React Notebook

~ BY SAGAR BISWAS

Prerequisites:

Before diving into React, make sure you're familiar with:

1. **HTML** – the structure of web pages
2. **CSS** – styling your web pages
3. **JavaScript** – adding interactivity and logic

Part 1: Introduction to React

What is React?

React is a **flexible**, **efficient**, and **open-source JavaScript library** used for building **user interfaces (UIs)**—mostly for the **frontend**.

- Developed by **Facebook** in **2013**.
- Created by **Jordan Walke**, a Facebook software engineer.
- Mainly used for building **interactive UI components**.

❓ Is React a Library or a Framework?

Answer: React is a **JavaScript library**, not a framework.

Library 🆚 Framework

Criteria	Library	Framework
Definition	Collection of functions and classes	Collection of tools and libraries
Control	**You** call the code	The **framework** calls your code
Flexibility	More flexible and lightweight	More structured and opinionated

💡 Why Use React?

- Helps you build complex UIs using **components**
- Encourages **code reuse** with reusable components
- Fast performance due to **component-based rendering**
- Supports **integration with other libraries and frameworks**
- Powering **~45% of websites** worldwide
- Used by top companies: **Facebook, Instagram, Yahoo, Airbnb, Dropbox, Netflix**

❓ What is a Prompt?

A **prompt** is a way to get input from the user.

★ Example:

```
const name = prompt("What is your name?");
```

❓ What is a Component?

A **component** is a **small, isolated, and reusable piece of code** in React.

Think of a component as a block of UI – like a button, header, or footer – that you can **reuse** across your website or app.

❓ What is State?

In React, **state** is a piece of data that **can change** over time.

★ Example: A user's name, login status, or the number of likes on a post.

★ Example: **A React Component (with HTML, CSS, and JavaScript)**

File: Message.js

```
// Import the React library
import React from 'react';

// Define a functional component called Message
export default function Message() {
  return (
    // Inline styling with black background and centered text
    <h1 style={{ background: 'black', color: 'white', textAlign: 'center' }}>
      Welcome to React
    </h1>
  );
}
```

✦ Using the Component:

To **reuse** the Message component in another file:

```
// Import the Message component
import Message from './Message';

// Use it like an HTML tag
<Message />
<Message />
```

📝 React lets you reuse components like this anywhere in your project.

🏳 Part 2: Environment Setup

Before building amazing React apps, let's first set up your development environment properly.

☠ Install the Necessary Tools

1. **Visual Studio Code (VS Code)**

   A powerful, lightweight code editor used by millions of developers.

🔗 Download it from: https://code.visualstudio.com/

2. **Node.js**

Node.js lets you run JavaScript **outside the browser** and includes npm (Node Package Manager), which is used to install React and other packages.

🔗 Download it from: https://nodejs.org/en/download/

🔍 Verify Node.js Installation

After installing Node.js, open your terminal or command prompt and run:

```
node –version
```

You should see a version number, like v18.17.1 — this means Node.js is installed correctly.

To check npm (Node Package Manager):

```
npm --version
```

☠ Useful Command Line Basics

Here are some important terminal/command prompt commands you'll use often:

Command	Description
pwd	Print working directory (shows current folder)
cd	Change directory
ls or dir	List files and folders in current directory
mkdir	Make a new folder/directory
rmdir	Remove an empty folder
rm	Remove files
cls	Clear the terminal (Windows)
clear	Clear the terminal (Mac/Linux)
cd ..	Go back one folder level
code .	Open current directory in VS Code (if installed)
npx	Run Node package without installing globally
npm install	Install project dependencies
npm start	Start a development server (used in React)

🏳 Part 3: Setting Up React with Vite

Vite is a **modern, fast** build tool for React applications. Unlike create-react-app, Vite offers **near-instant server start** and optimized builds.

☠ Why Vite?

- **Blazing Fast** – Uses native ES modules for quicker development.

