```
import React, { useState, useEffect, useRef } from "react";
import { Link } from "react-router-dom";
import Paper from "@mui/material/Paper";
import Grid from "@mui/material/Grid";
import initializeFirebase from "./firebase/firebase";
import { ref, get } from "firebase/database";
import { MapContainer, TileLayer, Marker, Popup } from "react-leaflet";
import "leaflet/dist/leaflet.css";
import L from "leaflet";
import { useTheme } from "@emotion/react";
import { tokens } from "../theme";
import { Button } from "@mui/material";
import NotificationImportantIcon from "@mui/icons-material/NotificationImportant";
import ErrorIcon from "@mui/icons-material/Error";
import PublishedWithChangesIcon from "@mui/icons-material/PublishedWithChanges";
import Chart from "chart.js/auto";
import progressIcon from "../icons/progress-icon.svg";
import pendingIcon from "../icons/pending-icon.svg";
import norequestIcon from "../icons/noreq-icon.svg";
const progIcon = L.icon({
  iconUrl: progressIcon,
  iconSize: [30, 30],
});
const pendIcon = L.icon({
  iconUrl: pendingIcon,
  iconSize: [30, 30],
});
const noreqIcon = L.icon({
  iconUrl: norequestIcon,
  iconSize: [30, 30],
});
export default function DashboardComponents() {
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);
  const [rows, setRows] = useState([]);
  const [center, setCenter] = useState([14.5798232, 120.98568789]);
  const mapRef = useRef();
  const chartRefWeek = useRef(null);
  const [maintenanceCounts, setMaintenanceCounts] = useState({
    pending: 0,
    nomaintenancereq: 0,
    inprogress: 0,
  });

  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 gutterLocations = Object.entries(data).map(
      ([deviceId, deviceData]) => {
        const {
          name,
          address,
          latitude,
          longitude,
          maintenanceStatus,
          isClogged,
        } = deviceData;
        let clogStatus = "Cleared";
        let clogHistory = [];
        if (isClogged) {
          const clogEvents = Object.entries(isClogged).map(
            ([timestamp, status]) => ({
              timestamp,
              status: status ? "Clogged" : "Cleared",
            }),
          );
          // Sorting the clog events by timestamp
          clogEvents.sort((a, b) => a.timestamp - b.timestamp);
          // Taking the latest clog status
          const latestClogEvent = clogEvents[clogEvents.length - 1];
          if (latestClogEvent) {
            clogStatus = latestClogEvent.status;
            clogHistory = clogEvents;
          }
        }
        return {
          name,
          address,
          latitude,
          longitude,
          maintenanceStatus,
          clogStatus,
          clogHistory,
        };
      },
    );
```

```
setRows(gutterLocations);
if (gutterLocations.length > 0) {
  const centerLocation = gutterLocations[0];
  setCenter([
    parseFloat(centerLocation.latitude),
    parseFloat(centerLocation.longitude),
  ]);
}
let counts = {
  pending: 0,
  nomaintenancereq: 0,
  inprogress: 0,
};

gutterLocations.forEach((deviceData) => {
  counts[deviceData.maintenanceStatus]++;
});

setMaintenanceCounts(counts);

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

const cloggingEvents = {
  Clogged: Array.from({ length: 7 }, () => 0),
  Cleared: Array.from({ length: 7 }, () => 0),
};
Object.values(data).forEach((deviceData) => {
  const { isClogged } = deviceData;
  if (isClogged) {
    Object.entries(isClogged).forEach(([timestamp, status]) => {
      const eventDate = new Date(
        parseInt(timestamp.substring(4, 8)),
        parseInt(timestamp.substring(0, 2)) - 1,
        parseInt(timestamp.substring(2, 4)),
      );

      // Check if the event date is within the current week
      if (eventDate >= currentWeekStart && eventDate <= currentDate) {
        const dayOfWeek = eventDate.getDay();
        cloggingEvents[status ? "Clogged" : "Cleared"][dayOfWeek]++;
      }
```

```
            });
          }
        });
        drawChart(cloggingEvents, "week");
      } 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.Clogged || !cloggingEvents.Cleared) {
    console.error("cloggingEvents or its properties are undefined");
    return;
  }
  const ctx = document.getElementById(`clogging-chart-${type}`);
  const labels = ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"];
  const datasets = [
    {
      label: `Clogged`,
      data: cloggingEvents.Clogged,
      borderColor: "rgba(255, 60, 60, 0.86)",
      backgroundColor: "rgba(255, 222, 222, 0.71)",
      borderWidth: 1,
      fill: true,
    },
    {
      label: `Unclogged`,
      data: cloggingEvents.Cleared,
      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,
          min: 0,
        },
      },
    },
    plugins: {
      title: {
        display: true,
        text: "Clogging Frequency per Week",
        font: {
          size: 14,
        },
      },
    },
  };
  if (chartRefWeek.current) {
    chartRefWeek.current.destroy();
  }
  chartRefWeek.current = new Chart(ctx, {
    type: "line",
    data: {
      labels: labels,
      datasets: datasets,
    },
    options: options,
  });
};
const maintenanceStatusMapping = {
  pending: "Pending",
  nomaintenancereq: "No maintenance required",
  inprogress: "In progress",
};
const maintenanceStatusIcons = {
  pending: pendIcon,
  nomaintenancereq: noreqIcon,
  inprogress: progIcon,
};
return (
  <Grid container spacing={2}>
    <Grid item xs={12} md={6}>
      <MapContainer
```

```
          center={center}
          zoom={16}
          ref={mapRef}
          style={{ height: "100%", width: "100%" }}
          maxZoom={18}
          minZoom={13}
        >
          <TileLayer
            url="https://api.maptiler.com/maps/dataviz/256/{z}/{x}/{y}.png?
