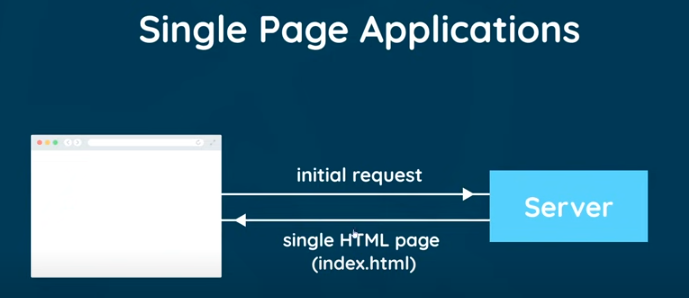
**1-Introduction**

* React is JavaScript library used to create interactive websites and use interfaces.
* Allows us to easily create Single Page Applications (SPA).
* In SPA server only ever needs to send single html page to the browser for the website to run fully and react takes over and manages the whole website in the browser including any kind of website data, or user interactivity/events and routing from page to page.



* In SPA user can navigate from Page to page using links but the new pages are not sent from the server when link is clicked, instead react changes(injects) all the content of the page in the browser depending on the route(URL of the link). This mechanism increases the speed and user experience of the app, unlike traditional website where request is sent to the server for every page rendering when link is clicked,
* Prerequisite:
  + Text editor like Vs code and set the following
    - Install extension called “simple react snippets”
    - Manage > Settings > search bar : emmet > under include languages > Add item> Key: javascript, value: javascriptreact > ok
  + Install latest version of node
  + If you need the complete code [https://github.com/iamshaunjp/Complet..](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbFVDQWpmTlNEUTZtY3JIcy1ESW1CbVJPZVNHQXxBQ3Jtc0ttNHR1dVlLWThnMHBjekRpMkZkRmt4eVptTC0wNUp2TFNxU3hXRHFQY3JHZTBYNG8yXzJDeUFqaTNnN2JwT1ZpRHU5ZU9Vb1dsR3ZXTThxSHV2Wk5XRnZLaHFvVEl4dENTVENxYXdVWXVWLWJLSnNzQQ&q=https%3A%2F%2Fgithub.com%2Fiamshaunjp%2FComplete-React-Tutorial&v=j942wKiXFu8)

**2 - Creating a React Application**

* Steps
  + Create Directory for your project and open it with vs code
  + Create your react app via “Vite” : # *npm create vite@latest -- --template react* then follow the prompt ; give the project name, select react and JavaScript as a language
    - # cd blog // Switch to the project directory let say “blog” :
    - # npm install //installs dependencies for us
  + Overview of the file structure
    - “node\_modules” contains all our projects dependencies including react library
    - “public” stores all public files(public to the browser)
    - “index.html” file is the one html file served to the browser (SPA) and all our react code is injected into this one file inside its root element.
    - “src” folder contains most of the code you write, by default it contains the entry point file(*main.jsx*) for our application which is responsible for taking all our react components and mounting them to the DOM inside the root element.
  + Cleaning file structure:
    - Delete the some of the default files and images that are not necessary for your project, such as:
    - Clear the content inside the parent <div> inside App.jsx
  + Viewing the project in the browser:
    - Open terminal on the Vs code ># npm run dev
    - Then the above script will provide us the URL at which we can run the app(<http://localhost:5173/> ), then open that URL on browser.

# 3 - Components & Templates

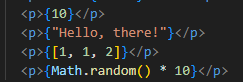
* Components are reusable building blocks of react application.
* Component is a self-contained section of content. Components come together to build the entire app.
* Component contains:
  + Template : html +content
  + Logic : JS
* The *main.jsx* takes the app.jsx (root component) and render it in the root element of the index.html file, this makes it the SPA.
* Component is a function returning a html styled template written with the mixture of “html” and JavaScript code known as “JSX (JavaScript -extension)” and in the background the transpiler called “babel” converts all of this JSX templates into html file and renders it to the DOM.
* JSX Vs HTML
  + In jsx when we add “className“ (camelCased) instead of class (for html class attribute) in element’s attribute, because “class” is a reserved keyword in JavaScript(Hence the code is integrated with JavaScript). Then when it is converted to pure html, it will be changed to just class during rendering in the browser.
* Components should:
  + Component name should start with Capital letter such as “App.jsx”
  + Must return one element: either <div> or fragment ( <>….. </>)

# 4 - Dynamic Values in Templates

* In “Jsx” template in react, we can output dynamic values or variables inside the it.
* Note, you can write any valid JavaScript code before the return statement (template) inside the component function.
* First store the value in the variable and pass it in the template with curly bracket.
* When the template finds the curly braces, it knows we are trying to output a dynamic value.



* React converts whatever data type to string before outputs it to the browser except Booleans and objects. i.e. you can output numbers, strings and arrays but not Boolean and objects
* You can also output dynamic values inside the curly braces to display the value or result of calculation. i.e. you can write any valid JavaScript statement inside the curly braces.

 // you can out put the result of a computation as well

//note, when it outputs array, it concatenates the elements together as a string even though they are numbers. In the above example the output of the array will be 112

* You can use dynamic values as an attribute values in the element tags





# - Multiple Components

* **App.jsx** is the root component of the application, meaning it is the first component that is rendered to the DOM and sits at the very top of our application. And it is rendered by index.html file
* In react application components are structured in a way that makes up a component tree, being the root component at the top of the tree and this is the component that is initially rendered inside the index.html file. This root component is ‘*App.jsx’*. then if we were going to make more components, we nest them inside this root component.

