CLEVERDASH

Designed and developed by

1. SUBHAPREET PATRO 2211CS010547
2. T. SANTHOSHI 2211CS010557
3 . MUNIMANDA SNEHA 2211CS010390
4. DUBBAKA PRASANNA 2211CS010620

Under the guidance of

MRS.SABITHA

Associate Professor

Department of Computer Science & Engineering

MALLA REDDY UNIVERSITY, HYDERABAD

2023-2024

MALLA REDDY UNIVERSITY

(Telangana State Private Universities Act No.13 of 2020 and G.O.Ms.No.14, Higher Education (UE) Department)

CERTIFICATE

This is to certify that this is the Application development lab record entitled **"CLEVERDASH"**, submitted by **SUBHAPREET PATRO (2211CS010547), T.SANTHOSHI (2211CS0100557), MUNIMANDA SNEHA (2211CS010390), DUBBAKA PRASANNA (2211CS010620)** B. Tech **II** year **II** semester, Department of CSE during the year 2022-23. The results embodied in this report have not been submitted to any other university or institute for the award of any degree or diploma.

GUIDE **HOD-CSE**

Mrs.Sabitha **Dr. Shaik Meeravali**

External Examiner

MALLA REDDY UNIVERSITY
(Telangana State Private Universities Act No.13 of 2020 and G.O.Ms.No.14, Higher Education (UE) Department)

DECLARATION

I declare that this project report titled **CLEVERDASH** submitted in partial fulfillment of the degree of B. Tech in CSE is a record of original work carried out by me under the supervision of **Mrs.Sabitha** and has not formed the basis for the award of any other degree or diploma, in this or any other Institution or University. In keeping with the ethical practice in reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

SUBHAPREET PATRO 2211CS010547

T. SANTHOSHI 2211CS010557

MUNIMANDA SNEHA 2211CS010390

DUBBAKA PRASANNA 2211CS010620

ACKNOWLEDGEMENT

With great pleasure we want to take this opportunity to express our heartfelt gratitudeto all the people who helped in making this mini project work a grand success.

We express our deep sense of gratitude to **Mrs.Sabitha** for her constant guidance throughout our mini project work.

We are grateful to **Mr M.Rakesh** for his/her valuable suggestions and guidance given by him during the execution of this mini project work.

We would like to thank **Dr. Shaik Meeravali**, **Head of the Departmentof Computer Science and Engineering,** for providing seamless support and right Suggestions in the development of App.

First of all we are highly indebted to **Vice Chancellor,** for giving us the permission to carry out this mini project.

We would like to thank the **Teaching & Non-Teaching staff of CSE Department** for sharing their knowledge with us.

Last but not the least we express our sincere thanks to **Chairman** for this continuous care towards our achievements.

SUBHAPREET PATRO **2211CS010547**

T. SANTHOSHI **2211CS010557**

MUNIMANDA SNEHA **2211CS010390**

DUBBAKA PRASANNA **2211CS010620**

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

"Clever Dash" is a sophisticated admin dashboard application built from scratch using the MERN stack (MongoDB, Express, React, Node.js). This application includes setting up the backend and frontend environments, configuring themes and colors, creating components such as navbar, sidebar, user profile menu, and product pages, handling user data and authentication, , and integrating data visualization components using libraries like Nivo. This application servers as a great tool for managing financial data.

Clever Dash introduces a groundbreaking approach to navigation efficiency with its innovative smart navbar integration. Designed to cater to the modern user's dynamic needs, Clever Dash offers a seamless and intuitive navigation experience across diverse applications and platforms.

1. INTRODUCTION

1.1 Introduction to application

Welcome to Clever Dash, the ultimate solution for optimizing your navigation experience across various digital platforms. In today's fast-paced world, efficient navigation is paramount, and Clever Dash is here to revolutionize the way you interact with your devices.

At the heart of Clever Dash lies its innovative smart navbar integration. Say goodbye to cluttered screens and cumbersome menus. Clever Dash's intelligent navbar adapts to your preferences and usage patterns, ensuring that the tools and features you need are always at your fingertips.

Gone are the days of endless scrolling and searching for essential functions. With Clever Dash, you can navigate between tasks seamlessly, access frequently used tools with a single tap.

Clever Dash empowers you to take control of your digital environment. Say hello to productivity, simplicity, and convenience – welcome to Clever Dash.

1.2 Objectives:

1. Enhanced Data Visualization:

Develop advanced visualization techniques using Nivo Charts to provide users with clear and insightful representations of complex datasets.

2.Scalability and Performance:

Utilize Node.js and Express.js to create a robust backend infrastructure that can handle large volumes of data while maintaining high performance and scalability.

3. Data Security and Reliability:

Employ Mongoose and MongoDB to establish a secure and reliable data storage solution, safeguarding sensitive information and ensuring data integrity.

4. Performance Analytics:

Offer built-in analytics features to track dashboard usage and monitor performance metrics

1.3 Goal of Project:

The primary goal of Clever Dash is to streamline and optimize navigation experiences across digital platforms by providing users with a user-friendly navbar. Clever Dash aims to:

1. Simplify Navigation: Clever Dash simplifies the navigation process, reducing the time and effort required to switch between tasks.

2. Enhance Productivity: Through intuitive design, Clever Dash empowers users to accomplish tasks more efficiently, ultimately boosting productivity and workflow effectiveness.

3. Optimize Screen Space: Clever Dash optimizes screen real estate by intelligently organizing navigation elements, maximizing usability without cluttering the interface.

4. Data Visualization: Clever Dash aims to provide a way to represent data in the form of different graphs that make it easier to identify trends.

Overall, Clever Dash aims to revolutionize the way users interact, making navigation simpler, more intuitive, and ultimately more enjoyable.

2. LITERATURE SURVEY

1. Tableau

Tableau is a leading data visualization and business intelligence tool known for its interactive and shareable dashboards that turn raw data into an understandable format.

Advantages:

- **Powerful Visualizations**: Offers a wide range of visualization options to create detailed and interactive dashboards.
- **User-Friendly**: Intuitive drag-and-drop interface makes it accessible for users without extensive technical knowledge.
- **Integration**: Supports a vast array of data sources including cloud and on-premises databases.

Limitations:

- **Cost**: Can be expensive, particularly for small businesses, due to its licensing model.
- **Performance**: Handling very large datasets can sometimes slow down performance.
- **Learning Curve**: While the interface is user-friendly, mastering all features can take time.

2. Google Data Studio

Google Data Studio is a free data visualization and reporting tool that enables users to create interactive and shareable reports with real-time data from various sources.

Advantages:

- **Cost**: Free to use, making it highly accessible for individuals and businesses.
- **Integration**: Seamless integration with Google ecosystem (Google Analytics, Google Ads, BigQuery) and other data sources.
- **Ease of Use**: Intuitive interface with drag-and-drop features and pre-built templates.

Limitations:

- **Feature Limitations**: Lacks some advanced features and customization options available in paid BI tools.
- **Data Handling**: May struggle with very large datasets or complex data manipulations.
- **Support**: Limited support options; primarily relies on community forums and documentation.

2.1 Existing System:

Dashboard applications have become essential tools in modern business intelligence, offering organizations the ability to visualize, analyze, and interact with their data in real-time. Tools like Tableau, Zoho Analytics, Qlik, and Google Data Studio exemplify the diversity and capabilities of these applications. Tableau is renowned for its powerful and interactive visualizations, making complex data accessible and understandable, though it can be costly and requires a learning curve. Zoho Analytics stands out for its affordability and seamless integration with the Zoho suite, offering AI-powered insights, though it may lack some advanced features. Qlik's associative data indexing engine enables deep, dynamic data exploration and real-time interaction, but its complexity and cost can be barriers. Google Data Studio, being free and tightly integrated with Google services, provides an intuitive and collaborative platform, albeit with some limitations in handling large datasets and advanced features. Collectively, these tools demonstrate the critical role of dashboard applications in transforming data into actionable insights, catering to a wide range of business needs and technical proficiencies.

2.2 Problem Identification

To effectively identify and address issues related to the navigation bar (navbar) in an app named CleverDash, it is essential to consider a comprehensive approach. Here are some common problems and how to diagnose and resolve them:

1. Visual Design Issues:

Problem: Navbar elements might not be visually appealing or consistent with the overall app design.

Diagnosis: Conduct a UI/UX audit to check for inconsistencies in color schemes, fonts, and element sizes.

Solution: Standardize the design elements according to the app's style guide. Use tools like Figma or Sketch for prototyping and design consistency.

2. Performance Problems:

Problem: The navbar and other app components might slow down the app's performance, especially if it includes pages with complex implementation.

Diagnosis: Use performance profiling tools like Google Lighthouse to identify bottlenecks.

Solution: Optimize code and minimize errors by using proper error handling techniques, and use Test APIs (like Talend) to ensure proper testing of frontend to backend connection.

3. Functionality Issues:

Problem: Some interactive elements in the navbar might not work as expected.

Diagnosis: Conduct thorough testing to identify broken links, unresponsive buttons, components, or malfunctioning dropdowns.

Solution: Debug and fix logical errors, ensure proper linkage of elements, and test and refer documentations of component design libraries like Material UI to ensure proper implementation of designs.

3. RESEARCH METHODLOGY

3.1 Proposed Algorithm

Developing the research methodology for an app like CleverDash, specifically focusing on the navbar using the MERN stack, involves several stages. Here's a structured approach to conduct comprehensive research:

1. Problem Definition and Objectives:

- Identify the Problem: Understand the specific issues the navbar aims to solve (e.g., user navigation efficiency, accessibility).
- Set Objectives: Define clear objectives, such as improving user experience, optimizing performance, or ensuring cross-device compatibility.

2. Literature Review:

- Review Existing Solutions: Study existing navbars in similar applications to identify best practices and common pitfalls.
- Technical Research: Investigate the latest trends and techniques in building navbars using the MERN stack.

3. Requirement Analysis:

- User Requirements: Gather user requirements through surveys, interviews, or through referring similar projects to understand their needs and preferences.
- Technical Requirements: Define technical requirements, including performance, responsiveness, and integration with the rest of the MERN stack.

4. Design and Prototyping:

- Wireframing: Create wireframes to visualize the layout and functionality of the navbar.
- Prototyping: Develop interactive prototypes using tools like Figma or Sketch to refine the design based on user feedback and refer to documentation of component design libraries like Material UI.

5. Technology Stack Analysis:

- MongoDB: Assess the data storage needs for user preferences and settings related to the navbar.
- Express.js: Determine how server-side logic will manage and serve navbar-related data.
- React.js: Plan the implementation of the navbar as a React component along with the components of other pages, considering state management and reusability.
- Node.js: Outline how backend services will support the navbar's functionality, including API endpoints.

6. Implementation Strategy:

- Development Plan: Create a detailed development plan with milestones and deliverables.
- Component Design: Design React components for the navbar, ensuring modularity and maintainability.
- Integration: Plan the integration of the navbar with other parts of the application, ensuring consistency and coherence.

4. REQUIREMENTS

4.1 Software Requirements

- React JS and Material UI for the user interface
- Nivo Charts for displaying various charts and to also display geographical information.
- React-Toastify for alerts and feedback for form actions.
- Formik and Yup for proper form management in various pages.
- Node JS as a runtime environment for the app to run outside of browser.
- Express JS as a backend framework for Node JS.
- Mongoose for a schema-based solution to model application data.
- MongoDB for database

4.2 Hardware Requirements

- **Processor**:
  - Minimum: Dual-core processor (Intel i5 or equivalent)
  - Recommended: Quad-core processor (Intel i7 or equivalent)
- **RAM**:
  - Minimum: 8 GB
  - Recommended: 16 GB or more (to handle multiple running processes like IDEs, servers, and database)
- **Storage**:
  - Minimum: 256 GB SSD
  - Recommended: 512 GB SSD or more (for faster read/write operations and to accommodate software dependencies)
- **Operating System**:
  - Windows 10/11, macOS, or a Linux distribution (such as Ubuntu)
- **Other Requirements**:
  - Modern web browser (Chrome, Firefox)
  - Code editor/IDE (VS Code, WebStorm)
  - Node.js installed
  - MongoDB installed (or using a cloud service like MongoDB Atlas)

5. DESIGN AND IMPLEMENTATION

5.1 Design

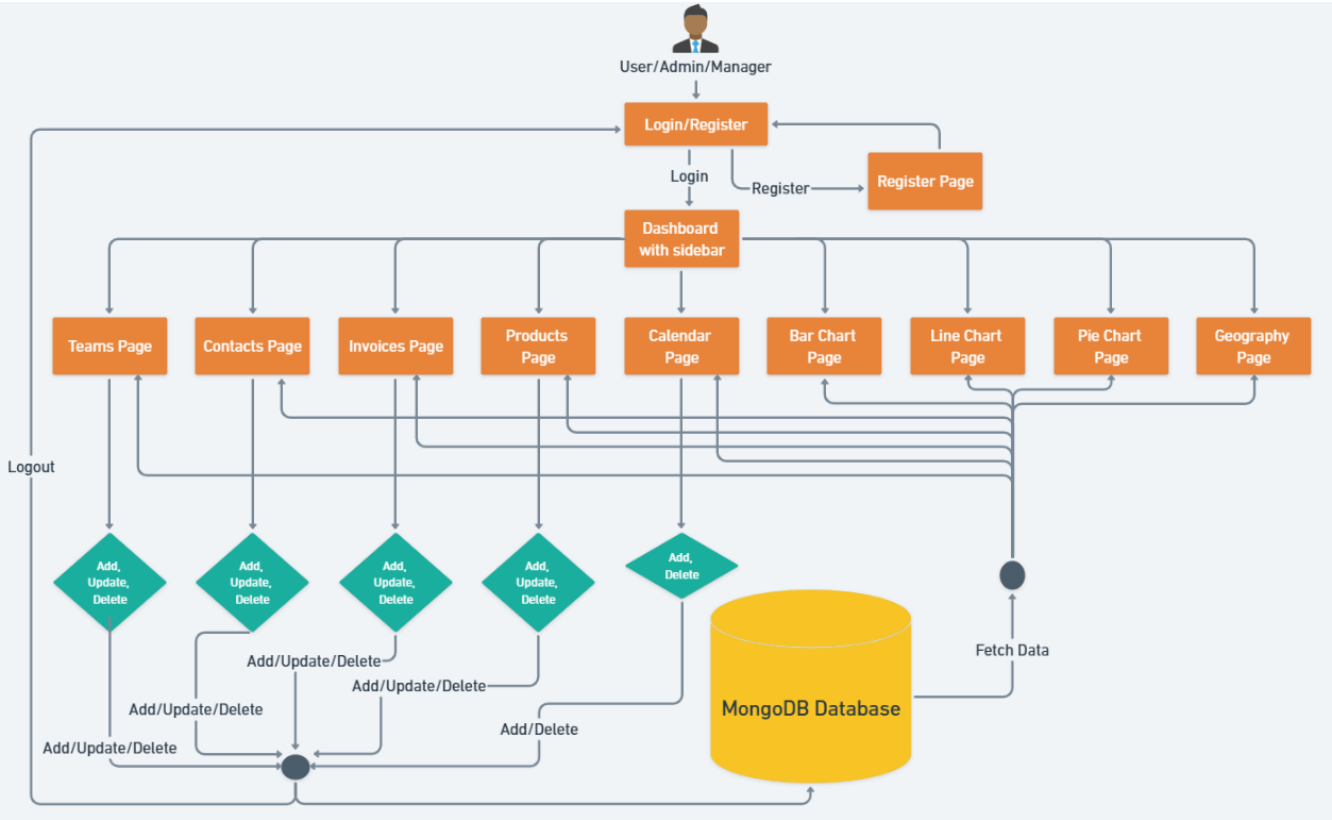

Fig: 5.1 Design of Clever-Dash

5.2 Implementation

Source code:

CLIENT: COMPONENTS

Authentication.css:

```
@import
url("https://fonts.googleapis.com/css?family=Poppins");
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
.in h2 {
  margin: 50px;
}

.container {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  width: 100vh;
  height: 100vh;
  /* margin: auto; */
  margin-bottom: 133px;
}

.main {
  position: relative;
  display: flex;
  flex-direction: column;
  /* background-color: #240046; */
  background-color: rgb(241, 231, 80);
  max-height: 482px;
  width: 350px;
  overflow: hidden;
  border-radius: 12px;
  box-shadow: 7px 7px 10px 9px #24004628;
  font-family: "Poppins", sans-serif;
}

.form {
  display: flex;
  align-items: center;
  flex-direction: column;
  gap: 14px;
  padding: 24px;
}

/* Checkbox to switch from Register to Login */
```

```
#chk {
  display: none;
}

.login {
  position: relative;
  width: 100%;
  height: 100%;
}

.login label {
  margin: 25% 0 5%;
  /* Addition */
  color: orangered;
}

label {
  color: #fff;
  font-size: 2rem;
  justify-content: center;
  display: flex;
  font-weight: bold;
  cursor: pointer;
  transition: 0.5s ease-in-out;
}

.input,
select {
  width: 100%;
  height: 40px;
  background: #e0dede;
  padding: 10px;
  /* border: none; */
  border: 0.1px solid black;
  outline: none;
  border-radius: 4px;
}

.register {
  /* background: #eee; */
  background: rgb(85, 221, 221);
  border-radius: 70%/10%;
  height: 100%;
  transform: translateY(5%);
  transition: 0.8s ease-in-out;
}

.register label {
  /* color: #573b8a; */
  color: orangered;
  transform: scale(0.6);
}

#chk:checked ~ .register {
  transform: translateY(-60%);
}
```

```
#chk:checked ~ .register label {
  transform: scale(1);
  margin: 10% 0 5%;
}

#chk:checked ~ .login label {
  transform: scale(0.6);
  margin: 5% 0 5%;
}

.form button {
  width: 85%;
  height: 48px;
  margin: 12px auto 10%;
  color: #fff;
  /* background: #5738ba; */
  background: rgb(255, 136, 0);
  font-size: 1rem;
  font-weight: bold;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  transition: 0.2s ease-in;
}

.form button:hover {
  background-color: #f34f4f;
}

.toast-container {
  .Toastify__toast {
    font-family: Arial, sans-serif;
  }

  .Toastify__toast--success {
    background: #5cb85c;
  }

  .Toastify__toast--error {
    background: #d9534f;
  }

  .Toastify__close-button {
    color: #fff;
  }
}
```

