FULL MODERN REACT TUTORIAL 2021

PLATFORM: D

STARTED: 8th of May 2022

TUTOR: The Net Ninja

FINISHED: 14th of May 2022

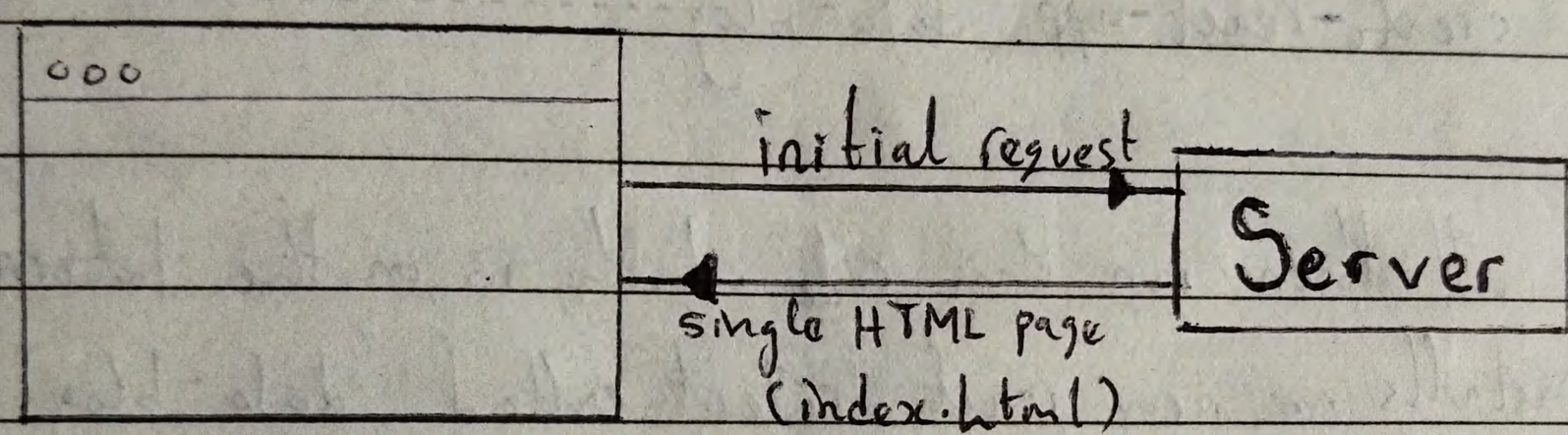
DURATION:

INTRODUCTION #1

What is React?

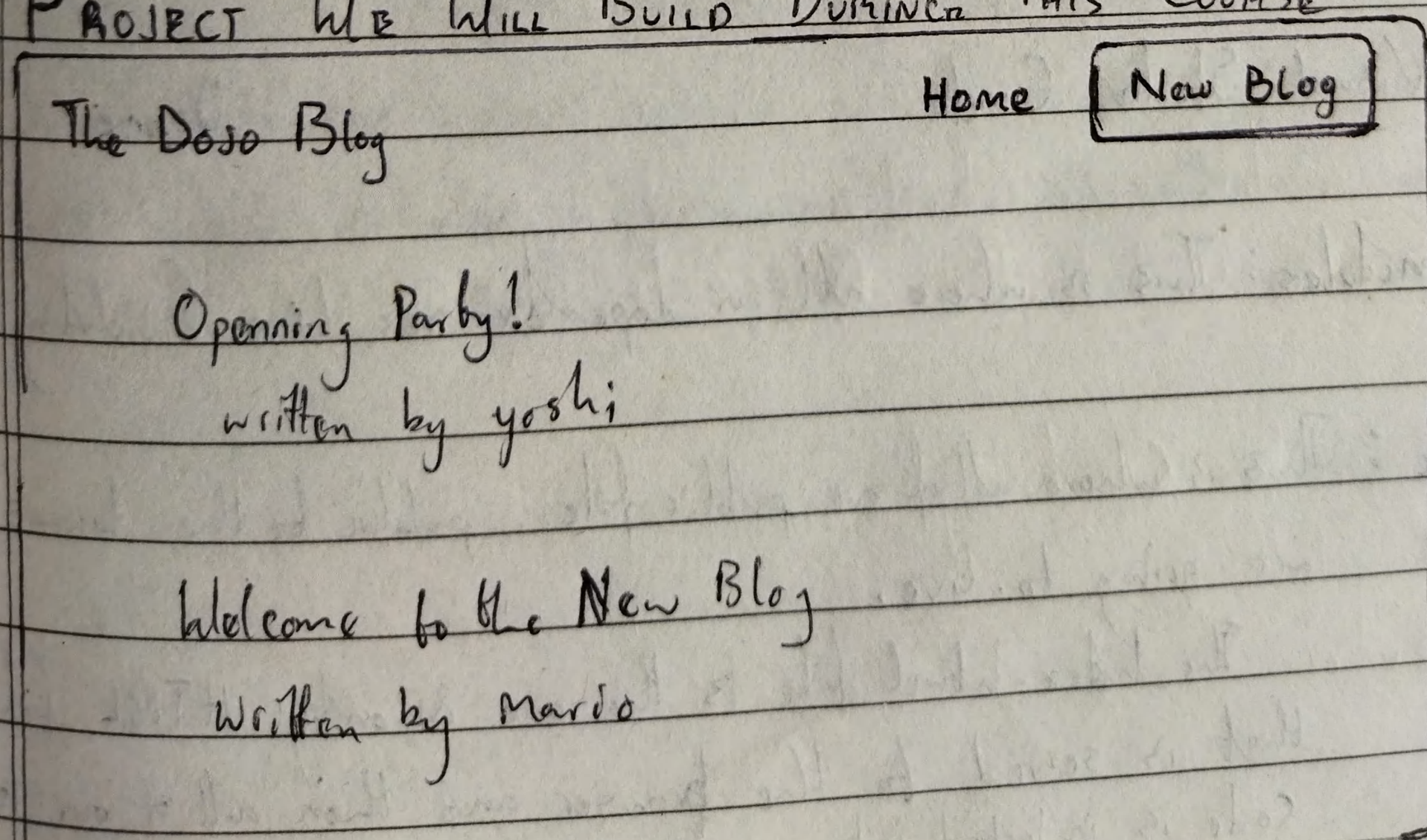
- React is a Javascript library used to create websites
- Allows us to easily create single page apps - SPA for short

By SPA - this means the server only ever needs to send a single HTML page to the Browser for the website to run fully



And then React takes over and manages the whole website in the Browser - This makes content and navigation through out our website happen very speedily resulting in better User Experience.

PROJECT WE WILL BUILD DURING THIS COURSE



Creating A React Application #2

Open any terminal and check if you have node installed

> node -v

v14.1.0

> cd Documents

> cd Tuts

> npx create-react-app dojo-blog

This will then run this code which is on the Internet and it installs a new react project called "dojo-blog"

This takes time to finish. But once it is

> cd dojo-blog

> code . (opens directory in Visual Studio Code)

Visual Studio Code Opens

- node_modules: This is where all our dependencies live (including Reac

- public: This is where all of our public files, public to the browser are going to live.

  The index.html file is the one & only HTML file that is served to the browser and then all of our react code is injected into

This is where

- Src: 99% of the Code we will be writing are going to live. All of our react components

Now to Start up our local development server in Terminal

> npm run start

NOTE: usually whenever we upload our react project to github we usually leave out the ☐ node_modules and the reason being that it is too large (Since it contains all of our project dependencies.

So when we later download this project to our system. How do we get back the ☐ nodes_modules

> npm install

This Command goes into our root directory and checks our packages.json (which remember contains all the listing of our projects dependencies) and downloads those dependencies back into ☐ node_modules

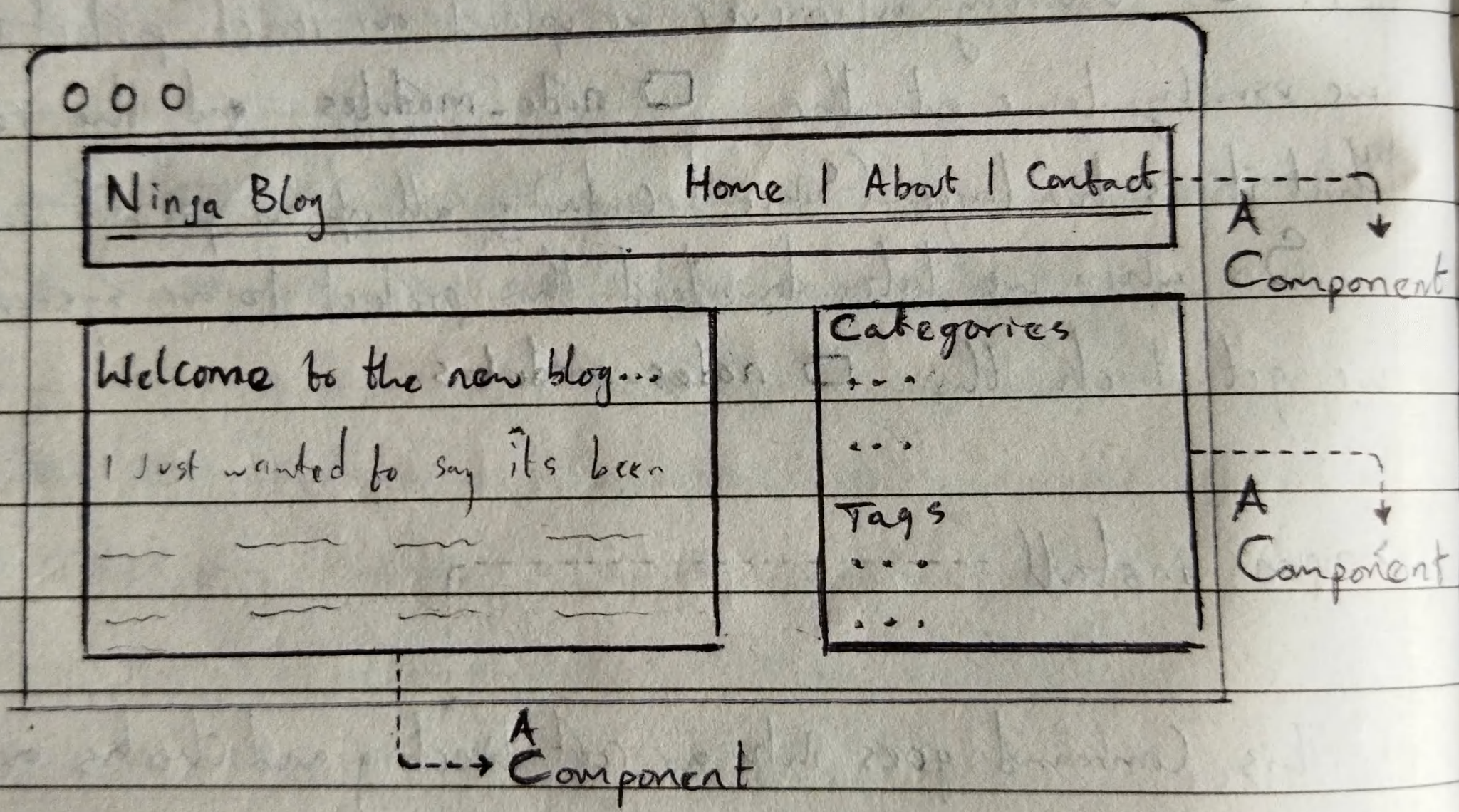
Components & Templates #3

What are Components?

Components are the building blocks and the beating heart of any react application.

A typical webpage built with react will be made up of several different components where each component is usually a self-contained section of content

E.g



It's our job as react developers to create these components which are rendered to the DOM and shown in the webpage

Components Contain:

- Template
- Logic

Dynamic Values In Templates #4

If we open up our App.Js Component we see that function App()

```
function App() {
  return (

    // returning something that looks like html

  )
}

export default App;
```

This is not actually HTML But JSX (Javascript XML)

What is JSX?

It is simply a syntax extension of Javascript. It allows us to directly write HTML in React.

NOTE: CLASS and CLASSNAME (in CamelCase)

In JSX we don't use the class attribute because it is a reserved keyword in ~~JSX~~ Javascript (and we are in App.Js)

Instead we use the className attribute in JSX and when it is converted to HTML the className becomes class.

NOTE: Returns In Components

Every Component must return some JSX ~~alternative~~ template.

NOTE: We always export our Components so we can use it in other files

Dynamic Values

An JSX in react, we are allowed to output dynamic values of variables inside it.

App.js

```
import './App.css

function App() {
  const title = 'Welcome to the new blog';
  const likes = 50;


  return (
    <div class Name = "App">
      <div class Name = "content">
        <h1> { title } </h1>
        <p> Liked { likes } times </p>
      </div>
    </div>
  );
```

Output:

Welcome to the new Blog

Liked 50 times

What Happen?