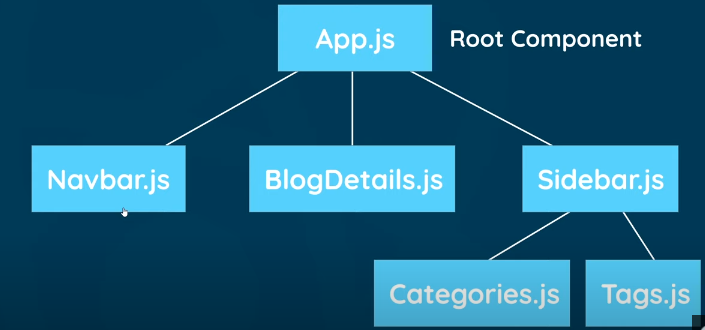
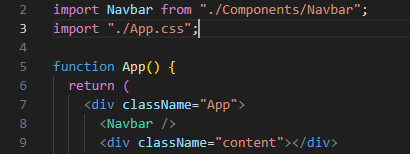


Fig: Component tree

* Creating Component

***Remember component is just a function returning a “jsx” template, and that function is exported at the bottom of the file, so that it can be reused in other components.***

* + Create components folder inside “src” folder that will hold all of our components.
  + Create new JSX file Navbar.jsx //note the first letter of the component name is capitalized.
  + If you install the “simple react snippet” extension to the VS code, you can create boilerplate component code snippet by just typing ‘sfc’ which creates stateless functional component and export it at the end. otherwise just type the entire code manually.
  + Then create a template inside the parent element
  + Import it in the component you want to use it and implement it inside the template as <Navbar/>



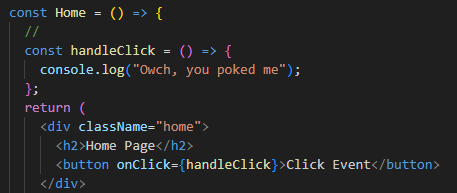
# 6 - Adding Styles

* App.css is imported in the root component (App.jsx) is an style file that is applied to the root component and nested components nested to the root during rendering in the browser.
  + So lets delete the App.css file and remove its import from App.jsx
* Add your css styles to the Index.css, because it acts as global style sheet that could be applied to all components. Index.css file is imported in the to main.jsx file which add all the styles to the <head> of every page.
  + So let use index.css for all of our style (recommended), so delete all its content
* Note, if you want to add inline style, you have to add it as dynamic value in curly braces as an object (key-value) pair where the key represents the attribute. and when the attribute name is two word chained by hyphen (like background-color) which is valid in CSS, we change it to the camel case, because JavaScript thinks it is a minus(-) sign.

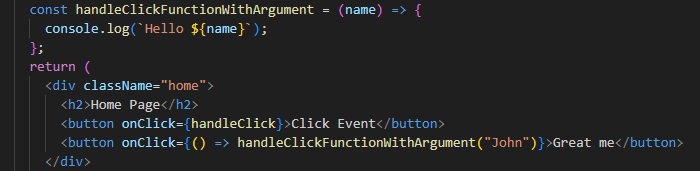


# 7 - Click Events

* Create the function outside the template (i.e. before the return statement) then hook it to the event handler with dynamic value(the created function) curly braces to the element in the template.
* Note, You do not invoke it when you do that like handleClick() (Wrong way), because it will run with before the event happen during page loading so, ***you just set a reference to the function*** .



* However, if the function takes argument, we wrap it inside anonymous function, otherwise if you still pass the argument in parenthesis and leave it as it is; the function will still automatically invoked.

 D

* If you want to access the event object , such as e.target
  + If the function doesn’t take argument, you can just pass the event to the function declaration as follow during definition and can be accessed as first argument:



* + However, when it does take argument, you will pass access the event object during the implementation by passing the event object to the anonymous function that wraps the event handler function, then it will automatically obtain it, so you pass it as argument to your custom function( the ‘e’ next to ‘John”) so you can use it inside function.





# 8 - Using State (use State hook)

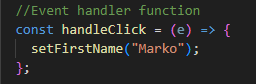
* State is the data being used in specific component at specific time.
* State can be an array of values, Booleans, strings, objects or any other data that our component can uses.
* Variables or data that change over time or in reaction to some event such as user clicking button. So, in response to these changes, we should able to propagate these changes (new values) after the effect during the re-rendering. We use “hooks” to handle this change in states of the elements known as “use state”.
* Hook in react is special type of function that does a certain job and its name start/prefixed with the word “use”,
* Use state hook gives us a way to make a reactive value and change that value whenever we want.
* To use “Use state hook”:
  + First you import them at the top of your JSX file by de-structuring from the react library.

*# import { useState } from "react";* //grabs the useState function from react library to use it inside our component

* + Inside the component , Create reactive value using useState function by passing the initial value as argument and store the values returned from the hook to the array, by array de-structuring syntax, the first value of the array being the initial value and the second being the function we can use to change the initial value(usually start with “set” word)



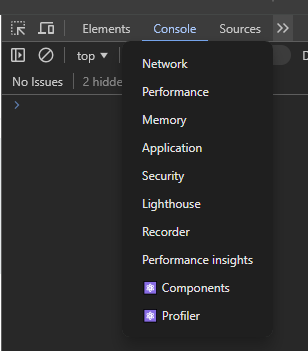
* + - //Note the value of “firstName” is “John”, which is the initial value
    - to change the initial value when an event happened, we will use the second argument (set function) inside a handler function to make the alteration to the initial value.

// he setFirstName function(second argument to the useState function, changes the initial value provided during declaration.

* + The initial value can be any data type: number, Boolean, string, object, array …

# 9 - Intro to React Dev Tools

* Install the “React Developer tool” from google store or Extinctions/Add-ons, which integrates with browsers developer tool to give us extra feature we can use on any website created with react.
* Can be access from the developer tools menu(the components and profiler)

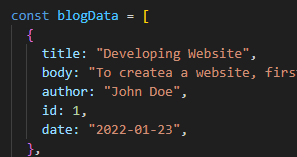
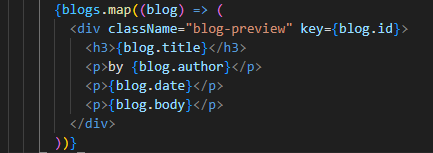
// Click the >> arrow to scaffold the content

* The components tab provides the component diagram or component tree the application, and clicking on the tree structure will give us details including the current state data.
* <https://www.youtube.com/watch?v=rb1GWqCJid4&list=PL4cUxeGkcC9gZD-Tvwfod2gaISzfRiP9d&index=9>

# 10 - Outputting Lists

* We use map( ) method which takes a callback function to iterate through the dataset to populate each data into the template
* Use unique Id as key for each list, because react keeps track of an item when outputs to the DOM, it uses these unique Id to know if there is a change to the data/array.





