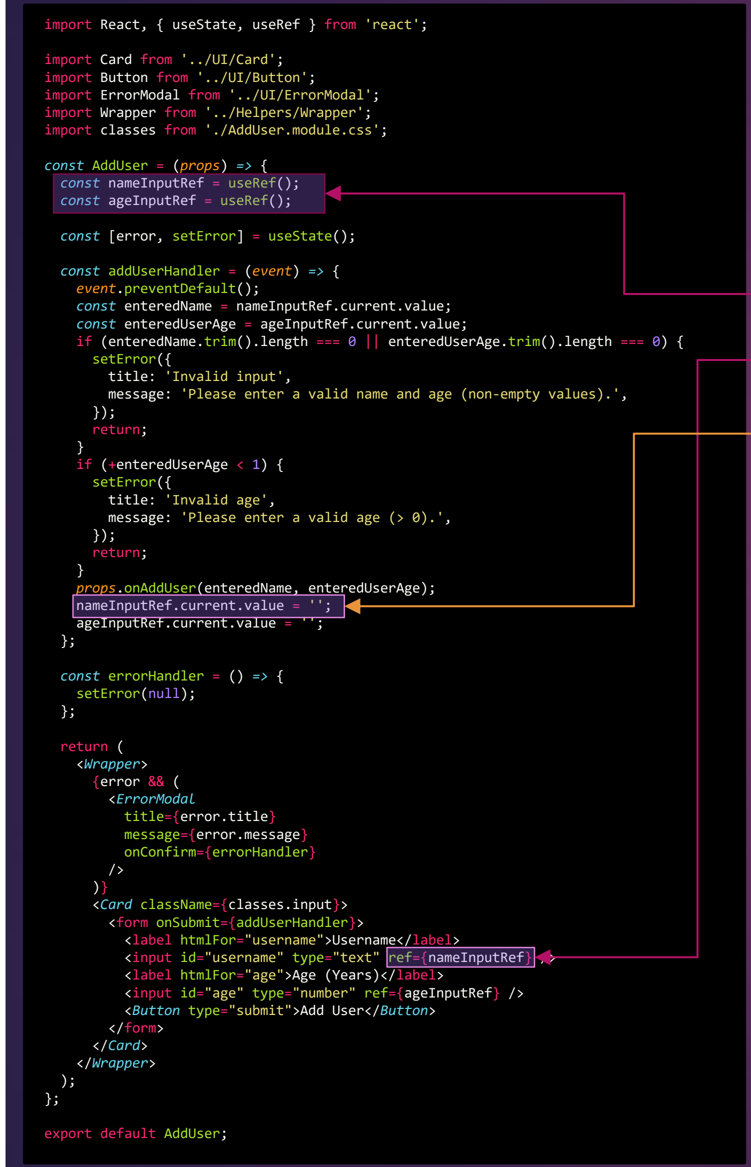
React&react-Hooks

- useRef
- React Portals
- useEffect
- useReducer
- React.forwordRef
- useContext

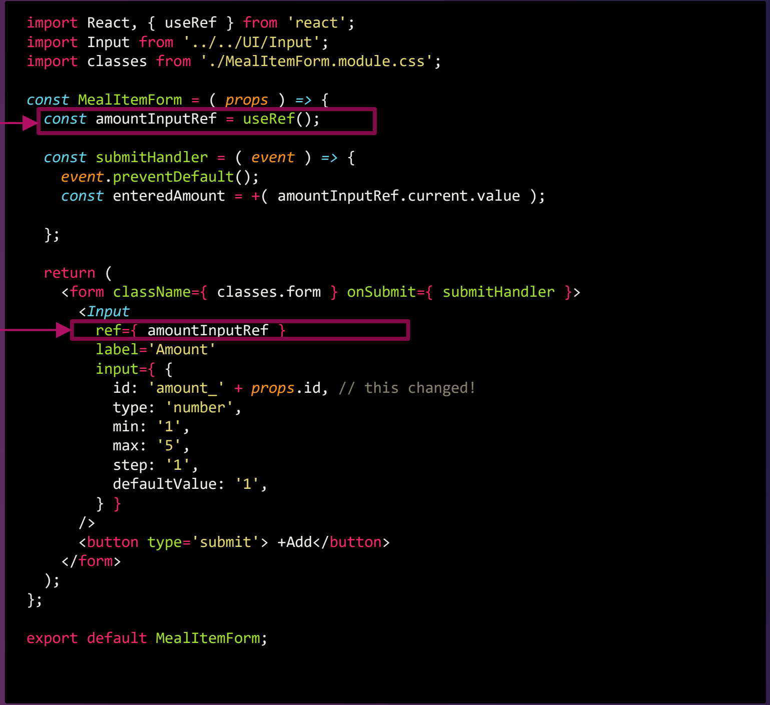
useImperativeHandle
UI input

useRef

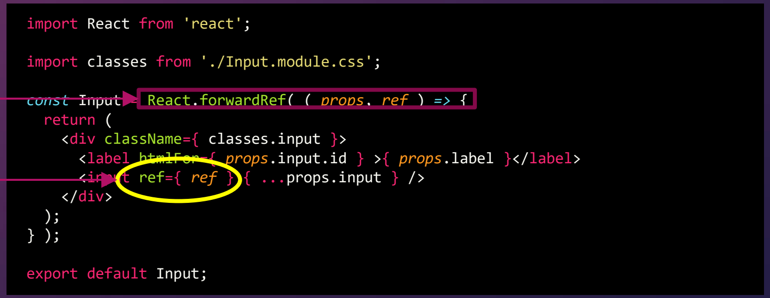


```
import React, { useState, useRef } from 'react';

import Card from '../UI/Card';
import Button from '../UI/Button';
import ErrorModal from '../UI/ErrorModal';
import Wrapper from '../Helpers/Wrapper';
import classes from './AddUser.module.css';

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;
```

- init useRef
- connect useRef with input
- Reset input
- Wrapper input component by React.forWwordRef
- reAdd ref={ref} to attribute component



```
import React, { useRef } from 'react';
import Input from '../../UI/Input';
import classes from './MealItemForm.module.css';

const MealItemForm = ( props ) => {
  const amountInputRef = useRef();

  const submitHandler = ( event ) => {
    event.preventDefault();
    const enteredAmount = +( amountInputRef.current.value );

  };

  return (
    <form className={ classes.form } onSubmit={ submitHandler }>
      <Input
        ref={ amountInputRef }
        label='Amount'
        input={ {
          id: 'amount_' + props.id, // this changed!
          type: 'number',
          min: '1',
          max: '5',
          step: '1',
          defaultValue: '1',
        } }
      />
      <button type='submit'> +Add</button>
    </form>
  );
};

export default MealItemForm;
```



```
import React from 'react';

import classes from './Input.module.css';

const Input = React.forwardRef( ( props, ref ) => {
  return (
    <div className={ classes.input }>
      <label htmlFor={ props.input.id } >{ props.label }</label>
      <input ref={ ref }{ ...props.input } />
    </div>
  );
} );

export default Input;
```

React Portals

- Create element overlay in index html
- Create Modal Component
- Crate const Backdrop
- Crate const ModalOverlay
- import ReactDOM from 'react-dom'
- Create const portalElement for access element overlay in index html
- Wrapper cart or content by Modal

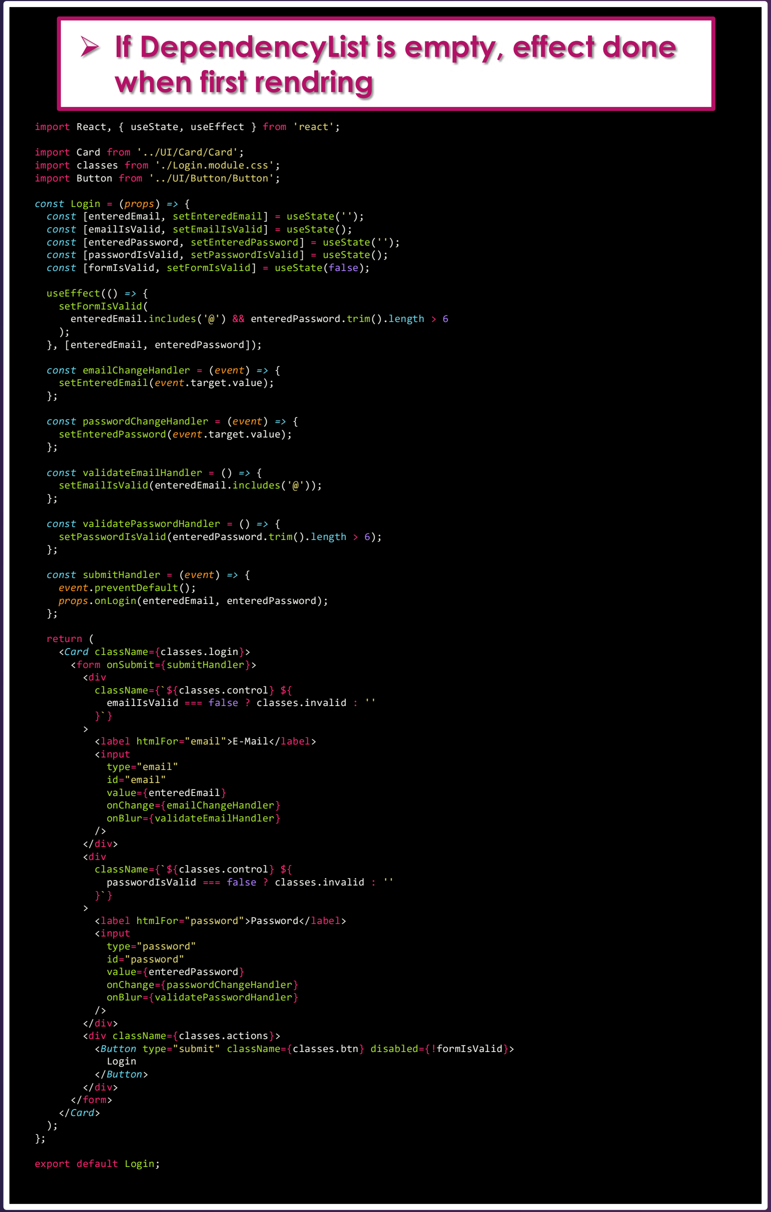
```html
<body>
  <div id="overlay"></div>
  <div id="root"></div>
</body>
```

```jsx
import React, { Fragment } from 'react';
import classes from './Modal.module.css';
import ReactDOM from 'react-dom';

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

const ModalOverlay = ( props ) => {
  return <div className={ classes.modal }>
    <div className={ classes.content }>
      { props.children }
    </div>
  </div>;
};

const portalElement = document.getElementById( 'overlays' );

const Modal = ( props ) => {
  return (
    <Fragment>
      { ReactDOM.createPortal( <Backdrop />, portalElement ) }
      { ReactDOM.createPortal( <ModalOverlay>
        { props.children }
      </ModalOverlay>, portalElement ) }
    </Fragment>
  );
};

export default Modal;
```