- **Lightweight** – No heavy tooling like Webpack (used in create-react-app).
- **Modern** – Supports React, Vue, Svelte, and more.

🔗 Official Site: https://vite.dev/guide/

☠ Creating a React Project with Vite

✓ Step 1: Run the Vite Setup Command

Open your terminal and run : `npm create vite@latest`

✓ Step 2: Follow the Prompts

You'll be asked:

1. **Project name** : my-app (or any name you prefer)
2. **Framework** : Select React
3. **Variant** : Choose JavaScript (or TypeScript if preferred)

★ Example:

```
Project name       : my-app
Select a framework : React
Select a variant   : JavaScript
```

✓ Step 3: Navigate into the Project & Install Dependencies

```
cd my-app                 # Move into the project folder
npm install               # Installs all required packages (creates node_modules)
code .                    # Opens VSCode in the current directory
```

✓ Step 4: Start the Development Server

```
npm run dev     # Runs the app in development mode
```

- Opens at http://localhost:5173 (default Vite port).

☠ Understanding the Project Structure

After setup, your project will look like this:

```
my-app/
├── node_modules/  # All installed packages
├── public/        # Static assets (images, etc.)
├── src/           # React source code
│   ├── App.jsx    # Main React component
│   ├── main.jsx   # Entry point
│   └── (Other files)
├── index.html     # Root HTML file
├── package.json   # Project config & scripts
└── vite.config.js # Vite configuration
```

☠ There are Two Types of React Components

React supports **two ways** to define components:

Functional Components	**Class Components**
Modern (recommended)	Older approach (legacy)
Uses **functions**	Uses **ES6 classes**
Uses **Hooks** (e.g., useState)	Uses this.state and lifecycle methods

★ Example: **Functional Component (Used in Vite Default Setup)**

```
// src/App.jsx
function App() {
  return (
    <div className="App">
      <h1>Hello Vite + React!</h1>
    </div>
  );
}
export default App;
```

Part 3 (continued): Create a React Project Using CRA (React 18)

While **Vite** is the modern tool of choice for speed, **CRA (Create React App)** is still widely used and perfect for beginners learning React with official configurations.

Step-by-Step: Create React App Using CRA

✓ Open Terminal in VS Code

Navigate to your desired folder and run:

```
npx create-react-app my-app
```

This will create a new folder named my-app and scaffold a full React project inside it.

Tip: Create React App in the Current Directory

If you want to install React **in the current folder** (without creating a new one), use:

```
npx create-react-app .
```

Make sure the folder is **empty** before doing this to avoid conflicts.

Run Your React Project

Once the setup is complete:

```
cd my-app       # Go into your project folder
npm start       # Start the React development server
```

This will automatically open the app in your default browser, usually at http://localhost:3000

Dependency Notes

When you create a project using CRA:

- The **react-dom** and other packages are listed in the package.json file.

- Their **exact versions** and dependency structure are locked using the package-lock.json file.

📂 All the actual downloaded packages are stored in the node_modules/ folder, If the node_modules/ folder does not exist, navigate to the file location and type `npm install` in the terminal.

★ Example: **A Simple "Hello World" Page in React**

You'll find the main component in src/App.js:

```
function App() {
  return (
    <div className="App">
      <h1>Hello World!</h1>
    </div>
  );
}

export default App;
```

This JSX code displays a simple "Hello World!" message on the web page.

Part 4: JSX and JavaScript Expressions

What is JSX?

JSX stands for **JavaScript XML**.

It's a **syntax extension for JavaScript** that allows you to write HTML-like code **directly inside JavaScript files**—which React understands and compiles into actual DOM elements.

Key Points About JSX:

- JSX code must be **wrapped inside one parent element** when passed to render(). Example: <div></div>, <></>.
- We can **embed JavaScript expressions inside {}** in JSX. Example: {2 + 2}, {name}, {isActive ? "Active" : "Inactive"}.

