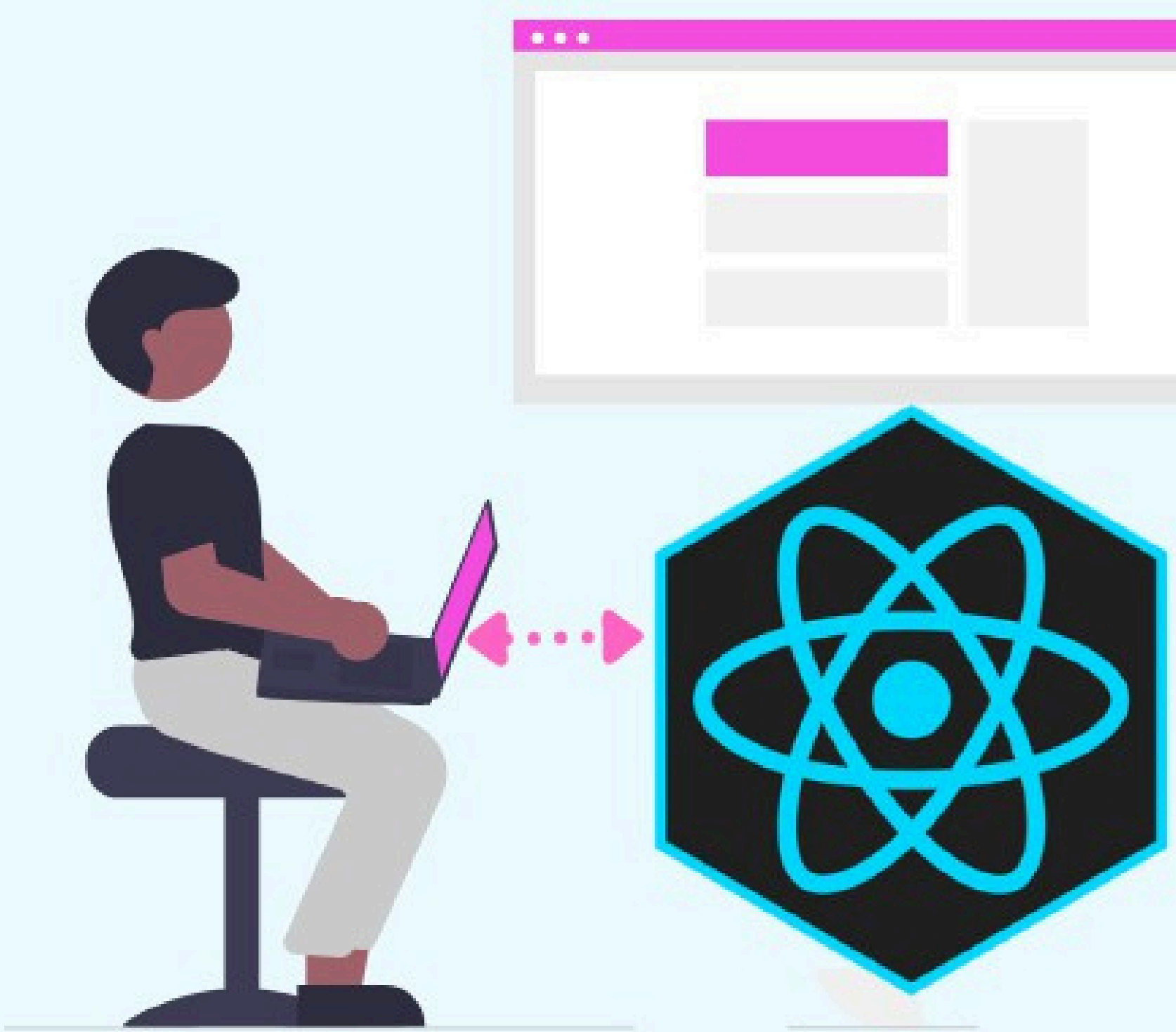
Day 25

Shifting from Theory to Practice (Project Day 1)



@oluwakemi Oluwadahunsi

Introduction

From Day 1 through Day 24, we've journeyed through the core basics and even some intermediate topics in React.js, building a solid understanding of what React is all about.