AUTHENTICATION.JSX:

```
import React, { useState } from "react";
import "./Authentication.css";
import { ToastContainer, toast } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";
import { useNavigate } from "react-router-dom";
```

```
const Authentication = ({ setAuthenticated }) => {
  const navigate = useNavigate();
  const [formData, setFormData] = useState({
    username: "",
    password: "",
  });
  const [Mode, setMode] = useState(false);

  function handleLoginSubmit(e) {
    e.preventDefault();
    fetch("http://localhost:3547/users")
      .then((res) => {
        if (res.ok) {
          setFormRegisterData({ name: "", username: "", password: "" });
          return res.json();
        }
        throw new Error("Network response was not ok.");
      })
      .then((users) => {
        const user = users.find(
          (user) =>
            user.username === formData.username &&
            user.password === formData.password
        );
        if (user) {
          toast.success("Login Successful");
          navigate("/dashboard");
          localStorage.setItem("Name", user.username);
          localStorage.setItem("Role", user.role);
          setAuthenticated(true); // Set authentication status to true
        } else {
          toast.error("Invalid Username or Password.");
        }
      })
      .catch((error) => {
        console.error("Error: ", error);
        toast.error("Login failed");
      });
  }

  function handleLoginChange(e) {
    const { name, value } = e.target;
    setFormData({ ...formData, [name]: value });
  }

  const [formRegisterData, setFormRegisterData] = useState({
    role: "",
    username: "",
    password: "",
  });

  function handleRegisterSubmit(e) {
    e.preventDefault();
    if (!formRegisterData.role) {
      toast.error("Please select a role.");
      return;
    }
    fetch("http://localhost:3547/users", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(formRegisterData),
    })
```

```
    .then((res) => {
      if (res.ok) {
        setFormData({ username: "", password: "" });
        return res.json();
      }
      throw new Error("Network response was not ok.");
    })
    .then((data) => {
      toast.success("Registered Successfully.");
      setMode(true);
    })
    .catch((error) => {
      console.error("Error:", error);
      toast.error("Register failed");
    });
}

function handleRegisterChange(e) {
  const { name, value } = e.target;
  setFormRegisterData({ ...formRegisterData, [name]: value });
}

return (
  <div
    className="container"
    style={{ height: "100vh", width: "200vh", backgroundColor: "#3631B3" }}
  >
    <div className="in">
      <h2 style={{ color: "#FBC903", fontSize: "25px" }}>CLEVER-DASH</h2>
    </div>
    <div className="container">
      <div className="main">
        <input type="checkbox" id="chk" aria-hidden="true" />
        <div className="login">
          <form className="form" onSubmit={handleLoginSubmit}>
            <label htmlFor="chk" aria-hidden="true">
              Log in
            </label>
            <input
              className="input"
              type="text"
              autoComplete="off"
              name="username"
              value={formData.username}
              onChange={handleLoginChange}
              placeholder="Username"
              required
            />
            <input
              className="input"
              type="password"
              autoComplete="off"
              name="password"
              value={formData.password}
              onChange={handleLoginChange}
              placeholder="Password"
              required
            />
            <button>Log in</button>
          </form>
        </div>
        <div className="register">
          <form className="form" onSubmit={handleRegisterSubmit}>
```

```
            <label htmlFor="chk" aria-hidden="true">
              Register
            </label>
            <select
              name="role"
              value={formRegisterData.role}
              onChange={handleRegisterChange}
              placeholder="Select Role"
            >
              <option value="">Select Role</option>
              <option value="Admin">Admin</option>
              <option value="Manager">Manager</option>
              <option value="User">User</option>
            </select>
            <input
              className="input"
              type="text"
              autoComplete="off"
              name="username"
              value={formRegisterData.username}
              onChange={handleRegisterChange}
              placeholder="Username"
              required
            />
            <input
              className="input"
              type="password"
              autoComplete="off"
              name="password"
              value={formRegisterData.password}
              onChange={handleRegisterChange}
              placeholder="Password"
              required
            />
            <button>Register</button>
          </form>
        </div>
      </div>
    </div>
    <ToastContainer />
  </div>
 );
};

export default Authentication;
```

HEADER.JSX:

```
import { Typography, Box, useTheme } from "@mui/material";
import { tokens } from "../theme";

const Header = ({ title, subtitle }) => {
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);
  return (
    <Box mb="30px">
      <Typography
        variant="h2"
        color={colors.grey[100]}
        fontWeight="bold"
        sx={{ m: "0 0 5px 0" }}
```

```
      >
        {title}
      </Typography>
      <Typography variant="h5" color={colors.greenAccent[400]}>
        {subtitle}
      </Typography>
    </Box>
  );
};

export default Header;
```

PROGRESSCIRCLE.JSX:

```
import { Box, useTheme } from "@mui/material";
import { tokens } from "../theme";

const ProgressCircle = ({ progress = "0.75", size = "40" }) => {
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);
  const angle = progress * 360;
  return (
    <Box
      sx={{
        background: `radial-gradient(${colors.primary[400]} 55%, transparent 56%),
            conic-gradient(transparent 0deg ${angle}deg, ${colors.blueAccent[500]} ${angle}deg 360deg),
            ${colors.greenAccent[500]}`,
        borderRadius: "50%",
        width: ${size}px,
        height: ${size}px,
      }}
    />
  );
};

export default ProgressCircle;
```

STATBOX.JSX:

```
import { Box, Typography, useTheme } from "@mui/material";
import { tokens } from "../theme";
import ProgressCircle from "./ProgressCircle";

const StatBox = ({ title, subtitle, icon, progress, increase }) => {
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);

  return (
    <Box width="100%" m="0 30px">
      <Box display="flex" justifyContent="space-between">
        <Box>
          {icon}
          <Typography
            variant="h4"
            fontWeight="bold"
            sx={{ color: colors.grey[100] }}
          >
            {title}
          </Typography>
```

```
        </Box>
        <Box>
          <ProgressCircle progress={progress} />
        </Box>
      </Box>
      <Box display="flex" justifyContent="space-between" mt="2px">
        <Typography variant="h5" sx={{ color: colors.greenAccent[500] }}>
          {subtitle}
        </Typography>
        <Typography
          variant="h5"
          fontStyle="italic"
          sx={{ color: colors.greenAccent[600] }}
        >
          {increase}
        </Typography>
      </Box>
    </Box>
  );
};

export default StatBox;
```

SCENES:

BAR:INDEX.JSX:

```
import React from "react";
import { Box } from "@mui/material";
import Header from "../../components/Header";
import { ResponsiveBar } from "@nivo/bar";
import { useTheme } from "@mui/material";
import { tokens } from "../../theme";
import { mockBarData as data } from "../../data/mockData";

const Bar = () => {
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);

  return (
    <Box m="20px">
      <Header title="BAR CHART" subtitle="Track the no. of products sold" />
      <Box height="75vh">
        <ResponsiveBar
          data={data}
          theme={{
            axis: {
              domain: {
                line: {
                  stroke: colors.grey[100],
                },
              },
              legend: {
                text: {
                  fill: colors.grey[100],
                },
              },
              ticks: {
                line: {
                  stroke: colors.grey[100],
                  strokeWidth: 1,
```

```
      },
      text: {
        fill: colors.grey[100],
      },
    },
  },
  legends: {
    text: {
      fill: colors.grey[100],
    },
  },
}}
keys={[
  "Category_A",
  "Category_B",
  "Category_C",
  "Category_D",
  "Category_E",
  "Category_F",
]}
indexBy="country"
margin={{ top: 50, right: 130, bottom: 50, left: 60 }}
padding={0.3}
valueScale={{ type: "linear" }}
indexScale={{ type: "band", round: true }}
colors={{ scheme: "nivo" }}
defs={[
  {
    id: "dots",
    type: "patternDots",
    background: "inherit",
    color: "#38bcb2",
    size: 4,
    padding: 1,
    stagger: true,
  },
  {
    id: "lines",
    type: "patternLines",
    background: "inherit",
    color: "#eed312",
    rotation: -45,
    lineWidth: 6,
    spacing: 10,
  },
]}
borderColor={{
  from: "color",
  modifiers: [["darker", "1.6"]],
}}
axisTop={null}
axisRight={null}
axisBottom={{
  tickSize: 5,
  tickPadding: 5,
  tickRotation: 0,
  legend: "Country",
  legendPosition: "middle",
  legendOffset: 32,
}}
axisLeft={{
  tickSize: 5,
  tickPadding: 5,
```

```
            tickRotation: 0,
            legend: "Sales Quantity",
            legendPosition: "middle",
            legendOffset: -40,
          }}
          enableLabel={false}
          labelSkipWidth={12}
          labelSkipHeight={12}
          labelTextColor={{
            from: "color",
            modifiers: [["darker", 1.6]],
          }}
          legends={[
            {
              dataFrom: "keys",
              anchor: "bottom-right",
              direction: "column",
              justify: false,
              translateX: 120,
              translateY: 0,
              itemsSpacing: 2,
              itemWidth: 100,
              itemHeight: 20,
              itemDirection: "left-to-right",
              itemOpacity: 0.85,
              symbolSize: 20,
              effects: [
                {
                  on: "hover",
                  style: {
                    itemOpacity: 1,
                  },
                },
              ],
            },
          ]}
          role="application"
          barAriaLabel={function (e) {
            return (
              e.id + ": " + e.formattedValue + " in country: " + e.indexValue
            );
          }}
        />
      </Box>
    </Box>
  );
};

export default Bar;
```

CALENDER.JSX:

```
import React, { useState, useEffect } from "react";
import FullCalendar, { formatDate } from "@fullcalendar/react";
import dayGridPlugin from "@fullcalendar/daygrid";
import timeGridPlugin from "@fullcalendar/timegrid";
import interactionPlugin from "@fullcalendar/interaction";
import listPlugin from "@fullcalendar/list";
import {
  Box,
  List,
```

```
  ListItem,
  ListItemText,
  Typography,
  useTheme,
} from "@mui/material";
import { toast, ToastContainer } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";
import Header from "../../components/Header";
import { tokens } from "../../theme";

const Calendar = () => {
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);
  const [currentEvents, setCurrentEvents] = useState([]);
  const [title, setTitle] = useState("");
  const [date, setDate] = useState("");

  useEffect(() => {
    fetchEvents();
  }, []);

  const fetchEvents = () => {
    fetch("http://localhost:9000/getEvents")
      .then((response) => {
        if (!response.ok) {
          throw new Error("Failed to fetch events");
        }
        return response.json();
      })
      .then((data) => setCurrentEvents(data))
      .catch((error) => {
        console.error("Error fetching events:", error);
        toast.error("Failed to fetch events");
      });
  };

  const handleDateClick = (selected) => {
    const newDate = selected.startStr;
    const newTitle = prompt("Please enter a new title for your event");
    if (newTitle) {
      addEvent(newTitle, newDate);
    } else {
      toast.error("Event title is required!");
    }
  };

  const handleEventClick = (selected) => {
    if (
      window.confirm(
        Are you sure you want to delete the event '${selected.event.title}'?
      )
    ) {
      deleteEvent(selected.event.id);
    }
  };

  const addEvent = (title, date) => {
    fetch("http://localhost:9000/addEvent", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ title, date }),
```

```
  })
    .then((response) => {
      if (!response.ok) {
        throw new Error("Failed to add event");
      }
      return response.json();
    })
    .then((data) => {
      fetchEvents();
      toast.success("Event added successfully!");
    })
    .catch((error) => {
      console.error("Error adding event:", error);
      toast.error("Failed to add event");
    });
};

const deleteEvent = (id) => {
  fetch(http://localhost:9000/deleteEvent/${id}, {
    method: "DELETE",
  })
    .then((response) => {
      if (!response.ok) {
        throw new Error("Failed to delete event");
      }
      return response.json();
    })
    .then((data) => {
      fetchEvents();
      toast.info("Event deleted successfully!");
    })
    .catch((error) => {
      console.error("Error deleting event:", error);
      toast.error("Failed to delete event");
    });
};

return (
  <Box m="20px">
    <Header title="CALENDAR" subtitle="Full Calendar Interactive Page" />
    <ToastContainer />

    <Box display="flex" justifyContent="space-between">
      {/* CALENDAR SIDEBAR */}
      <Box
        flex="1 1 20%"
        backgroundColor={colors.primary[400]}
        p="15px"
        borderRadius="4px"
      >
        <Typography variant="h5">Events</Typography>
        <List>
          {currentEvents.map((event) => (
            <ListItem
              key={event._id}
              sx={{
                backgroundColor: colors.greenAccent[500],
                margin: "10px 0",
                borderRadius: "2px",
              }}
            >
              <ListItemText
                primary={event.title}
```

```
              secondary={
                <Typography>
                  {formatDate(event.date, {
                    year: "numeric",
                    month: "short",
                    day: "numeric",
                  })}
                </Typography>
              }
            />
          </ListItem>
        ))}
      </List>
    </Box>

    {/* CALENDAR */}
    <Box flex="1 1 100%" ml="15px">
      <FullCalendar
        height="75vh"
        plugins={[
          dayGridPlugin,
          timeGridPlugin,
          interactionPlugin,
          listPlugin,
        ]}
        headerToolbar={{
          left: "prev,next today",
          center: "title",
          right: "dayGridMonth,timeGridWeek,timeGridDay,listMonth",
        }}
        initialView="dayGridMonth"
        editable={true}
        selectable={true}
        selectMirror={true}
        dayMaxEvents={true}
        select={(selected) => handleDateClick(selected)}
        eventClick={handleEventClick}
        events={currentEvents.map((event) => ({
          id: event._id,
          title: event.title,
          date: event.date,
        }))}
      />
    </Box>
  </Box>
</Box>
);
};

export default Calendar;
```