Add Backdrop and ModalOverlay to Modal inside Fragment

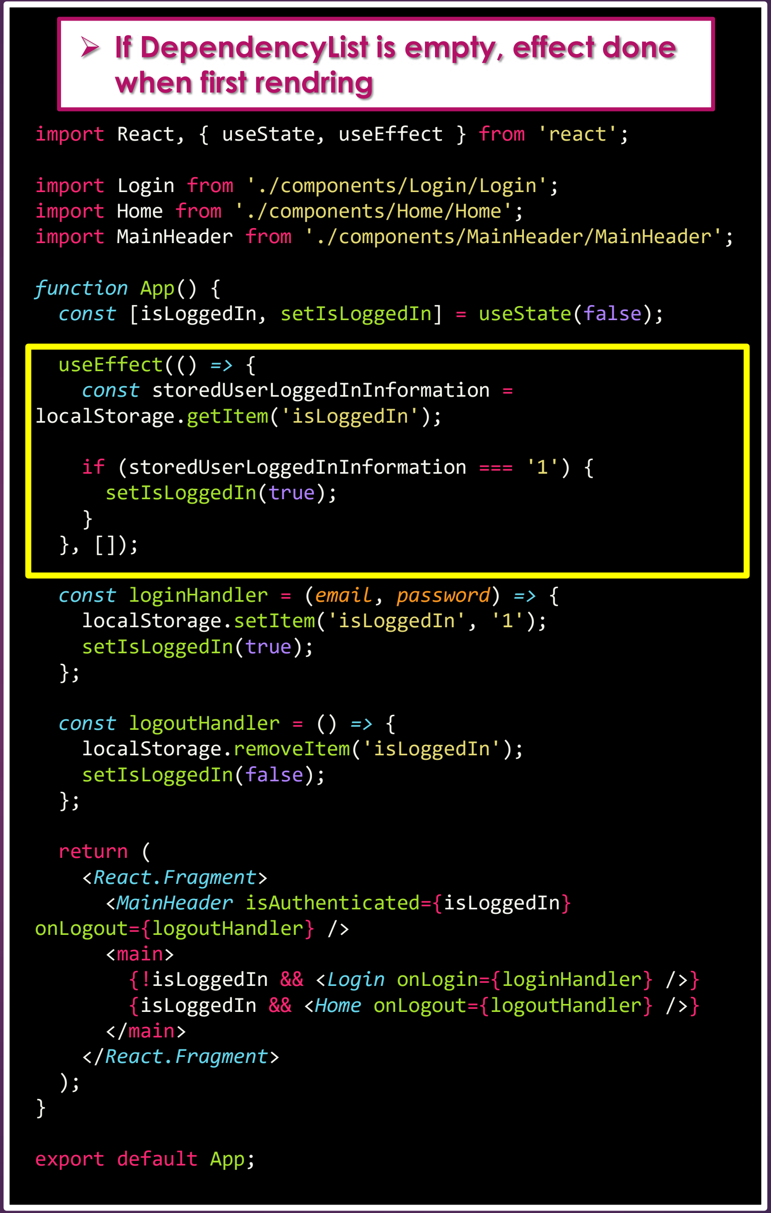
```jsx
import Modal from './Modal';

const Cart = () => {


  const cartItems = (
    <ul className={ classes[ 'cart-items' ] }>
      { [ { id: 'c1', name: 'Sushi', amount: 2, price: 12.99 } ]
        .map( item => (
          <li>{ item.name }</li>
        ) )
      }
    </ul>
  );

  return (
    <Modal>
      { cartItems }
      <div className={ classes.total }>
        <span>Total Amount</span>
        <span>35.62</span>
      </div>
    </Modal>
  );
};

export default Cart;
```

useEffect -1



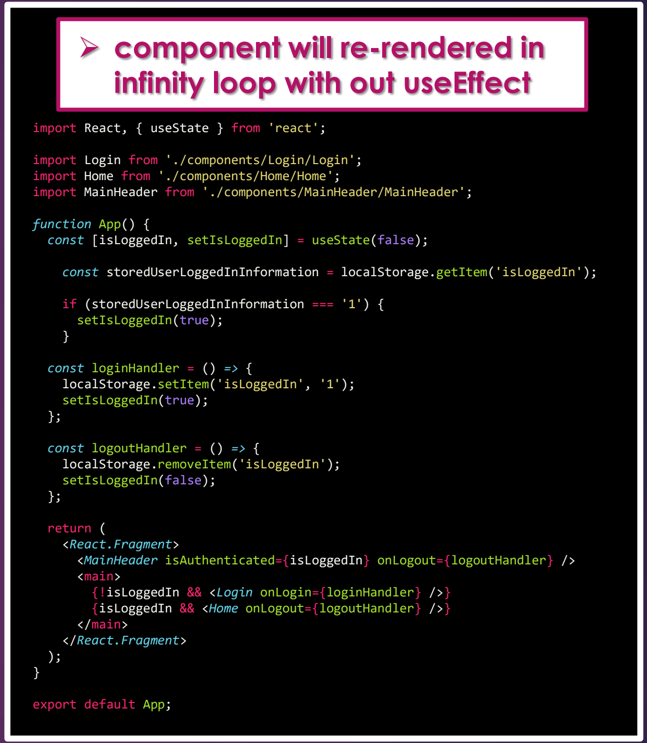
- If DependencyList is empty, effect done when first rendring

```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(() => {
    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('@'));
  };

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

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

  return (
    <Card className={classes.login}>
      <form onSubmit={submitHandler}>
        <div
          className={`${classes.control} ${
            emailIsValid === false ? classes.invalid : ''
          }`}
        >
          <label htmlFor="email">E-Mail</label>
          <input
            type="email"
            id="email"
            value={enteredEmail}
            onChange={emailChangeHandler}
            onBlur={validateEmailHandler}
          />
        </div>
        <div
          className={`${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;
```



- If DependencyList is empty, effect done when first rendring

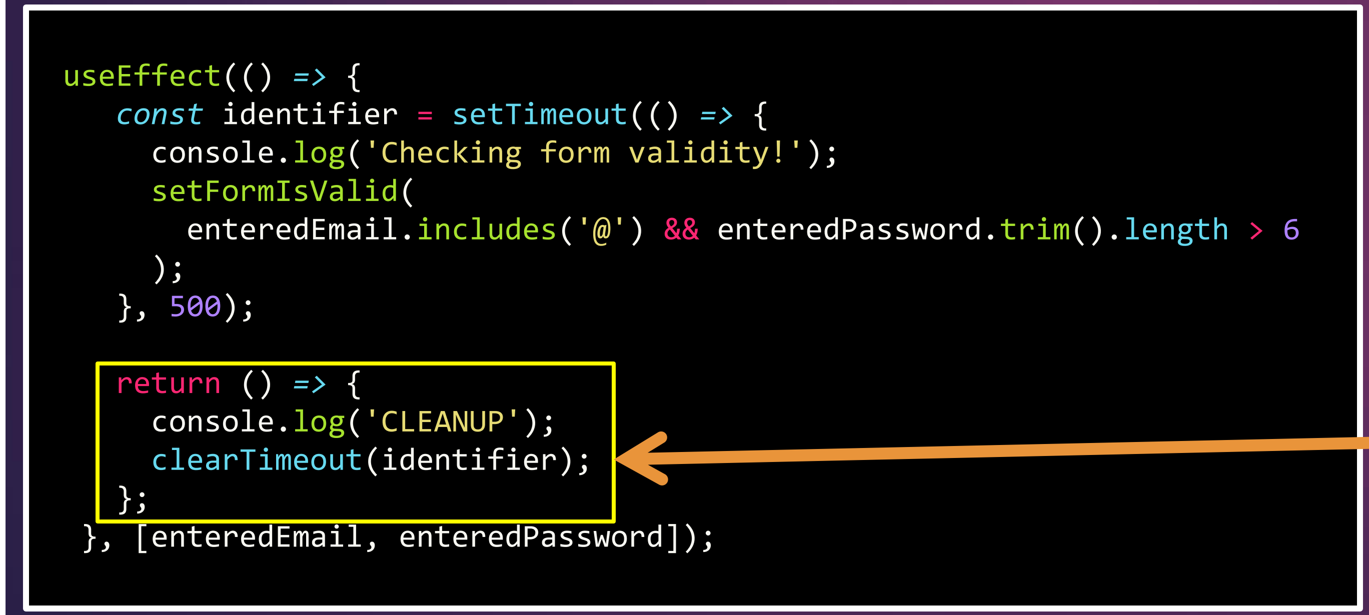
```jsx
import React, { useState, useEffect } from 'react';

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);

  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;
```



