APP.JS

```
import React, { useState } from "react";
import { Container, Typography } from "@mui/material";
import Grid from "@mui/material/Grid2";
import AddCost from "./components/addacost";
import MonthlyReport from "./components/monhtlyreport";
import PieChart from "./components/piecharts";
import { addingCost, getMonthlyCosts, getCostsByCategory } from "./utils/idb";

/**
 * Main application component for managing costs.
 *
 * @returns {JSX.Element} The rendered App component.
 */
const App = () => {
  const [costs, setCosts] = useState([]);
  const [chartData, setChartData] = useState({});
  const DB_NAME = "CostManagerDB";
  const STORE_NAME = "Costs";

  /**
     * Handles adding a new cost to the database.
     *
     * @param {Object} costItem - The cost item to be added, including sum, category,
     */
  const handleAddCost = async (costItem) => {
    await addingCost(DB_NAME, STORE_NAME, costItem);
    alert("Cost Added Successfully");
  };

  /**
     * Handles fetching costs for a specific month and year and updates the state.
     *
     * @param {number} month - The month for which costs are fetched (1-12).
     * @param {number} year - The year for which costs are fetched.
     */
  const handleFetchCosts = async (month, year) => {
    const data = await getMonthlyCosts(DB_NAME, STORE_NAME, month, year);
    setCosts(data || []); // Ensure costs is always an array

    const chartData = await getCostsByCategory(
      DB_NAME,
      STORE_NAME,
      month,
```

APP.JS

```
      year
    );
    setChartData(chartData || {});
  };

  return (
    <Container maxWidth="lg" sx={{ mt: 4 }}>
      <Typography variant="h3" component="h1" gutterBottom align="center">
        Cost Manager
      </Typography>

      <Grid container spacing={3}>
        {/* Center column for AddCost form */}
        <Grid
          xs={12}
          md={4}
          sx={{
            display: "flex",
            justifyContent: "center",
            alignItems: "flex-start",
          }}
        >
          <AddCost onAddCost={handleAddCost} />
        </Grid>

        {/* Left column for Monthly Report */}
        <Grid
          xs={12}
          md={4}
          sx={{
            display: "flex",
            justifyContent: "center",
            alignItems: "flex-start",
          }}
        >
          <MonthlyReport costs={costs || []} onFetchCosts={handleFetchCosts} />
        </Grid>

        {/* Right column for PieChart */}
        <Grid
          xs={12}
          md={4}
          sx={{
            display: "flex",
            justifyContent: "center",
            alignItems: "flex-start",
```

```
          }}
        >
          <PieChart chartData={chartData || {}} />
        </Grid>
      </Grid>
    </Container>
  );
};

export default App;
```

APP.CSS

```css
.App {
  text-align: center;
  background-color: #282c34;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

IDB.JS

```
// idb.js: IndexedDB Wrapper Library

/**
 * Initializes or retrieves the IndexedDB instance.
 *
 * @param {string} dbName - The name of the database.
 * @param {string} storeName - The name of the object store.
 * @returns {Promise<IDBDatabase>} A promise that resolves with the database instance.
 */
export const defineIdb = (dbName, storeName) => {
  return new Promise((resolve, reject) => {
    const request = indexedDB.open(dbName, 1);

    // Handle database initialization
    request.onupgradeneeded = (event) => {
      const db = event.target.result;
      if (!db.objectStoreNames.contains(storeName)) {
        db.createObjectStore(storeName, { keyPath: "id", autoIncrement: true });
      }
    };

    request.onsuccess = () => resolve(request.result);
    request.onerror = () => reject(request.error);
  });
};

/**
 * Performs a transaction on the IndexedDB store.
 *
 * @param {IDBDatabase} db - The database instance.
 * @param {string} storeName - The name of the object store.
 * @param {"readonly"|"readwrite"} mode - The transaction mode.
 * @param {Function} callback - The callback function to perform an operation on the store.
 * @returns {Promise<any>} A promise that resolves with the result of the operation.
 */
export const performTransaction = (db, storeName, mode, callback) => {
  return new Promise((resolve, reject) => {
    const transaction = db.transaction(storeName, mode);
    const store = transaction.objectStore(storeName);
    const request = callback(store);

    request.onsuccess = () => resolve(request.result);
    request.onerror = () => reject(request.error);
  });
};