Now, we could continue learning more topics over the remaining days of this 30-day series, but that’s not how I want us to conclude this journey.

I strongly believe in the "**Learn by Doing**" approach. Many of you who have interacted with me in direct messages know how much I emphasize the importance of **hands-on practice** in coding. It’s a principle I live by, and one that has shaped the way I approach learning and teaching.

So rather than just talking through more theory, I want us to apply everything we’ve covered. It’s time to take all that knowledge and put it into practice.

@oluwakemi Oluwadahunsi

Mood Tracker App

For the next few days, possibly 3 days or more depending on our pace, we'll be building a real, functional "**Mood Tracker**" application using React.js.

This project will serve as a hands-on way to reinforce everything we've learned.

This isn't just about following tutorials; it's about you **actively working** on the app, applying concepts, solving challenges, and gaining real-world experience.

By the end of it, you'll have a project you can not only be proud of but also showcase.

@oluwakemi Oluwadahunsi

Overview of Our App

The Mood Tracker is a React-based web application that empowers users to log and track their daily moods, activities, and gain insights into their emotional journey over time.

This app not only helps users reflect on their moods but also provides personalized suggestions based on their emotional state.

Key Features:

- Mood Logging: Users can log their daily moods and associate specific activities with each mood.
- Trend and Stats Visualization: The app offers insights into mood trends over time with the help of charts and graphs.

@oluwakemi Oluwadahunsi

- Personalized Suggestions: Based on the user's current mood, personalized recommendations are displayed to improve well-being.
- Mood Customization: Users can customize mood options to fit their preferences.
- Mood Persistence: Moods are stored in localStorage, ensuring users' entries are saved even after refreshing or closing the browser.

Tech Stack & Tools:

- React.js: The core framework used to build the app, managing components, and state.
- TailwindCSS: For styling the app with utility-first CSS for fast and responsive designs

@oluwakemi Oluwadahunsi

- Lucide React: A lightweight icon library for consistent and modern icons.
- Recharts: A powerful library for chart visualization to display mood trends and stats.
- Context API: Used for global state management to handle shared data across the app seamlessly.
- LocalStorage: Used to persist mood entries locally on the user's browser

Development Best Practices:

- Reusable Components: The app will use well-structured reusable components to maintain clean and maintainable code.
- Static Components: Certain static sections will be implemented for performance optimization.
- Performance Optimization: The app will adopt best practices like lazy loading, memoization, and effective state management to ensure it performs efficiently.

@oluwakemi Oluwadahunsi

Pre-requisites

With this solid foundation, we'll dive into building the Mood Tracker App step-by-step.

We will get started by:

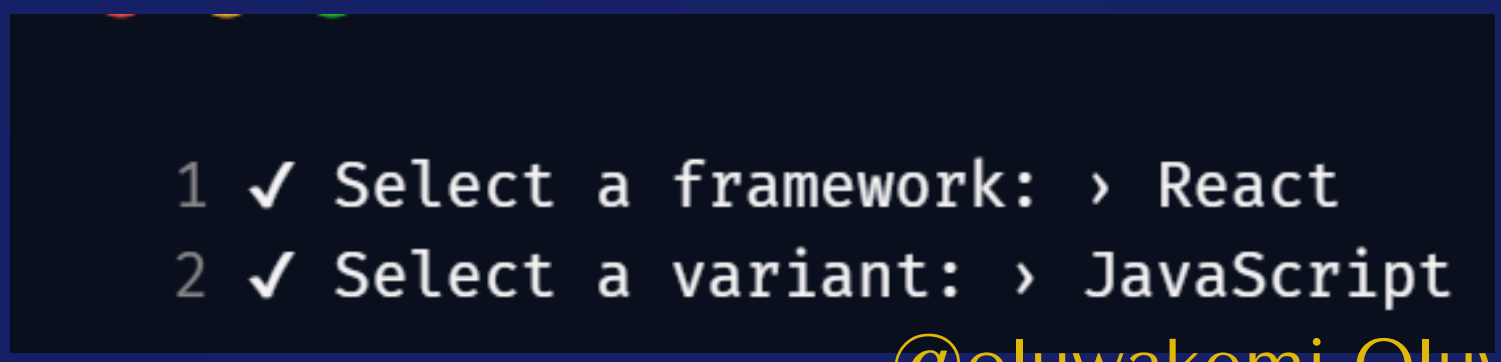
1. Create a React project Template from scratch
2. Install all necessary dependencies

