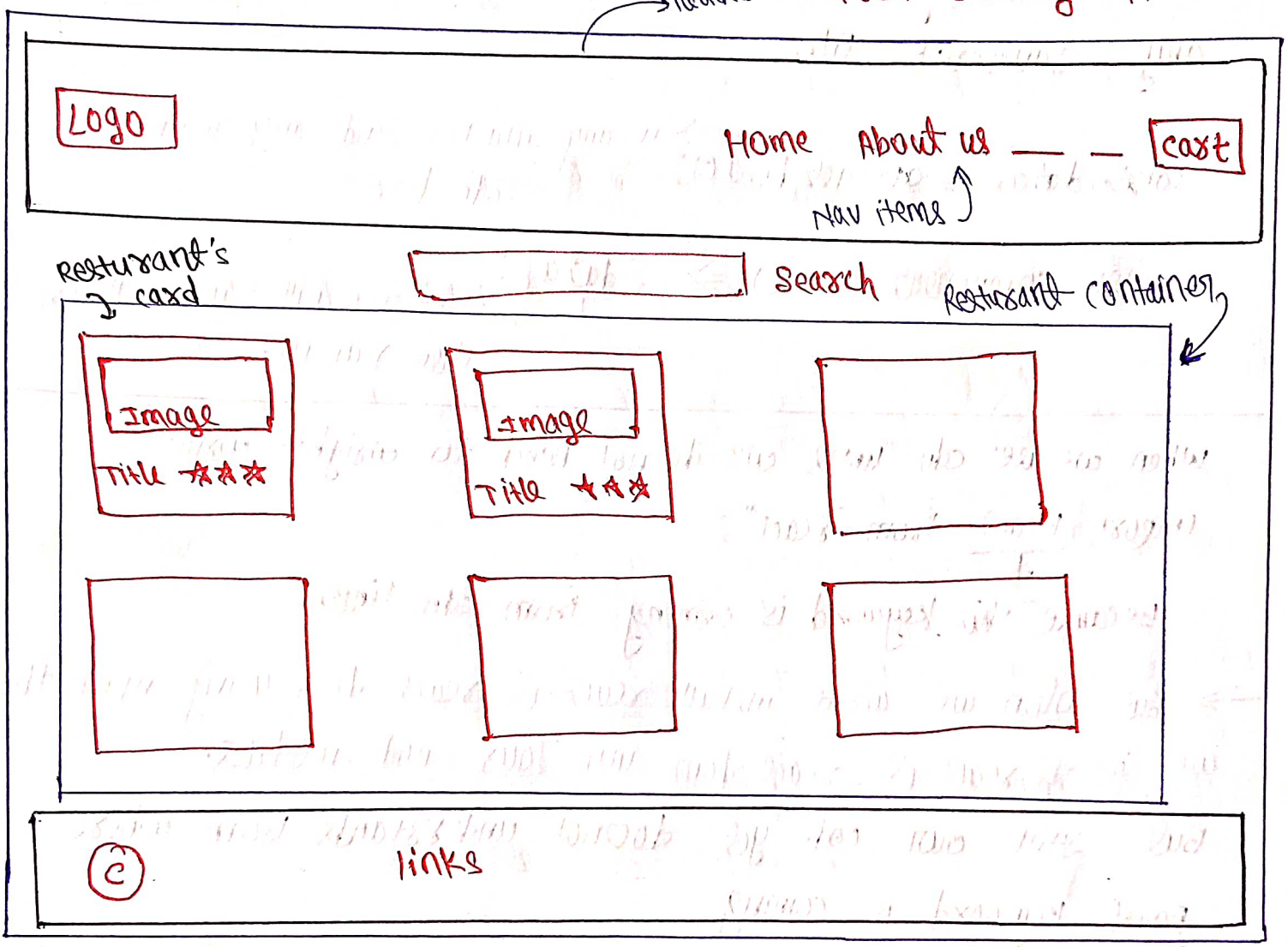
12/02/24 Lecture - 04 Talk is cheap Show me the code

Step 1: Before building any app do planning.
↳ what you are going to build (How it will look)
don't blindly Start coding !!!!

Food ordering app.



Step 2: Think about your components.
↳ ① Header
↳ logo, Nav items
→ ② Body
↳ Search component, Resturant container, Resturant card
→ ③ Footer
↳ copyright, Links, Address, contact.

→ for inline css.

```
const styleproperty = {
        background : "Black"
    }

const Header = () => {
    return ( <div style = {styleproperty} Hie </div>
    );
};
```

OR

```
const Header = () => {
    return ( <div style = {{background : "black"}}>
        Hie </div>
    );
};
```

start & end of the object

indicating there is ~~an object~~ a piece of JS inside it

→ Cross Scripting Attack ?