CONTACTS

INDEX.JSX:

```
import React, { useEffect, useState, useMemo, useCallback } from "react";
import {
  Box,
  Button,
  Dialog,
  DialogActions,
```

```
  DialogContent,
  DialogTitle,
  TextField,
} from "@mui/material";
import { DataGrid, GridToolbar } from "@mui/x-data-grid";
import { tokens } from "../../theme";
import Header from "../../components/Header";
import { useTheme } from "@mui/material";
import { Formik, Form, Field, ErrorMessage } from "formik";
import * as yup from "yup";

const validationSchema = yup.object().shape({
  registrarId: yup.number().required("Required").integer(),
  name: yup.string().required("Required"),
  age: yup.number().required("Required").positive().integer(),
  phone: yup
    .string()
    .required("Required")
    .matches(/^[0-9]{10}$/, "Invalid phone number"),
  email: yup.string().required("Required").email("Invalid email"),
  address: yup.string().required("Required"),
  city: yup.string().required("Required"),
  zipCode: yup.string().required("Required"),
});

const Contacts = () => {
  const [contacts, setContacts] = useState([]);
  const [open, setOpen] = useState(false);
  const [currentContact, setCurrentContact] = useState(null);
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);

  const fetchContactsData = async () => {
    try {
      const response = await fetch("http://localhost:9000/getContacts"); //
      const data = await response.json();
      const formattedData = data.map((row) => ({
        ...row,
        id: row._id,
      }));
      setContacts(formattedData);
    } catch (error) {
      console.error("Failed to fetch contacts data:", error.message);
    }
  };

  const handleAdd = () => {
    setCurrentContact(null);
    setOpen(true);
  };

  const handleEdit = useCallback(
    (id) => {
      const contact = contacts.find((contact) => contact._id === id);
      setCurrentContact(contact);
      setOpen(true);
    },
    [contacts]
  );

  const handleDelete = useCallback(async (id) => {
    if (window.confirm("Do you want to delete this contact?")) {
      try {
```

```
      const response = await fetch(
        http://localhost:9000/deleteContact/${id}, //
        {
          method: "DELETE",
        }
      );

      if (!response.ok) {
        const error = await response.json();
        return alert(error.error);
      }
      setContacts((prevContacts) =>
        prevContacts.filter((contact) => contact._id !== id)
      );
      alert("Contact deleted successfully.");
    } catch (error) {
      console.error("Failed to delete contact:", error.message);
    }
  }
}, []);

const handleClose = () => {
  setOpen(false);
  setCurrentContact(null);
};

const handleFormSubmit = async (values, { resetForm }) => {
  if (currentContact) {
    try {
      const response = await fetch(
        http://localhost:9000/updateContact/${currentContact._id}, //
        {
          method: "PUT",
          headers: {
            "Content-Type": "application/json",
          },
          body: JSON.stringify(values),
        }
      );

      if (!response.ok) {
        throw new Error("Failed to update contact");
      }

      const updatedContact = await response.json();
      setContacts((prevContacts) =>
        prevContacts.map((contact) =>
          contact._id === currentContact._id ? updatedContact : contact
        )
      );

      alert("Contact updated successfully");
      window.location.reload(false);
    } catch (error) {
      console.error("Failed to update contact:", error.message);
      alert("Failed to update contact");
    }
  } else {
    try {
      const response = await fetch("http://localhost:9000/addContact", {
        //
        method: "POST",
        headers: {
```

```
          "Content-Type": "application/json",
        },
        body: JSON.stringify(values),
      });

      if (!response.ok) {
        throw new Error("Failed to add contact");
      }

      const newContact = await response.json();
      setContacts((prevContacts) => [
        ...prevContacts,
        { ...newContact, id: newContact._id },
      ]);
      alert("Contact added successfully");
      window.location.reload(false);
    } catch (error) {
      console.error("Failed to add contact:", error.message);
      alert("Failed to add contact");
    }
  }

  handleClose();
  resetForm();
};

useEffect(() => {
  fetchContactsData();
}, []);

const columns = useMemo(
  () => [
    { field: "registrarId", headerName: "RegistrarID", flex: 0.55 },
    { field: "name", headerName: "Name", flex: 0.85 },
    { field: "age", headerName: "Age", flex: 0.5 },
    { field: "phone", headerName: "Phone", flex: 0.75 },
    { field: "email", headerName: "Email", flex: 1 },
    { field: "address", headerName: "Address", flex: 1 },
    { field: "city", headerName: "City", flex: 0.65 },
    { field: "zipCode", headerName: "Zip Code", flex: 0.5 },

    {
      field: "actions",
      headerName: "Actions",
      flex: 1,
      renderCell: (params) => (
        <>
          <Button
            variant="contained"
            onClick={() => handleEdit(params.row._id)}
            sx={{
              backgroundColor: colors.blueAccent[700],
              color: colors.grey[100],
              fontSize: "10px",
              fontWeight: "bold",
              padding: "7px 7px",
              margin: "3px",
            }}
          >
            Edit
          </Button>
          <Button
            variant="contained"
```

```
            onClick={() => handleDelete(params.row._id)}
            sx={{
              backgroundColor: colors.blueAccent[700],
              color: colors.grey[100],
              fontSize: "10px",
              fontWeight: "bold",
              padding: "7px 7px",
              margin: "3px",
            }}
          >
            Delete
          </Button>
        </>
      ),
    },
  ],
  [colors, handleEdit, handleDelete]
);

return (
  <Box m="10px">
    <Box display="flex" justifyContent="space-between" alignItems="center">
      <Header
        title="CONTACTS"
        subtitle="List of Contacts for Future Reference"
      />
      <Button
        variant="contained"
        color="primary"
        onClick={handleAdd}
        sx={{
          backgroundColor: colors.blueAccent[700],
          color: colors.grey[100],
          fontSize: "14px",
          fontWeight: "bold",
          padding: "10px 20px",
        }}
      >
        Add Contact
      </Button>
    </Box>
    <Box
      m="40px 0 0 0"
      height="75vh"
      sx={{
        "& .MuiDataGrid-root": {
          border: "none",
        },
        "& .MuiDataGrid-cell": {
          borderBottom: "none",
        },
        "& .MuiDataGrid-columnHeaders": {
          backgroundColor: colors.blueAccent[700],
          borderBottom: "none",
        },
        "& .MuiDataGrid-virtualScroller": {
          backgroundColor: colors.primary[400],
        },
        "& .MuiDataGrid-footerContainer": {
          borderTop: "none",
          backgroundColor: colors.blueAccent[700],
        },
        "& .MuiCheckbox-root": {
```

```
      color: ${colors.greenAccent[200]} !important,
    },
    "& .MuiDataGrid-toolbarContainer .MuiButton-text": {
      color: ${colors.grey[100]} !important,
    },
  }}
>
  <DataGrid
    rows={contacts}
    columns={columns}
    components={{ Toolbar: GridToolbar }}
    getRowId={(row) => row._id} // Use _id as the unique id for each row
  />
</Box>
<Dialog
  open={open}
  onClose={handleClose}
  PaperProps={{
    sx: {
      backgroundColor: "black",
      color: "#CECECE",
    },
  }}
>
  <DialogTitle>
    {currentContact ? "Edit Contact" : "Add Contact"}
  </DialogTitle>
  <DialogContent>
    <Formik
      initialValues={
        currentContact || {
          _id: "",
          registrarId: "",
          name: "",
          age: "",
          phone: "",
          email: "",
          address: "",
          city: "",
          zipCode: "",
        }
      }
      validationSchema={validationSchema}
      onSubmit={handleFormSubmit}
    >
      {({ handleSubmit }) => (
        <Form onSubmit={handleSubmit} style={{ width: "350px" }}>
          <div>
            <label
              htmlFor="registrarId"
              style={{ fontSize: "1rem", justifyContent: "left" }}
            >
              RegistrarID
            </label>
            <Field
              as={TextField}
              id="registrarId"
              name="registrarId"
              placeholder="RegistrarId"
              fullWidth
              margin="normal"
            />
            <ErrorMessage
```

```
    name="registratId"
    component="div"
    style={{ color: "tomato" }}
   />
  </div>
  <div>
   <label
    htmlFor="name"
    style={{ fontSize: "1rem", justifyContent: "left" }}
   >
    Name
   </label>
   <Field
    as={TextField}
    id="name"
    name="name"
    placeholder="Name"
    fullWidth
    margin="normal"
   />
   <ErrorMessage
    name="name"
    component="div"
    style={{ color: "tomato" }}
   />
  </div>
  <div>
   <label
    htmlFor="age"
    style={{ fontSize: "1rem", justifyContent: "left" }}
   >
    Age
   </label>
   <Field
    as={TextField}
    id="age"
    name="age"
    placeholder="Age"
    fullWidth
    margin="normal"
   />
   <ErrorMessage
    name="age"
    component="div"
    style={{ color: "tomato" }}
   />
  </div>
  <div>
   <label
    htmlFor="phone"
    style={{ fontSize: "1rem", justifyContent: "left" }}
   >
    Phone
   </label>
   <Field
    as={TextField}
    id="phone"
    name="phone"
    placeholder="Phone"
    fullWidth
    margin="normal"
   />
   <ErrorMessage
```

```
    name="phone"
    component="div"
    style={{ color: "tomato" }}
   />
  </div>
  <div>
   <label
    htmlFor="email"
    style={{ fontSize: "1rem", justifyContent: "left" }}
   >
    Email
   </label>
   <Field
    as={TextField}
    id="email"
    name="email"
    placeholder="Email"
    fullWidth
    margin="normal"
   />
   <ErrorMessage
    name="email"
    component="div"
    style={{ color: "tomato" }}
   />
  </div>
  <div>
   <label
    htmlFor="address"
    style={{ fontSize: "1rem", justifyContent: "left" }}
   >
    Address
   </label>
   <Field
    as={TextField}
    id="address"
    name="address"
    placeholder="Address"
    fullWidth
    margin="normal"
   />
   <ErrorMessage
    name="address"
    component="div"
    style={{ color: "tomato" }}
   />
  </div>
  <div>
   <label
    htmlFor="city"
    style={{ fontSize: "1rem", justifyContent: "left" }}
   >
    City
   </label>
   <Field
    as={TextField}
    id="city"
    name="city"
    placeholder="City"
    fullWidth
    margin="normal"
   />
   <ErrorMessage
```

```
            name="city"
            component="div"
            style={{ color: "tomato" }}
          />
        </div>
        <div>
          <label
            htmlFor="zipCode"
            style={{ fontSize: "1rem", justifyContent: "left" }}
          >
            Zip Code
          </label>
          <Field
            as={TextField}
            id="zipCode"
            name="zipCode"
            placeholder="Zip Code"
            fullWidth
            margin="normal"
          />
          <ErrorMessage
            name="zipCode"
            component="div"
            style={{ color: "tomato" }}
          />
        </div>
        <DialogActions>
          <Button
            onClick={handleClose}
            type="button"
            variant="contained"
            color="error"
            sx={{
              backgroundColor: "#FF1744",
              color: colors.grey[100],
              fontSize: "10px",
              fontWeight: "bold",
              padding: "7px 7px",
              margin: "3px",
            }}
          >
            Cancel
          </Button>
          <Button
            type="submit"
            color="primary"
            sx={{
              backgroundColor: "#4461ED",
              color: "#BBB2B2",
              fontSize: "10px",
              fontWeight: "bold",
              padding: "7px 7px",
              margin: "3px",
            }}
          >
            {currentContact ? "Update" : "Add"}
          </Button>
        </DialogActions>
      </Form>
    )}
  </Formik>
 </DialogContent>
</Dialog>
```

```
    </Box>
  );
};

export default Contacts;
```

DASHBOARD

INDEX.JSX:

```
import { Box, Button, IconButton, Typography, useTheme } from "@mui/material";
import { tokens } from "../../theme";
import { mockTransactions } from "../../data/mockData";
import DownloadOutlinedIcon from "@mui/icons-material/DownloadOutlined";
import EmailIcon from "@mui/icons-material/Email";
import PointOfSaleIcon from "@mui/icons-material/PointOfSale";
import PersonAddIcon from "@mui/icons-material/PersonAdd";
import TrafficIcon from "@mui/icons-material/Traffic";
import Header from "../../components/Header";
import LineChart from "../../components/LineChart";
import GeographyChart from "../../components/GeographyChart";
import BarChart from "../../components/BarChart";
import StatBox from "../../components/StatBox";
import ProgressCircle from "../../components/ProgressCircle";

const Dashboard = () => {
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);

  return (
    <Box m="20px">
      {/* HEADER */}
      <Box display="flex" justifyContent="space-between" alignItems="center">
        <Header title="DASHBOARD" subtitle="Welcome to your dashboard" />

        <Box>
          <Button
            sx={{
              backgroundColor: colors.blueAccent[700],
              color: colors.grey[100],
              fontSize: "14px",
              fontWeight: "bold",
              padding: "10px 20px",
            }}
          >
            <DownloadOutlinedIcon sx={{ mr: "10px" }} />
            Download Reports
          </Button>
        </Box>
      </Box>

      {/* GRID & CHARTS */}
      <Box
        display="grid"
        gridTemplateColumns="repeat(12, 1fr)"
        gridAutoRows="140px"
        gap="20px"
      >
        {/* ROW 1 */}
        <Box
          gridColumn="span 3"
          backgroundColor={colors.primary[400]}
```

```
  display="flex"
  alignItems="center"
  justifyContent="center"
>
  <StatBox
    title="12,361"
    subtitle="Emails Sent"
    progress="0.75"
    increase="+14%"
    icon={
      <EmailIcon
        sx={{ color: colors.greenAccent[600], fontSize: "26px" }}
      />
    }
  />
</Box>
<Box
  gridColumn="span 3"
  backgroundColor={colors.primary[400]}
  display="flex"
  alignItems="center"
  justifyContent="center"
>
  <StatBox
    title="431,225"
    subtitle="Sales Obtained"
    progress="0.50"
    increase="+21%"
    icon={
      <PointOfSaleIcon
        sx={{ color: colors.greenAccent[600], fontSize: "26px" }}
      />
    }
  />
</Box>
<Box
  gridColumn="span 3"
  backgroundColor={colors.primary[400]}
  display="flex"
  alignItems="center"
  justifyContent="center"
>
  <StatBox
    title="32,441"
    subtitle="New Clients"
    progress="0.30"
    increase="+5%"
    icon={
      <PersonAddIcon
        sx={{ color: colors.greenAccent[600], fontSize: "26px" }}
      />
    }
  />
</Box>
<Box
  gridColumn="span 3"
  backgroundColor={colors.primary[400]}
  display="flex"
  alignItems="center"
  justifyContent="center"
>
  <StatBox
    title="1,325,134"
```

```
    subtitle="Traffic Received"
    progress="0.80"
    increase="+43%"
    icon={
      <TrafficIcon
        sx={{ color: colors.greenAccent[600], fontSize: "26px" }}
      />
    }
  />
</Box>

{/* ROW 2 */}
<Box
  gridColumn="span 8"
  gridRow="span 2"
  backgroundColor={colors.primary[400]}
>
  <Box
    mt="25px"
    p="0 30px"
    display="flex "
    justifyContent="space-between"
    alignItems="center"
  >
    <Box>
      <Typography
        variant="h5"
        fontWeight="600"
        color={colors.grey[100]}
      >
        Revenue Generated
      </Typography>
      <Typography
        variant="h3"
        fontWeight="bold"
        color={colors.greenAccent[500]}
      >
        $59,342.32
      </Typography>
    </Box>
    <Box>
      <IconButton>
        <DownloadOutlinedIcon
          sx={{ fontSize: "26px", color: colors.greenAccent[500] }}
        />
      </IconButton>
    </Box>
  </Box>
  <Box height="250px" m="-20px 0 0 0">
    <LineChart isDashboard={true} />
  </Box>
</Box>
<Box
  gridColumn="span 4"
  gridRow="span 2"
  backgroundColor={colors.primary[400]}
  overflow="auto"
>
  <Box
    display="flex"
    justifyContent="space-between"
    alignItems="center"
    borderBottom={4px solid ${colors.primary[500]}}
```

```
    colors={colors.grey[100]}
    p="15px"
  >
    <Typography color={colors.grey[100]} variant="h5" fontWeight="600">
      Recent Transactions
    </Typography>
  </Box>
  {mockTransactions.map((transaction, i) => (
    <Box
      key={${transaction.txId}-${i}}
      display="flex"
      justifyContent="space-between"
      alignItems="center"
      borderBottom={4px solid ${colors.primary[500]}}
      p="15px"
    >
      <Box>
        <Typography
          color={colors.greenAccent[500]}
          variant="h5"
          fontWeight="600"
        >
          {transaction.txId}
        </Typography>
        <Typography color={colors.grey[100]}>
          {transaction.user}
        </Typography>
      </Box>
      <Box color={colors.grey[100]}>{transaction.date}</Box>
      <Box
        backgroundColor={colors.greenAccent[500]}
        p="5px 10px"
        borderRadius="4px"
      >
        ${transaction.cost}
      </Box>
    </Box>
  ))}
</Box>

{/* ROW 3 */}
<Box
  gridColumn="span 4"
  gridRow="span 2"
  backgroundColor={colors.primary[400]}
  p="30px"
>
  <Typography variant="h5" fontWeight="600">
    Campaign
  </Typography>
  <Box
    display="flex"
    flexDirection="column"
    alignItems="center"
    mt="25px"
  >
    <ProgressCircle size="125" />
    <Typography
      variant="h5"
      color={colors.greenAccent[500]}
      sx={{ mt: "15px" }}
    >
      $48,352 revenue generated
```

```
            </Typography>
            <Typography>Includes extra misc expenditures and costs</Typography>
          </Box>
        </Box>
        <Box
          gridColumn="span 4"
          gridRow="span 2"
          backgroundColor={colors.primary[400]}
        >
          <Typography
            variant="h5"
            fontWeight="600"
            sx={{ padding: "30px 30px 0 30px" }}
          >
            Sales Quantity
          </Typography>
          <Box height="250px" mt="-20px">
            <BarChart isDashboard={true} />
          </Box>
        </Box>
        <Box
          gridColumn="span 4"
          gridRow="span 2"
          backgroundColor={colors.primary[400]}
          padding="30px"
        >
          <Typography
            variant="h5"
            fontWeight="600"
            sx={{ marginBottom: "15px" }}
          >
            Geography Based Traffic
          </Typography>
          <Box height="200px">
            <GeographyChart isDashboard={true} />
          </Box>
        </Box>
      </Box>
    </Box>
  );
};

export default Dashboard;
```

GEOGRAPHY:

INDEX.JSX:

```
import React from "react";
import { Box, useTheme } from "@mui/material";
import Header from "../../components/Header";
import { ResponsiveChoropleth } from "@nivo/geo";
import { tokens } from "../../theme";
import { geoFeatures } from "../../data/mockGeoFeatures";
import { mockGeographyData as data } from "../../data/mockData";

const Geography = () => {
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);

  return (
    <Box m="20px">
```

```
<Header title="GEOGRAPHY" subtitle="Find where your users are located" />

<Box
  height="75vh"
  border={1px solid ${colors.grey[100]}}
  borderRadius="4px"
>
  <ResponsiveChoropleth
    data={data}
    theme={{
      axis: {
        domain: {
          line: {
            stroke: colors.grey[100],
          },
        },
        legend: {
          text: {
            fill: colors.grey[100],
          },
        },
        ticks: {
          line: {
            stroke: colors.grey[100],
            strokeWidth: 1,
          },
          text: {
            fill: colors.grey[100],
          },
        },
      },
      legends: {
        text: {
          fill: colors.grey[100],
        },
      },
    }}
    features={geoFeatures.features}
    margin={{ top: 0, right: 0, bottom: 0, left: 0 }}
    domain={[0, 1000000]}
    unknownColor="#666666"
    label="properties.name"
    valueFormat=".2s"
    projectionScale={150}
    projectionTranslation={[0.5, 0.5]}
    projectionRotation={[0, 0, 0]}
    borderWidth={1.5}
    borderColor="#ffffff"
    legends={[
      {
        anchor: "bottom-left",
        direction: "column",
        justify: true,
        translateX: 20,
        translateY: -100,
        itemsSpacing: 0,
        itemWidth: 94,
        itemHeight: 18,
        itemDirection: "left-to-right",
        itemTextColor: colors.grey[100],
        itemOpacity: 0.85,
        symbolSize: 18,
        effects: [
```

```
            {
              on: "hover",
              style: {
                itemTextColor: "#ffffff",
                itemOpacity: 1,
              },
            },
          ],
        },
      ]}
    />
   </Box>
  </Box>
 );
};

export default Geography;
```

GLOBAL

SIDEBAR.JSX:

```
import { useEffect, useState } from "react";
import { ProSidebar, Menu, MenuItem } from "react-pro-sidebar";
import { Box, IconButton, Typography, useTheme } from "@mui/material";
import { Link, useNavigate } from "react-router-dom";
import "react-pro-sidebar/dist/css/styles.css";
import { tokens } from "../../theme";
import HomeOutlinedIcon from "@mui/icons-material/HomeOutlined";
import PeopleOutlinedIcon from "@mui/icons-material/PeopleOutlined";
import ContactsOutlinedIcon from "@mui/icons-material/ContactsOutlined";
import ReceiptOutlinedIcon from "@mui/icons-material/ReceiptOutlined";
import CalendarTodayOutlinedIcon from "@mui/icons-material/CalendarTodayOutlined";
import BarChartOutlinedIcon from "@mui/icons-material/BarChartOutlined";
import PieChartOutlineOutlinedIcon from "@mui/icons-material/PieChartOutlineOutlined";
import TimelineOutlinedIcon from "@mui/icons-material/TimelineOutlined";
import MenuOutlinedIcon from "@mui/icons-material/MenuOutlined";
import MapOutlinedIcon from "@mui/icons-material/MapOutlined";
import LocalShippingOutlinedIcon from "@mui/icons-material/LocalShippingOutlined";

const Item = ({ title, to, icon, selected, setSelected }) => {
 const theme = useTheme();
 const colors = tokens(theme.palette.mode);
 return (
  <MenuItem
   active={selected === title}
   style={{
    color: colors.grey[100],
   }}
   onClick={() => setSelected(title)}
   icon={icon}
  >
   <Typography>{title}</Typography>
   <Link to={to} />
  </MenuItem>
 );
};

const Sidebar = () => {
 const theme = useTheme();
 const colors = tokens(theme.palette.mode);
 const [isCollapsed, setIsCollapsed] = useState(false);
```

```
const [selected, setSelected] = useState("Dashboard");

const name = localStorage.getItem("Name");
const role = localStorage.getItem("Role");
const navigate = useNavigate();

useEffect(() => {
  if (!name || !role) {
    navigate("/");
  }
}, []);

return (
  <Box
    sx={{
      "& .pro-sidebar-inner": {
        background: ${colors.primary[400]} !important,
      },
      "& .pro-icon-wrapper": {
        backgroundColor: "transparent !important",
      },
      "& .pro-inner-item": {
        padding: "5px 35px 5px 20px !important",
      },
      "& .pro-inner-item:hover": {
        color: "#868dfb !important",
      },
      "& .pro-menu-item.active": {
        color: "#6870fa !important",
      },
    }}
  >
    <ProSidebar collapsed={isCollapsed}>
      <Menu iconShape="square">
        {/* LOGO AND MENU ICON */}
        <MenuItem
          onClick={() => setIsCollapsed(!isCollapsed)}
          icon={isCollapsed ? <MenuOutlinedIcon /> : undefined}
          style={{
            margin: "10px 0 20px 0",
            color: colors.grey[100],
          }}
        >
          {!isCollapsed && (
            <Box
              display="flex"
              justifyContent="space-between"
              alignItems="center"
              ml="15px"
            >
              <Typography variant="h3" color={colors.grey[100]}>
                <b>CLEVER-DASH</b>
              </Typography>
              <IconButton onClick={() => setIsCollapsed(!isCollapsed)}>
                <MenuOutlinedIcon />
              </IconButton>
            </Box>
          )}
        </MenuItem>

        {!isCollapsed && (
          <Box mb="25px">
            <Box display="flex" justifyContent="center" alignItems="center">
```

```
      <img
        alt="profile-user"
        width="100px"
        height="100px"
        src={../../assets/profile.png}
        style={{ cursor: "pointer", borderRadius: "50%" }}
      />
    </Box>
    <Box textAlign="center">
      <Typography
        variant="h2"
        color={colors.grey[100]}
        fontWeight="bold"
        sx={{ m: "10px 0 0 0" }}
      >
        {name}
      </Typography>
      <Typography variant="h5" color={colors.greenAccent[500]}>
        {role}
      </Typography>
    </Box>
  </Box>
)}

<Box paddingLeft={isCollapsed ? undefined : "10%"}>
  <Item
    title="Dashboard"
    to="/dashboard"
    icon={<HomeOutlinedIcon />}
    selected={selected}
    setSelected={setSelected}
  />

  <Typography
    variant="h6"
    color={colors.grey[300]}
    sx={{ m: "15px 0 5px 20px" }}
  >
    Data
  </Typography>
  <Item
    title="Manage Team"
    to="/team"
    icon={<PeopleOutlinedIcon />}
    selected={selected}
    setSelected={setSelected}
  />
  <Item
    title="Contacts Information"
    to="/contacts"
    icon={<ContactsOutlinedIcon />}
    selected={selected}
    setSelected={setSelected}
  />
  <Item
    title="Invoices Balances"
    to="/invoices"
    icon={<ReceiptOutlinedIcon />}
    selected={selected}
    setSelected={setSelected}
  />

  <Typography
```

```
            variant="h6"
            color={colors.grey[300]}
            sx={{ m: "15px 0 5px 20px" }}
          >
            Pages
          </Typography>
          <Item
            title="Products"
            to="/products"
            icon={<LocalShippingOutlinedIcon />}
            selected={selected}
            setSelected={setSelected}
          />
          <Item
            title="Calendar"
            to="/calendar"
            icon={<CalendarTodayOutlinedIcon />}
            selected={selected}
            setSelected={setSelected}
          />
          <Typography
            variant="h6"
            color={colors.grey[300]}
            sx={{ m: "15px 0 5px 20px" }}
          >
            Charts
          </Typography>
          <Item
            title="Bar Chart"
            to="/bar"
            icon={<BarChartOutlinedIcon />}
            selected={selected}
            setSelected={setSelected}
          />
          <Item
            title="Pie Chart"
            to="/pie"
            icon={<PieChartOutlineOutlinedIcon />}
            selected={selected}
            setSelected={setSelected}
          />
          <Item
            title="Line Chart"
            to="/line"
            icon={<TimelineOutlinedIcon />}
            selected={selected}
            setSelected={setSelected}
          />
          <Item
            title="Geography Chart"
            to="/geography"
            icon={<MapOutlinedIcon />}
            selected={selected}
            setSelected={setSelected}
          />
        </Box>
      </Menu>
    </ProSidebar>
  </Box>
 );
};

export default Sidebar;
```

TOPBAR.JSX:

```
import { Box, IconButton, useTheme, Menu, MenuItem } from "@mui/material";
import { useContext, useState } from "react";
import { ColorModeContext, tokens } from "../../theme";
import InputBase from "@mui/material/InputBase";
import LightModeOutlinedIcon from "@mui/icons-material/LightModeOutlined";
import DarkModeOutlinedIcon from "@mui/icons-material/DarkModeOutlined";
import NotificationsOutlinedIcon from "@mui/icons-material/NotificationsOutlined";
import SettingsOutlinedIcon from "@mui/icons-material/SettingsOutlined";
import PersonOutlinedIcon from "@mui/icons-material/PersonOutlined";
import SearchIcon from "@mui/icons-material/Search";
import { useNavigate } from "react-router-dom"; // Import useNavigate

const Topbar = ({ onLogout }) => {
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);
  const colorMode = useContext(ColorModeContext);
  const [menuAnchor, setMenuAnchor] = useState(null);
  const navigate = useNavigate(); // Initialize useNavigate
  const [anchorEl, setAnchorEl] = useState(null);

  const handleMenuClick = (event) => {
    setMenuAnchor(event.currentTarget);
  };

  const handleMenuClose = () => {
    setMenuAnchor(null);
  };

  const handleLogout = () => {
    // Call logout handler passed from App.js
    onLogout();
    // Redirect to authentication page
    navigate("/");
  };
  const handleClick = (event) => setAnchorEl(event.currentTarget);
  const handleClose = () => {
    localStorage.clear();
    setAnchorEl(null);
    // navigate("/");
  };

  return (
    <Box display="flex" justifyContent="space-between" p={2}>
      {/* SEARCH BAR */}
      <Box
        display="flex"
        backgroundColor={colors.primary[400]}
        borderRadius="3px"
      >
        <InputBase sx={{ ml: 2, flex: 1 }} placeholder="Search" />
        <IconButton type="button" sx={{ p: 1 }}>
          <SearchIcon />
        </IconButton>
      </Box>
      {/* ICONS */}
      <Box display="flex">
```

```
        <IconButton onClick={colorMode.toggleColorMode}>
          {theme.palette.mode === "dark" ? (
            <DarkModeOutlinedIcon />
          ) : (
            <LightModeOutlinedIcon />
          )}
        </IconButton>
        <IconButton>
          <NotificationsOutlinedIcon />
        </IconButton>
        <IconButton>
          <SettingsOutlinedIcon />
        </IconButton>
        {/* Menu Button */}
        <IconButton onClick={handleMenuClick}>
          <PersonOutlinedIcon />
        </IconButton>
        {/* Menu */}
        <Menu
          anchorEl={menuAnchor}
          open={Boolean(menuAnchor)}
          onClose={handleMenuClose}
        >
          <MenuItem
            onClick={() => {
              handleClose();
              navigate("/");
            }}
          >
            Log Out
          </MenuItem>
        </Menu>
      </Box>
    </Box>
  );
};

export default Topbar;
```

INVOICES
INDEX.JSX:

```
import React, { useState, useEffect } from "react";
import {
  Box,
  Typography,
  Button,
  useTheme,
  Dialog,
  DialogTitle,
  DialogContent,
  DialogActions,
  TextField,
} from "@mui/material";
import { DataGrid, GridToolbar } from "@mui/x-data-grid";
import { DatePicker } from "@mui/lab";
import { tokens } from "../../theme";
import Header from "../../components/Header";
import { Formik, Form, Field, ErrorMessage } from "formik";
import * as yup from "yup";

const validationSchema = yup.object({
  name: yup.string().required("Name is required"),
```

```
  email: yup
    .string()
    .required("Email is required")
    .email("Invalid email format"),
  cost: yup.number().required("Cost is required").positive(),
  phone: yup.number().required("Phone number is required"),
  dueDate: yup.date().required("Date is required"),
});

const Invoices = () => {
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);
  const [invoices, setInvoices] = useState([]);
  const [open, setOpen] = useState(false);
  const [currentInvoice, setCurrentInvoice] = useState(null);

  useEffect(() => {
    // Fetch invoices data
    fetchInvoices();
  }, []);

  const fetchInvoices = async () => {
    try {
      const response = await fetch("http://localhost:9000/getInvoices");
      const data = await response.json();
      // Ensure the _id is used as the id for each row
      const formattedData = data.map((row) => ({
        ...row,
        id: row._id,
      }));
      setInvoices(formattedData);
    } catch (error) {
      console.error("Failed to fetch invoices:", error.message);
    }
  };

  const handleAdd = () => {
    setCurrentInvoice(null);
    setOpen(true);
  };

  const handleEdit = (id) => {
    const invoice = invoices.find((invoice) => invoice._id === id);
    setCurrentInvoice(invoice);
    setOpen(true);
  };

  const handleClose = () => {
    setOpen(false);
    setCurrentInvoice(null);
  };

  const handleDelete = async (id) => {
    if (window.confirm("Do you want to delete this invoice?")) {
      try {
        const response = await fetch(
          http://localhost:9000/deleteInvoice/${id},
          {
            method: "DELETE",
          }
        );

        if (!response.ok) {
```

```
        const error = await response.json();
        return alert(error.error);
      }
      setInvoices((prevInvoices) =>
        prevInvoices.filter((invoice) => invoice._id !== id)
      );
      alert("Invoice deleted successfully.");
    } catch (error) {
      console.error("Failed to delete invoice:", error.message);
    }
  }
};

const handleFormSubmit = async (values, { resetForm }) => {
  if (currentInvoice) {
    try {
      const response = await fetch(
        http://localhost:9000/updateInvoice/${currentInvoice._id},
        {
          method: "PUT",
          headers: {
            "Content-Type": "application/json",
          },
          body: JSON.stringify(values),
        }
      );

      if (!response.ok) {
        throw new Error("Failed to update invoice");
      }

      const updatedInvoice = await response.json();
      setInvoices((prevInvoices) =>
        prevInvoices.map((invoice) =>
          invoice._id === currentInvoice._id ? updatedInvoice : invoice
        )
      );

      alert("Invoice updated successfully");
      window.location.reload(false);
    } catch (error) {
      console.error("Failed to update invoice:", error.message);
      alert("Failed to update invoice");
    }
  } else {
    try {
      const response = await fetch("http://localhost:9000/addInvoice", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify(values),
      });

      if (!response.ok) {
        throw new Error("Failed to add invoice");
      }

      const newInvoice = await response.json();
      setInvoices((prevInvoices) => [
        ...prevInvoices,
        { ...newInvoice, id: newInvoice._id },
      ]);
```

```
      alert("Invoice added successfully");
      window.location.reload(false);
    } catch (error) {
      console.error("Failed to add invoice:", error.message);
      alert("Failed to add invoice");
    }
  }

  handleClose();
  resetForm();
};

const columns = [
  { field: "name", headerName: "Name", flex: 0.4 },
  { field: "phone", headerName: "Phone Number", flex: 0.3 },
  { field: "email", headerName: "Email", flex: 0.5 },
  {
    field: "cost",
    headerName: "Cost",
    flex: 0.3,
    minWidth: 100,
    renderCell: ({ row: { cost } }) => (
      <Typography color={colors.greenAccent[500]}>${cost}</Typography>
    ),
  },
  { field: "dueDate", headerName: "Due Date", flex: 0.3, minWidth: 100 },
  {
    field: "actions",
    headerName: "Actions",
    flex: 0.3,
    renderCell: (params) => (
      <>
        <Button
          variant="contained"
          onClick={() => handleEdit(params.row._id)}
          sx={{
            backgroundColor: colors.blueAccent[700],
            color: colors.grey[100],
            fontSize: "10px",
            fontWeight: "bold",
            padding: "7px 7px",
            margin: "3px",
          }}
        >
          Edit
        </Button>
        <Button
          variant="contained"
          onClick={() => handleDelete(params.row._id)}
          sx={{
            backgroundColor: colors.blueAccent[700],
            color: colors.grey[100],
            fontSize: "10px",
            fontWeight: "bold",
            padding: "7px 7px",
            margin: "3px",
          }}
        >
          Delete
        </Button>
      </>
    ),
  },
```

```
];

return (
  <Box m="10px">
    <Box display="flex" justifyContent="space-between" alignItems="center">
      <Header title="INVOICES" subtitle="List of Invoice Balances" />
      <Button
        variant="contained"
        color="primary"
        onClick={handleAdd}
        sx={{
          backgroundColor: colors.blueAccent[700],
          color: colors.grey[100],
          fontSize: "14px",
          fontWeight: "bold",
          padding: "10px 20px",
        }}
      >
        Add Invoice
      </Button>
    </Box>
    <Box
      m="40px 0 0 0"
      height="75vh"
      sx={{
        "& .MuiDataGrid-root": {
          border: "none",
        },
        "& .MuiDataGrid-cell": {
          borderBottom: "none",
        },
        "& .MuiDataGrid-columnHeaders": {
          backgroundColor: colors.blueAccent[700],
          borderBottom: "none",
        },
        "& .MuiDataGrid-virtualScroller": {
          backgroundColor: colors.primary[400],
        },
        "& .MuiDataGrid-footerContainer": {
          borderTop: "none",
          backgroundColor: colors.blueAccent[700],
        },
        "& .MuiCheckbox-root": {
          color: ${colors.greenAccent[200]} !important,
        },
        "& .MuiDataGrid-toolbarContainer .MuiButton-text": {
          color: ${colors.grey[100]} !important,
        },
      }}
    >
      <DataGrid
        rows={invoices}
        columns={columns}
        components={{ Toolbar: GridToolbar }}
        getRowId={(row) => row._id} // Use _id as the unique id for each row
      />
    </Box>
    <Dialog
      open={open}
      onClose={handleClose}
      PaperProps={{
        sx: {
          backgroundColor: "black",
```

```
      color: "#CECECE",
    },
  }}
>
  <DialogTitle>
    {currentInvoice ? "Edit Invoice" : "Add Invoice"}
  </DialogTitle>
  <DialogContent>
    <Formik
      initialValues={
        currentInvoice || {
          _id: "",
          name: "",
          email: "",
          cost: "",
          phone: "",
          dueDate: "",
        }
      }
      validationSchema={validationSchema}
      onSubmit={handleFormSubmit}
    >
      {({ handleSubmit }) => (
        <Form onSubmit={handleSubmit} style={{ width: "350px" }}>
          <div>
            <label
              htmlFor="name"
              style={{ fontSize: "1rem", justifyContent: "left" }}
            >
              Name
            </label>
            <Field
              as={TextField}
              id="name"
              name="name"
              placeholder="Name"
              fullWidth
              margin="normal"
            />
            <ErrorMessage
              name="name"
              component="div"
              style={{ color: "tomato" }}
            />
          </div>
          <div>
            <label
              htmlFor="email"
              style={{ fontSize: "1rem", justifyContent: "left" }}
            >
              Email
            </label>
            <Field
              as={TextField}
              id="email"
              name="email"
              placeholder="Email"
              fullWidth
              margin="normal"
            />
            <ErrorMessage
              name="email"
              component="div"
```

```
    style={{ color: "tomato" }}
   />
  </div>
  <div>
   <label
    htmlFor="cost"
    style={{ fontSize: "1rem", justifyContent: "left" }}
   >
    Cost
   </label>
   <Field
    as={TextField}
    id="cost"
    name="cost"
    placeholder="Cost"
    fullWidth
    margin="normal"
   />
   <ErrorMessage
    name="cost"
    component="div"
    style={{ color: "tomato" }}
   />
  </div>
  <div>
   <label
    htmlFor="phone"
    style={{ fontSize: "1rem", justifyContent: "left" }}
   >
    Phone
   </label>
   <Field
    as={TextField}
    id="phone"
    name="phone"
    placeholder="Phone"
    fullWidth
    margin="normal"
   />
   <ErrorMessage
    name="phone"
    component="div"
    style={{ color: "tomato" }}
   />
  </div>
  <div>
   <label
    htmlFor="dueDate"
    style={{ fontSize: "1rem", justifyContent: "left" }}
   >
    Due Date
   </label>
   <Field
    as={TextField}
    id="dueDate"
    name="dueDate"
    type="date"
    placeholder="Due Date"
    fullWidth
    margin="normal"
   />
   <ErrorMessage
    name="dueDate"
```

```
                component="div"
                style={{ color: "tomato" }}
              />
            </div>
            <DialogActions>
              <Button
                type="button"
                variant="contained"
                onClick={handleClose}
                color="error"
                sx={{
                  backgroundColor: "#FF1744",
                  color: colors.grey[100],
                  fontSize: "10px",
                  fontWeight: "bold",
                  padding: "7px 7px",
                  margin: "3px",
                }}
              >
                Cancel
              </Button>
              <Button
                type="submit"
                color="primary"
                sx={{
                  backgroundColor: "#4461ED",
                  color: "#BBB2B2",
                  fontSize: "10px",
                  fontWeight: "bold",
                  padding: "7px 7px",
                  margin: "3px",
                }}
              >
                {currentInvoice ? "Update" : "Add"}
              </Button>
            </DialogActions>
          </Form>
        )}
      </Formik>
    </DialogContent>
  </Dialog>
 </Box>
 );
};

export default Invoices;
```

