DAY 1: INTRODUCTION TO HTML,CSS

DATE: 16-07-2024

DAY: TUESDAY

Contents Covered:

Day 1: Understanding HTML Basics

- **What I Learned**:

HTML stands for Hyper Text Markup Language.

It is the backbone of any webpage, providing structure and content.

Key components include:

Elements: The building blocks of HTML (e.g., <p>, <h1>,).

Attributes: Additional information about elements (e.g., class, id).

Common tags include:

- <html>: The root element.
- <head>: Contains metadata like title and links.
- <body>: The visible content of the webpage.
- <h1> to <h6>: Headings.
- <p>: Paragraphs.
- <a>: Hyperlinks.

Practice:

Created a basic HTML file with a title, headings, and a paragraph.

Day 2: Digging Deeper into HTML

DATE: 17-07-2024

DAY: WEDNESDAY

What I Learned:

HTML lists:

Ordered lists (<ol>) for numbered items.

Unordered lists (<ul>) for bulleted items.

List items (<li>) for each list entry.

Adding images:

<img> tag with src for the image path and alt for description.

Forms and inputs:

<form> for creating forms.

<input>, <textarea>, <button>, and <select> for user inputs.

Practice:

Built a simple webpage with a list of favorite books, an image, and a feedback form.

Day 3: Introduction to CSS

DATE: 18-07-2024

What I Learned:

CSS stands for Cascading Style Sheets.

It styles HTML elements, including layout, colors, and fonts.

Ways to use CSS:

Inline: Inside the style attribute in an HTML tag.

Internal: Inside a <style> tag in the <head>.

External: In a separate .css file linked with <link>.

Basic syntax:

Selectors (e.g., h1, .class, #id) define the elements to style.

Declaration blocks contain properties (e.g., color, font-size) and values.

Practice:

Styled headings and paragraphs with colors and font sizes.

Experimented with inline, internal, and external CSS.

Day 4: CSS Styling and Layout Basics

DATE: 19-07-2024

What I Learned:

Styling basics:

color, background-color, font-family, and font-size for text.

Box model concepts: margin, border, padding, and width.

Layouts:

display property: block, inline, flex, etc.

position property: static, relative, absolute, fixed.

CSS Grid and Flexbox for responsive layouts.

Practice:

Created a webpage with a styled header, navigation bar, and content area.

Used Flexbox to align items horizontally.

Day 5: Putting It All Together

DATE: 22-07-2024

What I Learned:

Combining HTML and CSS to build a complete webpage.

Importance of semantic HTML (e.g., <header>, <footer>, <article>) for better structure and accessibility.

Debugging CSS using browser developer tools.

Practice:

Designed a personal portfolio webpage with:

A header and navigation menu.

A section for projects.

A footer with contact details.

Day 6: Introduction to JavaScript Basics

DATE: 23-07-2024

What I Learned Today

Today was my first step into JavaScript, a language that brings life and interactivity to web pages. I started with understanding the basics of the language:

What is JavaScript?

JavaScript is a high-level, interpreted programming language.

It runs in web browsers and allows developers to create dynamic content like image sliders, form validations, and interactive menus.

Variables and Their Types

Variables are containers that store data. JavaScript allows three ways to declare variables:

var: The older way to declare variables, function-scoped.

let: Modern, block-scoped, used for variables that may change.

const: Block-scoped, used for variables that won't change.

JavaScript Data Types

Primitive Data Types: These include strings, numbers, booleans, undefined, null, and symbol.

Non-Primitive Data Types: Objects and arrays.

Basic Operators

Arithmetic Operators: +, -, *, /, % for basic math operations.

Comparison Operators: ==, ===, <, >, etc., to compare values.

Logical Operators: && (AND), || (OR), ! (NOT) for combining conditions.

New Insights

JavaScript is case-sensitive, so let Name and let name are treated as different variables.

Using === instead of == for comparison is safer as it checks both value and type.

Day 7: Functions and Control Flow

DATE: 24-07-2024

What I Learned Today

Functions are the building blocks of reusable code. Today, I focused on:

1. Functions in JavaScript

- A function is a reusable block of code that performs a specific task. Functions help to avoid redundancy and improve code maintainability.

- Functions can be declared using the `function` keyword.

- Functions can also return values.

2. Control Flow and Conditional Statements

- I studied how to make decisions in JavaScript using `if`, `else if`, and `else` statements. These allow you to execute different blocks of code based on conditions.

- The `switch` statement is another way to handle multiple conditions more efficiently**.**

3. Logical Operators

- I explored logical operators: **`&&` (AND)**, **`||` (OR)**, and **`!` (NOT)**. These are used to combine or negate conditions.

Day 8: Arrays and Loops

DATE: 25-07-2024

What I Learned Today

Arrays are essential for storing multiple values in a single variable. Today, I learned:

1. Arrays in JavaScript

- Arrays store multiple values of the same type (or mixed types) in an ordered manner.

2. Array Methods

- `push()` and `pop()` add or remove elements from the end of an array.
- `shift()` and `unshift()` add or remove elements from the beginning of an array.
- `concat()` joins two or more arrays into one.
- `slice()` extracts a portion of an array.

3. Loops

- I explored different ways to iterate over arrays:
  - For loop: Useful for iterating a fixed number of times.
  - `for...of` loop: Better for iterating over the values of an array.
  - `forEach()` method: A functional approach to iterate over an array.

Day 9: Objects and Their Properties

DATE: 26-07-2024

What I Learned Today

Objects are crucial for organizing related data. I focused on:

1. Objects in JavaScript

- An object is a collection of key-value pairs, where keys are strings, and values can be any data type.

2. Accessing and Modifying Object Properties

- Properties can be accessed using dot notation or bracket notation.

3. Nested Objects

- Objects can contain other objects as properties.

Practical Exercises and Problem-Solving

1. Task 1: Store a Book's Information

I created a book object and printed its details.

2. Task 2: Add a Method to an Object

I added a method to the book object to toggle its availability.

Day 10: JavaScript StringS

DATE: 29-07-2024

What I Learned Today

Strings are one of the most commonly used data types. I explored:

1. String Methods

- **`charAt()`:** Returns the character at a given index.
- **`toUpperCase()` and `toLowerCase()`:** Convert the case of a string.
- **`slice()`:** Extracts a section of a string.
- **`replace()`:** Replaces parts of the string.