- component will re-rendered in infinity loop with out useEffect

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

    const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');

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

  const loginHandler = () => {
    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;
```

useEffect -2



```
useEffect(() => {
   const identifier = setTimeout(() => {
     console.log('Checking form validity!');
     setFormIsValid(
       enteredEmail.includes('@') && enteredPassword.trim().length > 6
     );
   }, 500);

   return () => {
     console.log('CLEANUP');
     clearTimeout(identifier);
   };
 }, [enteredEmail, enteredPassword]);
```

First when render dom will work the function useeffect, and not work the return .
When change Dependency values the retun will run cleanup function.
This will run as a cleanup process before useEffect executes this function the next time.

useReducer -1

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

The state snapshot used in the component rerender/ reevaluation 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.

useReducer -2

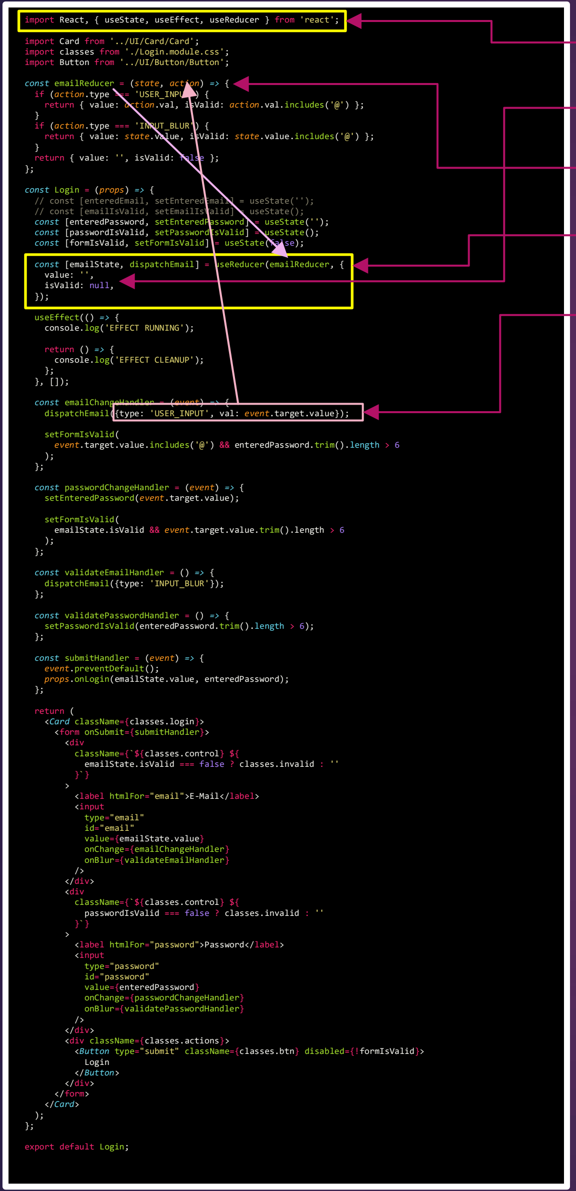
1 - import useReducer

2 - initialState

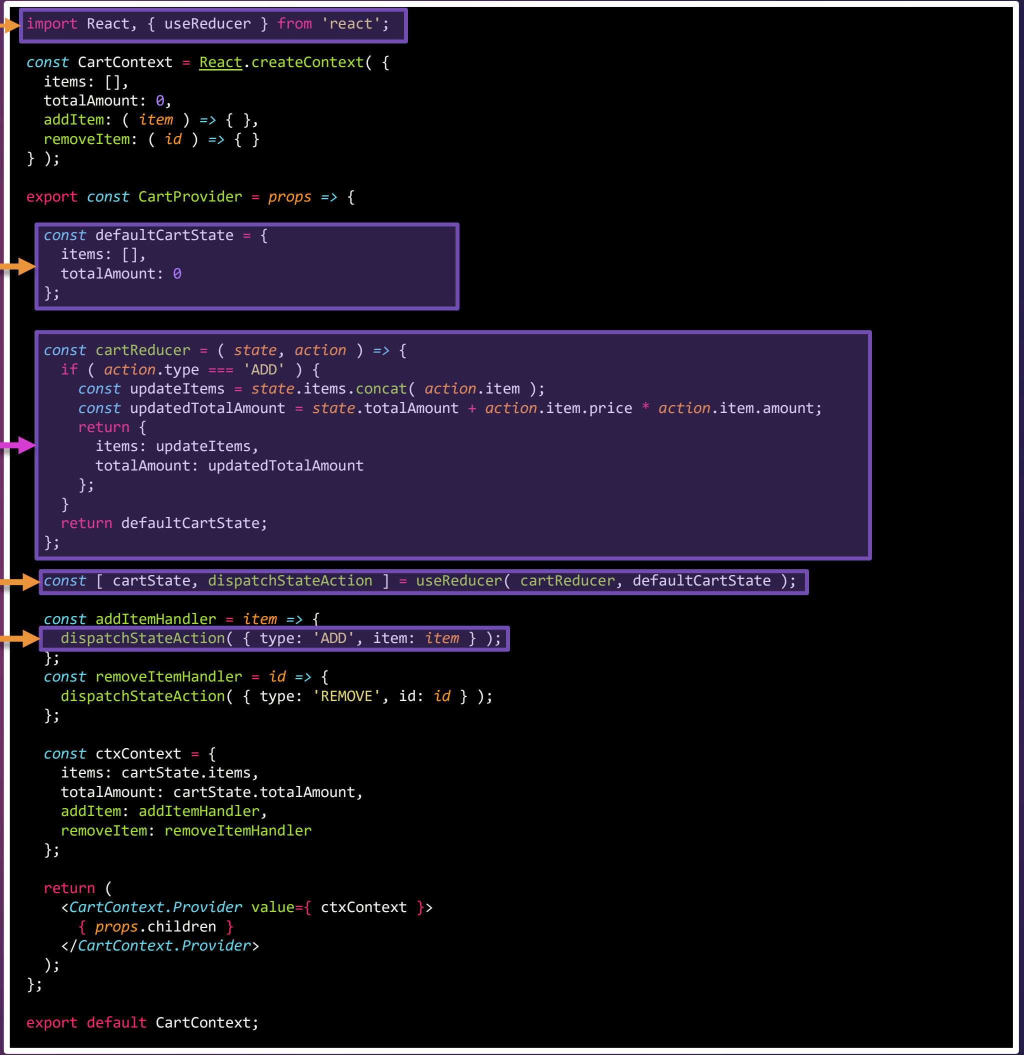
3 -Reducer Function (prevState, action) => newState

4 – use useReducer

5 – dispatch new action with object type action and new value



```
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: null,
  });

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

    return () => {
      console.log('EFFECT CLEANUP');
    };
  }, []);

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

    setFormIsValid(
      event.target.value.includes('@') && enteredPassword.trim().length > 6
    );
  };

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

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

  const validateEmailHandler = () => {
    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;
```