★ Example: **Using JSX and JavaScript Inside index.js**

```
// Import ReactDOM to render elements
import ReactDOM from "react-dom/client";

// A simple variable
const name = "Sagar Biswas";

// Getting the current date
const date = new Date();
const year = date.getFullYear();
const month = date.getMonth() + 1; // JavaScript months are 0-based
const day = date.getDate();

// A paragraph description
const pera = "I am a software engineer with a passion for building web applications and
learning new technologies.";

// Getting the current time
const time = new Date().toLocaleTimeString();

// Linking the root element from index.html
```

```
const root = ReactDOM.createRoot(document.getElementById("root"));

// Rendering JSX with JavaScript expressions
root.render(
  // JSX must return only ONE root element (like <div>)
  <div>
    <h1>Hello World!</h1>

    {/* Embedding a JavaScript variable */}
    <h3>I am {name}.</h3>

    {/* Displaying a string variable */}
    <p>Description: {pera}</p>

    {/* Displaying date values from JavaScript */}
    <p>Date: {month}/{day}/{year}</p>

    {/* Using an inline JavaScript expression inside JSX */}
    <h5>{"Time: " + time + " O'Clock"}</h5>
  </div>
);
```

💡 **Tips:**

- **Use JSX Fragment for Multiple Elements:**

  Wrap multiple elements using <>...</> (fragment syntax):

  ```
  return (
    <>
      <h1>Hello</h1>
      <p>World</p>
    </>
  );
  ```
- **Always return one element** in JSX (commonly a <div> or <React.Fragment>).
- **JS expressions** (not statements like **if** or **for**) can be embedded using **{}**.
- You can style elements using either:

  ```
   {/* inline CSS styling */}
  <h1 style={{ color: "red", fontSize: "50px" }}>Hello World!</h1>
  ```

Part 5: CSS Styling in JSX

Styling in React can be done in several ways. The most common methods include:

Ways to Add CSS in React:

1. **Inline Styling** (CSS as JavaScript objects)
2. **External CSS Stylesheets**
   - Method 1: **Link** CSS via public/index.html
   - Method 2: **Import** CSS directly into a .js or .jsx file

Method 1: Using CSS from the public/ Folder

This is similar to how we style in plain HTML files.

✓ **Step 1: Create style.css inside the public/ folder**

📂 **my-app/public/style.css**

```css
/* CSS stylesheet (public folder) */

.date {
  color: #dd07f0;
}

.time {
  color: #f0a207;
  text-decoration: underline;
  text-decoration-color: #f00707;
  font-size: 17px;
}
```

✓ **Step 2: Link it inside** 📂 **public/index.html**

```html
<!-- my-app/public/index.html -->
<link rel="stylesheet" href="style.css" />
```

⚠️ Only files in the **public/ folder** can be directly linked like this.

☠️ **Method 2: Importing CSS File Inside JavaScript**

This is the **modern React way** of applying CSS.

✓ **Step 1: Create index.css inside src/ folder**

📂 **Path: my-app/src/index.css**

```css
/* CSS file inside src folder */

#description {
  color: #2ef007;
  background-color: black;
}
```

