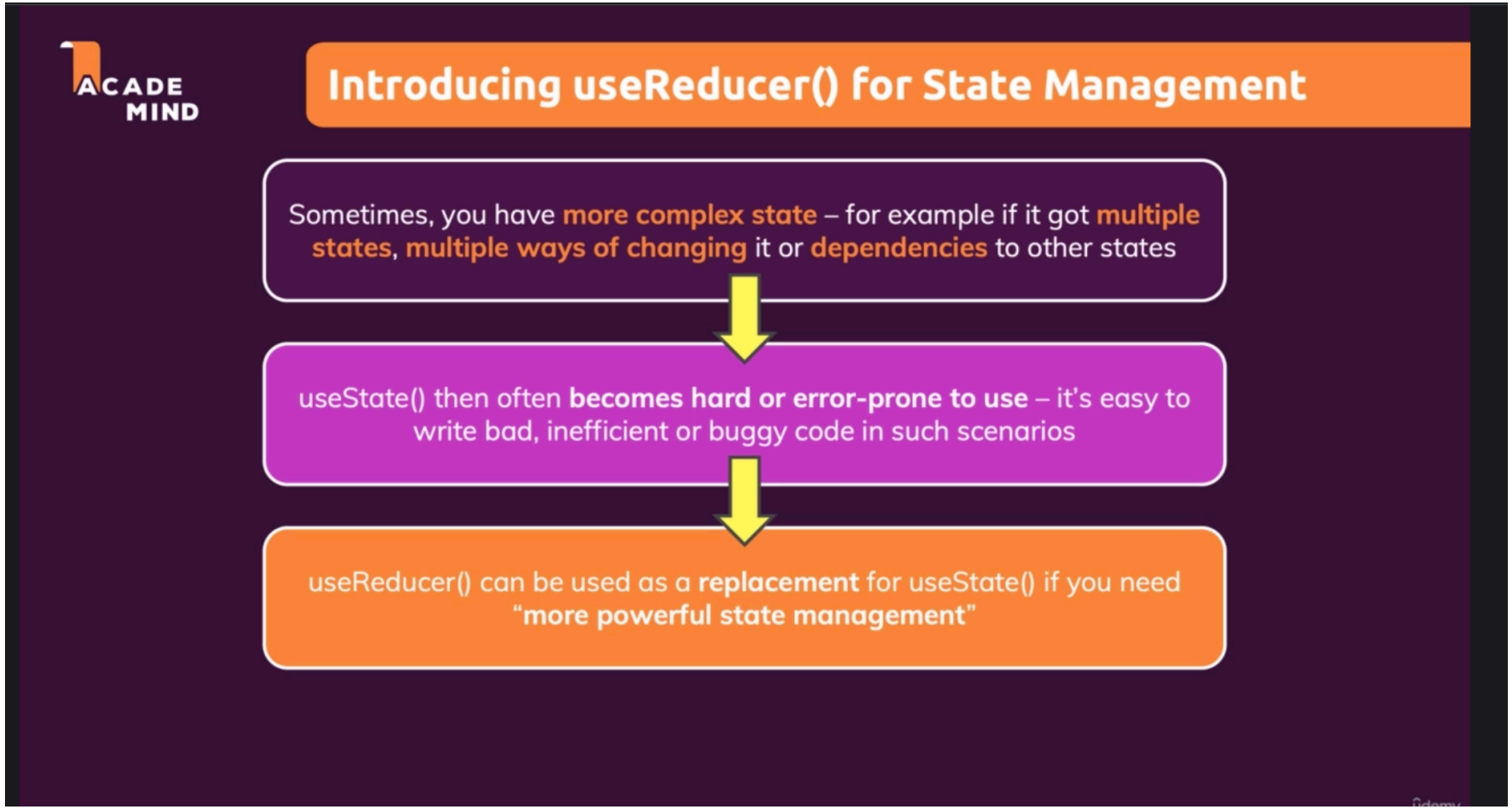
This runs for every component event such is every keystroke, start/stop typing.

```
useEffect(() => {
  console.log('EFFECT RUNNING');
}, []);
```

This runs once.

```
useEffect(() => {
  console.log('EFFECT RUNNING');
}, [enteredEmail]);
```

This runs for change in state for enteredEmail. Keystokes.


