

# *Front End*

*Class 10*

*September 8, 2021*

# Agenda

- Kahoot
- Code walkthrough with slides
- Studio



## Module Content

JSX Limitations & Fragments

Getting a Cleaner DOM with Portals

Working with Refs



## 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>
);
```

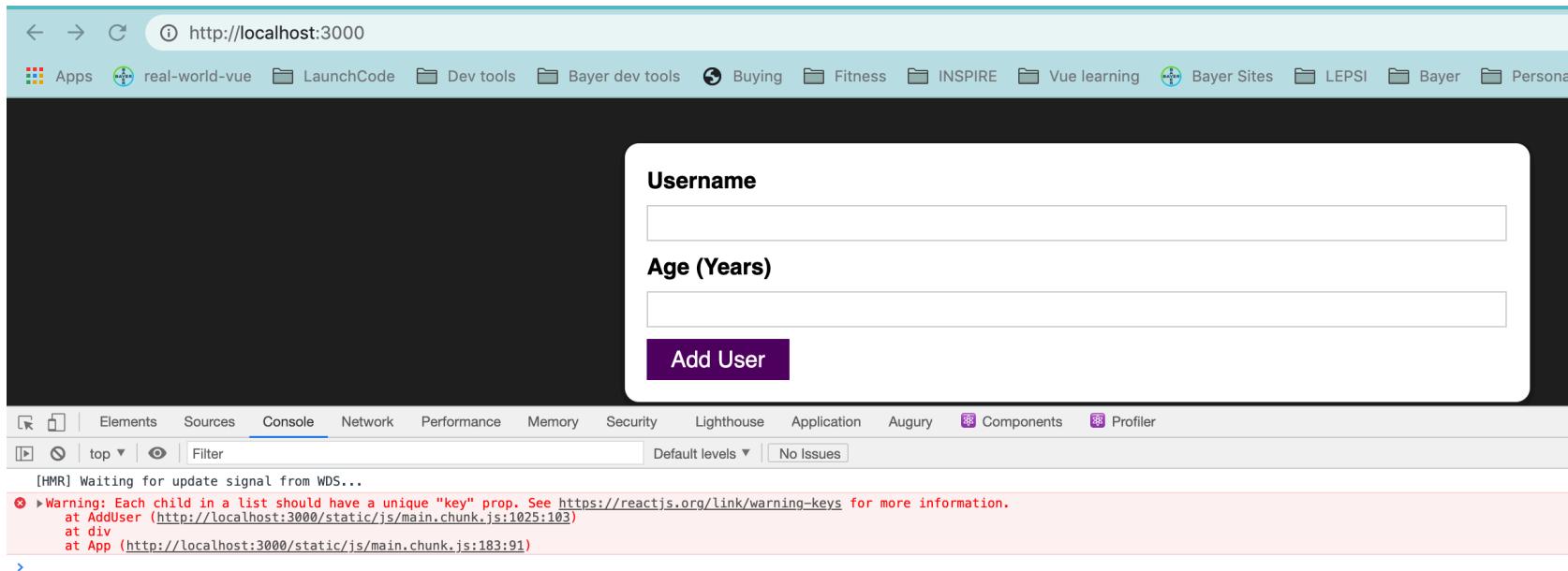
# Array of JSX elements.

- 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>,
];
```

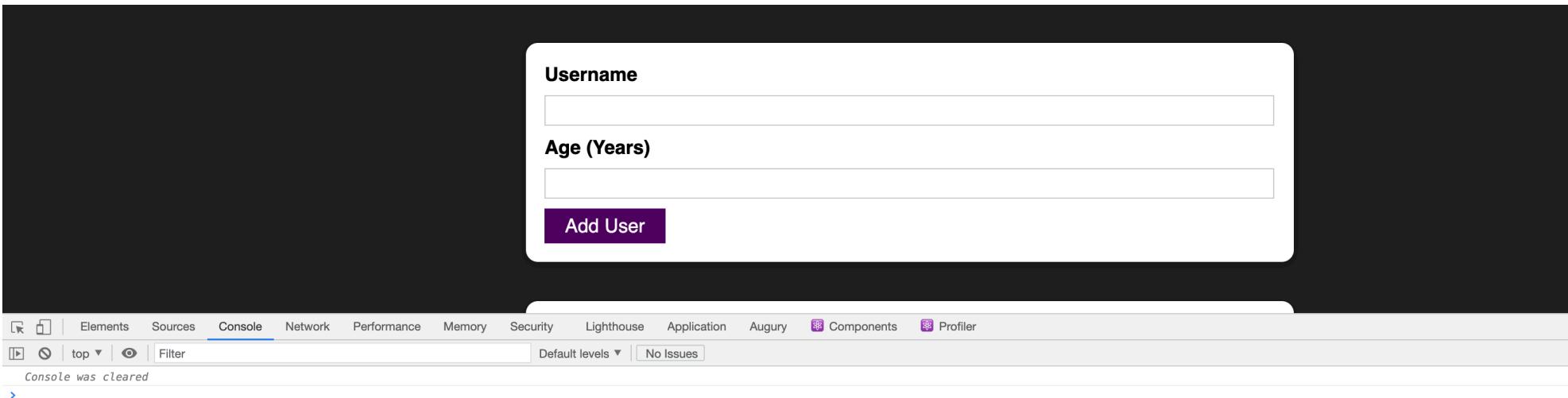
# Array of JSX elements.

- 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.