✓ **Step 2: Import CSS in index.js**

```js
import "./index.css";
```

★ **Example: Inline + External Styling Together**

```js
// 📂 File: src/index.js
import ReactDOM from "react-dom/client";
import "./index.css"; // Importing internal CSS (Method 2)

// JS variables for dynamic content
const name = "Sagar Biswas";
const date = new Date();
const year = date.getFullYear();
const month = date.getMonth() + 1;
const day = date.getDate();
const time = new Date().toLocaleTimeString();

// Inline style using a JS object
const inLineCss = {
  color: "blue",
  fontSize: "20px",
};

// React root render
const root = ReactDOM.createRoot(document.getElementById("root"));
```

```
root.render(
  <div>
    {/* Inline styling directly */}
    <h1 style={{ color: "red", fontSize: "50px" }}>Hello World!</h1>

    {/* Inline styling using a variable */}
    <h3 style={inLineCss}>I am {name}.</h3>

    {/* Styling with ID (via index.css) */}
    <p id="description">
      I am a software engineer with a passion for building web applications and learning new
technologies. I love coding and enjoy solving complex problems.
    </p>

    {/* Method 1: Using class from public/style.css */}
    <h5 className="date">
      Date: {month}/{day}/{year}.
    </h5>

    {/* Method 2: Full expression inside {} using class */}
    <h5 className="time">{"Time: " + time + " O'Clock;"}</h5>
  </div>
);
```

☠ Part 6: How to Create and Use React Components

In React, **components** are the building blocks of your UI. They let you **reuse** code and keep your codebase clean and organized.

💡 Naming Rule

- **Always start component names with a capital letter.**

    ✅ Good: MyComponent

    ❌ Bad: myComponent

React treats **lowercase tags like HTML** (<div>, <h1>) and **uppercase** tags as **custom components** (<MyComponent />).

☠ Method 1: Create Component Directly in index.js

📂 **File: src/index.js**

```
import ReactDOM from "react-dom/client";

// Define a simple functional component
function MyDetails() {
  return (
    // JSX must return a single root element
    <div>
      {/* Inline styling */}
      <h1 style={{ color: "red", fontSize: "50px" }}>Hello World!</h1>
    </div>
  );
}

// Render the component
const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(
  <div>
    {/* Using the component multiple times */}
    <MyDetails />
    <MyDetails />
    <MyDetails />
  </div>
);
```

☠ Method 2: Create Component in a Separate File (Import & Export)

This is the preferred and scalable way to organize your components in React apps.

📂 File Structure

```
my-app/
└── src/
    ├── components/
    │   └── HelloWorld.js
    └── index.js
```

✔ Step 1: Create the Component File

📂 File: src/components/HelloWorld.js

```
// Define a functional component
function MyDetails() {
  return (
    <div>
      <h1 style={{ color: "red", fontSize: "50px" }}>Hello World!</h1>
    </div>
  );
}

// Export the component so it can be used in other files
export default MyDetails;
```

✔ Step 2: Import and Use the Component

📂 File: src/index.js

```
import ReactDOM from "react-dom/client";
import MyDetails from "./components/HelloWorld"; // Import the component; <MyDetails />

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(
  <div>
    <MyDetails />
    <MyDetails />
    <MyDetails />
  </div>
);
```

☠ Key Takeaways:

Topic	Explanation
Component Name	Must start with a capital letter (e.g., MyDetails)
Functional Components	Are just JavaScript functions that **return JSX**
Reusability	You can use a component multiple times like <MyDetails />

Modularity	Use export to make a component usable in other files via import

---

🏳 Part 7: Props and Destructuring in React

☠ What Are Props?

Props (short for **properties**) are used to **pass data from one component to another**—usually from a **parent component** to a **child component**.

They allow components to be **dynamic** and **reusable** by accepting custom inputs.

🎯 Purpose of Props

"To pass values into a component so that the component can use those values."

☠ Step-by-Step Example

📂 File: src/components/HelloWorld.js

```
// MyDetails component receives props
function MyDetails(props) {
  // Log to see what props are received
  console.log("props:", props);

  // Destructure values from props
  const { titleText, descText } = props;

  return (
    <div>
      {/* Styling applied inline */}
      <h1 style={{ color: "red", fontSize: "50px" }}>Hello World!</h1>

      {/* Using props directly */}
      <p>{"Title (via props): " + props.titleText}</p>
      <p>{"Description (via props): " + props.descText}</p>

      {/* Using destructured props */}
      <p>{"Title (via destructuring): " + titleText}</p>
      <p>{"Description (via destructuring): " + descText}</p>
    </div>
  );
}

// Array destructuring (bonus example)
const myArray = ["Hello", "World", 123, true];
const [greeting, subject, number, isTrue] = myArray;
console.log(greeting, subject, number, isTrue); // Output: Hello World 123 true

// Export component for use in App.js or index.js
export default MyDetails;
```

