```
import React, { useState, useEffect, useRef } from "react";
import Paper from "@mui/material/Paper";
import TableContainer from "@mui/material/TableContainer";
import Table from "@mui/material/Table";
import TableBody from "@mui/material/TableBody";
import TableCell from "@mui/material/TableCell";
import TableHead from "@mui/material/TableHead";
import TablePagination from "@mui/material/TablePagination";
import TableRow from "@mui/material/TableRow";
import Grid from "@mui/material/Grid";
import initializeFirebase from "./firebase/firebase";
import { ref, get } from "firebase/database";
import Chart from "chart.js/auto";
import { useTheme } from "@emotion/react";
import { tokens } from "../theme";

export default function DataHistory() {
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);
  const [rows, setRows] = useState([]);
  const [page, setPage] = useState(0);
  const [rowsPerPage, setRowsPerPage] = useState(8);
  const chartRefWeek = useRef(null);
  const chartRefMonth = useRef(null);
  const [cloggingData, setCloggingData] = useState([]);

  function formatTimestamp(timestamp) {
    const month = parseInt(timestamp.substring(0, 2)) - 1;
    const day = parseInt(timestamp.substring(2, 4));
    const year = parseInt(timestamp.substring(4, 8));
    const hours = parseInt(timestamp.substring(9, 11));
    const minutes = parseInt(timestamp.substring(11, 13));
    const seconds = parseInt(timestamp.substring(13, 15));

    const date = new Date(year, month, day, hours, minutes, seconds);

    if (isNaN(date.getTime())) {
      return "Invalid Timestamp";
    }

    const dateString = date.toLocaleDateString();
    const timeString = date.toLocaleTimeString();
    return `${dateString}, ${timeString}`;
  }

  useEffect(() => {
```

```
const fetchDataFromFirebase = async () => {
  try {
    const database = initializeFirebase();
    const paramPath = "/GutterLocations";
    const paramRef = ref(database, paramPath);
    const snapshot = await get(paramRef);
    const data = snapshot.val();

    if (data) {
      const allTimestamps = [];

      const cloggingEvents = {
        true: [],
        false: [],
      };

      Object.entries(data).forEach(([deviceId, deviceData]) => {
        const { isClogged, clogHistory: originalClogHistory } = deviceData;

        if (isClogged) {
          Object.entries(isClogged).forEach(([timestamp, status]) => {
            allTimestamps.push({ timestamp, status });
            if (status) {
              cloggingEvents.true.push({ timestamp, status });
            } else {
              cloggingEvents.false.push({ timestamp, status });
            }
          });
        }

        if (originalClogHistory) {
          originalClogHistory.forEach((entry) => {
            allTimestamps.push(entry);
            if (entry.status) {
              cloggingEvents.true.push(entry);
            } else {
              cloggingEvents.false.push(entry);
            }
          });
        }
      });

      allTimestamps.sort((a, b) => {
        return parseInt(b.timestamp) - parseInt(a.timestamp);
      });
      setCloggingData(allTimestamps);

      const gutterLocations = Object.entries(data).map(
        ([deviceId, deviceData]) => {
          const {
            name,
            address,
            isClogged,
            clogHistory: originalClogHistory,
          } = deviceData;

          let clogHistory = [];
          if (isClogged) {
            clogHistory = Object.entries(isClogged).map(
              ([timestamp, status]) => ({
                timestamp,
                status: status ? "Clogged" : "Cleared",
              }),
            );
          } else if (originalClogHistory) {
            clogHistory = originalClogHistory.map((entry) => ({
              timestamp: entry.timestamp,
              status: entry.status ? "Clogged" : "Cleared",
            }));
          }

          return {
            name,
            address,
            clogHistory,
          };
        },
      );

