**BASIC**

**Setting up working environment**

Configure vs code:

* Esling: it automatically finds error in our code or the way of code we are violating
* Prettier: will automatically format our code empty line and other staff like that
* Material icon theme: this will give the best icon to the file we create

Autosave:

* onFocusChange: this will save the change once you move up on different tab

default Formatter: to activate prettier

* Prettier -code formatter
* Format on save, turn it on by checking it (the format should for done on save of the code)

esLint Run:

* onSave (by default it saves once you type new character, then you have to limit it)

snippets file to speed up our development: settings>user Snippets>new Global Snippets file>put your Json file. Like rfc to create react component, rsc for creating react components and so on

Auto Rename tag is vs extension: this will help you to rename tag at opening and to it for the closing one.

INITIATING:

We should start writing this react bey open our vs code and type in our terminal NPX CREATE-REACT-APP name

This will automatically create the project with the specified name. and there are some restriction on the name of the project. Like uppercase things and so other restriction

But this is not only way to create react app. We can also use VITE but VITE will require us to configure it from scratch. For beginner you can use create-react-app, but on real word project you have to use VITE

VITE is extremely faster to road pages and more other advantages like not to call you initial page index and so on.

CONDITIONAL RENDERING (three way of rendering JSX):

* AND operator:
  + let k = true;
  + console.log (k && "Halloween day"); output will be Halloween day. And when you put false the output is going to be false. But react doesn’t actually render true or false into the Dom but it may render 0 like once you conditioning based on the length of the array. So that why this will work. If it’s false it will do nothing other ways it will display the content inside the<p>
  + this is what we are going to do in this conditional rendering. { isOpen&&<p>We are open</p>} , if the condition is true then the second part will be executed
* TERNARY operator {isOpen? <p>we are open</p> : <p>we are not open</p>}
  + className=` first-class ${soldOut ? ’solidBlack’: ‘solidWhite’}`
  + The is better it give two alternatives and you can make it more. But once you chain more thing like this it can confuse the reader of the code and may not know what is going on. Ex: isOpen? Result1 ?(result2 : result3): (result4?(result5:result6) ). Imagin this at least without any brackets
* MULTIPLE RETURNS used to render entire page/component based on condition. But not some pieces of JSX:
  + If(condition) return (JSX) else return (other JSX)
  + To refactor variable like if (pizza.soldeout)return null; all sold out will be null

REACT FLAGMENT

When we create div all over the palaces it will not messed up but it may not look perfect. Then we use react fragment in different situations <>…..</> or <React.Fragment >…</React.Fragment>

* Fragment can not have className
* When you have multiple component to wrap all of them together under the single div it will mess up with it’s CSS, Then you have to wrap it into Fragment
  + In short fragment lets us group some elements without leaving any trace in HTML tree.
  + Some times we need to give fragment a key once we are using it to render the list. Then we use the second one <React.Fragment key={}>

COMPONENT SIZE MATTERS

The huge component have to many responsibilities: it may contain code that should be complex. is we need like 10 to 15 props to configure the component means that component is to be big, then it should be broken down.

The small component: we may ends up with hundreds of components. The code base is supper confusing to navigate and understand.

Here is criteria to base on

1 logical separation of content/layout: must contain only related content

2 reusability: how many time are you going to need it,

3 responsibility: it’s responsibility is to huge then break it into many components

4 Personal coding style:

If you write problem down clear and specifically you have solved the half of it.

When you take task, finding the best ways to achieve the desired results, is always you responsibility

If you prioritize knowledge and intelligence , money will continue to came

If you don’t have to make decision about something, then don’t decide.

TYPES OF COMPONETS:

* Presentational component: these are component that present data. They have no state, and may receive data as props
* Stateful component : these are components that have states
* Structural component: let think this like pages, layouts or screens. Then is result of composing many components

PROPS DRILLING

This is the problem of passing props to the child of child of child. This is the not cool into bit application. It may leads you to the confusion.

