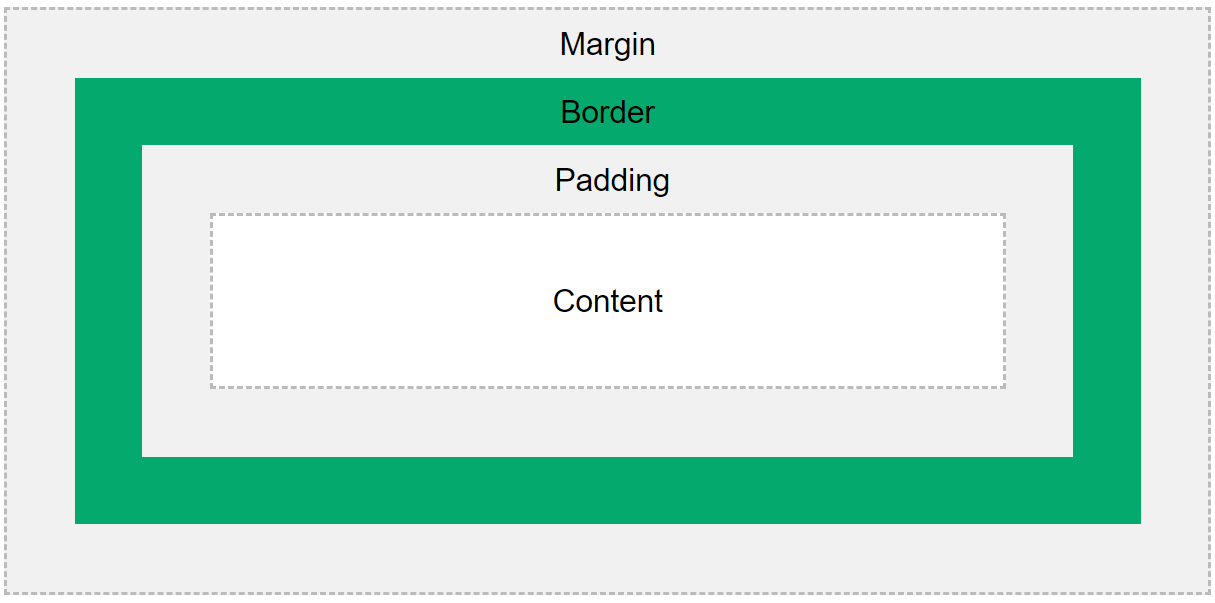
a) BOX MODEL



Padding - To add background to an element around the content

Margin – To have space between two elements apart from each other..margin is collages for two elements . If one has margin of 60 and another has margin of 60 .The space between these two is still 60.

Content Has height and width property:

To specify the content height and width

Box Sizing:

In default box size is addition of height,padding, border,margin size..To make the box size the size of height and width of the content has to use

box-sizing: border-box;

b)Arrow Functions

reactDOM renders the app component on the root DOM node

App component contains html which displays in browser

NORMAL FUNCTIONS:

const add=function(num1+num2)

{

return num1+num2;

}

console.log(add(5,6))

ARROW FUNCTION:

In arrow function no need to write function keyword to represent part of function use arrow keyword

const add=(num1,num2)=>{

return num1+num2;

}

if arrow function has single statement

const add=(num1,num2)=>return num1+num2;

1){ }-> for dynamic values

2)React cant output objects,booleans

3)react outputs all as string

4)component is just a function return jsx template and exported at bottom of a file

5)index.css is a global style sheet can be applied to all components...imported in index.js it adds all the styls in inspect--->elements...inside the

<style> ----------- </style>

6)inline styling in react.......

<a href="/create" style={

{

color:"white",

backgroundColor:'#f1356d',

borderRadius:'8px'

}

}>New Blog</a>

1 st bracket for dynamic value

2 nd bracket for object value pair

7)click events

<button onclick={handleClick}> Click</button> has to pass the reference

<button onclick={handleClick()}> Click</button>... This will invoke the function without clicking the button

For passing arguments to the onClick function has to pass as annonymous function otherwise it will invoke the function without clicking

CODE:

*const* Home = () *=>* {

*const* handleClick = () *=>* {

        console.log("Hello");

    }

*const* handleclickAgain = (*name*) *=>* {

        console.log("Hello" + *name*);

    }

    return(

        <div className="Home">

            <h2>Home Page</h2>

            <button onClick={handleClick}> Click</button>

            <button onClick={

                () *=>* {

                    handleclickAgain("Akil")

                }

            }></button>

        </div>

    );

}

export default Home;

8)Event parameter/object

1 st parameter .We have automatically have access to this when an event occurs

CODE:

*const* Home = () *=>* {

*const* handleClick = (*e*) *=>* {

        console.log("Hello",*e*);

    }

*const* handleclickAgain = (*name*,*e*) *=>* {

        console.log("Hello" + *name* , *e*.target);

    }

    return(

        <div className="Home">

            <h2>Home Page</h2>

            <button onClick={handleClick}> Click</button>

            <button onClick={

                (*e*) *=>* {

                    handleclickAgain("Akil", *e*)

                }

            }></button>

        </div>

    );

}

export default Home;

9)State

Data used by the component at that point.Data can be arrays,booleans,strings,objects

Eg:

*const* Home = () *=>* {

*let* name="akil";

*const* handleClick = () *=>* {

        name="sundhar";

        console.log(name);

    }

    return(

        <div className="Home">

            <h2>Home Page</h2>

            <p>{name}</p>

            <button onClick={handleClick}> Click</button>

        </div>

    );

}

export default Home;

The name isn’t reactive … That is in the template(browser) the updation of sundhar has not taken place ..but in console name is updated to sundhar.To make the change visible to browser have to use USESTATE hook

import { useState } from 'react';

*const* Home = () *=>* {

*const* [name, setName]=useState("akilan");

*const* handleClick = () *=>* {

        setName("sundhar");

        console.log(name);

    }

    return(

        <div className="Home">

            <h2>Home Page</h2>

            <p>{name}</p>

            <button onClick={handleClick}> Click</button>

        </div>

    );

}

export default Home;

10)Map method  
 Map method cycles through an array of each item ..Map fires a callback function for each item .At each time have to return a jsx

When output a array using map method it has key property react uses to keep track of each item in the dom.key should be unique for each item

CODE:

import { useState } from 'react';

*const* Home = () *=>* {

*const* [blogs,setBlogs]=useState([

        {title:"c",body:"C programming",author:"ABC",id:1},

        {title:"c++",body:"C++ programming",author:"DEF",id:2},

        {title:"c#",body:"C# programming",author:"GHI",id:3}

    ]);

    return(

        <div className="Home">

           {blogs.map((*blog*) *=>* (

            <div className='Blog-preview' key={*blog*.id}>

                <h2>{*blog*.title}</h2>

                <p>written by {*blog*.author}</p>

            </div>

           ))}

        </div>

    );

}

export default Home;

11)props

The blogs has to be displayed in varous pages like home page,search page….

Used to transfer the data from parent component to child component.Parent values passed will be in props object.

<div className="Home">

           <*Bloglist* blogs={blogs} title="All Blogs" />

 </div>

*const* Bloglist = (*props*) *=>* {

*const* blogs=*props*.blogs;

*const* title=*props*.title;

In the props it will be there ….

We can destructure the props in the parenthesis itself

*const* Bloglist = ({*blogs*,*title*}) *=>* {

12)Reusing Componenets(FILTER METHOD)

By using filter method..filter() method fires a callback function for every element in an array..By this method the Bloglist component is reused

CODE:

<div className="Home">

           <*Bloglist* blogs={blogs} title="All Blogs" />

           <*Bloglist* blogs={blogs.filter( (*blog*) *=>* *blog*.author==="ABC")} title="ABC Blogs"/>

</div>

14)Passing Function into the components (Passing function as props)

* + For deleting the data write the function for Deletion At where it is declared…
  + Delete button declared in Bloglist component..Blogs data declared in Home component ..
  + Delete Button handling function has to be declared in Home component as HandleDelete()…
    - Now pass this HandleDelete() function to Bloglist component as props(parameter)…Now the Bloglist component have access to HandleDelete() function in Home component

CODE:

Home.js

import { useState } from 'react';

import Bloglist from './Bloglist';

*const* Home = () *=>* {

*const* [blogs,setBlogs]=useState([

        {title:"c",body:"C programming",author:"ABC",id:1},

        {title:"c++",body:"C++ programming",author:"DEF",id:2},

        {title:"c#",body:"C# programming",author:"ABC",id:3}

    ]);

