Experiment 2

Aim: -

Experiment Based on React Hooks (useEffect, useContext, custom hooks)

Theory: -

a) useContext in this Project:

The useContext hook in React allows data to be shared globally across multiple components without passing props manually at every level. In this project, ThemeContext was created to hold the current theme (light, dark, solarized) and functions like changeTheme, resetStats, and getMostUsedTheme. By wrapping the app inside ThemeProvider, all child components can easily access or update the theme using useThemeContext() instead of prop drilling. This simplified theme management and ensured consistency across navigation, buttons, and the overall UI.

Example Code:

```
const ThemeContext = createContext();
export const useThemeContext = () => useContext(ThemeContext);
```

Role in Project:

- Stores and provides the current theme globally.
- Gives access to theme-related functions (changeTheme, stats, history).
- Reduces complexity compared to passing props down multiple levels.

b) useEffect in this Project:

The useEffect hook lets developers perform side effects in React, such as data fetching, subscriptions, or DOM updates.
In this project, useEffect was used for:

1. Loading saved theme data from localStorage on startup.
2. Saving theme, history, and stats whenever they change.
3. Auto theme switching based on time of day using an interval check.

This ensured the theme state persisted across sessions and changed automatically when the time condition matched.

Example Code:

```
useEffect(() => {
  document.body.className = theme;
  localStorage.setItem('app-theme', theme);
}, [theme]);
```

Role in Project:

- Keeps theme data persistent even after refresh.
- Updates the DOM with the current theme.
- Enables extra functionality like auto theme.

c) Custom Hook in this Project:

A custom hook in React is a JavaScript function that starts with "use" and allows developers to extract and reuse component logic.
In this project, the custom hook **useThemeContext** was created, which simply wraps **useContext(ThemeContext).** Instead of importing both **useContext** and **ThemeContext** everywhere, components can just use **useThemeContext()** to access all theme-related data and functions.

Example Code:

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

Role in Project:

- Provides a cleaner, reusable way to access ThemeContext.
- Improves readability by avoiding repetitive boilerplate code.
- Centralizes theme access in a single function.

d) 30% Additional Features

In addition to the three core React concepts, I have implemented four additional features in my project to make it more practical and user-friendly. These features go beyond the basic requirements and demonstrate how React can be used in real-world applications.

1. Theme History Tracking

- This feature keeps a **record of all theme changes** made by the user.
- It allows users (or developers) to see which themes were applied and in what sequence.
- Helps in **tracking behavior patterns** (e.g., how often users switch between light and dark mode).

Code Snippet Example:

```
const [history, setHistory] = useState([]);
useEffect(() => {
  if (theme) {
    setHistory(prev => [...prev, theme]);
  }
}, [theme]);
```

Here, every time the theme state changes, the new theme is added to the history array.

2. Usage Stats

- This feature tracks how many times a theme was applied.
- It can be used to generate simple **analytics** (like counts or charts).
- Example: If the user switched to **Dark Theme** 5 times and **Light Theme** 3 times, it shows those counts.

Code Snippet Example:

```
const [stats, setStats] = useState({ light: 0, dark: 0 });

useEffect(() => {
  if (theme) {
    setStats(prev => ({
      ...prev,
      [theme]: prev[theme] + 1
    }));
  }
}, [theme]);
```

This way, the project can show how often each theme is chosen.

3. Auto Theme Mode

- The app can **automatically switch themes based on time of day**.
- For example:
    - Daytime → **Light Mode**
    - Nighttime → **Dark Mode**
- This creates a **smart system-driven experience** for the user.

Code Snippet Example:

```
useEffect(() => {
  const hour = new Date().getHours();
  if (hour >= 19 || hour < 6) {
    setTheme('dark');
  } else {
    setTheme('light');
  }
}, []);
```

This runs once when the app loads and applies the theme according to system time.

4. Persistent Data with Local Storage

- Normally, when a user refreshes the page, the app state is lost.
- To solve this, I used **localStorage** to **save theme and history** so that the data stays even after reload.
- Makes the app feel more like a real-world application.

Code Snippet Example:

```
useEffect(() => {
  localStorage.setItem('theme', theme);
}, [theme]);

useEffect(() => {
  const savedTheme = localStorage.getItem('theme');
  if (savedTheme) {
    setTheme(savedTheme);
  }
}, []);
```

Here:

- The first useEffect saves the theme whenever it changes.
- The second useEffect restores the saved theme on app load.

