React Practice

App Component

```
export default function App() {
  async function getAdvice() {
    const response = await fetch("https://api.adviceslip.com/advice");
    const data = await response.json();
    console.log(data);
  }

  return (
    <div>
      <h1>Hello World!</h1>
      <button onClick={getAdvice}>Get Advice!</button>
    </div>
  );
}
```

JSX

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

export default function App() {
  /* useState is a function that returns an array in react
   and here we are destructuring that array 1st position of the
   array is value of the state, the 2nd value is a setter function that
   we can use to update the piece of state */
  const [advice, setAdvice] = useState("");
  const [count, setCount] = useState(0);

  async function getAdvice() {
    const res = await fetch("https://api.adviceslip.com/advice");
    const data = await res.json();
    setAdvice(data.slip.advice);
    /* Now in this getAdvice function we can use setAdvice function to
    update the state, whenever the piece of state is updated
    user interface will also be updated */
    // take current count add 1 and that will become the new count
    setCount((c) => c + 1);
  }
  // Generate very first piece of advice when loaded
  /* useEffect takes two arguments 1st a function that
  we want to execute at the beginning when the component loads and 2nd
  argument is dependency array */
```

```
  useEffect(function () {
    getAdvice();
  }, []);
  return (
    <div>
      <h1>{advice}</h1>
      <button onClick={getAdvice}>Get Advice!</button>
      {/* include this component like it is another html element */}
      {/* Pass count as a prop to Message. Props are like parameters
      to function. We call the prop count and pass in the prop
      value */}
      <Message count={count} />
    </div>
  );
}

// In react we divide user interfaces into components
// components are reusable pieces of code
// components are used to render UI
// Name of all components should start with capital letter(convention)
// Now accept the props object as a parameter. In this props object count is
now a property
function Message(props) {
  // update count dynamically
  return (
    <p>
      You have read <strong>{props.count}</strong> pieces of advice
    </p>
  );
}
```

JSX Template: Babel transpiles JSX code by transforming it into regular JavaScript code. JSX (JavaScript XML) is a syntax extension for JavaScript that allows developers to write HTML-like code within JavaScript. Babel is a toolchain that is primarily used to convert ECMAScript 2015+ (ES6+) code into a backwards-compatible version of JavaScript that can be run in older browsers or environments.

When Babel encounters JSX code, it transforms it into function calls that create React elements. Here's a simplified explanation of how Babel transpiles JSX:

1. **Parsing**: Babel first parses the JSX syntax into an Abstract Syntax Tree (AST), which is a structured representation of the code.

2. **Transformation**: Babel then transforms the JSX AST into equivalent JavaScript code. This transformation involves replacing JSX elements with `React.createElement()` function calls.

3. **Generation**: Finally, Babel generates the transpiled JavaScript code from the transformed AST.

For example, given the following JSX code:

```jsx
const element = <h1>Hello, world!</h1>;
```

Babel transpiles it into the following JavaScript code:

```javascript
const element = React.createElement("h1", null, "Hello, world!");
```

In this transpiled code, `React.createElement()` is used to create a React element representing the `<h1>` element with the text content "Hello, world!".

It's important to note that Babel itself doesn't understand JSX syntax natively; instead, it relies on plugins like `@babel/preset-react` to handle JSX transformation. This preset includes the necessary plugins to transform JSX into JavaScript that can be understood by browsers or other JavaScript environments.

App.js:
——————————————————
```
import "./App.css";
function App() {
  const title = "Welcome to the new blog!";
  const likes = 50;
  // can output: number, string, and array
  // can't output: boolean and objects
  const link = "https://www.google.com/";

  const ninja = { name: "Yoshi", age: 30 };
  // returning dynamic values
  return (
    <div className="App">
      <div className="content">
        <h1>{title}</h1>
        <p>Liked {likes} times</p>
        {/* <p>{ninja}</p> */}
```

```
        <p>{`${ninja.name} is a ${ninja.age} years old ninja!`}</p>
        {/* bunch different element in arrays together and bunch them as string
*/}
        <p>{[1, 25, 686]}</p>
        {/* use dynamic values as attribute values in element tags */}
        <p>{Math.random() * 20}</p>
        <a rel="noreferrer" target="_blank" href={link}>
          Link to google!
        </a>
      </div>
    </div>
  );
}

export default App;
```

Components contain template which makes up the html along with js logic. Components return jsx and In background a transpiler named babel converts all this jsx template into regular JavaScript code when we save the file. class is a reserved keyword in js show we can't use in jsx.

Components return jsx and exported to use elsewhere.

React converts out (whatever data type we give it to it) into strings before it renders it in the browser.