*const* HandleDelete = (*id*) *=>* {

*const* newblogs=blogs.filter( (*blog*) *=>* *blog*.id!==*id* );

        setBlogs(newblogs);

    }

    return(

        <div className="Home">

           <*Bloglist* blogs={blogs} title="All Blogs" HandleDelete={ HandleDelete }/>

           <*Bloglist* blogs={blogs.filter( (*blog*) *=>* *blog*.author==="ABC")} title="ABC Blogs"/>

        </div>

    );

}

export default Home;

Bloglist.js

*const* Bloglist = ({*blogs*,*title*,*HandleDelete*}) *=>* {

    // const blogs=props.blogs;

    // const title=props.title;

    return (

        <div className="Bloglist">

            <h1>{*title*}</h1>

            {*blogs*.map((*blog*) *=>* (

            <div className='Blog-preview' key={*blog*.id}>

                <h2>{*blog*.title}</h2>

                <p>written by {*blog*.author}</p>

                <button onClick={ () *=>* HandleDelete(*blog*.id) }>DELETE</button>

            </div>

           ))}

        </div>

    )

}

export default Bloglist;

13)useEffect Hook

This hook runs a function every render of the component.The component renders initially when it first loads and renders into DOM and also happens when the state changes it re renders the DOM So we can update the state in the browser

useEffect doesn’t return anything.Pass the argument as a function.This is the function going to run every time it re renders(Any time data changes).

Normally inside useEffect is used to fetch data , to communicate with authentication

Services.(This all known as side effects in React).

The useEffect function runs for every render(like any change in data)

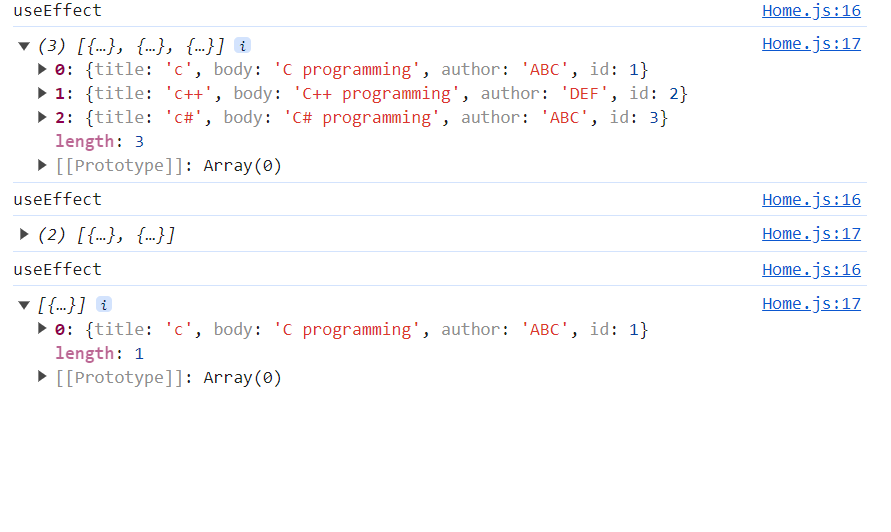
Initially the component renders to the DOM that will trigger the useEffect function to run,that will update the state and the state will change that would trigger re render..this re render will trigger the useEffect function..This creates a continous loop.

 useEffect( () *=>* {

        console.log("useEffect");

        console.log(blogs);

    });



For every state change it re renders

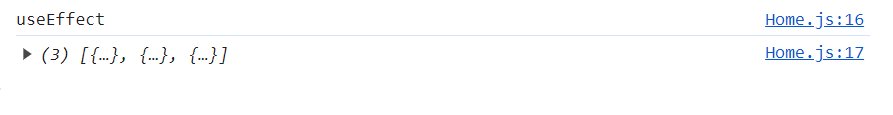
🡺To overcome this we use dependency array .It is passed as an 2nd argument .This will make the funtion runs only once on the first render.

useEffect( () *=>* {

        console.log("useEffect");

        console.log(blogs);

    },[]);



Even after deleting the blog(changing the state) the re render doesn’t occur

13 ) a) Adding dependencies to the Array

* To control the useEffect function when to run
* Any state value that would trigger the useEffect function to run when they change.
* Now want to run the useEffect function only when the state(name) changes.Not the state(blogs) when the state(blogs) changes not need to run useEffect.
* So the name become dependency …have to add the name state in the array

import { useState,useEffect } from 'react';

import Bloglist from './Bloglist';

*const* Home = () *=>* {

*const* [blogs,setBlogs]=useState([

        {title:"c",body:"C programming",author:"ABC",id:1},

        {title:"c++",body:"C++ programming",author:"DEF",id:2},

        {title:"c#",body:"C# programming",author:"ABC",id:3}

    ]);

*const* [name,setName]=useState("Akilan");

*const* HandleDelete = (*id*) *=>* {

*const* newblogs=blogs.filter( (*blog*) *=>* *blog*.id!==*id* );

        setBlogs(newblogs);

    }

    useEffect( () *=>* {

        console.log("useEffect");

        console.log(name);

    },[name]);

    return(

        <div className="Home">

           <*Bloglist* blogs={blogs} title="All Blogs" HandleDelete={ HandleDelete }/>

           <*Bloglist* blogs={blogs.filter( (*blog*) *=>* *blog*.author==="ABC")} title="ABC Blogs"/>

           <button onClick={ () *=>* setName("Raju") }>Change Name</button>

           <p>{name}</p>

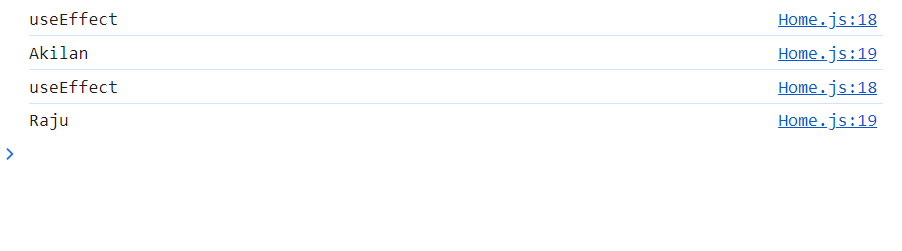
        </div>

    );

}

export default Home;

Now useEffect will watch the state value/(name) if it change it will run the useEffect function.



After given the DELETE for many time the useEffect didn’t triggered ..It triggers at first time and name state changes.

14)JSON Server

Used to create a domain local REST API to mimic backend.

useEffect used to fetch data from component/database using API Endpoint(REST API) .We can use that data in our application instead of hard coded data .

JSON Server allow us to build fake REST API by using JSON file

Db.json

{

*"blogs"*:[

        {

*"title"*:"Akilan Biography",

*"body"*:"B.Tech IT Final Year",

*"author"*:"Akilan",

*"id"*:1

        },

        {

*"title"*:"Sundhar Biography",

*"body"*:"BSc CSC 2nd  Year",

*"author"*:"Sundhar",

*"id"*:2

        }

    ]

