React Documentation

Day 1: 3/15/2024

# Introduction

KG-Coding: [Link](https://www.youtube.com/watch?v=eILUmCJhl64)

## What is React?

React is a JavaScript library for building web interfaces, making it easy to create interactive elements on websites. It breaks UI into reusable components, simplifying management and updates. Using a virtual DOM, React efficiently updates only the necessary parts of the interface, enhancing performance. In essence, React streamlines web app development by offering a structured approach to managing UI components.

## DOM(Document Object Model):

The DOM is like a blueprint of a web page, representing its elements and structure as a tree of objects. Developers use it to programmatically access, manipulate, and update the content, attributes, and layout of web pages using languages like JavaScript. Essentially, it enables dynamic interaction with web pages, allowing developers to create responsive and interactive user experiences.

## Working of DOM:

1. Browser takes HTML and create DOM.
2. JS helps us modify DOM based on user actions or events.
3. In big applications, Working with DOM becomes complicated.

## Problems with JavaScript:

1. React has a simpler mental model (encapsulation).
2. JS is cumbersome (time taking).
3. JS is Error-prone.
4. JS is Hard to maintain.

## Working of React:

1. No need to worry about querying and updating DOM.
2. React creates a webpage with small and reusable components.
3. React will take care of creating and updating DOM elements.
4. It saves a lot of time.

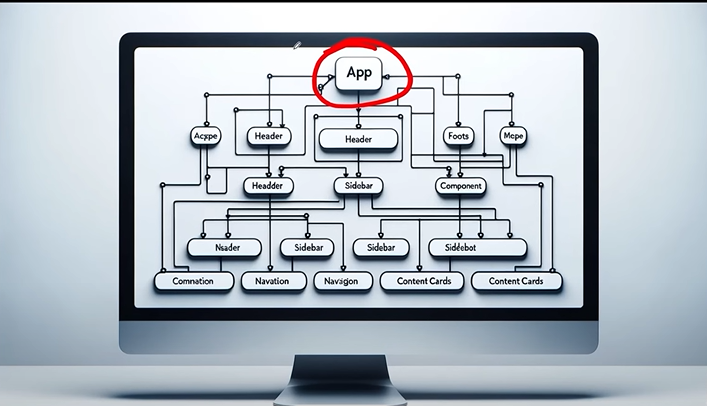
## JS V/S React

1. **Modular Components:** React encourages breaking down UIs into reusable components, making code organization and maintenance easier.
2. **Efficient Updates:** React virtual DOM efficiently updates only the necessary parts of the UI, boosting performance by minimizing unnecessary re-renders.
3. **Readable JSX Syntax:** JSX (JavaScript XML) allows developers to write UI components in a more intuitive HTML-like syntax within JavaScript, improving code readability.
4. **Built-in State Management:** React provides built-in mechanisms for managing component state, simplifying the creation of dynamic and interactive user interfaces.
5. **Large Ecosystem:** React boasts a vast ecosystem with numerous libraries, tools, and resources, enabling developers to find solutions for various web development challenges.
6. **Performance Optimization:** React offers features like memorization and lazy loading to optimize rendering performance and reduce load times for web applications.

## Introduction to Components:

Components helps us write reusable, modular, and better organized.





React application is a tree of components with App Component as the root bringing everything together.

Day 2: 3/16/2024

# Create a React App

* Setup IDE (Integrated Development Environment) E.g- Visual Studio Code, Intellij etc.
* Install Latest NodeJS- For back end.
* Installation & Setup

## Create a React App

1. Official Tool is CRA (Create React App)
2. Vite is a modern tool to create React Project
3. Vite produces Quick and Small Bundle Size
4. Vite: Use npm run dev to launch dev Server.
5. Use npm start for CRA.

->npm create vite@latest

-> npm install

-> npm run dev

create a new react app suing npx create-react-app your\_app\_name(traditional way)

Why to use vite:

1. **Fast Development Server**: Vite offers a fast development server with hot module replacement (HMR) support.
2. **Lightning-Fast Builds**: Vite utilizes modern bundling techniques to achieve incredibly fast build times.
3. **Optimized Production Build**: Vite optimizes production builds for smaller bundle sizes and faster load times.
4. **Plugin System**: Vite provides a flexible plugin system for extending functionality according to project needs.
5. **Zero Configuration**: Vite requires minimal configuration, allowing developers to focus on writing code.