→ ~~what ever~~ text ~~is code inside by component return () why It is shown on Screen ?~~

Que what is JSON ?

Ans JSON Stands for Java Script object notation, it is basically a popular text-based data format, used for Storing & transmitting Structured data. It offers a human readable as well as machine readable way of representing objects, arrays, and other data structure.

14/02/24

* Props {Shortform for properties}

→ Props are just normal attributes to a function.

Passing a prop to a component = Passing attributes to a func.

```
<Rest-card resName="Alpha-Biryani" rating="5*" />
```

Sending Props to Rest-card component

→ These props will be received as a JS object at Rest-card comp.

→ Imp Keyword

```
const Rest-card = (props) => {
    return (<h1> {props.resName} </h1>);
};
```

Inside {} we can write any JS code, so we are extracting resName from our prop which is a JS object.

→ we can also do destructuring of props inside React functional comp.

```
const Rest-card = ({resName, rating}) => { return (...); };
```

Same ↓

Plain JS object destructuring.

```
const Rest-card = () => {
    const {resName, rating} = props;
    return (
        <h1> resName </h1>
        <h2> rating </h2> );
};
```

we can also do destructuring like this.

directly use names of prop after doing destructuring.

* <h1> rating </h1>

★ ~~conflict~~ config driven UI :- All the UI is configured (API data) driven by api)

configuration driven UI

→ UI is getting changed as per the configuration (ie, data from.

→ on the website of Swiggy, for delhi there may be different offers as compared to mumbai. So the offers carousel will be different.

→ we will create our UI only once and then as per data (configuration) we are getting from API that offers carousel will be changed.

→ UI layer + Data layer = Good frontend application.

→ we can also pass ~~object~~ PROPS as

```
const Rest-card = (PROPS) => {
    const {RestData} = PROPS;
    return ( <h1>{RestData.data.name}</h1> );
};
```

This name should be same as it was sent through PROPS.

★ if you are giving name to your prop like this then you also need to receive it with that same name

```
const App = () => {
    return (<Rest-card RestData={Rest-obj} />)
};
```

2nd component

object sending as prop

★ For String concatination use {}

{"Hello" + name} This is a variable

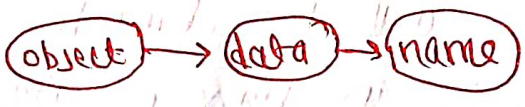
because we are writing JS inside

→ Join method in array JS

```
arr = ["Nihal", "Khushi", "Iksha"];
console.log(arr) ⟶ o/p Nihal Khushi Iksha
arr.join(", ") ⟶ o/p Nihal, Khushi, Iksha
```

→ optional chaining : chaining basically means when we are taking out data from a chain of objects

Example: object.data.name;



chaining

Now Suppose if we don't have name inside data, then through normal chaining we will get error. But with optional chaining we will not get any error

object.data?.name → optional chaining

undefined.

→ It will only access name, if it only exist if it is not not null or

* Higher order functions: A function which takes another function as an argument or returns any other function is called Higher order function

for example:- map(), filter(), reduce(),

→ let's understand map function

1.) map function takes a collection of things (array, ~~object etc~~) as an input.

2.) It applies a function to each of the items of collection

3.) returns new updated collection

```
const arr = [1,2,3,4];

console.log(arr.map((item) => (item*2)));
```

Arrow function

output: 2, 4, 6, 8

OR

```
const double = func(item) { return(item*2)};

console.log(arr.map(double));
```

output: 2,4,6,8.

↓ Performing a specific function

Note: Always remember whenever you are using map function in react, always ~~return~~ give a unique key to it.

to write JS ↓ {}

```
object_name.map((item) => (
    <Restaurant card restData={item} />
    key={item.data.id} />
));
```

* item → variable/ expression always use { }

prop.

arrow because item is JS object for writing JS use { }

unique key

Because we are writing or accessing (chaining) JS object. & to write JS use { }

Note: Syntax might be confusing we are using {} to write object inside another {} (which we used to wrap map function into it) But this is the correct Syntax

It is Similar to `Style={{background:"red"}}`

- To write JS object into it
- A Java Script object in itself

Ques Why we need to pass key into map?

Ans If we will not pass unique key into map function, then react will not be able to uniquely identify the new item added into array on which we are using map().

Let's Say we have an array of restaurants, and now we want to add new rest in front.

Rest = [AP , BP , CP , CD]

But here's the problem begins: React can't identify that new restaurant to be added what it will do, it will re-render all the restaurants.