As we know if two component need data from same state, you have to create that state in parent component of these two/many component and they receive those data as props. Then this may cause prop drilling

COMPONENT COMPOSITION

Function Model(){<div> <Success/> </div>}

Function Success(){…………}

If we do this then the success is inside the Model component. Then we couldn’t use this model component to display some other message inside of it. Ex: error message.

Function Model({ children}){<div> {children} </div>}

Then from this. The model is not predefined component. Instead it accepts children with the children prop. Then we can pass it success or error or any other thing we wanted. This is component composition: is combining different components using the children prop

SECTION 11

Modern JavaScript student[activeOne] === student.at(activeOne)

Virtual dom: is a tree of all react element created from all instances in the component tree. (all created ) react project). Then if at some point the parent component change the state no matter what, the children will also be re rendered as well.

SECTION 12

It’s not allowed to set state in render logic. Because it may cause the logic to re render infinite times.

Ex: fetch(‘Url’……………then(res)=> setArray(res)) this will cause infinite times re render of componet. then the best practice is to use useEffect hook.

useEffect(()=>{fetch(………..then(res)=setArray(res)) }, [Array]): at this level problem is solved because the data will be re rendered once the there is change in array state.

Mostly we use side effect hook when we need to interact with outside world ,not in exactly app render logic. And we use event handlers to react to a certain event that happened in the UI and we better not use to much side effect

As we know asnyc function return promise then once we put async function inside the useEffect Hoot we get an error because use effect dosent need to return anything. To handle that we put another function inside for preventing that error

useEffect(function kk(){async()=> {

const result = await fetch(‘url’);

const v = await result.json()

setArray(v)

}

Kk() // call kk in older to give effect those data.

},[])

useEffect(function () {

    async function datas() {

      try {

        setIsLoading(true); // the data at this points are loading

        let result = await fetch("http://localhot:500/api/v1/post/likes/:id");

        // once data cant be fetched

        if (!result.ok) throw new Error("something went wrong");

        let v = await result.json();

        // set Something here to change once the list changes

        console.log(v);

      } catch (err) {

        setError(err.message);

      } finally {

        // this brock will be executed at the end always

// and we have to set is loading to false because there is error /data

        setIsLoading(false);

      }

    }

    datas();

  }, []);

useEffect(function , [x,y,z]) //this effect will be executed at x,y,z states

useEffect(function , []) // this effect will be executed at first time render

useEffect(function ) // this effect will be executed at every lender of component

CLEANUP FUCNTION

Is function return from the use Effect and is optional:

Its run in two ways: before the effect is executed again, after the component has unmounted to reset the side effect that we created once we need it

BEST PLACTICE: Each effect should do only one thing! Use one useEffect hook for each side effect. This makes effect easier to clean up.

useEffect(

    function () {

      if (friend === null) {

        document.title = "NullFriend Friend";

      } else {

        return function () {

          document.title = "FriendSelected";

        };

      }

    },

    [friend]

  );

This returned function with inside value of FriendSelected will change as friend is setted.

SECTION 13

Vd5 : local storage in browser. As we know each time we reload browser the all functioning data will be lost then we have to handle that. Where those data must be stored in local storage.

Coll function called LOCALSTORAGE that available in all browser

We need to convert all of the data into a string because in local storage we can only store key value pairs, where the value is a string. In most cases we use JSON.stringfy(data)

These data have to be stored in effect in older to be updated once there are changes in data.

useEffect(() => {

    const found = AvailableMovies.filter((ele) => ele.id === moviesId);

    localStorage.setItem(

      "watched",

      found.length > 0

        ? JSON.stringify([...WatchList, { ...found[0], rating }])

        : JSON.stringify([...WatchList])

    );

  }, [WatchList]);

Then these data must be initial value of state, but the main problem these data should re render for first time

// watchList

  // this function in watchlist must be pure function. it must not get parameter

  const [WatchList, setWatchList] = useState(function () {

    return JSON.parse(localStorage.getItem("watched")) || [];

    // we need to convert data back from json(that json.parse) to be accessed other way we may get error

  });

useEffect(() => {

    localStorage.setItem("watched", JSON.stringify(WatchList));

  }, [WatchList]);

USEREF HOOK

We use the useRef hook to crete something called ref.