LAYOUT
INDEX.JSX:

```
import React, { useState } from "react";
import { Box, useMediaQuery } from "@mui/material";
import { Outlet } from "react-router-dom";
import { useSelector } from "react-redux";
import Sidebar from "../global/Sidebar";
import Topbar from "../global/Topbar";
// import { useGetUserQuery } from "state/api";

// Layout
const Layout = () => {
 // is desktop
 const isNonMobile = useMediaQuery("(min-width: 600px)");
 // is sidebar open
```

```
  const [isSidebarOpen, setIsSidebarOpen] = useState(true);
  // get user id from redux selector
  //   const userId = useSelector((state) => state.global.userId);
  // get data
  //   const { data } = useGetUserQuery(userId);

  return (
    <Box display={isNonMobile ? "flex" : "block"} width="100%" height="100%">
      {/* Sidebar */}
      <Sidebar
        // user={data || {}}
        isNonMobile={isNonMobile}
        drawerWidth="250px"
        isSidebarOpen={isSidebarOpen}
        setIsSidebarOpen={setIsSidebarOpen}
      />

      {/* Navbar */}
      <Box flexGrow={1}>
        <Topbar
          //   user={data || {}}
          isSidebarOpen={isSidebarOpen}
          setIsSidebarOpen={setIsSidebarOpen}
        />
        <Outlet />
      </Box>
    </Box>
  );
};

export default Layout;
```

LINE
INDEX.JSX:

```
import React from "react";
import { Box } from "@mui/material";
import Header from "../../components/Header";
import { ResponsiveLine } from "@nivo/line";
import { useTheme } from "@mui/material";
import { tokens } from "../../theme";
import { mockLineData as data } from "../../data/mockData";

const Line = () => {
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);

  return (
    <Box m="20px">
      <Header title="LINE CHART" subtitle="Track your Revenue" />
      <Box height="75vh">
        <ResponsiveLine
          data={data}
          theme={{
            axis: {
              domain: {
                line: {
                  stroke: colors.grey[100],
                },
              },
              legend: {
                text: {
                  fill: colors.grey[100],
```

```
      },
    },
    ticks: {
      line: {
        stroke: colors.grey[100],
        strokeWidth: 1,
      },
      text: {
        fill: colors.grey[100],
      },
    },
  },
  legends: {
    text: {
      fill: colors.grey[100],
    },
  },
  tooltip: {
    container: {
      color: colors.primary[500],
    },
  },
}}
colors={{ scheme: "nivo" }}
margin={{ top: 50, right: 110, bottom: 50, left: 60 }}
xScale={{ type: "point" }}
yScale={{
  type: "linear",
  min: "auto",
  max: "auto",
  stacked: true,
  reverse: false,
}}
yFormat=" >-.2f"
curve="catmullRom"
axisTop={null}
axisRight={null}
axisBottom={{
  orient: "bottom",
  tickSize: 0,
  tickPadding: 5,
  tickRotation: 0,
  legend: "Month",
  legendOffset: 36,
  legendPosition: "middle",
}}
axisLeft={{
  orient: "left",
  tickValues: 5,
  tickSize: 3,
  tickPadding: 5,
  tickRotation: 0,
  legend: "Sales(in $)",
  legendOffset: -40,
  legendPosition: "middle",
}}
enableGridX={false}
enableGridY={false}
pointSize={8}
pointColor={{ theme: "background" }}
pointBorderWidth={2}
pointBorderColor={{ from: "serieColor" }}
pointLabelYOffset={-12}
```

```
        useMesh={true}
        legends={[
          {
            anchor: "bottom-right",
            direction: "column",
            justify: false,
            translateX: 100,
            translateY: 0,
            itemsSpacing: 0,
            itemDirection: "left-to-right",
            itemWidth: 80,
            itemHeight: 20,
            itemOpacity: 0.75,
            symbolSize: 12,
            symbolShape: "circle",
            symbolBorderColor: "rgba(0, 0, 0, .5)",
            effects: [
              {
                on: "hover",
                style: {
                  itemBackground: "rgba(0, 0, 0, .03)",
                  itemOpacity: 1,
                },
              },
            ],
          },
        ]}
      />
    </Box>
  </Box>
 );
};

export default Line;
```

PIE
INDEX.JSX:

```
import React from "react";
import { Box } from "@mui/material";
import Header from "../../components/Header";
import { ResponsivePie } from "@nivo/pie";
import { useTheme } from "@mui/material";
import { tokens } from "../../theme";
import { mockPieData as data } from "../../data/mockData";

const Pie = () => {
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);

  return (
    <Box m="20px">
      <Header title="PIE CHART" subtitle="Track your product demand" />
      <Box height="75vh">
        <ResponsivePie
          data={data}
          theme={{
            axis: {
              domain: {
                line: {
                  stroke: colors.grey[100],
                },
              },
```

```
    legend: {
      text: {
        fill: colors.grey[100],
      },
    },
    ticks: {
      line: {
        stroke: colors.grey[100],
        strokeWidth: 1,
      },
      text: {
        fill: colors.grey[100],
      },
    },
  },
  legends: {
    text: {
      fill: colors.grey[100],
    },
  },
}}
margin={{ top: 40, right: 80, bottom: 80, left: 80 }}
innerRadius={0.5}
padAngle={0.7}
cornerRadius={3}
activeOuterRadiusOffset={8}
borderColor={{
  from: "color",
  modifiers: [["darker", 0.2]],
}}
arcLinkLabelsSkipAngle={10}
arcLinkLabelsTextColor={colors.grey[100]}
arcLinkLabelsThickness={2}
arcLinkLabelsColor={{ from: "color" }}
enableArcLabels={false}
arcLabelsRadiusOffset={0.4}
arcLabelsSkipAngle={7}
arcLabelsTextColor={{
  from: "color",
  modifiers: [["darker", 2]],
}}
defs={[
  {
    id: "dots",
    type: "patternDots",
    background: "inherit",
    color: "rgba(255, 255, 255, 0.3)",
    size: 4,
    padding: 1,
    stagger: true,
  },
  {
    id: "lines",
    type: "patternLines",
    background: "inherit",
    color: "rgba(255, 255, 255, 0.3)",
    rotation: -45,
    lineWidth: 6,
    spacing: 10,
  },
]}
legends={[
  {
```

```
            anchor: "bottom",
            direction: "row",
            justify: false,
            translateX: 0,
            translateY: 56,
            itemsSpacing: 0,
            itemWidth: 100,
            itemHeight: 18,
            itemTextColor: "#999",
            itemDirection: "left-to-right",
            itemOpacity: 1,
            symbolSize: 18,
            symbolShape: "circle",
            effects: [
              {
                on: "hover",
                style: {
                  itemTextColor: "#000",
                },
              },
            ],
          },
        ]}
      />
    </Box>
  </Box>
 );
};

export default Pie;
```

PRODUCTS
INDEX.JSX:

```
import React, { useEffect, useState } from "react";
import {
  Box,
  Button,
  Card,
  CardActions,
  CardContent,
  Collapse,
  Dialog,
  DialogContent,
  DialogTitle,
  MenuItem,
  TextField,
  Typography,
  Rating,
} from "@mui/material";
import { useTheme } from "@mui/material/styles";
import { tokens } from "../../theme";
import Header from "../../components/Header";
import * as yup from "yup";
import { Formik, Form, Field, ErrorMessage } from "formik";

const validationSchema = yup.object().shape({
  name: yup.string().required("Name is required"),
  price: yup.number().required("Price is required"),
  description: yup.string().required("Description is required"),
  rating: yup.number().required("Rating is required").min(0).max(5),
  category: yup
    .string()
    .oneOf(
```

```
      [
        "Category_A",
        "Category_B",
        "Category_C",
        "Category_D",
        "Category_E",
        "Category_F",
      ],
      "Invalid Category"
    )
    .required("Category is required"),
  supply: yup.number().required("Supply is required"),
  yearlySalesTotal: yup.number().required("Yearly Sales Total is required"),
  yearlyTotalSoldUnits: yup
    .number()
    .required("Yearly Total Sold Units is required"),
});

const Products = () => {
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);
  const [products, setProducts] = useState([]);
  const [open, setOpen] = useState(false);
  const [currentProduct, setCurrentProduct] = useState(null);
  const [expandedId, setExpandedId] = useState(null);

  const fetchProductData = async () => {
    try {
      const response = await fetch("http://localhost:9000/getProducts");
      const data = await response.json();
      setProducts(data);
    } catch (error) {
      console.error("Failed to fetch product data:", error.message);
    }
  };

  useEffect(() => {
    fetchProductData();
  }, []);

  const handleAdd = () => {
    setCurrentProduct(null);
    setOpen(true);
  };

  const handleClose = () => {
    setOpen(false);
    setCurrentProduct(null);
  };

  const handleFormSubmit = async (values) => {
    const productData = {
      ...values,
      stat: [
        {
          yearlySalesTotal: values.yearlySalesTotal,
          yearlyTotalSoldUnits: values.yearlyTotalSoldUnits,
        },
      ],
    };

    if (currentProduct && currentProduct._id !== null) {
      // Update product
```

```
    try {
      const response = await fetch(
        http://localhost:9000/updateProduct/${currentProduct._id},
        {
          method: "PUT",
          headers: {
            "Content-Type": "application/json",
          },
          body: JSON.stringify(productData),
        }
      );

      if (!response.ok) {
        throw new Error("Failed to update product");
      }

      const updatedProduct = await response.json();
      setProducts((prevProducts) =>
        prevProducts.map((product) =>
          product._id === currentProduct._id ? updatedProduct : product
        )
      );

      alert("Product updated successfully");
      window.location.reload(false);
    } catch (error) {
      console.error("Failed to update product:", error.message);
      alert("Failed to update product");
    }
  } else {
    // Add product
    try {
      const response = await fetch("http://localhost:9000/addProduct", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify(productData),
      });

      if (!response.ok) {
        throw new Error("Failed to add product");
      }

      // Refetch product data to update the state with the latest data
      fetchProductData();

      alert("Product added successfully");
    } catch (error) {
      console.error("Failed to add product:", error.message);
      alert("Failed to add product");
    }
  }

  handleClose();
};

const handleEdit = (product) => {
  setCurrentProduct(product);
  setOpen(true);
};

const handleDelete = async (productId) => {
```

```
  if (window.confirm("Do you want to delete this product?")) {
    try {
      const response = await fetch(
        http://localhost:9000/deleteProduct/${productId},
        {
          method: "DELETE",
        }
      );

      if (!response.ok) {
        const error = await response.json();
        return alert(error.error);
      }

      // Refetch product data to update the state with the latest data
      fetchProductData();

      alert("Product deleted successfully.");
    } catch (error) {
      console.error("Failed to delete product:", error.message);
      alert("Failed to delete product");
    }
  }
};

const toggleExpanded = (id) => {
  setExpandedId(expandedId === id ? null : id);
};

return (
  <Box m="1.5rem 2.5rem">
    <Box display="flex" justifyContent="space-between" alignItems="center">
      <Header title="PRODUCTS" subtitle="See your list of products." />
      <Button
        variant="contained"
        color="primary"
        onClick={handleAdd}
        sx={{
          backgroundColor: colors.blueAccent[700],
          color: colors.grey[100],
          fontSize: "14px",
          fontWeight: "bold",
          padding: "10px 20px",
        }}
      >
        Add Product
      </Button>
    </Box>
    <Box
      mt="20px"
      display="grid"
      gridTemplateColumns="repeat(4, minmax(0, 1fr))"
      justifyContent="space-between"
      rowGap="20px"
      columnGap="1.33%"
    >
      {products.map((product) => (
        <Card
          key={product._id}
          sx={{
            backgroundColor: colors.blueAccent[600],
            borderRadius: "0.55rem",
            "&:hover": {
```

```
      boxShadow: 0 4px 20px 0 ${colors.shadow},
    },
  }}
>
  <CardContent>
    {/* Category */}
    <Typography
      sx={{ fontSize: 14 }}
      color={theme.palette.secondary[700]}
      gutterBottom
    >
      {product.category || "N/A"}
    </Typography>

    {/* Name */}
    <Typography variant="h5" component="div">
      {product.name || "N/A"}
    </Typography>

    {/* Price */}
    <Typography
      sx={{ mb: "1.5rem" }}
      color={theme.palette.secondary[400]}
    >
      ${product.price ? Number(product.price).toFixed(2) : "N/A"}
    </Typography>

    {/* Rating */}
    <Box display="flex" alignItems="center">
      <Typography>Rating:</Typography>
      <Rating
        value={product.rating}
        readOnly
        precision={0.5}
        sx={{ fontSize: "1.2rem", ml: 0.5 }}
      />
    </Box>

    {/* Description */}
    <Typography variant="body2">
      {product.description || "No description available"}
    </Typography>
  </CardContent>

  {/* Actions */}
  <CardActions>
    <Button
      variant="contained"
      color="warning"
      size="small"
      onClick={() => toggleExpanded(product._id)}
    >
      {expandedId === product._id ? "See Less" : "See More"}
    </Button>
    <Button
      variant="contained"
      color="secondary"
      size="small"
      onClick={() => handleEdit(product)}
    >
      Edit
    </Button>
    <Button
```

```
            variant="contained"
            color="error"
            size="small"
            onClick={() => handleDelete(product._id)}
          >
            Delete
          </Button>
        </CardActions>

        {/* Extra Info */}
        <Collapse
          in={expandedId === product._id}
          timeout="auto"
          unmountOnExit
        >
          <CardContent>
            <Typography>ID: {product._id}</Typography>
            <Typography>Supply: {product.supply}</Typography>
            <Typography>
              Yearly Sales Total: {product.stat[0].yearlySalesTotal}
            </Typography>
            <Typography>
              Yearly Total Sold Units:{" "}
              {product.stat[0].yearlyTotalSoldUnits}
            </Typography>
          </CardContent>
        </Collapse>
      </Card>
    ))}
  </Box>

  {/* Dialog for adding/editing product */}
  <Dialog open={open} onClose={handleClose}>
    <DialogTitle>
      {currentProduct && currentProduct._id !== null
        ? "Edit Product"
        : "Add Product"}
    </DialogTitle>
    <DialogContent>
      <Formik
        initialValues={{
          _id: currentProduct ? currentProduct._id : null,
          name: currentProduct ? currentProduct.name : "",
          price: currentProduct ? currentProduct.price : "",
          description: currentProduct ? currentProduct.description : "",
          rating: currentProduct ? currentProduct.rating : 0,
          category: currentProduct ? currentProduct.category : "",
          supply: currentProduct ? currentProduct.supply : "",
          yearlySalesTotal: currentProduct
            ? currentProduct.stat[0].yearlySalesTotal
            : "",
          yearlyTotalSoldUnits: currentProduct
            ? currentProduct.stat[0].yearlyTotalSoldUnits
            : "",
        }}
        validationSchema={validationSchema}
        onSubmit={handleFormSubmit}
      >
        {({ errors, touched }) => (
          <Form style={{ width: "350px" }}>
            <label
              htmlFor="name"
              style={{ fontSize: "1rem", justifyContent: "left" }}
```

```
>
  Name
</label>
<Field
  as={TextField}
  name="name"
  label="Name"
  fullWidth
  margin="normal"
  error={touched.name && !!errors.name}
  helperText={<ErrorMessage name="name" />}
/>
<label
  htmlFor="price"
  style={{ fontSize: "1rem", justifyContent: "left" }}
>
  Price
</label>
<Field
  as={TextField}
  name="price"
  label="Price"
  type="number"
  fullWidth
  margin="normal"
  error={touched.price && !!errors.price}
  helperText={<ErrorMessage name="price" />}
/>
<label
  htmlFor="description"
  style={{ fontSize: "1rem", justifyContent: "left" }}
>
  Description
</label>
<Field
  as={TextField}
  name="description"
  label="Description"
  fullWidth
  margin="normal"
  multiline
  rows={3}
  error={touched.description && !!errors.description}
  helperText={<ErrorMessage name="description" />}
/>
<label
  htmlFor="rating"
  style={{ fontSize: "1rem", justifyContent: "left" }}
>
  Rating
</label>
<Field
  as={TextField}
  name="rating"
  label="Rating"
  type="number"
  fullWidth
  margin="normal"
  inputProps={{ step: 0.5, min: 0, max: 5 }}
  error={touched.rating && !!errors.rating}
  helperText={<ErrorMessage name="rating" />}
/>
<label
```

```
  htmlFor="category"
  style={{ fontSize: "1rem", justifyContent: "left" }}
>
  Category
</label>
<Field
  as={TextField}
  name="category"
  label="Category"
  fullWidth
  margin="normal"
  select
  error={touched.category && !!errors.category}
  helperText={<ErrorMessage name="category" />}
>
  {[
    "Category_A",
    "Category_B",
    "Category_C",
    "Category_D",
    "Category_E",
    "Category_F",
  ].map((option) => (
    <MenuItem key={option} value={option}>
      {option}
    </MenuItem>
  ))}
</Field>
<label
  htmlFor="supply"
  style={{ fontSize: "1rem", justifyContent: "left" }}
>
  Supply
</label>
<Field
  as={TextField}
  name="supply"
  label="Supply"
  type="number"
  fullWidth
  margin="normal"
  error={touched.supply && !!errors.supply}
  helperText={<ErrorMessage name="supply" />}
/>
<label
  htmlFor="yearlySalesTotal"
  style={{ fontSize: "1rem", justifyContent: "left" }}
>
  Yearly Sales Total
</label>
<Field
  as={TextField}
  name="yearlySalesTotal"
  label="Yearly Sales Total"
  type="number"
  fullWidth
  margin="normal"
  error={touched.yearlySalesTotal && !!errors.yearlySalesTotal}
  helperText={<ErrorMessage name="yearlySalesTotal" />}
/>
<label
  htmlFor="yearlyTotalSoldUnits"
  style={{ fontSize: "1rem", justifyContent: "left" }}
```

```
            >
              Yearly Total Sold Units
            </label>
            <Field
              as={TextField}
              name="yearlyTotalSoldUnits"
              label="Yearly Total Sold Units"
              type="number"
              fullWidth
              margin="normal"
              error={
                touched.yearlyTotalSoldUnits &&
                !!errors.yearlyTotalSoldUnits
              }
              helperText={<ErrorMessage name="yearlyTotalSoldUnits" />}
            />
            <Box mt={2}>
              <Button
                type="submit"
                variant="contained"
                color="primary"
                fullWidth
                sx={{
                  backgroundColor: "#4461ED",
                  color: "#BBB2B2",
                  fontSize: "10px",
                  fontWeight: "bold",
                  padding: "7px 7px",
                  margin: "3px",
                }}
              >
                Submit
              </Button>
            </Box>
          </Form>
        )}
      </Formik>
    </DialogContent>
  </Dialog>
 </Box>
);
};

export default Products;
```