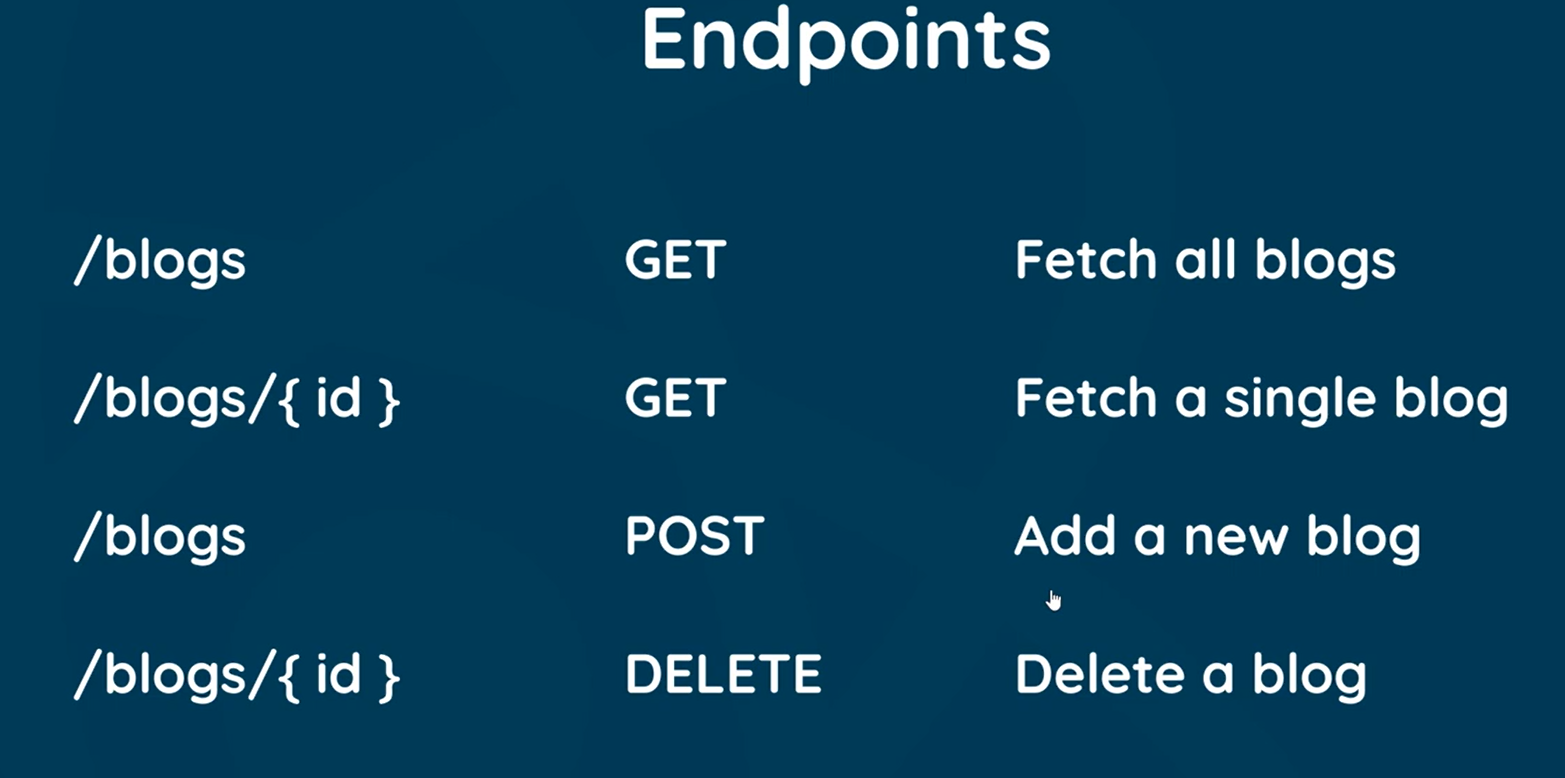
}

It has one property called blogs which has array of two objects.Each object is a blog which has title,body,author,id.When using JSON Server each top level property known as RESOURCE.

db.json has one top level property as blogs.It sees this as a resource and creates a endpoint to interact with this resource to delete,add,get items

JSON Server package watch db.json file and wrap with this Endpoint. The JSON Server Package is => npx json-server --watch data/db.json --port 8000

The <http://localhost:8000/blogs> is the endpoint.



This are the different endpoints JSON Server provides

15)GET Endpoint(Fetching data with useEffect)

Now the JSON Server is running and watching data/db.json.From the db.json we fetch the data and update the state blogs.Initially make the state blogs to null.Now we fetch the data from db.json using useEffect and update the state blogs using setblogs(useState).

fetch(<http://localhost:8000/blogs>) this will return a promise. .then() will fire a function once the promise is resolved. At 1st we get a response (res) object .To get the data from the object have to use res.json() . This( res.json() ) passess the json into javascript object. We have to return this .After returning (return res.json() )

fetch('http://localhost:8000/blogs')

            .then( *res* *=>* {

                return *res*.json()

            })

This whole thing(fetch) will return another promise because this is asynchronous this will take some time to do. Once this promise complete, .then() function will take the parameter of actual data(data)

.then(*data* *=>* {

console.log(*data*);

            })

In Home.js

{ blogs && <*Bloglist* blogs={blogs} title="All Blogs" HandleDelete={ HandleDelete }/> }

Once if the blogs are set have to call the component.Because Initially the blogs state is set to null.So we pass this null to bloglist at initial render.

16)Conditional loading  
 our fetch is very quick because fetching is on our own computer .But most times fetch are done over the internet it will be slower.In that case we will see a loading message.

Create a state ispending and set it to true.

Have to show the (Loading) while the fetch is going.Once have the data set the isPending state to false.

To see the Loading… , we will use setTimeout() this will fire fetch API function after a given 1000 ms time.

useEffect( () *=>* {

        setTimeout( () *=>* {

            fetch('http://localhost:8000/blogs')

            .then( *res* *=>* {

                return *res*.json()

            })

            .then(*data* *=>* {

                console.log(*data*);

                console.log("data");

                setBlogs(*data*);

                setisPending(false);

            });

        },1000);

    },[]);

17)Handling fetch Errors

Error that send from server (or) connection error where we can’t connect to the server. catch block will take care of this error. This catch block will capture any kind of network error and fires a function

useEffect( () *=>* {

        setTimeout( () *=>* {

            fetch('http://localhost:8000/blogs')

            .then( *res* *=>* {

                return *res*.json()

            })

            .then(*data* *=>* {

                console.log(*data*);

                console.log("data");

                setBlogs(*data*);

                setisPending(false);

            })

            .catch( *err* *=>* {

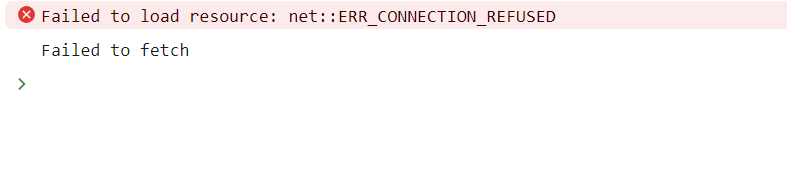
                console.log(*err*.message);

            })

        },1000);

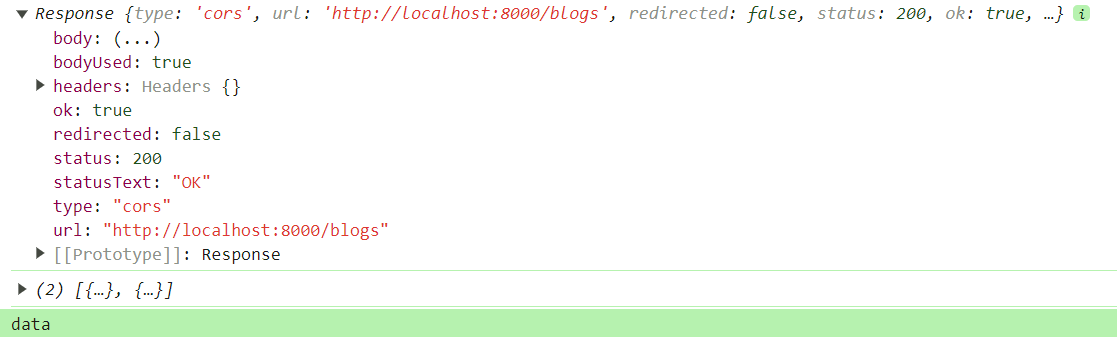
    },[]);

When we terminate the json server we will be getting a error message like this



The other kind of error like the request reaches the server , the server sends the error back ( like the endpoint doesn’t exist (or) the request id denied. In this case the catch block doesn’t automatically catch the error when we use fetch API.

The request reaches the server and the server still sends the response(res) object . Have to check the res object when we get it back.



The ok property is true. If suppose the endpoint(<http://localhost:8000/blogs>) doesn’t exist this will be false. So have to check this ok property . if res.ok is not true have to throw the error

If we change the endpoint like <http://localhost:8000/blogs> to <http://localhost:8000/data> we can see the error



useEffect( () *=>* {

        setTimeout( () *=>* {

            fetch('http://localhost:8000/data')

            .then( *res* *=>* {

                if(!*res*.ok)

                {

                    throw *Error*('Could not fetch the data for that resourse');

                }

                console.log(*res*);

                return *res*.json()

            })

            .then(*data* *=>* {

                console.log(*data*);

                setBlogs(*data*);

                setisPending(false);

            })

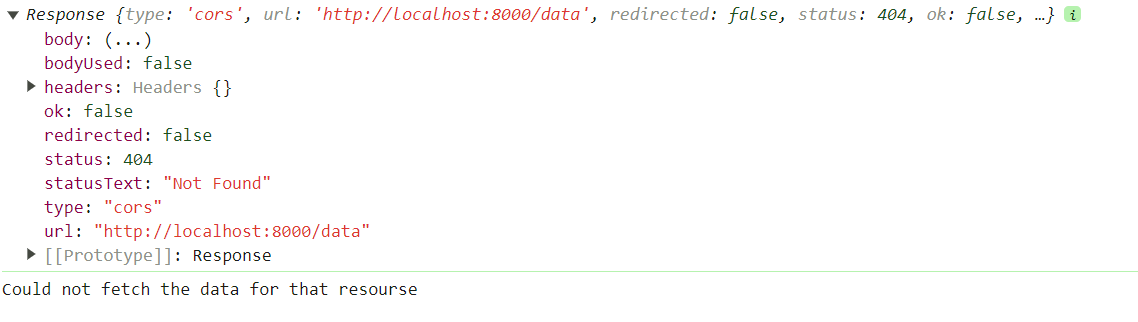
            .catch( *err* *=>* {

                console.log(*err*.message);

            })

        },1000);

    },[]);



We can see the ok property is also false.