ACADEMIND
Introducing useReducer() for State Management
Sometimes, you have more complex state – for example if it got multiple states, multiple ways of changing it or dependencies to other states
useState() then often becomes hard or error-prone to use – it's easy to write bad, inefficient or buggy code in such scenarios
useReducer() can be used as a replacement for useState() if you need "more powerful state management"

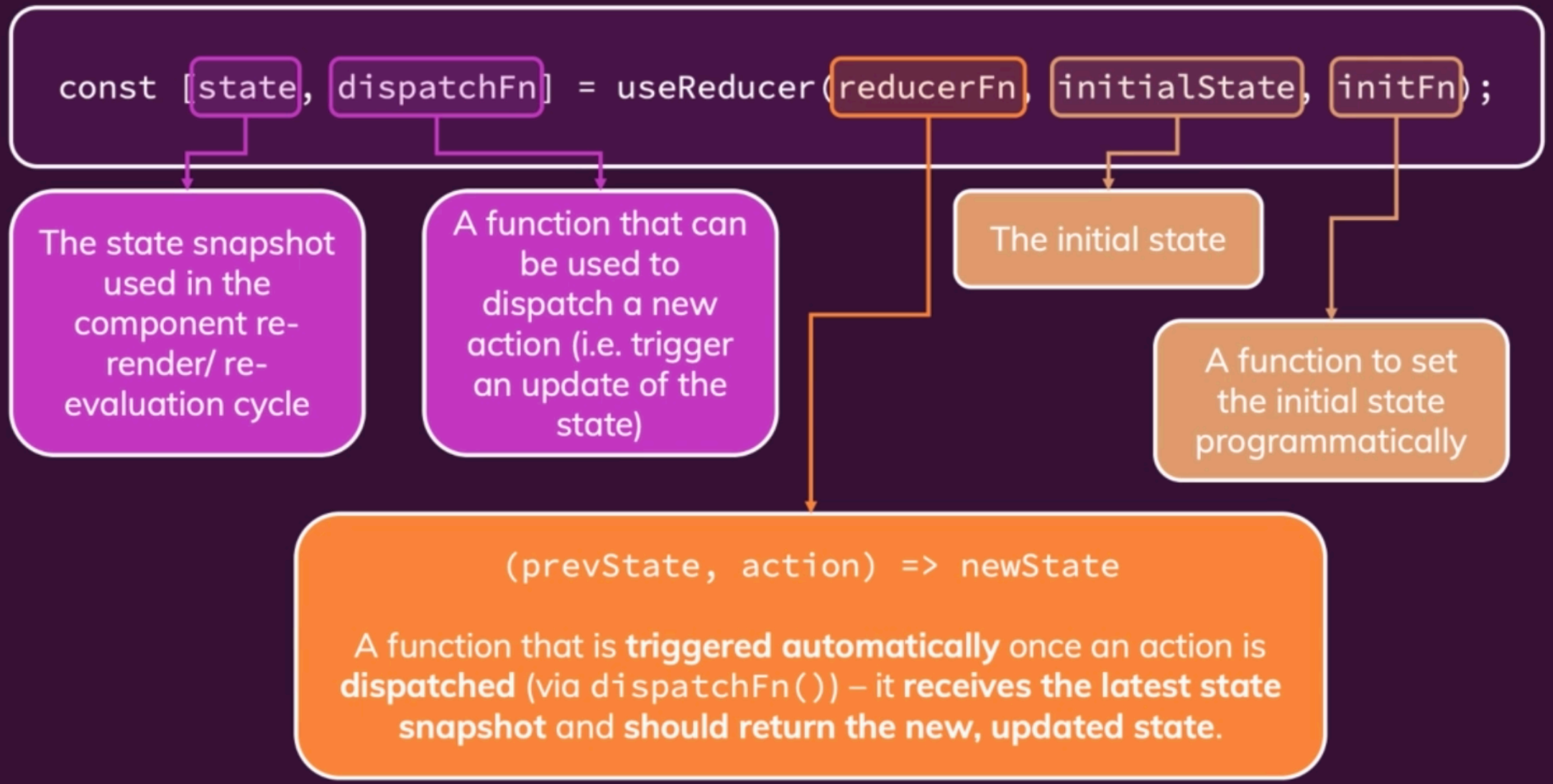
useReducer.

- A lot of state variables

```
const Login = (props) => {
  const [enteredEmail, setEnteredEmail] = useState('');
  const [emailIsValid, setEmailIsValid] = useState();
  const [enteredPassword, setEnteredPassword] = useState('');
  const [passwordIsValid, setPasswordIsValid] = useState();
  const [formIsValid, setFormIsValid] = useState(false);
```

Setting one state based on another state isn't a good practice and may not always work correctly.
Merging these into a state and useReducer instead.