2. Template Literals

- I used template literals to embed variables inside strings using backticks**.**

Practical Exercises and Problem-Solving

1. Task 1: Reverse a String

I used `split()`, `reverse()`, and `join()` to reverse a string.

2. Task 2: Count Vowels in a String

I wrote a function to count the vowels in a string using `match()` and regular expressions.

- String methods are powerful tools for manipulating text.

- Template literals are an excellent way to build strings with variables embedded, making the code cleaner and easier to read.

Day 11: Introduction to Object-Oriented Programming (OOP)**

DATE: 30-07-2024

What I Learned Today

Today, I took a deep dive into Object-Oriented Programming (OOP), a key paradigm used in JavaScript. OOP is about structuring code in a way that models real-world entities as objects. I covered the following concepts:

1. Classes and Objects

- A class is a blueprint for creating objects with common properties and methods.

- An object is an instance of a class.

2. Encapsulation

- Encapsulation is the concept of bundling data (properties) and methods (functions) that operate on the data into a single unit called a class.

- I also learned how to make some properties or methods private by using `#` (in newer versions of JavaScript).

3. Inheritance

- Inheritance allows a class to inherit properties and methods from another class.

- The extends keyword is used for inheritance.

Practical Exercises and Problem-Solving

1. Task 1: Create a Book Class

I created a class to model a book with properties like `title`, `author`, and `pages` and methods to display the book's details.

2. Task 2: Create a BankAccount Class with Inheritance I created a `BankAccount` class and a `SavingsAccount` subclass that inherits from it.

Day 12: Error Handling and Debugging

DATE: 31-07-2024

What I Learned Today

Today was focused on handling errors and debugging JavaScript code. I explored how to anticipate and handle issues gracefully to make code more robust.

1. Types of Errors

Syntax Errors: Mistakes in the syntax of the code (e.g., missing parentheses or semicolons).

Runtime Errors: Errors that occur while the code is running (e.g., calling a function on `undefined`).

Logical Errors: Errors where the code runs but doesn't behave as expected.

2. Error Handling with Try-Catch

- The `try` block is used to execute code that might throw an error, and the `catch` block catches the error if one occurs.

- The `finally` block always runs, regardless of whether an error occurred or not.

3. Throwing Custom Errors

- We can throw custom errors using the `throw` statement.

4. Debugging with Console Methods

- I used `console.log()` for simple debugging.

Practical Exercises and Problem-Solving

1. Task 1: Divide by Zero Error Handling

I created a function that divides two numbers and uses `try-catch` to handle divide-by-zero errors.

Day 13: Asynchronous JavaScript (Callbacks)

DATE: 01-08-2024

What I Learned Today

I dove into the asynchronous nature of JavaScript. Since many tasks (e.g., network requests) can take time, JavaScript needs to handle these operations without freezing the application.

1. Callbacks

- A callback is a function that is passed as an argument to another function and is executed when the task is complete.

- This allows JavaScript to perform non-blocking operations. While waiting for one task to complete, the program can execute other code.

2. Asynchronous Nature with `setTimeout()`

- `setTimeout()` is a built-in function that runs a piece of code after a specified delay (in milliseconds). This simulates asynchronous behavior.

Practical Exercises and Problem-Solving

1. Task 1: Delayed Message

I wrote a function that prints a message after a delay using `setTimeout()`.

2. Task 2: Using Callbacks in Data Fetching Simulation

I simulated fetching data asynchronously using a callback.

Day 14: Promises and Async/Await

DATE: 02-08-2024

DAY: Friday

What I Learned Today

I continued exploring asynchronous programming with Promises and Async/Await, two powerful techniques for handling asynchronous code.

1. Promises

- A Promise is an object representing the eventual completion (or failure) of an asynchronous operation.

- Promises can be in one of three states: pending, fulfilled, or rejected.

2. Async/Await

- `async` is used to declare a function as asynchronous, and `await` pauses the execution of the function until the Promise resolves.

Practical Exercises and Problem-Solving

1. Task 1: Simulate a Delay with Promises

I created a function that simulates a delay using a Promise.

2. Task 2: Use Async/Await with Promises

I used async/await to handle the promise result.

Day 15: JavaScript DOM Manipulation (Introduction)

DATE: 05-08-2024

DAY: Monday

What I Learned Today

Today was all about learning how to interact with the Document Object Model (DOM) in JavaScript. The DOM is a programming interface for web documents, representing the page structure as a tree of objects that can be manipulated. Here are the key topics I covered:

1. What is the DOM?

- The DOM represents an HTML document as a tree of nodes, where each element, attribute, and text is a node.

- JavaScript can be used to traverse, modify, and manipulate this tree dynamically.

2. Accessing DOM Elements

- `document.getElementById()`: Selects an element by its `id`.

- `document.getElementsByClassName()`: Selects elements by class name.

- `document.querySelector()`: Selects the first element that matches a CSS selector.

- `document.querySelectorAll()`: Selects all elements that match a CSS selector.

3. **Creating and Adding Elements**

- Creating a new element.

- Appending the element to the DOM:

Practical Exercises and Problem-Solving

1. Task 1: Update Heading Text

2. Task 2: Add a New List Item

Day 16: DOM Events and Event Listeners

DATE: 06-08-2024

DAY: Tuesday

What I Learned Today

Today, I learned how to make web pages interactive by handling events like clicks, mouse movements, and keyboard input.

1. What are Events?

- An event is an interaction by the user or browser, such as clicking a button, typing text, or loading a page.

- Examples: `click`, `mouseover`, `keydown`, `load`, etc.

2. Adding Event Listeners

- Event listeners are methods that wait for an event to occur and execute a function in response.

3. Removing Event Listeners

- You can remove an event listener using `removeEventListener`.

4. Event Object

- When an event occurs, an **event object** is passed to the callback function, which provides details about the event.

Task 1: Button Click Counter

Create a button that counts the number of times it has been clicked. Display the count on the webpage.

Task 2: Change Background Color on Mouseover

When the user hovers over a div, change its background color to blue. When the mouse leaves, revert to the original color.

Task 3: Key Press Logger

Day 17: Advanced DOM Manipulation

DATE: 07-08-2024

DAY: Wednesday

What I Learned Today

I explored advanced techniques to manipulate the DOM more effectively.

1. Traversing the DOM

   - Moving between parent, child, and sibling elements.