Ref stands for reference and essentially it’s like a box into which we can put any data that we want to be preserved between renders

Refs used in both

* creating a variable that stays the same between renders(eg; previous state)
* selecting and storing DOM elements

its only appear in event handlers or effects, but not in jsx

// useRef

  const inputEl = useRef(null);

  useEffect(() => {

    inputEl.current.focus();

  }, []);

  return (

    <>

      <div className="NavBar">

        <Log />

        <input ref={inputEl}/>

As the component re render the cursor will be in that input.

Useref is better once we are deling with DOM manipulation . initial must be null once we are dealing with DOM.

// useRef

  const inputEl = useRef(null);

  // once it renders for the first time

  useEffect(() => {

    inputEl.current.focus();

  }, []);

  // once you place inter

  useEffect(() => {

    document.addEventListener("keypress", (e) => {

      if (e.key === "Enter") return inputEl.current.focus();

    });

  });

CUSTOM HOOK

these are logic that needed to be reused. We create custom hook only if and only logic contains any hooks

It’s like normal function it can receive parameter and return the value, but inside there should be any hook like useState, useEffect or any other hook. You must give then name of the function by starting with use otherwise is going to be like normal react function.

SECTION 23

TAILWIND CSS

We can style our project by using inline CSS, EXTERNAL FILE AND WE EMPORT IT AS NAMAL WAY, CSS MODULES , TAILWINDCSS OR material UI/chakra UI/Mantine

Text color: Text- {color}

Font size : text-xs/text-sm/text-base/text-lg/text-xl/text-2xl…………….text-9xl

Font family: font-serif

Font size : font-bold/ font-light

cases: uppercase/ capitalize

letter spacing: tracking-widest/tracking-[2rem]

Background color : bg-{color}/bg-{color}-{intensity}

Place holder: placeholder:mt-2

Border : border-b-8 border-yellow-400

Width: w-72/ min-h-72

Height: h-72/ max-h-72

Add space between elements without flex: space-x-5

Hover: underline hover:text-blue-600

Border radius: rounded-large/-full/-sm/-none

Transition: transition-all duration-300 (this transition will be on all property in 300ms )

BOX MODEL

Margin: m-2/mr-2/ml-2/mt-2/mb-2 || mx-2/my-2 || m-auto/mx-auto/my-auto for margin auto

Padding: p-2/pr-2/px-2/py-2

Display: flex/hidden/inline/block/table/grid

RESPONSIVINESS

Tailwind have 5balck points which are sm (640px)/md(768px)/lg(1024px)/xl(1280px)/2xl(1536px) and you can even customize them. By building new name and different sizes.

Tailwind is mobile first. Class name will be applied From zero up to fixed size like sm == 640

My-10 sm:my-16 once the screen become bigger than 640px then the my-16 will be applied

FLEXBOX

Flex align-midle/align-top/align-bottom

Justify-betwee/justify-center/-end

FORM AND BUTTONS

Foucs: focus: ring (this will make rounded border to the selected item once its being focused on or clicked)

Focus: outline-none focus: ring focus:ring-yellow-400 fous:ring-offset-2

Accent-yellow-400: this is new way of color styling in CSS

RE USING STYLES

You should go in place index.css file under the pasted call like @tailwing base;

Then you type

@layer components{

.input{

@apply w-full rounded-full

}

}

Then once you call input class will be ready to apply all the styles you have applied. But this should be used once there are many class that you are going to apply multiple times. Otherwise there will be no difference in using vanilla CSS

POSITIONING

Absolute: absolute

GRID CSS.