Understanding useReducer()

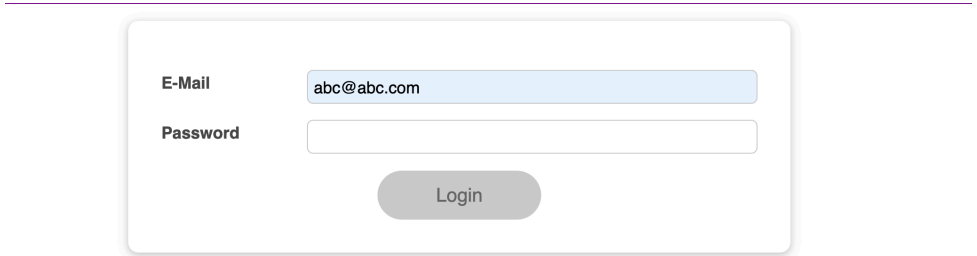


Reducer function.

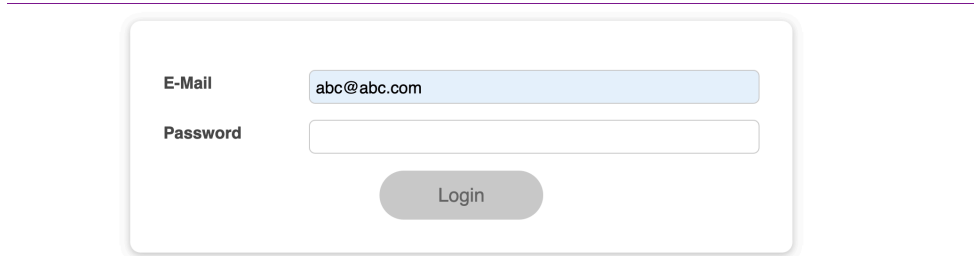
- emailReducer is defined outside of scope of the component function. Doesn’t need anything from within or inside the component itself.

```jsx
import React, { useState, useEffect, useReducer } from 'react';

import Card from '../UI/Card/Card';
import classes from './Login.module.css';
import Button from '../UI/Button/Button';
const emailReducer = (state, action) => {
  if (action.type === 'USER_INPUT') {
    return { value: action.val, isValid: action.val.includes('@') };
  }
  if (action.type === 'INPUT_BLUR') {
    return { value: state.value, isValid: state.value.includes('@') };
  }
  return { value: '', isValid: false };
};

const Login = (props) => {
  // const [enteredEmail, setEnteredEmail] = useState('');
  // const [emailIsValid, setEmailIsValid] = useState();
  const [enteredPassword, setEnteredPassword] = useState('');
  const [passwordIsValid, setPasswordIsValid] = useState();
  const [formIsValid, setFormIsValid] = useState(false);

  const [emailState, dispatchEmail] = useReducer(emailReducer, {
    value: '',
    isValid: false,
  });

  const emailChangeHandler = (event) => {
    dispatchEmail({ type: 'USER_INPUT', val: event.target.value });
    // setEnteredEmail(event.target.value);

    setFormIsValid(emailState.isValid && enteredPassword.trim().length > 6);
  };

  const passwordChangeHandler = (event) => {
    setEnteredPassword(event.target.value);

    setFormIsValid(event.target.value.trim().length > 6 && emailState.isValid);
  };

  const validateEmailHandler = () => {
    // setEmailIsValid(emailState.isValid);
    dispatchEmail({ type: 'INPUT_BLUR' });
  };

  const validatePasswordHandler = () => {
    setPasswordIsValid(enteredPassword.trim().length > 6);
  };

  const submitHandler = (event) => {
    event.preventDefault();
    props.onLogin(emailState.value, enteredPassword);
  };
```

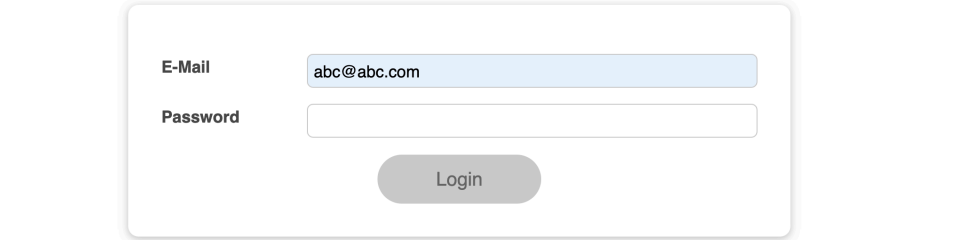
```
return (
  <Card className={classes.login}>
    <form onSubmit={submitHandler}>
      <div
        className={`${classes.control} ${
          emailState.isValid === false ? classes.invalid : ''
        }`}
      >
        <label htmlFor="email">E-Mail</label>
        <input
          type="email"
          id="email"
          value={emailState.value}
          onChange={emailChangeHandler}
          onBlur={validateEmailHandler}
        />
      </div>
      <div
        className={`${classes.control} ${
          passwordIsValid === false ? classes.invalid : ''
        }`}
      >
        <label htmlFor="password">Password</label>
        <input
          type="password"
          id="password"
          value={enteredPassword}
          onChange={passwordChangeHandler}
          onBlur={validatePasswordHandler}
        />
      </div>
      <div className={classes.actions}>
        <Button type="submit" className={classes.btn} disabled={!formIsValid}>
          Login
        </Button>
      </div>
    </form>
  </Card>
);
};

export default Login;
```