2. Cloning Elements

   - We can clone elements using `cloneNode`.

3. Replacing Elements

   - Replace one element with another using `replaceChild`.

Task 1: Navigate the DOM Tree

Create a nested HTML structure with parent, child, and sibling elements. Write JavaScript to navigate and log different parts of the DOM tree.

Task 2: Clone and Append an Element

Create a button that clones a div element and appends it to the body.

Task 3: Replace an Element

Replace an existing element with a new one dynamically when a button is clicked.

Task 4: Dynamic List Builder

Create an input box and a button to add items to a list dynamically.

Day 18: Working with Forms

DATE: 08-08-2024

DAY: Thursday

What I Learned Today

1. Accessing Form Elements

   - Form elements can be accessed directly via their `name` attribute or using DOM methods.

2. Validating User Input

   - Adding custom validation logic using `addEventListener`.

3. Displaying Validation Messages

   - Using HTML5 validation with JavaScript for custom error messages.

Practical Tasks for the Day

Task 1: Basic Form Validation

Create a form with a text input for a username and ensure that the field is not empty on submission. Display a custom error message if validation fails.

Task 2: Real-Time Input Validation

Create a password input field that shows feedback on password strength in real-time.

Task 3: Dynamic Form Field Creation

Add a button to dynamically create and add new input fields to a form.

Task 4: Select Dropdown and Event Handling

Create a dropdown menu and display the selected value in real-time.

Day 19: JavaScript ES6 Features - Part 1

DATE: 09-08-2024

DAY: Friday

What I Learned Today

1. Arrow Functions

   - Shorter syntax for writing functions.

     Eg:

     let sum = (a, b) => a + b;

     console.log(sum(5, 10));

2. Template Literals

   - String interpolation using backticks.

     Eg:

     let name = "John";

     console.log(`Hello, ${name}!`);

3. Default Parameters

   - Set default values for function parameters.

     Eg:

     function greet(name = "Guest") {

       console.log(`Hello, ${name}!`);

     }

     greet();

Day 20: JavaScript ES6 Features - Part 2

DATE: 12-08-2024

DAY: Monday

What I Learned Today

1. Destructuring

   - Extract values from arrays or objects.

2. Spread and Rest Operators

   - Spread: Expands arrays/objects into individual elements.

   - Rest: Combines multiple elements into an array.

3. Promises

   - Handle asynchronous operations more elegantly.

Day 21: Understanding APIs (Application Programming Interfaces)

DATE: 13-08-2024

DAY: Tuesday

What I Learned Today

Today, I explored APIs, their importance, and how they enable communication between different software systems. Key takeaways:

1. What is an API?

- APIs allow different applications to communicate and exchange data.

- APIs provide a defined set of rules and endpoints for accessing services or data.

2. Types of APIs:

- REST APIs: Use HTTP requests to perform CRUD operations (GET, POST, PUT, DELETE).

3. How APIs Work:

- A client sends a request to the API endpoint (URL).

- The API processes the request and sends a response, often in JSON or XML format.

4. HTTP Methods in REST APIs:

- GET: Retrieve data.

- POST: Send new data.

- PUT/PATCH: Update existing data.

- DELETE: Remove data.

5. Basic API Call Using Fetch:

- Fetch API is used in JavaScript to make HTTP requests.

Practical Tasks for the Day

1. Task 1: Fetch Public API Data

Use the Fetch API to retrieve data from a public API, such as a weather or jokes API. Display the results on the webpage.

2. Task 2: Display API Data in a Webpage

Create a webpage to fetch and display a random dog image from the Dog API.

3. Task 3: Post Data to an API

Use a dummy API to send data via a POST request.

Day 22: Modules and Imports/Exports

DATE: 14-08-2024

DAY: Wednesday

What I Learned Today

-Modules: Use `export` and `import` to organize code into separate files.

- Exporting:

  ```javascript
  export function greet() {
    console.log("Hello!");
  }
  ```

- Importing:

  ```javascript
  import { greet } from './module.js';
  greet();
  ```

Day 23: Closures and Scope

DATE: 16-08-2024

DAY: Friday

What I Learned Today

- Closures: Functions can access variables from their outer scope even after that scope has exited.

Eg:

```
function outer() {
  let count = 0;
  return function inner() {
    count++;
    console.log(count);
  };
}
const counter = outer();
counter(); // 1
counter(); // 2
```

- Lexical Scope: JavaScript determines scope during code compilation based on the placement of functions.

Task

- Create a counter function using closures.

Day 24: Prototypes and Inheritance

DATE: 20-08-2024

DAY: Tuesday

What I Learned Today

- Prototypes: Objects inherit properties and methods from their prototype.

  Eg:

  ```
  function Person(name) {
    this.name = name;
  }
  Person.prototype.greet = function () {
    console.log(`Hello, my name is ${this.name}`);
  };
  let john = new Person("John");
  john.greet();
  ```

- Inheritance: Use `Object.create()` to set up inheritance.

  Task

- Create a prototype-based inheritance structure for a "Vehicle" and "Car".

Day 25: Debugging and Error Handling

DATE: 21-08-2024

DAY: Wednesday

What I Learned Today

- Debugging: Use browser DevTools to add breakpoints and inspect variables during runtime.

- Error Handling: Use `try-catch` blocks to handle exceptions gracefully.

Task

- Write a function that uses `try-catch` to handle invalid JSON parsing.

Day 26: Regular Expressions and Local Storage

DATE: 22-08-2024

DAY: Thursday

What I Learned Today

- Regular Expressions (Regex): Powerful patterns for text matching and manipulation.

Eg: let regex = /\d+/; // Matches digits

console.log("123abc".match(regex)); // Output: ["123"]

- Local and Session Storage: Store key-value pairs in the browser.

Eg:

localStorage.setItem("username", "John");

console.log(localStorage.getItem("username")); // "John"

Task

- Create a simple form to save user preferences in local storage and retrieve them on page load.

Day 27: Wrapping Up JavaScript

DATE: 23-08-2024

DAY: Friday

Summary

- Mastered concepts from basic syntax to advanced topics like async programming, closures, and prototypes.

- Learned how to debug, handle errors, and use browser storage effectively.

- Ready to transition into Node.js for server-side development!