## Project Structure:

1. node\_modules/ has all the installed node packages
2. public/ directory: Contains Static files that don’t change.
3. Src/Directory: Main folder for thr React Code.
   1. Components/: Reusable parts of UI, like buttons or headers.
   2. Assets: Images, fonts and other static files
   3. Styles/: CSS or style sheets.
4. Package.json contains information about this project like name, versions, dependencies on other react packages.
5. Vite.config.js contains vite config.

[**Note:** If some time your project does not run delete the node\_modules and package.json and run npm install]

# Creating React Components:

## File Extensions

1. .JS

->Stands for Javascript

-> Contains regular JS code

-> Used for general logic and components.

1. .JSX

-> Stands for JavaScript XML

-> Combines JS with HTML like tags

-> Makes it easier to design the UI components

## Class V/s Function Components

1. Class Componensts:

* Stateful: Can manage state.
* Lifecycle: Access to life cycle methods.
* Verbose: More boilerplate code.
* Not preferred anymore.

1. Functional Components

* Intially stateless.
* Can use Hooks for state and effects.
* Simpler and more concise
* More popular.

## What is JSX

1. Definition: JSX determine show the UI will look whenever the component is used.
2. Not HTML: Though is resembles HTML, you’re actually writing JSX, which stands for JavaScript XML.
3. Conversion: JSX gets Converted to regular JavaScript.
4. Babeljs.io/repl is a tool that allows you to see how JSX is transformed into JavaScript.

Day 2: 3/18/2024

## Exporting components:

1. Enables the use of the components in other parts.
2. Default Export: Allows exporting a single component as the default from module.

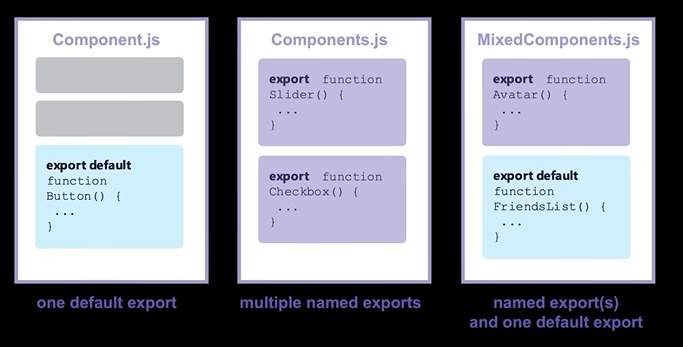
Eg: export default functionName;

1. Named Export: Allows exporting multiple items form a module.

Eg: export functionName;

While importing: import {functionName} from ……..Path….

1. Importing: To use an exported component, you need to import it in the destination file using import syntax.



## Other Important Points

1. Naming: Must be capitalizes: lowercase is for HTML.
2. HTML: In React you can embed HTML-like syntax using JSX.
3. CSSS; CSS can be directly imported into component files, allowing for the modular and component-specific styling.

## Dynamic Components

1. Dynamic Content: JSX allows the creation of dynamic and interactive UI components.
2. JS Expressions: Using {}, we can embed any JS expression directly within JSX. This includes variables, function calls, and many more.

Example:  
function Hello(){

  var myName="Sadik";

  return <h1>Hello Listen to me. I'm {myName}</h1>

}

## Reusable Components

1. Modularity: Components are modular, allowing for easy reuse across different parts of an application.
2. Consistency: Reusing components ensures UI consistency and reduces the chances of discrepancies.
3. Efficiency: Reduces development time and effort by avoiding duplication of code.
4. Maintainability: Changes made to a reused component reflect everywhere it’s used, simplifying updates and bug fixes.

[Note : Inline CSS definition in JSX: <h1 style={{'background-

color':'#776691'}}>Hello Sadik</h1>

# 17. Including Bootstrap

* Responsive: Mobile-fist design for all device sizes.
* Components: Pre-styled elements like buttons and navbars.
* Customizable: Modify default styles as needed.
* Cross-Browser: Consistent Look across browsers.
* Open-Source: Free with community support.
  1. Install:

npm i [bootstrap@5.3.2](mailto:bootstrap@5.3.2)

