**Lecture 1**

Import React CDN

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>React</title>  
</head>  
<div id="root"></div>  
  
<script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>  
<script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>  
  
<script src="./index.js"></script>  
  
</body>  
</html>

const ***parent*** = React.createElement("div", {id: "parent"},  
 [  
 React.createElement("div", {id: "child1"}, [  
 React.createElement("h1", {}, "Hello H1 Tag"),  
 React.createElement("h2", {}, "Hello H2 Tag")  
 ]),  
  
 React.createElement("div", {id: "child2"}, [  
 React.createElement("h1", {}, "Hello H1 Tag"),  
 React.createElement("h2", {}, "Hello H2 Tag")  
 ])  
 ]);  
  
const ***root*** = ReactDOM.createRoot(***document***.getElementById("root"));  
  
***root***.render(***parent***);

Using core react through CDN. It is simple as JavaScript. Rendering nested html in root using React CDN.

A screenshot of a computer

Description automatically generated

**Lecture 2**

**package.json** is a configuration of NPM.

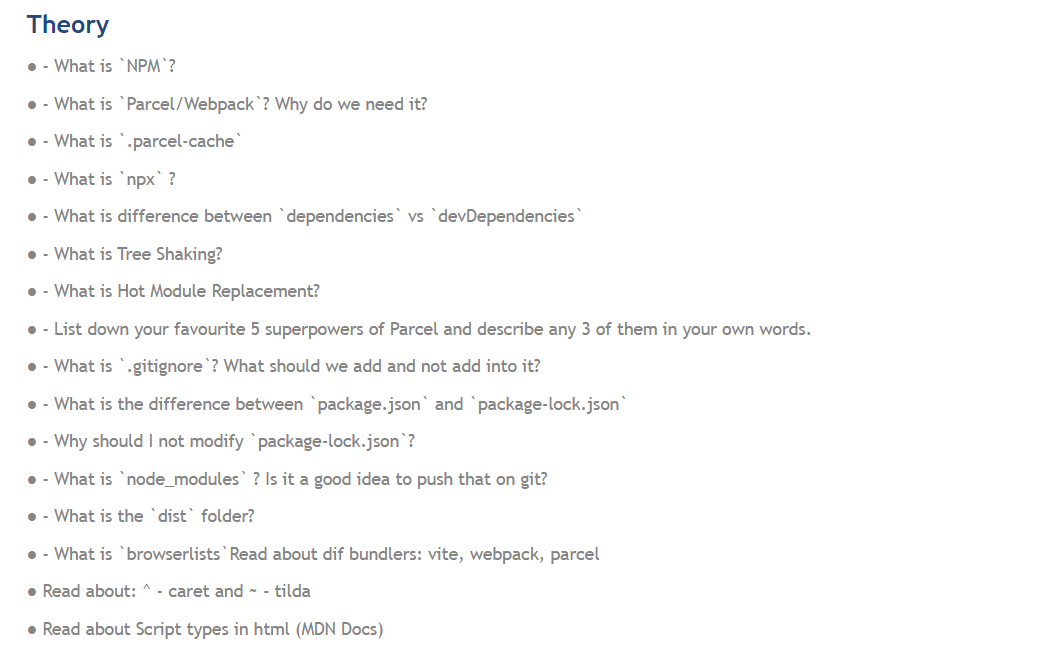
**Bundlers (Webpack, parcel, vite):** These are the bundles which helps to package your app to ship it to production.

**caret vs tilde npm:** ^ (Caret) is used to update minor versions. ~ is used to update major versions. It is best to use caret.

**Node Modules** are the collection of dependencies.

**Npm** for installing package.

**Npx** for executing package.



**Lecture 3**

**JSX** is not html in JavaScript. It is HTML like syntax.

**JSX** converts code to React.createElement.

How JSX converts? Babel transpiles it to React.createElement and then ReactElement Object to HtmlElement(render)

Who is covert JSX to React.createElement? BABEL is converting.

camelCase attributes in JSX like className, tabIndex.