📂 File: src/App.js

```
// Import child component
import MyDetails from "./components/HelloWorld"; // <MyDetails />

function App() {
  return (
    <div>
      {/* Passing props to MyDetails */}
```

```
        <MyDetails
          titleText="This component is a CHILD of App.js and index.js"
          descText="It is imported from HelloWorld.js and reused here."
        />

        {/* Text inside the App component */}
        <p>Inside App component in App.js – this is a return block.</p>
      </div>
    );
  }

  export default App; // Export App component for index.js
```

File: src/index.js

```
import ReactDOM from "react-dom/client";
import App from "./App";
import MyDetails from "./components/HelloWorld"; // (Optional reuse here); <MyDetails />

const root = ReactDOM.createRoot(document.getElementById("root"));

// Render the main component
root.render(
  <div>
    {/* App component contains MyDetails inside */}
    <App />

    {/* Reusing the same component again without props */}
    <MyDetails />
    <MyDetails />
    <MyDetails />
  </div>
);
```

Key Takeaways

Concept	Explanation
props	Data passed from parent to child component
Destructuring Props	A cleaner way to extract values from props
Reusability	You can use the same component with different props multiple times
Array Destructuring	Works the same way as object destructuring, useful for handling **arrays**

Best Practices:

- Use **destructuring** in function arguments or inside the component for cleaner code.
- Always **define default props or prop types** when building larger apps.
- Only use **props in child components**, never try to pass data upward directly (use state lifting or context for that).

Part 8: Mapping Data to Components in React

Purpose

To display dynamic data by **mapping** over an **array** and passing each item as **props** into a component.

Instead of writing <MyDetails /> multiple times, we can **loop through a JSON array** and generate components programmatically using **.map().**

✓ **Step 1: Add Your Data File**

📂 **File: src/data.json**

```
[
  {
    "title": "Introduction to JavaScript",
    "description": "Learn the basics of JavaScript, including variables, data types, and control structures.",
    "url": "https://example.com/javascript-intro"
  },
  {
    "title": "Advanced JavaScript Concepts",
    "description": "Dive deeper into JavaScript with topics like closures, promises, and async/await.",
    "url": "https://example.com/javascript-advanced"
  },
  {
    "title": "JavaScript Frameworks Overview",
    "description": "An overview of popular JavaScript frameworks such as React, Angular, and Vue.js.",
    "url": "https://example.com/javascript-frameworks"
  }
]
```

✓ **Step 2: Create the Reusable Component**

📂 **File: src/components/HelloWorld.js**

```
// A simple functional component that accepts props
function MyDetails(props) {
  const { titleText, descText } = props;

  return (
    <div>
      {/* Styled header */}
      <h1 style={{ color: "red", fontSize: "50px" }}>Hello World!</h1>

      {/* Using props directly */}
      <p>Title (via props): {props.titleText}</p>
      <p>Description (via props): {props.descText}</p>

      {/* Using destructured values */}
      <p>Title (destructured): {titleText}</p>
      <p>Description (destructured): {descText}</p>
    </div>
  );
}

// Optional array destructuring demo (for learning purposes)
const myArray = ["Hello", "World", 123, true];
const [greeting, subject, number, isTrue] = myArray;
console.log(greeting, subject, number, isTrue); // Hello World 123 true

export default MyDetails;
```