TEAM
INDEX.JSX:

```
import React, { useEffect, useState, useMemo, useCallback } from "react";
import {
  Box,
  Button,
  Dialog,
  DialogTitle,
  DialogContent,
  DialogActions,
  TextField,
  Select,
  MenuItem,
} from "@mui/material";
import { DataGrid, GridToolbar } from "@mui/x-data-grid";
import { useTheme } from "@mui/material/styles";
import { tokens } from "../../theme";
```

```
import AdminPanelSettingsOutlinedIcon from "@mui/icons-material/AdminPanelSettingsOutlined";
import SecurityOutlinedIcon from "@mui/icons-material/SecurityOutlined";
import LockOpenOutlinedIcon from "@mui/icons-material/LockOpenOutlined";
import Typography from "@mui/material/Typography";
import * as yup from "yup";
import { Formik, Form, Field, ErrorMessage } from "formik";
import Header from "../../components/Header";

const validationSchema = yup.object().shape({
  name: yup.string().required("Required"),
  age: yup.number().required("Required").positive().integer(),
  phone: yup
    .string()
    .required("Required")
    .matches(/^[0-9]{10}$/, "Invalid phone number"),
  email: yup.string().required("Required").email("Invalid email"),
  accessLevel: yup.string().required("Required"),
});

const Team = () => {
  const [team, setTeam] = useState([]);
  const [open, setOpen] = useState(false);
  const [currentMember, setCurrentMember] = useState(null);
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);

  const fetchTeamData = async () => {
    try {
      const response = await fetch("http://localhost:9000/getMembers");
      const data = await response.json();
      const formattedData = data.map((row) => ({
        ...row,
        id: row._id,
      }));
      setTeam(formattedData);
    } catch (error) {
      console.error("Failed to fetch team data:", error.message);
    }
  };

  const handleAdd = () => {
    setCurrentMember(null);
    setOpen(true);
  };

  const handleEdit = useCallback(
    (id) => {
      const member = team.find((member) => member._id === id);
      setCurrentMember(member);
      setOpen(true);
    },
    [team]
  );

  const handleDelete = useCallback(async (id) => {
    if (window.confirm("Do you want to delete this member?")) {
      try {
        const response = await fetch(
          http://localhost:9000/deleteMember/${id},
          {
            method: "DELETE",
          }
        );
```

```
      if (!response.ok) {
        const error = await response.json();
        return alert(error.error);
      }
      setTeam((prevTeam) => prevTeam.filter((member) => member._id !== id));
      alert("Member deleted successfully.");
    } catch (error) {
      console.error("Failed to delete member:", error.message);
    }
  }
}, []);

const handleClose = () => {
  setOpen(false);
  setCurrentMember(null);
};

const handleFormSubmit = async (values, { resetForm }) => {
  if (currentMember) {
    try {
      const response = await fetch(
        http://localhost:9000/updateMember/${currentMember._id},
        {
          method: "PUT",
          headers: {
            "Content-Type": "application/json",
          },
          body: JSON.stringify(values),
        }
      );

      if (!response.ok) {
        throw new Error("Failed to update member");
      }

      const updatedMember = await response.json();
      setTeam((prevTeam) =>
        prevTeam.map((member) =>
          member._id === currentMember._id ? updatedMember : member
        )
      );

      alert("Member updated successfully");
      window.location.reload(false);
    } catch (error) {
      console.error("Failed to update member:", error.message);
      alert("Failed to update member");
    }
  } else {
    try {
      const response = await fetch("http://localhost:9000/addMember", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify(values),
      });

      if (!response.ok) {
        throw new Error("Failed to add member");
      }
```

```
      const newMember = await response.json();
      setTeam((prevTeam) => [
        ...prevTeam,
        { ...newMember, id: newMember._id },
      ]);
      alert("Member added successfully");
      window.location.reload(false);
    } catch (error) {
      console.error("Failed to add member:", error.message);
      alert("Failed to add member");
    }
  }

  handleClose();
  resetForm();
};

useEffect(() => {
  fetchTeamData();
}, []);

const columns = useMemo(
  () => [
    { field: "name", headerName: "Name", flex: 1 },
    { field: "age", headerName: "Age", flex: 0.5 },
    { field: "phone", headerName: "Phone", flex: 1 },
    { field: "email", headerName: "Email", flex: 1 },
    // { field: "accessLevel", headerName: "Access Level", flex: 1 },
    {
      field: "accessLevel",
      headerName: "Access Level",
      flex: 1,
      minWidth: 100,
      renderCell: ({ row: { accessLevel } }) => {
        return (
          <Box
            width="100%"
            p="5px"
            display="flex"
            justifyContent="center"
            alignItems="center"
            backgroundColor={
              accessLevel === "Admin"
                ? colors.greenAccent[600]
                : accessLevel === "Manager"
                ? colors.greenAccent[600]
                : colors.greenAccent[600]
            }
            borderRadius="4px"
          >
            {accessLevel === "Admin" && <AdminPanelSettingsOutlinedIcon />}
            {accessLevel === "Manager" && <SecurityOutlinedIcon />}
            {accessLevel === "User" && <LockOpenOutlinedIcon />}
            <Typography color={colors.grey[100]} sx={{ ml: "5px" }}>
              {accessLevel}
            </Typography>
          </Box>
        );
      },
    },

    {
      field: "actions",
```

```
      headerName: "Actions",
      flex: 1,
      renderCell: (params) => (
        <>
          <Button
            variant="contained"
            onClick={() => handleEdit(params.row._id)}
            sx={{
              backgroundColor: colors.blueAccent[700],
              color: colors.grey[100],
              fontSize: "10px",
              fontWeight: "bold",
              padding: "7px 7px",
              margin: "3px",
            }}
          >
            Edit
          </Button>
          <Button
            variant="contained"
            onClick={() => handleDelete(params.row._id)}
            sx={{
              backgroundColor: colors.blueAccent[700],
              color: colors.grey[100],
              fontSize: "10px",
              fontWeight: "bold",
              padding: "7px 7px",
              margin: "3px",
            }}
          >
            Delete
          </Button>
        </>
      ),
    },
  ],
  [colors, handleEdit, handleDelete]
);

return (
  <Box m="10px">
    <Box display="flex" justifyContent="space-between" alignItems="center">
      <Header title="TEAM" subtitle="Managing the Team Members" />
      <Button
        variant="contained"
        color="primary"
        onClick={handleAdd}
        sx={{
          backgroundColor: colors.blueAccent[700],
          color: colors.grey[100],
          fontSize: "14px",
          fontWeight: "bold",
          padding: "10px 20px",
        }}
      >
        Add Member
      </Button>
    </Box>
    <Box
      m="40px 0 0 0"
      height="75vh"
      sx={{
        "& .MuiDataGrid-root": {
```

```
      border: "none",
    },
    "& .MuiDataGrid-cell": {
      borderBottom: "none",
    },
    "& .MuiDataGrid-columnHeaders": {
      backgroundColor: colors.blueAccent[700],
      borderBottom: "none",
    },
    "& .MuiDataGrid-virtualScroller": {
      backgroundColor: colors.primary[400],
    },
    "& .MuiDataGrid-footerContainer": {
      borderTop: "none",
      backgroundColor: colors.blueAccent[700],
    },
    "& .MuiCheckbox-root": {
      color: ${colors.greenAccent[200]} !important,
    },
    "& .MuiDataGrid-toolbarContainer .MuiButton-text": {
      color: ${colors.grey[100]} !important,
    },
  }}
>
  <DataGrid
    rows={team}
    columns={columns}
    components={{ Toolbar: GridToolbar }}
    getRowId={(row) => row._id}
  />
</Box>
<Dialog
  open={open}
  onClose={handleClose}
  PaperProps={{
    sx: {
      backgroundColor: "black",
      color: "#CECECE",
    },
  }}
>
  <DialogTitle>
    {currentMember ? "Edit Member" : "Add Member"}
  </DialogTitle>
  <DialogContent>
    <Formik
      initialValues={
        currentMember || {
          _id: "",
          name: "",
          age: "",
          phone: "",
          email: "",
          accessLevel: "",
        }
      }
      validationSchema={validationSchema}
      onSubmit={handleFormSubmit}
    >
      {({ handleSubmit }) => (
        <Form onSubmit={handleSubmit} style={{ width: "350px" }}>
          <div>
            <label
```

```
          htmlFor="name"
          style={{ fontSize: "1rem", justifyContent: "left" }}
        >
          Name
        </label>
        <Field
          as={TextField}
          id="name"
          name="name"
          placeholder="Name"
          fullWidth
          margin="normal"
        />
        <ErrorMessage
          name="name"
          component="div"
          style={{ color: "tomato" }}
        />
      </div>
      <div>
        <label
          htmlFor="age"
          style={{ fontSize: "1rem", justifyContent: "left" }}
        >
          Age
        </label>
        <Field
          as={TextField}
          id="age"
          name="age"
          placeholder="Age"
          type="number"
          fullWidth
          margin="normal"
        />
        <ErrorMessage
          name="age"
          component="div"
          style={{ color: "tomato" }}
        />
      </div>
      <div>
        <label
          htmlFor="phone"
          style={{ fontSize: "1rem", justifyContent: "left" }}
        >
          Phone
        </label>
        <Field
          as={TextField}
          id="phone"
          name="phone"
          placeholder="Phone"
          fullWidth
          margin="normal"
        />
        <ErrorMessage
          name="phone"
          component="div"
          style={{ color: "tomato" }}
        />
      </div>
      <div>
```

```
  <label
    htmlFor="email"
    style={{ fontSize: "1rem", justifyContent: "left" }}
  >
    Email
  </label>
  <Field
    as={TextField}
    id="email"
    name="email"
    placeholder="Email"
    type="email"
    fullWidth
    margin="normal"
  />
  <ErrorMessage
    name="email"
    component="div"
    style={{ color: "tomato" }}
  />
</div>
<div>
  <label
    htmlFor="accessLevel"
    style={{ fontSize: "1rem", justifyContent: "left" }}
  >
    Access Level
  </label>
  <Field
    as={Select}
    id="accessLevel"
    name="accessLevel"
    fullWidth
    margin="normal"
    placeholder="Select Access level"
  >
    <MenuItem value="" disabled>
      Select Access Level
    </MenuItem>
    <MenuItem value="Admin">Admin</MenuItem>
    <MenuItem value="Manager">Manager</MenuItem>
    <MenuItem value="User">User</MenuItem>
  </Field>
  <ErrorMessage
    name="accessLevel"
    component="div"
    style={{ color: "tomato" }}
  />
</div>
<Box mt={2} display="flex" justifyContent="space-between">
  <DialogActions>
    <Button
      type="button"
      variant="contained"
      onClick={handleClose}
      color="error"
      sx={{
        backgroundColor: "#FF1744",
        color: colors.grey[100],
        fontSize: "10px",
        fontWeight: "bold",
        padding: "7px 7px",
        margin: "3px",
```

```
                }}
              >
                Cancel
              </Button>
              <Button
                type="submit"
                color="primary"
                sx={{
                  backgroundColor: "#4461ED",
                  color: "#BBB2B2",
                  fontSize: "10px",
                  fontWeight: "bold",
                  padding: "7px 7px",
                  margin: "3px",
                }}
              >
                {currentMember ? "Update" : "Add"}
              </Button>
            </DialogActions>
          </Box>
        </Form>
      )}
    </Formik>
  </DialogContent>
 </Dialog>
</Box>
);
};

export default Team;
```

FOLDER: SERVER(BACKEND):