```
 can output: number, string, and array
 can't output: boolean and objects
```

Lesson 6 (Adding Styles):

React styles are not bound to a single component, it will be applied to all components in the browser at that time (inspect to see), cause react takes all these styles adds them to head of the webpage. check out styles tag inside head element.

The styles are not scoped into single component, we can use css modules or styled components to scope styles into components.

We can create a global index.css file to implement all our styles.

Inline styling in react:

```
        {/* Inline styling in jsx, the first {} denote dynamic value and inside
{} denote an object inside its key-value pair, key = css property, value = css
property value (string) */}
        <a
          href="/create"
          style={{
            color: "#fff",
            backgroundColor: "#f1356d",
            borderRadius: "8px",
          }}
        >
          New Blog
        </a>
```

Lesson 7 (click events):

Home.js

```
-------
const Home = () => {
  const handleClick = (e) => {
    console.log("Hello Ninjas!", e);
  };
  const handleClickAgain = (name, e) => {
    console.log(`Hello ${name}!, ${e.target}`);
  };
  return (
    <div className="home">
      <h2>Homepage</h2>
      {/* passing function reference {handleClick} on clicking function will be
invoked, calling it handleClick() will invoke the function without clicking
*/}
      <button onClick={handleClick}>Click me!</button>
      {/*Avoid this handleClickAgain(name) as it will invoke the function
without clicking it, passing in argument to function: wrap inside anonymous
function '() => {
          handleClickAgain("Mario");
        }', e.g. handleClick is similar like () => { console.log(Hello
Ninjas!)} as we are not invoking the function with () we are referencing it, on
clicking the anonymous function is invoked then handleClickAgain("Mario") with
"Mario" argument, we can also remove curly braces ({}) cause it's one line, in
jsx {} indicates dynamic value */}
      <button onClick={(e) => handleClickAgain("Mario", e)}>
        Click me again!
      </button>
```

```
      {/* Event object/parameter we automatically get access to it when an
event occurs. As the first parameter of the function (referenced function), In
the second case: anonymous function gets access to event object, now will pass
that in handleClickAgain("Mario") function as 2nd argument */}
    </div>
  );
};

export default Home;
```

Lesson 8(useState hook):

State of a component means data being used on that component in a particular time.
It could be an array of values, booleans, string, objects or any other data that our component uses.

We created some variables and used them in our template before (refer to React Practice.pdf App.js file) but if we need to change variable or data over time or in reaction to some event (user clicking a button)

Home.js (Before using useState Hook):

```
// Destructure useState hook from react library, this grabs us the function
from the react library, now we can use it in our component
import { useState } from "react";

const Home = () => {
  let name = "Mario";
  const handleClick = () => {
    // Updating the value will not reflect in the template
    name = "Luigi";
    /* name itself changed to luigi but it doesn't get updated in the template
(view in browser console) cause the variable we have created is not reactive
(let name = "Mario") means react doesn't watch it for changes, when its value
changes it doesn't trigger react to re-render the template with the new value
inside it (name variable in this case) and we continue to see the old value
(Mario) in the browser, to make this work we need to make the value reactive so
that when it changes react detects that and it re-renders the template with the
new value (where we output it in the template) and updated value is visible to
the browser to implement this we use hook which is called useState(). Hook in
react is a special type of function that does a certain job, it generally
starts with the word use. useState hook provides us a way to make reactive
```

```
value, also provides a way to change that value whenever we want. So to use
useState hook we need to import it  */
    console.log(name);
  };

