DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING

CU CHANDIGARH UNIVERSITY Discover. Learn. Empower.

Report File FULL STACK

Student Name: Palash Mathur

UID: 23BAI70673

Branch: BE-AIT-CSE

Section/Group: 23AIT-KRG-G2

Semester: 6th

Subject Code: 23CSP-339

Subject Name: Full Stack

Date of submission: 7th Feb 2026

Aim:

To optimize the performance of the EcoTrack React application using memoization techniques and code splitting, and to enhance the user interface using enterprise-grade Material UI components.

Objectives:

After completing this experiment, the student will be able to:

1. Understand the causes of unnecessary re-renders in React applications
2. Optimize React components using React.memo to prevent avoidable re-renders
3. Apply useMemo to efficiently compute derived data and avoid redundant calculations
4. Use useCallback to memoize event handler functions and improve component performance
5. Implement lazy loading of components and routes using React.lazy and Suspense
6. Reduce initial bundle size and improve application load performance through code splitting
7. Enhance the visual appearance and usability of the EcoTrack application using Material UI components
8. Design a clean, consistent, and responsive user interface using Material UI layouts and typography

Hardware Requirements:

- Processor: Intel i5/Ryzen 5 or higher
- RAM: 8GB minimum
- Display: 1920x1080 resolution Software Requirements:
- Node.js v18+
- React.js v18+
- VS Code with ES7+ extensions
- JSON Server (for mock API)

Code Implementation:

App.jsx

```jsx
import { Route, Routes } from "react-router-dom";
import { Suspense, lazy } from "react";
import Header from "./components/Header";
import ProtectedRoute from "./routes/ProtectedRoute";

/* ========= Lazy Loaded Pages (Code Splitting) ========= */
const PerformanceDemo = lazy(() => import("./pages/PerformanceDemo"));
const Login = lazy(() => import("./pages/Login"));
const DashboardLayout = lazy(() => import("./pages/DashboardLayout"));
const DashboardSummary = lazy(() => import("./pages/DashboardSummary"));
const DashboardAnalytics = lazy(() => import("./pages/DashboardAnalytics"));
const DashboardSettings = lazy(() => import("./pages/DashboardSettings"));
const Logs = lazy(() => import("./pages/Logs"));

function App() {
  return (
    <>
      {/* Global Header */}
      <Header />

      {/* Suspense handles lazy loading */}
      <Suspense
        fallback={
          <div
            style={{
              height: "80vh",
              display: "flex",
              justifyContent: "center",
              alignItems: "center",
              fontSize: "22px",
              fontWeight: "600",
            }}
          >
            Loading Page...
          </div>
        }
      >
```

```jsx
        <Routes>

          {/* Performance Demo */}
          <Route
            path="/performance"
            element={
              <ProtectedRoute>
                <PerformanceDemo />
              </ProtectedRoute>
            }
          />

          {/* Login */}
          <Route path="/Login" element={<Login />} />

          {/* Dashboard Layout Routes */}
          <Route
            path="/"
            element={
              <ProtectedRoute>
                <DashboardLayout />
              </ProtectedRoute>
            }
          >
            <Route index element={<DashboardSummary />} />
            <Route path="summary" element={<DashboardSummary />} />
            <Route path="analytics" element={<DashboardAnalytics />} />
            <Route path="settings" element={<DashboardSettings />} />
          </Route>

          {/* Logs Page */}
          <Route
            path="/logs"
            element={
              <ProtectedRoute>
                <Logs />
              </ProtectedRoute>
            }
          />

        </Routes>
      </Suspense>
    </>
  );
}

export default App;
```

logsSlice.jsx

```
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit";
 export const fetchLogs = createAsyncThunk(
  "logs/fetchLogs",   async () => {     await
new Promise((resolve) =>



     setTimeout(resolve, 1000)
   );      return [
       { id: 1, activity: "Car Travel", carbon: 4 },
       { id: 2, activity: "Electricity Usage", carbon: 6 },
       { id: 3, activity: "Cycling", carbon: 0 },
       { id: 4, activity: "Public Transport", carbon: 12 },
       { id: 5, activity: "Meat Consumption", carbon: 5 },
       { id: 6, activity: "Plant-based Meal", carbon: 2 },
       { id: 7, activity: "Air Travel", carbon: 1 },
   ];
  }
);  const logSlice  = createSlice({   name:
"logs",   initialState: {     data: [],     status:
"idle",     error: null,
  },   reducers: {},     extraReducers: (builder) => {
builder.addCase(fetchLogs.pending, (state) => {             state.status =
"loading";         })
       .addCase(fetchLogs.fulfilled, (state, action) => {             state.status =
"succeeded";             state.data = action.payload;
       })
       .addCase(fetchLogs.rejected, (state, action) => {             state.status = "failed";
          state.error = action.error.message;
       });
   },
});  export default logSlice.reducer;
```

store.jsx

```
import { configureStore } from "@reduxjs/toolkit"; import logsReducer from
"./logsSlice";
 const store = configureStore({   reducer:
{
   logs: logsReducer,
  },
});  export default store;
```