MODEL.JS:

```
const mongoose = require("mongoose");
const barSchema = new mongoose.Schema({
  country: { type: String, required: true },
  Category_A: { type: Number, required: true },
  Category_B: { type: Number, required: true },
  Category_C: { type: Number, required: true },
  Category_D: { type: Number, required: true },
  Category_E: { type: Number, required: true },
  Category_F: { type: Number, required: true },
});

const Bar = mongoose.model("Bar", barSchema);

const calendarSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
  },
  date: {
    type: Date,
    required: true,
  }, // Change type to Date to match mock data
});

const Calendar = mongoose.model("Calendar", calendarSchema);

const contactSchema = new mongoose.Schema({
  registrarId: { type: Number, required: true },
  name: { type: String, required: true },
  age: { type: Number, required: true },
  phone: { type: String, required: true },
  email: { type: String, required: true },
  address: { type: String, required: true },
  city: { type: String, required: true },
  zipCode: { type: String, required: true },
});

const Contact = mongoose.model("Contact", contactSchema);

const geographySchema = new mongoose.Schema({
  id: { type: String, required: true },
  value: { type: Number, required: true },
});
```

```
const Geography = mongoose.model("Geography", geographySchema);

const invoiceSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true },
  cost: { type: Number, required: true },
  phone: { type: String, required: true },
  dueDate: { type: Date, required: true }, // Added dueDate field
});

const Invoice = mongoose.model("Invoice", invoiceSchema);

const lineDataSchema = new mongoose.Schema({
  x: { type: String, required: true }, // Change field name to 'x' to match data
  y: { type: Number, required: true },
});

const lineSchema = new mongoose.Schema({
  id: { type: String, required: true }, // Change type to String
  color: { type: String, required: true },
  data: [lineDataSchema], // Nest lineDataSchema for 'data' field
});

const Line = mongoose.model("Line", lineSchema);
const pieSchema = new mongoose.Schema({
  id: { type: String, required: true },
  label: { type: String, required: true },
  value: { type: Number, required: true },
  color: { type: String, required: true },
});

const Pie = mongoose.model("Pie", pieSchema);

const productStatSchema = new mongoose.Schema({
  yearlySalesTotal: { type: Number, required: true },
  yearlyTotalSoldUnits: { type: Number, required: true },
});

const productSchema = new mongoose.Schema({
  name: { type: String, required: true },
  price: { type: Number, required: true },
  description: { type: String, required: true },
  rating: { type: Number, required: true, min: 0, max: 5 },
  category: {
    type: String,
    enum: [
```

```
      "Category_A",
      "Category_B",
      "Category_C",
      "Category_D",
      "Category_E",
      "Category_F",
    ],
    default: "Category_A",
    required: true,
  },
  supply: { type: Number, required: true },
  stat: [productStatSchema], // Use productStatSchema for 'stat' array
});

const Product = mongoose.model("Product", productSchema);

const teamSchema = new mongoose.Schema({
  name: { type: String, required: true },
  age: { type: Number },
  phone: { type: String },
  email: { type: String },
  accessLevel: {
    type: String,
    enum: ["Admin", "User", "Manager"],
    default: "User",
  },
});

const Team = mongoose.model("Team", teamSchema);
module.exports = {
  Bar,
  Calendar,
  Contact,
  Geography,
  Invoice,
  Line,
  Pie,
  Product,
  Team,
};
```

INDEX.JS:

```
const express = require("express");
const bodyParser = require("body-parser");
var mongoose = require("mongoose");
path = require("path"); //Added Code
const cors = require("cors");
const {
  Bar,
  Calendar,
  Contact,
  Geography,
  Invoice,
  Line,
  Pie,
  Product,
  Team,
} = require("./models/model.js");
// Configuration

const app = express();

app.use(express.json());
app.use(
  cors({
    // Allow requests from multiple origins, including your GitHub Pages site
    origin: [
      "http://localhost:3000", // Your development origin // Your GitHub Pages site
      ,
    ],
    optionsSuccessStatus: 200,
    methods: ["GET", "POST", "PUT", "DELETE"], // Methods you want to allow
    allowedHeaders: ["Content-Type", "Authorization"], // Headers to allow
    credentials: true, // If you want to allow cookies/credentials
  })
);
const URI =
  "mongodb+srv://subhapreetpatro2004:Patro202172112@cluster0.bjkxtm6.mongodb.net/Clever-
dash?retryWrites=true&w=majority&appName=Cluster0";
const PORT = 9000; //process.env.PORT ||
mongoose
  .connect(URI, {})
  .then(() => {
    console.log("MongoDB successfully connected");
  })
  .catch((error) => console.log(${error} did not connect.));

app.post("/addMember", async (req, res) => {
  const { name, email, age, phone, accessLevel } = req.body;

  if (!name || !email || !age || !phone || !accessLevel) {
    return res
```

```
      .status(422)
      .json({ error: "Please fill all the fields properly!!!" });
  }

  try {
    // Check if employee with the given empid already exists
    // Create a new employee document
    const newMember = new Team({
      name,
      email,
      age,
      phone,
      accessLevel,
    });

    // Save the new employee document to the database
    await newMember.save();
    console.log("New Member Registered Successfully...");
    // Send success response
    return res.status(201).json({ message: "Member registered successfully." });
  } catch (error) {
    console.log(error.message);
    return res.status(500).json({ error: "Internal server error." });
  }
});

app.post("/addContact", async (req, res) => {
  const { registrarId, name, email, age, phone, address, city, zipCode } =
    req.body;

  if (
    !registrarId ||
    !name ||
    !email ||
    !age ||
    !phone ||
    !address ||
    !city ||
    !zipCode
  ) {
    return res
      .status(422)
      .json({ error: "Please fill all the fields properly!!!" });
  }

  try {
    // Check if employee with the given empid already exists

    // Create a new employee document
    const newContact = new Contact({
      registrarId,
      name,
      email,
```

```
      age,
      phone,
      address,
      city,
      zipCode,
    });

    // Save the new employee document to the database
    await newContact.save();
    console.log("New Contact Registered Successfully...");
    // Send success response
    return res.status(201).json({ message: "Member registered successfully." });
  } catch (error) {
    console.log(error.message);
    return res.status(500).json({ error: "Internal server error." });
  }
});

app.post("/addInvoice", async (req, res) => {
  const { name, email, cost, phone, dueDate } = req.body;

  if (!name || !email || !cost || !phone || !dueDate) {
    return res
      .status(422)
      .json({ error: "Please fill all the fields properly!!!" });
  }

  try {
    // Check if employee with the given empid already exists

    // Create a new employee document
    const newInvoice = new Invoice({
      name,
      email,
      cost,
      phone,
      dueDate,
    });

    // Save the new employee document to the database
    await newInvoice.save();
    console.log("New Invoice Registered Successfully...");
    // Send success response
    return res
      .status(201)
      .json({ message: "Invoice registered successfully." });
  } catch (error) {
    console.log(error.message);
    return res.status(500).json({ error: "Internal server error." });
  }
});

app.post("/addProduct", async (req, res) => {
```

```
  const { name, price, description, rating, category, supply, stat } = req.body;

  if (
    !name ||
    !price ||
    !description ||
    !rating ||
    !category ||
    !supply ||
    !stat
  ) {
    return res
      .status(422)
      .json({ error: "Please fill all the fields properly!!!" });
  }

  try {
    const newProduct = new Product({
      name,
      price,
      description,
      rating,
      category,
      supply,
      stat,
    });

    await newProduct.save();
    console.log("New Product Registered Successfully...");
    return res
      .status(201)
      .json({ message: "Product registered successfully." });
  } catch (error) {
    console.log(error.message);
    return res.status(500).json({ error: "Internal server error." });
  }
});

app.post("/addEvent", async (req, res) => {
  const { title, date } = req.body;
  try {
    const newEvent = new Calendar({ title, date }); // Instantiate a new Calendar event
    await newEvent.save(); // Save the new event to the database
    res.status(201).json(newEvent); // Return the newly created event as JSON response
  } catch (error) {
    console.error("Error adding event:", error);
    res.status(500).json({ error: "Failed to add event" }); // Handle error if saving fails
  }
});

//implementing get route to get employees data
app.get("/getMembers", async (req, res) => {
  try {
```

```
    const allemployees = await Team.find();
    return res.status(200).json(allemployees);
  } catch (error) {
    console.log(error.message);
    return res.status(500).json({ error: "Internal server error" });
  }
});

app.get("/getContacts", async (req, res) => {
  try {
    const allContacts = await Contact.find();
    return res.status(200).json(allContacts);
  } catch (error) {
    console.log(error.message);
    return res.status(500).json({ error: "Internal server error" });
  }
});

app.get("/getInvoices", async (req, res) => {
  try {
    const allInvoices = await Invoice.find();
    return res.status(200).json(allInvoices);
  } catch (error) {
    console.log(error.message);
    return res.status(500).json({ error: "Internal server error" });
  }
});

app.get("/getProducts", async (req, res) => {
  try {
    const allProducts = await Product.find();
    return res.status(200).json(allProducts);
  } catch (error) {
    console.log(error.message);
    return res.status(500).json({ error: "Internal server error" });
  }
});

app.get("/getEvents", async (req, res) => {
  try {
    const allEvents = await Calendar.find();
    return res.status(200).json(allEvents);
  } catch (error) {
    console.error(error.message);
    return res.status(500).json({ error: "Internal server error" });
  }
});

// //specific id based search to get specific employee data
app.get("/getMember/:id", async (req, res) => {
  try {
    const memberId = req.params.id;
    const memberData = await Team.findOne({ _id: memberId });
```

```
    if (!memberData) {
      return res.status(404).json({ error: "Member not found" });
    }

    return res.status(200).json(memberData);
  } catch (error) {
    console.log(error.message);
    return res.status(500).json({ error: "Internal server error" });
  }
});

app.get("/getContact/:id", async (req, res) => {
  try {
    const contactId = req.params.id;
    const contactData = await Contact.findOne({ _id: contactId });

    if (!contactData) {
      return res.status(404).json({ error: "Contact not found" });
    }

    return res.status(200).json(contactData);
  } catch (error) {
    console.log(error.message);
    return res.status(500).json({ error: "Internal server error" });
  }
});

app.get("/getInvoice/:id", async (req, res) => {
  try {
    const invoiceId = req.params.id;
    const invoiceData = await Invoice.findOne({ _id: invoiceId });

    if (!invoiceData) {
      return res.status(404).json({ error: "Invoice not found" });
    }

    return res.status(200).json(invoiceData);
  } catch (error) {
    console.log(error.message);
    return res.status(500).json({ error: "Internal server error" });
  }
});

app.get("/getProduct/:id", async (req, res) => {
  try {
    const productId = req.params.id;
    const productData = await Product.findOne({ _id: productId });

    if (!productData) {
      return res.status(404).json({ error: "Product not found" });
    }
```

```
    return res.status(200).json(productData);
  } catch (error) {
    console.log(error.message);
    return res.status(500).json({ error: "Internal server error" });
  }
});

app.get("/getEvent/:id", async (req, res) => {
  try {
    const eventId = req.params.id;
    const event = await Calendar.findOne({ _id: eventId });

    if (!event) {
      return res.status(404).json({ error: "Event not found" });
    }

    return res.status(200).json(event);
  } catch (error) {
    console.error(error.message);
    return res.status(500).json({ error: "Internal server error" });
  }
});

// // Update Employee Details based on specific ID
app.put("/updateMember/:id", async (req, res) => {
  try {
    // Find the member by its ID and update it
    let member = await Team.findOneAndUpdate({ _id: req.params.id }, req.body, {
      new: true,
    });

    // If member not found, return 404
    if (!member) {
      return res.status(404).json({ error: "Member not found" });
    }

    // Return success response with the updated member
    return res.json({ message: "Member updated successfully", member });
  } catch (error) {
    // Log and return internal server error
    console.error("Error updating member:", error.message);
    return res.status(500).json({ error: "Internal server error" });
  }
});

app.put("/updateContact/:id", async (req, res) => {
  try {
    // Find the member by its ID and update it
    let contact = await Contact.findOneAndUpdate(
      { _id: req.params.id },
      req.body,
      {
        new: true,
```

```
      }
    );

    // If member not found, return 404
    if (!contact) {
      return res.status(404).json({ error: "Contact not found" });
    }

    // Return success response with the updated member
    return res.json({ message: "Contact updated successfully", contact });
  } catch (error) {
    // Log and return internal server error
    console.error("Error updating contact:", error.message);
    return res.status(500).json({ error: "Internal server error" });
  }
});

app.put("/updateInvoice/:id", async (req, res) => {
  try {
    // Find the member by its ID and update it
    let invoice = await Invoice.findOneAndUpdate(
      { _id: req.params.id },
      req.body,
      {
        new: true,
      }
    );

    // If member not found, return 404
    if (!invoice) {
      return res.status(404).json({ error: "Invoice not found" });
    }

    // Return success response with the updated member
    return res.json({ message: "Invoice updated successfully", invoice });
  } catch (error) {
    // Log and return internal server error
    console.error("Error updating invoice:", error.message);
    return res.status(500).json({ error: "Internal server error" });
  }
});

app.put("/updateProduct/:id", async (req, res) => {
  try {
    const productId = req.params.id;
    const updatedData = req.body;

    const updatedProduct = await Product.findOneAndUpdate(
      { _id: productId },
      updatedData,
      { new: true }
    );
```

```
    if (!updatedProduct) {
      return res.status(404).json({ error: "Product not found" });
    }

    return res
      .status(200)
      .json({ message: "Product updated successfully", updatedProduct });
  } catch (error) {
    console.error("Error updating product:", error.message);
    return res.status(500).json({ error: "Internal server error" });
  }
});

app.put("/updateEvent/:id", async (req, res) => {
  try {
    const eventId = req.params.id;
    const updatedData = req.body;

    const updatedEvent = await Calendar.findOneAndUpdate(
      { _id: eventId },
      updatedData,
      { new: true }
    );

    if (!updatedEvent) {
      return res.status(404).json({ error: "Event not found" });
    }

    return res
      .status(200)
      .json({ message: "Event updated successfully", updatedEvent });
  } catch (error) {
    console.error("Error updating event:", error.message);
    return res.status(500).json({ error: "Internal server error" });
  }
});

// DELETE endpoint to delete a single document by ID
// If empid is your unique identifier, query by it
app.delete("/deleteMember/:id", async (req, res) => {
  const Memberid = req.params.id;
  console.log("Deleting employee with empid:", Memberid);

  try {
    const deleteMember = await Team.deleteOne({ _id: Memberid });
    if (deleteMember.deletedCount === 0) {
      console.log("Employee not found with empid:", Memberid);
      return res.status(404).json({ error: "Employee not found" });
    }
    console.log("Employee deleted successfully");
    return res.status(200).json({ message: "Employee deleted successfully" });
  } catch (error) {
    console.error("Error deleting employee:", error.message);
```

```
    return res.status(500).json({ error: "Internal server error" });
  }
});

app.delete("/deleteContact/:id", async (req, res) => {
  const contactId = req.params.id;
  console.log("Deleting contact with id:", contactId);

  try {
    const deleteContact = await Contact.deleteOne({ _id: contactId });
    if (deleteContact.deletedCount === 0) {
      console.log("Contact not found with empid:", contactId);
      return res.status(404).json({ error: "Contact not found" });
    }
    console.log("Contact deleted successfully");
    return res.status(200).json({ message: "Contact deleted successfully" });
  } catch (error) {
    console.error("Error deleting contact:", error.message);
    return res.status(500).json({ error: "Internal server error" });
  }
});

app.delete("/deleteInvoice/:id", async (req, res) => {
  const invoiceId = req.params.id;
  console.log("Deleting invoice with id:", invoiceId);

  try {
    const deleteInvoice = await Invoice.deleteOne({ _id: invoiceId });
    if (deleteInvoice.deletedCount === 0) {
      console.log("Invoice not found with id:", invoiceId);
      return res.status(404).json({ error: "Invoice not found" });
    }
    console.log("Invoice deleted successfully");
    return res.status(200).json({ message: "Invoice deleted successfully" });
  } catch (error) {
    console.error("Error deleting invoice:", error.message);
    return res.status(500).json({ error: "Internal server error" });
  }
});

app.delete("/deleteProduct/:id", async (req, res) => {
  try {
    const productId = req.params.id;
    const deleteProduct = await Product.deleteOne({ _id: productId });

    if (deleteProduct.deletedCount === 0) {
      return res.status(404).json({ error: "Product not found" });
    }

    return res.status(200).json({ message: "Product deleted successfully" });
  } catch (error) {
    console.error("Error deleting product:", error.message);
    return res.status(500).json({ error: "Internal server error" });
```

```
  }
});

app.delete("/deleteEvent/:id", async (req, res) => {
  try {
    const eventId = req.params.id;
    const deleteEvent = await Calendar.deleteOne({ _id: eventId });

    if (deleteEvent.deletedCount === 0) {
      return res.status(404).json({ error: "Event not found" });
    }

    return res.status(200).json({ message: "Event deleted successfully" });
  } catch (error) {
    console.error("Error deleting event:", error.message);
    return res.status(500).json({ error: "Internal server error" });
  }
});

app.listen(PORT, (error) => {
  if (error) {
    console.log("Failed to connect server");
  } else {
    console.log(Server started and Server running on ${PORT});
  }
});
```