  // const handleClickAgain = (name, e) => {
  //   console.log(`Hello ${name}!, ${e.target}`);
  // };

  return (
    <div className="home">
      <h2>Homepage</h2>

      <p>{name}</p>
      {/* onclicking button change the value of name variable */}
      <button onClick={handleClick}>Click me!</button>
      {/* <button onClick={(e) => handleClickAgain("Mario", e)}>
        Click me again!
      </button> */}
    </div>
  );
};

export default Home;
```

Home.js (After using useState Hook):

```
// Destructure useState hook from react library, this grabs us the function
from the react library, now we can use it in our component
import { useState } from "react";

const Home = () => {
  // let name = "Mario";
  // Make a reactive value using useState, we give this useState a initial
value for example (Mario), we want to store this in some variable. We use array
destructuring to grab 2 values that this hook returns to us, first value is the
initial value (name), 2nd value is a function that we can use to change that
value most times it's called set[whatever the name of value to be changed]
  const [name, setName] = useState("Mario");
  const [age, setAge] = useState(23);
  const handleClick = () => {
    /* changing the state data using setName, this useState value is reactive
if we change it it's gonna change in the template as well, when we use this
function to change the value that triggers react to re-render the component
upon re-rendering it has the new value of name cause it's been updated. We can
```

```
use this hook (useState) as many times in our component for different values
other than name like it can be array, object, boolean etc. like 'const [age,
setAge] = useState(0)' The data type of state we are using doesn't matter */
    setName("Luigi");
    // update age to 30
    setAge(30);
  };

  return (
    <div className="home">
      <h2>Homepage</h2>
      {/* At first it will give us the initial value (Mario), to change this
value we can use the setName function */}
      <p>
        {name} is {age} years old.
      </p>
      <button onClick={handleClick}>Click me!</button>
    </div>
  );
};

/* Conclusion: When we need a reactive value something that might change at
some point we use the useState hook to do that we pass in an initial value and
we can output that value in the template and then we just call the set function
2nd value we get in the destructured array to update it and that triggers
re-render and the new value is going to be output to the browser in this
template, so this hook is very useful 😄 */

export default Home;
```

Lesson 9 (React Dev Tools):

React Dev tools integrate with browser development tools and gives us extra features that we can use on any website created with react. To extra tabs components, and profiler is available. Components is more useful. This gives us component diagram or component tree of our current app. Hovering over it provides extra info about the component.

Important tabs in react dev tools:
1. Inspect the matching DOM element. (eye icon)
2. Log this component data to the console. (bug icon)

Change of state (data) under hooks tab can be viewed in home (by selecting it from component tree) component by clicking the button (click me). Changes in state in a component can be tracked here.

If we log the home component to the console we can see the hooks property with array of objects and each object represents the piece of state we have with properties name, value, id, isStateEditable etc.

Lesson 10 (Outputting Lists):
Goal: Outputting list of blogs in our template.

Create some states to represent these blogs. We will be using useState hook cause data might change at some point we might delete the blog, we need react to update the DOM when that happens.

Home.js (lecture 10: Outputting List)

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

const Home = () => {
  /* destructure the two values, initial value of this state is an array of objects, each objects represent a blog with title, body, author, and id property. This id is going to be used by react when we output this data, each id needs to be unique for each one of the items/blogs. */
  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 className="home">
      {/* iterate/loop through blogs array using map method, for each iteration as we iterate through this we get access to the item we are currently iterating that is blog in this case, when we output list using map method each root element in the template that we return must have a key property now this key property is something that react uses to keep track of each item in the dom as it outputs it, so if data changes at any point say we remove/add new items to the array react can keep track of those items. So we must add a key attribute to each item that we output, otherwise react can't distinguish between list items in the DOM. This normally an id property for each item in the array (we already have that in our array of objects) */}
      {blogs.map((blog) => (
```

```
        // what we want to output for each blog
        <div className="blog-preview" key={blog.id}>
          <h2>{blog.title}</h2>
          <p>Written by {blog.author}</p>
        </div>
      ))}
    </div>
  );
};

/* Summary: That is how we output a list of data in react. We have a list in
this case an array of object which is stored in useState the we map through the
data and we take each item into that as we map through it an we output a bit of
template for each one (in this case see comment what we want to output for each
blog) and each one has a key property which is id in our case but it could be
any unique property. Now some css is added for each blog template ( view after
'blog previews / list' comment in index.css )  */

export default Home;
```