**Component Composition** means calling one component in another component.

JSX prevents cross site scripting attacks.

**Lecture 4**

**Destructing Arrays**

**Before:**

const vehicles = ['mustang', 'f-150', 'expedition'];

const car = vehicles[0];

const truck = vehicles[1];

const SUV = vehicles[2];

**After destructuring:**

const vehicles = ['mustang', 'f-150', 'expedition'];

const [car, truck, SUV] = vehicles;

**Config Driven UI**

Configuration-driven UI, also known as config-driven UI or configuration-based UI - When you build real a world application so you want should your website work in many country or many place we control our frontend it is known as using Config-Driven UI . API or Backend Driven which is data Coming from API.

or

In a configuration-driven UI, the layout, styles, and other properties of UI elements are defined in a configuration file or database, which can be easily modified without requiring changes to the codebase. This approach makes it easier to customize the UI for different use cases or user groups, without the need for extensive coding.

The configuration file or database may also define the data sources and the data to be displayed in the UI, as well as the interactions and behavior of the UI components. This allows for greater flexibility and adaptability of the UI to different use cases, as the configuration data can be easily modified or replaced without affecting the underlying application logic.

**Array Join**

Used to make array comma separated.

E.g. const arr = [‘Apple, ‘Banana’, ‘Orange’];

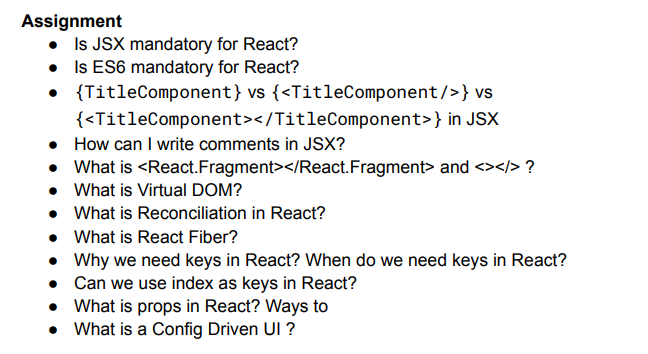
arr.join(‘, ’);

//output

Apple, banana, orange

Never use index in map or loops in React. Always use key. If data don’t have id then use index but it is not recommended by React.

<div className="restaurant-container">  
 {resList.map((restaurant) => (  
 <RestaurantCard key={restaurant.info.id} resData={restaurant}/>  
 ))}  
</div>



**Lecture 5**

To put comments inside JSX, you use the syntax {/\* \*/} to wrap around the comment text.

React use ES6.

The <></> syntax is shorthand of <React.Fragment> with a little difference. you Can't pass **key** or **attributes** to <>

**Virtual DOM:**

React creates a VIRTUAL DOM in memory. Instead of manipulating the browser's DOM directly, react creates a virtual DOM in memory, where it does all the necessary manipulating, before making the changes in the browser DOM. React only changes what needs to be changed! It a virtual representation of actual DOM.

**Reconciliation in React:**

The reconciliation algorithm, often referred to as the **diffing algorithm**, is responsible for comparing the previous virtual DOM with the new one and determining the most efficient way to update the actual DOM. React aims to make this process as optimized as possible to ensure that UI updates are fast and seamless.

The algorithm React uses to diff one tree with another to determine which parts need to be changed. Just like git only replace specific code with new code.

Virtual DOM and Reconciliation: React creates a copy of original dom to avoid making direct changes in the original. When a change occurs in any state, a new copy of virtual dom is created reflecting that change, Now react will do diffing on the new and earlier virtual dom to specify the changes, And now it will batchup changes of several virtual doms finally reflect it in the original DOM. This process is called Reconciliation. Diffing Algorithm   
  
1- If the root or parent element has changed, its child elements are considered to be changes as well and will be included in the reconciliation.   
2- When checking li elements, the diff algorithm checks all elements to match for any changes or removed or new element. This takes a toll on performance and thats why keys are used when printing list items as such, to make the differentiation easier and faster.