You start by creating adding grid to make grid container.

Once you set height of the div, it will stretch out the element inside it, ex if you have 4 div inside, it will stretch them equally based on the size of the container.

You may define the lows : grid-rows-3 (one for header , body and footer)

You may also define the columns : grid-cols-2(one for left menu and the content at right)

You can even set the gap between the created cells from grid row/columns : gap-4/gap-x-4/…..

SECTION 16

REDUCER HOOK

We use this state to manage complex state, not single value state

// use reducer hook

let initialValue = 1;

function reducerfunc(prevState, actioin) {

  return actioin;

//this returned action will be the new value of values

}

const [values, dispatchValue] = useReducer (reducerfunc, initialValue);

console.log(values);

// this dispatchEvent function will allow us to update values by running the reducerfunc

dispatchEvent({

  type: "die",

  payload: {

    name: "niyonkuru",

    age: 39,

    locations: { loc1: "kigali", loc2: "rubavu" },

    status: "single",

  },

});

// end of user reducer function

Better way

function reducer(prevState, action) {

  switch (action.type) {

    case "inputValiation":

      return { ...prevState, counter: action.payload };

    case "inc":

      return { ...prevState, step: prevState.step + action.payload };

    case "dec":

      return { ...prevState, step: prevState.step - action.payload };

    case "inputValue":

      return { ...prevState, step: action.payload };

    default:

      throw Error("this is un solvable");

  }

}

const initial = { counter: 0, step: 0 };

export const DatesVersion2 = () => {

  const [count, dispatch] = useReducer(reducer, initial);

  const { counter, step } = count;

dispatch function request for state update.

function HandleInputChange(e) {

    const { value } = e.target;

    dispatch({ type: "inputValiation", payload: Number(value) });

  }

  //dates

  function HandleNextDate() {

    dispatch({ type: "inc", payload: counter });

  }

CREATE FAKE SERVER WITH API

You have to go first in your package.json file and in script you to type NPM I JSON-SERVER

“script”{

“server”: “json-server –watch data/questions.json –port 800”

}

THE JSON API IS IN FILE NAMED QUESTIONS.JSON AND IS LOCATED IN FOLDER CALLED DATA

THEN ONCE YOU npm run server that file will be watched

SECTION 17

ROUTING

npm create vite@latest

npm I : to make sure all dependencies comes with vite must be installed

npm i react-router-dom : for routing

we can use <a> to link the pages but the page will reload each time we access the new page. This is why we use Link or NavLink or useNavigate hook to link the pages and avoid rendering of the page. Then that templet is single page application

the main different between Link and NavLink is once you click on the link then it will add class of active. This class should be used to style the active part.

its prohibited to use external CSS in large project. For avoiding wrong naming like two different class but the same name, you better use CSS MODULE: external file per component

BY USING MODEL YOU ARE NOT ALLOWED TO USE TAG NAME BECAUSE IT MAY BE APPLIED TO THE ALL PROJECT INSTEAD OF BEING APPLIED TO THE COMPONENT you are dealing with.

P1.jxs, p1.module.css

.nav{……}

You have to import it in p1.jxs as style or other name you want. The to consume the style

className = {style.nav}

if you want your class to be accessed globally you can use external normal CSS file or you use

:global(.nav){……} in your modul.css file, once you go in other file as normal and use className = (“nav”) style will be applied

THE URL FROM STATE MANAGEMENT

The URL is an excellent place to store UI state and an alternative to useState in some situations

* its is the easy way to store state in a global place , accessible to all components in the app

reactRouteParams is the best in passing data between pages: by using parameters like id and so on.

* First create new route
* Link to that route
* We read the state from url

useSearchParams is very powerful because it reset the query parameters even in the link

setSearchParams({lat:23, lng: 50}).

**Programmatic navigation with useNavigate hook:** allow user to be redirect to certain page without clicking on any link. It’s common on form submission.