Lesson 11 (Props):

If we are building a real blog we might have the list of blogs in various places on our website it might be the homepage, search, category or tag page. So several different areas may use the same logic where we cycling/iterating through blogs and outputting a blog preview for each one. To implement that in our project we'll be repeating the code (map function iteration) over and over again in different components for different pages. Where we have pieces of components or bits of templates that might be reused in different positions or different places in the website we like to make that bit of template into its own reusable component. e.g. if we make a component blog list then we could drop this blog list component in any other components in the project. So if we have category page later on we could just get the blog list component and drop it in. To pass in different data into reusable component every time use it we will do that in form of props. E.g. in the homepage we might list all the blogs and show a preview for all the blogs starting from the latest one but on a search page or search component, we might only show the blogs that match the search term so the data is going to be different the structure is the same (of map function iteration BlogList component) but the blogs that we are going to use (state array of objects/blogs) is going to be different, so we can pass in data into these external components as well in the form of props. An external component of BlogList will contain all the logic/template(jsx) of map iteration and listing blog preview.

Home.js

```
import { useState } from "react";
import BlogList from "./BlogList";

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 className="home">
      {/* {} - Dynamic value */}
      <BlogList blogs={blogs} title="All Blogs!" />
    </div>
  );
};
/* Conclusion: That's how we can make a component take in props data and then
use that data inside that component. It makes the Bloglist component more
reusable and it does, we can now use this BlogList component anywhere in our
application whether in home component or in different page component later on
*/
export default Home;
--------------------------------------------------------------------------------
BlogList.js
--------------------
// const BlogList = (props) => {
const BlogList = ({ blogs, title }) => {
  /* storing blog property of props object in blogs, we are passing in
properties to props object and grabbing this different properties from this
props object and storing these in this variables now an easier way to do this
is destructuring ('{blogs, title}') as we want blogs and title from props
object */
  //   const blogs = props.blogs;
  //   const title = props.title;
  //   const { blogs, title } = props;
  //   console.log(props, blogs);
  return (
    /*  In the BlogList component we are trying to map through the blogs data
but this component has no idea of what blogs is. The blogs data is not defined
```

in this component. We can't just use any data in another where it's defined (Home component in this case) right here in home component because it can't reach that. There is 2 ways to fix this: 1st option is to redeclare all this data into BlogList component instead of home (data - ' 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,
    },
  ]);' the 2nd option is to use props whereby we pass this data from this home component into the BlogList here the 2nd option props will be used for 3 reasons 1st: It's gonna make our BlogList component more reusable and we'll see exactly how later on. 2nd: It allows us to still use this data in home component later on (cause we are not removing it) if we need it in future cause the data is still gonna be declared here in home component. 3rd: It allows to learn how to use props 😁. Props are a way to pass data from one component(a parent component) into a child component. In this case Home is parent component and BlogList is child component, we will be passing the blogs data into the BlogList component to do that we will make a property name on <BlogList/> tag. We can call it whatever we want it's blogs in this case. So, <BlogList blogs={blogs} /> now this is being passed into BlogList component (check home component code) as a prop (blogs={blogs} - this is a prop). We need to receive it here in BlogList component we get access to an argument inside this function/component called props. Now this blogs property will be in props object. Any props that we send through (like blogs={blogs}) into a component will be attached to this props object which we automatically get as an argument in the component ('const BlogList = (props)=>{}') and we can access them like props.blogs. We can pass in multiple props if we want to like we can pass in title="All Blogs" with a string value */

```
    <div className="blog-list">
      <h2>{title}</h2>
      {/* {props.blogs.map((blog) => ( */}
      {blogs.map((blog) => (
        // what we want to output for each blog
        <div className="blog-preview" key={blog.id}>
          <h2>{blog.title}</h2>
          <p>Written by {blog.author}</p>
        </div>
      ))}
    </div>
```

```
  );
};

export default BlogList;
```

Lesson 12 (Reusing components):

We took all this logic (map iteration) from home component and externalized it into BlogList component and we passed props into that component the data which it uses and externalizing all this logic into different component, makes that code more reusable, we can reuse this component in different places in our application where we need it and we can pass different data to it each time.