CODE: -

a) Theme Context & State Management (ThemeContext.js)

ThemeContext.js > availableThemes

```
import React, { createContext, useContext, useEffect, useState, useRef, useCallback } from 'react';

const ThemeContext = createContext();

const availableThemes = ['light', 'dark', 'solarized'];

export const ThemeProvider = ({ children }) => {
  const [theme, setTheme] = useState('light');
  const [isAnimating, setIsAnimating] = useState(false);
  const [themeHistory, setThemeHistory] = useState([]);
  const [themeStats, setThemeStats] = useState({
    light: 0,
    dark: 0,
    solarized: 0
  });
  const [autoTheme, setAutoTheme] = useState(false);
  const [lastThemeChange, setLastThemeChange] = useState(null);
  const animationTimeoutRef = useRef(null);

  // Load saved data on startup
  useEffect(() => {
    const saved = localStorage.getItem('app-theme');
    const savedHistory = localStorage.getItem('theme-history');
    const savedStats = localStorage.getItem('theme-stats');
    const savedAutoTheme = localStorage.getItem('auto-theme');

    if (saved && availableThemes.includes(saved)) {
      setTheme(saved);
      document.body.className = saved;
    }

    if (savedHistory) {
      setThemeHistory(JSON.parse(savedHistory));
    }

    if (savedStats) {
      setThemeStats(JSON.parse(savedStats));
    }

    if (savedAutoTheme) {
      setAutoTheme(JSON.parse(savedAutoTheme));
    }
  }, []);

  // Save data when it changes
  useEffect(() => {
    document.body.className = theme;
    localStorage.setItem('app-theme', theme);
    localStorage.setItem('theme-history', JSON.stringify(themeHistory));
    localStorage.setItem('theme-stats', JSON.stringify(themeStats));
    localStorage.setItem('auto-theme', JSON.stringify(autoTheme));
  }, [theme, themeHistory, themeStats, autoTheme]);

  // Auto theme based on time
  useEffect(() => {
    if (!autoTheme) return;

    const checkTimeAndSetTheme = () => {
      const hour = new Date().getHours();
      let newTheme = 'light';

      if (hour >= 18 || hour < 6) {
        newTheme = 'dark';
      } else if (hour >= 6 && hour < 12) {
        newTheme = 'solarized';
      }

      if (newTheme !== theme) {
        changeTheme(newTheme);
      }
    };

    checkTimeAndSetTheme();
    const interval = setInterval(checkTimeAndSetTheme, 60000); // Check every minute

    return () => clearInterval(interval);
  }, [autoTheme, theme]);

```

```
  const changeTheme = useCallback((newTheme) => {
    if (availableThemes.includes(newTheme) && !isAnimating) {
      setIsAnimating(true);

      // Add to history
      const newHistory = [
        { theme: newTheme, timestamp: new Date().toISOString() },
        ...themeHistory.slice(0, 9) // Keep only last 10
      ];
      setThemeHistory(newHistory);

      // Update stats
      setThemeStats(prev => ({
        ...prev,
        [newTheme]: prev[newTheme] + 1
      }));

      // Set last change time
      setLastThemeChange(new Date().toISOString());

      // Animated theme change
      if (animationTimeoutRef.current) {
        clearTimeout(animationTimeoutRef.current);
      }

      animationTimeoutRef.current = setTimeout(() => {
        setTheme(newTheme);
        setIsAnimating(false);
      }, 300);
    }
  }, [availableThemes, isAnimating, themeHistory]);

  const resetStats = useCallback(() => {
    setThemeStats({ light: 0, dark: 0, solarized: 0 });
    setThemeHistory([]);
  }, []);

  const getMostUsedTheme = useCallback(() => {
    const entries = Object.entries(themeStats);
    return entries.reduce((a, b) => a[1] > b[1] ? a : b)[0];
  }, [themeStats]);

  return (
    <ThemeContext.Provider value={{
      theme,
      changeTheme,
      availableThemes,
      isAnimating,
      themeHistory,
      themeStats,
      autoTheme,
      setAutoTheme,
      lastThemeChange,
      resetStats,
      getMostUsedTheme
    }}>
      {children}
    </ThemeContext.Provider>
  );
};

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

b) Custom Hook (useTheme.js)

JS useTheme.js > ...