1. Create a React project Template from scratch

In your code editor's terminal, navigate to the directory where you want your project to be created (Refer to Day 2 of this series for a step by step explanation), and type:

"**npm create vite@latest mood-tracker-app**".

You will be prompted for some options, like your preferred type of library and language, please choose appropriately.



Navigate to your newly created folder (mood-tracker-app) and follow the instruction to install (run npm install).

Note: Do not start the development server yet, only run “npm install”.

Clean up the template: Open the project in your code editor and navigate to the “**src**” folder.

Delete everything you have in the following folders as we will be creating ours to avoid conflict of styles and codes:

- App.jsx
- App.css
- index.css
- assets folder
- public folder (optional)

Step 2: install necessary dependencies:

Fpr this project, we will be needing some libraries to work with. Let’s install our dependencies:

- date-fns
- lucide-react
- recharts
- Tailwindcss

In your terminal (make sure you are in the root directory of your project), run “npm install” followed by the dependencies (you can run 3 at a time or one after the other) like this:

“npm install date-fns lucide-react recharts”

To install Tailwind CSS, refer to Day 10 of this series (page 22). This set up has to be working before moving on to the next stage please.

Step 3: Creating and updating components

In the “**src**” folder, create another folder named “**components**”. this is where we will create and save all our components.

First, let’s setup our project body UI in the App.jsx file:

```jsx
function App() {
  return (
    <div className="min-h-screen bg-gradient-to-br from-purple-400 to-indigo-600
     dark:from-gray-600 dark:to-gray-900 transition-colors duration-200 shadow-2xl">

      <div className="max-w-4xl mx-auto px-4 sm:px-6 lg:px-8 py-6 sm:py-8 lg:py-12">

      </div>
    </div>
  );
}
export default App;
```

This is just the basic css styling for the body background and the card for our project itself.

Step 4: Creating the global theme context

In the “**src**” folder, create another folder named “**contexts**”. this is where we will create and save all our context API files. In this contexts folder, create a new file named “**ThemeContext.jsx**” and add this:

```jsx
import { createContext, useState, useEffect, useContext } from "react";

const ThemeContext = createContext();

export const ThemeProvider = ({ children }) => {
  const [darkMode, setDarkMode] = useState(false);

  useEffect(() => {
    const isDarkMode = localStorage.getItem("darkMode") === "true";
    setDarkMode(isDarkMode);
    applyTheme(isDarkMode);
  }, []);

  const toggleDarkMode = () => {
    const newDarkMode = !darkMode;
    setDarkMode(newDarkMode);
    localStorage.setItem("darkMode", newDarkMode);
    applyTheme(newDarkMode);
  };

  const applyTheme = (isDark) => {
    if (isDark) {
      document.documentElement.classList.add("dark");
    } else {
      document.documentElement.classList.remove("dark");
    }
  };

  return (
    <ThemeContext.Provider value={{ darkMode, toggleDarkMode }}>
      {children}
    </ThemeContext.Provider>
  );
};

export const useTheme = () => useContext(ThemeContext);
```

What we have in this context:

- **ThemeContext:** Holds the theme-related data and functions.
- **ThemeProvider:** Provides the darkMode state and toggleDarkMode function to the entire component tree.
- **useTheme():** Custom hook that allows any component to consume the theme context.
- **localStorage:** Used to persist the user's theme preference across page reloads.
- **applyTheme():** Applies the appropriate theme to the <html> element by adding/removing the dark class.