Home.js

```
import { useState } from "react";
import BlogList from "./BlogList";

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 className="home">
      <BlogList blogs={blogs} title="All Blogs!" />
      {/* pass in filtered data from blogs array, we will take out the blogs
which doesn't have a author of mario, pass in title of 'Mario's Blogs' which
was 'All Blogs!' previously, filter method fires a callBack function for each
item in the array now if we return true for that item it keeps it in the array
and if we return false it filters it out of the array and it returns a new
array with the items except filtered out ones and we are passing this that data
(filtered out array) as a prop, so this BlogList component can be reused and we
can reuse it with different data which makes it really useful for doing things
like a search page whereby the title matches the search term for example */}
      <BlogList
```

```
        blogs={blogs.filter((blog) => blog.author === "mario")}
        title="Mario's Blogs!"
      />
    </div>
  );
};

export default Home;
```

BlogList.js

```
const BlogList = ({ blogs, title }) => {
  return (
    <div className="blog-list">
      <h2>{title}</h2>

      {blogs.map((blog) => (
        <div className="blog-preview" key={blog.id}>
          <h2>{blog.title}</h2>
          <p>Written by {blog.author}</p>
        </div>
      ))}
    </div>
  );
};

export default BlogList;
```

Next Lesson: Updating the state by deleting items from data array (state of array of objects/blogs)

Lesson 13 (Functions as Props Delete an item):

Allow users to delete blogs by clicking on the button. We need a button inside the BlogList component for each blog that we output doing that below the author {blog.author}.

BlogList.js

```
const BlogList = ({ blogs, title, handleDelete }) => {
  /* function to delete blog using id: here we want to delete the blog with that id from blog data, now the data is initialize in home component useState (this is where the state is), we don't want to directly edit the blogs prop (blogs prop - that is passed in in this component from Home component) (we
```

```
shouldn't do that) instead we need to use setBlogs method inside Home component to the update the state (change the state after deleting an item (blog)) that's what we need to do, it's not good to defined handleDelete function in here (in BlogList component) instead we want it to be defined in the home component, so we can interact with the data directly and then we can pass in this data (returned data from handleDelete function, the data after deleting) through as prop, so we can create a prop called handleDelete in Home component and set it equal to handleDelete function e.g. handleDelete={handleDelete} and the in BlogList we can accept this function as a prop in BlogList component and we are using it in the onClick event see below ⬇️, So we are invoking this function which is defined in the parent home component and inside home component we can use setBlogs function to update the state and we can remove the blog with this id (check handleDelte function in Home component) */

  return (
    <div className="blog-list">
      <h2>{title}</h2>
      {blogs.map((blog) => (
        <div className="blog-preview" key={blog.id}>
          <h2>{blog.title}</h2>
          <p>Written by {blog.author}</p>
          {/* button with click event handler (onClick), invoke an anonymous function while clicking it (Note: We pass in function expression (body of function or function name)), we'll pass in the id of the blog we want to delete so that we can find it in the array and delete it, we have access to the blog and the id property is in the blog, so we are passing that (blog.id) into handleDelete function so we know in here which blog to delete, this click event invokes the handleDelete function defined in Home component */}
          <button onClick={() => handleDelete(blog.id)}>Delete blog!</button>
        </div>
      ))}
    </div>
  );
};

export default BlogList;


Home.js
-------
import { useState } from "react";
import BlogList from "./BlogList";

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,
    },
  ]);

  const handleDelete = (id) => {
    /* store the new array temporarily in newBlogs, the filter method doesn't
change the original (blogs) array it doesn't mutate (change) it, It returns a
new filtered array, blog id that doesn't match this passed id will remain in
the array id that is not matched is removed, the new array is stored in
newBlogs */
    const newBlogs = blogs.filter((blog) => blog.id !== id);
    /* we will use setBlogs to set new value (filtered array of objects/blogs
(newBlogs)) for blogs, see the defined array of objects up 👆 in the useState
that is previous (initial) value and we will update the state by setting the
value of blogs to newBlogs, fingers crossed 🤞, upon refreshing the modified
newBlogs state will not persist the state will set back to initial state as it
re rendering, re-running the code, that's how we are passing functions
(handleDelete) into other components (BlogList) as props from parent
component(Home) */
    setBlogs(newBlogs);
  };

  return (
    <div className="home">
      {/* we can pass in this data (returned data from handleDelete function,
the
      data after deleting) through as prop, so we can create a prop called
      handleDelete in Home component and set it equal to handleDelete function
      e.g. handleDelete={handleDelete} */}
      <BlogList blogs={blogs} title="All Blogs!" handleDelete={handleDelete} />
    </div>
  );
};

/* conclusion: handleDelete function is defined where the original initial data
is in the home component we will not be modifying blogs prop instead we will
pass this handleDelete function as prop to BlogList component and use it in
there in the onClick method by invoking it with the id of the blog to be
```