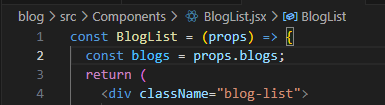
//note blogs is array of objects

# 11 - Props

* When you want to reuse a component in multiple pages or places across the app, you will create a reusable component via props and when you implement the component you will pass values of the props that corresponds to that particular instance need.
* Creating reusable component:
  + Create a separate JSX file: # touch BlogList
  + Create the code snippet making your reusable component



* + Go to the component or page you want import it and place it to the location you want to implement it. <BlogList/>
  + For example, we cut out the list we create above to iterate and list blogs for from home page and make it a reusable component in BlogList.jsx and we import and implemented it in home.jsx page as follows. However hence the “blogs” Array of object is defined in the home and not in “BlogPost” component, we run into error because we are not accessing that data.
    - So to solve this problem we either copy the entire data to the BlogList component as well (which will be duplicated, ) or we use props to access the data.
    - Why use props?
      * Makes our component more reusable
      * Pops are a way to pass data from parent component to the child component which Allows the data anywhere we want to use
    - So to make the above BlogList example work in “Home” component, we pass the props data form parent component(Home) to child Component (<BlogPost />) as follows:
      * Got to the child component definition(in BlogList.jsx) file and modify it to accepts the props object and extract the data from this props object and store it on variable for reuse

//*modified the BlogList*

* + - * Or you can use the destructuring syntax instead of extracting and storing props data on the variable
      * 
      * We pass the data as a property name to the child component

//Home.jsx

* + Any props we send through into component is attached to the props object passed as argument to reuseable component definition and accessed from this object.

# 12 - Reusing Components

* We can reuse a component with different data. for example, What if you want to filter only the blogs written by specific author:
  + We use a filter() method on our dataset(Array) which returns new array containing elements that match the specified criteria in the argument. pass callback function as argument to the filter method which actually takes each element and isolate those who match the criteria.



* + This mechanism is useful when you want to create a search page, where you find an item that matches by a title (as a search term)

# 13 - Functions as Props

* Delete an Item
  + Update the state by deleting item from dataset (array of objects), for example specific blog from the list.
  + First add Delete button for each blog in the list, and give click event handler to it.
  + And wrap the delete function inside callback function, so that it won’t executed automatically on page load and to pass argument to target for specific item(blog ID) from the list.



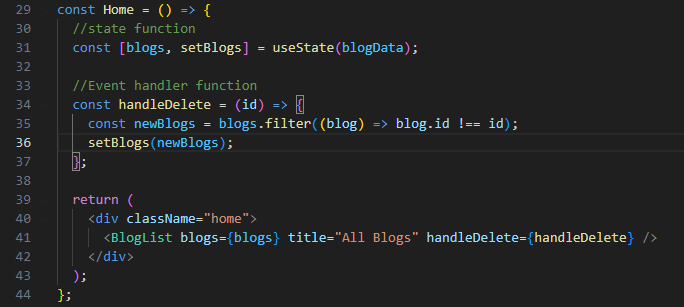
* + Go to the parent component(Home) where the data and state is defined so that we can interact with the data directly. So create the implementation of the delete function where the state is defined.
  + Then pass this method as a prop to the child component in the Home



* + Then go back to child component (BlogList) and add delete function as as props being passed



* + Then use the **setBlogs** function to update the data in the delete function, using filter() method



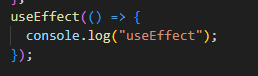
Note the filter method doesn’t mutate or change the original array, it just creates new array from original array by filtering elements that satisfy the criteria(in this case elements that doesn’t equaled to the id)

# 14 - useEffect Hook (the basics)

* useEffect is a hooks that runs a function for every render of the component
* Remember components renders initially when page loads to the DOM and then it also re-rendered every time state data changes to update that state in the browser.
* Usage:
  + First import it at the top of the file



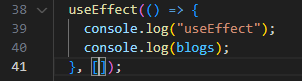
* + Use it in the component above the return statement/before the template
    - We don’t need to store it on variable since it doesn’t return anything to us, we just pass a function we want to run every time there is a re-render as argument to it.



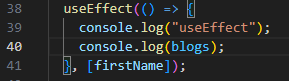
* Usually, we use it to fetch data, communicate with authentication service and such which are known as side effect, (for now we are just logging message to console in this example)
* You can also access state in useEffect.
* Warning: do not change state data in useEffect hook, you will end up with continuous loop of renders. Because for every render there will be state change which will trigger render all over again and so on.

# 15 - useEffect Dependencies

* You don’t always run function after every render, may be for certain renders, in such cases we use dependency array. It is an array we pass in to useEffect hook as a second argument.

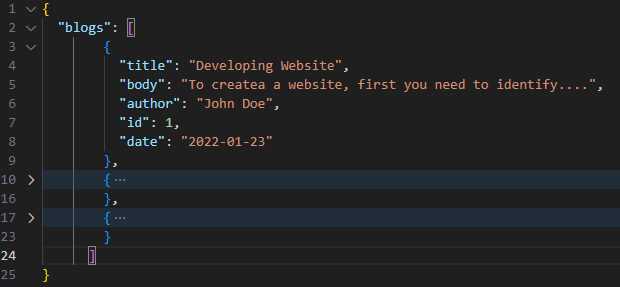
// hence we don’t specified nothing, the function inside the useEffect runs only once(during initial page render only)

* Adding dependency example:

//runs the function initially when the page loads and when the state of firstName changes, otherwise it blocks from running the function