In your tailwind.config.js file, add this:

darkMode: “class”

```
/** @type {import('tailwindcss').Config} */
export default {
  content: ["./index.html", "./src/**/*.{js,jsx,ts,tsx}"],
  darkMode: "class",
  theme: {
    extend: {},
  },
  plugins: [],
};
```

@oluwakemi Oluwadahunsi

The ThemeProvider component uses the ThemeContext.Provider to pass the darkMode state and toggleDarkMode function to any child components in the tree.

To make the theme states available for use in all components, wrap the **ThemeProvider** around your **app** component, I usually recommend to do this in the **main.jsx** file.

```jsx
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import App from "./App.jsx";
import "./index.css";
import { ThemeProvider } from "./contexts/ThemeContext.jsx";

createRoot(document.getElementById("root")).render(
  <StrictMode>
    <ThemeProvider>
      <App />
    </ThemeProvider>
  </StrictMode>
);
```

@oluwakemi Oluwadahunsi

Using this theme states in a component: let's create our Header component so we can see how this theme works:

Header image 1

```jsx
import { useTheme } from "../hooks/ThemeContext";
import { Sun, Moon, Calendar, BarChart2, Settings } from "lucide-react";

function Header({ activeTab, setActiveTab }) {
  const { darkMode, toggleDarkMode } = useTheme();
return (
  <header className="bg-indigo-600 dark:bg-gray-900 p-4 transition-colors duration-200">
    <div className="container mx-auto flex flex-col sm:flex-row justify-between items-center">
      <h1 className="text-2xl font-bold text-white mb-4 sm:mb-0">
        Mood Tracker
      </h1>
      <nav className="flex flex-wrap justify-center sm:justify-end space-x-2 sm:space-x-4">
        <button
          onClick={() => setActiveTab("today")}
          className={`text-white px-3 py-2 rounded-md text-sm font-medium ${
            activeTab === "today"
              ? "bg-indigo-700 dark:bg-gray-700"
              : "hover:bg-indigo-500 dark:hover:bg-gray-700"
          }`}
        >
          Today
        </button>
        <button
          onClick={() => setActiveTab("calendar")}
          className={`text-white px-3 py-2 rounded-md text-sm font-medium flex justify-center items-center ${
            activeTab === "calendar"
              ? "bg-indigo-700 dark:bg-gray-700"
              : "hover:bg-indigo-500 dark:hover:bg-gray-700"
          }`}
        >
          <Calendar size={20} className="inline-block mr-1" />
          <span className="hidden lg:inline ">Calendar</span>
        </button>
```

custom hook from ThemeContext for accessing states

@oluwakemi Oluwadahunsi

Header image 2

```
<button
  onClick={() => setActiveTab("statistics")}
  className={`text-white px-3 py-2 rounded-md text-sm font-medium flex justify-center items-center ${
    activeTab === "statistics"
      ? "bg-indigo-700 dark:bg-gray-700"
      : "hover:bg-indigo-500 dark:hover:bg-gray-700"
  }`}
>
  <BarChart2 size={20} className="inline-block mr-1" />
  <span className="hidden lg:inline ">Statistics</span>
</button>
<button
  onClick={() => setActiveTab("settings")}
  className={`text-white px-3 py-2 rounded-md text-sm font-medium flex justify-center items-center ${
    activeTab === "settings"
      ? "bg-indigo-700 dark:bg-gray-700"
      : "hover:bg-indigo-500 dark:hover:bg-gray-700"
  }`}
>
  <Settings size={20} className="inline-block mr-1" />
  <span className="hidden lg:inline ">Settings</span>
</button>
<button
  onClick={toggleDarkMode}
  className="text-white px-3 py-2 rounded-md text-sm font-medium hover:bg-indigo-500 dark:hover:bg-gray-700"
>
  {darkMode ? <Sun size={20} /> : <Moon size={20} />}
</button>
</nav>
</div>
</header>
);
}

export default Header;
```

the toggleDarkMode function applied when the button is clicked

conditionally render icons based on modes

Do note that all component's responsive views has been handled. These components are responsive up to mobile view.