deleted this handleDelete function defined in Home component since then invoked and uses setBlogs(newBlogs) to change value of blogs to newBlogs filtered array upon refreshing the code reruns and the state(data) of blogs is set back to initial value */

export default Home;

Lesson 14 (useEffect Hook):

We have seen a useState hook that is used to create some state for the component. But there are many other hooks that we can use in React. One such is useEffect - this hook runs a function at every render of the components, remember the component renders initially when it first loads and the rendering also happens when the state changes it re renders the dom so we can update that state in the browser, so this useEffect hook is a way to run code on every render and that can be useful for many different reasons which we're going to see later on, now focus on how can we use it.

First thing to do import it from react - (import { useState, useEffect } from "react")

Home.js

```
import { useState, useEffect } from "react";
import BlogList from "./BlogList";

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,
    },
  ]);

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

```
  /* use useEffect hook - we don't store it inside a constant it doesn't return
anything all we need to do is pass as an argument a function, this function the
function that's going to run every time there is a re-render so once initially
when the component first loads but thereafter anytime the data changes,
normally in this function inside useEffect we could do something like fetch
data or communicate with some kind of authentication service and those things
are known as side effects in react, but for now we are doing simple
console.log, if we see in browser console we can observe useEffect ran on
refresh and when we delete a blog(changing the data (state) (re-rendering)) so
useEffect runs on every render. We can also access the state inside useEffect,
so if we want to output the blogs we can do 'console.log(blogs)' inside
useEffect. If we inspect the browser console we can see the blogs data on every
render as useEffect runs, Need to be careful about changing the state inside
useEffect because because we could end up in a loop of continuous renders (e.g.
setting state inside useEffect 'setBlogs(newBlogs)') 😅. In this scenario,
Initially the component renders to the DOM  which will trigger useEffect
function to run that would then update the state and the state(data) would
change and that would trigger a re-render on that re-render again that triggers
this function in useEffect to run and this goes on again and again creating a
endless loop, there are ways to fix it which we will see later */

  useEffect(() => {
    console.log("useEffect ran");
    console.log(blogs);
  });

  return (
    <div className="home">
      <BlogList blogs={blogs} title="All Blogs!" handleDelete={handleDelete} />
    </div>
  );
};

/* Conclusion: This useEffect hook is really really useful for running any kind
of code that we need to run at every render. It can be used for things like
fetching data we're gonna see that later. Next up we gonna look at dependencies
of useEffect */
export default Home;
```

Lesson 15 (useEffect Dependencies):

useEffect hook the function inside it fires after every render, that happens once initially when the component first loads but thereafter every time the state changes and we re-render the template. But we don't want to run a function after every single render rather maybe only after certain renders to

do that we can use dependency array, this is an array that we can pass into useEffect hook as a second argument like this:

```
  useEffect(() => {
    console.log("useEffect ran");
    console.log(blogs);
  }, []);
     👆 (2nd argument)
```

Passing an empty array - this ensures that useEffect hook runs only after the first initial render, thereafter if the state (data) changes it won't run the function again. It only runs it once. This is useful if we want to only run the function once after the first render.

Now we can also add actual dependencies to this array, meaning any state (data) values that should trigger the useEffect function to run when they change. To demonstrate we are creating another piece of state.

Home.js