**npm = node package manager**

**i = install**

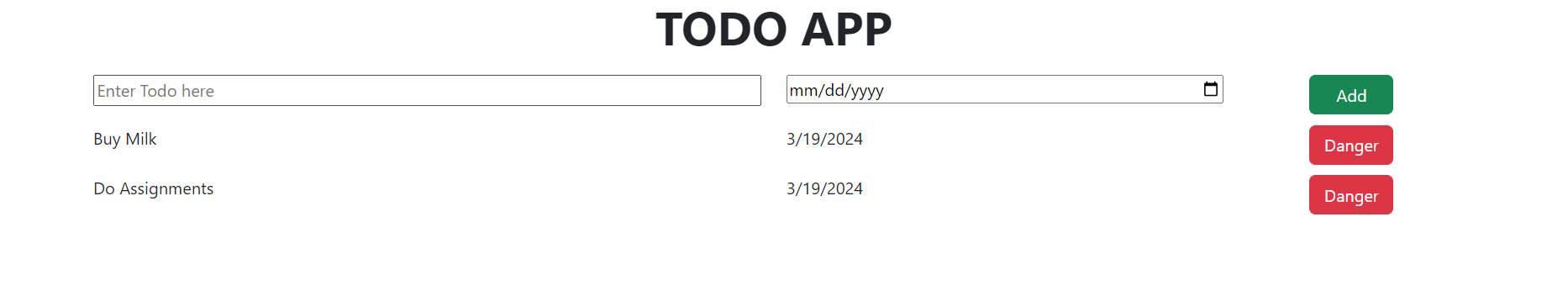
* 1. Import

import “bootstrap/dist/css/bootstrap.min.css”;

* Initiated Project(Todo App)

Day 2: 3/18/2024

* Completed the Static Todo App UI using the components and user defined CSS and bootstrap.



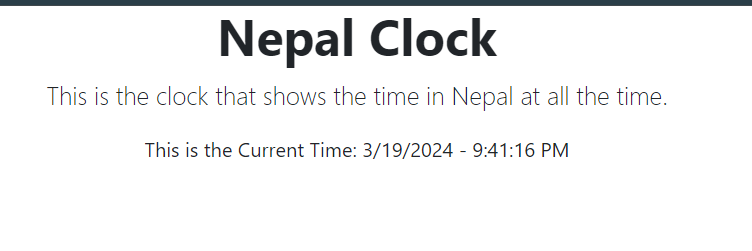
[Note: Other way to define function in .jsx file

let ClockHeading=()=>{

}

export default ClockHeading

* Nepal-Clock(Project)



# 18. Fragments

## What?

Allows grouping of multiple elements without extra DOM nodes.

## Why?

* Return multiple elements without a wrapping parent.
* Cleaner DOM and consistent styling.

## How? Two Syntaxes:

1. <React.Fragment>…</React.Fragment>

2. Short: <>…. </>

Day 4: 3/20/2024

# 19. Map Method

1. Purpose: Render lists from array data.
2. JSX Elements: Transform array items into JSX.
3. Inline Rendering: Directly inside JSX

{items.map(item=><li key={item.id}>{item.name}</li>)}

1. Key Prop. Assign unique key for optimized re-renders.

<div key={item.id}>{item.name}</div>

Eg:

function App() {

  let foodItems=['Daal','Bhat','Tarkari','Power','24 hour'];

  return (

    <>

      <h1>Healthy Food</h1>

      <ul class="list-group">

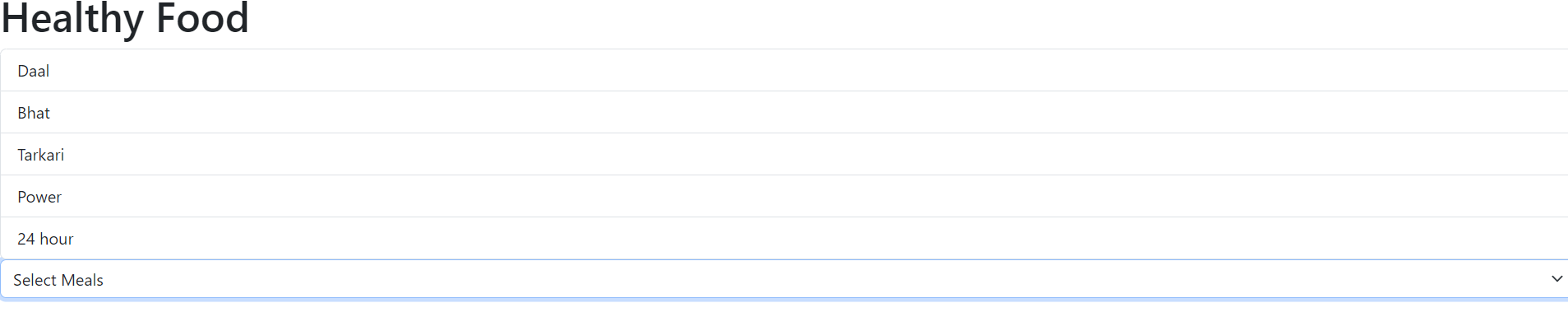
        {foodItems.map(item=><li class="list-group-item">{item}</li>)}