**Two types of export/import**

1. Export default name of component --- Import component from path.
2. Named const Component --- Import {component} from path.

**React Hooks**

Normal JavaScript functions

2 most common hooks:

useState() - Superpowerful react variables.

useEffect() -

When state variable changes react re-render my component.

**Lecture 6**

**Monolithic architecture vs Microservices architecture:**

A monolithic application is built as a single unified unit while a microservices architecture is a collection of smaller, independently deployable services.   
  
**Separation of concerns / Single responsibility principal:**  
  
Separate services into micro services. Each micro service has a project, and all micro services combine to make a complete project. Every service has its own job.

**Approach:**

**1)**

**Loads**

**Render**

**API**

**2)**

**Re-Render**

**API**

**Loads**

**Render**

**2nd one Is** **Better Approach and Better UX.**

**useState or useEffect** are simple JavaScript functions.

**UseEffect** will take two arguments. 1st argument is callback functions and 2nd argument is dependency array.

useEffect will call after body function render.

// Optional chaining (?) -> data?.data?.data

**useState** updates the variable and re-render the component but only those things will change which we want to change.

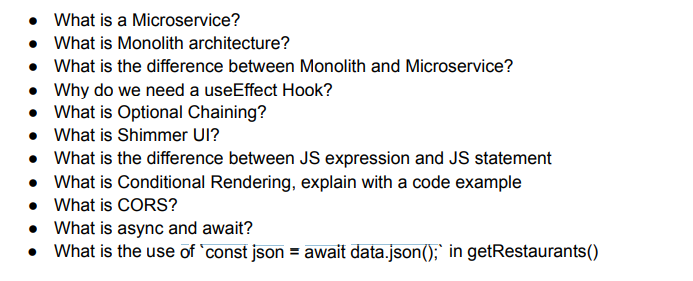
Whenever state variables update, react triggers a reconciliation cycle(re-render the component).

**E.g. In header I have login button. I changed login button to logout with useState. Header component re-render but only button changes. This is reconciliation. This is why react is fast.  
React finds the old html and compare with new html and only change the changed part does not replace the full new html.**

**In input field onChange, component is rendering again and again on every key press.**

**Shimmer UI**

Best practice to use instead of spinner.



**Lecture 7**

useEffect(() => {  
 ***console***.log("useEffect");  
}, []);

IF no dependency array 🡺 useEffect will call on every render.

IF dependency array is empty = [] 🡺 useEffect will call on initial render.

IF dependency array has [btnName], then It will call every time when btnName is updated.

**React Router DOM:**

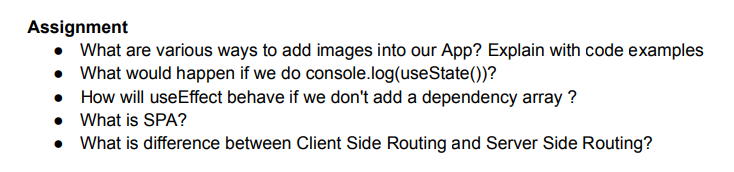
import {createBrowserRouter, RouterProvider} from "react-router-dom";

const appRouter = createBrowserRouter([  
 {  
 path: "/",  
 element: <AppLayout/>,  
 errorElement: <Error/>,  
 children: [  
 {  
 path: "/",  
 element: <Body/>,  
 },  
 {  
 path: "about",  
 element: <About/>,  
 },  
 {  
 path: "contact",  
 element: <Contact/>,  
 }  
 ]  
 }  
])

const root = ReactDOM.createRoot(***document***.getElementById("root"));  
  
root.render(<RouterProvider router={appRouter} />);