```
import React, { useReducer } from 'react';

const CartContext = React.createContext( {
  items: [],
  totalAmount: 0,
  addItem: ( item ) => { },
  removeItem: ( id ) => { }
} );

export const CartProvider = props => {

  const defaultCartState = {
    items: [],
    totalAmount: 0
  };

  const cartReducer = ( state, action ) => {
    if ( action.type === 'ADD' ) {
      const updateItems = state.items.concat( action.item );
      const updatedTotalAmount = state.totalAmount + action.item.price * action.item.amount;
      return {
        items: updateItems,
        totalAmount: updatedTotalAmount
      };
    }
    return defaultCartState;
  };

  const [ cartState, dispatchStateAction ] = useReducer( cartReducer, defaultCartState );

  const addItemHandler = item => {
    dispatchStateAction( { type: 'ADD', item: item } );
  };
  const removeItemHandler = id => {
    dispatchStateAction( { type: 'REMOVE', id: id } );
  };

  const ctxContext = {
    items: cartState.items,
    totalAmount: cartState.totalAmount,
    addItem: addItemHandler,
    removeItem: removeItemHandler
  };

  return (
    <CartContext.Provider value={ ctxContext }>
      { props.children }
    </CartContext.Provider>
  );
};

export default CartContext;
```

useContext -1

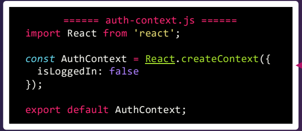
1 – create folder store > sonthing-context.js

2 – in parent componets in App.js

3 – get values context to target component

Import React

Init context by React.creatContext(initState)

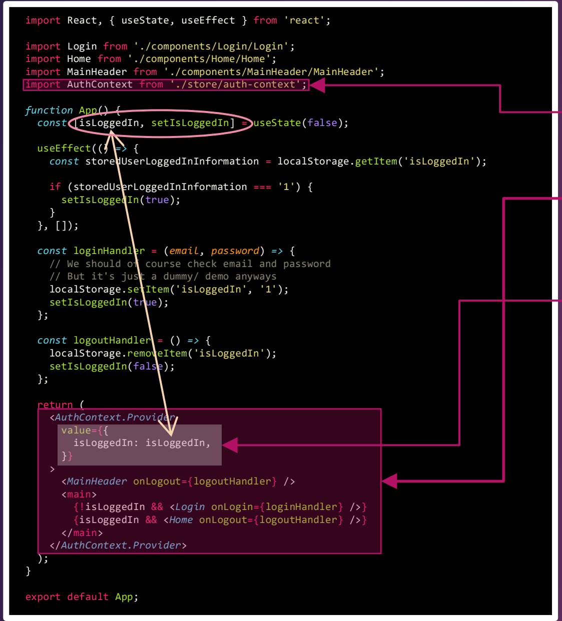


```
      ====== auth-context.js ======
import React from 'react';

const AuthContext = React.createContext({
  isLoggedIn: false
});

export default AuthContext;
```

1 Import SomethingContext from something-context.js in to root children's target

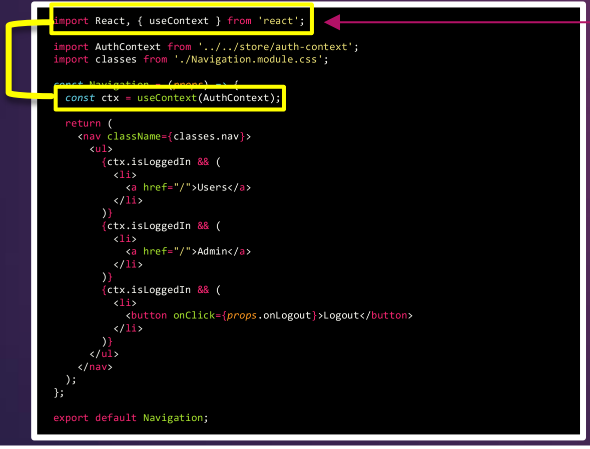
2 Wrapper and all target components by SomeThingContext.Provider

3 Add attribute 'value'={{propInitState:dynamicValue}}

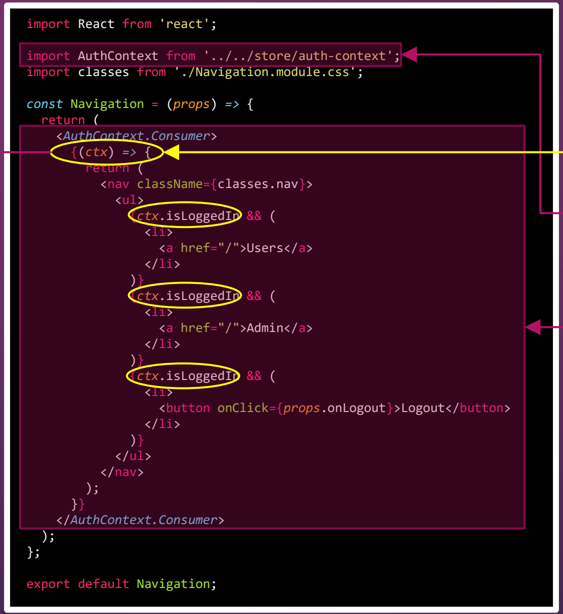


```
import React, { useState, useEffect } from 'react';

import Login from './components/Login/Login';
import Home from './components/Home/Home';
import MainHeader from './components/MainHeader/MainHeader';
import AuthContext from './store/auth-context';

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

  useEffect(() => {
    const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');

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

  return (
    <AuthContext.Provider
      value={{
        isLoggedIn: isLoggedIn,
      }}
    >
      <MainHeader onLogout={logoutHandler} />
      <main>
        {!isLoggedIn && <Login onLogin={loginHandler} />}
        {isLoggedIn && <Home onLogout={logoutHandler} />}
      </main>
    </AuthContext.Provider>
  );
}

export default App;
```

ctx useContext



```
import React, { useContext } from 'react';

import AuthContext from '../../store/auth-context';
import classes from './Navigation.module.css';

const Navigation = (props) => {
  const ctx = useContext(AuthContext);

  return (
    <nav className={classes.nav}>
      <ul>
        {ctx.isLoggedIn && (
          <li>
            <a href="/">Users</a>
          </li>
        )}
        {ctx.isLoggedIn && (
          <li>
            <a href="/">Admin</a>
          </li>
        )}
        {ctx.isLoggedIn && (
          <li>
            <button onClick={props.onLogout}>Logout</button>
          </li>
        )}
      </ul>
    </nav>
  );
};

export default Navigation;
```



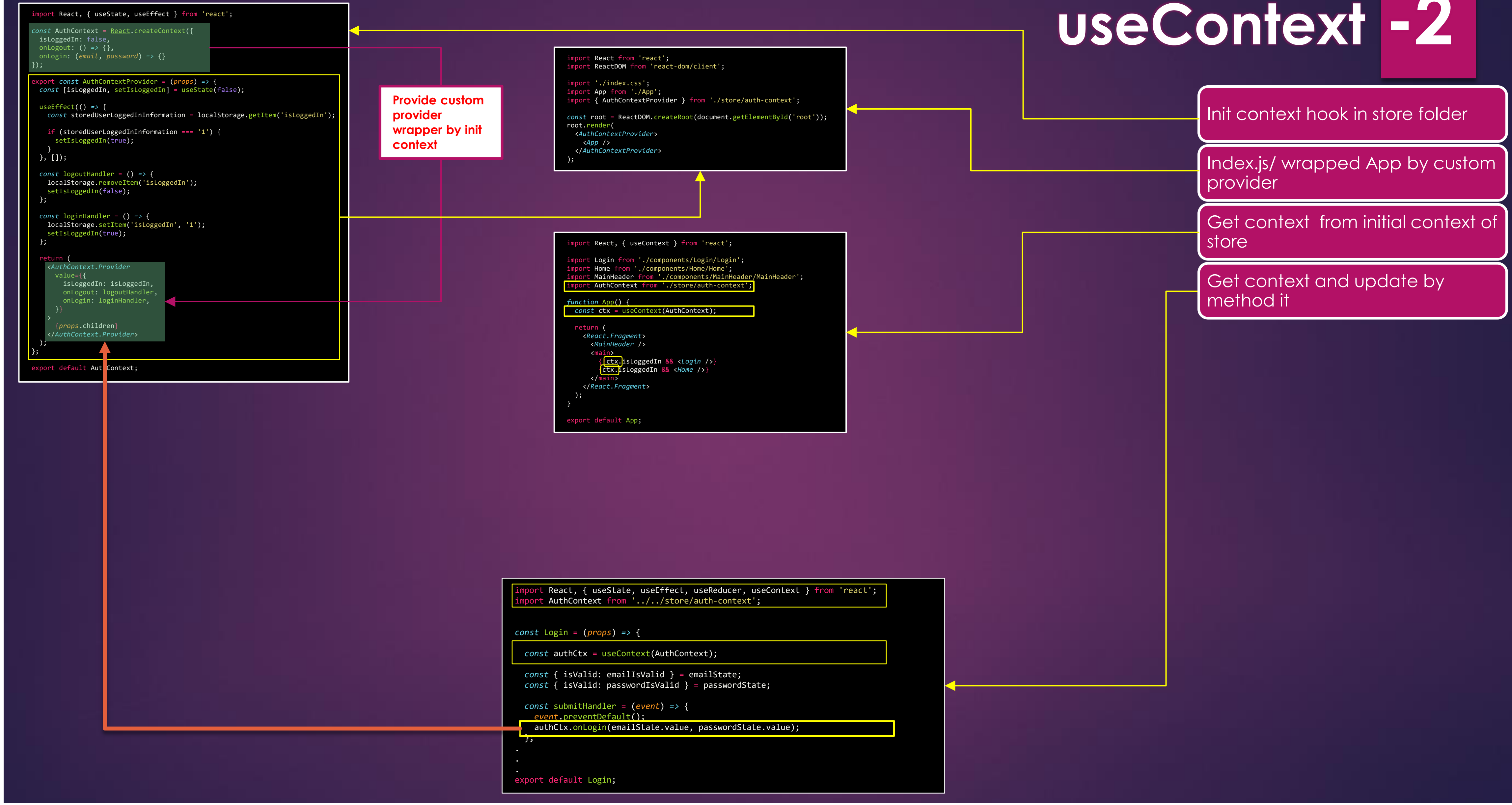
```
import React from 'react';

import AuthContext from '../../store/auth-context';
import classes from './Navigation.module.css';

const Navigation = (props) => {
  return (
    <AuthContext.Consumer>
      {(ctx) => {
        return (
          <nav className={classes.nav}>
            <ul>
              {ctx.isLoggedIn && (
                <li>
                  <a href="/">Users</a>
                </li>
              )}
              {ctx.isLoggedIn && (
                <li>
                  <a href="/">Admin</a>
                </li>
              )}
              {ctx.isLoggedIn && (
                <li>
                  <button onClick={props.onLogout}>Logout</button>
                </li>
              )}
            </ul>
          </nav>
        );
      }}
    </AuthContext.Consumer>
  );
};

export default Navigation;
```

1 Import SomethingContext from something-context.js in component target

2 Wrapper all element by SomeThingContext.Consumer

3 Get values from provider and set in ctx on consumer

useContext -2



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

const AuthContext = React.createContext({
  isLoggedIn: false,
  onLogout: () => {},
  onLogin: (email, password) => {}
});

export const AuthContextProvider = (props) => {
  const [isLoggedIn, setIsLoggedIn] = useState(false);

  useEffect(() => {
    const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');

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

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

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

  return (
    <AuthContext.Provider
      value={{
        isLoggedIn: isLoggedIn,
        onLogout: logoutHandler,
        onLogin: loginHandler,
      }}
    >
      {props.children}
    </AuthContext.Provider>
  );
};

export default AuthContext;
```