Navigate hook is used in single use cases know days.

<Route index element={<City />} />

<Route path={"cities"} element={<City />} />

Instead of having those to link pointing on single component. You should use navigate hook by doing this

 <Route index element={<Navigate to={"cities"} />} />

 <Route path={"cities"} element={<City />} />

But the main problem is to return back. You can place on back arrow thousand time and you stack there. To handle that we use **replace keyword**

 <Route index element={<Navigate replace to={"cities"} />} />

 <Route path={"cities"} element={<City />} />

SECTION 18

USECONTEXT

1ste: create context component that will handle our state PostContext = createContext()

This component must wrap all the code that will consume it’s values: <PostContext>…….</PostContext>

Value: these are value that are passed as props, like x, HandleClickX . like once HandleClickX is function corresponding to restX

Consumer

export const useMvs = () => {

  const context = useContext(MovieContext);

  if (context === undefined)

    throw new Error("Movie context was used outside of the MOvieProvider");

  return context;

};

Once you try to access the props outside of children of MovieContext you will get that error because it will not be available

To implement map we need package leaflet: npm I react-leaflet leaflet react-leaflet is build on top of leaflet

SECTION 19

PERFOMANCE OPTIMIZATION

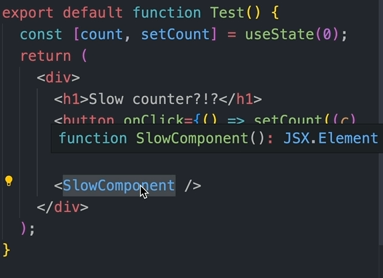
There are three main area we have to focus on once we need to increase performance optimization

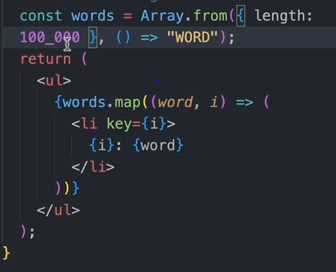
1. We should prevent wasted Renders : by using different library like
   1. Memo
   2. useMemo
   3. useCallback
   4. passing elements as children or regular props
2. Improve app speed/ responsiveness
   1. useMemo
   2. useCallback
   3. model Hook : useTransition
3. Reduce bundle size
   1. Using fewer 3rd party packages
   2. Code splitting and lazy loading

There are more ways like not to render component inside of other component. You better use children because it will avoid parent render once the children is re rendering.

In react the component instance re render in three ways

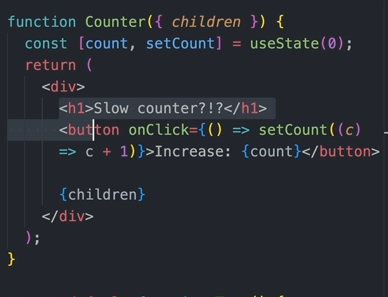
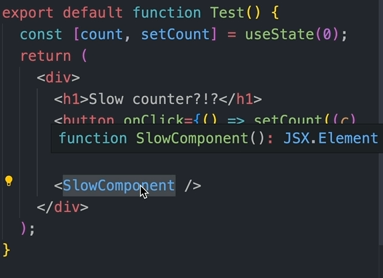
1. When the component state changes
2. Once there is change in context while that component has subscribed to that context
3. Parent re renders: once the state change the child component with props also render

Wasted render : a render that didn’t produce any change in the DOM. The calculation is done and ends 

From this scenario the slow component is component returning 100000 words into our ui

This will be much slower once you are trying to update count because also this part will be re-rendered

1st techniques to increase the optimization by resolving this is to use children props



If you look these two are most similar but only thing you have to do

Then in our test component we need to do this.

Test(){<Counter><SlowComponent/></Counter>}

This will no longer slow down.

