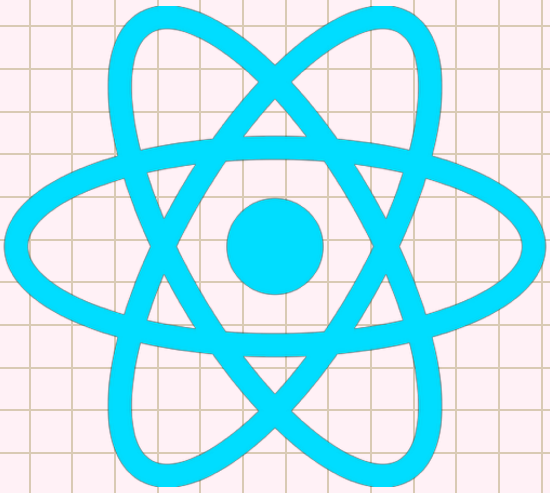
Assignments



Namaste React

Khushi Johri Notes

Assignment 1 - Inception

Q1 What is Emmet?

⇒ Emmet is an abbreviation which will dynamically set a boiler plate.
Eg:- XML, HTML etc

Q2 What is the difference between a Library and Framework?

⇒ Library can work in a small part of the app too unlike a framework.

Other frameworks need to be used in a full-fleged app to use them. They can't be used in already existing project.

But React can be used inside your existing project as well.

Q3 What is CDN? Why do we use it?

⇒ CDN stands for content delivery network.
These are the websites where react has been hosted and we pull react into our projects using CDN links.

We use CDN because it reduces latency by shortening the physical distance a user and the origin servers using a closer point of presence (PoP). CDNs use geographical proximity as the main criteria for determining which server will fulfill a given request.

Q4 Why is React known a React?

⇒ Because of its ability to react on change of the data.
When data in a React component changes, React will automatically re-render the component so that it reflects the new data. This makes it easy to create performance user interface that always look upto date.

Q5 What is cross-origin in the script-tag?

⇒ Cross origin is a request for resources from another domain.
This attribute sets the mode of the request to an HTTP CORS Request

Q6 What is the difference between React and React DOM?

⇒ The React package holds the react source for components, states, props and all the code that is in React.

The React-DOM glues between React and the React dom.

Q7 What is the difference between react.development.js and react.production.js files via CDN?

⇒ The development build is used for development reasons. You have source map, debugging and often times hot reloading ability in those builds.

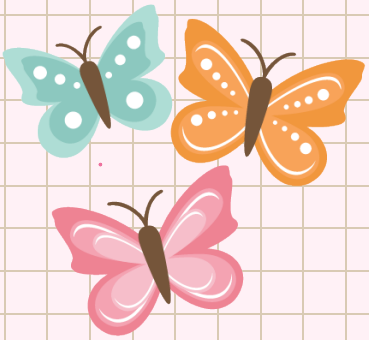
The production build runs in production mode which means this is the code running on your client's machine.

Q8 What are async and defer?

⇒ Async script are executed as soon as the script is loaded, so it doesn't guarantee the order of execution (a script you included at the end may execute before the first script file).

Defer script guarantees the order of execution in which they appear in the page.

Async allows your script to run as soon as it's loaded, without blocking other elements on the page. Defer means your script will only execute after the page has finished loading. In most cases is the better option.



Assignment 2 - Ignite Our App

Q1 What is NPM?

⇒ NPM is a package manager for the JavaScript programming language maintained by npm, Inc.
NPM will take care of the version of the packages. It consists of a command line client, also called npm and an online database of public and paid-for private packages, called the npm registry.

Q2 What is 'Parcel/Webpack'? Why do we need it?

⇒ Parcel/Webpack are the bundlers used mostly for JavaScript or TypeScript code that helps you to minify, clean and make your code compact so that it becomes easier to send a request or recieve the response from the server when it usually take you to transfer multiple files without using any bundler for loading the page of your application

Both of these bundlers sustantially reduce the time it takes for the transfer of data and files to the server from the application. Along with that both bundlers parcel and webpack removes the unnecessary comments, new lines, any kind of block delimiters, and white spaces while the functionality of the code remains unchanged

Q3 What is '.parcel-cache'?

⇒ Cache folders stores information about your project when parcel builds it, so that when it rebuilds, it doesn't have to re-parse and re-analyze everything from scratch. It's a key reason why parcel can be so fast in development mode.

Q4 What is npx?

⇒ NPX stands for Node Package Execute. It is simply an NPM package runner. It allows developers to execute any JavaScript Package available on the NPM registry without Package available on the NPM registry without even installing it.

Q5 What is the difference between dependencies and devDependencies?

⇒ Dependencies are used for production or in testing environment. Whereas devDependencies are used for project development purposes only.

Q6 What is Tree Shaking?

⇒ Tree Shaking is used within a JavaScript context to describe the removal of dead code. By using tree shaking and code splitting together, developers can create smaller, faster, and more efficient React application. These technique can help to eliminate unused code and split large application into manageable chunk, improving the performance and user experience of an application

Q7 What is Hot Module Replacement?

⇒ Hot Module Replacement is a feature that enables you to see code changes in the browser without having to refresh it, allowing you to preserve the state of your frontend application
It is used to retain application state which is used to retain application state which is lost during a full reload.

Q8 List down your favourite 5 superpowers of Parcel and describe any 3 of them in your words

⇒ Tree-shaking → It removes unwanted code like comments, white spaces while sending it to the production for speeding the application

Hot Module Replacement → It updates or removes or adds a module without refreshing the whole page. It make app fast

Caching → It creates caches of files which are built once and reuses the cache file so that rebuild of that file doesn't happen to faster the app.