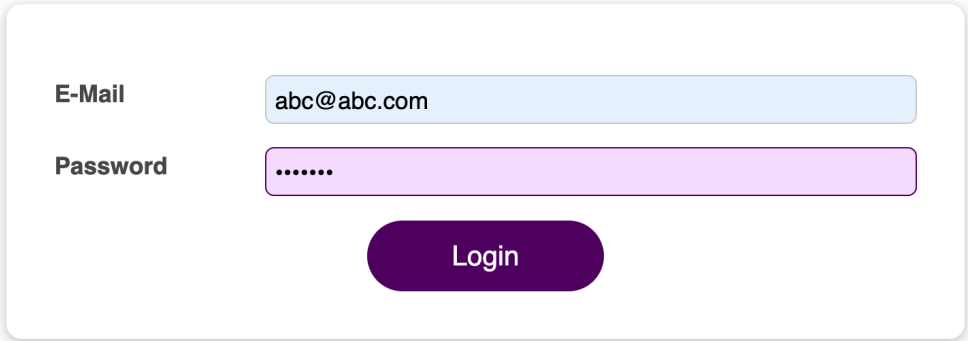


```jsx
import React, { useState, useEffect, useReducer } from 'react';

import Card from '../UI/Card/Card';
import classes from './Login.module.css';
import Button from '../UI/Button/Button';
const emailReducer = (state, action) => {
  if (action.type === 'USER_INPUT') {
    return { value: action.val, isValid: action.val.includes('@') };
  }
  if (action.type === 'INPUT_BLUR') {
    return { value: state.value, isValid: state.value.includes('@') };
  }
  return { value: '', isValid: false };
};

const pwdReducer = (state, action) => {
  if (action.type === 'USER_INPUT') {
    return { value: action.val, isValid: action.val.length > 6 };
  }
  if (action.type === 'INPUT_BLUR') {
    return { value: state.value, isValid: state.value.length > 6 };
  }
  return { value: '', isValid: false };
};

const Login = (props) => {
  // const [enteredEmail, setEnteredEmail] = useState('');
  // const [emailIsValid, setEmailIsValid] = useState();
  // const [enteredPassword, setEnteredPassword] = useState('');
  // const [passwordIsValid, setPasswordIsValid] = useState();
  const [formIsValid, setFormIsValid] = useState(false);

  const [emailState, dispatchEmail] = useReducer(emailReducer, {
    value: '',
    isValid: false,
  });

  const [pwdState, dispatchPwd] = useReducer(pwdReducer, {
    value: '',
    isValid: false,
  });

  const emailChangeHandler = (event) => {
    dispatchEmail({ type: 'USER_INPUT', val: event.target.value });
    // setEnteredEmail(event.target.value);

    setFormIsValid(event.target.value.includes('@') && pwdState.isValid);
  };
```



```
  const passwordChangeHandler = (event) => {
    // setEnteredPassword(event.target.value);
    dispatchPwd({ type: 'USER_INPUT', val: event.target.value });
    setFormIsValid(emailState.isValid && event.target.value.trim().length > 6);
  };

  const validateEmailHandler = () => {
    // setEmailIsValid(emailState.isValid);
    dispatchEmail({ type: 'INPUT_BLUR' });
  };

  const validatePasswordHandler = () => {
    // setPasswordIsValid(enteredPassword.trim().length > 6);
    dispatchPwd({ type: 'INPUT_BLUR' });
  };

  const submitHandler = (event) => {
    event.preventDefault();
    props.onLogin(emailState.value, pwdState.value);
  };

  return (
    <Card className={classes.login}>
      <form onSubmit={submitHandler}>
        <div
          className={`${classes.control} ${
            emailState.isValid === false ? classes.invalid : ''
          }`}
        >
          <label htmlFor="email">E-Mail</label>
          <input
            type="email"
            id="email"
            value={emailState.value}
            onChange={emailChangeHandler}
            onBlur={validateEmailHandler}
          />
        </div>
        <div
          className={`${classes.control} ${
            pwdState.isValid === false ? classes.invalid : ''
          }`}
        >
          <label htmlFor="password">Password</label>
          <input
            type="password"
            id="password"
            value={pwdState.value}
            onChange={passwordChangeHandler}
            onBlur={validatePasswordHandler}
          />
        </div>
        <div className={classes.actions}>
          <Button type="submit" className={classes.btn} disabled={!formIsValid}>
            Login
          </Button>
        </div>
      </form>
    </Card>
  );
};
```





Studio