      setRows(gutterLocations);

      drawChart(cloggingEvents, "week");
      drawChart(cloggingEvents, "month");
    } else {
      console.log("No data available under GutterLocations.");
    }
  } catch (error) {
    console.error("Error fetching data from Firebase:", error);
  }
};
```

```
  fetchDataFromFirebase();
}, []);

const drawChart = (cloggingEvents, type) => {
  if (!cloggingEvents || !cloggingEvents.true || !cloggingEvents.false) {
    console.error("cloggingEvents or its properties are undefined");
    return;
  }

  const ctx = document.getElementById(`clogging-chart-${type}`);
  const currentDate = new Date();
  const currentYear = currentDate.getFullYear();
  let labels;
  let clogged;
  let unclogged;

  if (type === "week") {
    labels = ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"];
    clogged = Array.from({ length: 7 }, () => 0);
    unclogged = Array.from({ length: 7 }, () => 0);

    const currentWeekStart = new Date(
      currentDate.getFullYear(),
      currentDate.getMonth(),
      currentDate.getDate() - currentDate.getDay(),
    );

    const currentWeekEnd = new Date(
      currentDate.getFullYear(),
      currentDate.getMonth(),
      currentDate.getDate() + (6 - currentDate.getDay()),
    );

    const filteredCloggedEvents = cloggingEvents.true.filter((event) => {
      const eventDate = new Date(
        parseInt(event.timestamp.substring(4, 8)),
        parseInt(event.timestamp.substring(0, 2)) - 1,
        parseInt(event.timestamp.substring(2, 4)),
      );
      return (
        eventDate >= currentWeekStart &&
        eventDate <= currentWeekEnd &&
        eventDate.getFullYear() === currentYear
      );
    });
    const filteredUncloggedEvents = cloggingEvents.false.filter((event) => {
      const eventDate = new Date(
        parseInt(event.timestamp.substring(4, 8)),
        parseInt(event.timestamp.substring(0, 2)) - 1,
        parseInt(event.timestamp.substring(2, 4)),
      );
      return (
        eventDate >= currentWeekStart &&
        eventDate <= currentWeekEnd &&
        eventDate.getFullYear() === currentYear
      );
    });

    filteredCloggedEvents.forEach((event) => {
      const dayOfWeek = new Date(
        parseInt(event.timestamp.substring(4, 8)),
        parseInt(event.timestamp.substring(0, 2)) - 1,
        parseInt(event.timestamp.substring(2, 4)),
      ).getDay();
      clogged[dayOfWeek] += 1;
    });

    filteredUncloggedEvents.forEach((event) => {
      const dayOfWeek = new Date(
        parseInt(event.timestamp.substring(4, 8)),
        parseInt(event.timestamp.substring(0, 2)) - 1,
        parseInt(event.timestamp.substring(2, 4)),
      ).getDay();
      unclogged[dayOfWeek] += 1;
    });
  } else if (type === "month") {
    labels = [
      "Jan",
      "Feb",
      "Mar",
      "Apr",
      "May",
      "Jun",
      "Jul",
      "Aug",
      "Sep",
      "Oct",
      "Nov",
      "Dec",
    ];
    clogged = Array.from({ length: 12 }, (_, i) => 0);
    unclogged = Array.from({ length: 12 }, (_, i) => 0);

    const filteredCloggedEvents = cloggingEvents.true.filter((event) => {
      const eventYear = parseInt(event.timestamp.substring(4, 8));
      return eventYear === currentYear;
    });

    const filteredUncloggedEvents = cloggingEvents.false.filter((event) => {
      const eventYear = parseInt(event.timestamp.substring(4, 8));
      return eventYear === currentYear;
    });

    filteredCloggedEvents.forEach((event) => {
      const month = parseInt(event.timestamp.substring(0, 2)) - 1;
      clogged[month] += 1;
    });

    filteredUncloggedEvents.forEach((event) => {
      const month = parseInt(event.timestamp.substring(0, 2)) - 1;
      unclogged[month] += 1;
    });
  }

  const datasets = [
    {
      label: `Clogged`,
      data: clogged,
      borderColor: "rgba(255, 60, 60, 0.86)",
      backgroundColor: "rgba(255, 222, 222, 0.71)",
      borderWidth: 1,
      fill: true,
    },
    {
      label: `Unclogged`,
      data: unclogged,
