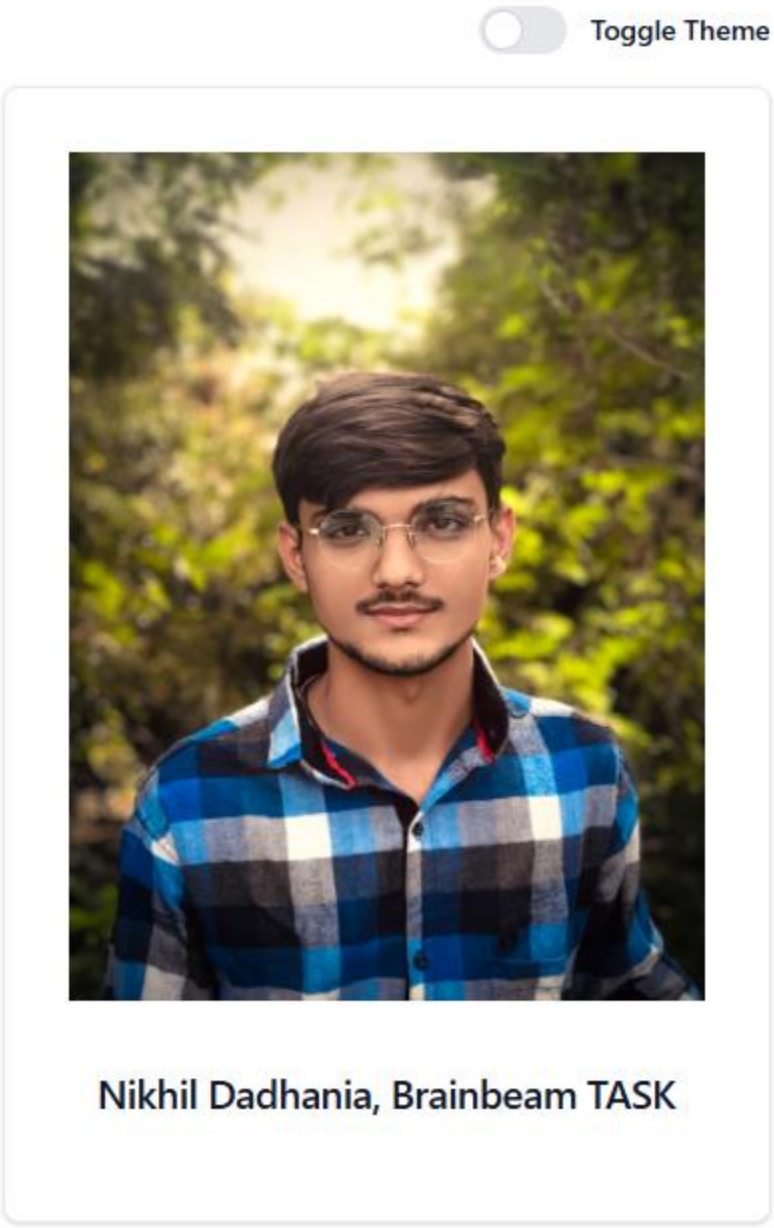
1. **Task Description**

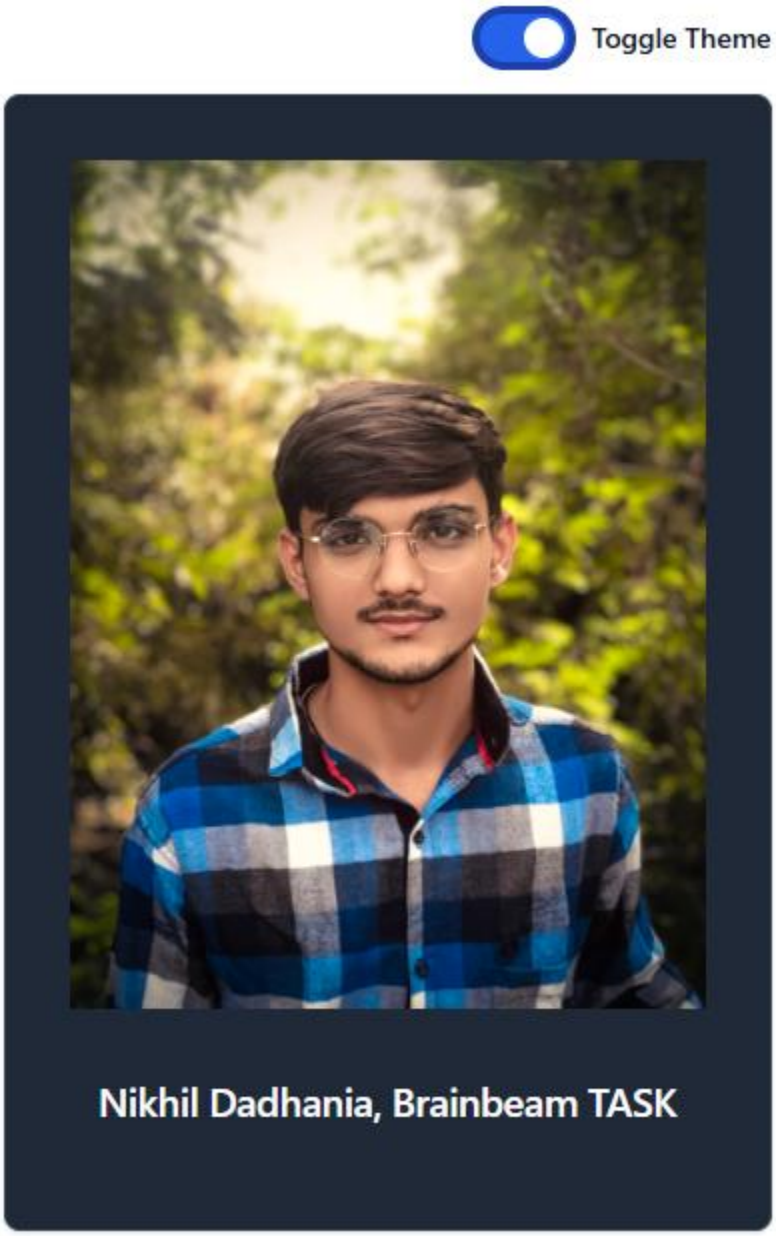
Create a component that uses React context for global state management.

2. **Task Output Screenshot**

Home PAGE –



white one color change to yellow –



3. Widget/Algorithm Used In Task

1. **useState (from React):**
   - Manages the themeMode state, which determines whether the app is in "light" or "dark" mode.
2. **useEffect (from React):**
   - Listens to changes in themeMode and dynamically updates the classList of the html element to apply the correct theme.
3. **ThemeProvider (Custom Context Provider):**
   - Provides the current theme state (themeMode) and functions (lightTheme, darkTheme) to toggle the theme.
   - Makes these values available to child components via React Context.
4. **ThemeBtn (Custom Component):**
   - A button component to toggle the theme.
   - Consumes the ThemeProvider to trigger theme changes.
5. **Card (Custom Component):**
   - A reusable component styled to reflect the current theme.
   - Renders content within the app, inheriting theme-based styles.
6. **CSS Classes:**

- light and dark classes dynamically applied to the html element to control global styles.
- Tailwind-style utility classes like flex, min-h-screen, justify-end, etc., used for layout and responsiveness.

Algorithm Used:

Theme Toggle Algorithm:

- **Task:** Switch the app's theme between "light" and "dark" modes.
- **Steps:**
  1. **State Management:**
     - themeMode is initialized with "light".
     - lightTheme and darkTheme functions update themeMode via setThemeMode.
  2. **Theme Application:**
     - useEffect listens to changes in themeMode.
     - Removes existing theme classes (light and dark) from the html element.

        - Adds the new theme class (light or dark) to the html element.

Context API Algorithm (ThemeProvider):

- **Task:** Share theme state and toggling functions across the app.
- **Steps:**
    1. ThemeProvider wraps the app, providing the themeMode, lightTheme, and darkTheme values via React Context.
    2. Child components (like ThemeBtn) consume these values using the useContext hook to access and modify the theme.

Dynamic Styling Algorithm:

- **Task:** Dynamically apply styles based on the theme.
- **Steps:**
    1. themeMode determines the applied CSS class (light or dark).
    2. Global CSS rules define styles for these classes, which cascade throughout the app.