```
1   import { useThemeContext } from './ThemeContext';
2
3   export default function useTheme() {
4     const {
5       theme,
6       changeTheme,
7       availableThemes,
8       isAnimating,
9       themeHistory,
10      themeStats,
11      autoTheme,
12      setAutoTheme,
13      lastThemeChange,
14      resetStats,
15      getMostUsedTheme
16    } = useThemeContext();
17
18    return {
19      theme,
20      changeTheme,
21      availableThemes,
22      isAnimating,
23      themeHistory,
24      themeStats,
25      autoTheme,
26      setAutoTheme,
27      lastThemeChange,
28      resetStats,
29      getMostUsedTheme
30    };
31  }
```

c) Theme Switcher UI (App.js)

App.js > default

```jsx
import React from 'react';
import { ThemeProvider } from './ThemeContext';
import useTheme from './useTheme';
import ThemeStats from './ThemeStats';
import AutoThemeToggle from './AutoThemeToggle';
import './App.css';

function ThemeSwitcher() {
  const { theme, changeTheme, availableThemes, isAnimating } = useTheme();

  return (
    <div className="theme-switcher">
      <h2>Current Theme: {theme}</h2>
      {isAnimating && <div className="animation-indicator"> Changing...</div>}
      <div className="theme-buttons">
        {availableThemes.map((themeName) => (
          <button
            key={themeName}
            onClick={() => changeTheme(themeName)}
            className={`theme-button ${theme === themeName ? 'active' : ''} ${isAnimating ? 'disabled' : ''}`}
            disabled={isAnimating}
          >
            {themeName.charAt(0).toUpperCase() + themeName.slice(1)}
          </button>
        ))}
      </div>
    </div>
  );
}

function App() {
  return (
    <ThemeProvider>
      <div className="app-container">
        <h1>Advanced Theme Switcher</h1>
        <ThemeSwitcher />

        <div className="advanced-features">
          <div className="feature-section">
            <AutoThemeToggle />
          </div>

          <div className="feature-section">
            <ThemeStats />
          </div>
        </div>

        <div className="content">
          <p>Welcome to the Advanced Theme Switcher!</p>
          <p>This app demonstrates advanced React Hooks usage including Context, Custom Hooks, and more!</p>
          <div className="feature-list">
            <h3>Advanced Features:</h3>
            <ul>
              <li> Multiple theme options (Light, Dark, Solarized)</li>
              <li> Theme persistence using localStorage</li>
              <li> Theme statistics and usage tracking</li>
              <li> Auto theme based on time of day</li>
              <li> Theme change history</li>
              <li> Animated theme transitions</li>
              <li> Responsive design</li>
              <li> Advanced React Hooks usage</li>
            </ul>
          </div>
        </div>
      </div>
    </ThemeProvider>
  );
}

export default App;
```

d) Auto Theme Toggle (AutoThemeToggle.js)

AutoThemeToggle.js > AutoThemeToggle > useEffect() callback

```jsx
import React, { useState, useEffect } from 'react';
import useTheme from './useTheme';

function AutoThemeToggle() {
  const { autoTheme, setAutoTheme } = useTheme();
  const [currentTime, setCurrentTime] = useState(new Date());
  const [nextChange, setNextChange] = useState('');

  // Update time every minute
  useEffect(() => {
    const updateTime = () => {
      const now = new Date();
      setCurrentTime(now);

      // Calculate next theme change
      const hour = now.getHours();
      let nextTheme = '';
      let nextHour = 0;

      if (hour < 6) {
        nextTheme = 'light';
        nextHour = 6;
      } else if (hour < 12) {
        nextTheme = 'dark';
        nextHour = 18;
      } else if (hour < 18) {
        nextTheme = 'solarized';
        nextHour = 6;
      } else {
        nextTheme = 'light';
        nextHour = 6;
      }

      const nextChangeTime = new Date();
      nextChangeTime.setHours(nextHour, 0, 0, 0);

      if (nextChangeTime <= now) {
        nextChangeTime.setDate(nextChangeTime.getDate() + 1);
      }

      const diffMs = nextChangeTime - now;
      const diffHours = Math.floor(diffMs / (1000 * 60 * 60));
      const diffMinutes = Math.floor((diffMs % (1000 * 60 * 60)) / (1000 * 60));

      setNextChange(`${diffHours}h ${diffMinutes}m until ${nextTheme}`);
    };