17)

a) Storing the error and outputing in the browser(using state)

import { useState,useEffect } from 'react';

import Bloglist from './Bloglist';

*const* Home = () *=>* {

*const* [blogs,setBlogs]=useState(null);

*const* [name,setName]=useState("Akilan");

*const* [error,setError]=useState(null);

*const* [isPending,setisPending]=useState(true);

*const* HandleDelete = (*id*) *=>* {

*const* newblogs=blogs.filter( (*blog*) *=>* *blog*.id!==*id* );

        setBlogs(newblogs);

    }

    useEffect( () *=>* {

        setTimeout( () *=>* {

            fetch('http://localhost:8000/data')

            .then( *res* *=>* {

                console.log(*res*);

                if(!*res*.ok)

                {

                    throw *Error*('Could not fetch the data for that resourse');

                }

                console.log(*res*);

                return *res*.json()

            })

            .then(*data* *=>* {

                console.log(*data*);

                setBlogs(*data*);

                setisPending(false);

            })

            .catch( *err* *=>* {

                setError(*err*.message);

                console.log(*err*.message);

            })

        },1000);

    },[]);

    return(

        <div className="Home">

            { isPending && <h4>Loading....</h4> }

            { error && <h4>{ error }</h4> }

           { blogs && <*Bloglist* blogs={blogs} title="All Blogs" HandleDelete={ HandleDelete }/> }

           {/\* <Bloglist blogs={blogs.filter( (blog) => blog.author==="ABC")} title="ABC Blogs"/>

           <button onClick={ () => setName("Raju") }>Change Name</button> \*/}

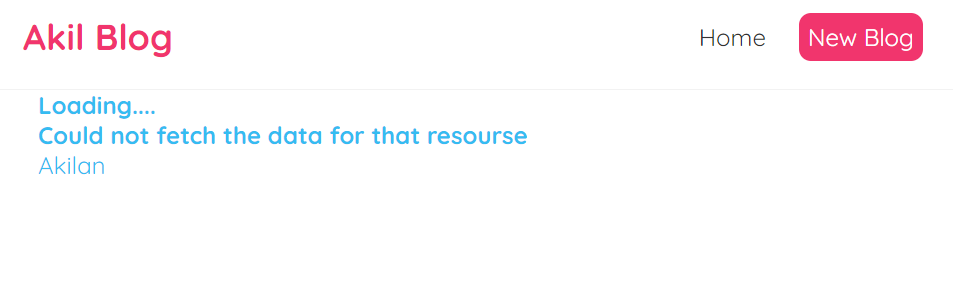
           <p>{name}</p>

        </div>

    );

}

export default Home;



We can see the loading.. . If there is an error no need for displaying loading………

If we get the error make the isPending state to false.

If we get the data no need to display the error message ..make the error state to null

Hence we have done with all the errors.

import { useState,useEffect } from 'react';

import Bloglist from './Bloglist';

*const* Home = () *=>* {

*const* [blogs,setBlogs]=useState(null);

*const* [name,setName]=useState("Akilan");

*const* [error,setError]=useState(null);

*const* [isPending,setisPending]=useState(true);

*const* HandleDelete = (*id*) *=>* {

*const* newblogs=blogs.filter( (*blog*) *=>* *blog*.id!==*id* );

        setBlogs(newblogs);

    }

    useEffect( () *=>* {

        setTimeout( () *=>* {

            fetch('http://localhost:8000/blogs')

            .then( *res* *=>* {

                console.log(*res*);

                if(!*res*.ok)

                {

                    throw *Error*('Could not fetch the data for that resourse');

                }

                console.log(*res*);

                return *res*.json()

            })

            .then(*data* *=>* {

                console.log(*data*);

                setBlogs(*data*);

                setisPending(false);

                setError(null);

            })

            .catch( *err* *=>* {

                setError(*err*.message);

                setisPending(false);

                console.log(*err*.message);

            })

        },1000);

    },[]);

    return(

        <div className="Home">

            { isPending && <h4>Loading....</h4> }

            { error && <h4>{ error }</h4> }

           { blogs && <*Bloglist* blogs={blogs} title="All Blogs" HandleDelete={ HandleDelete }/> }

           {/\* <Bloglist blogs={blogs.filter( (blog) => blog.author==="ABC")} title="ABC Blogs"/>

           <button onClick={ () => setName("Raju") }>Change Name</button> \*/}

           <p>{name}</p>

        </div>

    );

}

export default Home;

18)Making a custom hook

*const* [blogs,setBlogs]=useState(null);

*const* [error,setError]=useState(null);

*const* [isPending,setisPending]=useState(true);

All the state are working good . If we want to use this state in other component in future where we fetch some data where all these states are needed has to re-write the code in that component again.

If we are using this code in different places in the application we would write this code in different component it will become re-usable again and again.

So rip all of this component and put it in own javascript file and can be import it in any component.

In order to make this custom hook create a js file.Custom hook has to start with use.

Example---------> useFetch.js

Put the logic of useEffect in useFetch component . By usng this useFetch we can display any kind of data . So make the state blogs to data for convinient .

Need to return some values for this function like when we call some kind of hook like useState it returns some value right.

So for the custom hook useFetch returning an object (inside the object has three values) .

We can return string,boolean,array anything.

I am returning an object with three properties(data,ispending,error).These three are the states used by useFetch hook.

Atlast have to pass the endpont (or) url into the function instead of hard coding.As we not going to use the same endpoint in whole application and have to make the url as dependency in useEffect dependency array.Whenever the url changes it going to re-run the function to get the data from the url(new endpoint).

Now import this useFetch function inside the Home component to use it. Have to de-structure the three properties we get from useFetch. For the useFetch pass the endpoint for fetching data.

import { useState,useEffect } from 'react';

import Bloglist from './Bloglist';

import useFetch from './usefetch';

*const* Home = () *=>* {

*const* { data,isPending,error }= useFetch('http://localhost:8000/blogs');

    return(

        <div className="Home">

            { isPending && <h4>Loading....</h4> }

            { error && <h4>{ error }</h4> }

            { blogs && <*Bloglist* blogs={blogs} title="All Blogs" />}

        </div>

    );

}

export default Home;

Now the de-structured one is data but in Bloglist component passings as blogs. So change to data.

{ blogs && <*Bloglist* blogs={blogs} title="All Blogs" /> }

So in de-structure by colon(:) we can add whatever name we want…..

import { useState,useEffect } from 'react';

import Bloglist from './Bloglist';

import useFetch from './useFetch';

*const* Home = () *=>* {

*const* { data:blogs ,isPending,error }= useFetch('http://localhost:8000/blogs');

    return(

        <div className="Home">

            { isPending && <h4>Loading....</h4> }

            { error && <h4>{ error }</h4> }

            { blogs && <*Bloglist* blogs={blogs} title="All Blogs" /> }

        </div>

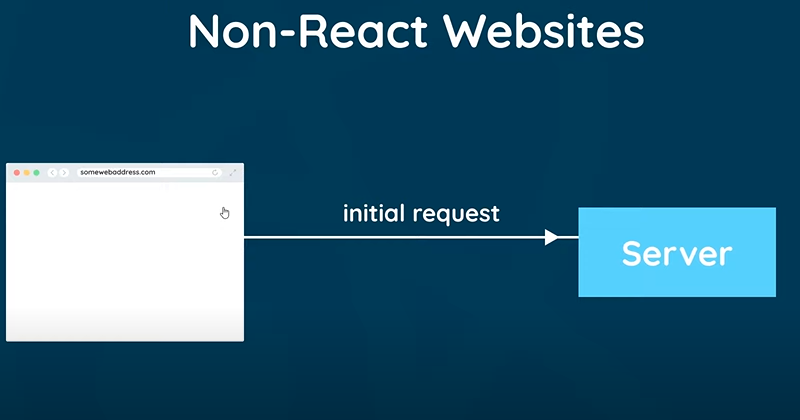
    );

}

export default Home;