Bundling

Minification

Q9 What is 'gitignore'? What should we add and not add into it?

⇒ When we don't want anything to not go on Github or production, we use .gitignore

Go to .gitignore
and type /node-modules

Q10 What is the difference between 'package.json' and 'package-lock.json'?

⇒ Package-lock.json keeps the track of exact version of the dependency.
While package.json keeps approx version

Q11 Why should I not modify 'package-lock.json'?

⇒ As it contains exact version of dependencies rather than approximation like package.json. It will automatically regenerate new version when package-json change. It has nested dependencies along with exact version unlike package.json.

Q12 What is 'node-module'? Is it a good idea to push that on Git?

⇒ All the code we fetch from npm. Its like database for our dependency. Dependencies required for a dependency is called Transitive dependencies.

Every dependency will have package.json with its own dev & normal dependencies

Q13 What is the 'dist' folder?

⇒ It is where the compiled code is stored. This is the code that is ready to be deployed to production

Q14 What is 'browserlists'?

⇒ It defines & shares the list of target browsers between various frontend build tools. It is used by autoprefixer, Babel. It is a tool that allows specifying which browser should be supported in your frontend app.

Assignment 3 - Laying the Foundation

Q1 What is Jsx?

⇒ JSX is a syntax extension for JavaScript that lets you write HTML-like markup inside a JavaScript file keeping rendering logic and content in the same place

Q2 What are the superpowers / benefits of JSX?

⇒ 1) JSX prevents from cross-site scripting
2) JSX makes it easier to write code as we are no longer creating elements using React.createElement
3) Makes code simple & elegant
4) Show more useful error and warning message
5) JSX prevents from code injections (attacks)

Q3 What is the Role of 'type' attribute in script tag ? What options can I use there?

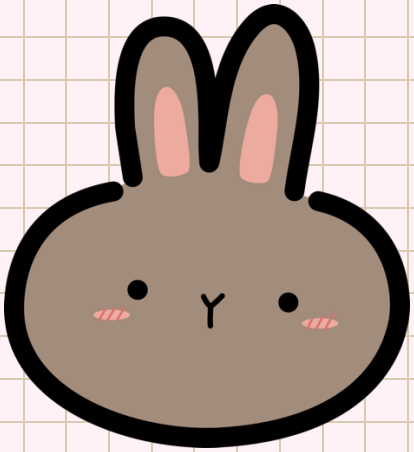
⇒ The type attribute specifies the type of the script. The type of the script. The type attribute identifies the content between the <script> and </script> tag.
It specifies the media type of the script. Media type indicates the format and nature of file.
By default is "application/javascript"

Script types

text/javascript
image/apng
image/avif
image/gif
image/jpeg
image/png
image/svg+xml
image/webg

Audio & video files

multipart/form-data



Q4 What is Babel?

⇒ Babel is JavaScript compiler which is used to convert ECMAScript 2015+ code into a backward compatible version of Js in current and older browsers

Powers of Babel:-

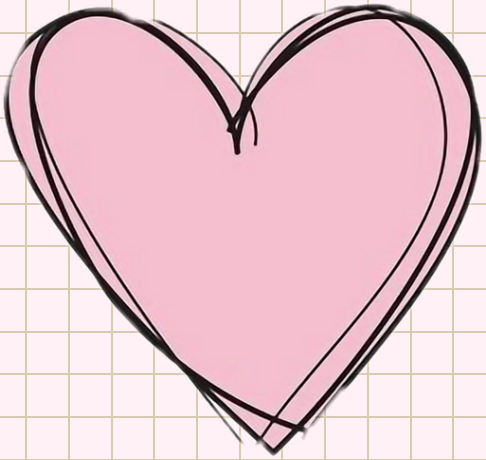
1) Transform syntax
2) Source code transformation
3) Polyfill

Q5 React without JsX

⇒ JSX is not required for using React. Each JSX element is just syntactic sugar for calling React.createElement (component, props, ... children). So anything you can do with JsX can also be done with just plain JavaScript

Q6 What is a React Component?

⇒ Components let you split the UI into independent, reusable pieces and think about each piece in isolation. Components are like javascript functions.



Assignment 3 - Talk is Cheap, Show me the Code

Q1 Is JSX mandatory for React?

⇒ JSX is mandatory but not mandatory for using React. Each JSX element is just syntactic sugar for calling React.createElement(component, props, ...children). So anything you can do with JSX can also be done with just plain JavaScript.

Q2 Is ES6 mandatory for React?

⇒ ES6 is important but not mandatory for using React. ES6 is syntatic sugar for calling create-react-class module

Q3 {TitleComponent()} vs {<TitleComponent/>} vs {<TitleComponent></TitleComponent>}

⇒ All are same, calling a component

Q4 How can I write comments in JSX?

⇒
```
{ /*
    Write your comment here
  */ }
```

Q5 What is <React.Fragment></React.Fragment> and <></>?

⇒ React Fragment often used via <>... </> syntax, lets you group elements without a wrapper node.

Eg

```
<>
  <OneChild />
  <AnotherChild />
</>
```

Wrapper elements in <React.Fragment> or <Fragment> to group them together in situations where you need a single element. Grouping element in Fragment has no effect on the resulting DOM; it is the same as if the elements were not grouped. The empty JSX tag <></> is shorthand for <Fragment></Fragment> in most cases.