    updateTime();
    const interval = setInterval(updateTime, 60000);
    return () => clearInterval(interval);
  }, []);

  const getCurrentThemeBasedOnTime = () => {
    const hour = currentTime.getHours();
    if (hour >= 18 || hour < 6) return 'dark';
    if (hour >= 6 && hour < 12) return 'solarized';
    return 'light';
  };

  const currentAutoTheme = getCurrentThemeBasedOnTime();

  return (
    <div className="auto-theme-toggle">
      <h3> Auto Theme</h3>

      <div className="auto-theme-info">
        <div className="time-based-theme">
          <span className="info-label">Current Time:</span>
          <span className="info-value">{currentTime.toLocaleTimeString()}</span>
        </div>

        <div className="time-based-theme">
          <span className="info-label">Auto Theme:</span>
          <span className="info-value">{currentAutoTheme}</span>
        </div>

        {autoTheme && (
          <div className="time-based-theme">
            <span className="info-label">Next Change:</span>
            <span className="info-value">{nextChange}</span>
```

```
            </div>
          )}
        </div>

        <div className="auto-theme-schedule">
          <h4>Theme Schedule:</h4>
          <div className="schedule-grid">
            <div className="schedule-item">
              <span className="time-range">6:00 AM - 12:00 PM</span>
              <span className="theme-name">Solarized</span>
            </div>
            <div className="schedule-item">
              <span className="time-range">12:00 PM - 6:00 PM</span>
              <span className="theme-name">Light</span>
            </div>
            <div className="schedule-item">
              <span className="time-range">6:00 PM - 6:00 AM</span>
              <span className="theme-name">Dark</span>
            </div>
          </div>
        </div>

        <div className="auto-theme-control">
          <label className="toggle-label">
            <input
              type="checkbox"
              checked={autoTheme}
              onChange={(e) => setAutoTheme(e.target.checked)}
              className="toggle-input"
            />
            <span className="toggle-slider"></span>
            <span className="toggle-text">
              {autoTheme ? 'Auto Theme: ON' : 'Auto Theme: OFF'}
            </span>
          </label>
        </div>

        <div className="auto-theme-description">
          <p>
            When enabled, the theme will automatically change based on the time of day:
          </p>
          <ul>
            <li> Morning (6 AM - 12 PM): Solarized theme</li>
            <li> Day (12 PM - 6 PM): Light theme</li>
            <li> Night (6 PM - 6 AM): Dark theme</li>
          </ul>
        </div>
      </div>
    );
  }

  export default AutoThemeToggle;
```

e) Theme Statistics (ThemeStats.js)

ThemeStats.js > ThemeStats

```jsx
import React, { useState, useEffect, useMemo } from 'react';
import useTheme from './useTheme';

function ThemeStats() {
  const { themeStats, themeHistory, resetStats, getMostUsedTheme, lastThemeChange } = useTheme();
  const [showDetails, setShowDetails] = useState(false);
  const [timeAgo, setTimeAgo] = useState('');

  // Calculate total theme changes
  const totalChanges = useMemo(() => {
    return Object.values(themeStats).reduce((sum, count) => sum + count, 0);
  }, [themeStats]);

  // Calculate percentage for each theme
  const themePercentages = useMemo(() => {
    if (totalChanges === 0) return {};
    return Object.entries(themeStats).reduce((acc, [theme, count]) => {
      acc[theme] = ((count / totalChanges) * 100).toFixed(1);
      return acc;
    }, {});
  }, [themeStats, totalChanges]);

  // Update time ago every minute
  useEffect(() => {
    const updateTimeAgo = () => {
      if (lastThemeChange) {
        const now = new Date();
        const lastChange = new Date(lastThemeChange);
        const diffInMinutes = Math.floor((now - lastChange) / (1000 * 60));

        if (diffInMinutes < 1) {
          setTimeAgo('Just now');
        } else if (diffInMinutes < 60) {
          setTimeAgo(`${diffInMinutes} minute${diffInMinutes > 1 ? 's' : ''} ago`);
        } else {
          const diffInHours = Math.floor(diffInMinutes / 60);
          setTimeAgo(`${diffInHours} hour${diffInHours > 1 ? 's' : ''} ago`);
        }
      }
    };

    updateTimeAgo();
    const interval = setInterval(updateTimeAgo, 60000);
    return () => clearInterval(interval);
  }, [lastThemeChange]);

  const mostUsedTheme = getMostUsedTheme();

  return (
    <div className="theme-stats">
      <h3> Theme Statistics</h3>

      <div className="stats-summary">
        <div className="stat-item">
          <span className="stat-label">Total Changes:</span>
          <span className="stat-value">{totalChanges}</span>
        </div>