✓ **Step 3: Read JSON Data and Pass It as Props**

📂 **File: src/App.js**

```
import MyDetails from "./components/HelloWorld";
import values from "./data.json"; // Importing the JSON data array

function App() {
  // Debugging logs
  console.log("values from JSON:", values);
  console.log("First Title:", values[0].title);
  console.log("Third Description:", values[2].description);

  return (
    <div>
      {/* Passing props manually */}
      <MyDetails
        titleText="This component is a CHILD of App.js"
        descText="Props passed from App.js manually"
      />

      <p>Inside the App component. This is the returned JSX block.</p>
    </div>
  );
}

export default App;
```

✓ **Step 4: Use .map() to Render Data Dynamically**

📂 **File: src/index.js**

```
import ReactDOM from "react-dom/client";
import MyDetails from "./components/HelloWorld";
import App from "./App";
import Values from "./data.json"; // Importing array of data

const root = ReactDOM.createRoot(document.getElementById("root"));

// Mapping each object from the JSON file into a React component
const items = Values.map((item, index) => (
  <MyDetails
    key={index}                      // Unique key for each component as a prop
    titleText={item.title}           // Passing title as a prop
    descText={item.description} // Passing description as a prop
  />
));

root.render(
  <div>
    <App />

    {/* Dynamically rendering MyDetails for each JSON item */}
    {items}
  </div>
);
```

☠ **Key Concepts**

Concept	Description
.map()	Loops through an array and returns a new array of components
props	Used to pass individual values into each component
key prop	A unique identifier to help React efficiently update and render components
Reusable UI	The same component can show different content based on the passed data

☠ **Why Use .map() Instead of Writing Manually?**

- Easier to maintain
- Dynamically scalable
- Cleaner, more readable code

Part 9: Mapping Data with a Unique ID in React

Purpose

To **uniquely identify** each component instance when rendering a list of data with .map() — so that React can update the DOM efficiently and avoid rendering bugs.

Step 1: Install UUID

To generate unique IDs, use the uuid library.

Terminal Command:

```
npm install uuid
```

This allows us to generate a **universally unique identifier** using uuidv4().

File Structure

```
my-app_Part-9/
└── src/
    ├── components/
    │   └── UniqueList/
    │       └── List.js
    ├── App.js
    └── index.js
```

List.js – Mapping Data with Unique ID

File: src/components/UniqueList/List.js

```
import React from "react";
import { v4 as uuidv4 } from "uuid"; // UUID library for generating unique IDs

// Job data array with unique IDs generated using uuidv4()
const Jobs = [
  {
    id: uuidv4(),
    title: "Software Engineer",
    description: "Develop and maintain software applications.",
  },
  {
    id: uuidv4(),
    title: "Web Developer",
    description: "Design and develop websites and web applications.",
  },
  {
    id: uuidv4(),
    title: "UI/UX Designer",
    description: "Create user interfaces and user experiences.",
  },
];

const List = () => {
  console.log("Example generated unique ID:", uuidv4());
  return (
    <div>
      {/* Map through each job item */}
      {Jobs.map((job) => {
        const { id, title, description } = job;

        return (
          <div key={id}>
            {/* key is used to uniquely identify each job item */}
            <h3>{title}</h3>
            <p>{description}</p>
            <p>The unique id is: {id}</p>
          </div>
        );
      })}
    </div>
  );
};

export default List;
```

✯ **App.js – Using the List Component**

📁 **File: src/App.js**

```
import React from "react"; // Required for React components
import List from "./components/UniqueList/List"; // Import the List component

export default function App() {
  return (
    <div>
      <List />
    </div>
  );
}
```

✯ **index.js – Starting Point of the App**

📁 **File: src/index.js**

```
import ReactDOM from "react-dom/client";
import App from "./App";

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(
  // React apps must render one root element
  <div>
    <App />
  </div>
);
```

☠ **Why Unique IDs Matter in React**

Feature	**Why It's Important**
key prop in .map()	Helps React identify which items changed, added, or removed
UUID	Ensures globally unique values across renders
Re-render safety	Prevents bugs and improves performance in large lists

🏳 Part 10: Nested Mapping in React

🎯 What is Nested Mapping?