**16 - Using JSON Server**

* One of the use case we use useEffect is when we need to fetch data, because the useEffects runs a function when the component initially renders, so it is good time to fetch data on the initial page load. Then we use that data in our application.
* JSON server allow us to build fake rest Api that simulate the real life database and endpoint in apps.
  + First create JSON file that act as database:
    - Create folder called “data” inside root directory(blog) and create file in it named “db.json” then paste the json into it
  + In Json server, top level property is considered a resource,



for example: “blogs” here is considered a resource and creates an end points for us to interact with this resource, so we can do things delete items from it, add items to it, edit items, get items etc

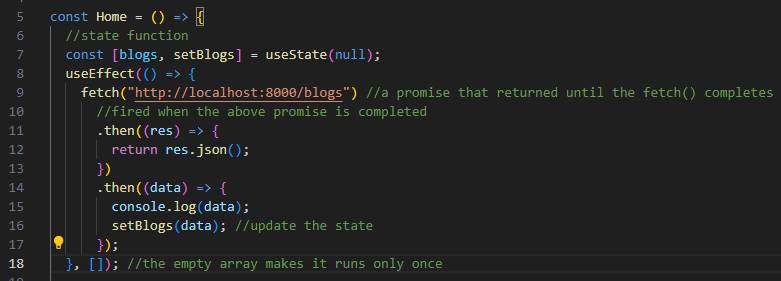
* Then use Json server package to watch this file and wrap it with some endpoints.
  + Either install the Json server package locally to this project or use “npx” to run script/code from web.
  + Here lets use the second option(running “npx”)
    - Open new terminal (hence we are running the app on development server on the first terminal)

# blog> *npx json-server --watch data/db.json --port 8000* // watchs the data located at “data/db.json” on port 8000// note the port can be any port which is not occupied right now by other services

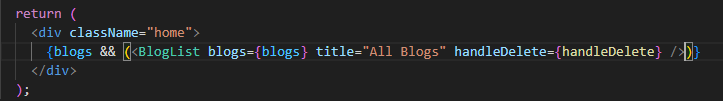


# 17 - Fetching Data with useEffect

* Now lets fetch the data using the useEffect hook, the data we stored in “db.json”(which acts as DB) and watched by Json-server to serve the data on endpoints.
* Remember when the second argument of useEffect (i.e. dependency array) is empty(do not specified when to run) the function inside it will only runs once during initial load of the component. So we would like to fetch this data once the component first renders.
  + Empty the array of blogs from home component, hence we are about to bring data from external Json file which serves data at the endpoints likes servers do.
  + Set the initial value of the blog to null in state function, so later we will update the state using the setBlog function of the state
  + Then make fetch request inside the useEffect hook, using fetch() / fetch api: hence it is a promise we attach then() on it which receives a response object when the promise completes. Note the response object is not the actual data, so to get data out of it we apply json() to the response which parses(or extracts) the data to Json format for us and return to us
  + Again hence the fetching and parsing also asynchronous and takes time to complete it become promise, so we will attach another then() method which takes the actual data returned in the above operation
  + Then update the state using setBlog function. Note these do note lead to infinite loop of rendering, because the dependence array is empty, which means the function inside useEffect runs only once during initial rendering of the component not wherever data changes.



* + Note, you might run into error when you save and run the application, because hence the fetching of data takes time and the initial value of the blogs is “null”, when it try to map() though run it will generate error. So to handle this problem check to see if the data fetching is completed and not null before rendering the template.



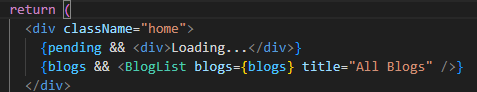
* + This above validation method using Logical and(“&&”) is called conditional templating, in which the logical and evaluate the left first and if it is “false” it never bother to output/render the right side template. However if the evaluation on the left becomes true, it will next evaluate the right side and outputs it to the browser. That means, the right side of the logical “and” is renders only if the left side evaluates to true.

# 18 - Conditional Loading Message

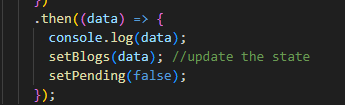
* Lets create “loading…” message, while the data is fetched from the server
  + Create additional state that trucks the change data loading inside home component



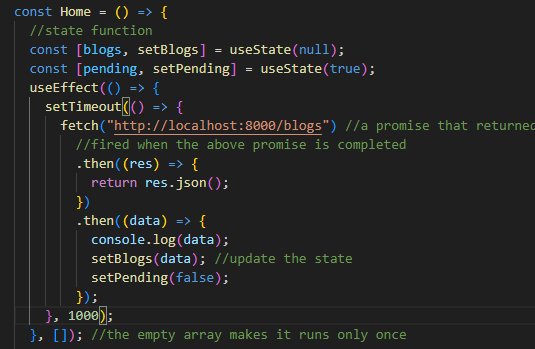
* + Then create conditional template for the loading



* + To show loading only while data fetching is ongoing and hidde it when data fetching completes setPending() state to false inside the useEffect hood after the data is brought.



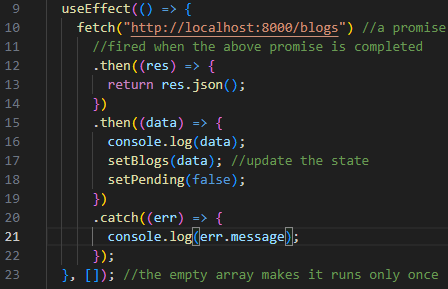
* + This make the loading message and the blogs NOT to be shown at same. Note hence we are fetching data from local PC the fetching process happens in split of second and you might not notice the loading message for long time. If you want to simulate real life delay of data loading and view it for long time put the fetch() inside setTimeOut() as follows:



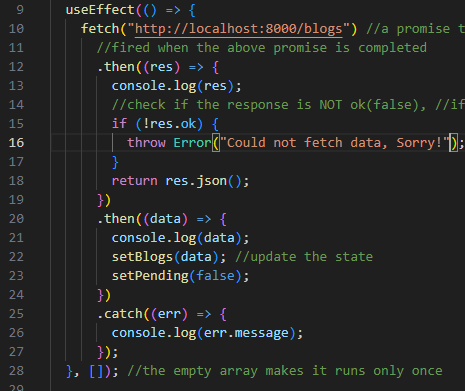
* + The above setTimeout() method delays the request for 1second, don’t to this real time applications(this is only for simulation purpose only)

# 19 - Handling Fetch Errors

* Handling error that might happen while fetching data from server, which could be error sent by server or connection error where we can’t even able to connect to the server.
  + First attach a catch() block to the fetch() which takes a callback function which logs the error message to console or do other things you want.