        <div className="stat-item">
          <span className="stat-label">Most Used:</span>
          <span className="stat-value">{mostUsedTheme}</span>
        </div>

        {lastThemeChange && (
          <div className="stat-item">
            <span className="stat-label">Last Change:</span>
            <span className="stat-value">{timeAgo}</span>
          </div>
        )}
      </div>

      <div className="theme-breakdown">
        <h4>Theme Usage Breakdown:</h4>
        {Object.entries(themeStats).map(([themeName, count]) => (
          <div key={themeName} className="theme-stat-row">
            <div className="theme-stat-info">
              <span className="theme-name">{themeName}</span>
              <span className="theme-count">{count} times</span>
            </div>
            <div className="theme-stat-bar">
```

```
function ThemeStats() {
        {Object.entries(themeStats).map(([themeName, count]) => (
            <div className="theme-stat-bar">
              <div
                className="theme-stat-fill"
                style={{
                  width: `${themePercentages[themeName] || 0}%`,
                  backgroundColor: themeName === 'light' ? '#007bff' :
                                   themeName === 'dark' ? '#6c757d' : '#28a745'
                }}
              ></div>
            </div>
            <span className="theme-percentage">{themePercentages[themeName] || 0}%</span>
          </div>
        ))}
      </div>

      <button
        className="reset-stats-btn"
        onClick={resetStats}
      >
        Reset Statistics
      </button>

      <button
        className="toggle-details-btn"
        onClick={() => setShowDetails(!showDetails)}
      >
        {showDetails ? 'Hide History' : 'Show History'}
      </button>

      {showDetails && (
        <div className="theme-history">
          <h4>Recent Theme Changes:</h4>
          {themeHistory.length === 0 ? (
            <p>No theme changes recorded yet.</p>
          ) : (
            <div className="history-list">
              {themeHistory.map((entry, index) => (
                <div key={index} className="history-item">
                  <span className="history-theme">{entry.theme}</span>
                  <span className="history-time">
                    {new Date(entry.timestamp).toLocaleString()}
                  </span>
                </div>
              ))}
            </div>
          )}
        </div>
      )}
    </div>
  );
}

export default ThemeStats;
```

f) Styling (App.css)

```
/* Theme background and text color for each theme */
body.light {
  background-color: #f5f5f5;
  color: #333;
}
body.dark {
  background-color: #1a1a1a;
  color: #ffffff;
}
body.solarized {
  background-color: #fdf6e3;
  color: #586e75;
}

/* Theme button styles for each theme */
body.light .theme-button {
  background-color: #e0e0e0;
  color: #333;
  border: 2px solid #ccc;
}
body.light .theme-button.active {
  background-color: #007bff;
  color: white;
  border-color: #007bff;
}
body.dark .theme-button {
  background-color: #333;
  color: #fff;
  border: 2px solid #555;
}
body.dark .theme-button.active {
  background-color: #007bff;
  color: white;
  border-color: #007bff;
}
body.solarized .theme-button {
  background-color: #eee8d5;
  color: #586e75;
  border: 2px solid #93a1a1;
}
body.solarized .theme-button.active {
  background-color: #268bd2;
  color: white;
  border-color: #268bd2;
}

/* Animation indicator for theme change */
.animation-indicator {
  text-align: center;
  margin-bottom: 1rem;
  font-weight: bold;
  color: #007bff;
  animation: pulse 1s infinite;
}
@keyframes pulse {
  0%, 100% { opacity: 1; }
  50% { opacity: 0.5; }
}