Whatever data type we pass inside the curly braces is converted into a string before it outputs it to the browser

We can pass a string, integer, an array But the only thing we can't output are booleans or objects

We can Also:

As well as outputting variables we can also write the dynamic values directly

```
<p> { 10 } </p>
<p> { "hello, ninjas" } </p>
<p> { [1, 2, 3, 4, 5] } </p>
<p> { Math.random() * 10 } </p>
```

```
...
10
hello, ninjas
12345
1.5313158881384 98
```

We can Also:

use dynamic values as attribute values in element tags

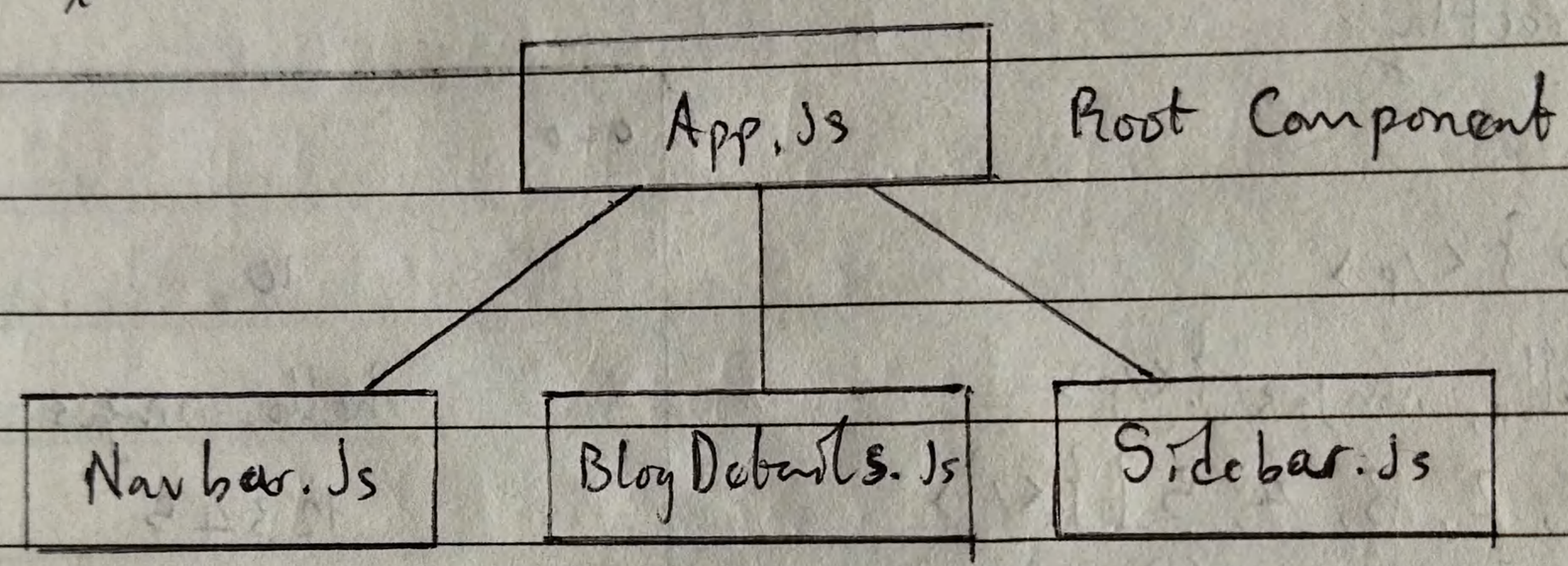
```
const link = "http://www.google.com";

return (
  <div class Name = "App">
    ...

    <a href = { link } > Google Site </a>
    ...
  </div>
)
```

MULTIPLE COMPONENTS #5

App.js

Now In our react application the App.js Component is known as the root Component.

It is the first Component that gets rendered to the DOM and seats at the top of ar application.



- Other Components are nested to it forming a Component tree.

! Remember : A component is just a function that returns a JSX template.

Project : Let's Create a Navbar Component

Navbar.Js

```
const Navbar = () => {
  return (
    <nav  class Name = "navbar">
      <h1> The Dojo Logo </h1>
      <div class Name = "links">
        <a href = "/" home </a>
      ...<a href = "/create"> New Blog </a>
```

Create
Arrow function also works

export default Navbar

Let's Import our Component now and nest them it

□ App.Js

```
import Navbar from './Navbar';      // No need to say .Js
...
function App():
  return (
    <div class Name = "App">        //or <Navbar></Navbar>
      <Navbar />                    # Nested
      ...
  )
```

Next: Create a Component for the homepage Content

□ Home.Js

```
const Home = ( ) => {
  return (
    <div class Name = "home">
      <h2> Homepage </h2>           #Let's output this for now
    </div>
  );
}

export default Home;
...
```

□ App.Js

```
import Home from './Home';
...
<div class Name = "Content">
  <Home />                          //Nested
</div>
```

The Dojo Blog
Home New Blog
Homepage

ADDING STYLES #6

How Do we Add CSS to our Project? CSS file

This can be achieved by importing the ~~Appcss~~ directory, Just like how react did with.

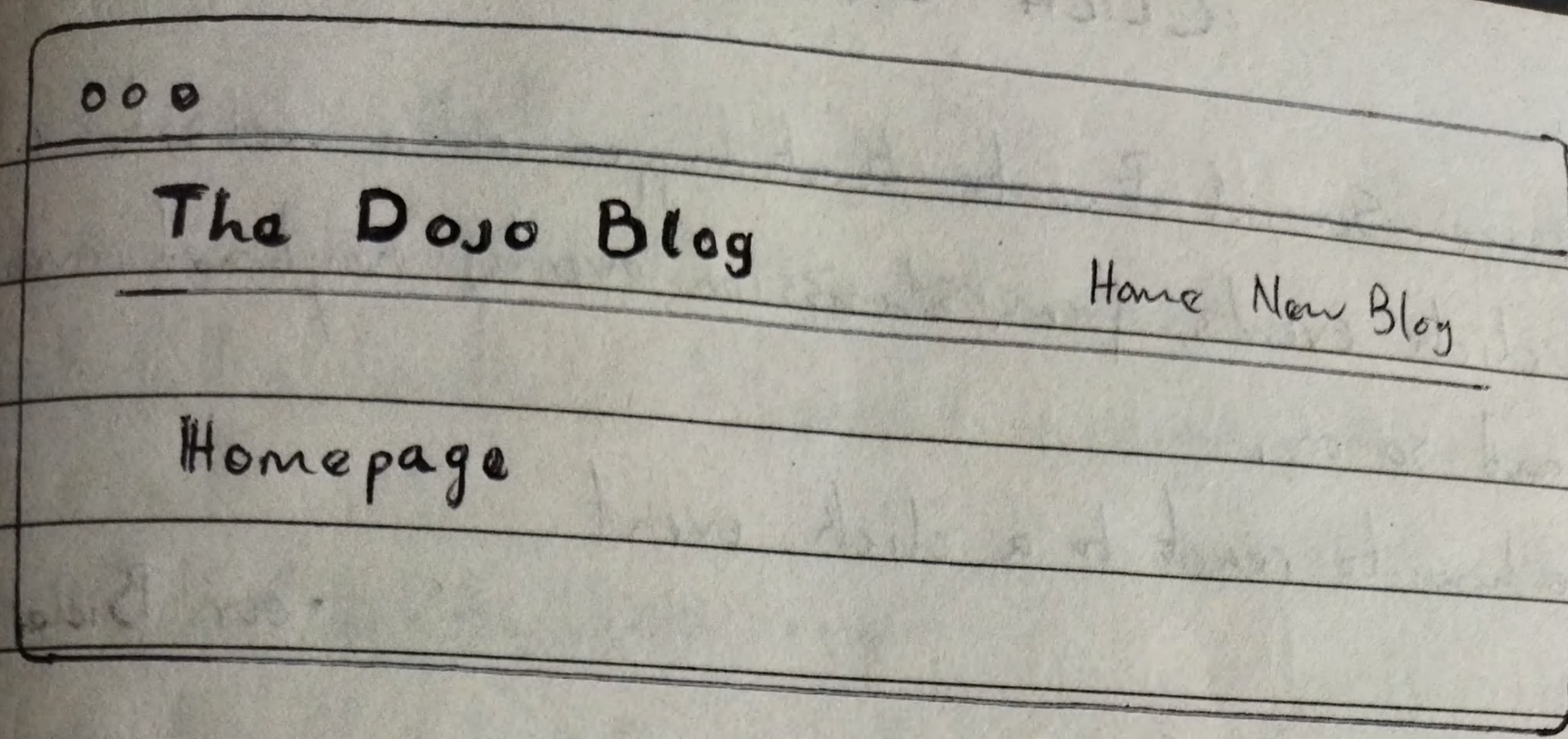
import './App.css'

Since we won't be using this any longer, remove it and delete the file.

We will be using the "index.css" for all our styles

index.css // Let's style

```
@import url('https://fonts.googleapis.com/css2?family=Quicksand...

/* base styles */
*{
  margin: 0;
  font-family: "Quicksand";
  color: #333;
}
.navbar
  padding: 20px;
  display: flex;
  align-items: center;
  max-width: 600px;
  margin: 0 auto;
  border-bottom: 1px solid #f2f2f2;
}

.navbar h1{ color: #f13566;

.navbar .links { margin-left: auto;

.navbar a {
  margin-left: 16px;
  text-decoration: none;
  padding: 6px;
}

.navbar a:hover{ color: #f13566}

.Content { max-width: 600px;
  margin: 40px auto;
  padding: 20px; }
```



Assuming: we wanted to style inline inside our component

E.g

dynamic value

```
<a href="/create" style={{
    color: "#fff",
    backgroundColor: "pink",
    borderRadius: "8px",
}}> New Blog </a>
```

Inline styles are done using Javascript Object Notation

or → #f1356d

We won't be keeping this inline styling so go ahead and remove

CLICK EVENTS #7

There are Several Events that happen on a website e.g hover events, click events, form submission events, keyboard events, scroll events and so on.

- Let's See how to react to a click event

- Home.js • our Bidarplate

```
const Home = () => {
  return (
    <div class Name = "Home">
      <h2> Homepage </h2>
      <button> click me </button>
    </div>
  )
}
```

NOTE: Except when we are adding dynamic values inside our return()

All Javascript code for a component is written b4 return() statement.

- Let's create the function for our button on Click Event

```
const Home = () => {
  ~~console.log('hello, ninjas');~~
  const handle Click = () => {
    console.log ('hello, ninjas')
  }
  return (
    ...
```

- Now Let's add our onClick Event

```
    <button onClick={ handle Click } > Click me </button>
  )
  ...
```

NOT

handle Click() -----> This Invokes the Function without being Clicked

NOTE: knowing that we can't ~~use~~ pass the handleClick() because it will invoke the function without the click Event.

What if the handleClick needed an argument to be passed

E.g.

What do we do?

```
const handleClickAg(a) => ...
const handleClickAgain = (name) => {
    Console.log('hello' + name);
}
return ( ...
    <button onClick={handleClickAgain('mario')}
```

This is the wrong way to do it since the function will be invoked without ~~Day~~ the button being clicked on

Correct Way: use ~~an~~ ~~unknown~~ anonymous function to call the handleClickAgain function and pass it an argument.

```
<button onClick={() => {
    handleClickAgain('Daniel')}>
```

Console: hello Daniel

called function

NOTE: By default each event has access to ~~an~~ default parame
- ~~ter~~ popularly known as e.

e: has alot of methods we can use.

E.g

```
const handleClick = (e) => {
    Console.log(e)
}
...

<button onClick={handleClick}> click me </button>
```

Console: SyntheticBaseEvent {_reactName: 'onClick', _targetInst: null, ... and more

- To do this for an event called function with parameters

E.g
```
const handleClickAgain = (name, e) => {
    console.log(name, e)
}

<button onClick={(e) => handleClickAgain('Daniel', e)}>
Click me Again </button>
```

Console: Daniel SyntheticBaseEvent { ...
...

USING STATE (useState hook) # 8

What is the State of a Component?

This is simply the data being used in that component in that point in time.

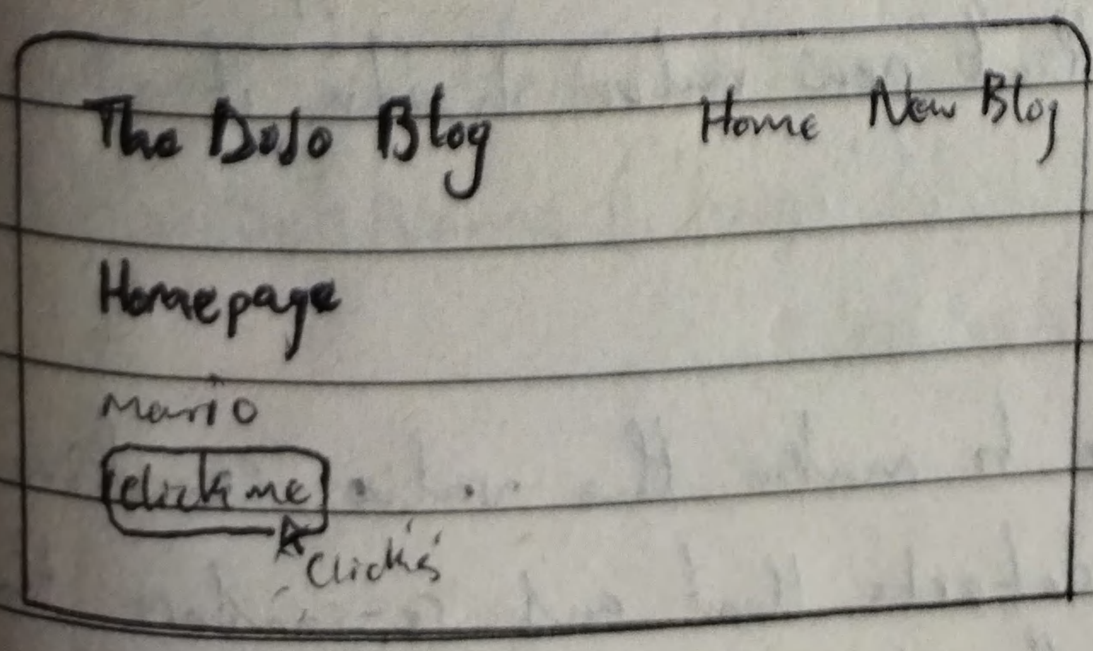
The state could be an Array of values, boolean's, strings, objects or any of the Data that our component uses.