Provide custom provider wrapper by init context

```
import React from 'react';
import ReactDOM from 'react-dom/client';

import './index.css';
import App from './App';
import { AuthContextProvider } from './store/auth-context';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <AuthContextProvider>
    <App />
  </AuthContextProvider>
);
```

```
import React, { useContext } from 'react';

import Login from './components/Login/Login';
import Home from './components/Home/Home';
import MainHeader from './components/MainHeader/MainHeader';
import AuthContext from './store/auth-context';

function App() {
  const ctx = useContext(AuthContext);

  return (
    <React.Fragment>
      <MainHeader />
      <main>
        { ctx.isLoggedIn && <Login />}
        {ctx.isLoggedIn && <Home />}
      </main>
    </React.Fragment>
  );
}

export default App;
```

```
import React, { useState, useEffect, useReducer, useContext } from 'react';
import AuthContext from '../../store/auth-context';


const Login = (props) => {

  const authCtx = useContext(AuthContext);

  const { isValid: emailIsValid } = emailState;
  const { isValid: passwordIsValid } = passwordState;

  const submitHandler = (event) => {
    event.preventDefault();
    authCtx.onLogin(emailState.value, passwordState.value);
  };
.
.
.
export default Login;
```

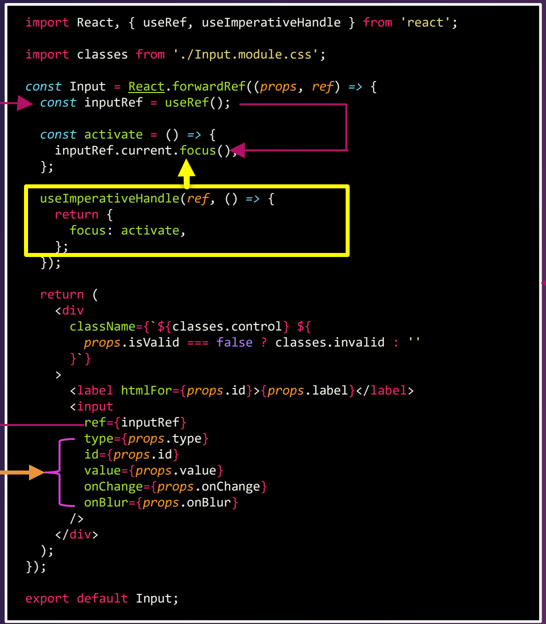
Init context hook in store folder

Index.js/ wrapped App by custom provider

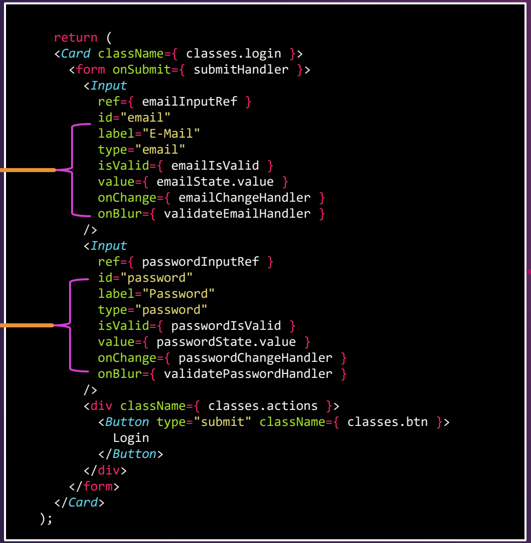
Get context from initial context of store

Get context and update by method it

useImperativeHandle UI input



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

import classes from './Input.module.css';

const Input = React.forwardRef((props, ref) => {
  const inputRef = useRef();

  const activate = () => {
    inputRef.current.focus();
  };

  useImperativeHandle(ref, () => {
    return {
      focus: activate,
    };
  });

  return (
    <div
      className={`${classes.control} ${
        props.isValid === false ? classes.invalid : ''
      }`}
    >
      <label htmlFor={props.id}>{props.label}</label>
      <input
        ref={inputRef}
        type={props.type}
        id={props.id}
        value={props.value}
        onChange={props.onChange}
        onBlur={props.onBlur}
      />
    </div>
  );
});

export default Input;
```

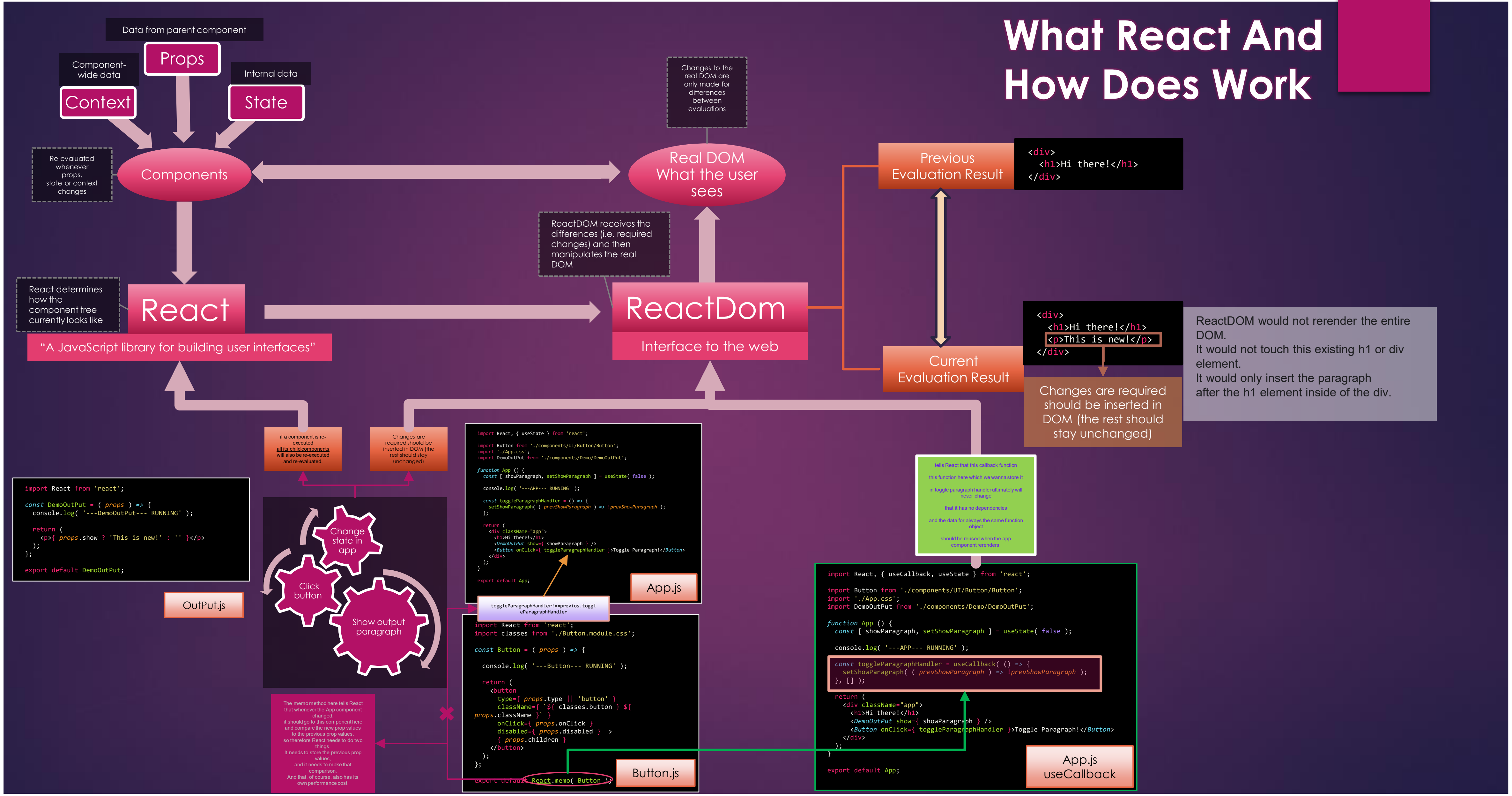


```jsx
      return (
      <Card className={ classes.login }>
        <form onSubmit={ submitHandler }>
          <Input
            ref={ emailInputRef }
            id="email"
            label="E-Mail"
            type="email"
            isValid={ emailIsValid }
            value={ emailState.value }
            onChange={ emailChangeHandler }
            onBlur={ validateEmailHandler }
          />
          <Input
            ref={ passwordInputRef }
            id="password"
            label="Password"
            type="password"
            isValid={ passwordIsValid }
            value={ passwordState.value }
            onChange={ passwordChangeHandler }
            onBlur={ validatePasswordHandler }
          />
          <div className={ classes.actions }>
            <Button type="submit" className={ classes.btn }>
              Login
            </Button>
          </div>
        </form>
      </Card>
    );
```

Init UI Component / Input

Use component UI Input in component Login

What React And How Does Work



Re-evaluated whenever props, state or context changes

React determines how the component tree currently looks like

"A JavaScript library for building user interfaces"

Changes to the real DOM are only made for differences between evaluations

ReactDOM receives the differences (i.e. required changes) and then manipulates the real DOM