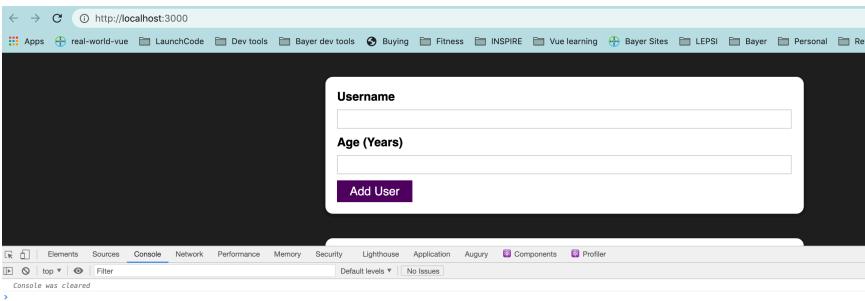
## A New Problem: "<div> Soup"

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

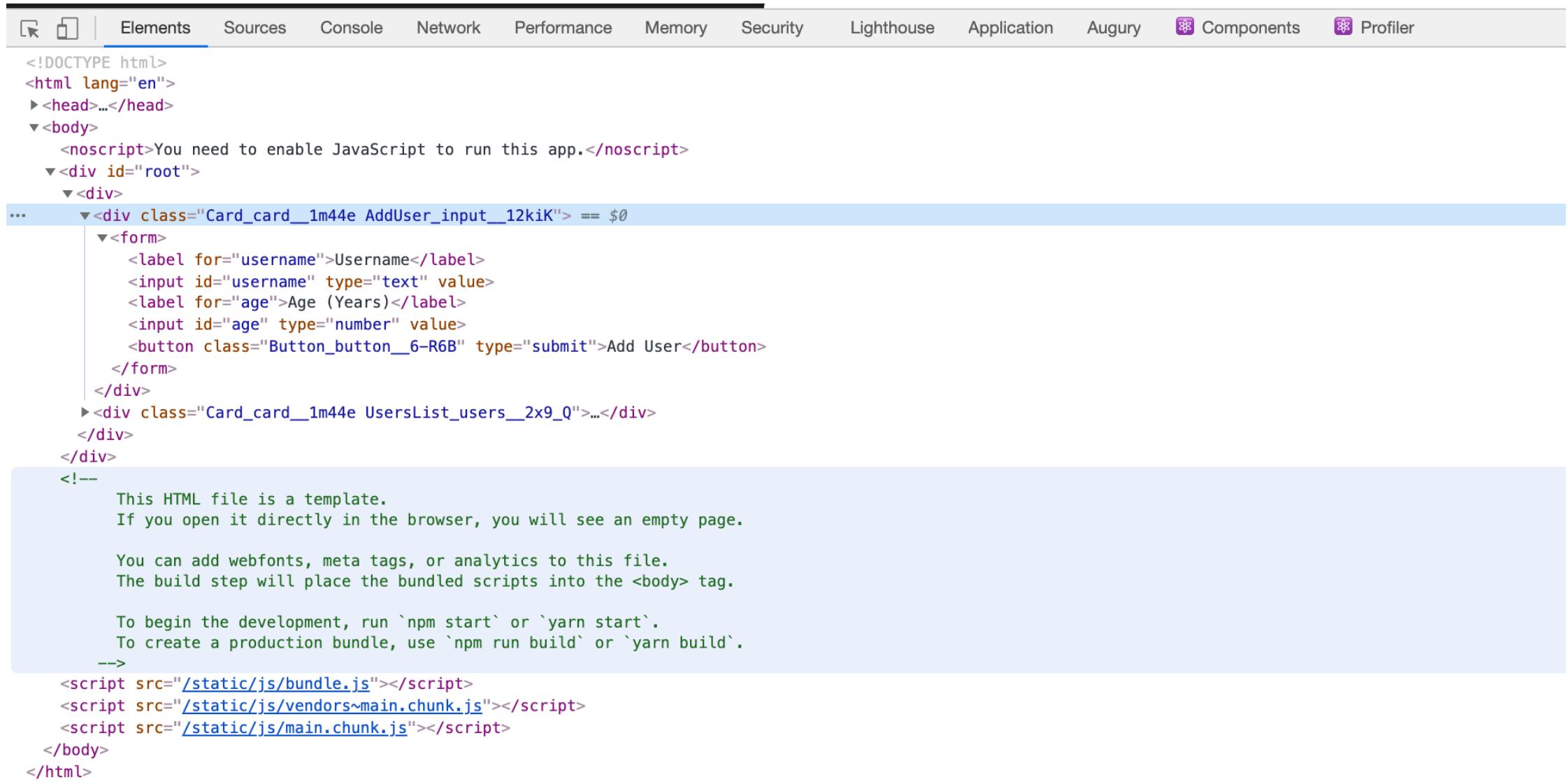
In bigger apps, you can easily end up with **tons of unnecessary <div>s** (or other elements) which add **no semantic meaning or structure** to the page but **are only there because of React's/ JSX' requirement**.

# Wrapper Component.

```
Components / Helpers / Wrapper.js / ...  
const Wrapper = (props) => {  
  return props.children;  
};  
export default Wrapper;
```



```
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>  
);
```



# React Fragments – Built in wrapper.

The diagram illustrates two equivalent ways to use a `<React.Fragment>` component in a React application. A central purple rounded rectangle contains the word "OR". Two lines extend from it to two separate code snippets, each enclosed in a white rounded rectangle.