StatCard.jsx-

```
import { Card, CardContent, Typography, Stack } from "@mui/material";

const StatCard = ({ title, value, unit }) => {
  return (
    <Card
      sx={{
        borderRadius: 6,
        boxShadow: 30,
        px: 2,
        py: 2,
        margin: 2,
        minHeight: 120,
        display: "flex",
        alignItems: "center",
        backgroundColor: "#6fffca"
      }}
    >
      <CardContent sx={{ p: 0 }}>
        <Stack spacing={1.5}>

          {/* Small label */}
          <Typography
            variant="caption"
            sx={{
              textTransform: "uppercase",
              letterSpacing: 2.2,
              color: "black",
              fontWeight: 800
```

```
            }}
          >
            {title}
          </Typography>

          {/* Main number */}
          <Typography
            variant="h4"
            sx={{
              fontWeight: 500,
              lineHeight: 1
            }}
          >
            {value}
          </Typography>

          {/* Unit */}
          <Typography
            variant="body2"
            sx={{
              color: "text.warning",
              fontWeight: 800
            }}
          >
            {unit}
          </Typography>

        </Stack>
      </CardContent>
    </Card>
  );
};

export default StatCard;
```

EmmisionCard.jsx-

```
import React from "react";

const EmissionCard = React.memo(({ title, value }) => {
  console.log("Rendering:", title);

  return (
    <div className="card">
      <h3>{title}</h3>
      <p>{value}</p>
    </div>
  );
});

export default EmissionCard;
```

UseEmissionStats-

```
import { useMemo } from "react";

export default function useEmissionStats(data) {
  return useMemo(() => {
    console.log("Calculating emissions...");

    let total = 0;
    data.forEach(item => total += item.emission);

    const avg = data.length ? total / data.length : 0;

    return { total, avg };
  }, [data]);
}
```

Logs.jsx-

```
import { useSelector, useDispatch } from "react-redux";
import { fetchLogs } from "../logsSlice";
import { useEffect, useCallback, useMemo } from "react";
import {
  Table, TableBody, TableCell,
  TableContainer, TableHead,
  TableRow, Paper, Button
} from "@mui/material";
import { Skeleton, Stack } from "@mui/material";


const Logs = () => {
  const dispatch = useDispatch();
  const { data, status, error } = useSelector((state) => state.logs);

  // 🔥 MEMOIZED CALCULATION
  const xyz = useMemo(() => {
    console.log("Calculating total carbon...");
    return data.reduce((total, log) => total + log.carbon, 0);
  }, [data]);

  useEffect(() => {
    if (status === "idle") {
      dispatch(fetchLogs());
    }
  }, [status, dispatch]);

  const handlefetch = useCallback(() => {
    dispatch(fetchLogs());
  }, [dispatch]);

  if (status === "loading") {
  return (
```

```
    <Stack spacing={2} padding={2}>
      <Skeleton variant="rectangular" height={50} />
      <Skeleton variant="rectangular" height={50} />
      <Skeleton variant="rectangular" height={50} />
      <Skeleton variant="rectangular" height={50} />
    </Stack>
  );
}
  if (status === "failed") return <p>Error: {error}</p>;

  return (
  <div style={{ padding: "1rem", backgroundColor: "black" }}>
    <h2 style={{color: "white"}}>Daily Logs (Redux)</h2>

    <TableContainer component={Paper} style={{backgroundColor: "#6fffca"}}>
      <Table>
        <TableHead>
          <TableRow>
            <TableCell><b>Activity</b></TableCell>
            <TableCell><b>Carbon (kg CO₂)</b></TableCell>
          </TableRow>
        </TableHead>

        <TableBody>
          {data.map((log) => (
            <TableRow key={log.id}>
              <TableCell>{log.activity}</TableCell>
              <TableCell
                sx={{
                  color: log.carbon > 4 ? "error.main" : "success.main",
                  fontWeight: 600
                }}
              >
                {log.carbon}
              </TableCell>
            </TableRow>
          ))}
        </TableBody>
      </Table>
    </TableContainer>

    <Button
      variant="contained"
      color="primary"
      sx={{ mt: 2 }}
      onClick={handlefetch}
    >
      Refresh ↻
    </Button>

    <h3 style={{ marginTop: "15px", color: "white"}}>
      Total Carbon Emission: {xyz} kg CO₂
    </h3>
  </div>
```

```
);

};

export default Logs;
```

PerformanceDemo-

```
import React, { useState, useMemo, useCallback } from "react";
import CounterChild from "../components/CounterChild";

function expensiveCalculation(num) {
  console.log("🟡 Expensive calculation running...");
  let result = 0;
  for (let i = 0; i < 1_000_000_000; i++) {
    result += num;
  }
  return result;
}

const PerformanceDemo = () => {
  const [count, setCount] = useState(0);
  const [dark, setDark] = useState(false);

  // 🔥 useMemo prevents recalculation unless count changes
  const total = useMemo(() => {
    return expensiveCalculation(count);
  }, [count]);

  // 🔥 useCallback prevents child re-render
  const handleIncrement = useCallback(() => {
    setCount((c) => c + 1);
  }, []);

  return (
    <div style={{ padding: "20px", color: dark ? "white" : "black", background: dark ? "#111"
: "#fff" }}>
      <h2>React Performance Optimization Demo</h2>

      <button onClick={() => setDark(!dark)}>Toggle Theme</button>
      <p>Theme: {dark ? "Dark" : "Light"}</p>

      <CounterChild onIncrement={handleIncrement} total={total} />
    </div>
  );
};

export default PerformanceDemo;
```

Expected Output:

- Learn’t about reduxjs.
- Learn’t about its implications.
- Learn’t how to implement the redux.