Q6 What is Virtual Dom? What is Reconcitiation?

⇒ The Virtual DOM <VDOM> is a programming concept where an ideal, or 'virtual' representation of a UI is kept in memory and synced with the "real" DOM by a library such as ReactDOM. This process is called Reconcitiation aka React Fiber

VDOM is usually associated with React elements since they are they are the objects representing the user interface

VDOM is a representation of actual dom

Q7 What is React Fiber?

⇒ React also uses internal objects called 'fibers' to hold additional information about the component tree.

Fiber is the new reconciliation engine in React 16. It's main goal is to enable incremental rendering of the virtual DOM.

Q8 Why we need keys in React? When do we need keys in React?

⇒ Keys help React identify which items have changed (added/removed/re-ordered). To give a unique identity to every element inside the array, a key is required. We use keys while using map function

If you changes the order of the list items, weired things will happen.

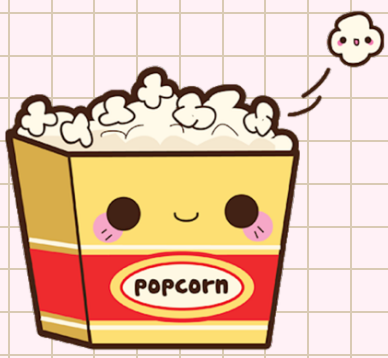
Q9 Can we use index as keys in React?

⇒ Using index as key is not recommended but if there is no unique id you can use index as key.

Q10 What is props in React?

⇒ The props are a type of object where the value of attributes of a tag is stored. The word "props" implies "properties" and its working functionality is quite similar to HTML attributes

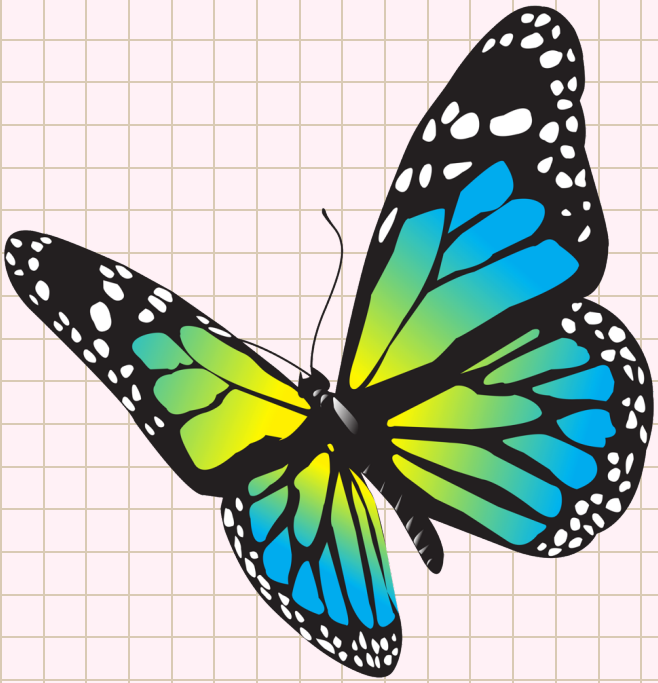
Props are used to store data that can be accessed by the children of a React component.

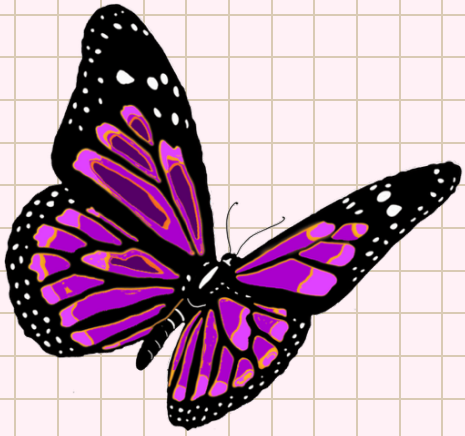


Q11 What is a Config Driven UI?

⇒ A "config-driven UI" is a user interface that is built and configured using a declarative configuration file or data structure, rather than being hardcoded. The configuration file or data structure typically contains information about the structure of the UI, such as the layout of the elements, the properties and behaviors of each element, and any data source or APIs that the UI interact with.

Using a config-driven UI approach, the UI can be easily modified and customized without requiring changes to the underlying codebase.





Assignment 5 - Lets get hooked

Q1 What is the difference between Named Export, Default Export and * as Export ?

⇒ There are two primary ways to export values with JS - Default Export and Named Export
But you can use one or both of them in the same file. A file can have not more than one default export, but it can have as many named exports

Exports

```
export default function Button () {}      // default export

export function Button () {}      // named export
```

Imports

```
import Button from `./button.js'      // default export

import {Button} from `./button.js'      // named export
```

When you write a default import, you can put any name you want after import and it would still provide you with the same default export. In contrast, with named imports, the name has to match on both sides. They are called named imports

Q2 What is the importance of config.js file?

⇒ A configuration (config) file is code on your computer that allows the selection of various features and settings. It can determine parameter, preferences and alternative option in the many realms of your IT environment

Using a JSON file:
You can store the configuration data in a JSON file. Just like a component, the JSON file can be loaded using the import statement. And then you can use the data as any other JavaScript object

Q3 What are React Hooks?

⇒ Hooks let you use different React features from your components. You can either use the built-in Hooks or combine them to build your own. React Hooks acts as a replacement for the class system.

Q4 Why do we need a useState Hooks?