```
<div>
  <h1>Hi there!</h1>
</div>
```

```
<div>
  <h1>Hi there!</h1>
  <p>This is new!</p>
</div>
```

Changes are required should be inserted in DOM (the rest should stay unchanged)

ReactDOM would not rerender the entire DOM.
It would not touch this existing h1 or div element.
It would only insert the paragraph after the h1 element inside of the div.

if a component is re-executed
all its child components
will also be re-executed and re-evaluated.

Changes are required should be inserted in DOM (the rest should stay unchanged)

OutPut.js

```
import React from 'react';

const DemoOutPut = ( props ) => {
  console.log( '---DemoOutPut--- RUNNING' );

  return (
    <p>{ props.show ? 'This is new!' : '' }</p>
  );
};

export default DemoOutPut;
```

App.js

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

import Button from './components/UI/Button/Button';
import './App.css';
import DemoOutPut from './components/Demo/DemoOutPut';

function App () {
  const [ showParagraph, setShowParagraph ] = useState( false );

  console.log( '---APP--- RUNNING' );

  const toggleParagraphHandler = () => {
    setShowParagraph( ( prevShowParagraph ) => !prevShowParagraph );
  };

  return (
    <div className="app">
      <h1>Hi there!</h1>
      <DemoOutPut show={ showParagraph } />
      <Button onClick={ toggleParagraphHandler }>Toggle Paragraph!</Button>
    </div>
  );
}

export default App;
```

toggleParagraphHandler!==previos.togglеParagraphHandler

Button.js

```
import React from 'react';
import classes from './Button.module.css';

const Button = ( props ) => {

  console.log( '---Button--- RUNNING' );

  return (
    <button
      type={ props.type || 'button' }
      className={ `${ classes.button } ${
props.className }` }
      onClick={ props.onClick }
      disabled={ props.disabled }  >
      { props.children }
    </button>
  );
};

export default React.memo( Button );
```

The memo method here tells React that whenever the App component changed,
it should go to this component here and compare the new prop values to the previous prop values,
so therefore React needs to do two things.
It needs to store the previous prop values,
and it needs to make that comparison.
And that, of course, also has its own performance cost.

tells React that this callback function

this function here which we wanna store it

in toggle paragraph handler ultimately will never change

that it has no dependencies

and the data for always the same function object

should be reused when the app component rerenders.

App.js useCallback

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

import Button from './components/UI/Button/Button';
import './App.css';
import DemoOutPut from './components/Demo/DemoOutPut';

function App () {
  const [ showParagraph, setShowParagraph ] = useState( false );

  console.log( '---APP--- RUNNING' );

  const toggleParagraphHandler = useCallback( () => {
    setShowParagraph( ( prevShowParagraph ) => !prevShowParagraph );
  }, [] );

  return (
    <div className="app">
      <h1>Hi there!</h1>
      <DemoOutPut show={ showParagraph } />
      <Button onClick={ toggleParagraphHandler }>Toggle Paragraph!</Button>
    </div>
  );
}

export default App;
```

useMemo

App

```
import React, { useState, useCallback, useMemo } from 'react';

import './App.css';
import DemoList from './components/Demo/DemoList';
import Button from './components/UI/Button/Button';

function App () {
  const [ listTitle, setListTitle ] = useState( 'My List' );

  const changeTitleHandler = useCallback( () => {
    setListTitle( 'New Title' );
  }, [] );

  const listItems = useMemo( () => [ 5, 3, 1, 10, 9 ], [] );

  return (
    <div className="app">
      <DemoList title={ listTitle } items={ listItems } />
      <Button onClick={ changeTitleHandler }>Change List Title</Button>
    </div>
  );
}

export default App;
```

DemoList

```
import React, { useMemo } from 'react';

import classes from './DemoList.module.css';

const DemoList = ( props ) => {
  const { items } = props;

  const sortedList = useMemo( () => {
    console.log( 'Items sorted' );
    return items.sort( ( a, b ) => a - b );
  }, [ items ] );
  console.log( 'DemoList RUNNING' );

  return (
    <div className={ classes.list }>
      <h2>{ props.title }</h2>
      <ul>
        { sortedList.map( ( item ) => (
          <li key={ item }>{ item }</li>
        ) ) }
      </ul>
    </div>
  );
};

export default React.memo( DemoList );
```

Button

```
import React from 'react';

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

const Button = ( props ) => {
  console.log( 'Button RUNNING' );
  return (
    <button
      type={ props.type || 'button' }
      className={ `${ classes.button } ${ props.className }` }
      onClick={ props.onClick }
      disabled={ props.disabled }
    >
      { props.children }
    </button>
  );
};

export default React.memo( Button );
```

to ensure that we don't unnecessarily pass a new array here,

keep expensive, resource intensive functions from needlessly running

tells React that this callback function

this function here which we wanna store it

in toggle button title handler ultimately will never change

The memo method here tells React
that whenever the App component changed,
it should go to this component here
and compare the new prop values to the previous prop values,
so therefore React needs to do two things.
It needs to store the previous prop values,
and it needs to make that comparison.
And that, of course, also has its own performance cost.

fetchData -1

```
fetchimport React, { useState } from 'react';

import MoviesList from './components/MoviesList';
import './App.css';

function App() {
  const [movies, setMovies] = useState([]);

  function fetchMoviesHandler() {
    fetch('https://swapi.dev/api/films/')
     .then((response) => {
       return response.json();
     })
     .then((data) => {
       const transformedMovies = data.results.map((movieData) => {
         return {
           id: movieData.episode_id,
           title: movieData.title,
           openingText: movieData.opening_crawl,
           releaseDate: movieData.release_date,
         };
       });
       setMovies(transformedMovies);
     });
  }

  return (
    <React.Fragment>
      <section>
        <button onClick={fetchMoviesHandler}>Fetch Movies</button>
      </section>
      <section>
        <MoviesList movies={movies} />
      </section>
    </React.Fragment>
  );
}