**Left Snippet:**

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

**Right Snippet:**

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

**Note:**

We have written a note here. It's an empty wrapper component: It doesn't render any real HTML to the DOM. But it fulfills React's/ JSX' requirement.

# React Fragments – Built in wrapper.

```
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.

The image shows two side-by-side browser developer tool windows, both titled "Elements".

**Left Window:** Displays the raw HTML code structure. A specific fragment node is highlighted with a blue selection bar. The code includes a form for adding a user, a users list, and some explanatory text at the bottom about webfonts, meta tags, and analytics.

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root">
      <div>
        ...<div class="Card_card__1m44e AddUser_input__12kiK" style="border: 1px solid #ccc; padding: 10px; margin-bottom: 10px;">
          <form>
            <label for="username">Username</label>
            <input id="username" type="text" value="John Doe" />
            <label for="age">Age (Years)</label>
            <input id="age" type="number" value="30" />
            <button class="Button_button__6-RGB" type="submit">Add User</button>
          </form>
        </div>
        <div class="Card_card__1m44e UsersList_users__2x9_Q" style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">...</div>
      </div>
    </div>
  </body>
</html>
```

**Right Window:** Shows the same HTML structure, but the entire body content is highlighted with a grey selection bar. This indicates that the browser has treated the entire body as a single fragment, which is a common optimization in React.

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  ...<body> == $0
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root">
      <div class="Card_card__1m44e AddUser_input__12kiK" style="border: 1px solid #ccc; padding: 10px; margin-bottom: 10px;">
        <form>
          <label for="username">Username</label>
          <input id="username" type="text" value="John Doe" />
          <label for="age">Age (Years)</label>
          <input id="age" type="number" value="30" />
          <button class="Button_button__6-RGB" type="submit">Add User</button>
        </form>
      </div>
      <div class="Card_card__1m44e UsersList_users__2x9_Q" style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">...</div>
    </div>
  </body>
</html>
```

Annotations in the right window:  
- "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`."

# React Fragments – Alternate Syntax.

```
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>
  );
}

export default App;
```

function App(): JSX.Element

# React Fragments.

```
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;
```



## Understanding React Portals

It's a bit like styling a <div> 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>
```

# Understanding React Portals

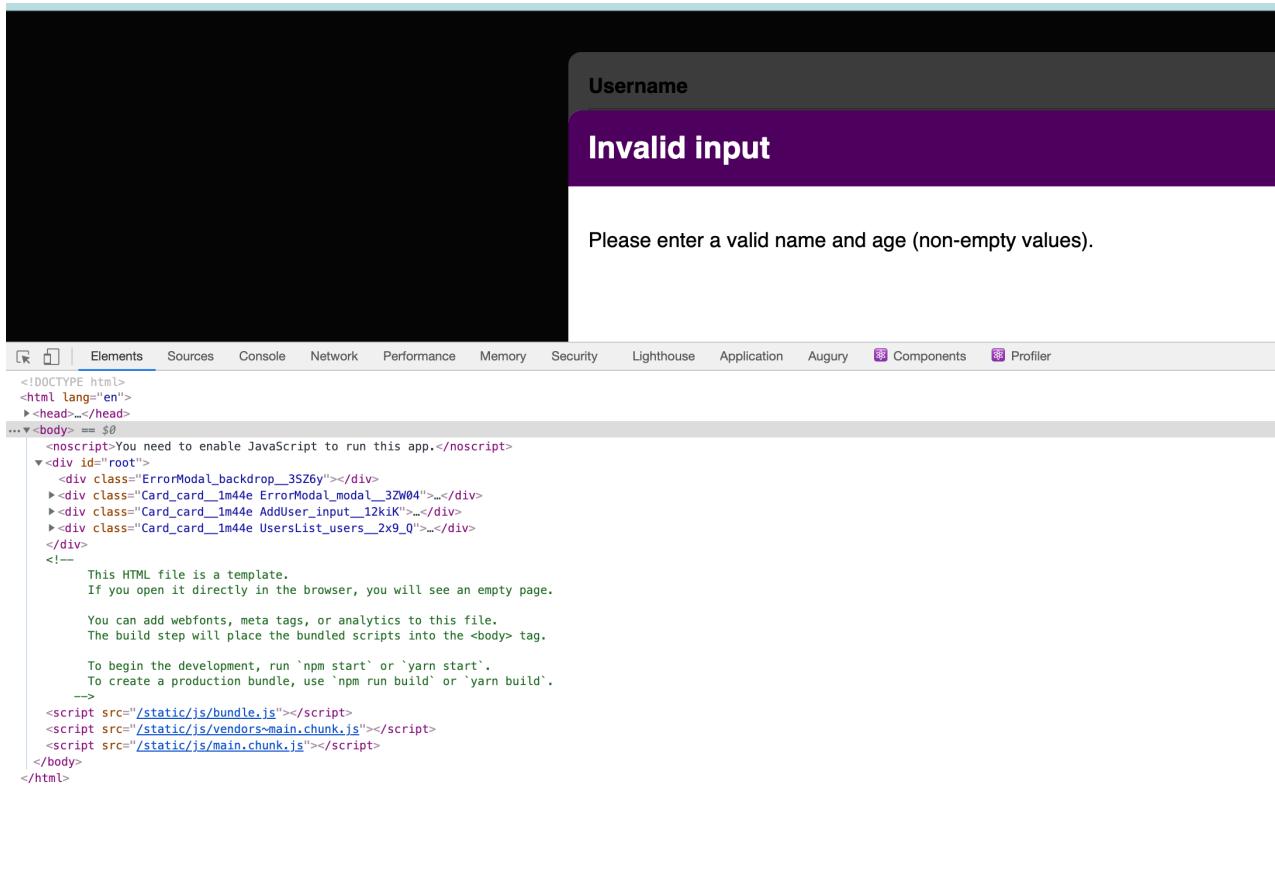
```
return (  
  <React.Fragment>  
    <MyModal />  
    <MyInputForm />  
  </React.Fragment>  
) ;
```