⇒ State lets a component "remember" information like user input

useState declares a state variable that you can update directly. It is a hook that allows you to have state variable in functional components so basically usestate is the ability to encapsulate local state in a functional component.

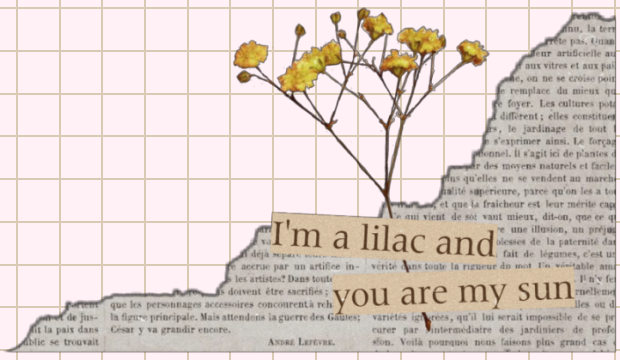
```
Const [state, setstate] = useState(initialState);
```

Q5 What is the diff algorithm in DOM?

⇒ React compares the Virtual DOM and pre-updated Virtual DOM and only marks the sub-tree of components that are update. This process is called diffing. The algorithm behind diffing is called Diffing algorithm React uses keys to avoid unnecessary re-renders

Q6 What is incremental rendering in React?

⇒ React Fiber takes charges of solving problems like it. Bringing a features named "incremental rendering" which split rendering work into chunks and spread it out over multiple frames. The new rendering logic allows a better approximation of the principles of an animation.



Assignment 6 - Exploring the World

Q1 What is a Microservices?

⇒ Microservices architecture is a distinctive method of developing software system that tries to focus on building single-function modules with well-defined interfaces and operations.

Netflix, eBay, Amazon, Twitter, PayPal and other tech stars have all evolved from Monolithic to Microservice architecture.

Q2 What is Monolithic Architecture?

⇒ Unlike Microservices, a Monolith Application is built as a single, autonomous unit. So any changes to the application are slow, as it affects the entire system. A modification made to a small section of code might require building and deploying an entirely new version of software. So to scale a specific function of an application also means you have to scale the entire application.

Q3 What is the difference between Monolithic and Microservice?

⇒ A monolithic application is built as a single unified unit while a monolithic architecture is a collection of smaller, independently deployable services.

Q4 Why do we need a useEffect Hook?

⇒ The useEffect Hook allows you to perform side effects in your components. Some example of this side effects are: fetching data, directly updating the DOM, and timers.
By using this Hook, you tell React that your component needs to do something after render. React will remember the function you passed and call it later after performing the DOM updates.

Q5 What is Optional Chaining?

⇒ You can use Optional Chaining when attempting to call a method which may not exist. This can be helpful, For eg: when using an API in which a method might be unavailable, either due to the age of the implementation or because of a feature which isn't available on the user's device.

Q6 What is Shimmer UI?

⇒ A Shimmer UI resembles the page's actual UI, so users will understood how quickly the web or mobile app will load even before the content has shown up. It gives people an idea of what's about to come and what's happening (its currently loading) when a page full of content/data takes more than 3-5 seconds to load.

Q7 What is the difference between Js Expression and Js Statement?

⇒ Js Expression
Any line of code that produces some value is expression

Eg
→ x + 10
→ square(x)

In short, evaluation of some part of code (it could be block, function call, one/many line of code) should produce value.

Js Statement
Statement is any line of code that performs some action

Eg
→ console.log('Hello World')
→ let y = if (x > 10) { return 25; } else { return false }

Q8 What is Conditional Rendering, explain with a code example

⇒ Conditional Rendering in React works the same way conditions work in Js. Use Js operators like to create elements representing the current state, and let React updates the UI to match them

Eg
```
if (isLoggedIn) {
    return <UserGreeting />;
}
return <GuestGreeting />
}
```

Q9 What is CORS?

⇒ Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any origins (domains, scheme, or port) other than its own from which a browser should permit loading resources.

Q10 What is async-await?

⇒ Async function return a promise. This promise state can be either resolved or rejected. Await suspends the called function execution until the promise returns a result for that execution.

Q11 What is the use of 'const json = await data.json();' in getRestaurants()

⇒ Converting the data into json form and we have to use await as it is an async function

Assignment 7 - Finding the Path

Q1 What are various ways to add images into our App? Explain with code Eg

⇒ Image Tag

```
<img src="./house.jpg" height="200" width="200" />
```

Import Keyword

```
import house from "./house.jpg";

<img src={house} alt="House img"/>
```

Require Keyword

```
<img src={require('./house.jpg')} />
```

Add SVG file

```
import {ReactComponent as Logo} from './logo.svg';

<Logo/>
```

Add Network Images

```
<img src={"https://micro.medium.com/..."} />
```

Add Image from Cloudinary

1) Creating an account on Cloudinary
2) Create the react Js project
3) Install the node modules for Cloudinary Image Upload
   // npm install cloudinary
   // npm install axios
4) Create Image Uploads Component Code
5) Create Avatar Uploads Function
   Search on internet
6) Include Image Upload Component in App.js file
7) Upload the Image

Or

1) Upload Image on Cloudinary
2) Paste the link of images in React App
3) Import Whenever needed

Q2 What would happen if we do console.log(useState())?

⇒ [undefined, function()]

Q3 What will useEffect behave if we don't add a dependency array?

⇒ If we don't have a dependency array it will be called everytime the component renders

Q4 What is SPA?