export default App;
```

1- add async to function handler fetch data

2- active loader

3 – init response const and add await to fetch

4- convert response to json and store it in data const

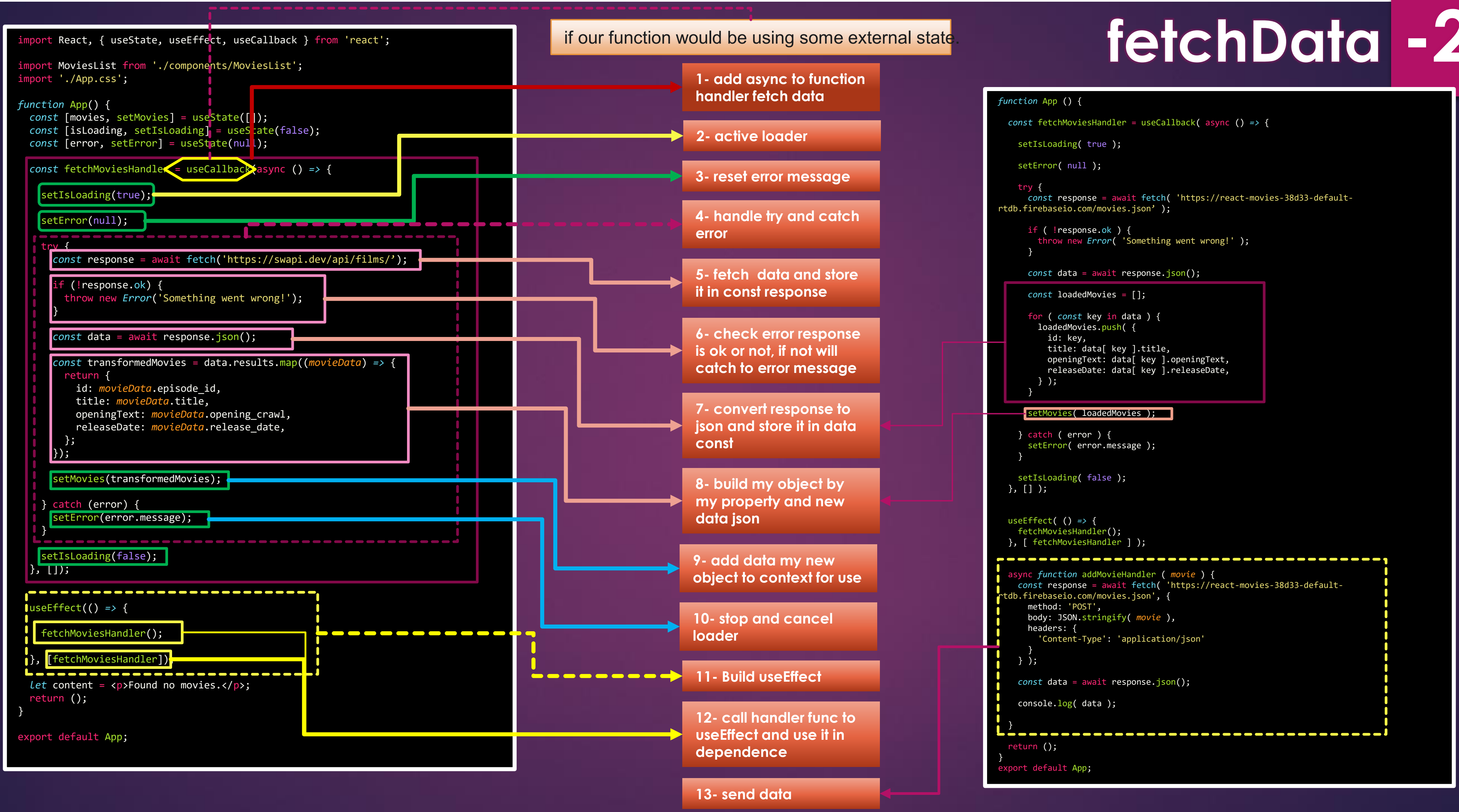
5- build my object by my property and new data json

6- add data my new object to context for use

7- stop and cancel loader

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

import MoviesList from './components/MoviesList';
import './App.css';

function App () {
  const [ movies, setMovies ] = useState( [] );
  const [ isLoading, setIsLoading ] = useState( false );

  async function fetchMoviesHandler () {
    setIsLoading( true );
    const response = await fetch( 'https://swapi.dev/api/films/' );
    const data = await response.json();
    const transformedMovies = data.results.map( ( movieData ) => {
      return {
        id: movieData.episode_id,
        title: movieData.title,
        openingText: movieData.opening_crawl,
        releaseDate: movieData.release_date,
      };
    } );
    setMovies( transformedMovies );
    setIsLoading( false );
  }

  return (
    <React.Fragment>
      <section>
        <button onClick={ fetchMoviesHandler }>Fetch Movies</button>
      </section>
      <section>
        { !isLoading && movies.length > 0 && <MoviesList movies={ movies } /> }
        { !isLoading && movies.length === 0 && <p>Found no movies.</p> }
        { isLoading && <p>Loading...</p> }
      </section>
    </React.Fragment>
  );
}

export default App;
```

fetchData -2



if our function would be using some external state.

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

import MoviesList from './components/MoviesList';
import './App.css';

function App() {
  const [movies, setMovies] = useState([]);
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState(null);

  const fetchMoviesHandler = useCallback(async () => {
    setIsLoading(true);
    setError(null);
    try {
      const response = await fetch('https://swapi.dev/api/films/');
      if (!response.ok) {
        throw new Error('Something went wrong!');
      }
      const data = await response.json();
      const transformedMovies = data.results.map((movieData) => {
        return {
          id: movieData.episode_id,
          title: movieData.title,
          openingText: movieData.opening_crawl,
          releaseDate: movieData.release_date,
        };
      });
      setMovies(transformedMovies);
    } catch (error) {
      setError(error.message);
    }
    setIsLoading(false);
  }, []);

  useEffect(() => {
    fetchMoviesHandler();
  }, [fetchMoviesHandler]);

  let content = <p>Found no movies.</p>;
  return ();
}

export default App;
```

1- add async to function handler fetch data

2- active loader

3- reset error message

4- handle try and catch error

5- fetch data and store it in const response

6- check error response is ok or not, if not will catch to error message

7- convert response to json and store it in data const

8- build my object by my property and new data json

9- add data my new object to context for use

10- stop and cancel loader

11- Build useEffect

12- call handler func to useEffect and use it in dependence

13- send data

```
function App () {

  const fetchMoviesHandler = useCallback( async () => {

    setIsLoading( true );

    setError( null );

    try {
      const response = await fetch( 'https://react-movies-38d33-default-
rtdb.firebaseio.com/movies.json’ );

      if ( !response.ok ) {
        throw new Error( 'Something went wrong!' );
      }

      const data = await response.json();

      const loadedMovies = [];

      for ( const key in data ) {
        loadedMovies.push( {
          id: key,
          title: data[ key ].title,
          openingText: data[ key ].openingText,
          releaseDate: data[ key ].releaseDate,
        } );
      }

      setMovies( loadedMovies );

    } catch ( error ) {
      setError( error.message );
    }

    setIsLoading( false );
  }, [] );


  useEffect( () => {
    fetchMoviesHandler();
  }, [ fetchMoviesHandler ] );

  async function addMovieHandler ( movie ) {
    const response = await fetch( 'https://react-movies-38d33-default-
rtdb.firebaseio.com/movies.json', {
      method: 'POST',
      body: JSON.stringify( movie ),
      headers: {
        'Content-Type': 'application/json'
      }
    } );

    const data = await response.json();

    console.log( data );

  }

  return ();
}
export default App;
```

customHooks -2

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

const useCounter = (forwards = true) => {
  const [counter, setCounter] = useState(0);

  useEffect(() => {
    const interval = setInterval(() => {
      if (forwards) {
        setCounter((prevCounter) => prevCounter + 1);
      } else {
        setCounter((prevCounter) => prevCounter - 1);
      }
    }, 1000);

    return () => clearInterval(interval);
  }, [forwards]);

  return counter;
};

export default useCounter;
```

```
import useCounter from '../hooks/use-counter';

const ForwardCounter = () => {
  const counter = useCounter();

  return <Card>{counter}</Card>;
};

export default ForwardCounter;
```

```
import useCounter from '../hooks/use-counter';

const BackwardCounter = () => {
  const counter = useCounter(false);

  return <Card>{counter}</Card>;
};

export default BackwardCounter;
```

1 – create folder hooks

3 – create custom hook file 'use-counter.js'

2 - select the logic looped

4 – reuse the logic selected in custom hook

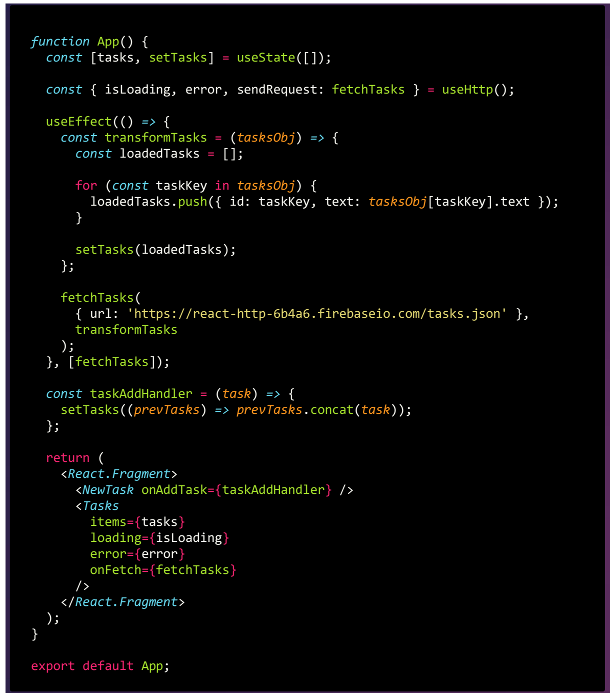
5 – edite logic for Matching all states in components where I need to use

6 – return in custom hook what I need from this custom hook for get it when use the custom hook in my components

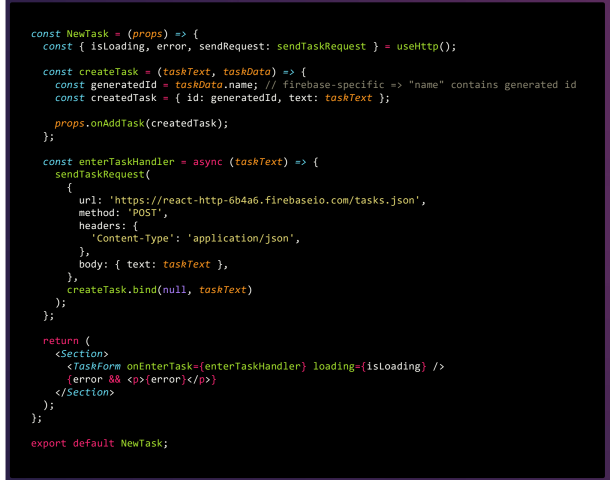
```
import { useState, useEffect } from 'react';

import Card from './Card';

const ForwardCounter = () => {
  const [counter, setCounter] = useState(0);

  useEffect(() => {
    const interval = setInterval(() => {
      setCounter((prevCounter) => prevCounter + 1);
    }, 1000);

    return () => clearInterval(interval);
  }, []);

  return <Card>{counter}</Card>;
};

export default ForwardCounter;
```