/* Responsive design for mobile */
@media (max-width: 768px) {
  .app-container {
    padding: 1rem;
  }
  .theme-buttons {
    flex-direction: column;
    align-items: center;
  }
  .theme-button {
    width: 200px;
  }
}
```

OUTPUT

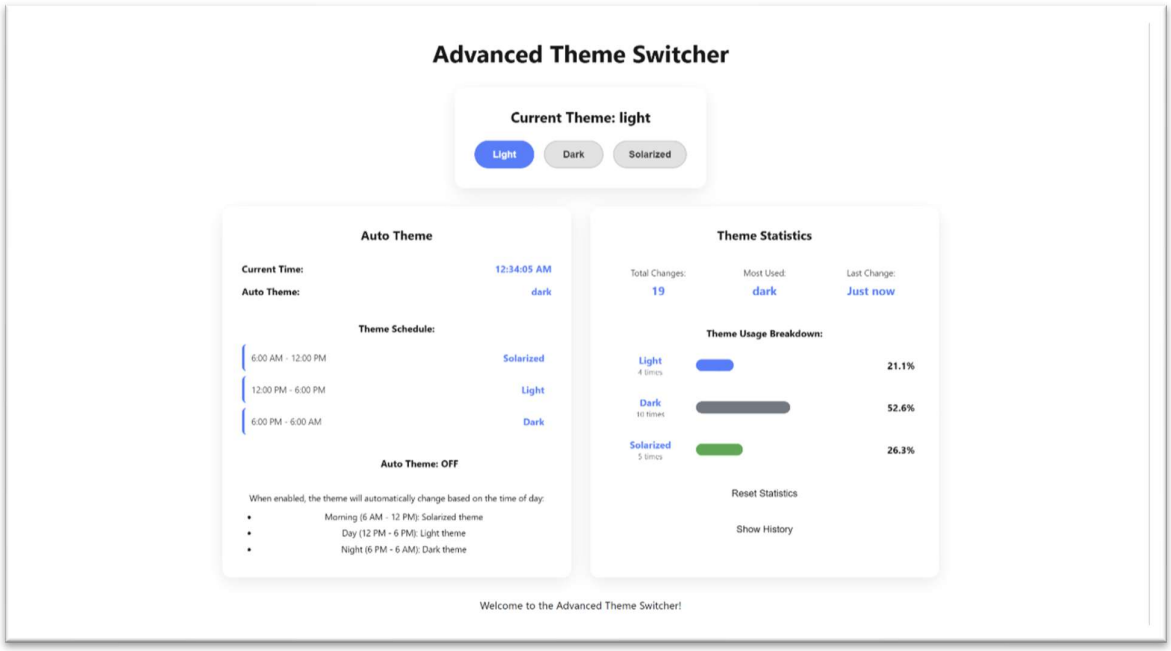

Fig 1

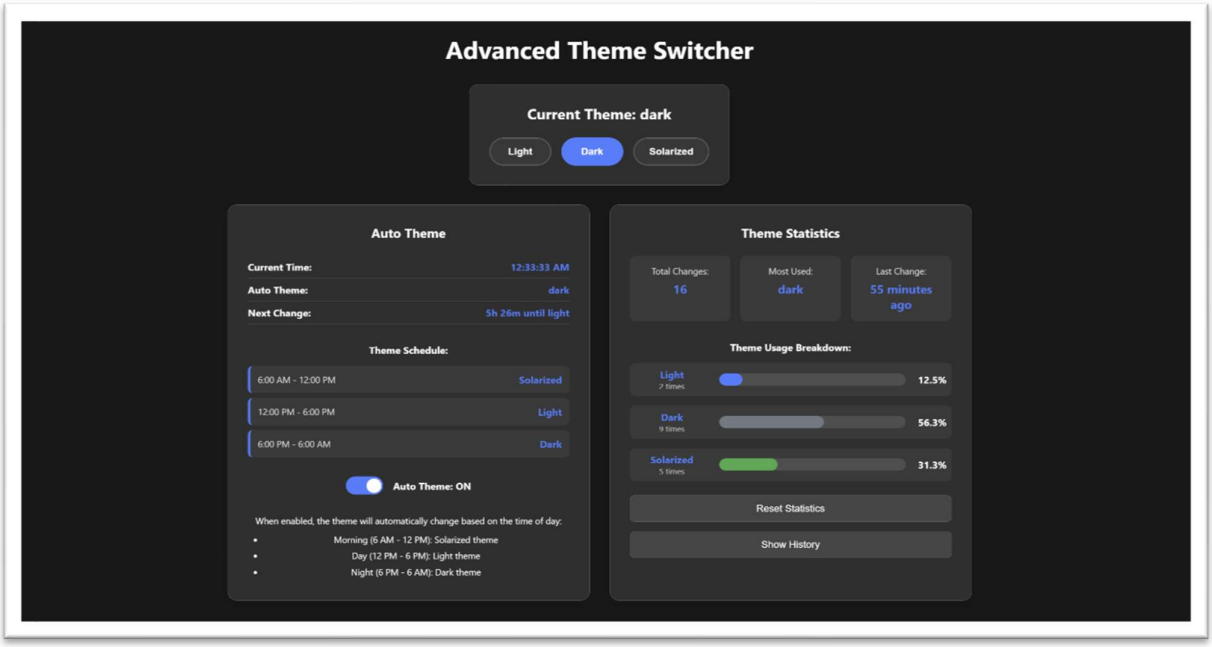

Fig 2

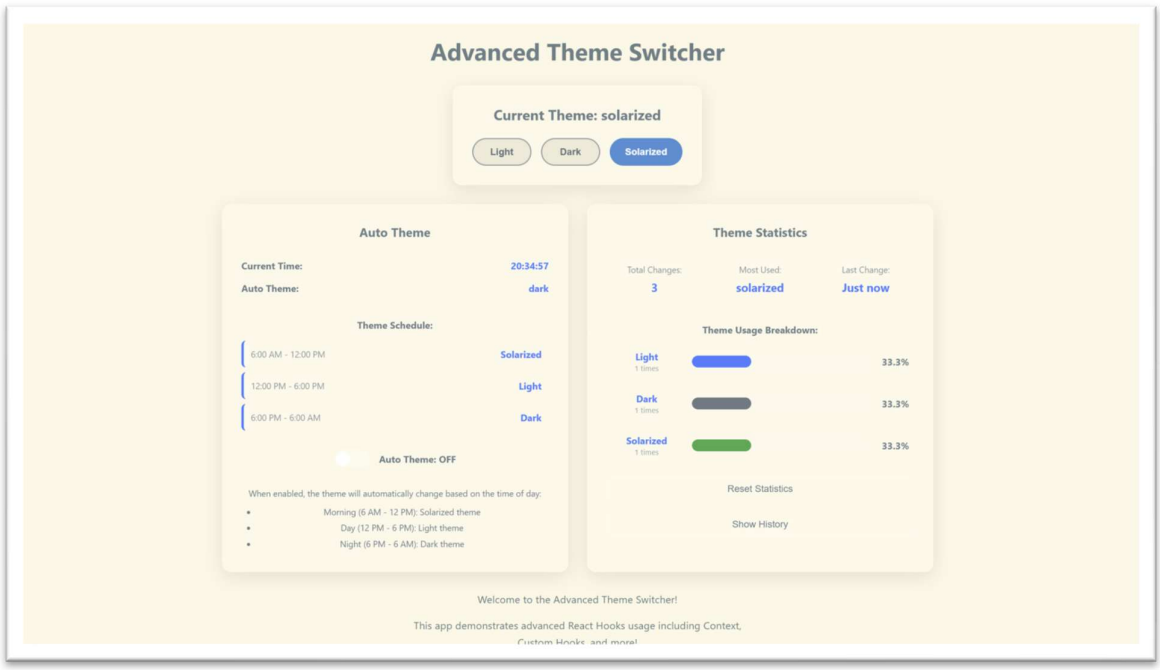
Advanced Theme Switcher
Current Theme: solarized
Light
Dark
Solarized
Auto Theme
Current Time:
20:34:57
Auto Theme:
dark
Theme Schedule:
6:00 AM - 12:00 PM
Solarized
12:00 PM - 6:00 PM
Light
6:00 PM - 6:00 AM
Dark
Auto Theme: OFF
When enabled, the theme will automatically change based on the time of day:
Morning (6 AM - 12 PM): Solarized theme
Day (12 PM - 6 PM): Light theme
Night (6 PM - 6 AM): Dark theme
Theme Statistics
Total Changes:
3
Most Used:
solarized
Last Change:
