Side-effects:

When ever state updates it re-render whole component. So when http request(sideeffect) was their in also run again. So we use useeffect() hooks

Use effect:

UseEffect run after all component renders and and when dependency changes

If dependency is not changed it run only once.

  useEffect(()=>{

    const storedUserLoggedInformation = localStorage.getItem('isLoggedIn')

    if(storedUserLoggedInformation =='1'){

      setIsLoggedIn(true);

    }

  },[])

In this example ui renders first..then use effect run ..the in setIsLoggedIn state changed then ui re renders

Without dependency:

  useEffect(()=>{

    setFormIsValid(

      enteredEmail.target.value.includes('@') && enteredPassword.trim().length > 6

    );

  })

As same as without use effect..run every time when ui rerenders

With dependencies:

  useEffect(()=>{

    setFormIsValid(

      enteredEmail.target.value.includes('@') && enteredPassword.trim().length > 6

    );

  },[enteredEmail,enteredPassword])

Renders when enteredEmail,enteredPassword changes, we can omit setFormIsValid because it will always not change

enteredEmail,enteredPassword changes also a side effect.side effect of keystroke

* You **DON'T need to add state updating functions** (as we did in the last lecture with setFormIsValid): React guarantees that those functions never change, hence you don't need to add them as dependencies (you could though)
* You also **DON'T need to add variables or functions** you might've **defined OUTSIDE of your components** (e.g. if you create a new helper function in a separate file): Such functions or variables also are not created inside of a component function and hence changing them won't affect your components (components won't be re-evaluated if such variables or functions change and vice-versa)

Here's a made-up dummy example to further clarify the above-mentioned scenarios:

1. import { useEffect, useState } from 'react';
3. let myTimer;
5. const MyComponent = (props) => {
6. const [timerIsActive, setTimerIsActive] = useState(false);
8. const { timerDuration } = props; // using destructuring to pull out specific props values
10. useEffect(() => {
11. if (!timerIsActive) {
12. setTimerIsActive(true);
13. myTimer = setTimeout(() => {
14. setTimerIsActive(false);
15. }, timerDuration);
16. }
17. }, [timerIsActive, timerDuration]);
18. };

In this example:

* timerIsActive is **added as a dependency** because it's component state that may change when the component changes (e.g. because the state was updated)
* timerDuration is **added as a dependency** because it's a prop value of that component - so it may change if a parent component changes that value (causing this MyComponent component to re-render as well)
* setTimerIsActive is **NOT added as a dependency** because it's that **exception**: State updating functions could be added but don't have to be added since React guarantees that the functions themselves never change
* myTimer is **NOT added as a dependency** because it's **not a component-internal variable** (i.e. not some state or a prop value) - it's defined outside of the component and changing it (no matter where) **wouldn't cause the component to be re-evaluated**
* setTimeout is **NOT added as a dependency** because it's **a built-in API** (built-into the browser) - it's independent from React and your components, it doesn't change

Debounce (CleanUp):

  useEffect(()=>{

    const identifier = setTimeout(()=>{

      setFormIsValid(

        enteredEmail.trim().includes('@') && enteredPassword.trim().length > 6

      );

    },500)

    return ()=>{

      console.log("Cleanup")

      clearTimeout(identifier)

    };

  },[setFormIsValid,enteredEmail,enteredPassword])

Return function not run at first render. And it will run first for next every key stroke. So keep cleanup in return. And by this after entering the input I will run only once for setFormIsValid(or api’s).

  useEffect(()=>{

    const identifier = setTimeout(()=>{

console.log(“effect”)

    return ()=>{

      console.log("Cleanup")

      clearTimeout(identifier)

    };

  },[])

If array is empty ..cleanup will run only when component get unmount.

UserReducer():

Effective compare to useState()

The useReducer Hook is similar to the useState Hook.

It allows for custom state logic.

If you find yourself keeping track of multiple pieces of state that rely on complex logic, useReducer may be useful.

Syntax

The useReducer Hook accepts two arguments.

useReducer(<reducer>, <initialState>,<initfn>)

The reducer function contains your custom state logic and the initialStatecan be a simple value but generally will contain an object.