- To Demo this:

Lets say we have a variable with a data we want to change over time or in reaction to some event.

- Home.Js

```
Const Home = () => {
  let name = 'mario';                    # variable set
                                  name = 'Luigi'; }
Const handleClick = () => {              # Event handler Set
return (
  ...
  <p> { name } </p>
  <button onClick = { handleClick } > click me </button>   # Event Listener
)
```



NOW:

If we click this button, Mario doesn't change on the template

- Let's say we add a Console.log(name) and see if variable content changes

```
...
{
  name = 'luigi';
  Console.log(name)
}
...
```

click me
clicks

CONSOLE
luigi

CONCLUSION

From the above we see that name becomes luigi onClick but it doesn't update in the template.

mario
Clicks me

The same after click

Why Is that?

This is because the variable we created `let name = 'mario';` is not reactive.

Which means that React doesn't watch if it changes nor react to it changing

So whenever the variable changes it doesn't trigger React to re-render the template with that new value inside it

How Will this Work?

For this to work we have to make the value reactive. So that whenever it changes React detects that and re-renders the template where the value is on the browser.

How do we make this value of ours reactive?
We do these in React using something called a Hook. and that hook is called

use State

Note: you can tell a hook by it's name because it always starts with "use".

So the use State hook gives us a way to make a reactive value and also provides us with a way to change that value whenever we want.

0 Home.Js

```
import {useState} from 'react';          # useState hook imported!

const Home = () => {
// let name = 'mario';                   # Comment Out
const [name, setName] = useState('mario');

const handleClick = () => {
    setName('luigi');
}
```

Explained!

The use State hook returns 2 values so we use array destructuring to grab them

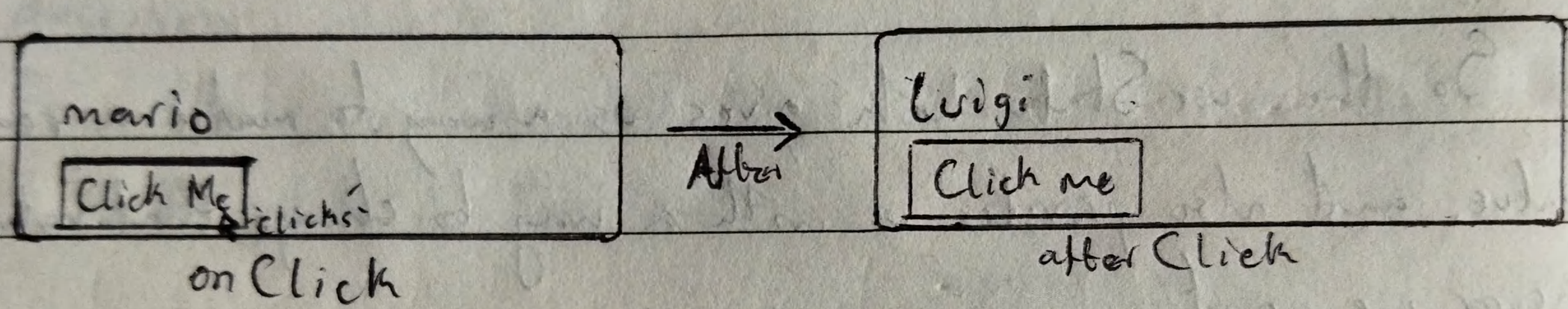
name = all the argument inside useState(...)

setName = a function for changing anything inside useSt

name: this variable if used will get us whatever the value of our useState is. In this case it will be "mario".

setName
~~useState~~: If we wanted to change the value in our useState. we would use the second variable which is a function.

Now: If we try this code we will see that it changes in the browser



TAKE AWAY

With all this said. When you need a reactive value, something that might change at some point, we use the "useState" hook to do that.

In [name, setName]

int, string, object, array, etc.

name (first variable): will be the entire value passed in useState(—)

setName: will be used to update the values in the useState and then trigger a re-render

INTRO TO REACT DEV TOOLS #9

What does React Developer Tools DO?

This Integrates with the browser development tools to give us extra features that we can use on any website created with React.

If we take a look at the browser developer tools navbar, we can see 2 new additions: Components and Profiler

Outputting List #10

Now what we are building is a mini simple Blog... So Ideally we are going to need to have a List of Blogs to output in a template.

To do this we will be using the "useState" hook to create the data since the data might change at some point in time. (eg Like we might delete a blog).

- Let's Create a useState with Blogs. #Clear home.js. Let's Refactorit
- Home.js

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

const Home = () => {
  const [blogs, setBlogs] = useState([
    { title: 'My new website', body: 'lorem ipsum...', author: 'mario', id:1 },
    { title: 'Welcome Party!', body: 'lorem ipsum...', author: 'yoshi', id:2 },
    { title: 'Web dev top tips', body: 'lorem ipsum...', author: 'mario', id:3 }
  ]);

  return (
    <div class Name = "home">

    </div>
  );
}export default Home;
```

How do we output our Blog Data now in the Template?

If we think about it we need a template for each blog and one thing we could do is hard code it in our Home.js Template.

But hard Coding is not the best route because we might happen to have more blog objects in our useState

BEST WAY (PRACTICE)

We want to instead use some Javascript to cycle through our list of objects and then output a bit of template for each Blog.

So we will use the map method in Javascript which function is basically to cycle through an Array and allows us to do something with each item in the Array.

```
id:1 }, Home.js
id:2 },  ...
         return (
id:3 }     <div class Name = "home" >
             { blogs.map ( (blog) => (
               <div class Name = "blog-preview"  key={blog.id} >
                 <h2> { blog.title } </h2>
                 <p> Written by { blog.author } </p>
               </div>
             )) } ;
           </div>
         );
```

- Now Let's Style our Blog Previews
- Index.css

```
.blog-preview {
    padding: 10px 16px;
    margin: 20px 0;
    border-bottom: 1px solid #fafafa;
}

.blog-preview:hover {
  box-shadow: 1px 3px 5px rgba(0,0,0,0.1);
}

.blog-preview h2 {
  font-size: 20px;
  color: #f1356d;
  margin-bottom: 8px;
}
```

DONE ☑

The Dojo Blog Home New Blog

My new website
Written by mario

Welcome party!
Written by yoshi

Web dev top tips

PROPS & REUSABLE COMPONENTS #11

If you think about it, if we were building a real blog website. We might have the List of Blogs in various places on our website. It might be the Homepage, Search page, Category page and so on.

So there is going to be soo many areas where we are cycling through the Blogs and then outputting a Blog Preview.

So if we were to do this, we would essentially be repeating the code in our Home.js over and over and over again in different components and for different pages.

BEST PRACTICE

The best way to go about this is wherever we have bits of templates that might be re-used in different positions or places in the website.

It is best to make that bit of template into it's own re-usable Component.

E.g: If we were to make a Component called "BlogList.js", then we would be able to drop that BlogList.js Component in any other Components in a project.

- Kinda like how a function works when programming (DRY).

And this is where Props come in

WHAT IS/ARE PROPS?

Props are arguments passed into React Components. They are passed to components via HTML attributes. Lastly Props stands for properties.

- Let's now Create our BlogList.js
- BlogList.js

```
const BlogList = () => {                    // Copy Blog Component from Home.js
                                               into Here
  return (
    <div class Name = "blog-list">          // NOTICE: This Convention
      {blogs.map ((blog) => (
        <div class Name = "blog-preview" key = { blog.id } >
          <h2> { blog.title } </h2>
          <P> Written by { blog.author } </P>
        </div>
      ))}
    </div>
  );
}

export default BlogList;
```

NEXT: The next thing we need to do is nest this BlogList.js component inside our Home Component.

0 ~~BlogList.js~~ Home.js

```
import BlogList from './BlogList';

const Home = () => {
  const ...

  return (
    <div class Name = "home">
      <BlogList />
    </div>
  );
```

⊗ If we run our code we see that this won't work.

☑ For this to work we have to use props to pass the blogs data from our Home Component.

- So Props are a way to pass data from a parent Component into a child Component.

```
<div class Name = "home">
  <BlogList blogs = { blogs } />
</div>
```

This is a Prop

// Props used ☑

- One Last thing: We need to go over to our Bloglist and ~~use Props~~ access props.

// Props : here is an object. So to access our passed in props called blogs, we use the . notation

- BlogList.Js

```
const BlogList = (props) => {
    const blogs = props.blogs;          // blogs gotten

    console.log (props, blogs);         # Let's see what props and blogs are
    ...
}
```

CONSOLE:

```
▸ { blogs : Array (3) }
▸ (3) [{...}, {...}, {...}]
```

- Let's pass In another Props
- Home.Js

```
...
return (
    <div class Name = "home">
        <BlogList blogs = {blogs}  title = "All Blogs" />
    </div>
);
```

Another Props

- BlogList.Js

```
...
    const title = props.title;

    return (
        <div class Name = "blog-list">
            <h2> {title} </h2>
            ...
```

NOTE:

We could also use Array, Object Destructuring to recieve the passed in props e.g

```
= ({ blogs, title }) => (
...
```

RE-USING COMPONENTS #12

- Let's see how we can re-use our BlogList Component to get a specific persons blogs
- Home.js

```
...

  return (

...

...

<BlogList blogs={blogs.filter(blog => blog.author ===
        "mario")} title="Marios blogs" />
</div>

...
```

// This is how we can re-use Components

```
ooo

All Blogs!
  My new website
  written by mario
  Welcome party!
  written by yoshi
  Web dev top tips
  written by mario

Marios blogs
  My new website
  written by mario
  Web dev top tips
  written by mario
```

FUNCTIONS AS PROPS (delete an item) #13

So now what if we want to allow users now to delete this Blog by clicking on a "delete blog" button.
We would need to create a button with an onClick Event that Calls a function and passes in the blog id.

BlogList.Js

```
return (
  ... <p>Written by {...}</p>
      <button onClick={() => handleDelete(blog.id)}> delete blog
      </button
  </div>
... )
```

But we haven't Created this function yet.
So let's do that.

We Could Create the handleDelete(id) function in BlogList. But Best Practices demands we create this function where the state is. So let's go over to Home.Js

Home.Js

```
...
const [blogs, setBlogs] = useState([ ... ])

const handleDelete = (id) => {
  const newBlogs = blogs.filter((blog) => blog.id !== id)
  setBlogs(newBlogs)
}
...
```

So what happens in our handleDelete is that we use the filter method (which returns an Array) to get blogs that there id don't match the blog id to be deleted.

Then after we've gotten a filtered Array with the Blog to be deleted not included, we setBlogs (newBlogs) ☑

Lastly:

We need to pass this function as a props to BlogList.js in order for it to work

☐ Home.js

```
...
return (
    ...
        <p> Written by {...} </p>
        <button onClick =

return (
    ...
    <BlogList ... ... handleDelete = {handleDelete} />
    ...
    )
```

☐ BlogList.js

```
Const BlogList = ({ blogs, title, handleDelete}) => {
```

My new website
written by mario
[delete blog]

Welcom Party!
written by yoshi
[delet blog]
=Click' and it is removed!

passed In Successfully
Now we should be able to Delete ☑

useEffect Hook #14

So far we've seen 1 Hook in this course which is the useState Hook. But there are many other hooks in react and one such hook is called the useEffect.

What does the useEffect do?

This hook runs a function every render of the component / it runs a function on page load and change of state.

In short "This useEffect hook is a way to run code on every render".

- Home.js

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

const Home = () => {
  ...

  useEffect(() => {
    console.log('useEffect ran')
  });
```

Usually inside useEffect, we could do something like fetch data or communicate with some kind of authentication service.

These things are known as "Side Effects" in React.

NOTE: useEffect returns nothing so we don't put it inside a variable

CONSOLE: If we refresh or delete blog

② useEffect ran

NOTE: One thing to be careful of is actually changing the state inside a useEffect because (think about it) you could end up in a continuous loop of renders. There are ways around this but we will look into that later on

useEffect DEPENDENCIES #15

Our UseEffect current runs on any render of the template

Now we don't always want to run a function after every render, maybe only certain renders but definitely not every single one.

To run a function only at certain renders we can use something called a "DEPENDENCY ARRAY".