Day 28: Introduction to React

DATE: 27-08-2024

DAY: Tuesday

What I Learned Today

Today, I started learning about React, a JavaScript library for building user interfaces. I understood that React is primarily used to create Single Page Applications (SPAs) and relies on components for reusability and modularity. A component is essentially a small, self-contained piece of code that renders part of the UI. React uses JSX (JavaScript XML), which is a syntax extension allowing us to write HTML directly within JavaScript.

React applications are built using a hierarchy of components. The root component (usually App.js) is the starting point, and other components are nested within it. React embraces declarative programming, focusing on what the UI should look like rather than how to achieve it procedurally.

React's approach to building UI using components is both fascinating and efficient. The introduction to JSX made me appreciate how it combines HTML and JavaScript seamlessly, making UI design more intuitive.

Tasks for Today

Install Node.js and set up a React project using npx create-react-app my-app.

Run the React development server with npm start.

Modify the default App.js file to display "Hello, React!".

Reflect on the structure of a React app and the role of each file (e.g., index.js, App.js).

Day 29: React Components

DATE: 28-08-2024

DAY: Wednesday

What I Learned Today

React applications are built from components, which can be either functional or class-based. Functional components are simpler and defined as JavaScript functions that return JSX. Class components, on the other hand, use ES6 classes and can manage their own state.

Props (short for properties) are how we pass data from one component to another. Props are immutable, making the flow of data unidirectional in React. This approach enhances predictability and maintainability.

Learning about components and props reinforced the idea of React's modular structure. The ability to pass data dynamically via props is a powerful tool for creating reusable components.

Tasks for Today

Create a functional component that accepts props to display a personalized greeting.

Write a class component that logs a message in the console when rendered.

Understand the difference between functional and class components through experimentation.

Day 30: State Management in React

What I Learned Today

DATE: 29-08-2024

DAY: Thursday

State is a built-in object used to track data that changes over time. In functional components, state is managed using the useState hook. State updates cause React to re-render the component, ensuring the UI reflects the latest data. React state is local to the component where it is defined but can be passed as props to child components.

State management is crucial for building interactive applications. The useState hook simplifies state handling and eliminates the need for class components in many cases.

Tasks for Today

Create a counter app that increments and decrements the count using useState.

Add a reset button to reset the count to zero.

Experiment with managing multiple state variables in a single component.

Day 31: React Lifecycle Methods and useEffect

DATE: 30-08-2024

DAY: Friday

What I Learned Today

Lifecycle methods are a feature of class components, but in functional components, they are replaced by the useEffect hook. useEffect handles side effects like data fetching, subscriptions, and manual DOM manipulations. It runs after the component renders and can be controlled by specifying dependencies.

The useEffect hook is incredibly versatile and simplifies handling side effects in functional components. Understanding dependencies is key to optimizing performance and avoiding unwanted re-renders.

Tasks for Today

Create a timer that counts seconds using useEffect.

Add a start/stop button to control the timer.

Implement cleanup logic to stop the timer when the component unmounts.

Day 32: useReducer Hook in React

DATE: 02-09-2024

DAY: Monday

What I Learned Today

The `useReducer` hook is an alternative to `useState` for managing more complex state logic. It is especially useful when dealing with state that involves multiple sub-values or requires sophisticated state transitions. `useReducer` works by dispatching actions to a reducer function, which determines how the state should be updated based on the action type. It provides better structure for managing complex state logic in React applications.

useReducer is helpful when state updates become complex or involve multiple actions. It organizes the logic in a clearer manner, making it easier to maintain as the application grows.

Tasks for Today

1. Implement a counter app using `useReducer`.

2. Modify the reducer function to handle multiple types of actions (reset, multiply, etc.).

3. Experiment with managing form state using `useReducer` instead of `useState`.

Day 33: useContext Hook in React

DATE: 03-09-2024

DAY: Tuesday

What I Learned Today

The `useContext` hook is used to access the value of a context directly within a component. It provides an easier and more direct way of consuming context values in functional components, without the need for a `Consumer` wrapper. This is especially useful when passing data through many layers of components (avoiding props drilling).

useContext simplifies how we access shared data across components. It is a great alternative to props drilling and makes passing data easier when working with deeply nested components.

Tasks for Today

1. Create a theme toggle using the `useContext` hook.

2. Refactor a component that needs to access a global value (e.g., user authentication status) using `useContext`.

3. Experiment with context nesting and dynamic updates.

Day 33: useRef Hook in React

DATE: 04-09-2024

DAY: Wednesday

What I Learned Today

The `useRef` hook allows us to create mutable references to DOM elements or other values that persist across renders. This is helpful for interacting with the DOM directly, like accessing input fields, or keeping track of previous state or props without triggering a re-render. It returns a mutable `ref` object that can hold any value.

`useRef` is a powerful hook for handling DOM references and mutable values that need to persist across renders. It's an excellent tool for interacting with the DOM directly in React without affecting the component state.

Tasks for Today

1. Create an input field and use `useRef` to focus it programmatically.

2. Use `useRef` to store a value that persists across renders without triggering a re-render.

3. Experiment with accessing and manipulating DOM elements using `useRef`.

Day 34: useMemo Hook in React

DATE: 05-09-2024

DAY: Thursday

What I Learned Today

The `useMemo` hook is used to memoize expensive calculations and prevent unnecessary re-computations. It returns a cached value of the result of a function unless one of its dependencies has changed. This is particularly useful when performing heavy computations or operations that could impact performance.

`useMemo` helped me optimize performance by caching values that don’t change often. It's a great tool for expensive operations or ensuring components don’t re-render unnecessarily.

Tasks for Today

1. Use `useMemo` to optimize a function that performs heavy calculations.

2. Experiment with memoization and observe performance differences.

3. Use `useMemo` to prevent unnecessary re-renders when passing props to child components.

Day 35: useCallback Hook in React

DATE: 06-09-2024

DAY: Friday

What I Learned Today

The `useCallback` hook is similar to `useMemo`, but instead of memoizing values, it memoizes functions. This is useful when passing callbacks to child components to prevent unnecessary re-renders. It only re-creates the function if the dependencies change, making it more efficient when working with functional components that re-render often.

The `useCallback` hook is very useful for optimizing performance in functional components, especially when passing functions to child components. It ensures that a function is not recreated on each render.