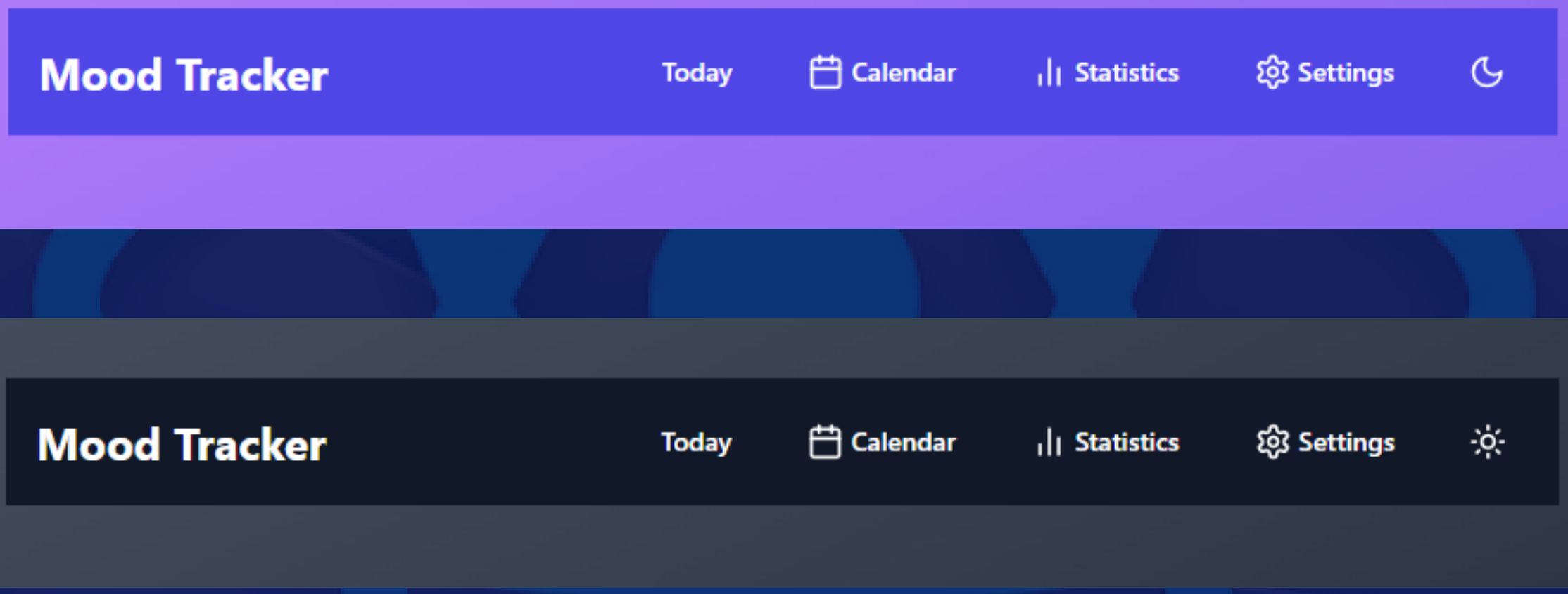
Now, call the Header comport in your App.jsx file.

```
<div className="max-w-4x
  <Header />
</div>
```

@oluwakemi Oluwadahunsi

Run the application to view what we have:
“npm run dev” in your terminal.

You should have the theme icons toggling the dark and light mode on and off, with our beautiful header ready 💃



We will stop here today.

We will continue with other parts of the projects tomorrow. I hope you are looking forward to Project Day 2 (tomorrow) as I am 😊.

@oluwakemi Oluwadahunsi

I hope you found this material useful and helpful.

Remember to:

Like

Save for future reference

&

Share with your network, be helpful to someone 👌

@oluwakemi Oluwadahunsi

Hi There!

Thank you for reading through

Did you enjoy this knowledge?

💼 Follow my LinkedIn page for more work-life balancing and Coding tips.

🌐 LinkedIn: Oluwakemi Oluwadahunsi

kodemaven-portfolio.vercel.app