Notes Campus Compass

Chapter 1

This Chapter Contains Building Blocks of Web Application , intro to react and next.js .

- react - is a javascript library for building interactive UI interfaces.
- Next.js -is a React framework that gives you building blocks to create web applications,it is more user friendly than react.

Building Blocks of Web Application

- User Interface - how users will consume and interact with your application.
- Routing - how users navigate between different parts of your application.
- Data Fetching - where your data lives and how to get it.
- Rendering - when and where you render static or dynamic content
- Integrations - what third-party services you use (for CMS, auth, payments, etc.) and how you connect to them.
- Infrastructure - where you deploy, store, and run your application code (serverless, CDN, edge, etc.).
- Performance - how to optimize your application for end-users.
- Scalability - how your application adapts as your team, data, and traffic grow.
- Developer Experience - your team's experience building and maintaining your application.

Chapter2(Rendering UI)

How browser Interpret code? It reads HTML and create Document Object model(DOM).

What is DOM? It is a tree like structure ,object reprensentation of HTML element ,It is a bridge between code and UI. You can manipulate DOM.

Chapter3(Updating UI with JavaScript)

How to add h1 tag using plain javascript. Ans->

important points

- div id="app" this gives it unique Id so it can be targeted later.
- The getElementById() method of the Document interface returns an Element object representing the element whose id property matches the specified string. Since element IDs are required to be unique if specified, they're a useful way to get access to a specific element quickly.

What is difference between HTML and DOM? In DOM you will see inserted roots (in above example h1 will be seen in DOM while not in HTML). This is because the HTML represents the initial page content, whereas the DOM represents the updated page content which was changed by the JavaScript code you wrote.

| imperative programming | while writting a code you elaborate each and every step. |
| declarative programming | it is not such elaborate,it uses builtin user interfaces|

Chapter4

In this chapter we will be using React to write our code which we have written previously using Plain JavaScript for this we have to import two libraries i.e. react and react-dom the latter library provides methods through which we can use react with DOM.

- here in this code we have imported libraries by - ReactDOM.createRoot() - a. Creates a React root in that DOM container;b.) Enables Concurrent Mode ,React can pause, interrupt, or prioritize UI updates. This new root supports Concurrent Rendering, which means: c. Returns a "root" object with a .render() method - root.render - to show your react UI inside the container.

This above script will give error because we have to follow JSX rules.

What is JSX? -JSX is a syntax extension for JavaScript that allows you to describe your UI in a familiar HTML-like syntax.

rules of JSX 1). Return a single root element . To return multiple elements from a component, wrap them with a single parent tag. 2). Close all the tags 3). camelCase all most of the things!

To convert JSX to javaScript we need Babel. to add babel

- In addition, you will need to inform Babel what code to transform by changing the script type to type=text/jsx.

Chapter5

There are three core concepts of React

- Components
- Props
- State

Components

User interfaces can be broken down into smaller building blocks called components.

In React, components are functions.

Applications usually include more content than a single component. You can nest React components inside each other like you would regular HTML elements.

NESTING COMPONENTS Applications usually include more content than a single component. You can nest React components inside each other like you would regular HTML elements.

Component trees You can keep nesting React components this way to form component trees.

Note: In React, data flows down the component tree. This is referred to as one-way data flow. State, which will be discussed in the next chapter, can be passed from parent to child components as props.

Chapter6(Displaying Data with Props)

what if you want to pass different text or you don't know the information ahead of time because you're fetching data from an external source? you can pass pieces of information as properties to React components. These are called prop.

Using PROPS passing prop

> *function Header({ title }) { console.log(title); return*
>
> ***{title}***
>
> *; } **Embedding JavaScript in JSX** Curly Braces ({}): Wrap any JS expression in {} to switch from “JSX land” into “JavaScript land” (chatgpt) Next.js React .*

Valid Expressions: You can embed:

- Dot notation: {props.title}
- Template literals: {Cool ${title}}
- Function calls: {createTitle(title)}
- Ternaries: {title ? title : 'Default Title'}

Rendering Lists with array.map() Mapping Data to Elements: Use names.map(name =>

- {name}

) inside JSX to generate list items dynamically.

Chapter7(Adding Interactivity with State)

onClick : event ,used in syntax like <button onClick{may be function name which activates while clicking}> name of button

handling event

- You can define a function to "handle" events whenever they are triggered.

State and hooks

React has a set of functions called hooks. Hooks allows us to add additional logic such as state to your components.

State: it is like any information in your UI that changes over time, usually triggered by user interaction.

example:You can use state to store and increment the number of times a user has clicked the "Like" button.

useState it is an react hook,used to manage state.

> *function HomePage() { // ... const [likes, setLikes] = React.useState(0);*

> *function handleClick() { setLikes(likes + 1); }*

> *return (*
>
> *{/* ... */}* Likes ({likes})
> *); }*

- const [likes, setLikes] = React.useState(0); uses the useState Hook to declare a state variable likes initialized to 0, returning the current state and a setter function setLikes

- he handleClick function calls setLikes(likes + 1), updating the likes state and causing React to re-render the component with the new value.
- The JSX `Likes ({likes})` attaches a camelCase onClick event handler and injects the dynamic likes value into the button label using {}
- Each click triggers React's one-way data flow—state changes via setLikes flow down into the UI, ensuring the displayed count always reflects the latest likes value.

_Chapter 8(From react to Next.js)

complete code:' '

```
<script src="https://unpkg.com/react@18/umd/react.development.js"></script>
<script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>

<script type="text/jsx">
  const app = document.getElementById("app")

  function Header({ title }) {
    return <h1>{title ? title : "Default title"}</h1>
  }

  function HomePage() {
    const names = ["Ada Lovelace", "Grace Hopper", "Margaret Hamilton"]

    const [likes, setLikes] = React.useState(0)

    function handleClick() {
      setLikes(likes + 1)
    }

    return (
      <div>
        <Header title="Develop. Preview. Ship." />
        <ul>
          {names.map((name) => (
            <li key={name}>{name}</li>
          ))}
        </ul>

        <button onClick={handleClick}>Like ({likes})</button>
      </div>
    )
  }

  const root = ReactDOM.createRoot(app);
  root.render(<HomePage />);
</script>
```

note-Next.js uses file-system routing. This means that instead of using code to define the routes of your application, you can use folders and files.

Chapter10(Server and client components)

environment : example of environment are client and server , it is a place where application code can run **network boundary** : it is like a separation between client and a server. **client**:it is basically user's device browser which is requesting server to give a code which it will represent in UI. **server**:The server refers to the computer in a data center that stores your application code, receives requests from a client, does some computation, and sends back an appropriate response.

tip:, by moving rendering and data fetching to the server, you can reduce the amount of code sent to the client, which can improve your application's performance.

network boundary

- In React, you choose where to place the network boundary in your component tree

  React divides your app into two trees: a server module graph (all Server Components) and a client module graph (all Client Components) The server renders its components into a compact binary React Server Component Payload (RSC Payload) and streams it to the browser The RSC Payload includes the HTML output of Server Components, “holes” marking where Client Components go, URLs for their JS bundles, and any props to pass along