/**
 * Adds a cost item to the database.
 *
 * @param {string} dbName - The name of the database.
 * @param {string} storeName - The name of the object store.
 * @param {Object} costItem - The cost item to add, containing details like `sum`,
 *  `category`, `description`, and `date`.
 * @returns {Promise<number>} A promise that resolves with the ID of the added item.
 */
```

IDB.JS

```
export const addingCost = async (dbName, storeName, costItem) => {
  const db = await defineIdb(dbName, storeName);
  return performTransaction(db, storeName, "readwrite", (store) =>
    store.add(costItem)
  );
};

/**
 * Retrieves costs for a specific month and year.
 *
 * @param {string} dbName - The name of the database.
 * @param {string} storeName - The name of the object store.
 * @param {number} month - The month to filter costs (1-12).
 * @param {number} year - The year to filter costs.
 * @returns {Promise<Object[]>} A promise that resolves with an array of cost items for
 *  the specified month and year.
 */
export const getMonthlyCosts = async (dbName, storeName, month, year) => {
  const db = await defineIdb(dbName, storeName);

  return new Promise((resolve, reject) => {
    const transaction = db.transaction(storeName, "readonly");
    const store = transaction.objectStore(storeName);
    const request = store.getAll();

    request.onsuccess = () => {
      const allItems = request.result;
      const filteredItems = allItems.filter((item) => {
        const itemDate = new Date(item.date);
        return (
          // getMonth returns 0-based month
          itemDate.getMonth() + 1 === month && itemDate.getFullYear() === year
        );
      });
      resolve(filteredItems);
    };

    request.onerror = () => reject(request.error);
  });
};

/**
 * Retrieves the total costs grouped by category for a specific month and year.
 *
 * @param {string} dbName - The name of the database.
 * @param {string} storeName - The name of the object store.
 * @param {number} month - The month to filter costs (1-12).
 * @param {number} year - The year to filter costs.
 * @returns {Promise<Object>} A promise that resolves with an object where keys are categories
 * and values are the total sums.
 */
export const getCostsByCategory = async (dbName, storeName, month, year) => {
  const costs = await getMonthlyCosts(dbName, storeName, month, year);
  return costs.reduce((acc, cost) => {
    acc[cost.category] = (acc[cost.category] || 0) + cost.sum;
    return acc;
  }, {});

```

```
        };
```

ADD COST

```jsx
import React, { useState } from "react";
import { Box, TextField, Button, Typography, Stack } from "@mui/material";

/**
 * Componnet for adding a new cost.
 *
 * @param {Object} props - The component props.
 * @param {Function} props.onAddCost - Callback function to handle adding a new cost.
 * @returns {JSX.Element} The rendered AddCost component.
 */
const AddCost = ({ onAddCost }) => {
  const [formData, setFormData] = useState({
    sum: "",
    category: "",
    description: "",
    date: "", // Stores the selecte month and year
  });

  /**
     * Handles chagnes in form fields and updates the state.
     *
     * @param {React.ChangeEvent<HTMLInputElement>} e - The input change event.
     */
  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  /**
     * Handles form submittion and calls the `onAddCost` callback.
     *
     * @param {React.FormEvent<HTMLFormElement>} e - The form submit event.
     */
  const handleSubmit = (e) => {
    e.preventDefault();

    // Convert the selected month-year to a full date
    const [year, month] = formData.date.split("-");
    const formattedDate = new Date(year, month - 1, 1); // First day of the selected month

    onAddCost({
      ...formData,
      date: formattedDate
    });

    // Reset the form
    setFormData({ sum: "", category: "", description: "", date: "" });
  };

  return (
    <Box component="form" onSubmit={handleSubmit}>
      <Typography variant="h5" gutterBottom>
        Add Cost
      </Typography>
      <Stack spacing={2}>
        <TextField
```

ADD COST

```
          type="number"
          label="Sum"
          name="sum"
          value={formData.sum}
          onChange={handleChange}
          fullWidth
          required
        />
        <TextField
          label="Categoty"
          name="category"
          value={formData.category}
          onChange={handleChange}
          fullWidth
          required
        />
        <TextField
          label="Descripton"
          name="description"
          value={formData.description}
          onChange={handleChange}
          fullWidth
          multiline
          rows={3}
          required
        />
        <TextField
          type="month"
          label="Month & Year"
          name="date"
          value={formData.date}
          onChange={handleChange}
          fullWidth
          required
          InputLabelProps={{ shrink: true }} // Ensures lable is always visible
        />
        <Button variant="contained" type="submit" fullWidth>
          Add
        </Button>
      </Stack>
    </Box>
  );
};

export default AddCost;
```

MONTHLY REPORT

```jsx
import React, { useState } from "react";
import { Typography, Box, Button, MenuItem, Select } from "@mui/material";