      </ul>

    </>

  );

}



# 20. Conditional Rendering

1. Conditional Rendering
   * Displaying content based on certain conditions.
   * Allows for dynamic user interfaces.
2. Methods
   * If-else statements: Choose between two blocks of content.
   * Ternary Operators: Quick way to choose between two options.
   * Logical operators: Useful for rendering content when a condition is true.
3. Benefits
   * Enhances user experience.
   * Reduces unnecessary rendering.
   * Makes apps more interactive and responsive.

Note:

**Here are the six falsy values in JavaScript:**

* False: The Boolean value false.
* 0: The number zero.
* "" or '' or ``: An empty string.
* Null: The null keyword, representing the absence of any object value.
* Undefined: The undefined keyword, representing an uninitialized value.
* NaN: Stands for "Not a Number".

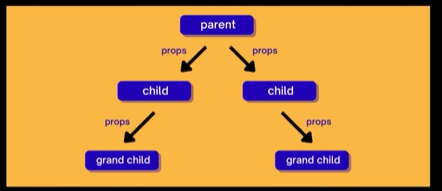
Day 5: 3/21/2024

# 21. Passing Data via Props

1. Props in React
   * Short for properties.
   * Mechanism for passing data.
   * Read-only by default
2. Usage:
   * Pass data from parent to child component.
   * Makes components reusable.
   * Defined as attribute in JSX.
3. Key points
   * Data flows one-way (downwards).
   * Props are immutable.
   * Used for communication between components.

Eg:

*<Header title = “My App”/>*



Tried my own props too.

See the work on: E:\React\21 props\learning-props.

Note to send the multiple props to the components:

<AppList appl={Apps} cardStyle={cardStyle}></AppList>

And to receive this the function definition should be like

const AppList = ({ appl, cardStyle }) => { }

# 22. CSS Modules

1. **Localized** class names to avoid global conflicts.
2. **Styles** are scoped to **individual components**.
3. Helps in creating **component-specific styles**.
4. Automatically generates **unique class names**.
5. Promotes **modular** and **maintainable** CSS.
6. Can use **alongside** global **CSS** when needed.

Day 6: 3/25/2024

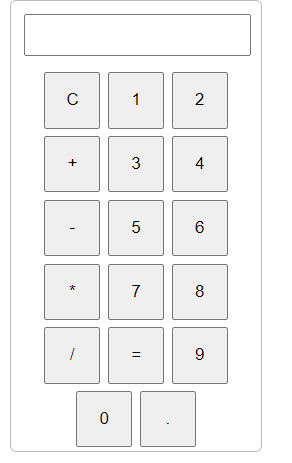
# 23. Project Calculator

Made a simple UI of the calculator using components and CSS modules.

The further calculation part will be done in the upcoming Project using the event handling

File path=> ‘E:\React\Projects\4-calculator-version-1\calculator’

This is how it looks



Day 7: 3/26/2024

# 24. Passing Children

1. Children is a special prop for passing elements into components.
2. Used for flexible and reusable component designs.
3. Common in layout or container components.
4. Accessed with props.children.
5. Can be any content: strings, number, JSX or components.
6. Enhances component composability and reusability.

const Container=({children})=>{

  return <div className={styles.container}>{children}</div>

}

export default Container;

OR {both are same thing}

const Container=(props)=>{

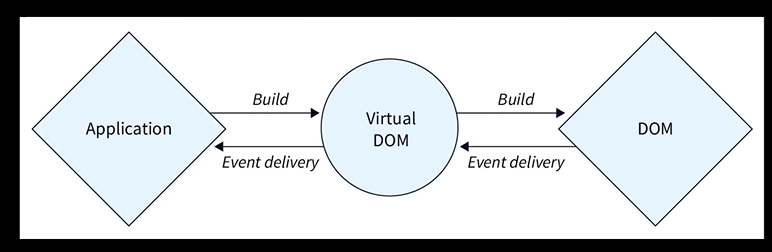
  return <div className={styles.container}>{props.children}</div>