Nested mapping is when you use .map() **inside another .map()** — usually needed when your data is structured in **arrays inside objects inside arrays**.

This is especially useful when you have **nested arrays**, like a user having multiple phone numbers.

☠ Example Data Structure

We have an array of users. Each user has:

- A name
- An age
- A phones array (which includes both home and office numbers)

★ Component: Mapping Nested Data

📁 File: src/App.js

```
const users = [
  {
    name: "Sagar Biswas",
    age: 22,
    phones: [
      {
        home: "01939471097",
        office: "01818261077",
      },
    ],
  },
  {
    name: "XYZ Biswas",
    age: 20,
    phones: [
      {
        home: "01626351077",
        office: "01626351078",
      },
    ],
  },
];

export default function App() {
  return (
    <div>
      <h1>Nested List</h1>

      {/* Looping through users array */}
      {users.map((user, index) => (
        <article key={index}>
          {/* article tag used for grouping related content */}
          <h3>Full Name: {user.name}</h3>
          <p>Age: {user.age}</p>

          {/* Looping through each user's phones array */}
          {user.phones.map((phone, index) => (
            <div key={index}>
              <h4>Phones</h4>
              <p>Home: {phone.home}</p>
              <p>Office: {phone.office}</p>
            </div>
          ))}
        </article>
      ))}
    </div>
  );
}
```

☠ **Output:**

The UI will display:

Nested List

Full Name: Sagar Biswas

Age: 22

Phones

Home: 01939471097

Office: 01818261077

Full Name: XYZ Biswas

Age: 20

Phones

Home: 01626351077

Office: 01626351078

☠ **Key Points**

Concept	**Explanation**
.map()	Loops through an array to return JSX for each item
key prop	Required for each element inside a .map() to uniquely identify it
Nested .map()	Needed when dealing with **arrays inside objects** (like phones)
Semantic HTML	Using <article>, <h3>, <p> makes your structure clean and meaningful

🏁 **Entry Point**

📁 **File: src/index.js**

```
import ReactDOM from "react-dom/client";
import App from "./App";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<App />);
```

🏳 Part 11: Assignment-1 | Product Listing App

🎯 **Assignment Title: React Product Listing App**

Total points = 5

Purpose of this assignment :

- Testing students skills on
  - creating, styling & mapping components
  - props

☠ **Assignment steps:**

- part 1: Create the Products component (point: 1)
- part 2: Pass products data from the App.js to the Products component (point: 1)
- part 3: In the Products component map the Product component based on the products data (point: 2)
- part 4: Make all the necessary adjustment (styles, accessing props value etc.) in the Product component (point: 1)
- finally check the project demo and try to match your one as much as possible