key=qKtzXYmOKKYYAxMzX6D4"
            attribution='&copy; <a href="https://www.maptiler.com/copyright/">MapTiler</a>
contributors'
          />
          {rows.map((row, index) => (
            <Marker
              key={index}
              position={[parseFloat(row.latitude), parseFloat(row.longitude)]}
              icon={maintenanceStatusIcons[row.maintenanceStatus.toLowerCase()]}
            >
              <Popup>
                <div>
                  <h2>{row.name}</h2>
                  <p>Address: {row.address}</p>
                  <p>Overflow Status: {row.clogStatus}</p>
                  <p>
                    Maintenance Status:{" "}
                    {maintenanceStatusMapping[row.maintenanceStatus]}
                  </p>
                </div>
              </Popup>
            </Marker>
          ))}
        </MapContainer>
      </Grid>
      <Grid item xs={12} md={6}>
        <Paper style={{ height: "320px", width: "100%" }}>
          <canvas id="clogging-chart-week" />
        </Paper>
      </Grid>
      <Grid item xs={12} md={3}>
        <Paper elevation={2}>
          <div
            style={{ display: "flex", alignItems: "center", padding: "30px" }}
          >
            <ErrorIcon
              style={{ fontSize: 80, color: "red", marginRight: "20px" }}
```

```
      />
      <div style={{ textAlign: "center" }}>
        <h3
          style={{
            fontWeight: "normal",
            margin: "0",
            marginLeft: "10px",
          }}
        >
          Pending Maintenance
        </h3>
        <h3 style={{ margin: "0", fontSize: "50px", marginLeft: "10px" }}>
          {maintenanceCounts.pending}
        </h3>
      </div>
    </div>
  </Paper>
</Grid>
<Grid item xs={12} md={3}>
  <Paper elevation={2}>
    <div
      style={{
        display: "flex",
        alignItems: "center",
        padding: "30px",
      }}
    >
      <PublishedWithChangesIcon
        style={{
          fontSize: 80,
          color: "orange",
          marginRight: "30px",
        }}
      />
      <div style={{ textAlign: "center" }}>
        <h3
          style={{
            fontWeight: "normal",
            margin: "0",
          }}
        >
          Under maintenance
        </h3>
        <h1 style={{ margin: "0", fontSize: "50px" }}>
          {maintenanceCounts.inprogress}
        </h1>
```

```
      </div>
    </div>
  </Paper>
</Grid>
<Grid item xs={12} md={3}>
  <Paper elevation={2}>
    <div
      style={{
        display: "flex",
        alignItems: "center",
        padding: "30px",
      }}
    >
      <NotificationImportantIcon
        style={{
          fontSize: 80,
          color: "darkgray",
          marginRight: "30px",
        }}
      />
      <div style={{ textAlign: "center" }}>
        <h3
          style={{
            fontWeight: "normal",
            margin: "0",
          }}
        >
          Overflowing Gutters
        </h3>
        <h1 style={{ margin: "0", fontSize: "50px" }}>
          {rows.filter((row) => row.clogStatus === "Clogged").length}
        </h1>
      </div>
    </div>
  </Paper>
</Grid>
<Grid item xs={12} md={3} style={{ textAlign: "center" }}>
  <Link to="/device-config">
    <Button
      variant="contained"
      sx={{
        backgroundColor: colors.orangeAccent[400],
        color: colors.primary[900],
        "&:hover": {
          backgroundColor: colors.orangeAccent[300],
        },
```

```
            borderRadius: "5px",
            width: "100%",
            height: "100%",
            fontSize: "20px",
          }}
          fullWidth
        >
          ADD NEW DEVICE
        </Button>
      </Link>
    </Grid>
  </Grid>
 );
}
```