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>
```

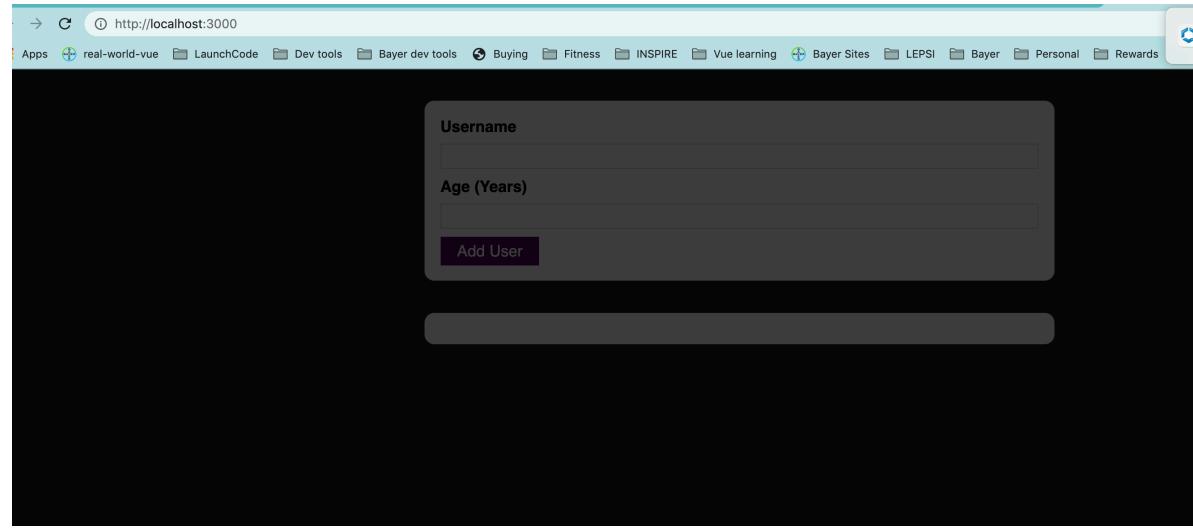
```
import Button from './Button';
import classes from './ErrorModal.module.css';