Tasks for Today

1. Create a button that updates the count and use `useCallback` to memoize the click handler.

2. Pass the memoized callback to a child component and observe the difference in renders.

3. Experiment with `useCallback` in combination with `useMemo` for performance optimization.

Day 36: Custom Hooks in React

DATE: 09-09-2024

DAY: Monday

What I Learned Today

Custom hooks allow us to extract component logic into reusable functions. They are a powerful feature in React that lets you share logic between components without modifying the component tree. Custom hooks are just JavaScript functions that use other hooks internally, allowing for the encapsulation of complex logic.

Custom hooks are an elegant way to reuse stateful logic. They help keep components clean and maintainable, while enabling better code organization. I found them useful for abstracting repeated logic across the app.

Tasks for Today

1. Create a custom hook to track the window width or other application-specific logic.

2. Use custom hooks to extract form validation logic, state management, or data fetching logic.

3. Share custom hooks between multiple components in your app.

Day 37: Event Handling in React

DATE: 10-09-2024

DAY: Tuesday

What I Learned Today

React provides a declarative way to handle events using camelCase syntax (e.g., `onClick`, `onChange`). Event handlers in React are passed as functions. React's synthetic events ensure compatibility across browsers.

Code Example

```
function ButtonClick() {
  const handleClick = () => {
    alert("Button was clicked!");
  };
  return <button onClick={handleClick}>Click Me</button>;
}
export default ButtonClick;
```

React's event handling system is intuitive and works seamlessly across browsers. The ability to define event handlers inline or as separate functions adds flexibility.

Tasks for Today

1. Create a button that displays an alert on click.

2. Implement an input field that logs the entered value in real time using `onChange`.

Day 38: Forms and Controlled Components

DATE: 11-09-2024

DAY: Wednesday

What I Learned Today

In React, form elements are typically controlled components, meaning their value is controlled by React state. This approach ensures that React handles all updates, making it easier to manage and validate form data.

Controlled components provide precise control over user inputs, making it easier to implement validations and dynamic behaviors in forms.

Code Example

```
function Form() {
  const [name, setName] = useState("");

  const handleSubmit = (e) => {
    e.preventDefault();
    alert(`Hello, ${name}!`);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
```

```
        value={name}
        onChange={(e) => setName(e.target.value)}
        placeholder="Enter your name"
      />
      <button type="submit">Submit</button>
    </form>
  );
}

export default Form;
```

Tasks for Today

1. Create a form to collect a user’s name and display it in an alert.
2. Add validation to ensure the input field is not empty before submission.
3. Experiment with handling multiple input fields in a single form.

Day 39: Conditional Rendering in React

DATE: 12-09-2024

DAY: Thursday

What I Learned Today

Conditional rendering in React allows us to render different UI elements based on certain conditions. This can be achieved using simple JavaScript operators such as `if`, ternary operators, or logical operators. It's a powerful way to build dynamic interfaces where parts of the UI change based on user interactions, application state, or props.

Conditional rendering allows React to be very flexible. It feels natural to use JavaScript expressions to handle dynamic changes in the UI. This was a great way to improve the user experience based on application states.

Tasks for Today

1. Create a login/logout toggle that changes the greeting message.

2. Use a ternary operator to show different components based on the state (logged in or logged out).

3. Implement a user interface that displays a loading message until data is fetched from an API.

Day 40: Lists and Keys in React

DATE: 13-09-2024

DAY: Friday

What I Learned Today

Lists in React can be rendered dynamically using the `.map()` function. Each item in the list should have a unique `key` prop to help React efficiently update the list when items change. Keys help React identify which items have changed, are added, or are removed, improving performance during re-renders.

Handling lists in React is simple yet powerful. Using `.map()` to render items allows for flexibility in displaying dynamic data, while keys ensure React performs updates efficiently.

Code Example

```javascript
function ShoppingList() {
  const items = ["Apples", "Bananas", "Cherries"];
  return (
    <ul>
      {items.map((item, index) => (
        <li key={index}>{item}</li>
      ))}
    </ul>
  );
```

Day 41: React Router for Navigation

DATE: 16-09-2024

DAY: Tuesday

What I Learned Today

React Router allows for navigation between different components or pages in a React application. It uses Route components to define paths and associate them with specific components. React Router provides Link components to navigate between these paths without reloading the page, making it suitable for SPAs.

React Router made it much easier to manage navigation in a single-page application. I appreciate how simple it is to map components to specific routes and how dynamic routing can be handled effectively.

Tasks for Today

1. Set up React Router in your app with two or more pages.

2. Add navigation links between these pages using the `Link` component.

3. Experiment with dynamic routing by using route parameters.

Day 42: Props Drilling and Context API

DATE: 17-09-2024

DAY: Wednesday

What I Learned Today

Props drilling refers to passing props from a parent component to a deeply nested child component. While this works, it can lead to unnecessary complexity in large applications. The Context API solves this by allowing data to be shared globally without passing props down manually at every level.

The Context API is a great tool for avoiding props drilling. It makes managing shared state much easier and more elegant, especially in large applications.

Code Example

```
import React, { useContext } from 'react';

const ThemeContext = React.createContext('light');

function ThemedComponent() {
  const theme = useContext(ThemeContext);
  return <div>The current theme is {theme}</div>;
}

function App() {
  return (
    <ThemeContext.Provider value="dark">
      <ThemedComponent />
    </ThemeContext.Provider>
  );} export default App;
```

Day 43: Higher-Order Components (HOCs)

DATE: 18-09-2024

DAY: Thursday

What I Learned Today

A Higher-Order Component (HOC) is a function that takes a component and returns a new component with enhanced functionality. It is a pattern for reusing component logic. HOCs are often used for adding common functionality like authentication or data fetching to multiple components.

Code Example

```
function withLoading(Component) {
  return function WithLoading(props) {
    if (props.isLoading) {
      return <div>Loading...</div>;
    }
    return <Component {...props} />;
  };
}
function MyComponent() {
  return <div>Data Loaded</div>;
}
const MyComponentWithLoading = withLoading(MyComponent);
export default MyComponentWithLoading;
```

Day 44: Code Splitting and Lazy Loading

DATE: 19-09-2024

DAY: Friday

What I Learned Today

Code splitting allows React to load only the code required for the current page, improving performance. The `React.lazy()` function enables lazy loading, which defers loading of a component until it is needed. This is especially useful for large applications where loading all components upfront is inefficient.

Lazy loading and code splitting are powerful techniques for optimizing React applications. I was able to significantly reduce the initial load time by deferring the loading of certain components.