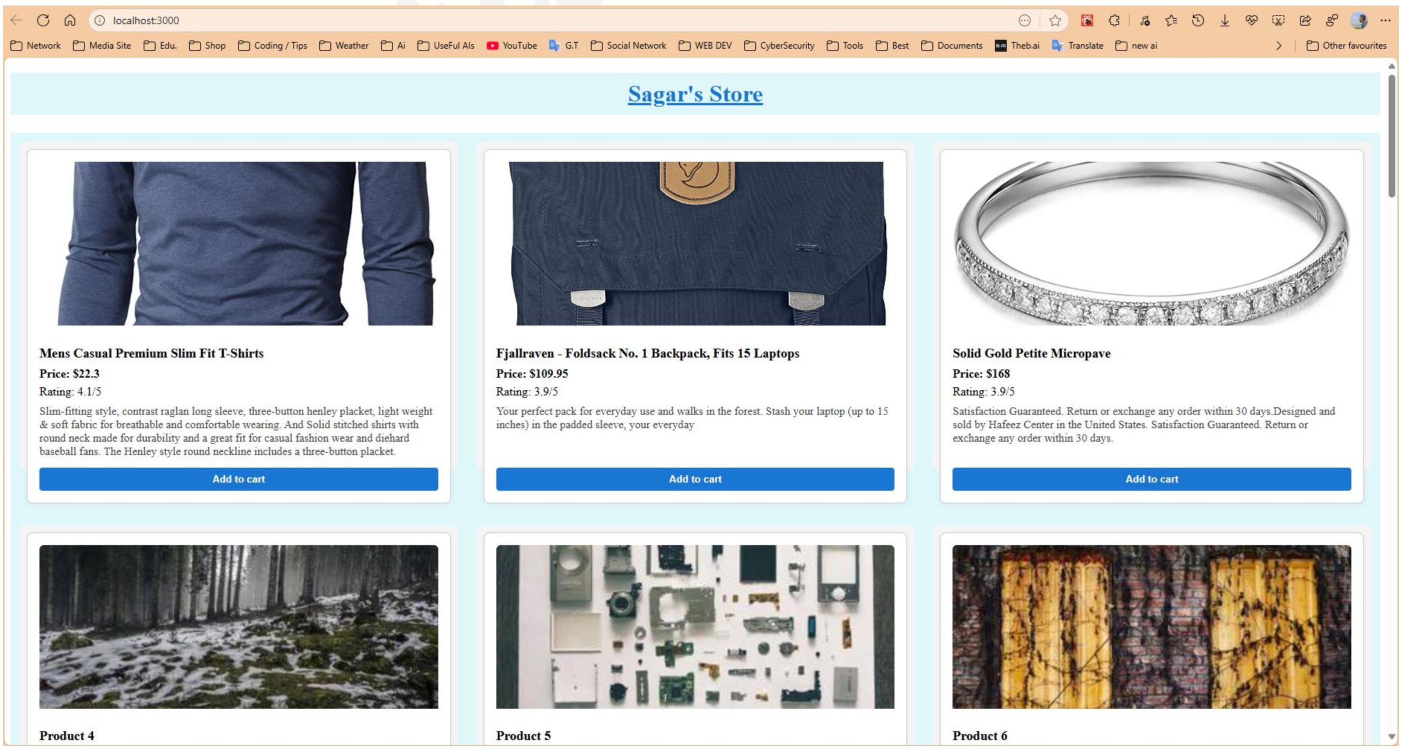
📌 **Final Goal**

Your product listing app should **match the provided demo as closely as possible**.

Demo Link : https://react-assignment-1-product-listing-ap.netlify.app/

Source Code : https://github.com/SagarBiswas-MultiHAT/react-assignment-1-product-listing-app

(Use this to compare your final design and structure.)



Sagar's Store

Mens Casual Premium Slim Fit T-Shirts
Price: $22.3
Rating: 4.1/5
Slim-fitting style, contrast raglan long sleeve, three-button henley placket, light weight & soft fabric for breathable and comfortable wearing. And Solid stitched shirts with round neck made for durability and a great fit for casual fashion wear and diehard baseball fans. The Henley style round neckline includes a three-button placket.
Add to cart

Fjallraven - Foldsack No. 1 Backpack, Fits 15 Laptops
Price: $109.95
Rating: 3.9/5
Your perfect pack for everyday use and walks in the forest. Stash your laptop (up to 15 inches) in the padded sleeve, your everyday
Add to cart

Solid Gold Petite Micropave
Price: $168
Rating: 3.9/5
Satisfaction Guaranteed. Return or exchange any order within 30 days.Designed and sold by Hafeez Center in the United States. Satisfaction Guaranteed. Return or exchange any order within 30 days.
Add to cart

Product 4

Product 5

Product 6

☠ Suggested Folder Structure

→

```
Assignment_1/
  public/
    index.html
  src/
    App.js
    components/
      Products.js
      Product.js
    Photos/
      Product 1/
      Product 2/
      Product 3/
  package.json
  README.md
```

--------------- X ---------------