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;
```



**Username**

**Age (Years)**

Add User

ddd (12 years old)

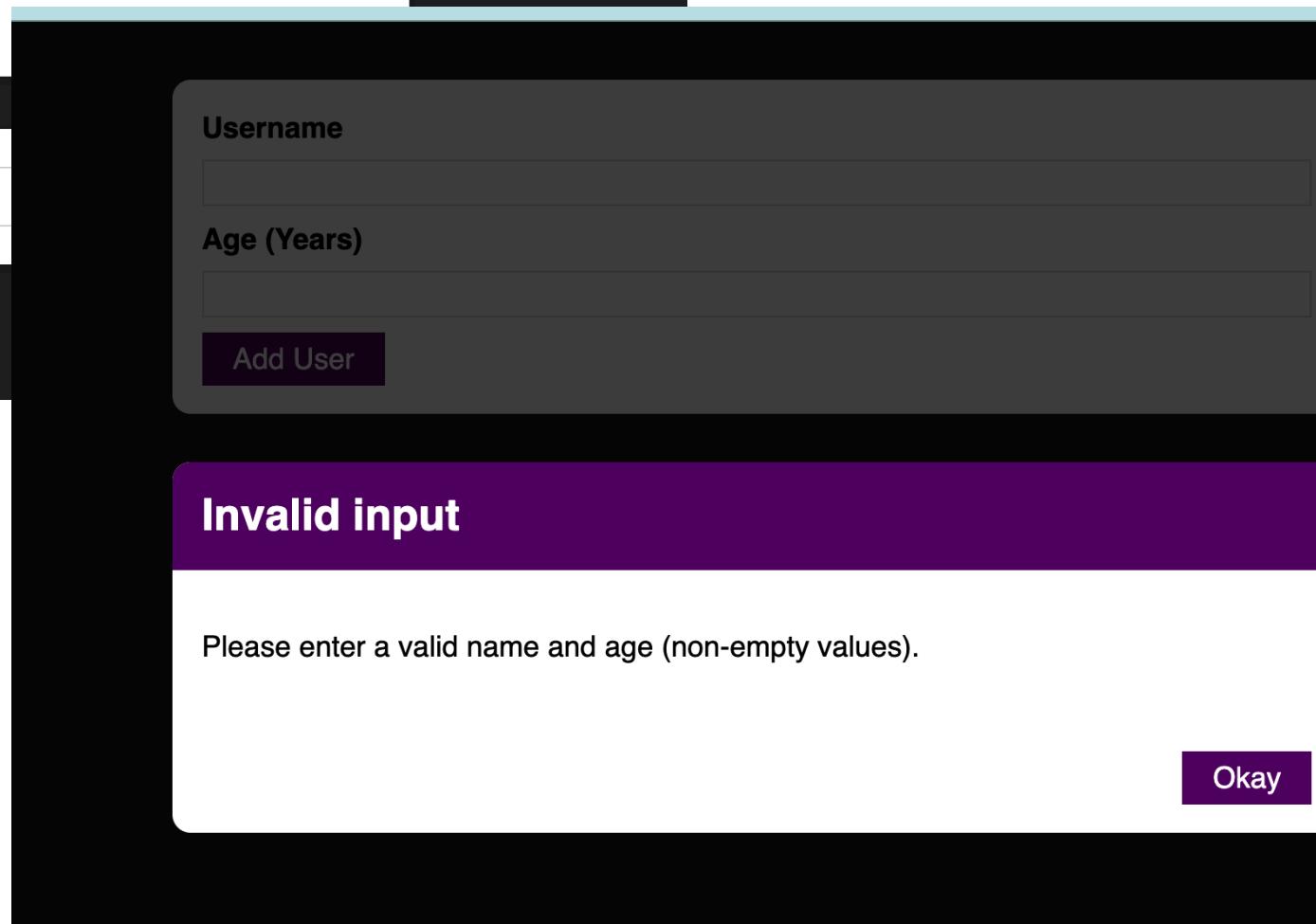
```
src / components / ErrorModal.js / ErrorModal.js
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import Card from './Card';
4 import Button from './Button';
5 import classes from './ErrorModal.module.css';
6
7 const Backdrop = (props) => {
8   return <div className={classes.backdrop} onClick={props.onConfirm} />;
9 };
10
11 const ModalOverlay = (props) => {
12   return (
13     <Card className={classes.modal}>
14       <header className={classes.header}>
15         <h2>{props.title}</h2>
16       </header>
17       <div className={classes.content}>
18         <p>{props.message}</p>
19       </div>
20       <footer className={classes.actions}>
21         <Button onClick={props.onConfirm}>Okay</Button>
22       </footer>
23     </Card>
24   );
25 };
26
27 const ErrorModal = (props) => {
28   return (
29     <>
30       {ReactDOM.createPortal(
31         <Backdrop onConfirm={props.onConfirm} />,
32         document.getElementById('backdrop-root')
33       )}
34       {ReactDOM.createPortal(
35         <ModalOverlay
36           title={props.title}
37           message={props.message}
38           onConfirm={props.onConfirm}
39         />,
40         document.getElementById('overlay-root')
41       )}
42     </>
43   );
44 };
45
46 export default ErrorModal;
47
```

**Username**

**Age (Years)**

**Add User**

ddd (12 years old)



# 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.

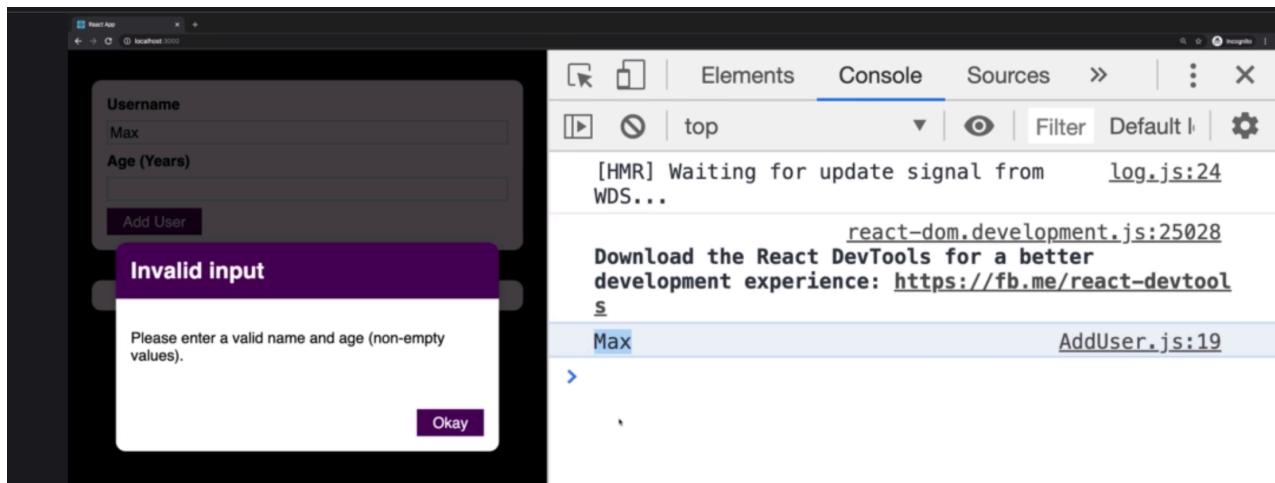
```
import React, { useState, useRef } from 'react';

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.



```

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>
  );
};

export default AddUser;

```

Username

Age (Years)

Add User

sdd (21 years old)

# 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.



## Module Content

Working with (Side) Effects

Managing more Complex State with Reducers

Managing App-Wide or Component-Wide State with Context



## 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  
...

## Handling Side Effects with the `useEffect()` Hook

```
useEffect(() => { ... }, [ dependencies ]);
```

A function that should be executed AFTER every component evaluation IF the specified dependencies changed