}

export default Container;

# 25. Handling Events

1. React events use camelCase e.g. onClick.
2. Uses synthetic events, not direct browser events.
3. Event handlers can be functions arrow functions.
4. Use onChange for controlled form inputs.
5. Avoid inline arrow functions in JSX for performance.



 const handleOnChange=(event)=>{

    console.log(event.target.value)

  }

  return (

    <input

      type="text"

      className={`${styles.input} form-control`}

      placeholder="Enter Food Item Here"

      onChange={handleOnChange}

    ></input>

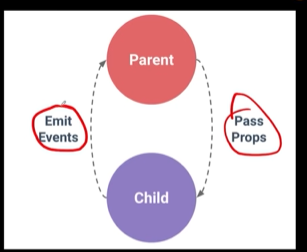
  );

Here I used the onChange event to check want inpt is being given in the <input> tag and print the input in the console and see the changes being made on the input field.

# 26. Passing Functions via Props

1. Pass dynamic behavior between components.
2. Enables upward communication from child to parent.
3. Commonly used for event handling.
4. Parent defines a function, child invokes it.
5. Enhances component interactivity.
6. Example:

<Button onClick={handleClick} />



Parent – Child communication handled by **Passing Props**

If child want to communicate with parent it **Emit Events**

# 27. Managing State

1. State represents data that changes over time.
2. State is local and private to the component.
3. **State changes** cause the component to **re-render**.
4. For **functional components**, use the **useState** hook.
5. React functions that start with word **use** are called **hooks**.
6. **Hooks** should **only** be used **inside the components**.
7. Parent components can pass state down to children via props.
8. Lifting state up: share state between components by moving it to their closest common ancestor.

useState example:

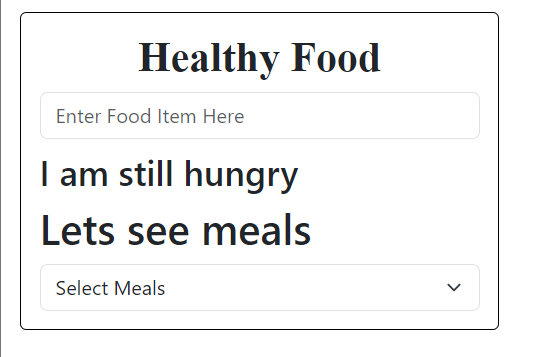
let textStateArr = useState("Food Item Entered by user");

  let textToShow = textStateArr[0];

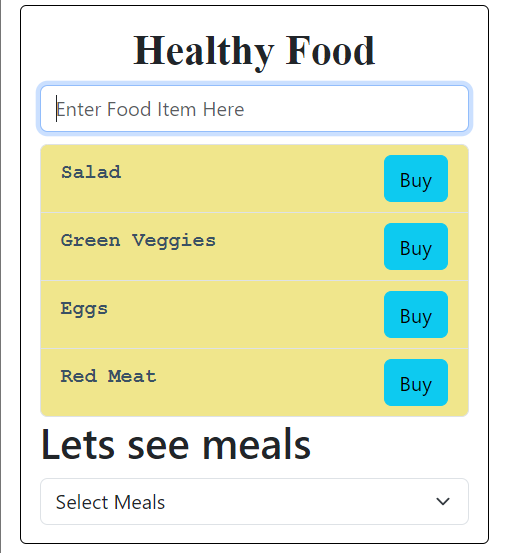
  let setTextState = textStateArr[1];

OR

let [textToShow, setTextState] = useState("Food Item Entered by user");



Whenever the input field is entered with the entered key that will get automatically updated in the list down below.



# 28. State vs. Props

1. State:
   * Local and mutable data within a component.
   * Initialized within the component.
   * Can change over time.
   * Causes re-render when updated.
   * Managed using useState in functional component
2. Props:
   * Passed into a component from its parent.
   * Read-only (immutable) within the receiving component.
   * Allow parent-to-child component communication.
   * Changes in props can also cause a re-render.