Code Example

```javascript
import React, { Suspense } from 'react';

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

function App() {
  return (
    <Suspense fallback={<div>Loading...</div>}>
      <LazyComponent />
    </Suspense>
  );
}

export default App;
```

Day 45: Introduction to Express.js

DATE: 22-09-2024

DAY: Monday

What I Learned Today:

Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It simplifies the routing, middleware, and request handling process, making it ideal for building APIs and full-stack applications. Today, I installed Express and created a simple web server to understand how it handles HTTP requests.

Tasks for Today:

1. Install Express and set up a basic server.

2. Create routes for different HTTP methods like `GET`, `POST`, and `DELETE`.

3. Test the server by making requests using Postman or a browser.

Day 46: Middleware in Express

DATE: 23-09-2024

DAY: Tuesday

What I Learned Today:

Middleware functions in Express are used to modify the request object, the response object, or end the request-response cycle. Middleware can be built-in, like `express.json()` for parsing JSON, or custom middleware. It plays a crucial role in logging requests, handling errors, and authentication.

Tasks for Today:

1. Create custom middleware to log requests and response times.

2. Use `express.json()` to parse incoming JSON data.

3. Implement middleware to handle 404 errors and send appropriate responses.

Day 47: Routing in Express

DATE: 24-09-2024

DAY: Wednesday

What I Learned Today:

Routing is one of the core features of Express. It allows you to define the paths that clients will visit and specify how to handle each path. I learned how to define routes using `app.get()`, `app.post()`, and route parameters like `req.params` to capture dynamic values.

Tasks for Today:

1. Create different routes for an Express app (e.g., home, user profile).

2. Use route parameters to fetch data dynamically (e.g., `/users/:id`).

3. Handle different HTTP methods for each route.

Day 48: Handling HTTP Requests and Responses

DATE: 25-09-2024

DAY: Thursday

What I Learned Today:

Express simplifies handling HTTP requests and responses. I learned how to access query parameters using `req.query`, form data with `req.body`, and the request method with `req.method`. I also practiced sending JSON responses using `res.json()` and redirecting users using `res.redirect()`.

Tasks for Today:

1. Handle different types of requests (e.g., `GET`, `POST`, `PUT`, `DELETE`).

2. Send a JSON response to a client with dynamic data.

3. Practice using `req.query` to read query parameters from the URL.

Day 49: Template Engines with Express (EJS)

DATE: 26-09-2024

DAY: Friday

What I Learned Today:

Template engines allow you to render dynamic HTML pages with data passed from the server. I learned how to integrate EJS (Embedded JavaScript) with Express to create dynamic web pages. EJS makes it easy to inject variables and loops directly into HTML templates.

Tasks for Today:

1. Set up and configure EJS in an Express project.

2. Create a dynamic HTML page that loops through data passed from the server.

3. Render dynamic content based on user input or query parameters.

Day 50: Working with Static Files in Express

DATE: 29-09-2024

DAY: Monday

What I Learned Today:

Express makes serving static files, such as images, CSS, and JavaScript, simple using the `express.static()` middleware. This is important for serving assets in web applications and APIs.

Tasks for Today:

1. Set up a static directory for assets (CSS, images, etc.).

2. Serve a static HTML page along with a CSS file and image.

3. Ensure that the static files are being correctly served and loaded in the browser.

Day 51: Error Handling in Express

DATE: 30-09-2024

DAY: Tuesday

What I Learned Today:

Error handling is crucial for building robust applications. I learned how to use middleware to catch errors in Express. By defining an error-handling middleware function with 4 arguments (`err, req, res, next`), Express can handle errors that occur in routes and provide meaningful responses to the client.

Tasks for Today:

1. Implement custom error handling middleware.
2. Handle HTTP errors like 404 and 500 and send appropriate responses.
3. Test error handling by throwing errors intentionally in routes.

Day 52: Connecting to a Database with Express (MongoDB)

DATE: 01-10-2024

DAY: Wednesday

What I Learned Today:

I learned how to connect an Express app to MongoDB using the Mongoose library. Mongoose makes it easier to work with MongoDB, providing a schema-based solution to model application data. I used Mongoose to define a model for a user and perform basic CRUD operations.

Tasks for Today:

1. Install Mongoose and set up a connection to a local MongoDB database.

2. Define a simple schema for a user (e.g., `name`, `email`, `password`).

3. Create routes for creating, reading, updating, and deleting users in the database.

Day 53: POST Request Handling in Express

DATE: 03-10-2024

DAY: Thursday

What I Learned Today:

Handling `POST` requests is important for accepting and processing data from the client. I learned how to handle form submissions and send data to the server using `req.body`. This data is typically sent in JSON format when interacting with APIs.

Tasks for Today:

1. Create a route to handle `POST` requests and process the body.

2. Use middleware like `express.json()` to parse JSON data.

3. Add validation and error handling for incoming data.

Day 54: Authentication in Express

DATE: 04-10-2024

DAY: Friday

What I Learned Today:

Authentication is critical in many web applications. I implemented a simple authentication system using JSON Web Tokens (JWT). JWT allows the server to verify the identity of users without storing session data on the server.

Tasks for Today:

1. Install `jsonwebtoken` and set up JWT authentication in Express.
2. Create login and register routes that generate and verify JWT tokens.
3. Protect certain routes by verifying JWT tokens in request headers.

Day 55: Authorization in Express

DATE: 07-10-2024

DAY: Monday

What I Learned Today:

Authorization ensures that authenticated users have the right permissions to access certain resources. I learned how to implement authorization by checking user roles (admin, user) before allowing access to sensitive routes.

Tasks for Today:

1. Implement role-based authorization middleware.

2. Protect admin routes to allow only users with the “admin” role.

3. Test authorization logic by trying to access routes with different user roles.

Day 56: Express and File Uploads

DATE: 08-10-2024

DAY: Tuesday

What I Learned Today:

File uploads are commonly required in many applications. I learned how to use the `multer` middleware to handle file uploads in Express. Multer can handle multipart form-data and store files on the server or cloud.

Tasks for Today:

1. Set up `multer` to handle file uploads.
2. Create an endpoint to accept files and store them on the server.
3. Handle validation for file types and sizes.

Day 57: Using Express for RESTful APIs

DATE: 09-10-2024

DAY: Wednesday

What I Learned Today:

Express is great for building RESTful APIs. I learned how to structure APIs using REST principles, such as using appropriate HTTP methods (`GET`, `POST`, `PUT`, `DELETE`) and status codes (e.g., 200 for success, 404 for not found).