 // this catches only the Network error

* + For example, Lets test it by catching a network error by canceling the Json-server.//*check console*
  + What if the request reaches the server but the server sends error back or may be the endpoint we try to fetch from doesn’t exist or if the request is denied. Hence the above catch block doesn’t catches such errors automatically when we use fetch() api, since the request is reaching the server and the response is returned from it, it is just the response doesn’t have a data(it contains different status).
  + In this case we check the response we are getting back and
    - Re-run the Json-server via “npx” script
    - Check if the response object returns with data and if NOT throw the error so that it can be cached inside catch block, which log the message to console or take other action

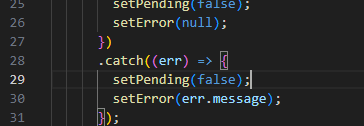


To test it change URL to resourse in fetch() by adding single character, this time the custom error you created for the server error is thrown from if Check and caught by catch blog and the error message is logged to the console.//*Could* not fetch data, Sorry!

* + - You can also store the error on state and output it to the browser
      * Create state at the top

 D

* + - * Then in the catch block, instead of logging to the console update the error state with setError function by passing “err.message” property



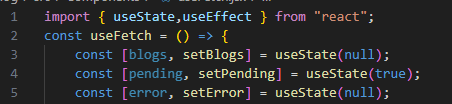
//Note we also set the Loading message to false(#29) so that it won’t displayed in the browser and setError to null in then()(#26) block so that we can’t see error message we fetching data succeed

* + - * Then out put the message in the template via conditional rendering



# 20 - Making a Custom Hook

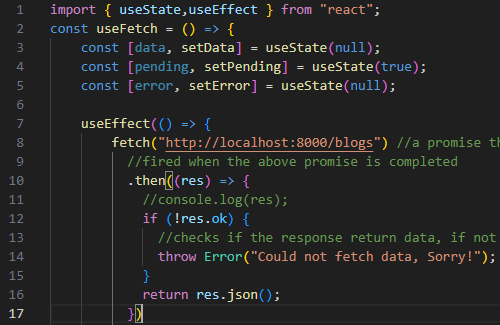
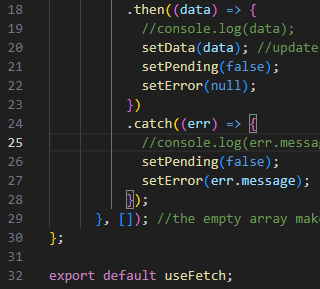
* Is a way of making hooks like the above useEffect (which fetch data, update state and catches and log error (lesson #19) reusable, by moving the code in separate JavaScript file so that it can be reused anywhere instead of writing these call all over again where we want to use it.
* Steps:
  + First make new file inside “src/components” folder named “useFetch.js”
  + Create function inside it named “useFetch” similar to filename and then add all the functionality to fetch the data in the function
    - *Note, custom hooks in react need to start with the word “use”, otherwise it wont work.*
  + Now copy all the fetch logic from home component and paste it inside this function and make some tweaks to it;
    - Import useState and useEffect
    - Register all of the states need by fetch()



* + - Export the function:

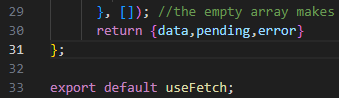


* + - Change the state data and function names , so they take generic names or make sense when reused for other cases too(i.e. blogs to data, and setBlogs to setData both in state and function implementation #line20

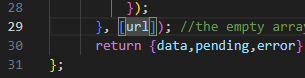
Note the “data” variable on line #18 and 20 is not a state data, it is a local variable that have the same name as the state variable and have local scope.

* + - Now return some values from this function for each of the states. The values can be numbers, strings, Booleans, arrays or objects. So at the end of the function return an object containing all the state data for data, pending and error.

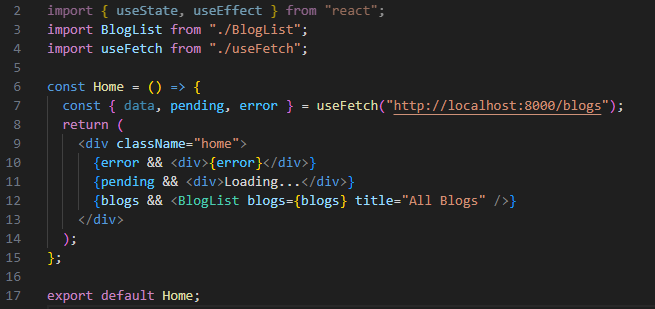


* + - Finally pass the endpoint or URL into the function, rather than hard code it in the fetch() argument. Because for each file where the hook is used the URL to the endpoint might be different. So cut the url from fetch() argument(line #8) and pass the url as a parameter to the useFetch hook/Function(line #2) and pass it down to fetch method too. In addition pass this url in the dependency array of useEffect(line #29) because it wherever the URL changes, it rerun the fetch function to get the data from that endpoint.