But if it have a unique key

AP	BP	CP	CD	New to add CR
↓	↓	↓	↓	↓
id1	id2	id3	id4	id5

Now react can identify all the existing restaurants with their key, so that it will now identify which is the new element (restaurant) came which was previously not present in the list/array.

So now instead to re-rending all the elements again, it will only render the new restaurant (item) to its correct place.

→ So if we are using key re-rending of all elements/items/restaurants will not take place is our app will run fast.

Que → But what if I don't have any unique key (id)?

In such cases you can use the index.

```
obj-name.map((item, index) => ( key={index} <comp /> ));
```

index → keyword

Note :- using index, or key in map, is not recommended by react itself.

→ If you will not give key then react itself will use index as a key

→ you can even use `key={Math.random()}` But it is also not recommended

Que Why index or math.random() is not recommended ?

→ Index :- when ever you insert, delete or array gets reordered, the order in which the elements will render will change.

due to this it will re-render all the elements. (or simply index of each ele. will be changed)

→ Math.random() :- Each time when you map function will run, it will generate random numbers or new number /id every time, due to this react can not identify the array elements uniquely & it will again do re-rending.

15/02/24

★ Filter :- used to filter out input.

→ It is basically used to filter out our input array

ip array = [1, 2, 3, 4] —(filter out odd elem)→ [2, 4]

→ Syntax is completely same as map.

① It takes a collection of things as an input (array, ~~objects etc~~)
② It applies a specific function to all elements of collection
③ returns filtered array.

→ const arr = [5, 1, 2, 3, 6]

→ const output = arr.filter (findEven);
keyword function

const findEven = (element) => {
return (element % 2 == 0);
};

Both will give same output

OR

cost output = arr.filter (element => {
return (element % 2 == 0);
};

★ Reduce : It does not reduce anything.

→ const arr = [5, 1, 3, 2, 6];

function findSum (arr){
const Sum = 0;
for (let i = 0; i < arr.length; i++) {
Sum = Sum + arr[i];
} acc. curr.
return Sum;
}.

1) Where to use reduce ?
ans : When you want to iterate over all the values of any collection & wants to calculate any one single value
for example : Sum.

It is accumulator which is used to store result

It represents the current value of the array.

const output = arr.reduce (function (acc, curr) {
acc = acc + curr ; // curr == arr[i]
return acc;] → from here we will get our final answer.
}, 0);
↑
Initial value of acc

1) Reduce takes a collection of things as an input
2) Performs a specific function to all the elements of the collection
3) Returns **a specific value**

↳ It takes 2 arguments :-
- 1) A function which we need to perform
- 2) An initial value of acc (accumulator)

In func. it takes 2 arguments
1) curr which represents curr. element
2) acc which gives a specific output

→ curr (current) is like arr[i], which is inside a loop.

for arr = [1, 2, 3, 4]

curr = 1
curr = 2
curr = 3
curr = 4

⟷ same thing

```
for (int i=0; i<4; i++)
{ arr[i]; }
```

Q) write a reduce function to get max element.

```
const output = arr.reduce (function (acc, curr) {
    if (acc < curr)
    { acc = curr; }
    return acc;
}, 0);
```

Note: ① When ever we need to iterate through the whole array & find out a one single value as an answer, we should use reduce.

② When we need to filter out our array on the basis of any logic use filter.

③ When we need to traverse whole array & want to change or want to return a value for each element of the array use map.

```
const users = [
  { fName: "akshay", lName: "saini", age: 26 },
  { fName: "Nihal", lName: "Singh", age: 21 },
  { fName: 'x', lName: "y", age: b }
];
```

Ques: Join 1st & last name output should be like akshay saini

Ans ① we need to iterate all over the array & find a particular value for all elements of array so use map.

```
const ans = users.map ( (obj) => {
        return( fName + " " + lName);
    };
```

Ques 2: Return an object like { 26: 1, 21: 1, b: 1 } (age : count)

Ans ① we need a particular output i.e, only one object so use reduce.

```
const ans = users. reduce ( function (acc, curr) {
    if (acc [curr.age])
    {
        acc [curr.age] = ++ acc[curr.age];
    }
    else {
        acc [curr.age] = 1;
    }
    return acc;
}, { } );
```

↑ Initial value of output → an empty object

Case 3 find 1st name of all in an array whose age > 30

And ① want a specific output array, a single array output so we use reduce.

```
const ans = users.reduce(function(acc, curr) {
        if (curr.age > 30)
        { acc.push(curr.fname);
        }
        return acc;
}, [ ]);
```

→ initial value of output array.