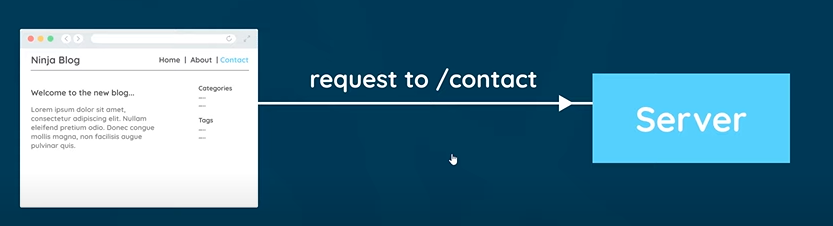
19) React Router

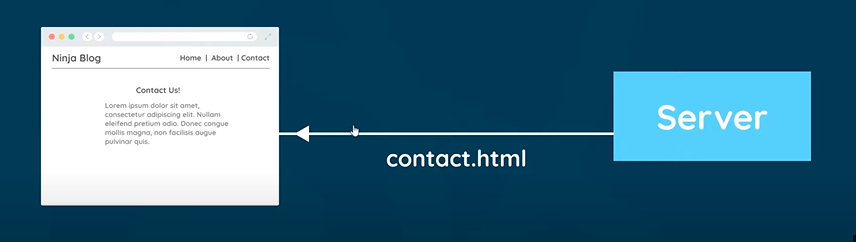
Upto now we have only single page.To introduce multiple different pages in react has to use route.In typical multiple page website , In browser first we type a URL in address bar that sends request to server . Server sends the full HTML page which we view in the browser.





If user again clicks the Contact , it then again sends a brand new request to server , server sends a new contact HTML page, then we can view in the browser

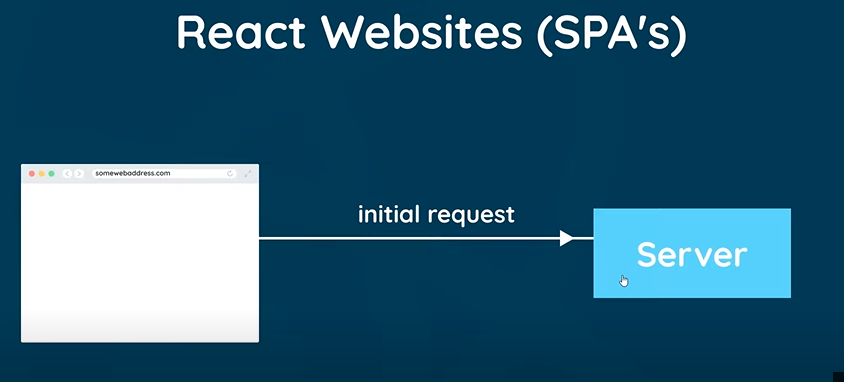


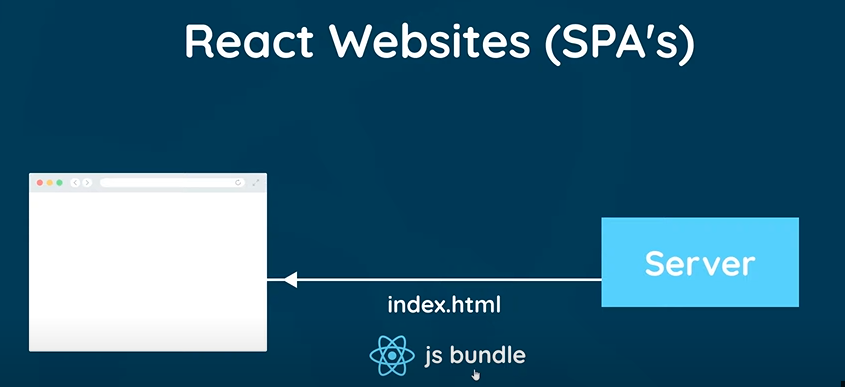


This cycle will continue over and over again as clicking other pages in the website.So we constantly making requests to server for new pages.

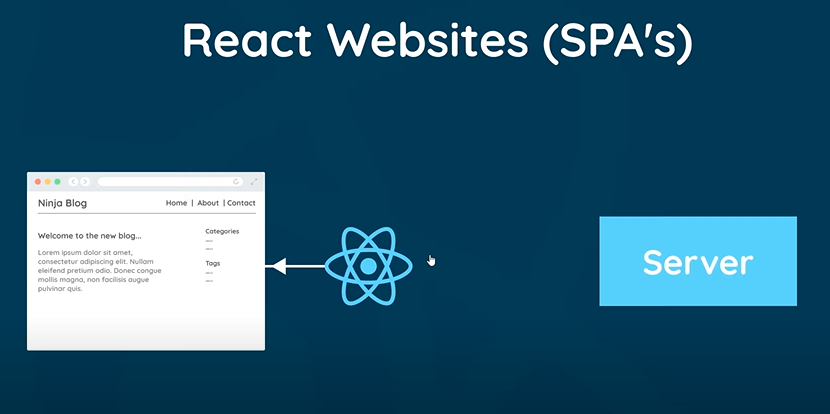
React application don’t work like this ..they deligate all the routing and change of page content to browser.

In react …. I browser first we type a URL in address bar that sends request to server . Server sends the full HTML page and also compiled react javascript files which controls react application.



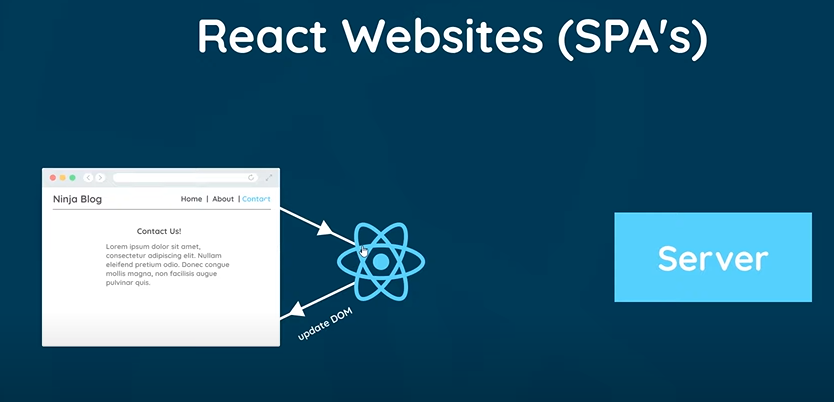


Intitially the html page we get back will be virtually empty …..react injects the content dynamically using the component …..In the above image it is virtually empty…

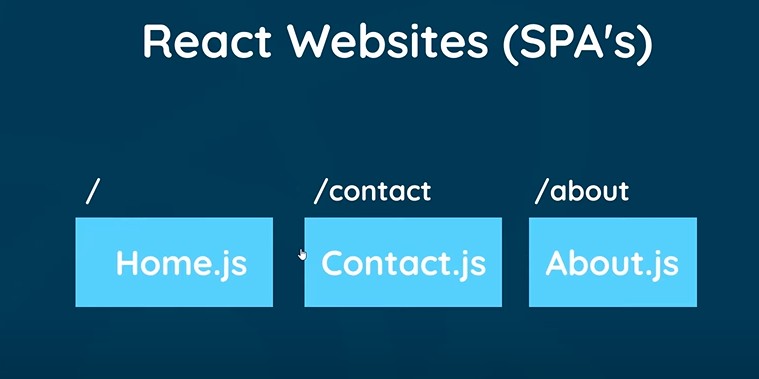


This is the page after react injects the content dynamically using the component….

If we click on a Contact link ..react router will tell the react to inject Contact component to browser. If we click on a About link react router will tell the react to inject About component to browser.



The request wont go to the server .React router intercepts the request and stops.



1)Install react router package ..( npm install react-router-dom@5)

2)

In root component (app.js)

import { BrowserRouter as Router,Route,Switch } from 'react-router-dom';

3)Now surround the entire application with router component

import Navv from './Navv';

import Home from './Home';

import { BrowserRouter as Router,Route,Switch } from 'react-router-dom';

*function* App() {

  return (

    <*Router*>

      <div className="App">

        <*Navv* />

        <div className="Content">

          <*Home* />

        </div>

      </div>

      </*Router*>

  );

}

export default App;

4)Now have to decide where the page content to go when we go to different pages.I choose to go to the div of class content

The <Switch> </Switch> component makes sures only one route shows at any time.All the route goes to Switch component.At now we have only one route (Home page)

The route Component has one property called path .The path is basically the route . For home page the path is forward slash(“/”).If it is about page the path is forward slash(“/about”).

Now nest the component(Home component) inside the route that we want to show when user visits .