⇒ SPA is a Single-page Application, a webpage that dynamically interacts with the web browser by rewriting the current web page with data from the webserver

Q5 What is difference between Client Side Routing and Server Side Rendering?

⇒ Client-side Routing
It involves React Router. The routing logic is handled on the client side, within the browser. When a user clicks on a link or performs a navigation action, React Router intercepts the request and renders the appropriate components without a full page reload.

Server-side Routing
It involves handling the routing logic on the server before delivering the final HTML to the client. Server side routing framework like Next.js or Gatsby.js. When a user requests a specific URL, the server generates a specific URL, the server generates the corresponding HTML and delivers it to the client.

Assignment 8 - Let's Get Classy

Q1 How do you create Nested Routes react-router-dom configuration?

⇒ In React Router Dom, createBrowserRouter() has children property where it takes an array of Route objects with nested routes on the children property

// Code

```
createBrowserRouter([
    {
        path: "/",
        element: <AppLayout/>,
        errorElement: <Error/>,
        children: [
            {
                path: "/",
                element: <Body/>,
            },
            {
                path: "about",
                element: <About/>,
                children: [{
                    path: "profile",
                    element: <Profile/>
                }]
            }
        ]
    }
])
```

"/" means from the root

It is not "/profile" but "about/profile"

To render this we need to create an outlet in the parent element means inside <About/> or we can directly import Profile component inside Parent

Q2 What is the order of life cycle methods calls in Class Based Components?

⇒ There are 2 phases in React Life Cycle Methods:
Render Phase and Commit Phase

1) Render Phase:
   constructor() → A special method for creating and initializing an object created with a class
   render() → Invoked to examine this.props and this.state in constructor()

React first tries to batch up render phase (render phase of all children as well if present) then commit phase is called.

2) Commit Phase
In commit phase, Mounting, Updating and Unmounting stages are performed

- Mounting → componentDidMount()
  Invoked immediately after a component is inserted into the tree (mounted)

- Updating → componentDidUpdate()
  Called everytime after any update

Q3 Why do we use componentDidMount()?

⇒ This method is not called for the initial render. This is the best lifecycle for DOM and setstate updates. This is also an excellent place to make network requests as long as you compare the current props to previous props.
It is first called in Mounting phase and then everytime in Updating Phase.

Q4 Why do we use componentWillUnmount? Show with example.

⇒ componentWillUnmount() is invoked immediately before a component is unmounted and destroyed. Perform any necessary clean up in this method, such as invalidating timers, cancelling network requests, or cleaning up any subscriptions created in componentDidMount()

You should not call setState() in componentWillUnmount() because the component will never be re-rendered. Once a component instance is unmounted, it will never be mounted again

Eg
Unmouting in Functional Component:

```
useEffect(() => {
    const timer = setInterval(() => {
                    console.log("Namaste React")
                  },1000)

    timer();

    return () => {
                    console.log("useEffect Return")
                    clearInterval(timer)
                 };, []);
```

Unmounting means removing from the page

Q5 Why do we use super (props) in constructor?

⇒ super (props) is used to inherit the properties and access variable of the React parent class when we initialize our component. super () is used inside constructor of a class to derive the parent's all properties inside the class that extended it. If super () is not used, then Reference Error. Must call super constructor in derived classes before accessing 'this' or returning from derived constructor is thrown in the console. The main difference between super () and super (props) is the this.props is undefined in child's constructor in super() but this.props contains the passed props if super(props) is used.

Q6 Why can't we have the callback function of useEffect async?

⇒ useEffect expects it's callback function to return nothing or return a function to return nothing or return a function (cleanup function that is called when the component is unmounted). If we make the callback function as async, it will return a promise and the promise will affect the clean-up function from being called.

Assignment 9 - Optimizing Our App

Q1 When and why do we need Lazy()?

⇒ React.lazy() allows developers to easily create components with dynamic imports and renders them as normal components. When the component is rendered, it will automatically load the bundle that contains the rendered component.

You need to provide a single parameter to call React.lazy(). It accepts a function as an input parameter, and that function should return a promise after loading the component using imports(). Finally, the returned promise from React.lazy() will give you a module with a default export containing the React component.

// Code

```
const AboutComponent = React.lazy(() => import('./AboutComponent'));
```

Q2 What is Suspense?

⇒ When we use lazy loading, components are rendered as we navigate. So, we need to have a placeholder for those components until they are loaded. As a solution, React.suspense was introduced and it acts as a wrapper for the lazy component.

You can wrap a single lazy component, multiple lazy components, or multiple lazy component with different hierarchy level with React.Suspense. In addition, it accepts a prop named fallback as the placeholder, and you can pass a component or an element for that

Q3 Why we got this error: A component suspended while responding to synchronous input. This will cause the UI to be replaced with a loading indication. To fix, updates that suspend should be wrapped with startTransition? How does suspense fix this error?

⇒ This error is typically encountered in asynchronous contexts where components are fetching encountered in asynchronous contexts where components are fetching data or handling code splitting

Suspense is used to manage async data and code-splitting, allowing you to display a loading indicator while the data or code is being fetching.

This error is telling you that a component that was responding to synchronous input (meaning its not supposed to be waiting for anything) encountered a suspension. This should not happen because Suspense is primarily designed to handle asynchronous operations, and you generally don't want to introduce

delays in the rendering of synchronous user interactions.

How to fix this issue:

1) You should identify which part of your code is causing the synchronous component to suspend. This could be due to a network request, a dynamic import of a component, or another asynchronous operation

2) Ensure that the asynchronous code which might suspend, is wrapped within a suspense boundary and that you provide a fallback UI to display while waiting for the operation to complete

Q4 Advantages and disadvantages of using this code splitting pattern?