/**
 * Component to display the monthly report of costs.
 *
 * @param {Object} props - Component props.
 * @param {Array} props.costs - List of costs for the month.
 * @param {Function} props.onFetchCosts - Function to fetch costs for
 * a specific month and year.
 */
const MonthlyReport = ({ costs, onFetchCosts }) => {
  const currentMonth = new Date().getMonth() + 1; // Current month (1-12)
  const currentYear = new Date().getFullYear(); // Current year

  const [selectedMonth, setSelectedMonth] = useState(currentMonth);
  const [selectedYear, setSelectedYear] = useState(currentYear);

  const handleFetch = () => {
    onFetchCosts(selectedMonth, selectedYear);
  };

  const filteredCosts = costs.filter((cost) => {
    const costDate = new Date(cost.date);
    return (
      costDate.getMonth() + 1 === selectedMonth &&
      costDate.getFullYear() === selectedYear
    );
  });

  return (
    <Box sx={{ width: "100%", textAlign: "center", p: 2 }}>
      <Box sx={{ display: "flex", justifyContent: "center", mb: 2 }}>
        <Select
          value={selectedMonth}
          onChange={(e) => setSelectedMonth(e.target.value)}
          sx={{ mr: 2 }}
        >
          {[...Array(12).keys()].map((month) => (
            <MenuItem key={month + 1} value={month + 1}>
              {new Date(0, month).toLocaleString("default", { month: "long" })}
            </MenuItem>
          ))}
        </Select>
        <Select
          value={selectedYear}
          onChange={(e) => setSelectedYear(e.target.value)}
        >
          {[
            currentYear - 5,
            currentYear - 4,
            currentYear - 3,
            currentYear - 2,
            currentYear - 1,
            currentYear,
```

MONTHLY REPORT

```
          ].map((year) => (
            <MenuItem key={year} value={year}>
              {year}
            </MenuItem>
          ))}
        </Select>
      </Box>
      {filteredCosts.length === 0 ? (
        <Typography variant="body1" sx={{ mb: 2 }}>
          No costs found for the selected month and year.
        </Typography>
      ) : (
        <Box>
          <Typography variant="h6" sx={{ mb: 2 }}>
            Monthly Report for{" "}
            {new Date(0, selectedMonth - 1).toLocaleString("default", {
              month: "long",
            })}{" "}
            {selectedYear}
          </Typography>
          <ul>
            {filteredCosts.map((cost, index) => (
              <li key={index}>
                {new Date(cost.date).toLocaleDateString()} : {cost.description}{" "}
                - {cost.sum} ({cost.category})
              </li>
            ))}
          </ul>
        </Box>
      )}
      <Button
        variant="contained"
        color="primary"
        onClick={handleFetch}
        sx={{ mt: 2 }}
      >
        Refresh Report
      </Button>
    </Box>
  );
};

export default MonthlyReport;
```

PIE CHART

```jsx
import React from "react";
import { Pie } from "react-chartjs-2";
import { Box, Typography } from "@mui/material";
import { Chart as ChartJS, ArcElement, Tooltip, Legend } from "chart.js";

// Manually register the necessary Chart.js components
ChartJS.register(ArcElement, Tooltip, Legend);

/**
 * Component to display a pie chart for visualizing cost distribution by category.
 *
 * @param {Object} props - The component props.
 * @param {Object} props.chartData - An object where keys are cost categories
 *  and values are the total costs for those categories.
 * @returns {JSX.Element} The rendered PieChart component.
 */
const PieChart = ({ chartData }) => {
  const data = {
    labels: Object.keys(chartData),
    datasets: [
      {
        label: "Costs by Category",
        data: Object.values(chartData),
        backgroundColor: ["#FF6384", "#36A2EB", "#FFCE56", "#4BC0C0"],
        hoverBackgroundColor: ["#FF6384", "#36A2EB", "#FFCE56", "#4BC0C0"],
      },
    ],
  };

  return (
    <Box
      sx={{
        width: "240px",
        height: "240px",
        margin: "0 auto",
      }}
    >
      <Typography variant="h5" gutterBottom>
        Cost Distribution
      </Typography>
      <Pie data={data} />
    </Box>
  );
};

export default PieChart;
```