* + Import the useFetch custom hook into the home component use it:
    - De-structure the three properties returned at the end by the hook which represents the three state datas of data, pending and error; and pass the url/endpoint as argument to the hook pointing to the resource we are trying to fetch (line#7)



* + - Final change the “blogs” to “data”, because we make it to be generic(line#12 inside curly braces}

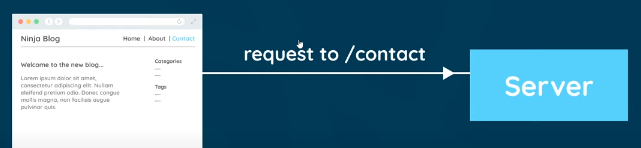
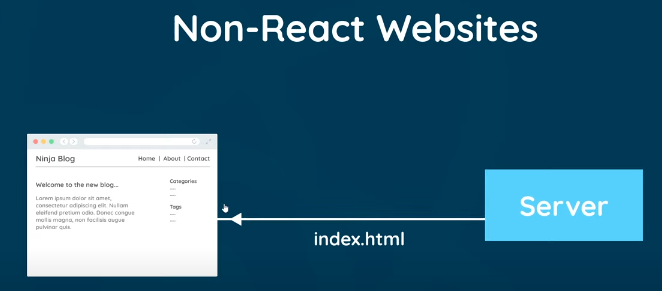
 or give it an alias when de-structuring (on line #7) so that the blogs can mean data also



This way the fetch function is change to custom hook, which can be imported to your component wherever you want to fetch a resource and pass the URL or endpoint of that resource as argument during declaration in your component(line #7) and destructure the state data from it and use it in your template as if that hook function and states are defined in the same component. This will make your code reusable (you don’t have to rewrite wherever you need to fetch data) and your code looks neat.

# 21 - The React Router

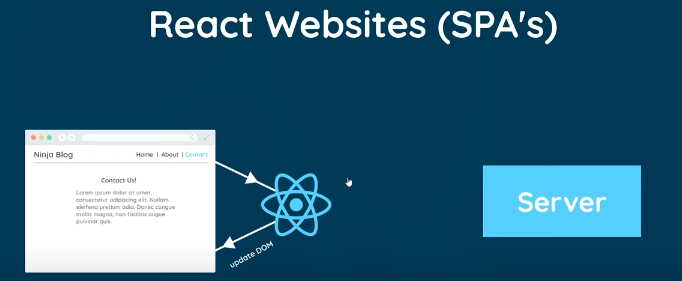
* Allows to navigate between pages
* On traditional website we go from page to page first we provide URL of the website in the browser which sends request to server then the server returns full HTML page. Then when user click on link on the browser for different page(like contact page) the browser again sends NEW request and server responds by new contact HTML page. In which we are making request to the server for every new pages and server responds with new full html page for every request.



* + Image 1: initial request to server and server responding with full html page
  + Image 2: new request via link from the landing page to the server for contact page
  + Image 3: server responding with new full html page contain contact data
* However, react applications delegate all routing and changing page content to the browser.
* It starts the same way like traditional website with initial request to the server via URL provided on the browsers address bar, and the server responds by sending back html page and compiled JavaScript files which controls the react application



  
However from this point on wards, react and react router take full control of the application.

* This means, initial the index.html file page we get from server is virtually empty, then react injects the content dynamically using the components we create. Then if we click the link to new page, the react router will intercept that request and stop it from going to the server; instead it looks at the new page request and inject the required content on the screen/index.html page.
* For example, clicking on the contact link, the react router will tell the react to inject contact component into the browser.

* Generally the react route work is, We assign top level component for each route or page and that component is dynamically injected into the browser when we visit that route.



* This process, reduces the request to the server and thus make the website faster.
* Setting up React router:
  + First install react router package(b/c it is not part of react core/library)
    - Open the 3rd terminal(*because the first is running the development server and the second is running the Json-server*) and run the following command to install react router,



// if you want use version 5 # npm install react-router-dom@6

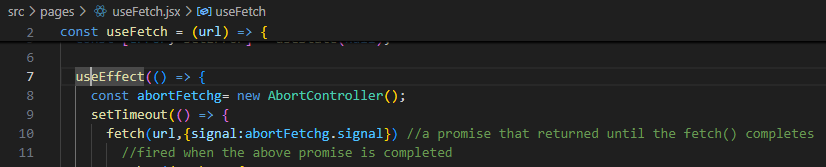
* + - Make react router available anywhere in your app, to do this open the root component “App.jsx” file in the src folder and import *BrowserRouter* .
* This tutorial works with react-router-dom version 5, however Seena used version 6 so please refer the other note from [React-Router6.4](React%20Router6.4.docx) which have new features and implementations. #Lesson 3
* Jump lesson 22, Which covers about “Exact Match Routes” & switches ; //deprecated in V6.4

# 23 - Router Links

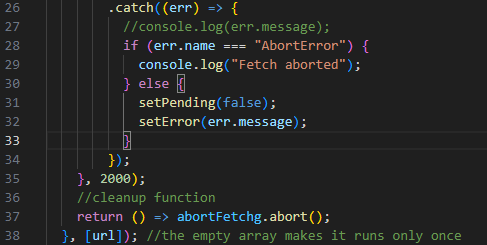
* Create new component “CreateBlog.jsx” in pages folder, a form for creating new blog
* Add it to the “RootLayout” as a <NavLink>, import it to the App.jsx and add it as child rout to the parent route of “RootLayout”; specify the path and element which renders the corresponding component to be displayed when this route requested.
* Note, when you use anchor <a> tag for route you are still sending request to the server, react is not interrupting it. So, for react to intercept the request instead of sending to server we have to use special Link tag.
* Implementation
  + Import the “Link” tag from react-router
  + Replace all the <a> anchor tags with <Link> tag.
  + Note <Link> tag doesn’t have “href” like anchor tag, instead it uses “to” property.
  + When rendered in the browser the react-router will automatically converts the <Link > tag to an anchor <a> tag under the hood. Still the converted anchor tag tries to send request to server for new page, however the <Link> has built in special functionality that prevent request to server and instead it look at URL or path and matches that against one of our routes and tries to inject content associate with it.