Tasks for Today:

1. Design and create RESTful API routes for managing users (CRUD operations).
2. Implement proper HTTP status codes for success and error cases.
3. Test the API using Postman or similar tools.

Day 58: Integrating Express with React

DATE: 10-10-2024

DAY: Thursday

What I Learned Today:

Express can be used as a backend API for a React frontend. I set up an Express server and a React app to communicate via API requests. Express serves the backend logic, and React consumes the API for dynamic content.

Tasks for Today:

1. Set up a basic React app to interact with the Express backend.
2. Create API calls in React using `fetch` or `axios`.
3. Handle API responses and update the React state accordingly.

Day 59: Deploying Express App

DATE: 11-10-2024

DAY: Friday

What I Learned Today:

I learned how to deploy an Express application to production using platforms like Heroku. This involves configuring the server, setting up environment variables, and ensuring the app is ready to run in a production environment.

Tasks for Today:

1. Deploy the Express app to Heroku or a similar platform.
2. Set up environment variables for production (e.g., database URL).
3. Test the deployed app in the production environment.

Day 60: Advanced Topics in Express (Session Management, Caching, Performance)

DATE: 14-10-2024

DAY: Monday

What I Learned:

Over these days, I deep-dived into advanced Express topics, such as session management, API rate-limiting, and caching. I implemented session handling using `express-session`, rate-limiting to prevent abuse, and caching to improve API performance.

Tasks for Today:

1. Implement session management to store user data between requests.

2. Set up rate-limiting for APIs to prevent DoS attacks.

3. Implement caching to optimize API response times.

Day 61: Project Kickoff - Introduction to Service Nest

DATE: 15-10-2024

DAY: Tuesday

Today marks the beginning of my project, **Service Nest**, an online service-based platform. I started by outlining the purpose of the project—connecting users to various services such as home repairs, personal training, tutoring, etc. The platform will have user accounts, service listings, and a booking system.

Tasks done Today:

1. Define project scope, goals, and objectives.
2. Research and choose the tech stack: MERN (MongoDB, Express, React, Node).
3. Start planning user flows and wireframes for the core pages (Home, Service Listings, Service Details, User Profile, etc.)

Day 62: Setting up the Project Structure

DATE: 16-10-2024

DAY: Wednesday

I began by setting up the initial project structure for both the backend (Node + Express) and frontend (React). I also configured Git for version control to keep track of changes and collaborate effectively.

Tasks done Today:

1. Initialize the backend (Express app) and frontend (React app).
2. Set up folder structure for both backend and frontend
3. Commit the initial project structure to GitHub.

Day 63: Database Design and Setup

DATE: 17-10-2024

DAY: Thursday

I designed the MongoDB schema to manage the services, user data, and bookings. I also started setting up MongoDB Atlas for a cloud-based database solution.

Tasks for Today:

1. Create MongoDB schemas for Users, Services, and Bookings.
2. Connect the backend to MongoDB using Mongoose.
3. Test database operations (Create, Read, Update, Delete) for the services.

Day 64: User Authentication with JWT

DATE: 18-10-2024

DAY: Thursday

I implemented user authentication using JWT (JSON Web Tokens). This allows users to sign up, log in, and securely access restricted areas of the platform, like their profile and bookings.

Tasks for Today:

1. Set up user authentication routes (register, login).
2. Implement JWT token generation and validation.
3. Protect certain routes using JWT for authorization.

Day 65: User Profile Setup

DATE: 19-10-2024

DAY: Friday

Today, I focused on creating a user profile page, where users can view their personal information and service bookings. I also set up a basic user dashboard for users to update their details.

Tasks for Today:

1. Build the user profile page with editable fields (name, email, password).
2. Set up routes to manage user data.

Day 66: Service Listings Page

DATE: 22-10-2024

DAY: Monday

I created the service listings page, where users can browse different services offered.

Tasks done Today:

1. I Builded the service listings page with filters (category, price, location).
2. I Created a service card component to display individual service details.

Day 67: Service Detail Page

DATE: 23-10-2024

DAY: Tuesday

The service detail page shows more information about a selected service, including provider details, pricing, and availability. I learned how to dynamically display data from the database using React.

Tasks done Today:

1. Create a service detail page.
2. Display data dynamically (e.g., provider name, description, images).
3. Add a booking button that directs to the booking page.

Day 68: Booking System Setup

DATE: 24-10-2024

DAY: Wednesday

I developed the booking system, where users can select a service, choose a time slot, and confirm their booking. This required integrating the service with a time slot availability system.

Tasks done Today:

1. Set up a booking form on the service detail page.
2. Allow users to select a date and time for booking.
3. Store booking data in the MongoDB database.

Day 69: Notifications

DATE: 25-10-2024

DAY: Thursday

I learned how to send notifications using react toastify.

Day 70: Service Provider / User Dashboard Setup

DATE: 26-10-2024

DAY: Friday

I created dashboard for users, services, and bookings.

Day 71: Services Provided

DATE: 29-10-2024

DAY: Monday

I added advanced search and filter functionality to the service listings page. This makes it easier for users to find services based on specific criteria like category, price range, or location.

Day 72: Rating and Reviews System

DATE: 30-10-2024

DAY: Tuesday

I implemented a rating and review system for services. Users can now leave feedback after booking a service, helping others make informed decisions.

Tasks for Today:

1. Create a rating component for users to leave reviews.
2. Set up a review form where users can add their comments and ratings.
3. Display reviews and ratings on the service detail page.

Day 73: Refactoring and Optimizing Code

DATE: 31-10-2024

DAY: Wednesday

I focused on refactoring and optimizing the codebase to improve performance and readability. This involved breaking down large components into smaller, reusable ones.

Tasks done Today:

1. Refactor the service listings and booking components.
2. Use React hooks (like useEffect and useState) to optimize component re-renders.
3. Test for any redundant code and remove it.

Day 74: Responsive Design for Mobile

DATE: 04-11-2024

DAY: Monday

I focused on making the platform mobile-responsive. This included adjusting layouts, font sizes, and buttons for better usability on smartphones and tablets.

Tasks for Today:

1. Implement media queries for responsive layouts.
2. Test the platform on multiple screen sizes.
3. Optimize images and content for mobile devices.