⇒ Code splitting is a technique used to break down a large monolithic JavaScript bundle into smaller, more manageable pieces, which can be loaded on demand

Advantages:

1) Smaller initial bundles result in faster load times for your web app.

2) Code Splitting can lead to better performance, as smaller bundles can be passed and executed more quickly. This can reduce the time it takes to render the initial page and improve the overall responsiveness of the application

3) Components or features that are rarely used may never be loaded unless needed. This conserves bandwidth and memory, making your application more efficient

4) Simpler Maintenance. You can make updates to a specific parts of your application, you can be more confident that you won't introduce unexpected issue in unrelated components.

Disadvantages:

1) Setting up code splitting and configuring it correctly can be complex, especially in large applications. You may need to make adjustments to your build tools and bundler settings

2) When a component is loaded on-demand, there may be a slight delay the first time it is needed, which can impact user perception of your application speed. However, this delay is usually minimal

3) Handling asynchronous loading and rendering of components requires careful design to ensure a seamless user experience. You need to consider scenarios such as loading indicators and error handling

4) Not all frameworks and libraries have built-in support for code-splitting

Q5 When do we and why do we need suspense?

⇒ There will be a slight delay when the code split component is fetched, so it is important to display a loading state otherwise it will not render the page & show error

1) Suspense helps to display fallback UI components like shimmer component while data is being fetched or code is being loaded

2) Increases performance as it allows you to load code and data only when its needed reducing the initial bundles size and making your app faster to load

3) Use suspense for data fetching when you want to make your application more responsive and provide a smooth loading experience for data-driven components. It simplifies the management of loading states and error handling

4) Use suspense for code splitting when you want to improve your application's initial load time and performance. It allows you to load parts of your application on-demand which can lead to faster rendering times and better resource usage

Assignment 10 - Joh Dikhta Hai Wahi Bikta Hai

Q1 Explore all the ways of writting CSS?

⇒ CSS can be added to HTML document in 3 ways:

1) Inline Css
An inline CSS is used to apply a unique style to a single HTML element. An inline CSS uses the style attribute of an HTML element

Eg
```
<h1 style="color:blue;"> A Blue Heading </h1>
```

2) Internal Css
An internal CSS is used to define a style for a single HTML page. An internal CSS is defined in the <head> section of an HTML page, within a <style> element

Eg
```
<html>
    <head>
        <style>
            body {background-color: blue;}
        </style>
    </head>
    <body>
        <h1> This is a heading </h1>
    </body>
</html>
```

3) External Css
An external stylesheet is used to define the style for many HTML pages

Code
```
<head>
    <link rel="stylesheet" href="style.css"/>
</head>
```

The external stylesheet can be written in any text edition. This file must not contain any HTML code, and must be saved with a .css extension.

Q2 How do we configure tailwind CSS?

⇒ Step1: Create a new project
Step2: Install Tailwind CSS
You can install Tailwind CSS using npm or yarn. Open your terminal or command prompt and navigate to your project's root directory.

Run one of the following command :

Using npm:

⇒ npm install tailwindcss

Step 3: Create a Configuration File

You can generate a basic configuration file using the following command:

⇒ npm tailwind init

This command creates a tailwind.config.js file in your project's root directory

Step 4: Customize the Configuration (Optional)

Open the generated tailwind.config.js file, and you customize various aspects of Tailwind CSS according to your project's needs. This file includes options for colors, fonts, spacing, breakpoints, and more.

```
//Code
module.exports = {
  theme: {
    extend: {
      colors: {
        primary: '#3490dc',
        secondary: '#ffed4a',
      },
    },
  },
};
```

Step 5: Create CSS File

Create a CSS file where you will import Tailwind CSS and any additional styles. Typically, this file is named styles.css or similar. Import Tailwind CSS using the @import directive

```
/* styles.css */
@import 'tailwindcss/base';
@import 'tailwindcss/components';
@import 'tailwindcss/utilities';
```

Step 6: Build Your Styles

Include your CSS file in your HTML or import it in your JavaScript file if you are using a bundler like Webpack

Step 7: Use Tailwind CSS Classes in HTML

You can start using Tailwind CSS classes in your HTML files to apply styles.

```
<div class="bg-primary text-white p-4">
    This is a primary-colored box with white text and padding.
</div>
```

Step 8: Build Your Project

Depending on your setup, you might need to build your project to apply the Tailwind CSS styles. If you're using a bundler like Webpack, make sure to run the appropriate build command.

With npm:

npm run build or npm start

Q3 In tailwind.config.js, what does all the keys mean (content, theme, extend, plugins)?

⇒ In tailwind.config.js, the various keys serve different purposes and allow you to customize and configure different aspects of Tailwind CSS.

1) content Key:

It specifies the files that Tailwind CSS should analyze to generate its utility classes.

Usage:

```
// Code
module.exports = {
  content: [
    './src/**/*.html',
    './src/**/*.js',
  ],
}
```

The content key helps Tailwind CSS identify which files to process and extract utility classes from. It is particularly useful when working with frameworks like React or Vue

2) theme Key:

It defines the default styles and configurations for various aspects of Tailwind CSS, such as colors, spacing, fonts, and more.

Usage:

```
// Code
module.exports = {
  theme: {
    extend: {
      colors: {
        primary: '#3490dc',
        secondary: '#ffed4a',
      },
    },
  },
};
```

The theme key allows you to customize default styles and extend or override the default configuration provided by Tailwind CSS. It is where you can define your project-specific design system.