2nd TECHINIQUE TO OPTIMIZE THE PERFORMANCE

MEMOIZATION: optional technique that executes a pure function once , and save result In memory (cache). If we try to execute the function again with the same arguments as before, the previously saved result will be returned, without executing the function again

This will prevent wasted re-render as well as improve app speed/responsiveness and expensive calculations

We should go Ahed and memo all of our components?

No: memo is used once we are dealing with heavy components, re-renders often, and does so with the same props. If the props are usually different between re-renders memo has no effect any way,

Ex: Archive component is rendering 30000 output. If we call this component into our app and we try to make some search, it will slow down all our application. Then the better way to handle that is to wrap that function into memo.

Const Archive = memo(function Archive(){……return 300000}

Instead of using archive function we wrap it into memo function

If we memorize function and we give it object porps like(<Archive {queantity:30000,running:true}) then there will be no role of that memo because each time it will render. The object in JavaScript will never be equal with other object, then the memo will think there is change in object, and then re-render again.

THEN AS SOLUTION FROM THIS

We can make object and function stable by memorizing them as well. Then react give us two more hook to interact with this which are. USEMEMO AND USECALLBACK. USEMEMO to memorize values and USECALLBACK to memorize the function between renders. (use callback is special case of use memo)

As we know in use effect there is dependency array that well make effect run once they changes, also these hooks use the same strategy. Once their dependency array changes, they will no longer return value from cache, there will be re-created.

They should be used to avoid like infinite useEffect loop

WHEN TO OPTIMIZE THE CONTEXT. THERE SHOULD BE THREE THINGS.

1. When the state in the context need to change all the time
2. Context have more consumer
3. The app is actually slow

If all of this are true is the time to optimize the context

Once the app re-renders and the context provider is the child of app, then the context provider will re-render. From there if the provider render, then means the value object will also be re created, and js compare that object and previous and see there are different objects, then all components that subscribe to that context will also be re-render. This is the wrong practice then.

The solution for this is to memorize the object

If you have function in your context provider and asked to use it in useMemo in dependency array then you need to wrap that function into useCallback hook, other wise it will cause infinite re-render, because once the context re-render also the function will re-render then the contextProvider also will re-render

BUNDLE AND CODE SPLITTING

Bundle is JavaScript file containing the entire application code. This file need to be downloaded before use start to anything. This optimization deal with take less time do download the bundle.

We use technique called code splitting: splitting bundle into multiple parts that can be downloaded over time(lazy loading). The most common way is to split the bundle at routes level by taking the component of the page and load it separately

Does and don’ts :

1. Don’t optimize prematurely : don’t optimize anything if there is nothing to optimize. If your app is running fine don’t wrap your function into memo, useMemo or useCallbacks
2. Use profiler and visual inspection for performance
3. Implement cod splitting + lazy loading for SPA routes.

SECTION 20

REDUX

Redux Is 3rd party library to manage global state. Its stand alone, then it allow it to be used in any library and can even be used in vanilla JavaScript

All global state is stored in one globally accessible store which is easy to update using actions (like useReducer)

WHEN TO USE REDUX

Used in global management systems (its some how similar to combination of context + useReducer)

Use global management library like redux only if there are a lot global us state that need to update frequently

We need to packages which are “REDUX and REACT-REDUX”

REDUX COMES WITH DEVELOPER TOOLS LIKE REACT(reactdevTool). We open chrom as usual and then we install it.

And we install corresponding package. Which is NPM I REDUX-DEVTOOLS-EXTENSION

**THEN LET JUMP TO THE MODELM WAY OF WRTING REDUX**

By using redux toolkit. It allow us to write less code compare to classic redux.

To use it we start by installing @reduxjs/toolkit (NPM I @reduxjs/toolkit)

Then into our store we only need to import configureStore. Then in the config store we insert Object like {reducer:{reducer1,reducer2,reducer3,…..}} and it should be sotred in variable like store and exported. In older to access all dispatched values.