```
-------
import { useState, useEffect } from "react";
import BlogList from "./BlogList";

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,
    },
  ]);

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

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

  /* We want to run this useEffect at the beginning when the component first
renders but also whenever a certain value changes (in this case name meaning if
```

this state changes - `const [name, setName] = useState("Mario")` so the useEffect function will run only when the name state changes, given name will become the dependency we add into the 2nd argument `[name]` - now useEffect is going to watch this value and if it changes it will run the function. Now on the first render useEffect still runs displayed console.log(name) output mario on browser console. If if we delete the blogs it will not run cause it's only watching for changes in name state not in blogs state cause blogs is not in the dependency array, but if we change the name it does run the function (see the console) cause name is in the dependency array & when it changes the function inside useEffect is ran). Now one thing to notice: after changing the name state to luigi, if we click the change name button again it will not run again. Cause although it is using setName("Luigi") function to change the state it's not actually changing the value anymore cause it's already luigi at this point (we already clicked the button changing the state to luigi) so state is not changing and we are not triggering that re-render and therefore the function inside useEffect (useEffect) in general is not running */

```
  useEffect(() => {
    console.log("useEffect ran");
    // console.log(blogs);
    console.log(name);
  }, [name]);

  return (
    <div className="home">
      <BlogList blogs={blogs} title="All Blogs!" handleDelete={handleDelete} />
      {/* this onClick event will invoke an anonymous function which then
invokes the setName(function that changes the state - data) */}
      <button onClick={() => setName("Luigi")}>Change name</button>
      {/* Outputting, the name in a paragraph */}
      <p>{name}</p>
    </div>
  );
};

/* Conclusion: That's how we can use dependencies this dependency array is the
2nd argument to useEffect to control when this useEffect function runs */

export default Home;
```

Lesson 16(Using Json Server):

Fetching Data using useEffect - it is good place to fetch data in a component cause we know it runs the function (function inside useEffect sent as 1st argument) when the component first renders initially, and that's generally when

we want to go and fetch some data and then we can use that data in our application instead of the data that we already have in the blogs state:

```
`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,
    },
  ])`
```

Cause typically in an web app we will not have hard coded data like this, instead it will probably come from a database using an api endpoint (rest api). We are gonna using json server which will allow us to build a fake rest api just using a json file that we can use to test this out.

1st step is to create a json file which is going to act as our database, this will reside in data folder of root directory of dojo-blog. in `data/db.json`

[db.json is one property called blogs with an array of two other objects]

db.json