**Types of Routing:**

* Client-side routing
* Server-side routing



**Lecture 8**

**Functional Component:**

A simple JavaScript function function which returns JSX.

import React from 'react';  
  
const User = () => {  
 return (  
 <div className="user-card">  
 <h2 className="user-name">Name: Daniyal</h2>  
 <h3 className="user-location">Location: Lahore</h3>  
 <h3 className="user-contact">Contact: 03247733301</h3>  
 <p className="user-description">  
 Lorem ipsum dolor sit amet, consectetur adipisicing elit. Adipisci animi commodi dolores  
 eligendi et, ex ipsum itaque laboriosam nam, nisi nostrum odit quaerat ratione recusandae saepe sint  
 tempora veritatis voluptates!  
 </p>  
 </div>  
 );  
};  
  
export default User;

**Class Component:**

A simple JavaScript class which extends React component and have render function which return JSX.

import React from 'react';  
  
class User extends React.Component{  
 render(){  
 return (  
 <div className="user-card">  
 <h2 className="user-name">Name: Daniyal</h2>  
 <h3 className="user-location">Location: Lahore</h3>  
 <h3 className="user-contact">Contact: 03247733301</h3>  
 <p className="user-description">  
 Lorem ipsum dolor sit amet, consectetur adipisicing elit. Adipisci animi commodi dolores  
 eligendi et, ex ipsum itaque laboriosam nam, nisi nostrum odit quaerat ratione recusandae saepe sint  
 tempora veritatis voluptates!  
 </p>  
 </div>  
 );  
 }  
}  
  
export default User;

**Class based components:**

LifeCycle of class-based components. It has two phases Render phase and Commit phase.

**Render Phase**

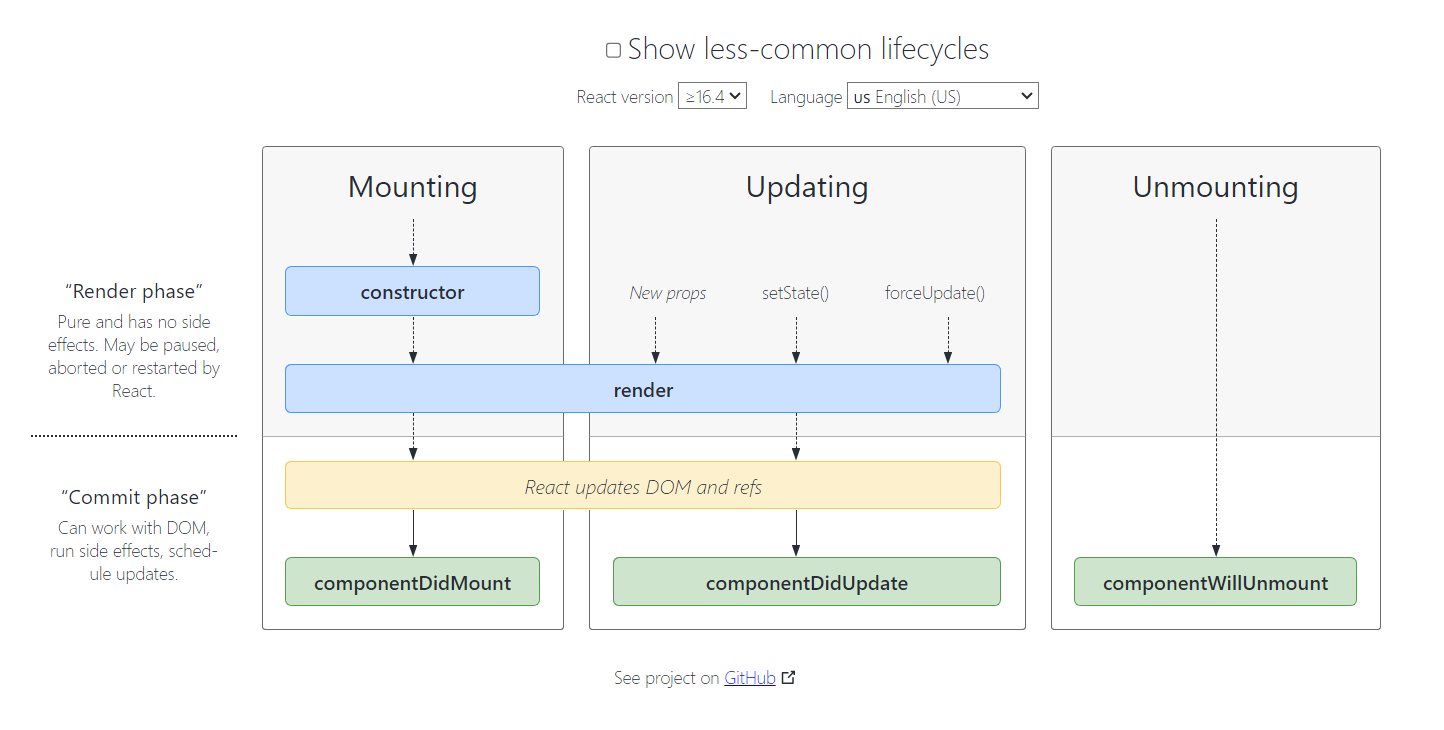
In **Mounting,** first Constructor will call then Render will call.