In slice file we have to import createSlice method it have three benefit

1. Automatically create action creators form reducers.
2. Make writing reducer a lot easy
3. Mutate state inside our reducers.

**How to plan and build a react project**

From thinking in react we see that to create react app we start on break the desired UI into components, build a static version (no state yet) and then thing about state management + data flow

In large project we start with these stages

1. Gether application requirements and features
2. Divide the application into pages
   1. Think about the overall and page-level UI
   2. Break the desired UI into components
   3. Design and build a static version (no state Yet)
3. Divide the application into feature categories
   1. Think about state management + data flow
4. Decide on what libraries to use (technology decisions)

Then let we start one by one.

**1st step**

Project requirements from the business.

* Application, where users can order one or more pizzas from a menu.
* Requires no user accounts and no login: users just input their names before using the app
* The pizza menu can change, so it should be loaded from an API
* Users can add multiple pizzas to a cart before ordering
* Ordering requires just the user name, phone number, and address
* If possible GPS location should also be provided to make delivery easier.
* Users can mark their order as priority for an additional 20% of the cart price
* Orders are made by sending a POST request with the order data (user data + selected pizzas) to the API
* Payments are made on delivery, so no payment processing is necessary in the app.
* Each order will get a unique id that should be displayed, so the user can later look up their order based on id
* User should be able to mark their order as priority order even after it has been placed.

**2nd step**

From the above info we do know exactly what we are going to design or build. Then we need to distribute this into features.

1. User: everything related to the user
2. Menu: everything related to the menu items
3. Cart: to older pizzas.
4. Order: placing new order, looking ordered pizzas.

From that we can think of the pages we need.

1. Home page /
2. Pizza menu /menu
3. Cart /cart
4. Placing a new order /order/new
5. Looking up an order /order/:id

**3rd step**

State management and data flow.

We will need slices for user, menu, cart and order

**4th step**

Routing: react Router

Styling: tailwind CSS

Ui state management: redux

Remote state management: react Router. But preferred way is react query

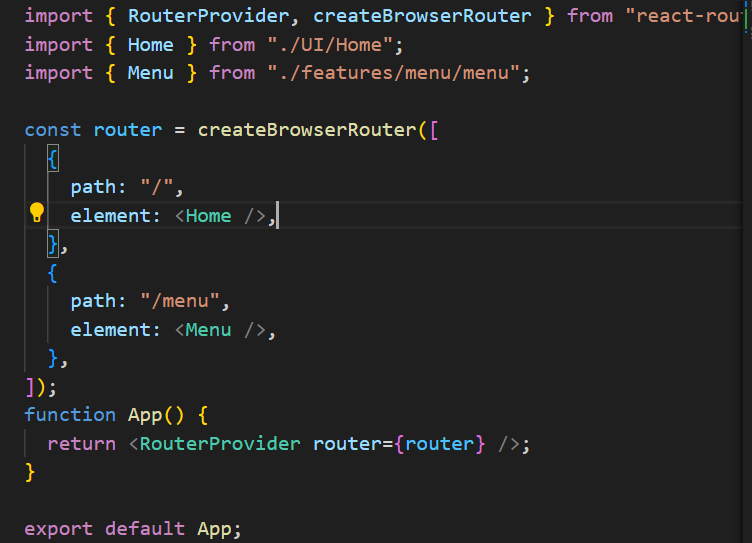
We create features files which contain the features like menu

Then we create services folder: is for reusable code that interact with API.

Utils folder: used to create helper function without side effects.

Ui folder : for re-usable components like buttons and so on.

Almost we put homepage and error Page in UI folder because they are not belong in any features category.

React version 6.4 introduce new way of defining routes and working with react router. Now we can use powerful mechanism inside react-router, for fetching data into pages and submitting data using forms.

This is the new way to allow data fetching and form submission

SECTION 24

We only need to dependencies one is @reduxjs/toolkit to interact with redux and react-redux to connect our redux with our react app.