We use the Dependency Array by passing it as a second argument inside our useEffect

0 Home.Js

```
...
  useEffect(() => {
    Console.log('use effect ran')
  }, [ ]);
...
```

→ Second Argument: Passing in an Empty Dependency Array ~~thus~~ tells the useEffect hook to only run once and that's on the first initial render.

```
Const [name, setName] = useState('mario');        # Create this
...
```

⚙ What if we add actual dependencies?

```
useEffect (() => {
    console.log ('use effect ran')
}, [name]);                          // dependency Added!
...
```

→ Now this useEffect will run only when the state of our Dependecy "name" changes

Now Let's add a button and output the name. Next we add an onClick event that changes the state of our useState "name"

```
return (
  <div class Name="home">
    <BlogList ... />
    <button onClick={() => setName('luigi')}> change name
    </button>
    <p> { name } </p>
  </div>
```

DONE ☑

Now: useEffect is going to watch [name], if it changes when we click on button we should see in our console | useEffect ran

CONSOLE	
>	All Blogs!
	Web dev top tips
	written by mario
useEffect doesn't run	delete blog ← clicks
use effect ran	Change name ← Clicks
	Mario
useEffect now runs since the "name" state changed from mario → luigi	↓
	Luigi

Note: Our useEffect also ~~runs~~ gets called fires on page load

Using Json Server #16

Now we know how to use the useEffect hook, we are going to be using it to fetch some data.

Now useEffect is a good place to fetch data in a component because we know that it runs the function when the Components first renders initially (if we passed in an Empty Dependency Array).

And then we can use that data in our application instead of the hard-coded data that we have in our Home.js (const [blogs, setBlogs] = ...).

In a real world Application our data would usually come from a data-base using an api (Instead of hard Coding it).

What Are We Going to be Doing?

We will be using a package called JSON server which is going to allow us to build a fake REST Api just using a Json file that we can use to TEST this.

First Steps

- Create a folder in the root directory and name it data
- Inside data create a file named it db.Json
- Now open db.Json and paste the below

data

~~db.Json~~ > {...} db.Json

```
{
  "blogs": [
    { "title": "My First Blog",
      "body": "Why do we use it? \n It is a long established fact ...",
      "author": "mario",
      "id": 1
    },
    {
      "title": "Opening Party!",
      "body": "Why do we use it? \n It is a long established fact ...",
      "author": "yoshi"
      "id": 2
    }
  ]
}
```

Note: When we are using JSON server, each top-level property is considered a "resource".

Therefore JSON server sees "blogs" above as a resource and it creates endpoints for us to interact with this resource. So we can do things like delete items from it, add items to it, Edit items, Get the items e.t.c.

NEXT: the next thing we need to do is use the JSON server package to essentially watch this file and wrap it with some endpoint.

There are 2 Options here
- We can install the JSON server package locally into our project
- We can use npx to run the code from the web and still watch our file

we will go with Option b.

TERMINAL

```
> npx Json-server --watch data/db.Json --port 8000
```

This should do it. **Note**: The reason we ~~added~~ specified a port number is that by default the Json-server tries to go on port 3000 where our React App local development server is running on.

If we check the terminal we should see our Json-server endpoint

TERMINAL:

```
Resources
http://localhost:8000/blogs

Home
http://localhost:8000
```

If we paste this endpoint in the ~~web~~ browser we will actually see all of our JSON DATA. But we won't be using the Browser for this though.
We will be using a fetch request inside our Component using the endpoints.

Some Of The Endpoints we will be using are

EndPoints

/blogs	GET	Fetch all blogs
/blogs/{id}	GET	Fetch a single blog
/blogs	POST	Add a new blog
/blogs/{id}	DELETE	Delete a blog

NEXT UP: We will see how to make a fetch request in useEffect to fetch all of the blogs

Fetching Data in useEffect #17

Now that we have our Jason-server running, Let's try to fetch our data from db.json in useEffect.

We want the useEffect to fetch our data when the Component first renders so we will use an empty dependency array.

So before we do this let's empty our useState (since we won't be using that data anymore) and set the initial value to null.

- Lastly Let's also remove our deletBlog functionality
- Home.Js
- Let's now Implement our fetch

```
const Home = () => {                    //NOTE: comment out <BlogList ...>
  const [blogs, setBlogs] = useState(null);

  const handleDelete = (id) => {        // Delete this and all its functions

  useEffect(() => {
    fetch('http://localhost:8000/blogs')
      .then(res => {
        return res.Json()
      })
      .then(data => {
        Console.log(data)
      })
  }, []);   // So this useEffect runs only on the first initial render
```

Refresh Browser

CONSOLE : ▶ (2) [{...}, {...}]

Now that we see that the data is successfully fetched. Let's add the data to our blogs.

```
useEffect(() => {
...
.then(data => {
  //console.log(data);
  setBlogs(data);
});
...
```

= Uncomment <BlogList ...> and run

An Error When we run (Type Error: cannot read property map of null

Why Does this Error Occur?

This error occurs because on the initial render of the template <BlogList ...>

Blogs is set to null so therefore we can't map through null value

But blog is supposed to be data on initial render?

This is ~~through~~ brug but it takes some time to fetch that data because of that when ~~blog~~ our BlogList Component is called blogs is still set to null.

How Do We Fix This Annoying Error?

We do a little dynamic checking that doesn't output our blogs ~~unless~~ until we have a value for blogs (not null).

- Home.js

```
...

return ~~Catalog Name~~
  <div class Name = "home">
    {blogs && <BlogList blogs = {blogs} title = "All Blogs!" />}
  </div>
```

→ Logical And

≡ If we Refresh our Browser now, we see

```
All Blogs!
My First Blog
written by mario

Openning Party!
written by yoshi
```

☑

Why does this Work?

Logical And evaluates the left first and if this is false it never bothers with the right.

NEXT UP: We will see how to output a "Loading message" as we are trying to fetch the data.

CONDITONAL TEMPLATES #18

Now Currently In our application we render the BlogList once we have blog data.

```
{blogs && <BlogList blogs={blogs} title="All Blogs!" />
```

WHAT WE WANT TO DO?

We want to create a loading message whilst the data is being fetched So that our users know something is loading if it takes time to do.

HOW DO WE DO THIS?

We create an additional/new state for this. Let's do that

Home.js

```
...
const Home = () => {
  ...
  const [isPending, setIsPending] = useState(true);  #Create

  useEffect(() => ... );

  return (
    <div className="home">
      { isPending && <div> Loading... </div> }  #Add
      {blogs && ... }
    </div>
  );
}
```

PREVIEW IN BROWSER

Loading...

All Blogs!

Now we see our Loading works and it shows before the BlogList shows up. The only problem is that it still shows after BlogList shows up and we don't want to see it after BlogsList shows up.

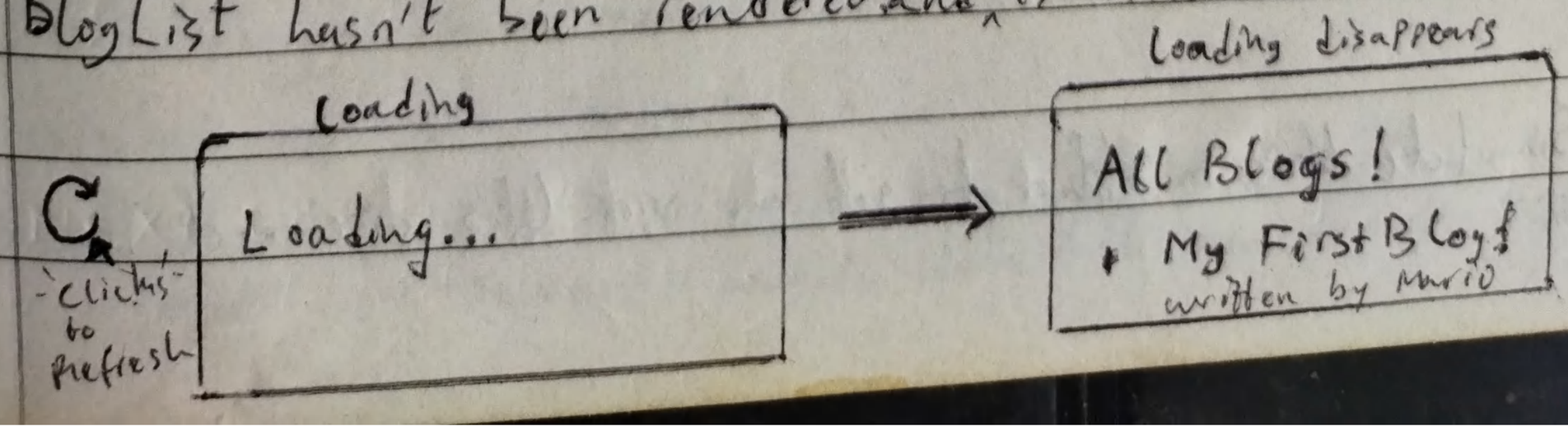
HOW DO WE STOP THAT?

We can do this by setting isPending to false after setting data to blogs

Home.js

```
...
const Home = () => {
  ...
  useEffect(() => {
    ...
      ...
      .then(data => {
        setBlogs(data);
        setIsPending(false);        // Added
      }
```

Now if we refresh Browser Loading should only show when BlogList hasn't been rendered and blog is still null.



HANDLING ERRORS # 19

One more thing we want to do is handle any kind of error when we try to make our fetch request.

It could be an error sent back from the server or it could be a connection error.

Whichever case it is we want to let the user know that there is some kind of error

So how exactly do we do this?

- Home.js
  - First off Let's add a catch

```
...

useEffect(() => {
  fetch(...)
    .then ...
    .then ...
    .catch(err => {
      console.log(err.message);
    })
}, []);
```

- To simulate this in order to get an error let's disconnect from the server

- Open Up the Terminal and Cancel out of our Json-server process.

TERMINAL:

```
[]                                    //Press Ctrl + C
^C Terminate batch Job (Y/N)?
PS C:\Users\Daniel Chukwu\            // Disconnected Successfully
```

≡ Now if we refresh our browser we should see an error logged out on the ~~Browser~~ CONSOLE

CONSOLE:

```
Failed to Fetch
```

This is a CONNECTION ERROR

What if there is another type of error e.g Imagine our request reaches the server but the server sends an error back.

Maybe ~~the~~ the endpoint that we tried to fetch from doesn't exist or if the request is denied. In such a case then our catch block doesn't automatically catch the error.

NOTE: .Catch only catches Connection errors

→ HOW DO WE CATCH SUCH ERRORS?

Before we learn how to catch these errors let us log to the console our initial fetch response.

```
npx Json-server --watch data/db.Json --port 8000
```

Home.Js

```
fetch ('http://localhost:8000/blogs')
  .then (res => {                          //
    console.log (res)                      // Added : Let's see this in console
    return res.Json();
  })
```

```
Response {type: "cors", url: "..."
... ... ... ... ... }
...
ok: true
...
```

NOTICE

The ok (okay) property is true. That means that the fetch was okay and that we got data back.

Now if we don't get data back, for example if we send a request to an endpoint that is faulty or doesn't exist then our ok is going to be **false**.

Now: To answer the earlier question how do we catch such errors?

We can do so by checking if res.ok is false. If it is then we want to throw an error.

So Let's Implement this ...

Home.js

```
...
fetch('http://localhost:8000/blogs')
  .then(res => {
    if (!res.ok) {                    // Says: if response.okay == false
      throw Error('could not fetch the data for that resource');
      //New error message
    }
    return ~~response~~ res.Json();   if this executes This won't be called
...
  .catch(err => {
    Console.log(err.message)          // Consoles out the new Error message
  })
...
```

jumps to

This won't work right now if we run though and thats because our fetch okay status is true.

If we fault the fetch endpoint by add extra characters, we should see this error.

```
? fetch('http://localhost:8000/blogs55')
  //incorrect/invalid endpoint
```

Now that we have a faulty endpoint if we run (save). The fetch makes a request to the server.

The server sends back an error, therefore res.ok == false will be and our error will be thrown

CONSOLE: Could not fetch the data for that resource

NEXT: We want to store the error in some kind of state so that we can output it to the browser

Home.js

```
const Home = () => {
  ...
  const [error, setError] = useState(null);          // Added
  ...
  useEffect(() => {
    ...
    .catch(err => {
      setError(err.message);                         // Added
    })
  ...

  return (
    <div class Name = "home">
      { error && <div>{ error }</div> }              // Added
      ...
```

RESULT:

We should see this working now if we run (save)

Could not fetch the data for that resource
Loading...

☑ ☐ We don't want to see Loading

Now we don't want to see Loading... if we have an error. Because it's not actually still Loading.

WHAT DO WE DO?

So if we do get an error we want to set is ~~Loading~~ Pending to false so it doesn't show that loading message

...

```
.catch(err => {
    set IsPending(false);          // Added
})
...
```

Now if we try to get data again (make a subsequent request) we ~~want~~ don't want the Error message showing up

...

```
.then(data => {
    ...
    ...
    ...
    setError(null);                // Added
})
...
```

CONCLUSION: If we refresh the browser with a faulty endpoint we should get only the Error message but then if we fix the endpoint and try again we should get our blogs and no Error Message in sight.

☐ Save and refresh faulty Endpoint

Could not fetch the data for that resource

→

☐ Fix faulty Endpoint & Refresh

All Blogs!

My First Blog

Written by mario

Creating A Custom Hook #20

So we've put together 3 ~~useef~~ useState Hooks and 1 one useEff to either display the data, or a loading message or an error if th is one and everything is working fine.