```
{
"blogs": [
  {
    "title": "My First Blog",
    "body": "Why do we use it?\nIt is a long established fact that a reader
will be distracted ",
    "author": "mario",
    "id": 1
  },
  {
    "title": "Opening Party",
    "body": "Why do we use it?\nIt is a long established fact that a reader
will be distracted ",
    "author": "yoshi",
    "id": 2
  }
]
}
```

so each object is a blog with title, body, author, and id, we have two objects/blogs in total and when we're using json server each top level property is considered a resource so we just have one top level property blogs (which contains the array with 2 objects) so it sees that as resource and 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 etc so that is db.json file in a nutshell.
// comments are not allowed in json

Now we'll use json server package to watch this file (db.json) and wrap it with some endpoints. So there is 2 options here either 1. Install json server package locally into this project and then use it or 2. Use npx like we did to create-react-app to run the code from web and it will still watch our file right here `db.json`, that's is what we are gonna do. Open up different terminal rather than that's running our local development server(localhost:3000).

We need to install json-server with 2 flags: (watch and port)
`npx json-server --watch data/db.json --port 8000`

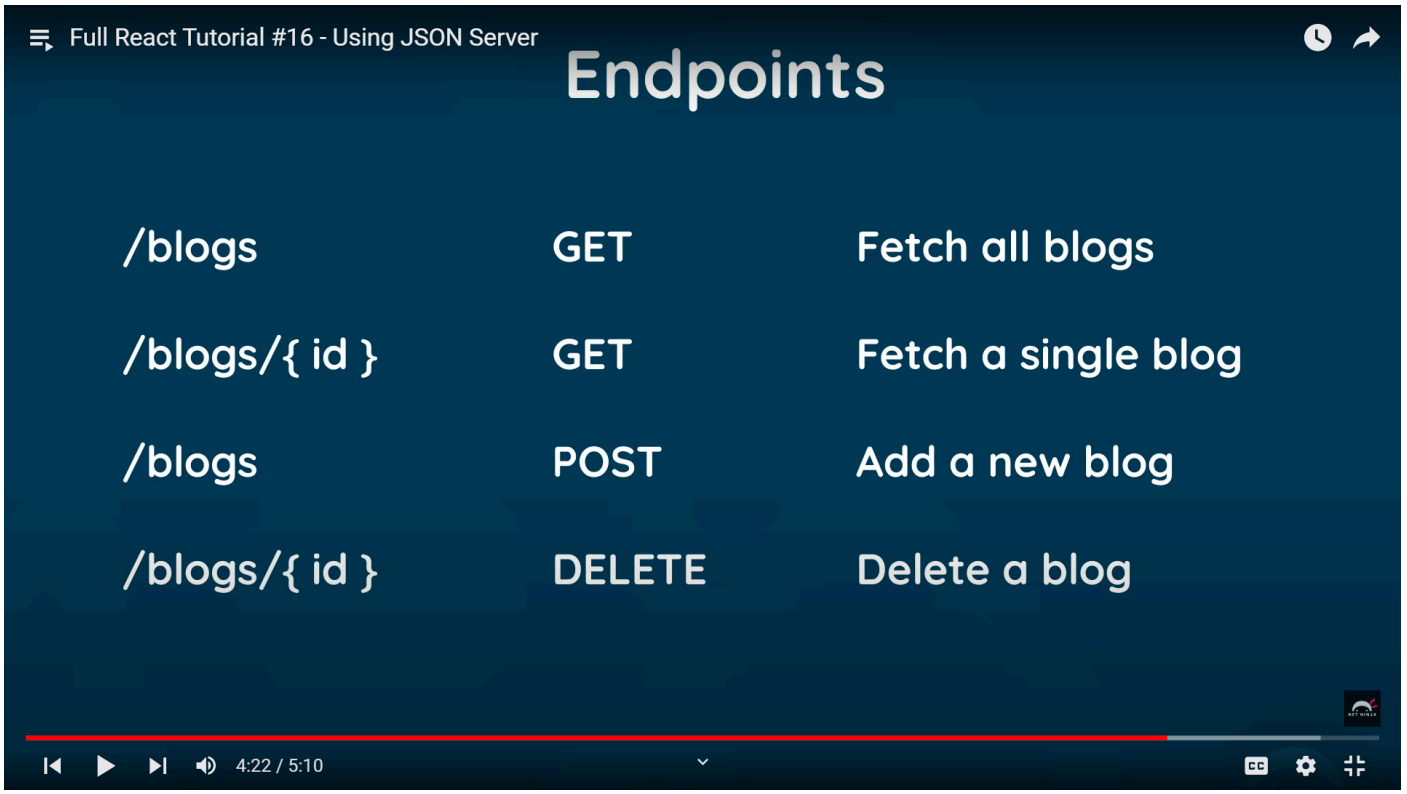
watch followed by the path of the file to watch.
And the port number where json-server will run.

After running it (the command above) is going to watch db.json and will wrap it with some api endpoints.

It picked up that we have a blogs resource: http://localhost:8000/blogs
So if we want to send a get request to get all of the blogs we would use this endpoint: http://localhost:8000/blogs - we can view this url in browser and see the data typically we're not gonna be using the browser to get that data instead we're gonna be using a fetch request inside our component using these different endpoints right here. Now at the moment we can see this: http://localhost:8000/blogs endpoint but it also provides us with other endpoints. The endpoints we are gonna be using (provided by json-server to us) is shown in the JSON server API endpoint image in lecture summary. This is not all of the endpoints, just the ones we will be using. The use of these endpoints is also provided in the image. The first one we already tested in the browser (refer to the image) - http://localhost:8000/blogs.

/blogs/{id} - http://localhost:8000/blogs/1 (1 is the id)

The Image:



Next up - make a fetch request in useEffect to fetch all the blogs.