3) extend Key:
It extends or overrides the default configuration provided by Tailwind CSS. The extend key is often used to add new styles or extend existing ones. It is especially useful for adding project-specific utility classes or modifying existing ones.

4) plugins Key:
It allows you to use or define custom plugins to extend or modify Tailwind CSS functionality.
Usage:

```
// Code
module.exports = {
  plugins: [
    require('@tailwindcss/forms'),
  ],
};
```

The plugins key lets you incorporate third-party plugins or create your own custom plugins. Plugins can add new features, styles, or utilities to Tailwind CSS.

Q4 Why do we have .postcssrc file?

⇒ The .postcssrc file, often named postcss.config.js, is a configuration file for PostCSS. PostCSS is a tool for transforming styles with JavaScript plugins, and it is commonly used in conjunction with build tools like webpack or parcel for processing and optimizing CSS.

Uses of .postcssrc

1) To configure the plugins that PostCSS use. These plugins can handle tasks such as autoprefixing, minification, and syntax enhancements

2) To behavior of PostCSS beyond the default settings provided by the build tool (eg. webpack). This allows you to have fine-grained control over the PostCSS transformations.

3) It is a convenient place to define these options and presents.

4) Separating the PostCSS configuration into its own file makes the build configuration more maintainable and organized. It allows you to centralize PostCSS-related settings and keep them distinct from other build tool configurations

Assignment 11 - Data is the New Oil

Q1 What is prop drilling?

⇒ Prop drilling refers to the process of passing down props (short for properties) through multiple layers of nested components. This happens when a piece of data needs to be transferred from a higher-level component to a deeply nested child component, and it must pass through several intermediary components in between.

Q2 What is lifting the state up?

⇒ Lifting the state up in React refers to the practice of moving the state from a lower-level (child) component to a higher-level (parent or common ancestor) component in the component tree. This is done to share and manage state across multiple components.

When a child component needs access to certain data or needs to modify the data, instead of keeping that data and the corresponding state management solely within the child component, we move the state to a shared ancestor component. By doing so, the parent component becomes the source of truth for the state, and it can pass down the necessary data and functions as props to its child components.

Q3 What are Context Provider and Context Consumer?

⇒ In React, the Context API provides a way to pass data through the component tree without having to pass props manually at every level. The two main components associated with the Context API are the Context Provider and Context Consumer.

Context Provider: It is a component that allows its children to subscribe to a context's changes. It accepts a value prop, which is the data that will be shared with the components that are descendants of this provider. The Provider component is created using React.createContext() and then rendered as part of the component tree. It establishes the context and provides the data to its descendants.

Context Consumer: It is a component that subscribes to the changes in the context provided by its nearest Context Provider ancestor. It allows components to access the context data without the need for prop drilling. The Consumer component is used within the JSX of a component to consume the context data. It takes a function as its child, and that function receives the current context value as an argument.

Q4 If we don't pass a value to the provider does it take the default value?

⇒ Yes

Assignment 12 - Let's Build Our Store

Q1 useContext vs Redux

⇒ useContext and Redux are both tools used for state management in React applications, but they serve different purposes and have different use cases

useContext
This is a hook provided by React, used to manage global state or pass data across components in React applications. useContext allows data to be passed through the component tree without the need for prop drilling, typically suitable for managing a smaller amount of global state in smaller or medium-sized application.

redux
It's a state management library that provides a single global state store to manage the entire application's state. Redux employs the concepts of action and reducers to update state and can be used in conjunction with React. However, it requires specific architecture and setup, commonly employed in larger or more complex applications or scenarios requiring intricate state management.

Q2 Advantages of using Redux Toolkit over Redux?

⇒ Redux Toolkit is a set of utility functions and abstractions that simplifies and streamlines the process of working with Redux. It is designed to address some of the common pain points and boilerplate associated with using plain Redux.

Advantages:

1) It helps us write less code. It provides shortcuts that save us from typing a lot of repetitive code

2) It makes things like fetching data simpler. It has a tool called createAsyncThunk that handles async actions in a way that's easy to understand and use.

3) Setting up your Redux store is easier with Redux Toolkit. It has a function called configureStore that simplifies the process, and it comes with sensible defaults, so you don't have to configure everything from scratch.

4) It has built-in debugging support. Enabling it is as easy as adding one line of code when setting up your store.

5) Working with immutable data is usually a bit tricky. It uses a library called Immer to handle this behind the scenes, so you can write more straightforward and readable code.

Q3 Explain Dispatcher?

⇒ In Redux, a dispatch is not a standalone concept; instead it's a term often used to refer to a function called dispatch. We use it to send actions to the store. An action is a plain JavaScript object that describes what should change in the applications state.

When we want to update the state in our Redux store, we create an action and dispatch it using the dispatch function.

```
// Code
const myAction = { type: 'INCREMENT' };
store.dispatch(myAction);
```

Q4 Explain Reducer?

⇒ In Redux Toolkit, the createSlice function is commonly used to create reducers. It simplifies the process of defining actions and the corresponding reducer logic, reducing boilerplate code.

We use createSlice to define a "slice" of your Redux store. A slice includes actions, a reducer, and the initial state. Reducers take in two things: previous state and an action. Then it returns the new updated instance of state. Whenever an action is dispatched, all the reducers are activated. Each reducer filters out the action using a switch statement switching on the action type. Whenever the switch statement matches with the action passed, the corresponding reducers take the necessary action to make the update and return a fresh new instance of the global state.