Just now
Theme Usage Breakdown:
Light
1 times
33.3%
Dark
1 times
33.3%
Solarized
1 times
33.3%
Reset Statistics
Show History
Welcome to the Advanced Theme Switcher!
This app demonstrates advanced React Hooks usage including Context,

Fig 3

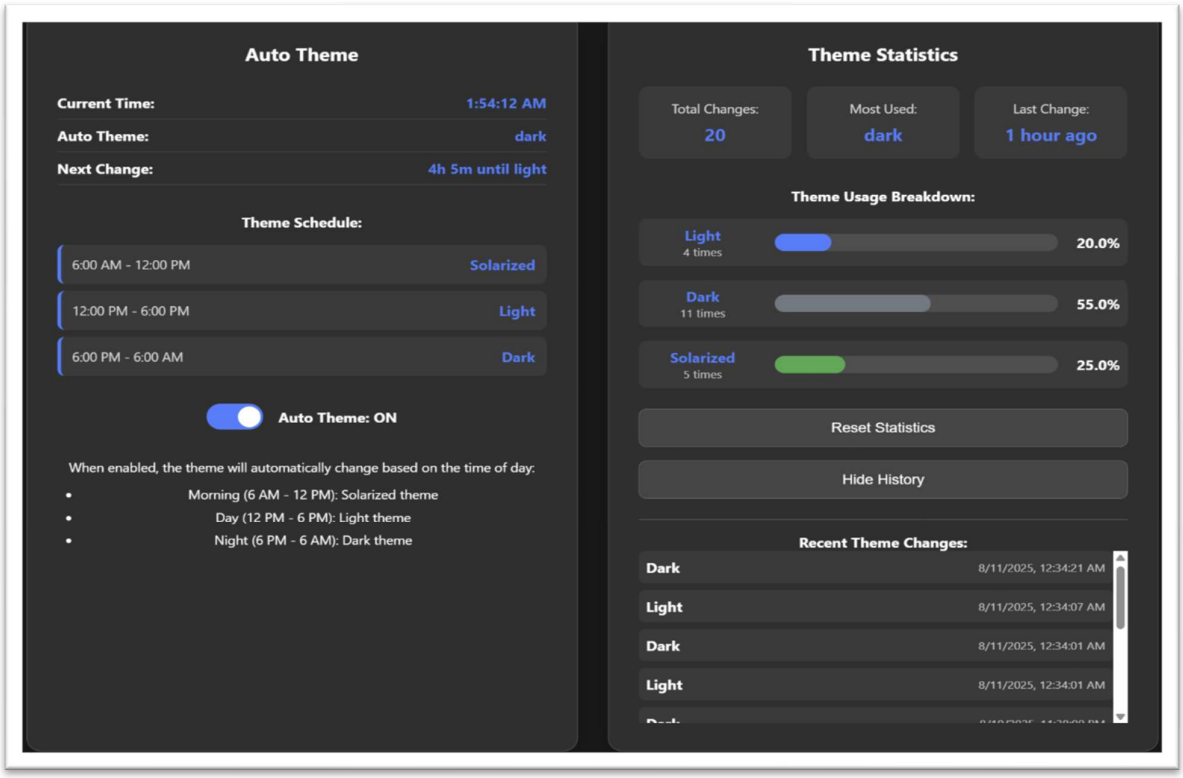
Auto Theme
Current Time:
1:54:12 AM
Auto Theme:
dark
Next Change:
4h 5m until light
Theme Schedule:
6:00 AM - 12:00 PM
Solarized
12:00 PM - 6:00 PM
Light
6:00 PM - 6:00 AM
Dark
Auto Theme: ON
When enabled, the theme will automatically change based on the time of day:
Morning (6 AM - 12 PM): Solarized theme
Day (12 PM - 6 PM): Light theme
Night (6 PM - 6 AM): Dark theme
Theme Statistics
Total Changes:
20
Most Used:
dark
Last Change:
1 hour ago
Theme Usage Breakdown:
Light
4 times
20.0%
Dark
11 times
55.0%
Solarized
5 times
25.0%
Reset Statistics
Hide History
Recent Theme Changes:
Dark
8/11/2025, 12:34:21 AM
Light
8/11/2025, 12:34:07 AM
Dark
8/11/2025, 12:34:01 AM
Light
8/11/2025, 12:34:01 AM

Fig 4

Conclusion

The project demonstrates how React hooks can be effectively used to build a dynamic and user-friendly theme management application. Core hooks like useState, useEffect, and useContext were applied to manage state, handle side effects, and enable seamless theme switching, while a custom hook was introduced to improve modularity and reusability. In addition to these, features such as theme history tracking, auto theme mode, usage statistics, and persistent storage with localStorage were implemented to enhance the overall functionality and user experience. Together, these elements highlight the flexibility of React and its capability to create interactive, scalable, and customizable web applications.