② Second method: ① we need to filter out our array of objects so we can use ~~map~~ filter.

② But we only want 1st name, with filter we can not edit our input But with map we can do so, then we map also.

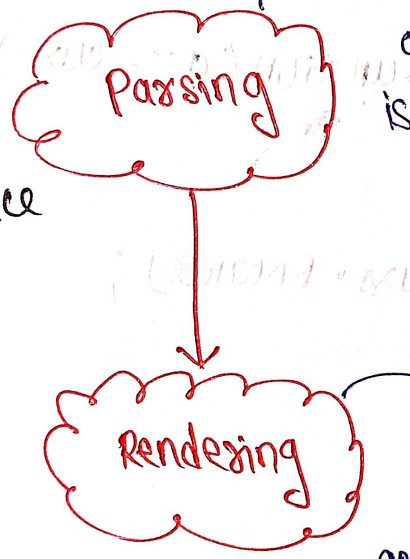
```
const ans = users.filter((obj) => {
        return (obj.age > 30).map((obj) => {
                return (fname);
        });
});
```

This will filter out our users array with objects age > 30

→ This will work on that filtered array & return firstname of all of them.

10/02/24

- ~~Rendering~~



Parsing takes place first and then rendering.

Take HTML code, and convert it into DOM (document object model) where each tag is like node.

Now after having dom structure it takes each element apply css ~~and other~~ styling like exact size color, etc and then translates into what users see on screen

Who downloads the node modules at server because we don't upload it?

There are several ways `node_modules` can be downloaded on a server, and who does it depends on your specific setup. Here are the most common scenarios:

1. **Automated Deployments:**

- **CI/CD Pipeline:** When you push code changes, a CI/CD pipeline (Continuous Integration/Continuous Delivery) automatically builds and deploys your application. This pipeline often includes a step to install dependencies on the server using commands like `npm install` or `yarn install`. These commands fetch dependencies from the public registry and store them in `node_modules` on the server.
- **Deployment Tools:** Specific deployment tools like Capistrano or Deploy.js might have built-in functionality to handle dependency installation during the deployment process.

2. **Containerized Environments:**

- **Docker Image:** If you use containerization with tools like Docker, your containers can include specific tools and configurations to install dependencies at startup. This could involve embedding a package manager and its configuration within the container image itself.

3. Manual Installation:

- **Rarely:** In less common scenarios, you might manually install dependencies on the server using the same `npm install` or `yarn install` commands you use locally. This approach is becoming less frequent due to the rise of automation.

Who Does It?

In most cases, **nobody needs to explicitly download or install dependencies manually** since automated processes handle it. The specific actor depends on your setup:

MODULES

Modules are designed to be reusable, which means that they can be imported and used in multiple places in an application. In contrast, plain old JavaScript code is not organized into reusable units, which makes it more difficult to reuse and maintain.

In modules we have the feature of import and export, which was not available into old js.

React Fragment

<></> or <React.Fragment> <React.Fragment /> is a feature in React that allows you to return multiple elements from a React component by allowing you to group a list of children without adding extra nodes to the DOM.

`<></>` is the shorthand tag for React.Fragment. The only difference between them is that the shorthand version does not support the key attribute.

Reconciliation

Reconciliation is the process through which React updates the Browser DOM and makes React work faster. React use a `diffing algorithm` so that component updates are predictable and faster. React would first calculate the difference between the real DOM and the copy of DOM (Virtual DOM) when there's an update of components.
React stores a copy of Browser DOM which is called `Virtual DOM`. When we make changes or add data, React creates a new Virtual DOM and compares it with the previous one. Comparison is done by `Diffing Algorithm`.

React compares the Virtual DOM with Real DOM. It finds out the changed nodes and updates only the changed nodes in Real DOM leaving the rest nodes as it is. This process is called Reconciliation.

(or)

Reconciliation is the process by which React updates the UI to reflect changes in the component state. The reconciliation algorithm is the set of rules that React uses to determine how to update the UI in the most efficient way possible. React uses a virtual DOM (Document Object Model) to update the UI.

React Fiber

Imagine you have a big, heavy painting you want to hang on the wall. Traditionally, you'd need two strong people to lift it and put it up at once. But, what if you could break the painting into smaller, lighter pieces, hang them one by one, and then magically reassemble them on the wall? That's kind of what `React Fiber` does!

React Fiber is a technology within React that helps render your web pages **smoothly and efficiently**, especially when dealing with complex updates or large amounts of data. Here's how it works in simple terms:

1. **Breaking down:** When you update your React components, Fiber first **breaks down the changes into smaller, more manageable pieces**. It analyzes what actually needs to be updated and prioritizes them.