Dependencies of this effect – the function only runs if the dependencies changed

```
src > App.js > App
1 import React, { useState } from 'react';
2
3 import Login from './components/Login/Login';
4 import Home from './components/Home/Home';
5 import MainHeader from './components/MainHeader/MainHeader';
6
7 function App() {
8   const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');
9
10  const [isLoggedIn, setIsLoggedIn] = useState(false);
11
12  if (storedUserLoggedInInformation === '1') {
13    setIsLoggedIn(true);
14  }
15
16  const loginHandler = (email, password) => {
17    // We should of course check email and password
18    // But it's just a dummy/ demo anyways
19    localStorage.setItem('isLoggedIn', '1');
20    setIsLoggedIn(true);
21  };
22
23  const logoutHandler = () => {
24    setIsLoggedIn(false);
25  };
26
27  return (
28    <React.Fragment>
29      <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />
30      <main>
31        {!isLoggedIn && <Login onLogin={loginHandler} />}
32        {isLoggedIn && <Home onLogout={logoutHandler} />}
33      </main>
34    </React.Fragment>
35  );
36}
37
38 export default App;
39
```



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 "771544dcc6674ef3bf5a4c3b82262fd4" true
com.adobe.reactor.dataElements.user   UID	
com.adobe.reactor.dataElementCookiesMigrated	

```
1 import React, { useState, useEffect } from 'react';
2
3 import Login from './components/Login/Login';
4 import Home from './components/Home/Home';
5 import MainHeader from './components/MainHeader/MainHeader';
6
7 function App() {
8   const [isLoggedIn, setIsLoggedIn] = useState(false);
9
10  //run this code only once when the app starts up.
11  useEffect(() => {
12    const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');
13    if (storedUserLoggedInInformation === '1') {
14      setIsLoggedIn(true);
15    }
16  }, []);
17
18  const loginHandler = (email, password) => [
19    localStorage.setItem('isLoggedIn', '1');
20    setIsLoggedIn(true);
21  ];
22
23  const logoutHandler = () => {
24    setIsLoggedIn(false);
25  };
26
27  return (
28    <React.Fragment>
29      <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />
30      <main>
31        {!isLoggedIn && <Login onLogin={loginHandler} />}
32        {isLoggedIn && <Home onLogout={logoutHandler} />}
33      </main>
34    </React.Fragment>
35  );
36}
37
38 export default App;
39
```

# A Typical Page

Users Admin

Logout

## Welcome back!

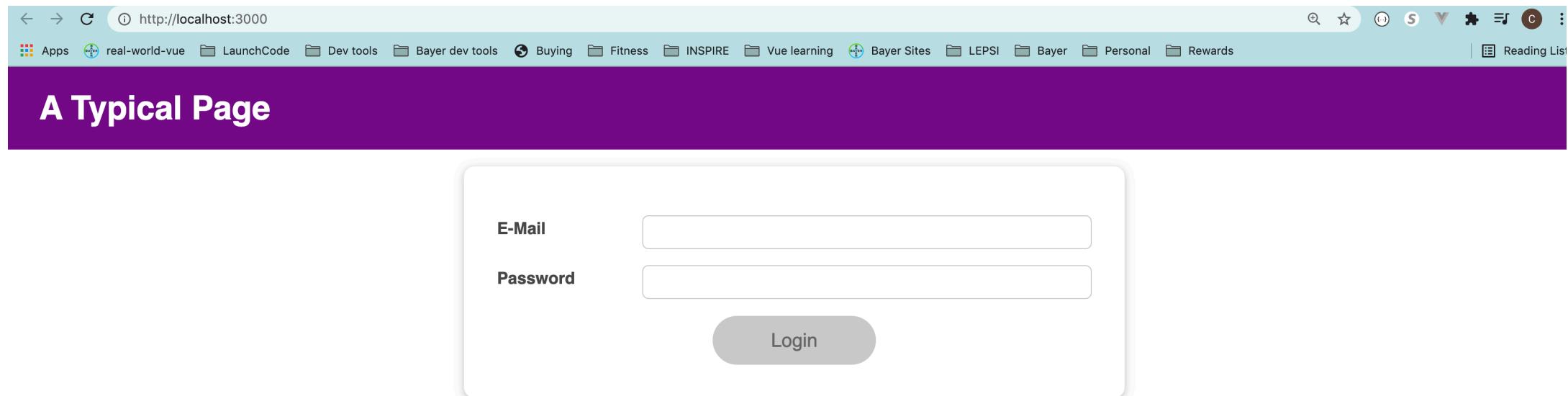
The screenshot shows the Chrome DevTools Application tab interface. On the left, there's a sidebar with sections for Application (Manifest, Service Workers, Storage), Storage (Local Storage, Session Storage, IndexedDB, Web SQL, Cookies, Trust Tokens), and Network (Console, Network, Performance, Memory, Security). The Local Storage section for the domain `http://localhost:3000` is selected and expanded. In the main pane, the Application tab is active, showing a table of storage items. One item is highlighted with a yellow background:

Key	Value
<code>isLoggedIn</code>	1
<code>com.adobe.reactor.dataElements.user   UID</code>	"771544dcc6674ef3bf5a4c3b82262fd4"
<code>com.adobe.reactor.dataElementCookiesMigrated</code>	true

## Welcome back!

The screenshot shows the Chrome DevTools interface with the Application tab selected. The left sidebar lists sections for Application (Manifest, Service Workers, Storage), Storage (Local Storage, Session Storage, IndexedDB, Web SQL, Cookies, Trust Tokens), and Cache (Cache Storage, Application Cache). The Storage section for Local Storage is expanded, showing items for the domain `http://localhost:3000`. One item, `isLoggedIn`, is highlighted and has a context menu open with options: Header Options, Refresh, Edit "Key", and Delete. The `isLoggedIn` entry has the value `1` and the raw value `"771544dcc6674ef3bf5a4c3b82262fd4"`. The Storage table has a yellow highlight under the first column.