And it is fine to be left like this, But Just what if? we want to do this same kind of thing in another Component in the futu

Where we fetch some data, create states, set errors and all that, We would essentially have to write that all over again.

That's not very efficient and not easy to manage especially if we are copying and pasting and using this code in several Components

It would be Best Practice / Good if we Could make this code re-usable in different components.

How Do We Go About This?

We could and should rip the code all out of our Home Compone And put it in it's own Javascript file and then we Just Import that into any component that wants to use that in the future.

In this way we are only writing and maintaining the Code in one place.

"When we do Something Like this. We are Creating a Custom Hook"

Let's Get started with our Custom Hook Creation

- src
  - useFetch.js

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

const useFetch = () => {
  const [data, setData] = useState(null)
  const [isPending, setIsPending] = useState(null)
  const [error, setError] = useState(null)

  useEffect(() => {
    setTimeout(() => {
      fetch('http://localhost:8000/blogs')
      .then(res => {
        if(!res.ok) {
          throw Error('could not fetch the data for that resource');
        }
        return res.json();
      })
      .then(data => {
        setData(data);
        setIsPending(false);
        setError(null);
      })
      .catch(err => {
        setIsPending(false);
        setError(err.message);
      })
```

// blogs and setblogs renamed to be more generic and covering

```
    }, 1000);                    // setTimeout closer
  }, []);                        // useEffect closer
}                                // Custom Hook: useFetch closer

export default useFetch;
```

NOTE: the reason we named this useFetch is because Custom Hooks in React needs to start with a "use" and then something.

What we want to do next is return some value because right now we are not returning anything

```
...

const useFetch = () => {
  ...

  return { data, isPending, error }    // Returning an Object
}
```

THE REASON WHY WE ARE RETURNING AN OBJECT?

This is because whenever we use this useFetch hook on other components in the future.

We want to be able to use object destructuring to grab whatever return value we want (1, 2 or All) and in whatever order.

LASTLY: We want to pass in the endpoint (URL) into the useFetch hook rather than to hard code it (as we currently are).

- useFetch.js

```
...

const useFetch = (url) => {                    //updated
  ...
  ...

    fetch(url)                                 //updated
    ...
```

DONE ☑

- Now what we want to do is Import and use our useFetch in our Home.js

- Home.Js

```
...
import useFetch from './useFetch';             //Added


const Home = () => {
  const { data, isPending, error } = useFetch('http://localhost:8000/blogs')
        update to ---> data: blogs   //This means grab the data but call it blogs

  return (
    <div class Name = "home">
      { error && <div>{ error }</div> }
      { is Pending && <div>Loading... </div> }
      { blogs && <BlogList blogs={blogs} title = "All Blogs!" /> }
    </div>
                    // Test this out and see that it works just as normal
  );
}
```

THE REACT ROUTER #21

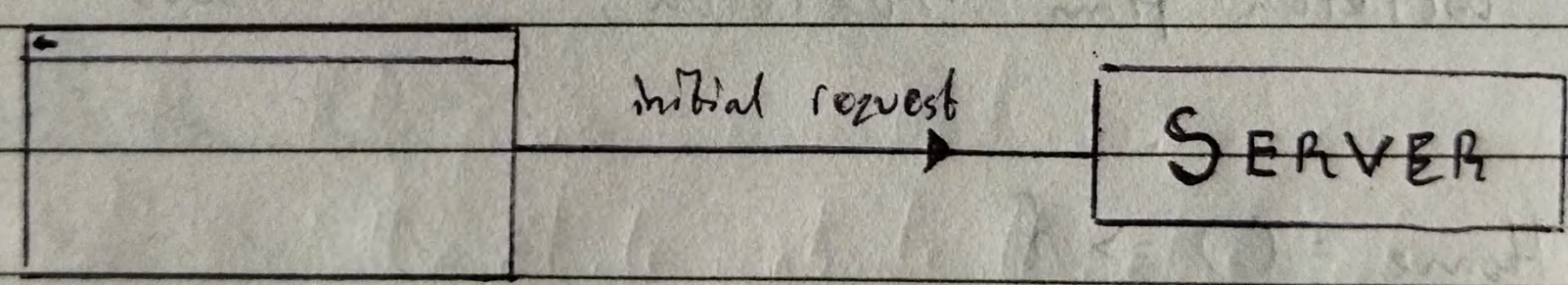
So far, our application only has one single page. We don't currently navigate to other pages.

And most websites that we create are probably gonna have more than 1 page. So therefore we need a way to introduce multiple different pages and routes in our React Application.

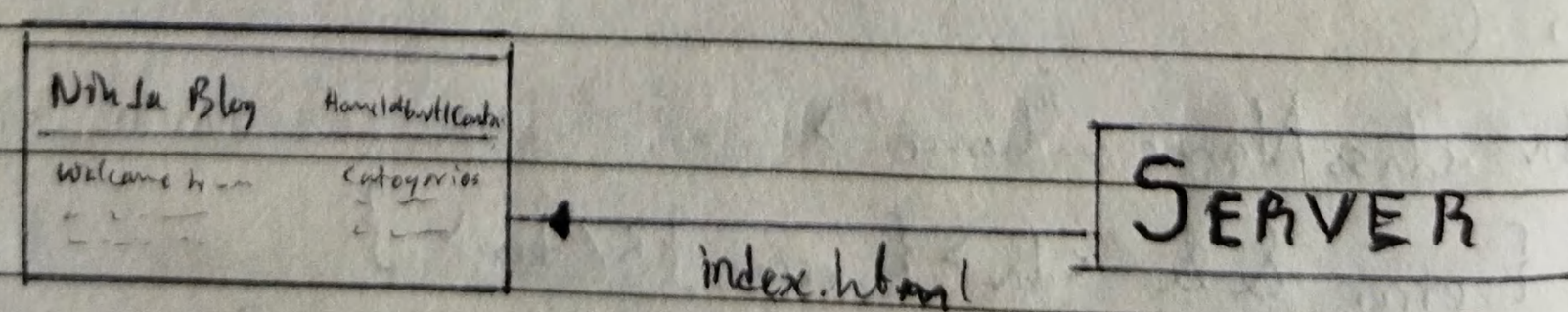
"The way we do this in React is through the React Router"

Non-React Websites

How does a typical website thats not using reactt handle multiple pages



- We visit the websites URL which sends an initial request to the server and the server handles the request and sends a response



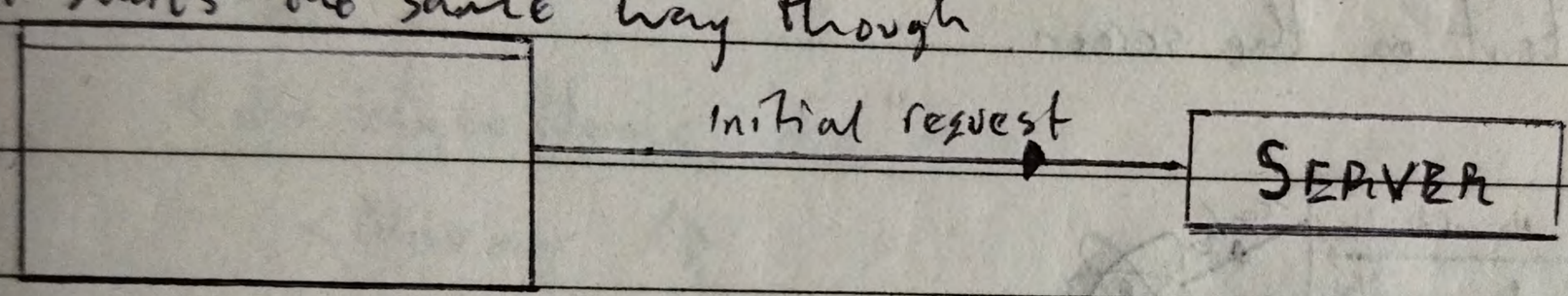
- And then we see the page on our browser

So if a user was to send a brand new request (to let's say the contact page) and then the server handles it and sends a response. And the cycle would continue like that.

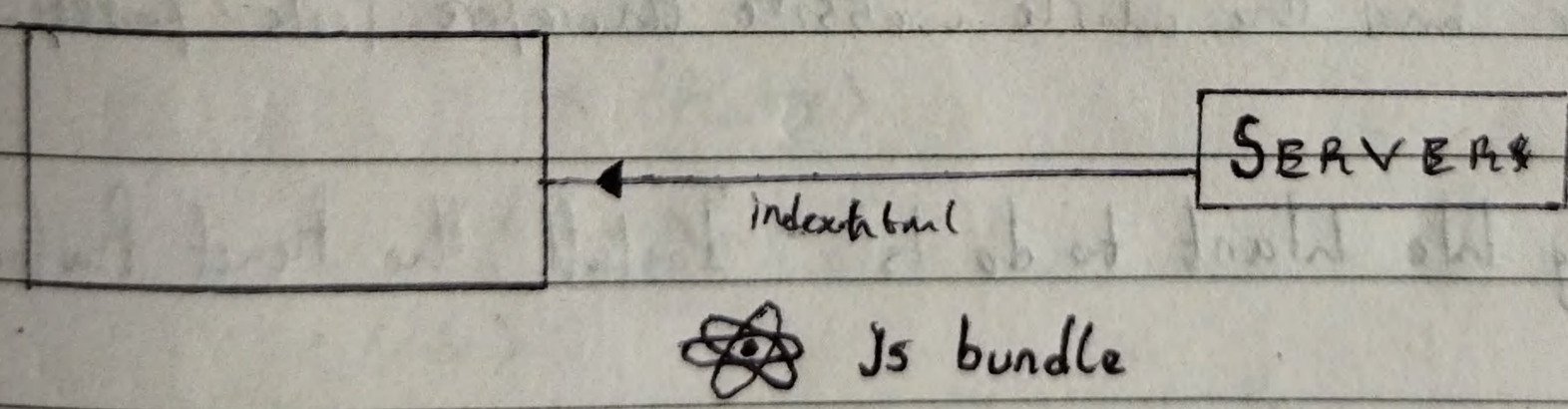
React Websites (SPA's)

Now react applications doesn't work like the above, they delegate all the routing and changing of page content to the browser only

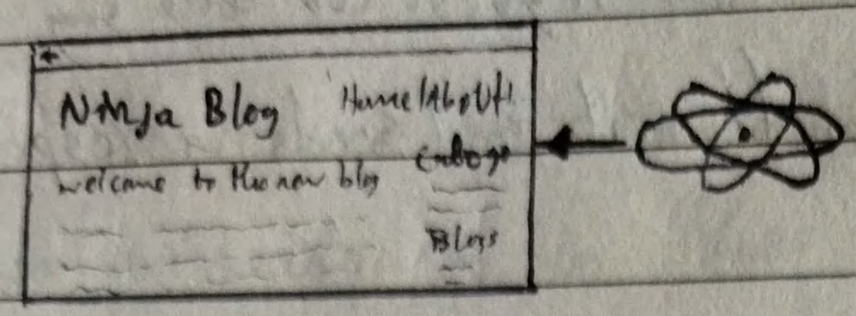
- It starts the same way though



- The Server then responds to that by sending back the html page as well as our Compiled React Javascript files (which controls our React application).



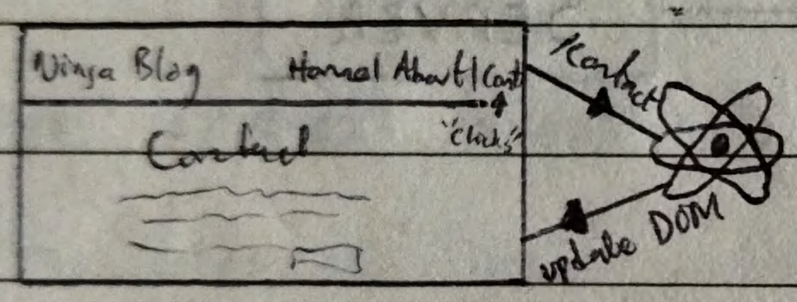
- From this point on React and the React Router can take full control of the application.
- Initially the html page that we get back is empty and then React injects the content dynamically using the Components that we created.



React Injects Components

If we click on a link to another page, the React Router intercep the request from going to the server.

And instead it's going to look at the new page request and inject the required Content on the screen.



Therefore this whole process means that we are making less request to the ~~browser~~ server and the whole website therefore feels faster.

First Thing We Want to do Is - Install the React Router Packa

TERMINAL :	> npm install react-router-dom @ 5

We Install version 5 because version 6 isn't stable just yet and is still in beta.

- Let's Start the Routing for our application
- App.Js

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

≡ Now we want to surround our entire application with the Router Component

```
function App () {
  return (
    <Router>                                   // Added new wrapper
      <div class Name = "App">
        <Navbar />
        </div class Name = "Content">
          <Switch>          // in V6 <Routes>
            <Route path = "/">
              <Home />
            </Route>
          </Switch>
        </div>
      </div>
    </Router>
  );
}

export default App;
```

ADDED PARTS

NOTE : The <Switch> makes sure that only one (1) route shows at any 1 time

☑ DONE

Check, run (Save) to see that our routing works.

More Routes & Exact Matches #22

Now we've our Router set up let's try adding another route
To do this we need another page Component.

And this new page Component is going to be for a web form
that helps users add a new blog

- Create.Js // created ⊙ Form Boilerplate