2. **Rendering in bits:** Instead of re-rendering the entire page each time, Fiber can **update just the specific parts that have changed**. Think of it like painting small sections of the big picture instead of the whole thing at once.

3. **Prioritizing smoothly:** Fiber has a **scheduling system** that prioritizes urgent updates first, ensuring that essential changes like user interactions happen smoothly, even when other parts are still being updated.

4. **Reconstructing the view:** Once all the pieces are updated, Fiber **magically recombines them** into the final, updated UI. Just like the individual sections forming the complete painting.

Benefits of React Fiber:

- **Faster and smoother updates:** By prioritizing and breaking down updates, Fiber makes your website feel more responsive and less laggy.
- **Better handling of complex UIs:** It can efficiently handle large amounts of data or intricate components, providing a smoother experience.
- **More efficient use of resources:** By rendering only what's necessary, Fiber reduces the strain on your computer and browser.

Remember:

- `React Fiber` is an internal technology that helps React work efficiently, you don't directly interact with it as a developer.
- It's like the behind-the-scenes magic that ensures your website runs smoothly even with complex updates!

I hope this simple explanation helps you understand `React Fiber` in a clearer way!

The relationship between React Fiber and the reconciler in React!

Fiber is not just a reconciler, but it utilizes a new and improved reconciler.

Think of the **reconciler** as the core algorithm responsible for comparing the virtual DOM (lightweight in-memory UI representation) with the real DOM (actual elements on the screen) and updating the latter efficiently. Before React Fiber, the traditional reconciler used a "stack-based" approach, which could struggle with complex UIs or large amounts of data.

React Fiber introduced a significant upgrade:

- It relies on a **fiber data structure** to represent each element in the virtual DOM. These fibers form a linked list, enabling a more **flexible and asynchronous** reconciler.

- Instead of processing changes sequentially, Fiber can **prioritize urgent updates** (like user interactions) and pause or interrupt less critical ones, ensuring a smoother experience.

So, while Fiber isn't solely a reconciler, it **integrates an enhanced reconciler** that brings about several advantages.

- Fiber leverages a new, more flexible reconciler compared to the traditional approach.
- This enhanced reconciler leads to significant performance improvements and smoother UI experiences in React.
- You don't directly interact with the reconciler or Fiber; they work behind the scenes to optimize your React app.

I hope this clarifies the connection between Fiber and the reconciler in React!

Props

props stands for properties. Props are arguments passed into React components. props are used in React to pass data from one component to another (from a parent component to a child component(s)). They are useful when you want the flow of data in your app to be dynamic.

DOM

- **DOM is a way for the browser to organize and understand the structure of a web page.**
- It represents the web page as a tree of elements, where each element is like a building block.
- Each element has properties, like its type, text content, and position on the page.

Technical definition:

- The DOM is a programming interface that allows web developers to interact with and modify the content and structure of a web page.
- It's an API (Application Programming Interface) designed for JavaScript, but other languages can also access it.
- It's a tree-based structure where each node represents an element, and the nodes are connected in a hierarchical way, reflecting the layout of the page.

Why is the DOM important?

- It allows developers to dynamically change the content and appearance of a web page without refreshing the entire page.
- It's essential for making web pages interactive, responsive, and engaging for users.

Remember:

- The DOM is not the actual web page you see, but rather a representation of it in the browser's memory.

Real DOM vs Virtual DOM

The **real DOM (Document Object Model)** and **virtual DOM** are two key concepts in React development, each playing a crucial role in displaying your website's UI. Here's a breakdown of their differences and how they work together:

Real DOM:

The DOM represents the web page often called a document with a logical tree and each branch of the tree ends in a node and each node contains object, programmers can modify the content of the document using a scripting language like javascript and the changes and updates to the dom are fast because of its tree-like structure but after changes, the updated element and its children have to be re-rendered to update the application UI so the re-rendering of the UI which make the dom slow all the UI components you need to be rendered for every dom update so real dom would render the entire list and not only those item that receives the update .

Virtual DOM:

The Virtual DOM is a light-weight abstraction of the DOM. You can think of it as a copy of the DOM, that can be updated without affecting the actual DOM. It has all the same properties as the real DOM object, but doesn't have the ability to write to the screen like the real DOM.

- Virtual DOM is just like a blueprint of a machine, can do the changes in the blueprint but those changes will not directly apply to the machine.
- Reconciliation is a process to compare and keep in sync the two files (Real and Virtual DOM). Diffing algorithm is a technique of reconciliation which is used by React.