Key	Value
<code>isLoggedIn</code>	<code>1</code> <code>"771544dcc6674ef3bf5a4c3b82262fd4"</code> true



A screenshot of the Chrome DevTools Application tab. The tab bar includes Elements, Sources, Console, Network, Performance, Memory, Security, Lighthouse, Application (which is selected), Augury, Components, and Profiler. On the left, there's a sidebar with sections for Application (Manifest, Service Workers, Storage), Storage (Local Storage, Session Storage, IndexedDB, Web SQL, Cookies, Trust Tokens), and Cache (Cache Storage, Application Cache). The main area shows a table of storage logs. Under Application, there are entries for Manifest, Service Workers, and Storage. Under Storage, under Local Storage for 'http://localhost:3000', there are two entries: 'com.adobe.reactor.dataElements.user | UID' with value "'771544dcc6674ef3bf5a4c3b82262fd4'" and 'com.adobe.reactor.dataElementCookiesMigrated' with value 'true'. Under Session Storage, there is an entry for '1' with value "'771544dcc6674ef3bf5a4c3b82262fd4'". The bottom status bar says "Line 1, Column 1".

A screenshot of a web browser window. The address bar shows the URL `http://localhost:3000`. The top navigation bar has a purple background with various links like 'real-world-vue', 'Buying', 'Fitness', etc. Below this is a dark purple header bar with the text 'A Typical Page' on the left, and 'Users', 'Admin', and a 'Logout' button on the right. A large white rounded rectangle in the center contains the text 'Welcome back!'. The overall theme is a modern web application interface.

A screenshot of the Chrome DevTools Application tab. The sidebar on the left lists 'Application', 'Storage', and 'Cache' sections. Under 'Storage', 'Local Storage' is expanded, showing an item for the URL `http://localhost:3000` with the key `com.adobe.reactor.dataElements.user | UID` and value `"771544dcc6674ef3bf5a4c3b82262fd4"`. The 'Value' column is highlighted with a yellow background. The 'Cache' section is also visible. The main panel shows a table with one row and one column, labeled 'Line 1, Column 1'.

Key	Value
<code>com.adobe.reactor.dataElements.user   UID</code>	<code>"771544dcc6674ef3bf5a4c3b82262fd4"</code>
<code>com.adobe.reactor.dataElementCookiesMigrated</code>	<code>true</code>
<code>isLoggedIn</code>	<code>1</code>

```
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);
  };

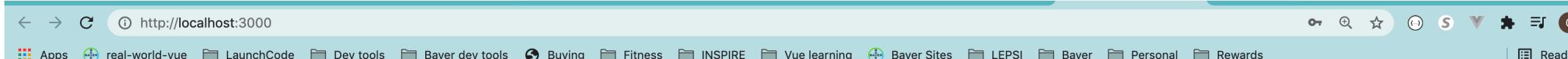
  return (
    <React.Fragment>
      <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />
      <main>
        {!isLoggedIn && <Login onLogin={loginHandler} />}
        {isLoggedIn && <Home onLogout={logoutHandler} />}
      </main>
    </React.Fragment>
  );
}

export default App;
```

# A Typical Page

Users Admin Logout

## Welcome back!

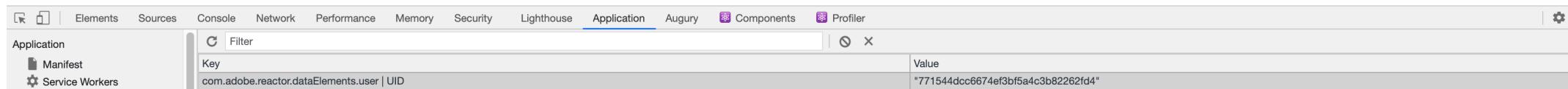


## A Typical Page

E-Mail

Password

**Login**



# useEffect for Login.

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

c > components > Login > Login.js > [x] Login > [x] emailChangeHandler

```
1 import React, { useState, useEffect } from 'react';
2
3 import Card from '../UI/Card/Card';
4 import classes from './Login.module.css';
5 import Button from '../UI/Button/Button';
6
7 const Login = (props) => {
8   const [enteredEmail, setEnteredEmail] = useState('');
9   const [emailIsValid, setEmailIsValid] = useState();
10  const [enteredPassword, setEnteredPassword] = useState('');
11  const [passwordIsValid, setPasswordIsValid] = useState();
12  const [formIsValid, setFormIsValid] = useState(false);
13
14  useEffect(() => {
15    setEmailIsValid(
16      enteredEmail.includes('@') && enteredPassword.trim().length > 6
17    );
18  }, [enteredEmail, enteredPassword]);
19
20  const emailChangeHandler = (event) => {
21    setEnteredEmail(event.target.value);
22  };
23
24  const passwordChangeHandler = (event) => {
25    setEnteredPassword(event.target.value);
26  };
27
28  const validateEmailHandler = () => {
29    setEmailIsValid(enteredEmail.includes('@'));
30  };
31
```