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

import Card from './Card';

const BackwardCounter = () => {

  const [counter, setCounter] = useState(0);

  useEffect(() => {
    const interval = setInterval(() => {
      setCounter((prevCounter) => prevCounter - 1);
    }, 1000);

    return () => clearInterval(interval);
  }, []);

  return <Card>{counter}</Card>;
};

export default BackwardCounter;
```

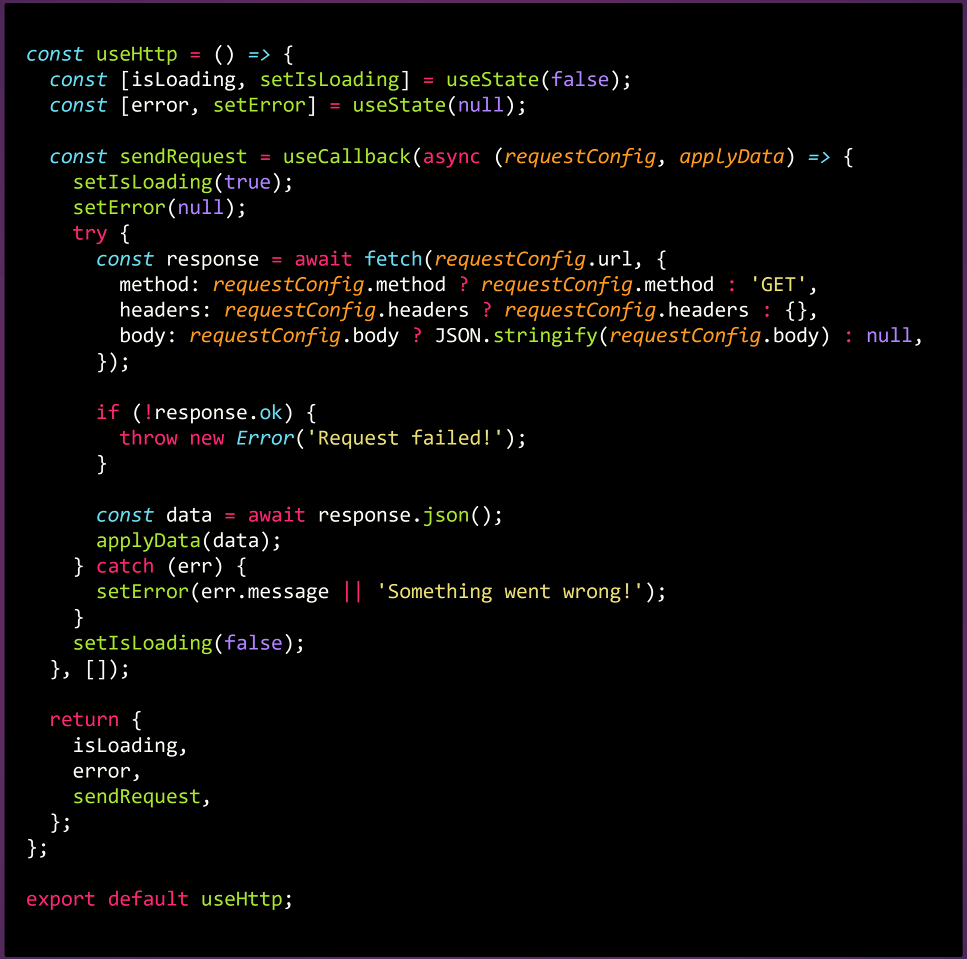
customHooks -3



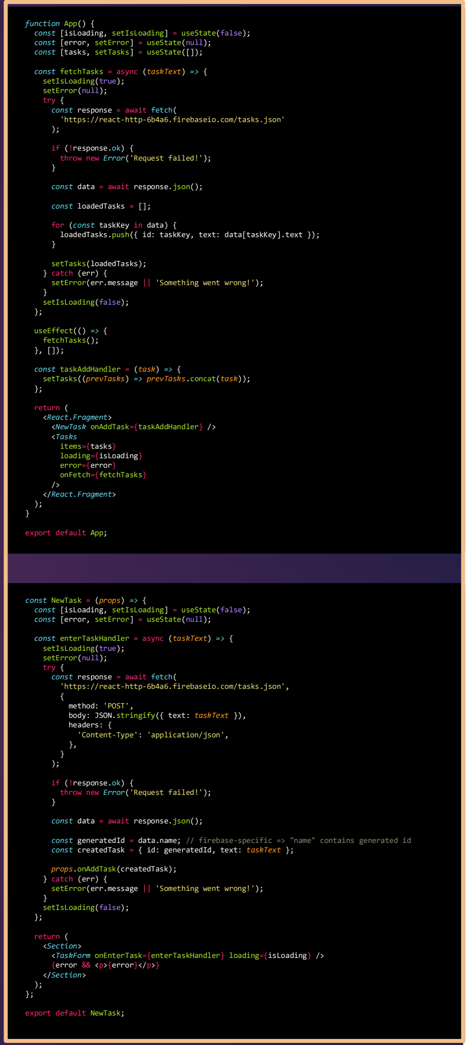
```
function App() {
  const [tasks, setTasks] = useState([]);

  const { isLoading, error, sendRequest: fetchTasks } = useHttp();

  useEffect(() => {
    const transformTasks = (tasksObj) => {
      const loadedTasks = [];

      for (const taskKey in tasksObj) {
        loadedTasks.push({ id: taskKey, text: tasksObj[taskKey].text });
      }

      setTasks(loadedTasks);
    };

    fetchTasks(
      { url: 'https://react-http-6b4a6.firebaseio.com/tasks.json' },
      transformTasks
    );
  }, [fetchTasks]);

  const taskAddHandler = (task) => {
    setTasks((prevTasks) => prevTasks.concat(task));
  };

  return (
    <React.Fragment>
      <NewTask onAddTask={taskAddHandler} />
      <Tasks
        items={tasks}
        loading={isLoading}
        error={error}
        onFetch={fetchTasks}
      />
    </React.Fragment>
  );
}

export default App;
```



```
const NewTask = (props) => {
  const { isLoading, error, sendRequest: sendTaskRequest } = useHttp();

  const createTask = (taskText, taskData) => {
    const generatedId = taskData.name; // firebase-specific => "name" contains generated id
    const createdTask = { id: generatedId, text: taskText };

    props.onAddTask(createdTask);
  };

  const enterTaskHandler = async (taskText) => {
    sendTaskRequest(
      {
        url: 'https://react-http-6b4a6.firebaseio.com/tasks.json',
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: { text: taskText },
      },
      createTask.bind(null, taskText)
    );
  };

  return (
    <Section>
      <TaskForm onEnterTask={enterTaskHandler} loading={isLoading} />
      {error && <p>{error}</p>}
    </Section>
  );
};

export default NewTask;
```



```
const useHttp = () => {
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState(null);

  const sendRequest = useCallback(async (requestConfig, applyData) => {
    setIsLoading(true);
    setError(null);
    try {
      const response = await fetch(requestConfig.url, {
        method: requestConfig.method ? requestConfig.method : 'GET',
        headers: requestConfig.headers ? requestConfig.headers : {},
        body: requestConfig.body ? JSON.stringify(requestConfig.body) : null,
      });

      if (!response.ok) {
        throw new Error('Request failed!');
      }

      const data = await response.json();
      applyData(data);
    } catch (err) {
      setError(err.message || 'Something went wrong!');
    }
    setIsLoading(false);
  }, []);

  return {
    isLoading,
    error,
    sendRequest,
  };
};

export default useHttp;
```



```
function App() {
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState(null);
  const [tasks, setTasks] = useState([]);

  const fetchTasks = async (taskText) => {
    setIsLoading(true);
    setError(null);
    try {
      const response = await fetch(
        'https://react-http-6b4a6.firebaseio.com/tasks.json'
      );

      if (!response.ok) {
        throw new Error('Request failed!');
      }

      const data = await response.json();

      const loadedTasks = [];

      for (const taskKey in data) {
        loadedTasks.push({ id: taskKey, text: data[taskKey].text });
      }

      setTasks(loadedTasks);
    } catch (err) {
      setError(err.message || 'Something went wrong!');
    }
    setIsLoading(false);
  };

  useEffect(() => {
    fetchTasks();
  }, []);

  const taskAddHandler = (task) => {
    setTasks((prevTasks) => prevTasks.concat(task));
  };

  return (
    <React.Fragment>
      <NewTask onAddTask={taskAddHandler} />
      <Tasks
        items={tasks}
        loading={isLoading}
        error={error}
        onFetch={fetchTasks}
      />
    </React.Fragment>
  );
}

export default App;
```

```
const NewTask = (props) => {
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState(null);

  const enterTaskHandler = async (taskText) => {
    setIsLoading(true);
    setError(null);
    try {
      const response = await fetch(
        'https://react-http-6b4a6.firebaseio.com/tasks.json',
        {
          method: 'POST',
          body: JSON.stringify({ text: taskText }),
          headers: {
            'Content-Type': 'application/json',
          },
        }
      );

      if (!response.ok) {
        throw new Error('Request failed!');
      }

      const data = await response.json();

      const generatedId = data.name; // firebase-specific => "name" contains generated id
      const createdTask = { id: generatedId, text: taskText };

      props.onAddTask(createdTask);
    } catch (err) {
      setError(err.message || 'Something went wrong!');
    }
    setIsLoading(false);
  };

  return (
    <Section>
      <TaskForm onEnterTask={enterTaskHandler} loading={isLoading} />
      {error && <p>{error}</p>}
    </Section>
  );
};

export default NewTask;
```