The useReducer Hook returns the current stateand a dispatchmethod.

    const passwordReducer =(state,action) =>{

      if (action.type ==='USER\_INPUT'){

        return {value:action.val,isValid:action.val.trim().length > 6}

      }

      if(action.type=="INPUT\_BLUR"){

        return {value:state.value,isValid:state.value.trim().length > 6}

      }

      return {value:'',isValid:false}

    }

    const [passwordState,dispatchPassword] =useReducer(passwordReducer,

      {value:'',isValid:null});

  const passwordChangeHandler = (event) => {

    dispatchPassword({type:'USER\_INPUT',val:event.target.value});

    setFormIsValid(

      emailState.value.includes('@') && passwordState.value.length > 6

          );

  };

useReducer & useEffect:

  const {isValid:emailIsValid} = emailState

  const {isValid:emailIsValid} = emailState

  const {isValid:passwordIsValid} = passwordState

  useEffect(()=>{

    const identifier = setTimeout(()=>{

      setFormIsValid(

        emailIsValid && passwordIsValid

      );

    },500)

    return ()=>{

      console.log("Cleanup")

      clearTimeout(identifier)

    };

  },[emailIsValid,passwordIsValid])

We use emailState.isValid by object destruction///.so that it will check only is valid is changed

import Card from '../UI/Card/Card';

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

import Button from '../UI/Button/Button';

import { useReducer,useState } from 'react';

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 emailChangeHandler = (event) => {

    dispatchEmail({type:'USER\_INPUT',val:event.target.value});

    // setFormIsValid(

    //   emailState.value.includes('@') && passwordState.value.trim().length > 6

    //       );

  };

  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 [emailState,dispatchEmail] =useReducer(emailReducer,

    {value:'',isValid:null});

    const passwordReducer =(state,action) =>{

      if (action.type ==='USER\_INPUT'){

        return {value:action.val,isValid:action.val.trim().length > 6}

      }

      if(action.type=="INPUT\_BLUR"){

        return {value:state.value,isValid:state.value.trim().length > 6}

      }

      return {value:'',isValid:false}

    }

    const [passwordState,dispatchPassword] =useReducer(passwordReducer,

      {value:'',isValid:null});

  const passwordChangeHandler = (event) => {

    dispatchPassword({type:'USER\_INPUT',val:event.target.value});

    // setFormIsValid(

    //   emailState.value.includes('@') && passwordState.value.length > 6

    //       );

  };

  const {isValid:emailIsValid} = emailState

  const {isValid:passwordIsValid} = passwordState

  useEffect(()=>{

    const identifier = setTimeout(()=>{

      setFormIsValid(

        emailIsValid && passwordIsValid

      );

    },500)

    return ()=>{

      console.log("Cleanup")

      clearTimeout(identifier)

    };

  },[emailIsValid,passwordIsValid])

  const validateEmailHandler = () => {

    dispatchEmail({type:"INPUT\_BLUR"});

  };

  const validatePasswordHandler = () => {

    dispatchPassword({type:"INPUT\_BLUR"});

  };

  const submitHandler = (event) => {

    event.preventDefault();

    props.onLogin(emailState.value, passwordState.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} ${

            passwordState.isValid === false ? classes.invalid : ''

          }`}

        >

          <label htmlFor="password">Password</label>

          <input

            type="password"

            id="password"

            value={passwordState.value}

            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;

**Adding Nested Properties As Dependencies To useEffect**

In the previous lecture, we used object destructuring to add object properties as dependencies to useEffect().

1. const { someProperty } = someObject;
2. useEffect(() => {
3. // code that only uses someProperty ...
4. }, [someProperty]);

This is a **very common pattern and approach**, which is why I typically use it and why I show it here (I will keep on using it throughout the course).

I just want to point out, that they **key thing is NOT that we use destructuring** but that we **pass specific properties instead of the entire object** as a dependency.

We could also write this code and it would **work in the same way**.

1. useEffect(() => {
2. // code that only uses someProperty ...
3. }, [someObject.someProperty]);

This works just fine as well!

But you should **avoid** this code:

1. useEffect(() => {
2. // code that only uses someProperty ...
3. }, [someObject]);

Why?