Day 75: Testing Functionalities (Manual)

DATE: 05-11-2024

DAY: Tuesday

I conducted manual testing to ensure that all functionalities, including user registration, booking, work as expected. This helped catch any bugs or usability issues.

Tasks for Today:

1. Test the user registration and login flows.
2. Test the booking process from start to finish.

Day 76: Bug Fixes and Refinement

DATE: 06-11-2024

DAY: Wednesday

After the manual testing, I fixed several bugs related to the booking system, and mobile responsiveness. Refining the user experience was a key focus.

Tasks for Today:

1. Fix bugs and address edge cases found during testing.
2. Improve form validation for booking.
3. Test the application again after implementing fixes.

Day 79 : Finalizing the Project

DATE: 07-11-2024

DAY: Thursday

In these final days, I focused on completing the remaining features, fixing bugs, testing, and preparing for deployment. I also worked on user documentation. The last phase involved thorough testing (unit tests, integration tests) to ensure the app functions correctly under different scenarios.

Tasks:

1. Complete any remaining features and finalize the user interface.
2. Conduct unit and integration tests for all major functions.
3. Create and update user documentation for using the platform

Day 80: Enhancing the Home Page

DATE: 08-11-2024

DAY: Thursday

Focused on improving the aesthetics of the home page. Added engaging banners, refined the service categories section, and ensured better navigation.

Tasks:

1. Add a hero section with a call-to-action (CTA) for quick service booking.

2. Style the category cards with hover effects and subtle animations.

3. Optimize the header and footer for consistency.

Day 81: Service Listing

Worked on the service listing page.Ensured the layout adapts well on all screen sizes.

Day 82: Service Details Page

Developed a detailed service page that displays all essential information like service description, ratings, reviews, and booking options.

Tasks:

1. Designed an intuitive layout for the service details page.

2. Added a review section with ratings.

3. Implemented a booking button that leads to a form/modal.

Day 83: User Profile Section

Focused on the user profile page. Designed it to allow users to update their details, view booking history, and manage preferences.

Day 84: Integrating Frontend and Backend (Part 1)

Began integrating the frontend with the backend using APIs for user authentication and service management.

Tasks done:

1. Set up API calls for user login and signup.

2. Test API integration for service listings.

3. Handle errors gracefully with user-friendly messages.

Day 86: Integrating Frontend and Backend (Part 2)

Completed integration for features like booking services, fetching user data, and updating profiles.

Tasks done:

1. Add dynamic booking functionality linked to the backend.

2. Integrate user profile management with database updates.

3. Test the integration extensively for potential edge cases.

Day 87: Bug Fixing and User Feedback

Fixed bugs identified during user testing and started incorporating feedback for better usability.

Tasks:

1. Fix alignment issues on mobile screens.

2. Correct API calls that were returning incorrect data.

3. Add loading spinners for long API requests.

Day 88: Adding Enhancements

Focused on user engagement by adding smooth enhancements.

Day 89: Testing Cross-Browser Compatibility

Tested the application on browsers to ensure compatibility and fixed any inconsistencies.

Tasks:

1. Test on Chrome.

2. Resolve style differences due to browser-specific behavior.

3. Validate HTML and CSS for standards compliance.

Day 90: Performance Optimization

Worked on improving the performance of the frontend by optimizing assets and reducing render times.

Tasks done:

1. Minify CSS, JavaScript, and images.

2. Use lazy loading for images and components.

3. Implement efficient state management to avoid unnecessary re-renders.

Day 91: Adding Accessibility Features

Focused on making the platform accessible for all users by adhering to web accessibility standards.

Tasks done:

1. Ensure proper navigation for the entire app.

2. Test color contrast ratios for readability.

Day 92: Error Pages and Alerts

Created custom error pages and alert messages for a better user experience during errors or downtime.

Tasks done:

1. Design 404 and 500 error pages with a link back to the home page.

2. Add toast notifications for success and error messages.

3. Test error handling for all user actions.

Day 94: Finalizing the Frontend Codebase

Reviewed and cleaned up the frontend codebase, ensuring it adheres to best practices.

Tasks done:

1. Remove unused styles and components.

2. Refactor repetitive code into reusable components.

3. Document the structure and important functions.

Day 95: Frontend Testing and Feedback Incorporation

Conducted the final round of frontend testing and made small adjustments based on user feedback.

Tasks done:

1. Run UI tests on all major workflows (booking, profile updates, etc.).

2. Adjust styles for better readability on small screens.

3. Fix any last-minute issues reported by testers.

Day 96-107: Final Adjustments

Prepared the frontend by ensuring all features are production-ready. Monitored the live application for any issues and resolved them promptly.

Tasks:

1. Monitor user feedback and fix bugs in real-time.

2. Document the frontend workflows for easy handoff or future reference.

Steps Undertaken to Finalize the Project:

Frontend Refinements:

Polishing the UI: Ensuring the design is clean, consistent, and responsive across all devices. Adjustments were made to improve the overall user experience, such as optimizing button placements, refining color schemes, and ensuring font readability.

Improving Animations and Transitions: Adding subtle animations and smooth transitions for better user interaction and engagement.

Testing Across Devices: Conducting thorough cross-browser and cross-device testing to ensure the application functions seamlessly on desktops, tablets, and smartphones.

Bug Fixing:

Critical Bug Fixes: Addressing issues such as form validation errors, broken links, and inconsistencies in data flow between components.

Database Connectivity Issues: Ensuring that the database queries are executed efficiently without errors.

Error Handling: Adding proper error messages for failed actions, such as incomplete form submissions or payment processing failures, to guide users effectively.

Implementing Feedback:

Gathering input from testers and mentors to identify areas of improvement. This included making the navigation more intuitive, refining the search functionality, and simplifying the admin panel.

Adding features that enhance usability, such as tooltips, progress indicators, and better call-to-action buttons.

Backend Optimization:

Ensuring the API endpoints are optimized for performance. Queries were reviewed and updated to reduce latency.

Improving error logs for easier debugging and monitoring in the production environment.

Testing backend scalability to ensure it handles an increasing number of concurrent users without crashing.

Changes and Enhancements:

Adding Missing Features: For example, integrating a simple FAQ or help section, setting up email notifications for certain actions, or refining the invoice generation system.

Performance Optimizations: Compressing images, enabling caching, and optimizing scripts to reduce load times.