```
SFC
const Create = () => {
  return (
    <div class Name = "create">
      <h2> Add a New Blog </h2>
    </div>
  );
}
export default Create;
```

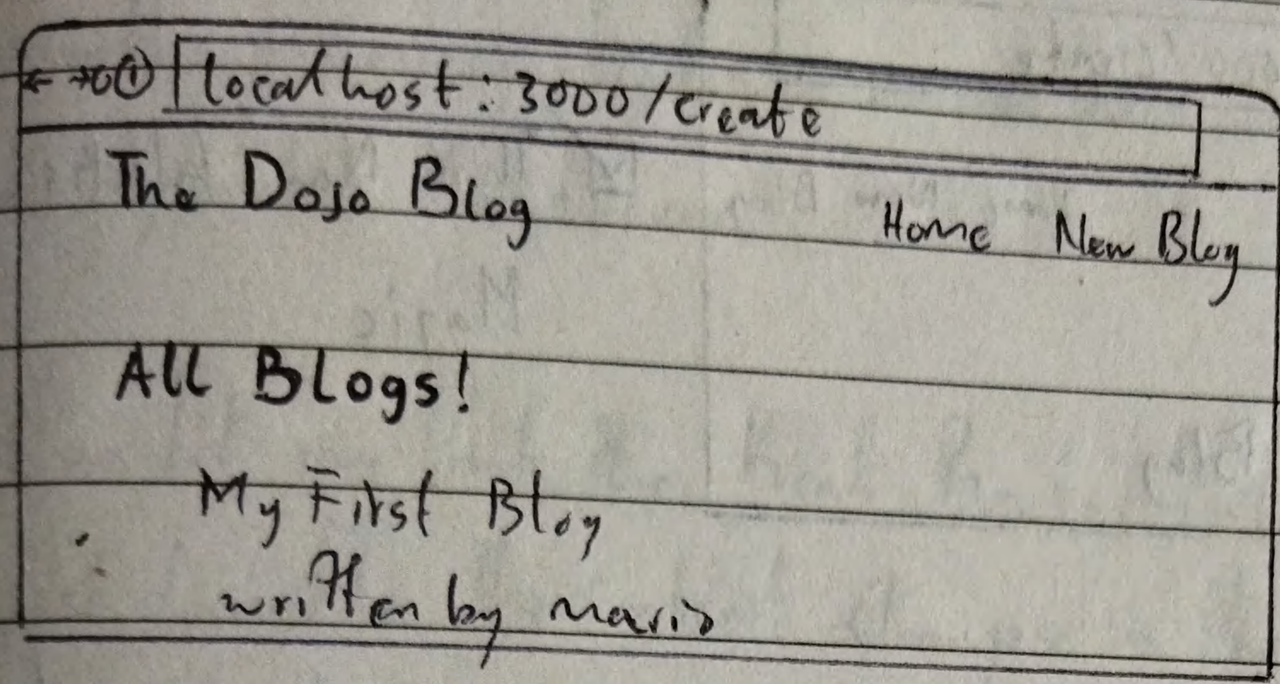
≡ Now Let's add a new route to /create

- App.Js

```
<switch>
  ...
  <Route path="/create">
    < Create />
  </Route>
  ...
```

Added Route

☒ But when we visit the route we end up at our home



☐ The route doesn't take us to our component. Since we are still in Home

Why Doesn't The ~~Rage~~ Route Take us to Home?

This is because goes into our switch and sees the path of our first route "<Route path="/">" as a match.

Because "/" is inside the Route "/create" we are trying to go to

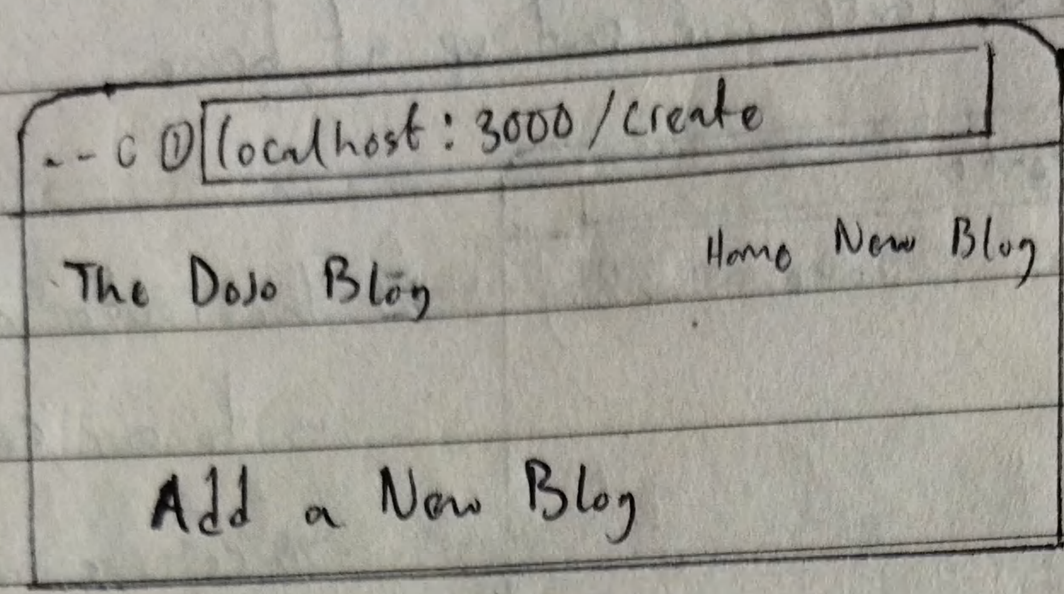
How Do We Fix This?

We can tell React look, I don't want you to match this if it is including in our Route, I want you to match this if it is exactly the same as our Route. We do this by adding "exact"

☐ App.js

```
<Switch>
  <Route exact path="/">
    <Home />
  </Route>
  ...
</Switch>
```

☑ This should now work



☑ This New looks Like Magic

NOTE: It is always a good idea to wrap our Routes inside the <Switch> ... </Switch> tag.

This helps us access one Route and wherever we want to access a particular Route it searches from top to bottom for a match

If the Switch tag is absent we might end up with the entire Routes that matched displayed on the Browser.

NOTE: When we click on different Links to go to different pages, it's still sending a request to the server for each click.

But we said that React Router should intercept those request and handle it in the Browser instead

"And It does"

But instead of using Anchor Tags, we've to use a special React Router Link Instead. Next Up: We will be seeing how to do that

ROUTE LINKS #23

Currently on every link to a new page clicked: We are still sending a request to the server.

But we did say that the React Router could intercept those request and just handle the content changes in the Browser Instead.

And it can but to do it we have to use a special link tag.

Navbar.js

Let's Import the Link tag and use it.

```
import { Link } from 'react-router-dom';          // Added new Import

const Navbar = () => {
  return (
    ...
      <Link to="/">Home</Link>
      <Link to="/create">New Blog</Link>
    ...
  );
```

Added

This works perfectly and now ~~we don't~~ every request is intercepted from going to the Server and the React Router simply matches and Injects the Components we need into the Browser

THERE IS A PROBLEM THOUGH

When we click on {New Blog} and then {Home} (before fetch of data completes we click on New Blog) and then

In the console is an error that shows up.

Warning: Can't perform a React state update on an unmounted Component.

NEXT UP: We will look at a way of stopping the fetch when we leave fetch page.

use Effect Clean Up #24

Currently when we land on our homepage, our fetch for data sta
using our custom hooks and once it completes it tries to update the state
in our Home Component.

And we do see that because when the state changes, it does output
it to the dom.

However, if while the data is still trying to be fetched or is still
loading, we then go to another page before the fetch completes. We g
an error.

CONSOLE: Warning: Can't perform a React state update on an unmount
Component

Because when the data has finally been fetched it tries to update
the state in the Home Component. But then the Home Component is
longer in the Browser.

And thats why we got this Error Saying we can't perform
an update on an unmoted component

Therefore, we need a way of stopping the fetch when the Component is
it unmounts.

How Do We Do This?

We want to first of all run a clean up function when the component that uses the useFetch hook unmounts.

- We will place the "cleanup function" inside the useEffect and then we just return it. So when the component that uses the useFetch hook unmounts. It fires that returned "cleanup function" in the useEffect.

- Let's Demo this by Console.logging out 'cleanup'
- useFetch.js

```
...

  fetch(url)
    ...

  return () => Console.log('cleanup')          //Added
```

- Now if we were to try clicking to try moving from Home to another page before or after the fetch for blogs complete.

The cleanup function fires!

① cleanup

It's At this point that we want to actually stop the fetch going on in the background so we don't try to update the state.

How Do We Do That? We do that by using an AbortController();

Now we have this AbortController()
What we can do with it is, we can associate it with a specific fetch request and then once thats done.

We can use that AbortController to stop the fetch

useFetch.js

```
...
const useFetch = (Url() => {
  ...
useEffect (() => {
  const abortCont = new AbortController ();      // Added/Created

  setTimeout (() => {
    fetch (url, { signal : abortCont.signal })   // This can now be used to stop this fetch
      .then ...
      .then ...
      .catch ...
  }, 1000);

  return () => abortCont.abort ();               // Added
}, [url]);
...
...
```

If we Test this out. We still get an error even through it is aborting

WHY IS THAT?

When we abort a fetch, the fetch throws an Error and we are catching that error.

And if you Remember, when we catch an error we are still updating states.

```
setIsPending(false);
setError (err.message);
```

How To Fix This?

We want to recognize the error inside the catch block and if it's a specific type of error (e.g if it's an abort error, then we don't want to update the state)

```
...
...

.catch (err => {
    if (err.name === 'AbortError') {
        console.log ('fetch aborted');
    } else {
        setIsPending (false);
        setError (err.message);
    }
})
```

☑ DONE

≡ Now if we test this out we shouldn't get any ERROR!

CONSOLE:	fetch aborted

NEXT UP: We want to ~~both~~ go back to Routing and talk about Routing parameters.

Route Parameters # 25

Sometimes we need to pass dynamic values as ~~pt~~ paths of a route.

In other words, a route where certain parts of ~~of~~ it is changeable. Regardless of what that changeable part is, it still renders the same page or component.

An Example of this is for a Blogs Details page. That looks something like this

/blogs/123 ---> Route Parameter

which could easily be something else like

/blogs/456 or /blogs/789

In this cases we would still render the Blogs Details Component But instead we would show different kinds of blogs depending on the id that was passed.

So these Changeable parts inside of a Route is known as a ROUTE PARAMETER.

In our React Application we need to be able to use Route Parameters and access these Route Parameters from our components.

So that In the Component we can use these id's to then fetch data for that particular blog.

- Let's Create a BlogDetails Component and see how these Route Parameters work!

- BlogDetails.js

```
const BlogDetails = () => {                    // Created
  return (
    <div class Name = "blog-details">
      <h2> Blog Details </h2>
    </div>
  );
}

export default BlogDetails;
```

≡ Now Let's add a Route to our new Page/Component and pass in a Route Parameter Variable

- App.js

```
...
function App() {
  return ( ...
    //v6    <Route path = "/blogs/:id" element = { <BlogDetails /> }
                  or
    //v5    <Route path = "/blogs/:id">
                                                   // Also Import this
            <BlogDetails />
          </Route>
          ...
```

NOTE: The :id can now be anything/value/number passed into. They will all take you to the same Component.

≡ Now if we test this we should see that Regardless of what we pass in as the id. It still takes us to the Blog Detail Component/Page

E.g
localhost:3000/blogs/1
localhost:3000/blogs/5
localhost:3000/blogs/1000

BROWSER

The Doso Blog	Home	New Blog
Blog Details		

- We also ~~need~~ want to be able to access what the :id is in the routes
- Blog Details.js ≡ We use a hook provided to us by react-router

```
import {useParams} from "react-router-dom";          //Added

const BlogDetails = () => {
  const {id} = useParams();                          //Added
                  ↳ we use Object destructuring to grab which param we want to access

  return (
    <div class Name = "blog-details">
      <h2> Blog Details - {id} </h2>                 //Updated
                          ↳ Just to See this works
    </div>
  );
```

- localhost:3000/blogs/5

Blog details - 5

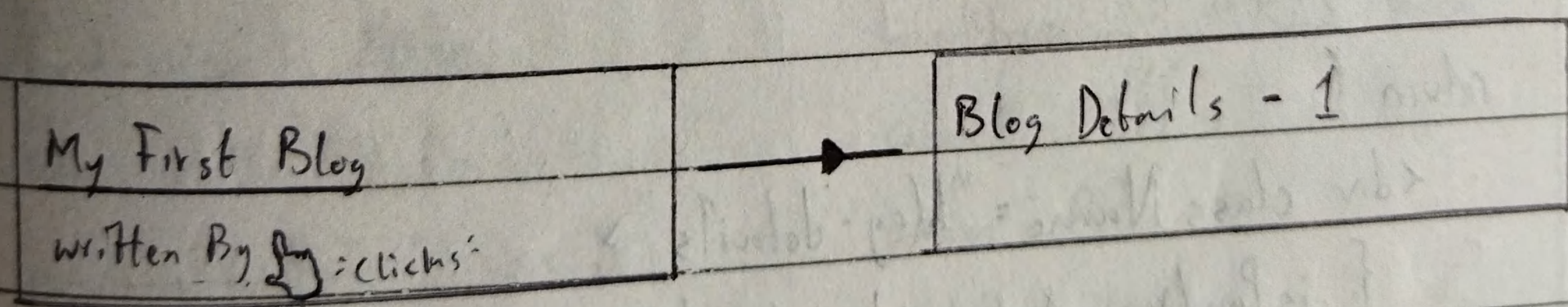
☑ So this works and we can now use this later to do a fetch request for a particular blog

- We want to wrap our Blogs with <Link> Tags
- import { Link } from "react-router-dom"; // Added Import

```
const BlogList = (
  ...