Because now the **effect function would re-run whenever ANY property** of someObject changes - not just the one property (someProperty in the above example) our effect might depend on.

ContextAPI:

TO AVOID PROPS DRILLING

Method1     <AuthContext.Consumer>

Below one help in autocomplete while typing

import React from "react";

const AuthContext = React.createContext({

    isLoggedIn: false

});

export default AuthContext;

App.js

  return (

    <React.Fragment>

      <AuthContext.Provider value={{

    isLoggedIn: isLoggedIn

}}>

        <MainHeader onLogout={logoutHandler} />

        <main>

          {!isLoggedIn && <Login onLogin={loginHandler} />}

          {isLoggedIn && <Home onLogout={logoutHandler} />}

        </main>

      </AuthContext.Provider>

    </React.Fragment>

  );

    <AuthContext.Consumer>

      {(ctx)=>{

        console.log(ctx)

        return (

          <nav className={classes.nav}>

          <ul>

            {ctx.isLoggedIn && (

              <li>

                <a href="/">Users</a>

              </li>

            )}

          </ul>

        </nav>

        )

      }}

    </AuthContext.Consumer>

Method2 (useContext())

import React,{useContext} from 'react';

import AuthContext from '../../store/auth-context';

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

const Navigation = (props) => {

  ctx = useContext(AuthContext)

  return (

          <nav className={classes.nav}>

          <ul>

            {ctx.isLoggedIn && (

              <li>

                <a href="/">Users</a>

              </li>

            )}

          </ul>

        </nav>

  );

};

export default Navigation;

StandAlone Auth Controller:

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 storedUserLoggedInformation = localStorage.getItem('isLoggedIn')

      if(storedUserLoggedInformation =='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,onLogin:loginHandler,onLogout:logoutHandler}}>

            {props.children}

        </AuthContext.Provider>

      )

}

export default AuthContext;

index.js

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>

);

App.js

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>

      {/\* <AuthContext.Provider value={{

            isLoggedIn: isLoggedIn,

            onLogout:logoutHandler

        }}> \*/}

        <MainHeader />

        <main>

          {!ctx.isLoggedIn && <Login />}

          {ctx.isLoggedIn && <Home />}

        </main>

      {/\* </AuthContext.Provider> \*/}

    </React.Fragment>

  );

}

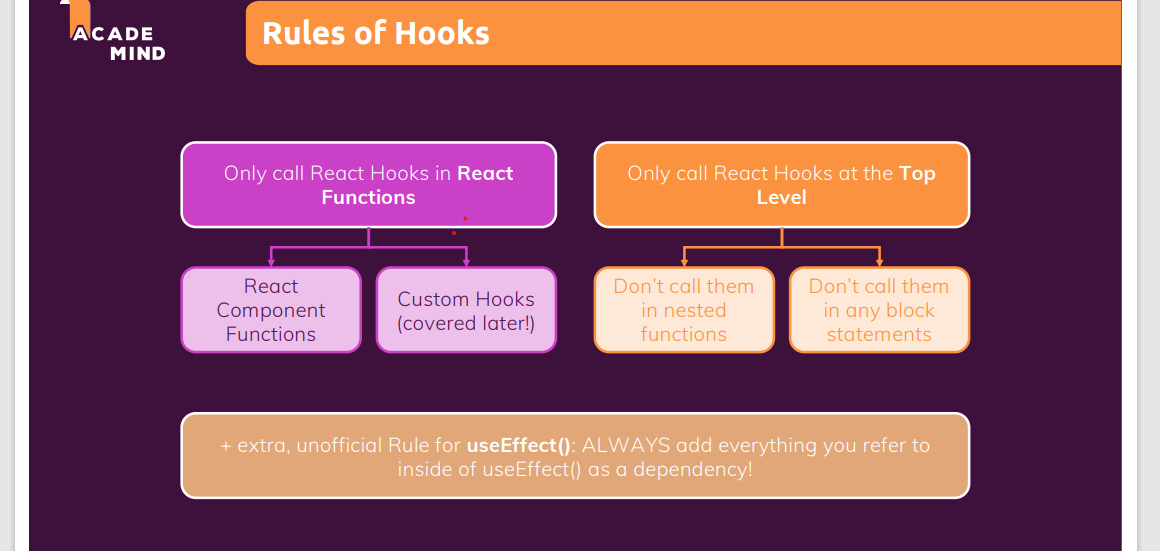
export default App;

React Context Limitations:

Props for configuration.context for state management across the component.