Atlast I want to show the Home component inside the div of class content when visit with just forward slash(“/”).The navbar is always going to show because it is not inside Switch component the navbar will be there for every single route it does not matter what route it is.

Content inside the Switch is going to change depend on the route as when we build up more routes.

20)Exact Match Routes

To add another route need another page component(form for adding blog). Create a new component as Create.js

Now need to show the component when user goes to “/Create”

Create.js

*const* Create = () *=>*

{

    return(

        <div className="Create">

            <h2>Add a new blog</h2>

        </div>

    );

}

export default Create;

App.js

import Navv from './Navv';

import Home from './Home';

import Create from './Create';

import { BrowserRouter as Router,Route,Switch } from 'react-router-dom';

*function* App() {

  return (

    <*Router*>

      <div className="App">

        <*Navv* />

        <div className="Content">

          <*Switch*>

            <*Route* path="/">

              <*Home*/>

            </*Route*>

            <*Route* path="/Create">

              <*Create*/>

            </*Route*>

          </*Switch*>

        </div>

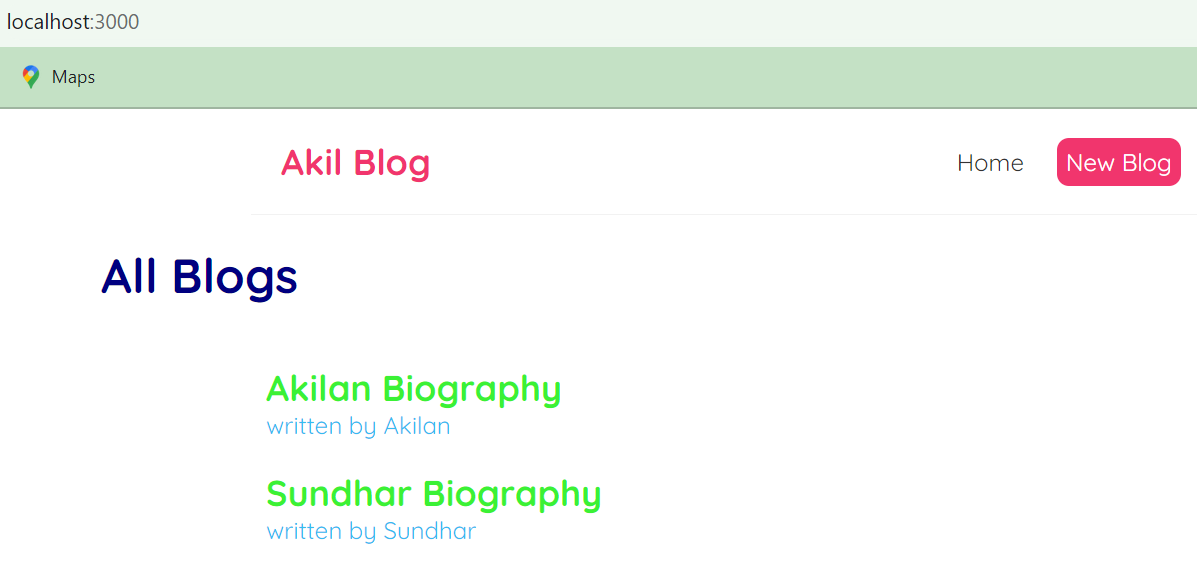
      </div>

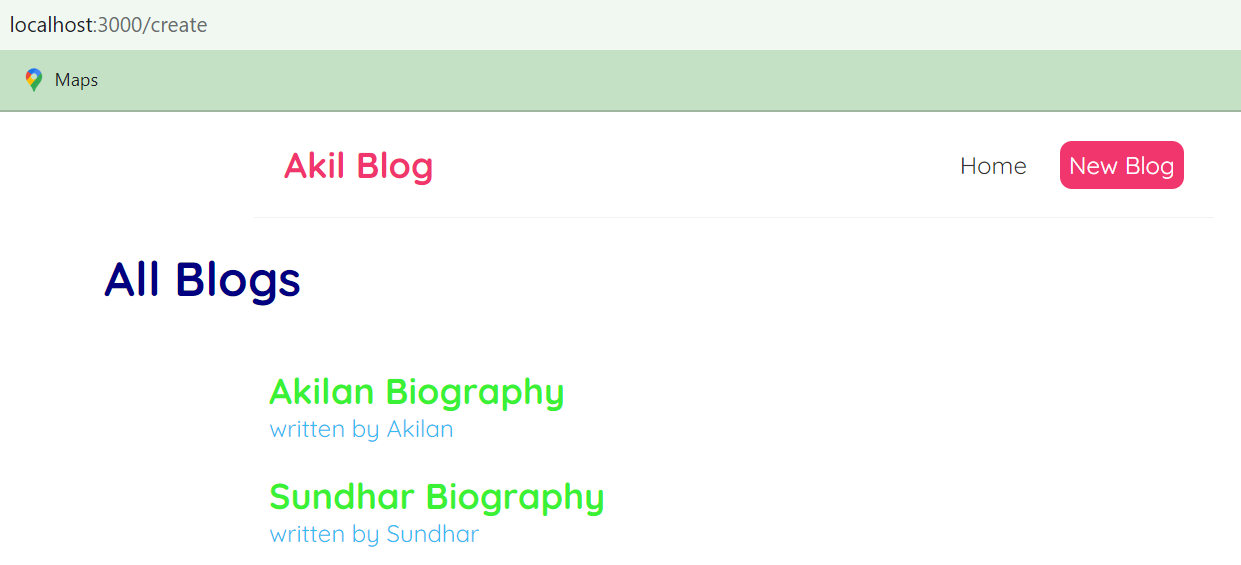
      </*Router*>

  );

}

export default App;





For “/Create” route also it shows the content of Home component. Because of …..when we type “/create” react look up the route paths (“/” and “/create”) and match them against our route(“/create”) .. React finds out first one to be a correct match(“/”) and stops there, react go from top to bottom in the <Switch> and finds the 1 st match and render that component not the exact match.

React okay with the partial match . So to avois this use (exact) in the <route> .

App.js

import Navv from './Navv';

import Home from './Home';

import Create from './Create';

import { BrowserRouter as Router,Route,Switch } from 'react-router-dom';

*function* App() {

  return (

    <*Router*>

      <div className="App">

        <*Navv* />

        <div className="Content">

          <*Switch*>

            <*Route* exact  path="/">

              <*Home*/>

            </*Route*>

            <*Route* path="/Create">

              <*Create*/>

            </*Route*>

          </*Switch*>

        </div>

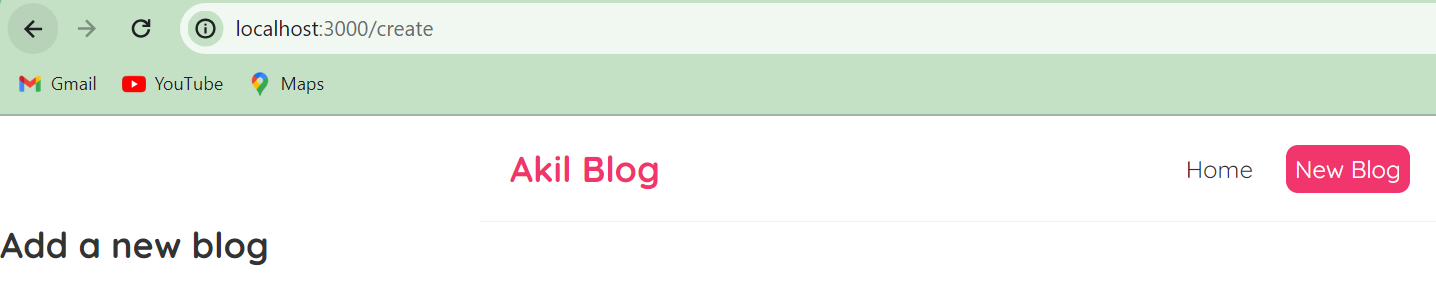
      </div>

      </*Router*>

  );

}

export default App;



App.js

<*Switch*>

            <*Route* exact  path="/">

              <*Home*/>

            </*Route*>

            <*Route* path="/Create">

              <*Create*/>

            </*Route*>

</*Switch*>

21)Router Links

We set up different routes “/” and “/create” . When clicked on links Home and New Blog which declared in navv bar browser sends a fresh request to server at each time and returns a same HTML page then the react inject the correct content.

But the react router can intercept the request and can handle the content change in browser itself.