  <div class Name = "blog-preview" key={blog.id}>
    <Link to={`/blogs/${blog.id}`}>
      <h2>{ blog.title }</h2>
      <P> Written by { blog.author } </P>
    </Link>
  </div>

  ...
```

Added Link Wrapper

- ☑ Now if we go to home and click on the Links we should go to any blogs Individual Blog Details Page.



LASTLY: Add some styling

- index.css

```
.blog-preview a {
    text-decoration: none;
}
```

☑ DONE

NEXT UP: We want to show the Data for that single Blog.

Reusing Custom Hooks #26

Now we want to use the id parameter passed into the BlogDetails to make a fetch request for the details of that particular blog from the db.json file.

All we have to do is use/re-use our useFetch (Built by us hook) hook in the Blogs Details Component and pass in the url or the endpoint we want to use to fetch data from

- BlogDetails.js

```
import useFetch from './useFetch';                    // Added Import

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

return (
  <div class Name = "blog-details">
    { isPending && <div>Loading... </div> }
    { error && <div> { error } </div> }
    { blog && (
      <article>
        <h2> { blog.title } </h2>
        <p> Written by { blog.author } </p>
        <div> { blog.body } </div>
...) } </article>
```

```
</div>
);
}
export default BlogDetails;
```

☑ DONE

≡ Now if we try clicking from the Home Page to which ~~[illegible]~~ ever blog we want, it should open up displaying info about the blog

My First Blog written by mario	→	The Dojo Blog Home New Blog My First Blog! written by ~~[illegible]~~ mario why do we use it? it is a long establi- -shed fact that a reader will be distracted

index.css

≡ LASTLY: Let's style it up a bit

```
/* blog details page */
.blog-details h2 {
  font-size: 20px;
  color: #f1356d;
  margin-bottom: 10px;
}
.blog-details div {
  margin: 20px 0;
}
```

```
.blog-details button {
  background: #f1356d;
  color: #fff;
  border: 0;
  padding: 8px;
  border-radius: 8px;
  cursor: pointer;
}
```

For Future Blog Delete Button

☑ DONE

Controlled Inputs (Forms) #27

So Far We've Been Crucci. Now we want to look at Web Forms in React. So we can have users type in a new blog and add that

Then Later we will send a POST Request that adds that data. But in order to do this we need to talk about controlled Input.

What Are Controlled Inputs?

Controlled Inputs are basically a way in React of setting up input fields in forms so that we can track their values

If we had a text Input for example, a user can type into it and we could store that value of what they typed in some kind of state.

And we can also make it so that if the state changes. That in turn updates the value that we see in the input field.

So therefore, we are always keeping the input field and our state in "Sync".

≡ Let's Go Ahead and Set Up the boilerplate for our form in our Create.js (or Create Component)

o Create.Js

```
const Create = () => {
  return (
    <div class Name = "create">
      <h2> Add a New Blog </h2>
      <form >
        <label> Blog title : </label>
        <input
          type = "text"
          required
        />
        <label> Blog body : </label>
        <textarea>
          required
        ></textarea>
        <label> Blog author : </label>
        <select>
          <Option value = "mario" > mario </option>
          <Option value = "yoshi" > yoshi </option>
        </select>

        <button> Add Blog </button>
      </form>
    </div>
  );
}
```

☑ DONE with boilerplate

BROWSER: Preview

Add a New Blog

Blog title: [ ] Blog body: [ ] Blog author: [Mario v]

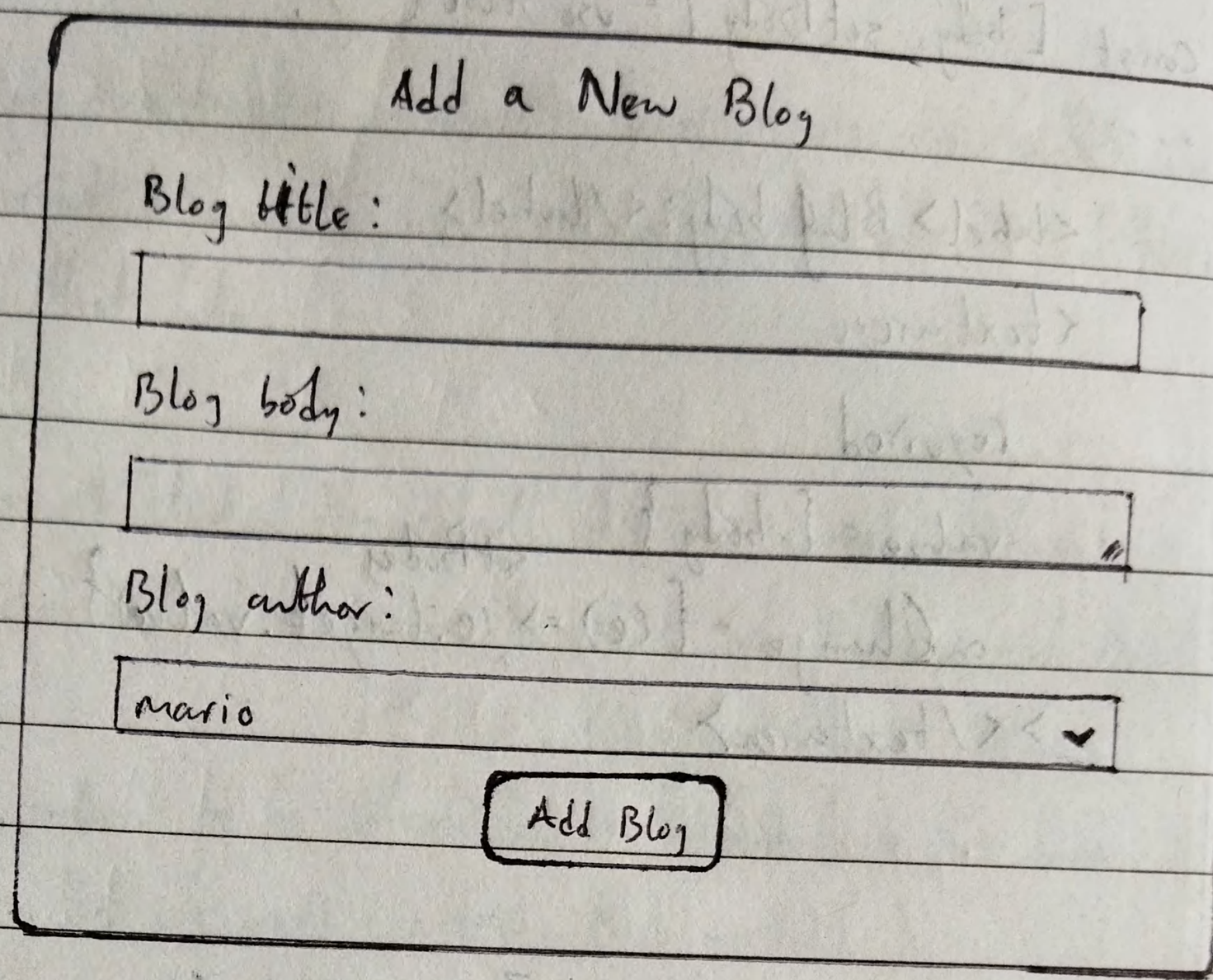
[Add Blog]

≡ NEXT: Let's Add some styling to the above

○ index.css

```
.create {
  max-width: 400px;
  margin: 0 auto;
  text-align: center;
}

.create label {
  text-align: left;
  display: block;
}

.create h2 {
  font-size: 20px;
  color: #f1356d;
  margin-bottom: 30px;
}

.create input, .create textarea, .create
select {
  width: 100%
  padding: 10px 10px;
  margin: 10px 0;
  border: 1px solid #ddd;
  box-sizing: border-box;
  display: block;
}

.create button {
  background: #f1356d;
  color: #fff;
  border: 0;
  padding: 8px;
  border-radius: 8px;
  cursor: pointer;
}
```

BROWSER:
After styling



≡ Now we want to be able to track the values entered in our input fields and store it in some state.

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

const Create = () => {
  const [title, setTitle] = useState('');
  ...

    <label>Blog title:</label>
    <input
      type="text"
      required
      value={title}
      onChange={(e) => setTitle(e.target.value)}
    />
```

i. For Title

// this updates our title state on Change and is required

Title Linked Up

Body Linked Up

```
const [body, setBody] = useState('');          ii, For Body
...
    <label> Blog body: </label>
    <textarea
      required
      value = {body}
      onChange = {(e) => setBody(e.target.value)}
    ></textarea>
```

iii, For Author

Author Linked Up

```
const [author, setAuthor] = useState('mario')
...
    <label> Blog author : </label>
    <Select
      value = {author}
      onChange = {(e) => setAuthor(e.target.value)}
    >
      <option value="mario">mario</option>
      <option value="yoshi">yoshi</option>
    </Select>
```

we want the select field to have mario selected as default.

≡ Now our states get successfully updated when we input values in our fields

NEXT UP: We will be looking at how we can **submit** our form

SUBMITTING A FORM #28

Hello, So now we are tracking what a user inputs or selects inside our form. Next we want to handle the form being submitted. And Eventually do something with that data.

When the button is clicked, by default it fires a submit Event on the form itself. We can therefore Listen to that Submit Event and React to it.

Another Option would be to attach a click event to the button itself and react to that, But we won't be using this method

- Create.Js

```
...
const Create = () => {
...
const handleSubmit = (e) => {
  e.preventDefault()                    // Prevents Page from being Refreshed
  const blog = {title, body, author};
                                        NOTE: When using Json-Server you don't have to provide an id because whenever we make a POST request, it automatically adds unique id's for us.

  Console.log(blog);
}
...
<form onSubmit={handleSubmit}>
```

☑ If we fill the form up and Submit ⟹

CONSOLE: ▸{title: "Snickers", body: "i love it", author: "yoshi"}

NEXT UP: We will be sending a POST request to an endpoint that Json-Server provides us with and add any new object we create

SENDING POST REQUEST #29

Now we can submit the form and create blog objects, the next step is to make a POST request to JSON-server to add that data to our data file.

- Create.js

```
const Create = () => {
  ...

  const handleSubmit = (e) => {
    e.preventDefault();
    const blog = {title, body, author};

    fetch('http://localhost:8000/blogs', {
      method: 'POST',
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(blog)
    }).then(() => {
      console.log('new blog added')
    })
  }
```

Remember this returns a Promise. So let's tag a .then

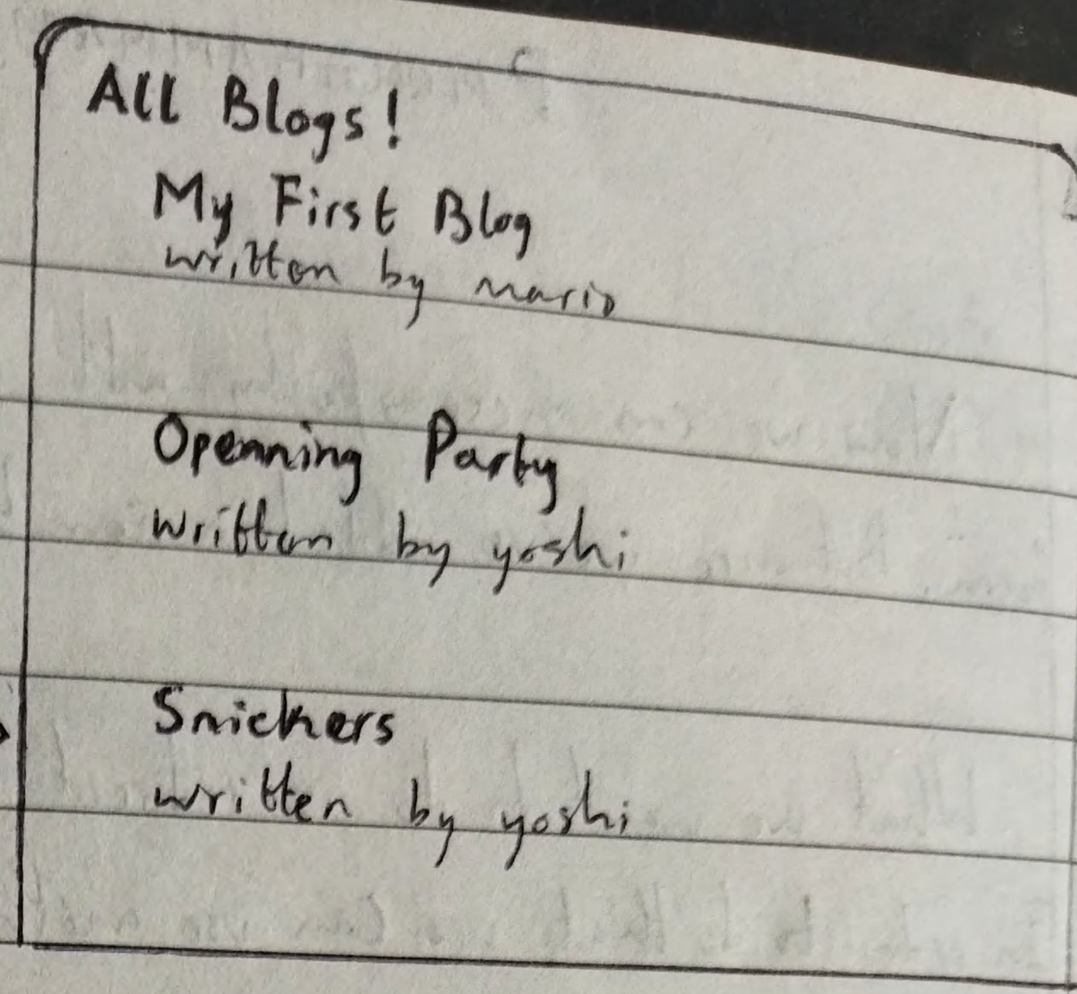
Added

☑ This Now Successfully adds a new blog to our blog file

≡ If we try re-adding the previous blog from the last page of should and we should see in the console.