React Context not optimized for high frequency changes

Rules of hooks:



Forward Ref:

Used to forward ref to other component

Input.js:

import { useRef,useImperativeHandle } from "react";

import React from 'react';

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

    const inputRef = useRef();

    const activate=() =>{

        inputRef.current.focus()

    }

    useImperativeHandle(ref,()=>{

        return {

          focus:activate

        }

      })

  return (

    <div

          className={props.className}

        >

          <label htmlFor={props.htmlFor}>{props.label}</label>

    <input

    type={props.type}

    id={props.id}

    value={props.value}

    onChange={props.onChange}

    onBlur={props.onBlur}

    ref={inputRef}

  />

  </div>

  );

});

export default Input;

login.js:

import Card from '../UI/Card/Card';

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

import Button from '../UI/Button/Button';

import { useReducer,useState,useEffect, useContext,useRef  } from 'react';

import AuthContext from '../../store/auth-context';

import Input from '../UI/Input/Input';

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 AuthCtx = useContext(AuthContext)

  const emailRef  =useRef()

  const passwordRef  =useRef()

  const emailChangeHandler = (event) => {

    dispatchEmail({type:'USER\_INPUT',val:event.target.value});

    // setFormIsValid(

    //   emailState.value.includes('@') && passwordState.value.trim().length > 6

    //       );

  };

  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 [emailState,dispatchEmail] =useReducer(emailReducer,

    {value:'',isValid:null});

    const passwordReducer =(state,action) =>{

      if (action.type ==='USER\_INPUT'){

        return {value:action.val,isValid:action.val.trim().length > 6}

      }

      if(action.type=="INPUT\_BLUR"){

        return {value:state.value,isValid:state.value.trim().length > 6}

      }

      return {value:'',isValid:false}

    }

    const [passwordState,dispatchPassword] =useReducer(passwordReducer,

      {value:'',isValid:null});

  const passwordChangeHandler = (event) => {

    dispatchPassword({type:'USER\_INPUT',val:event.target.value});

    // setFormIsValid(

    //   emailState.value.includes('@') && passwordState.value.length > 6

    //       );

  };

  const {isValid:emailIsValid} = emailState

  const {isValid:passwordIsValid} = passwordState

  useEffect(()=>{

    const identifier = setTimeout(()=>{

      setFormIsValid(

        emailIsValid && passwordIsValid

      );

    },500)

    return ()=>{

      console.log("Cleanup")

      clearTimeout(identifier)

    };

  },[emailIsValid,passwordIsValid])

  const validateEmailHandler = () => {

    dispatchEmail({type:"INPUT\_BLUR"});

  };

  const validatePasswordHandler = () => {

    dispatchPassword({type:"INPUT\_BLUR"});

  };

  const submitHandler = (event) => {

    event.preventDefault();

    if(formIsValid){

      AuthCtx.onLogin(emailState.value, passwordState.value);

    }else if (!emailIsValid){

      emailRef.current.focus()

    }else if (!passwordIsValid){

      passwordRef.current.focus()

    }

  };

  return (

    <Card className={classes.login}>

      <form onSubmit={submitHandler}>

          <Input type="email"

            id="email"

            className={`${classes.control} ${

              emailState.isValid === false ? classes.invalid : ''

            }`}

            ref={emailRef}

            htmlFor="email"

            label="Email"

            value={emailState.value}

            onChange={emailChangeHandler}

            onBlur={validateEmailHandler}/>

          <Input

            type="password"

            id="password"

            htmlFor="password"

            label="Password"

            ref={passwordRef}

            className={`${classes.control} ${

              passwordState.isValid === false ? classes.invalid : ''

            }`}

            value={passwordState.value}

            onChange={passwordChangeHandler}

            onBlur={validatePasswordHandler}

          />

        <div className={classes.actions}>

          <Button type="submit" className={classes.btn} >

            Login

          </Button>

        </div>

      </form>

    </Card>

  );

};

export default Login;