Q5 Explain Slice?

⇒ In Redux Toolkit, a slice is a collection of Redux-related code, including reducer logic and actions, that corresponds to a specific piece of the application state. Slices are created using the createSlice utility function provided by Redux Toolkit. The primary purpose of slices is to encapsulate the logic related to a specific part of the code more modular and easier to manage.

It auto generates your action creators and types which you have to define them as constants before redux-toolkit.

Q6 Explain Selector?

⇒ In Redux Toolkit, a selector is a function that extracts specific piece of data from the Redux store. It allows you to compute derived data from the store state and efficiently access specific parts of the state tree.

Redux Toolkit provides the createSlice and createAsyncThunk utilities along with the createSelector function from the reselect library to help manage selectors easily.

Selectors encapsulate logic for data retrieval, optimizing performance, and promoting code reusability. By using, memoization, selectors cache results to prevent unnecessary re-renders, thus enhancing overall application efficiency.

Q7 Explain createSlice and the configuration it takes

createSlice() → A function that accepts an initial state, an object of reducer functions, and a "slice name", and automatically generates action creators and action types that correspond to the reducers and state

initialState → The initial state value for the slice. This is the starting point for your state before any actions are dispatched.

reducers → An object where each key-value pair represents a reducer function. The keys are the names of the actions, and the values are the reducer logic.

//Code

```
import {createSlice} from '@reduxjs/toolkit'

const initialState = {value: 0}

const counterSlice = createSlice({
    name: 'counter',
    initialState,
    reducers: {
        increment(state) {
            state.value++
        },
        decrement(state) {
            state.value--
        },
    },
})

export const {increment, decrement} = counterSlice.actions
export default counterSlice.reducer
```

Assignment 13 - Time for the Test

Q1 What are different types for testing ?

⇒ Software testing is the process of finding errors in a product, whether it be a mobile or web application. Errors include bugs in the code, missing requirements, glitches, and more. Software testing can also determine whether the outcome when engaging with the application differs from the expectation.

→ Manual Testing:
It is done in person, by clicking through the application or interacting with the software and APIs with the appropriate tooling. This is very expensive since it requires someone to setup an environment and execute the tests themselves, and it can be prone to human error as the tester might make typos or omit steps in the test script.

→ Automated Testing:
It is performed by a machine that executes a test script that was written in advance. These tests can vary in complexity, from checking a single method in a class to making sure that performing a sequence of complex actions in the UI leads to the same results. It's much more robust and reliable than manual tests - but the quality of your automated tests depends on how well your tests scripts have been written.

There are different types of tests, some popular types are described below:

1) Unit tests:
They consist in testing individual methods and functions of the classes, components or modules used by your software. Unit tests are generally quite cheap to automate and can run very quickly by a continuous integration server.

2) Integration tests:
It verify that different modules or services used by your application work well together. For example, it can be testing the interaction with the database or making sure that microservices work together as expected. These types of tests are more expensive to run as they require multiple parts of the application to be up and running.

3) End-to-end tests:
It replicates a user behavior with the software in a complete application environment. It verifies that various user flows work as expected and can be as simple as loading a web page or logging in or much more complex scenarios verifying email notification, online payments, etc.

End-to-end tests are very useful, but they're expensive to perform and can be hard to maintain when they're automated.

Q2 What is Enzyme?

⇒ Enzyme is a JavaScript testing utility that was created by Airbnb. It is designed to make it easier to test React components by providing a set of tools for asserting, manipulating, and traversing React components output. Enzyme simplifies the testing process by allowing developers to interact with components as if they were in the browser.

Enzyme provides three different rendering methods: shallow rendering, full DOM rendering, and static rendering. Shallow rendering allows developers to render only the component they are testing, while full DOM rendering renders the entire component tree. Static rendering is used to render components without a DOM.

Q3 Enzym vs React Testing Library

⇒

	React Testing Library	Enzyme
1)	Developed in 2017 by Kent C. Dodds and the React community to encourage testing in a way that reflects how users interact with the application	Developed by Airbnb in 2015 to provide a comprehensive testing utility for React components.
2)	Uses a lightweight DOM testing library	Offers both shallow rendering and full rendering capabilities
3)	RTL Uses queries like getBy, queryBy, findBy for more user-centric testing	Provides utility functions for DOM manipulation and querying, Enabling precise control
4)	RTL Focuses on testing components in a more isolated manner	Allows shallow rendering to isolate the tested component. It can also perform full rendering.
5)	Limited direct access to component state and props	Enzyme allows direct access to component state and props
6)	Encourages making assertions based on user interaction and visible changes in the DOM.	Supports various assertions, including snapshot testing for both shallow and deep tests
7)	Gained popularity for its simplicity and user-centric approach. Widely used in React community	Popular for its longer presence, extensive community and comprehensive documentation
8)	No direct integration with Jest; can be used independently or with other test runners.	Officially integrated with Jest, making it seamless for projects using Jest as the test runner.

Q4 What is Jest and why do we use it?

⇒ Jest is a JavaScript Testing Framework by Facebook. It is used mostly commonly for Unit Testing. It is when you provide input to a unit of code (usually, a function) and match the output with the expected output.

Jest is a Node environment attempting to mock out a real DOM via JsDOM. It's essentially a hybrid framework built on JavaScript, designed majorly to work with React and React Native-based web applications. It can validate almost everything around JavaScript, especially the browser rendering of web applications.