# 24 - useEffect Cleanup

* While we land on our home page, our custom “useEffect” hook fetchs data and tries to update the state in our home components. however, while the data is being fetched and loading if we go to other page(say Contact Page) before fetch completes we might get an error; which is caused when we switch to new page and fetch is complete, it still try to update the state in the Home component but the Home component is not in the browser anymore.
* The Error say “we can’t perform the react state update on unmounted component” where the unmounted component is <Home>. So we need a way of stopping the fetch, when the <Home> component using it unmounts. Todo this we use cleanup function within “useEffect” hook and an abort controller.
* Implementation:
  + Go to the “usefetch” hook we have created earlier,
    - Inside use Effect at the top create the abort Controller object and store it on variable; and associate it with specific fetch request to stop the fetch by adding second argument to the fetch method next to URL, which is an options for the fetch. Line#10



* + - This is done by first of all, runing the cleanup function before anything when the component that uses this hook and “*useEffects*” on mounts. To do this we place the cleanup function in the *useEffect* and return it. Also catch the error in catch block



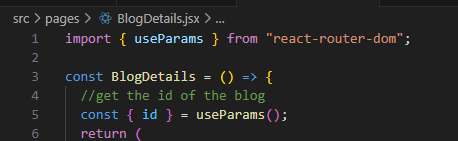
# 25 - Route Parameters

* For cases when dynamic values are passed as a part of the route. i.e. a route where certain part of it is changeable and regardless of the changeable parts it still renders same page or component.
* For example, in blog details page the route might look like “*/blogs/123*” where the “123” is the id of the blog we want to see, it is like a variable in route which varies for each blog data.
* Implementation:
  + Create a “BlogDetails.jsx” component inside pages folder
  + Create route for this Component inside App.jsx by importing it here.

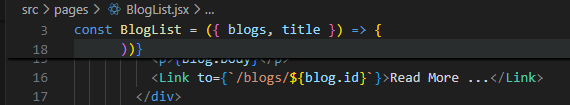


i.e. whatever the id is provided after the “/blogs” route we will display “BlogDetails”

* + Access this “id” inside “BlogDetails” component using “useParams” hook and fetch the blog matching that id to render it within this component.

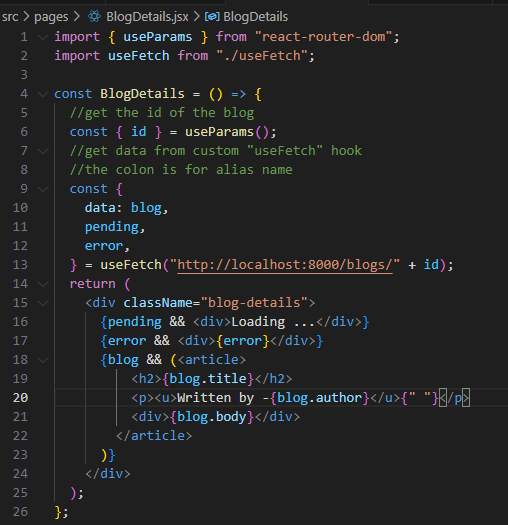


* + In <Home> add links that will redirect us to specific blog when clicked. Go to “BlogList” component create <Link> tag and put each blog inside it. //Read more …



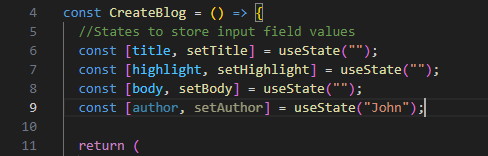
# 26 - Reusing Custom Hooks

* Making fetch request based on the captured “id” on the previous step to display the targer blog on the “BlogDetails” component, by reusing our custom “useFetch” hook we create earlier which returns three values(the data we trying to fetch, pending status(True/Fasle) and error if there is one)
* i.e. we will use “useFetch” hook in “BlogDetails” component pass in the URL or endpoint we want to use to fetch data from.
* Implementation steps:
  + Go to “BlockDetails” component, import “useFetch” inside it and grab data from this hook via de-structuring syntax by passing endpoint plus the id for the blog.// line#8
  + Conditionally display the data, if data is loaded or has no error, and render the data in template if the blog have data to return(not null)



# 27 – Controlled Inputs (forms)

* Controlled input is the way of setting up input fields in forms so that we can track their values by storing on state and if state changes the values, we see in input will also get updated. So that we are keeping our input field and states in sync with each other.
* Implementation:
  + Open “CreateBlog” component and create form input fields
  + Now when value is entered or updated into any of the input fields track that value and store it the states inside this component, so that we can do something with that data alter on. So,
    - First setup some state for each input field



//note , for select option(#line 9), we gave one of the option as initial value as default

* + - Associate this initial state value with the value of corresponding input field on “value” property; so whatever the state value is, it will be shown on the field. i.e. it will propagate the current value of the state to the input field



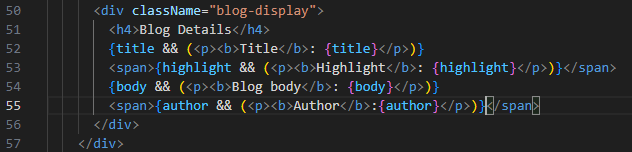
* + - To change the value of the state from the input field use “onChange()” method of the element and pass to it the “setTitle()” function of the state inside anonymous function so it wouldn’t automatically run on loading



i.e. when we try to insert/change the value on the form, the “onChange” property triggers the “setTitle()” method by passing the entered value(captured by event object/e.target.value) which updates the initial value of state with the newly inserted one. At same time when the state value is updated, the value of the “value” property also replaced with the new updated state value and sync with the state value resulting in two ways binding b/n input and state.

So we are tracking what the user types into the input field.

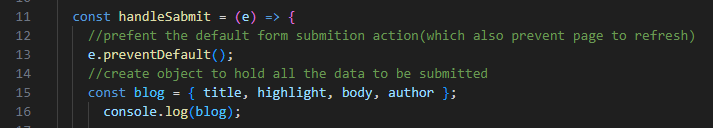
* + - You can output the value in the template to see the update live. Note we have displayed the values conditionally so that when the fields is empty we leave out the label



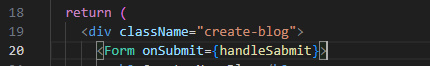
# 28 - Submit Events

* When the button is clicked inside a form it fires submit event on the form itself, so we can listen for that submit event and react to it or we attach the click event to the button itself and react to that event.
* To react to the submit event:
  + Create a function that handles the submit event outside the template named “handleSubmit()”, which takes the event object ,
  + Inside the function Create a blog object that is untimely submitted, which contains all the values at one place.