To to this need to use the special link tag

Step 1: Go to navv.js

Step 2: import { Link } from 'react-router-dom';

Step 3: Change the Anchor tag to Link tag

<a href="/">Home</a>

                <a href="/create" style={

                    {

                        color:"white",

                        backgroundColor:'#f1356d',

                        borderRadius:'8px'

                    }

                }>New Blog</a>

href is changed for to in Link tag.

<*Link* to="/">Home</*Link*>

                <*Link* to="/create" style={

                    {

                        color:"white",

                        backgroundColor:'#f1356d',

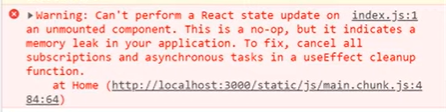
                        borderRadius:'8px'

                    }

                }>New Blog</*Link*>

22)useEffect Cleanup

In Homepage when we try to fetch the data using Custom Hook(useFetch) and try to update the state in the Home component …….while the data is trying to be fetched while it is still loading …….When we try to go to another page ( New Blog) before the fetch completes we will get error.



The unmounted component is Home component where we stopped the fetch and gone to another page.To avoid this we need to stop the fetch carrying on

It can be done using the combination of Cleanup function and abort controller.

When the component uses the useEffect/useFetch hook on mounts it fires the returned cleanup function.



At this point we need to stop the fetch going on …in the background.

import { useState,useEffect } from 'react'

*const* useFetch = (*url*) *=>* {

*const* [data,setData]=useState(null);

*const* [error,setError]=useState(null);

*const* [isPending,setisPending]=useState(true);

    useEffect( () *=>* {

        setTimeout( () *=>* {

            fetch(*url*)

            .then( *res* *=>* {

                console.log(*res*);

                if(!*res*.ok)

                {

                    throw *Error*('Could not fetch the data for that resourse');

                }

                console.log(*res*);

                return *res*.json()

            })

            .then(*data* *=>* {

                console.log(*data*);

                setData(*data*);

                setisPending(false);

                setError(null);

            })

            .catch( *err* *=>* {

                setError(*err*.message);

                setisPending(false);

                console.log(*err*.message);

            })

        },1000);

        return () *=>* console.log("cleanup");

    },[*url*]);

    return { data,isPending,error}

}

export default useFetch;

Abort Controller:

We need the associate the abort controller with specific fetch request to stop the fetch.Now link it with the return function …the cleanup function .

const abortCont = new AbortController();

Associating with fetch:

fetch(url,{ signal:abortCont.signal })

return the cleanup function:

return () => abortCont.abort();

Now the fetch will be passed when we go t the another page.

When we abort the fetch the fetch throws the error we are catching the error and updating the state of isPending and error. This means we are still trying to update the whole component

.catch( *err* *=>* {

                setError(*err*.message);

                setisPending(false);

                console.log(*err*.message);

})

So if it is a AbortError ..no need to update the state …if the error is related to network user need to know the error.

if(*err*.name=='AbortError')

                {

                    console.log("Fetch Aborted");

                }

                else{

                    setError(*err*.message);

                    setisPending(false);

                    console.log(*err*.message);

                }



import { useState,useEffect } from 'react'

*const* useFetch = (*url*) *=>* {

*const* [data,setData]=useState(null);

*const* [error,setError]=useState(null);

*const* [isPending,setisPending]=useState(true);

    useEffect( () *=>* {

*const* abortCont = new *AbortController*();

        setTimeout( () *=>* {

            fetch(*url*,{ signal:abortCont.signal })

            .then( *res* *=>* {

                console.log(*res*);

                if(!*res*.ok)

                {

                    throw *Error*('Could not fetch the data for that resourse');

                }

                console.log(*res*);

                return *res*.json()

            })

            .then(*data* *=>* {

                console.log(*data*);

                setData(*data*);

                setisPending(false);

                setError(null);

            })

            .catch( *err* *=>* {

                if(*err*.name==='AbortError')

                {

                    console.log("Fetch Aborted");

                }

                else{

                    setError(*err*.message);

                    setisPending(false);

                    console.log(*err*.message);

                }

            })

        },1000);

        return () *=>* abortCont.abort();

    },[*url*]);

    return { data,isPending,error}

}

export default useFetch;

23)Route parameters

Sometims we need to pass dynamic values as part of route.

 These 123,456,789 changeable part of the route is route parameter.

When a user visits a route “/blogs/id” need to show blogdetails component.Now create a route for Blogdetails component.

BlogDetails component:

*const* BlogDetails = () *=>* {

    return (

        <div className="BlogDetails">

            <p>Blogs</p>

        </div>

    );

}

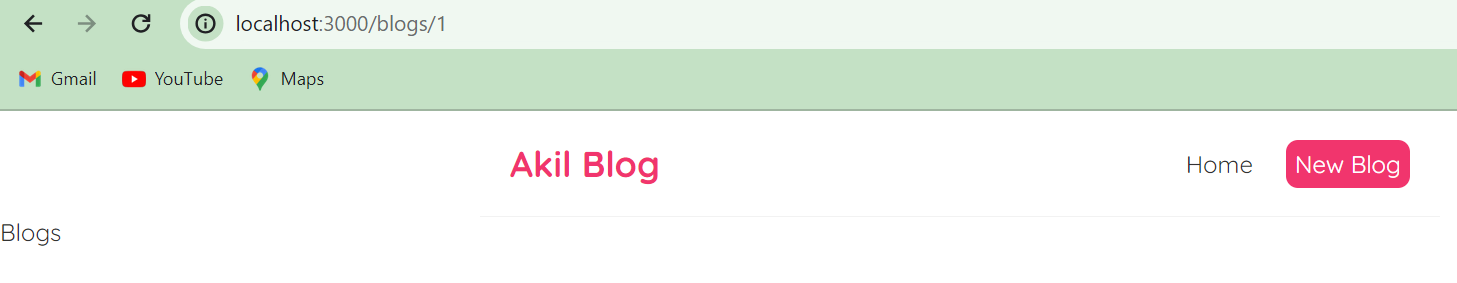
export default BlogDetails;

/blogs/---- As it is changeable we cant hard code an id have to use colon(:) then give the route parameter a name

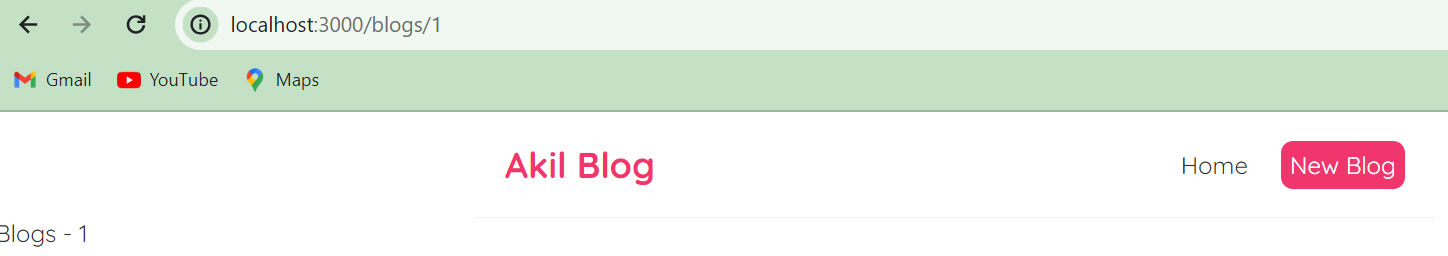
<*Route* path="/blogs/:id">

              <*BlogDetails*/>

        </*Route*>



To grab the route parameter (1) in the url(<http://localhost:3000/blogs/1>) have to use useParam hook. UseParam allows use to grab route parameters.As we named the route parameter as id while de-structuring use id



Now need to make the links in Home Component exactly Bloglist if user clicks goes to BlogDetails route then goes to id of whatever this blog is

As the is going to change for each blog we cant hard code . So using curly braces and to output the variable (template string)

import { Link } from 'react-router-dom';

*const* Bloglist = ({*blogs*,*title*}) *=>* {

    return (

        <div className="Bloglist">

            <h1>{*title*}</h1>

            {*blogs*.map((*blog*) *=>* (

            <div className='Blog-preview' key={*blog*.id}>

                <*Link* to={ `/blogs/${*blog*.id}` }>

                    <h2>{*blog*.title}</h2>

                    <p>written by {*blog*.author}</p>

                </*Link*>

            </div>

           ))}

        </div>

    )

}

export default Bloglist;

24)Reusing Custom hook

Now have to use useFetch hook in the BlogDetails component and passing the endpoint from where have to fetch.