      borderColor: "rgba(50, 168, 255, 0.71)",
      backgroundColor: "rgba(186, 225, 255, 0.71)",
      borderWidth: 1,
      fill: true,
    },
  ];

  const options = {
    responsive: true,
    maintainAspectRatio: false,
    scales: {
      x: {
        grid: {
          display: false,
        },
      },
      y: {
        beginAtZero: true,
        ticks: {
          stepSize: 1,
          precision: 0,
        },
      },
    },
    plugins: {
      title: {
        display: true,
        text:
          type === "week"
            ? "Clogging Frequency per Week"
            : "Clogging Frequency per Month",
        font: {
          size: 13,
        },
      },
    },
  };

  const chartRef = type === "week" ? chartRefWeek : chartRefMonth;

  if (chartRef.current) {
    chartRef.current.destroy();
  }

  chartRef.current = new Chart(ctx, {
    type: "line",
    data: {
      labels: labels,
      datasets: datasets,
    },
    options: options,
  });
};

const handleChangePage = (event, newPage) => {
  setPage(newPage);
};
```

```
const handleChangeRowsPerPage = (event) => {
  setRowsPerPage(+event.target.value);
  setPage(0);
};

return (
  <Grid container spacing={2}>
    <Grid item xs={6}>
      <Paper>
        <TableContainer>
          <Table stickyHeader aria-label="clog-history-table">
            <TableHead>
              <TableRow>
                <TableCell align="center">Timestamp</TableCell>
                <TableCell align="center">Name</TableCell>
                <TableCell align="center">Address</TableCell>
                <TableCell align="center">Overflow Status</TableCell>
              </TableRow>
            </TableHead>
            <TableBody>
              {rows
                .map((row) =>
                  row.clogHistory.map((entry, idx) => ({
                    timestamp: entry.timestamp,
                    name: row.name,
                    address: row.address,
                    status: entry.status,
                  })),
                )
                .flat()
                .sort((a, b) => {
                  const timestampTo24HourFormat = (timestamp) => {
                    const hour = parseInt(timestamp.substring(9, 11));
                    const isPM = timestamp.substring(20) === "PM";
                    return isPM ? hour + 12 : hour;
                  };

                  const dateA = parseInt(a.timestamp.substring(0, 8));
                  const timeA = timestampTo24HourFormat(a.timestamp);

                  const dateB = parseInt(b.timestamp.substring(0, 8));
                  const timeB = timestampTo24HourFormat(b.timestamp);

                  if (dateA !== dateB) {
                    return dateB - dateA;
                  } else {
                    return timeB - timeA;
                  }
                })
                .slice(page * rowsPerPage, page * rowsPerPage + rowsPerPage)
                .map((entry, index) => (
                  <TableRow key={index}>
                    <TableCell align="center">
                      {formatTimestamp(entry.timestamp)}
                    </TableCell>
                    <TableCell align="center">{entry.name}</TableCell>
                    <TableCell align="center">{entry.address}</TableCell>
                    <TableCell align="center">{entry.status}</TableCell>
                  </TableRow>
                ))}
            </TableBody>
          </Table>
        </TableContainer>
        <TablePagination
          rowsPerPageOptions={[8]}
          component="div"
          count={cloggingData.length}
          rowsPerPage={rowsPerPage}
          page={page}
          onPageChange={handleChangePage}
          onRowsPerPageChange={handleChangeRowsPerPage}
        />
      </Paper>
    </Grid>
    <Grid item xs={6}>
      <Paper style={{ height: 260 }}>
        <canvas id="clogging-chart-week" />
      </Paper>
      <Paper style={{ height: 260, marginTop: 15 }}>
        <canvas id="clogging-chart-month" />
      </Paper>
    </Grid>
  </Grid>
);
}
```