***Note***, when we are using a Json-server we don’t have to provide an “id” property for uniqueness of an item; b/c when we make a post request the Json-server will automatically add a unique id for us



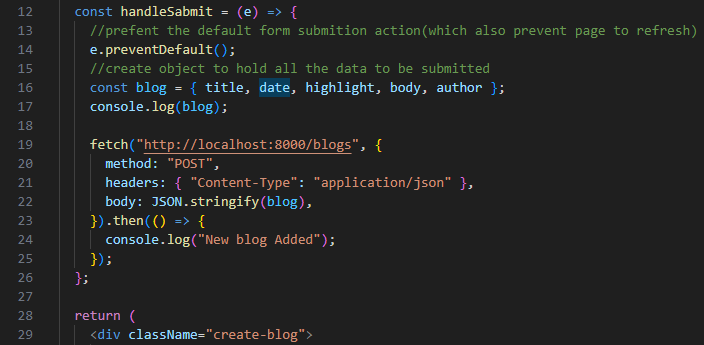
* + In the Form tag, pass this function to the on “onSubmit” property



Now when you click the submit button the object will be submitted, you can verify by adding console.log(blog) to the function declaration. //below line #16

# 29 - Making a POST Request

* Adding the submitted data to the data file by sending post request to the Json-server. You can do this by using “useFetch” hook we created earlier by editing it to handle post request as well, or by making the post request inside the “handleSubmit” function. For now lets take the second option, making post request inside function; b/c we are going to make that request once in the whole application not in multiple paces.
* Implementation:
  + Inside the handleSubmit() function make a fetch request using fetch API, which takes endpoint/URL of the resource as first argument, the second argument object which specify the method(type of request we are sending), the headings(content type being sent) and the body( actual data being submitted; which should be converted to string from a “blog” object we created in the previous lesson28-(line#15)) .

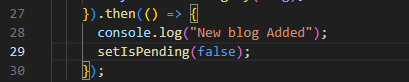


This will make a post request to the specified endpoint(1st argument) to add a new blog to Json-server where it automatically give unique id to it as well. Hence this fetch request is asynchronous and return a promise we attach a then() method to it, which fires anonymous function when it is complete.// for this

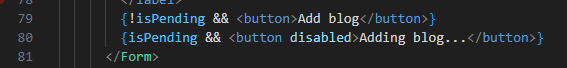
* + Let’s add a loading message at the bottom:
    - Create separate state for this purpose



* + - Inside handleSubmit() function; update the setIsPending() method
    - 
    - Then inside then() block set isPending to false again, b/c the promise is returned when data fetching is completed



Then conditionally display the button only when isPending is false; and if it is true show disabled button



* + - Here you can add other loading message or do anything you like

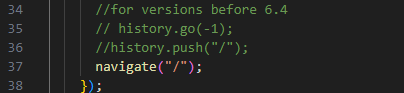
# 30 - Programmatic Redirects

* Let’s redirect user to another page after form is submitted using a “useHistory” hook, which allow us to go back and forward through the history and also add new page into the history. i.e. redirect them to new route. //this is changed in version 6.4 , use useNavigate instead of useHistory, navigate = useNavigate() instead of history = useHistory(), and navigate('/') instead of history.push('/'), it should work then^^
* Implementation:
  + Inside CreateBlog Component import “useNavigate” and invoke it on variable, which result in object having many method we can use to go back and forth in browsing history



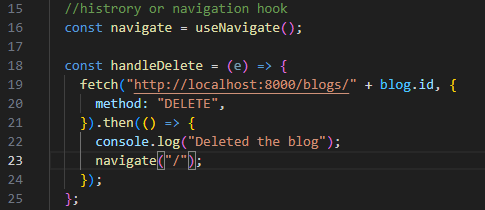
* + To go back in history: invoke the “go()” method with -ve value. For e.g. -1 take us back one step
  + So Inside handleSubmit method’s then() block after isPending place the f/f to go one page back

* + And if you want to go forward just provide +ve value to the go() //history(1)
  + If you want redirect user to specific page(say home page) do use the push() method on the history object we have created by passing the route to the page we want to redirect the user:



# 31 - Deleting Blogs

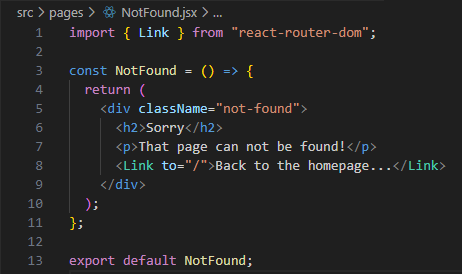
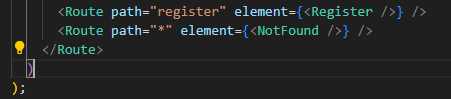
* Implementation
  + Go to “BlogDetails” component and add a delete button to each blog
  + Create function that handles a Delete event when button’s onClick event triggered, so inside first make a fetch request which is going to be a delete request, that takes an endpoint/URL to item( by its id) to be deleted as first argument, and an object which specify the method(type of request to be sent) as a second agreement and attach then() block witch confirms the deletion and redirect the user. b/c the fetch is asynchronous and “then()” method will be fired when the fetch() is complete.
    - Import and Use a useNavigate” hook inside the then() block to redirect the user to homepage after deletion



* + Attach a click handler function to the button the “onClick” property



# 32 - 404 Pages & Next Steps

* Not Found Error when the route requested doesn’t exist
* Implementation:
  + Create a “NotFound.jsx” component that be shown when someone requests the route that doesn’t exist
  + 
  + Go to “App.jsx” and at the end(if it is at the top it matches any route that comes in”) of all routes add “catch all route” for this by passing “\*” to the path property of the route. i.e. if none of the above routes match, then catch all( “\*” ) any other route.
  + 
  + //View react with firebase authentication, build data visualization with D3.js and firebase, build web apps with vue j2 and firebase.