# A Typical Page

E-Mail

Password

**Login**

E-Mail

Password

**Login**

# 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.

```
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, setPassword] = 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) => {
    setPassword(event.target.value);
  };

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

← → ⌂ ⓘ http://localhost:3000

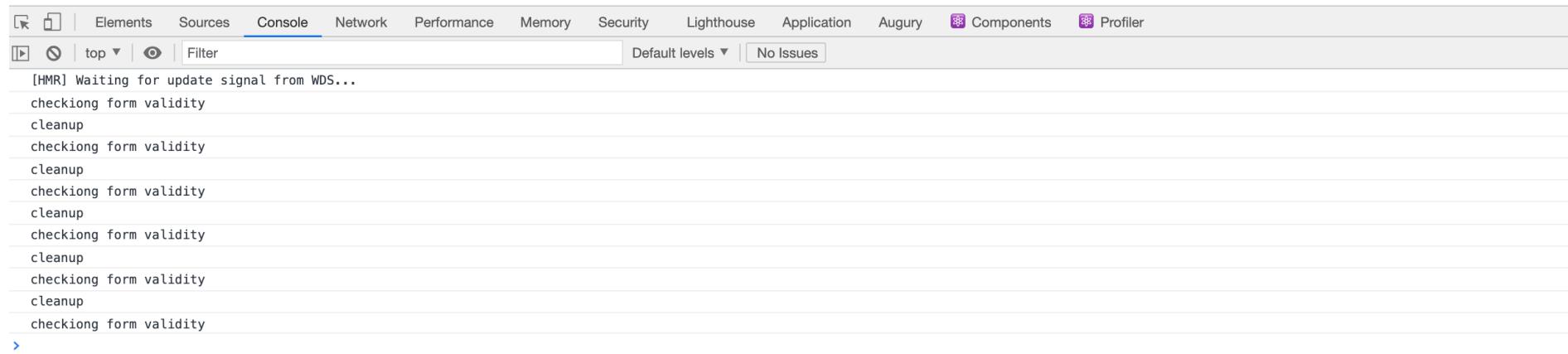
\_apps real-world-vue LaunchCode Dev tools Bayer dev tools Buying Fitness INSPIRE Vue learning Bayer Sites LEPSI Bayer Personal

# A Typical Page

E-Mail ss

Password ...

Login



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

This runs for every component event such as 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. Keystrokes.

## 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()

```
const [state, dispatchFn] = useReducer(reducerFn, initialState, initFn);
```

The state snapshot used in the component render/ re-evaluation cycle

A function that can be used to dispatch a new action (i.e. trigger an update of the state)

The initial state

A function to set the initial state programmatically

```
(prevState, action) => newState
```

A function that is triggered automatically once an action is dispatched (via `dispatchFn()`) – it receives the latest state snapshot and should return the new, updated state.

# Reducer function.

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

```
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;
```

The image shows a user interface for a login form. It consists of two input fields: one for 'E-Mail' containing the value 'abc@abc.com' and one for 'Password' which is currently empty. Below the inputs is a large, rounded rectangular button labeled 'Login'.

```

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, setPasswordEnteredPassword] = 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);
  };

```

E-Mail

Password

Login

```

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

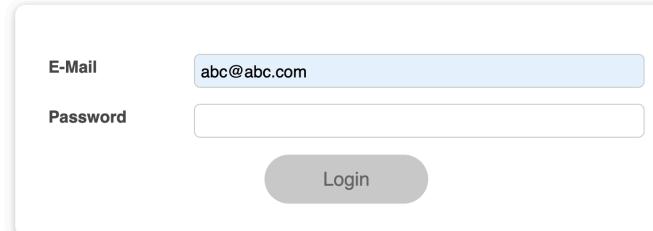
const validateEmailHandler = () => {
  // setEmailIsValid(emailState.isValid);
  dispatchEmail({ type: 'INPUT_BLUR' });
};

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

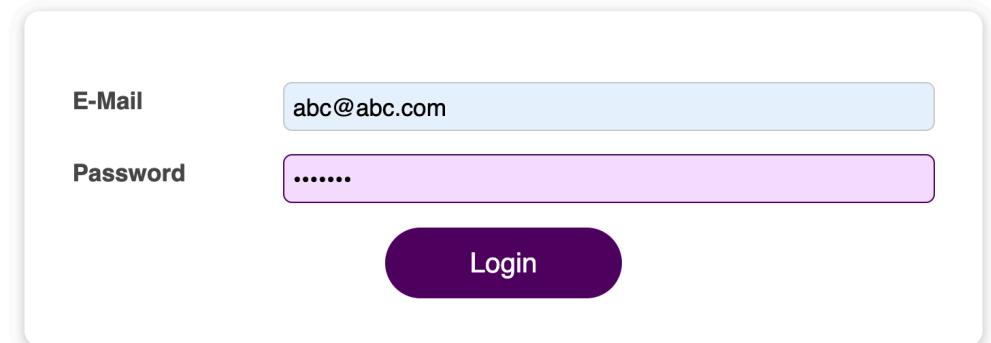
const submitHandler = (event) => {
  event.preventDefault();
  props.onLogin(emailState.value, pwdState.value);
};

return (
  <Card className={classes.login}>
    <form onSubmit={submitHandler}>
      <div>
        <label htmlFor="email">E-Mail</label>
        <input
          type="email"
          id="email"
          value={emailState.value}
          onChange={emailChangeHandler}
          onBlur={validateEmailHandler}
        />
      </div>
      <div>
        <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>
);

```



A screenshot of a login form. It has two input fields: 'E-Mail' containing 'abc@abc.com' and 'Password' which is empty. Below the inputs is a grey 'Login' button.



A screenshot of a login form. The 'E-Mail' field contains 'abc@abc.com'. The 'Password' field is filled with five dots ('.....'). Below the inputs is a dark purple 'Login' button.

# Studio