In BlogDetails components we are fetch the data of the particular id (id which de-structured using useParams)

*const* { data:blog,error,isPending }=useFetch('http://localhost:8000/blogs'+id)

If there is blog ..we will return a <article> template consisting of title , author , body

import { useParams } from "react-router-dom";

import useFetch from "./useFetch";

*const* BlogDetails = () *=>* {

*const* { id } = useParams();

*const* { data: blog,error,isPending }=useFetch('http://localhost:8000/blogs/' + id);

    return (

        <div className="BlogDetails">

        { isPending && <div>Loading..</div> }

        { error && <div>{ error }</div> }

        { blog && (

            <article>

                <p>{ blog.title }</p>

                <p>Written By{ blog.author }</p>

                <p>{ blog.body }</p>

            </article>

        )}

        </div>

    );

}

export default BlogDetails;

25)Controlled Inputs

From the user we are going to fetch the Title,Author,Body to track these values have to maintain states

*const* [title,settitle] = useState('');

First declare with empty string ..have to associate the value of title state with forms title (input value)

<label>Blog Title:</label>

                <input

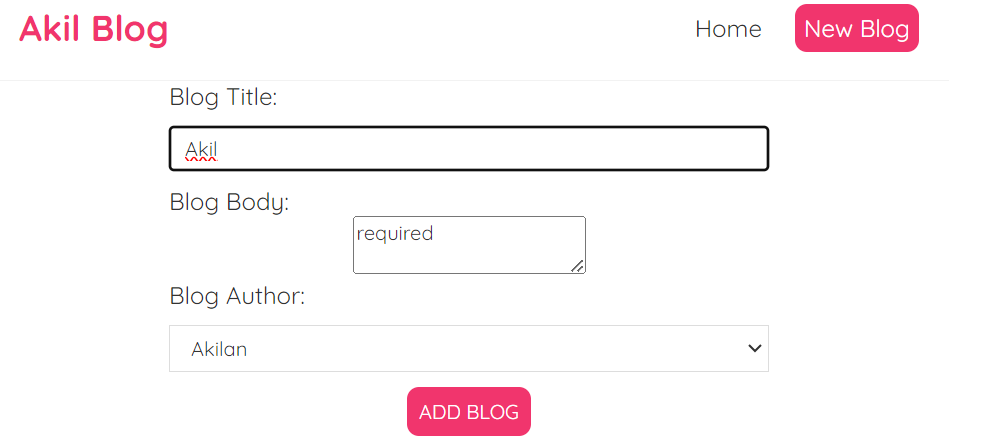
                    type="text"

                    required

                    value={title}

                />

As useState title is empty input box will also be empty if it is ‘hello’ the textbox in browser will also be hello……As it is fixed to ‘Hello’ we cant change



To update the change in textbox ..We can do by using onChange event .It is set equal to anonymous function which invokes settitle function when we try to change input value.Inside anonymous function function we have access to event object

<input

                    type="text"

                    required

                    value={title}

                    onChange={ (*e*) *=>* settitle(*e*.target.value)}

                />

When we type into input field it triggers onChange event then updation of title state takes place .

26)Submitting the form

For submitting the form there is onSubmit event . It will take function referrece as argument.Once form is submitted function will be triggered,

Once form submitted have to prevent default action of form ( content in textfield will be refreshed to empty)         *e*.preventDefault();

Now create a blog object to add it in json server.. No need to add Id when we make post request json server will create a unique id for the blogs

import { useState } from "react";

*const* Create = () *=>*

{

*const* [title,settitle] = useState('');

*const* [body,setbody] = useState('');

*const* [author,setauthor] = useState('');

*const* Handlesubmit = (*e*) *=>* {

*e*.preventDefault();

*const* blogs = {title,body,author};

        console.log(blogs);

    }

    return(

        <div className="Create">

            <form onSubmit={Handlesubmit}>

                <label>Blog Title:</label>

                <input

                    type="text"

                    required

                    value={title}

                    onChange={ (*e*) *=>* settitle(*e*.target.value)}

                />

                <label>Blog Body:</label>

                <textarea

                    required

                    value={body}

                    onChange={ (*e*) *=>* setbody(*e*.target.value)}

                ></textarea>

                <label>Blog Author:</label>

                <select

                    value={author}

                    onChange={ (*e*) *=>* setauthor(*e*.target.value)}

                >

                    <option value="Akilan">Akilan</option>

                    <option value="Sundhar">Sundhar</option>

                </select>

                <button> ADD BLOG </button>

                <h1>{title}</h1>

                <h1>{body}</h1>

                <h1>{author}</h1>

            </form>

        </div>

    );

}

export default Create;

27)POST Request

We can achieve this by fetch request using fetch API , it takes following argument. 1=>Endpoint (<http://localhost:8000/blogs>)

2=>Type of request (sending method)

3=>header property (type of content that is being sent) 🡪 json type

4=>body ( actual data we are sending with the request ). we cant send the blog object as it is …have to turn from an object into json string

Json server automatically add the id property.

fetch('http://localhost:8000/blogs' , {

            method:"POST",

            header: {"Content-Type":"application/json"},

            body: JSON.stringify(blog)

        })

Sincs this fetch is asynchronous this returns a promise , this fires the function when the promise is complete.

When we add the blog in the button it should show like loading until the blog is added.We will use ispending state.

Initially we didn’t click the ( Add Blog ) button so set the ispending to false.

*const* [isPending,setispending]=useState(false);

Once clicked ..it will go to Handlesubmit..now make the ispending to true….After when the fetch get completed again set the ispending to false..

ispending true means have to show the Loading message.

27)Redirects

Once new blog added it stays on new blog page itself. Need to redirect to the Home page.To do this we need a hook called useHistory.

UseHistory allows to go back and forward through the history much like browser arrows and adds new page to the history (re-direct them to new route)

*const* history=useHistory();

Now we have object that represents history on this object we can use several methods go back or forward through history and also redirect the user

To go-back we have method like go..

history.go(-1);

To go-forward change to (+ve) integer

history.go(1);

Now have to re-direct to Home Page. To do this have to use push() method . It accepts the a parameter of page route.

  history.push('/');

fetch('http://localhost:8000/blogs' , {

            method:"POST",

            header: {"Content-Type":"application/json"},

            body: JSON.stringify(blogs)

        }).then( () *=>* {

            console.log("New blog added successfully");

            setispending(false);

history.push('/');

        })

28)Deleting Blog:

We are placing a button event(handleDelete) in BlogDetails component to delete the blog. Inside the handleDelete we have fetch request which is going to be a delete request.

In fetch request as it is delete we have to include an id of the blog in the endpoint.In BlogDetails component as we have access to an id we can use that

import { useParams } from "react-router-dom";

import useFetch from "./useFetch";

import useHistory from "react-router-dom";

*const* BlogDetails = () *=>* {

*const* { id } = useParams();

*const* { data: blog,error,isPending }=useFetch('http://localhost:8000/blogs/' + id);

*const* history=useHistory();

*const* handleDelete = () *=>* {

        fetch('http://localhost:8000/blogs'+ blog.id ,{

            method:'DELETE'

        }).then( () *=>* {

            history.push('/');

        })

    }

    return (

        <div className="BlogDetails">

        { isPending && <div>Loading..</div> }

        { error && <div>{ error }</div> }

        { blog && (

            <article>

                <p>{ blog.title }</p>

                <p>Written By{ blog.author }</p>

                <p>{ blog.body }</p>

                <button onClick={ handleDelete }>DELETE</button>

            </article>

        )}

        </div>

    );

}

export default BlogDetails;

Once the fetch request is complete .then() will fire a function…

Once the blog is deleted have to re-direct the user to Home Page . To do this have to use useHistory hook.

29) 404 Pages and next steps

When the user goes to URL that does not exist because of no route setup. To display error message we are creating a component called NotFound and we are re-directing to Home page.

import { Link } from 'react-router-dom';

*const* NotFound = () *=>* {

    return(

        <div class="not-found">

            <h1>Sorry</h1>

            <p>The page cant be found</p>

            <*Link* to="/">Back to Home Page</*Link*>

        </div>

    )

}

export default NotFound;

Have to call this component when user types wrong URL … Calling in App.js

 <*Route* path="\*">

              <*NotFound*></*NotFound*>

 </*Route*>

If it catches the URL other then defined NotFound component will be called.It always place in last otherwise it will match with “/”.

          <*Switch*>

            <*Route* exact  path="/">

              <*Home*/>

            </*Route*>

            <*Route* path="/Create">

              <*Create*/>

            </*Route*>

            <*Route* path="/blogs/:id">

              <*BlogDetails*/>

            </*Route*>

            <*Route* path="\*">

              <*NotFound*/>

            </*Route*>

          </*Switch*>

export default App;