CONSOLE: new blog added

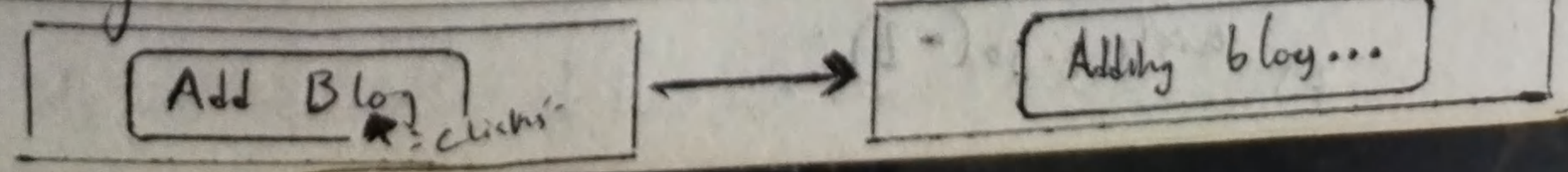
☑ DONE with adding of blogs



≡ Lastly: We want to add a Loading button as the form is being submitted

☐ Create.js

```
const Create = () => {
  ...
  const [isPending, setIsPending] = useState(false);

const handleSubmit = (e) => {
  ...
  setIsPending(true);

  fetch('http://localhost:8000/blogs', {...
  }).then(() => {
    setIsPending(false)
  })
return (
  ...
  {!isPending && <button> Add Blog </button>}
  {isPending && <button disabled> Adding blog... </button>}
  ...
)
```



PROGRAMMATIC REDIRECTS #30

Now we can successfully add new blogs by submitting our Create form. But once we've added a new blog it remains on the Create Page.

What we want to do is redirect us after submission to the Home page. In order to do that we can use another hook in 'react-router-dom' called "useHistory"

Create.js

```
import { useHistory } from 'react-router-dom'

const Create = () => {
...
  const history = useHistory();
```

In this hook are several methods we can use to go back in history, forward, and also redirecting the user.

≡ Let's see how to go back through history after submission.

```
const handleSubmit = (e) => {
  ...

  // history.go(-1);      // Goes back once through history
  // history.go(-3);      // Goes back 3 times through history
  // history.go(1);       // Goes forward once through history
  // history.push('/');   // Redirects one to a page

  fetch(..., {...})
    .then(() => {
      ...
      history.go(-1);
    })
```

→ navigate(-1)

→ history.go(+1)

→ ⓑ useNavigate

→ ⓐ useHistory

≡ Now if

After add

But we d

want to re

Create.

...

const Crea

...

const h

...

fetch

.th

})

}

Add

DONE

NEXT UP

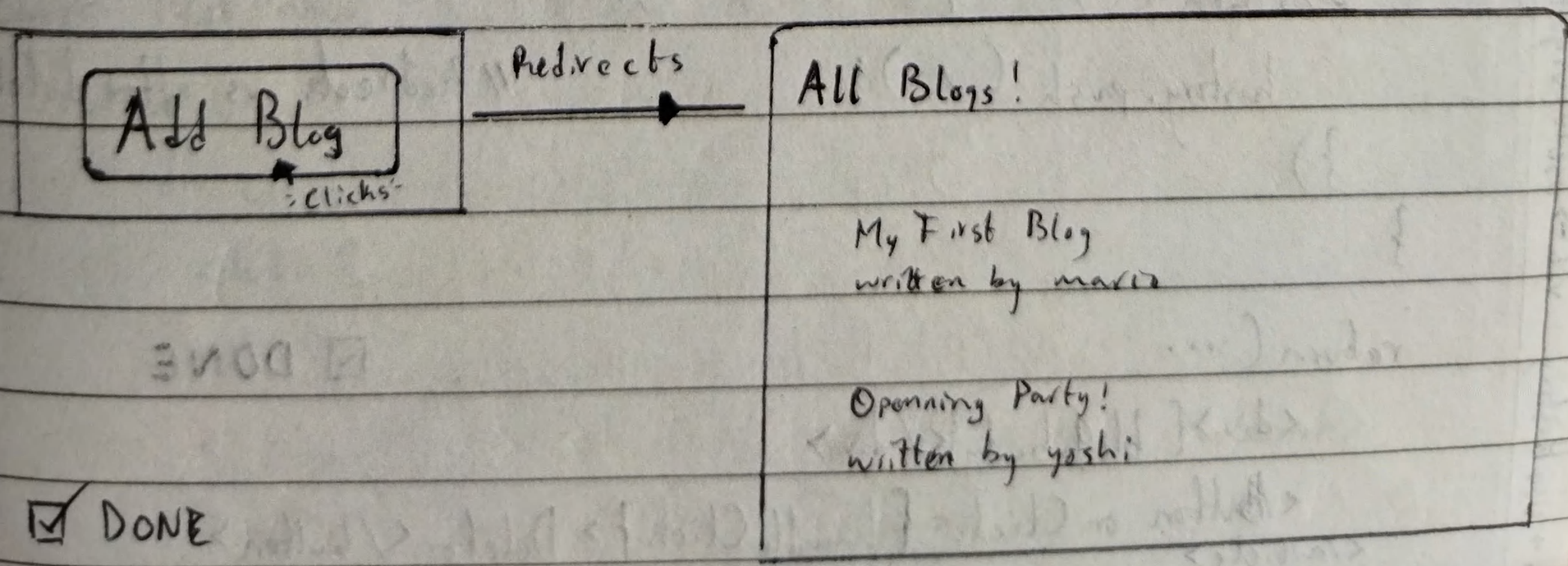
≡ Now If we come from home or blog details page to the create form. After adding our blog we go back in history Once ☑ Works

◎ But We don't want to go back once through history In our case. We want to redirect the user to be redirected to the home

○ Create.js

```
...

const Create = () => {
    ...

    const handleSubmit = (e) => {
        ...

        fetch(..., {...})
            .then(() => {
                ...
                history.push('/');      // Should Redirect us now
            })
    }
```

It WORKS



☑ DONE

NEXT UP: We will be looking at how we can DELETE BLOGS.

DELETE REQUEST #31

So Far Soo Good, things have been looking pretty good, we can successfully list all the blogs, add new blogs, see blog details and so on

Now what we want to be able to do is DELETE a Blog by Clicking on a button at the bottom of the BlogDetails Page.

☐ BlogDetails

```
import { useHistory, useParams } from "react-router-dom";
...

const BlogDetails = () => {
  ...
  const history = useHistory();

  const handleClick = () => {
    fetch('http://localhost:8000/blogs/' + blog.id, {
      method: 'DELETE'
    }).then(() => {
      history.push('/');          // Redirects us after delete
    })
  }

  return ( ...
    <div>{ blog.body }</div>
    <button onClick={handleClick}> Delete </button>
</article>
```

v.5 → useHistory v.6 → useNavigate

☑ DONE

≡ This should Now Delete Perfectly

NEXT UP: Setting up a 404 page for an unknown URL

404 Pages & Next Steps #32

Hey there, Congrats on making it to the final Chapter You deserve to be taken out 🙂.

There is one Last thing we want to do in this project and that is to create some kind of 404 Component or a Not Found Component.

That we want to show in the Browser when a user tries to go to some kind of URL like the one below.

Localhost:3000/sfdkjbcfddk

That Obviously doesn't Exist... To Do this we will have to create a component

- NotFound.Js

```
import {Link} from "realt-rater-dom"
const NotFound = () => {
  return (
    <div className="not-found">
      <h2> Sorry </h2>
      <P> That page Cannot be found </P>
      <Link to="/"> Back to the Homepage... </Link>
    </div>
  );
}
export default NotFound;
```

Now that our NotFound Component is ready what's left to do is set it up so that if anyone tries to go to an unknown route. We show them that Component.

How Do We Do This?

We do this by going to our App.Js and adding in, a catch-all Route for this Component at the bottom of all our routes.

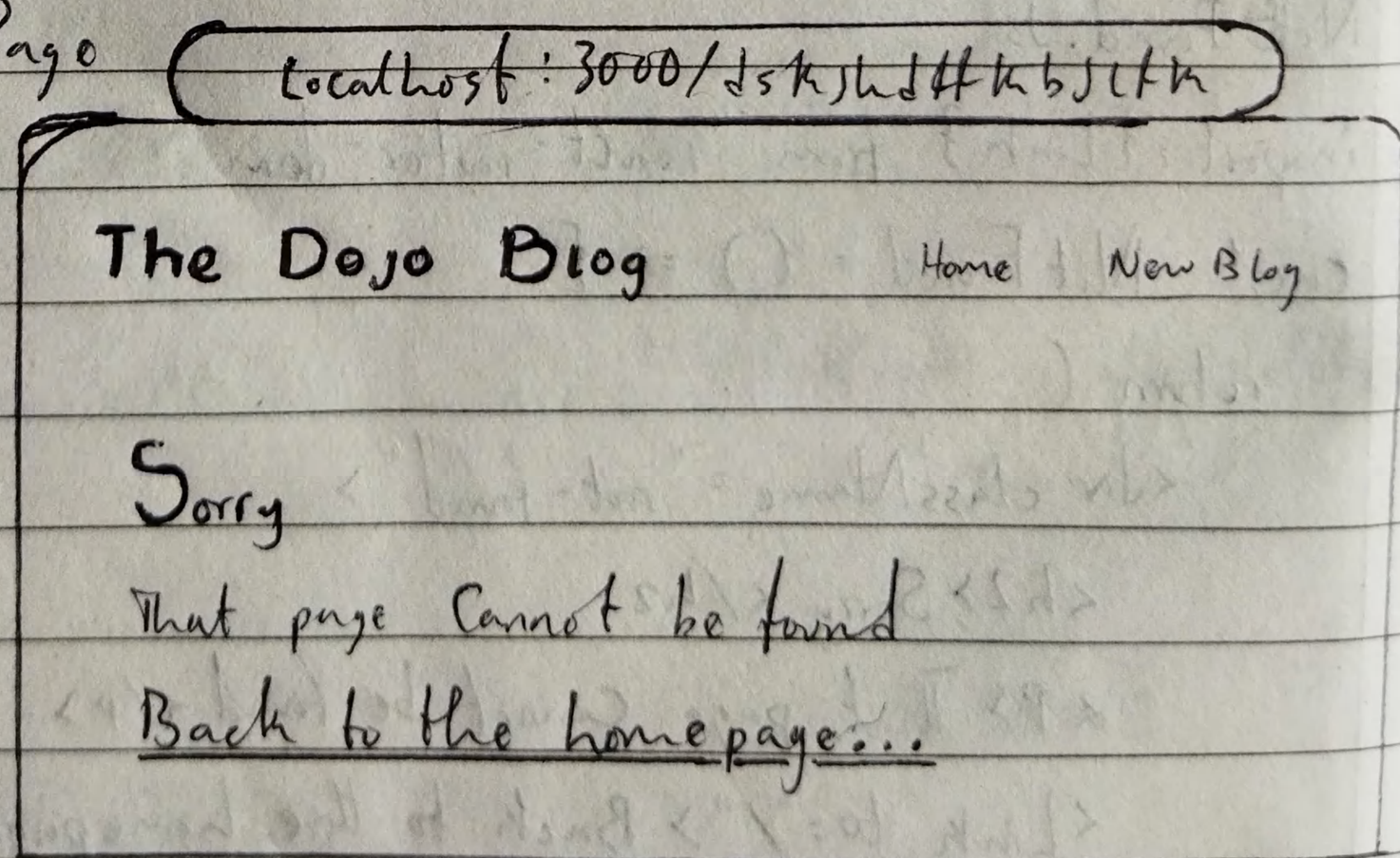
- App.Js

```
...
                    --> This Catches any Route
  <Route path="*">
@
V5 →  <NotFound />
  </Route>
```

```
@
V6 ⇒ <Route path="*" element=
        { <NotFound /> } />
```

...

☑ Now if we visit an unknown Route we should get back our 404 NotFound Page



COURSE COMPLETED!