**Commit Phase**

React Updates the DOM then componentDidMount will call.

**Constructor** --> **Render** --> **DOM updated -**-> **componentDidMount.**

API will call in componentDidMount because it calls after component render.



A close-up of a text

Description automatically generated

**Lecture 9**

**Single Responsibility Principal:** Every component has its own responsibility.

**Lazy Loading**

**When your app is big and extending, then we should use lazy loading. It helps to break down bundle of code in small files to make app fast.**

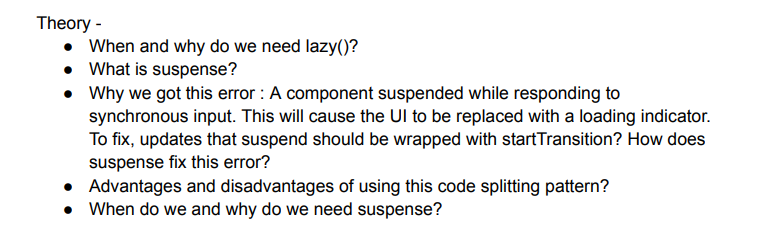
**Chunking**

**Code Splitting**

**Dynamic Bundling**

**On Demand Loading**

**Dynamic Import**



**React Suspense** allows you to postpone rendering until the data is available, enhancing user experience by offering fallback content or loading indications. We use it while lazy loading.

{  
 path: "grocery",  
 element: <Suspense fallback={<h1>loading...</h1>}><Grocery/></Suspense>,   
},

**Lecture 10**

**Tailwind CSS**

**Lecture 11**

**High Order Components:**

High order component is a function that takes the component and returns a component.

//Higher Order Component  
  
export const withHighRating = (restaurantCard) => {  
 return((props) => {  
 return(  
 <div>  
 <label className="res\_high\_rated">High Rated</label>  
 <RestaurantCard {...props} /> {/\*//It will pass all the props I received.\*/}  
 </div>  
 );  
 })  
}

const RestaurantRatingCards = withHighRating(RestaurantCard);

{restaurant.info.avgRating >= 4.4 ? <RestaurantRatingCards resData={restaurant} /> : <RestaurantCard resData={restaurant} />}

**Controlled and Uncontrolled Components:**

* The component which is controlled by parent component is known as controlled component.
* The component which is controlled itself is known as controlled component.

**Lifting State Up**

This concept is to control the Childrens by parent.

**The problem of Props Drilling?**

Sending data to children then other children then other and so on. This is props drilling. We can fix this issue by using React context API.

**Context API**

Used as a global scope. Use to get data from any component to any other component.