Screenshots of Application

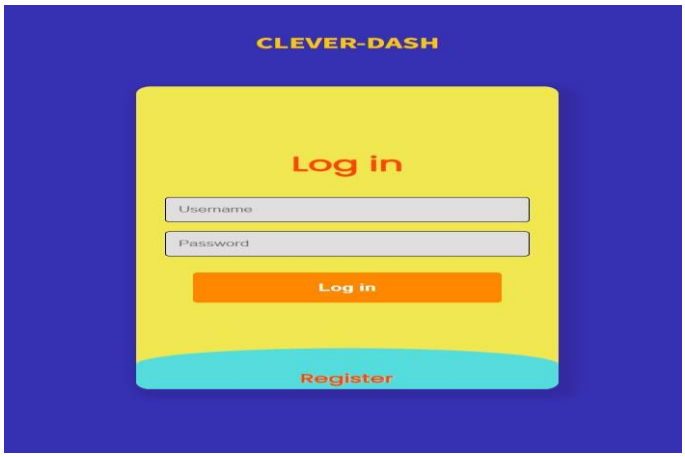

Fig 5.2 Login page

Fig 5.3Registration page

Fig 5.4 Dashboard

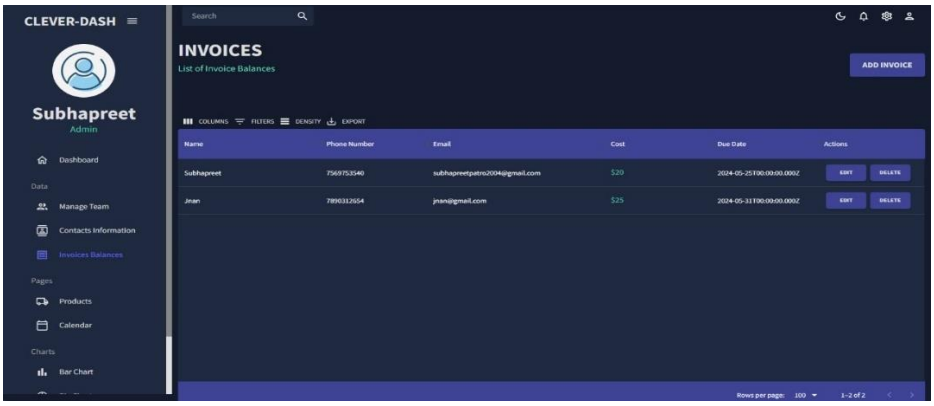

Fig 5.5 Invoices pages

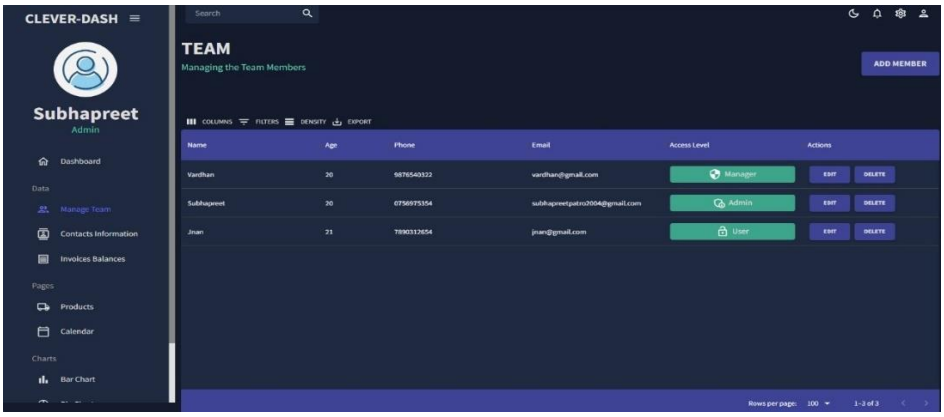

Fig 5.6 Teams page

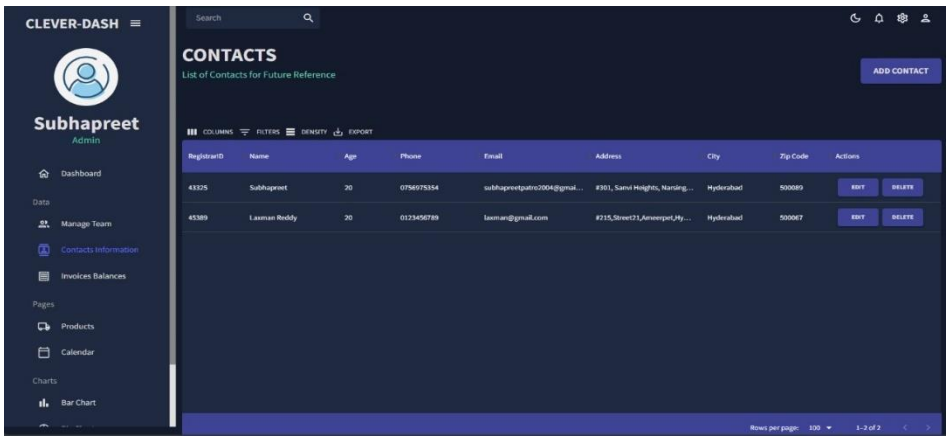

Fig 5.7 Contact page

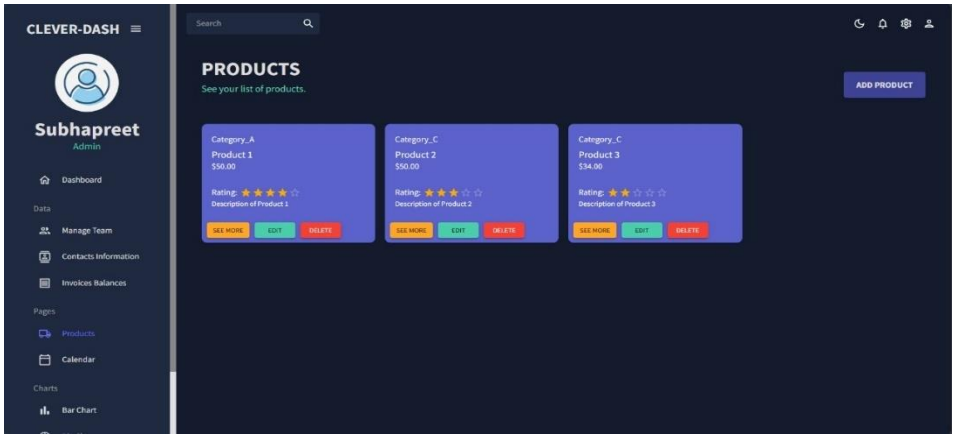

Fig 5.8 Products page

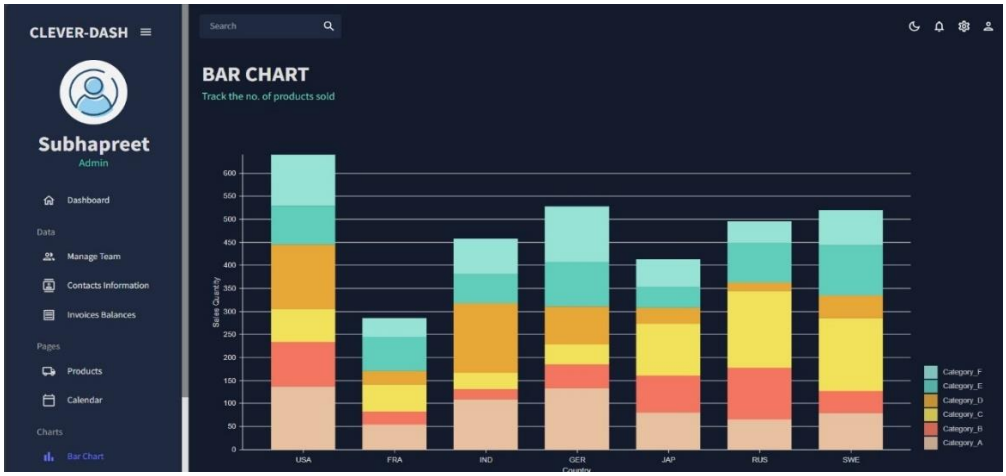

Fig 5.9 Bar chart page

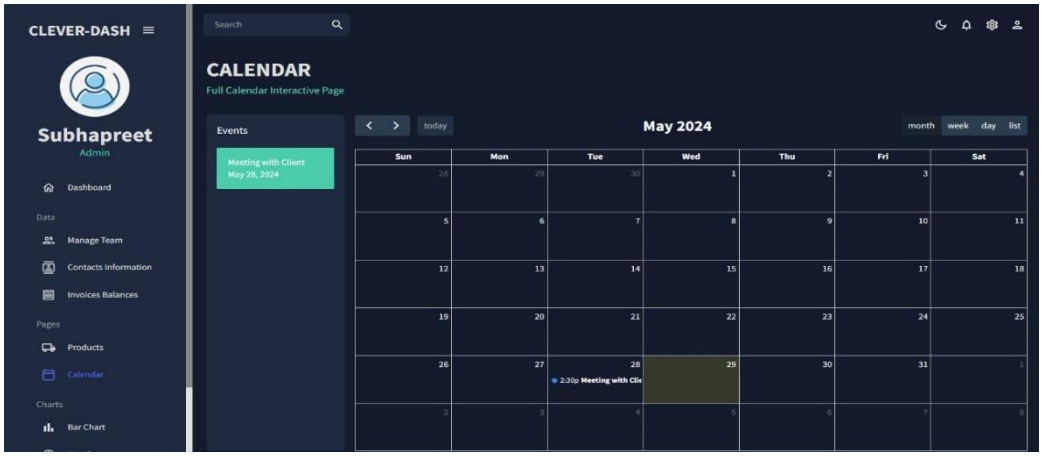

Fig 5.10 Calendar page

Fig 5.11Pie chart page

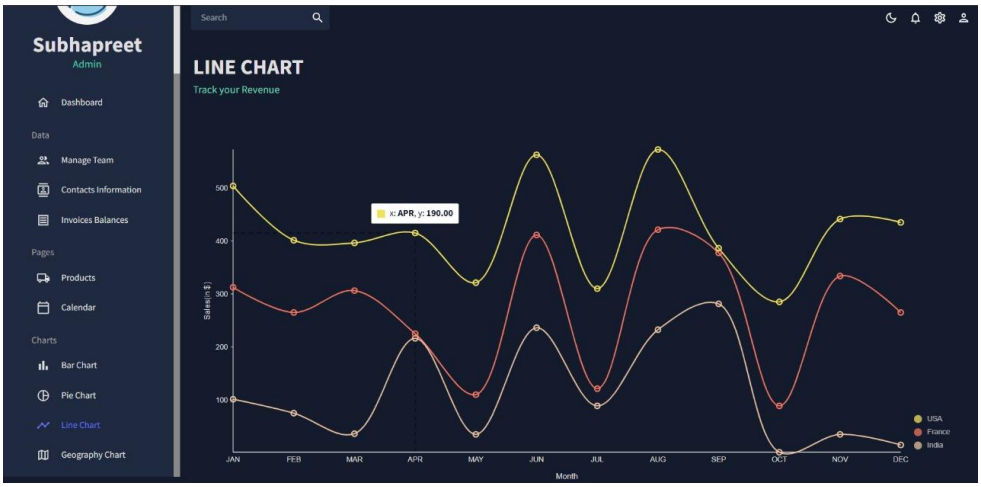

Fig 5.12 Line chart page

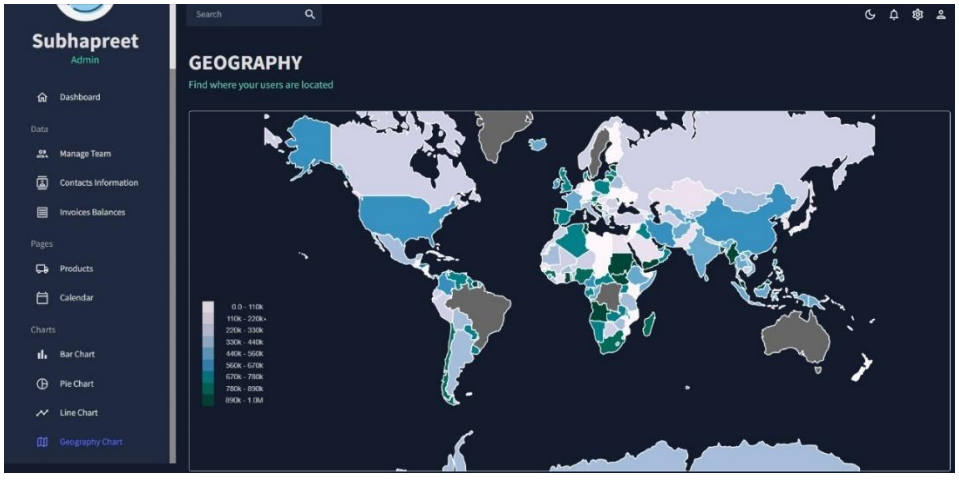

Fig 5.13 Geography page

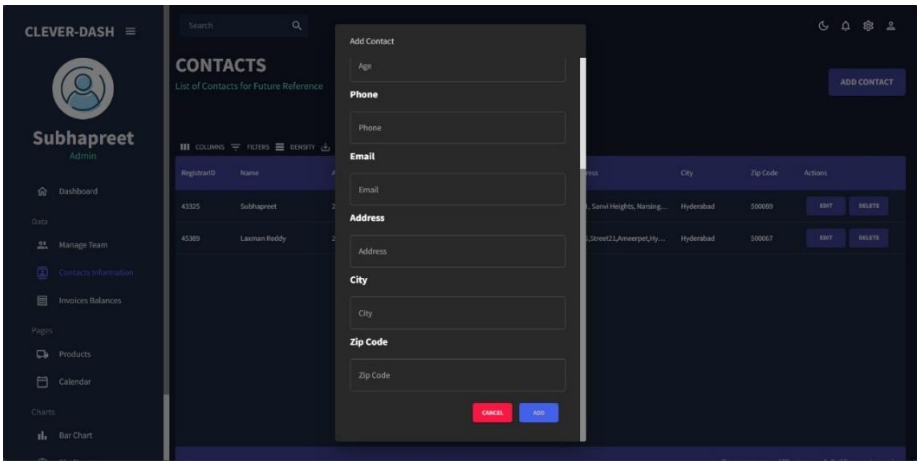

Fig 5.14 Add/edit contact

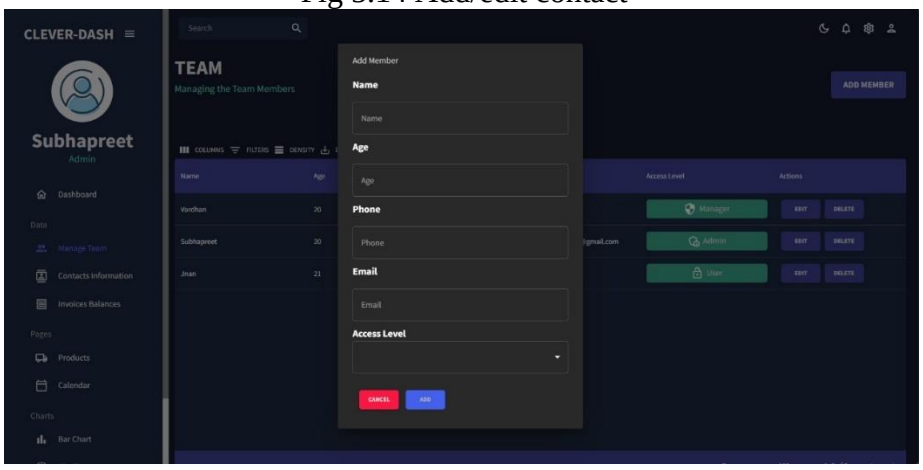

Fig 5.15 Add/edit member

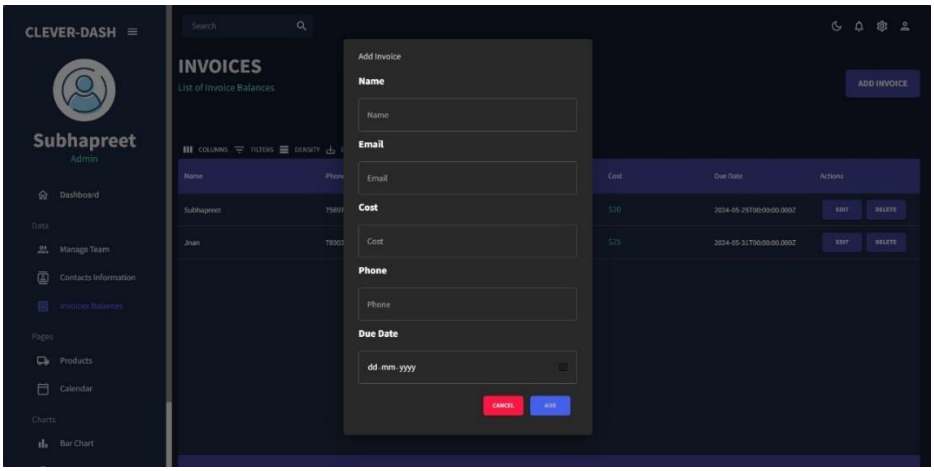

Fig 5.16 Add/Edit invoice

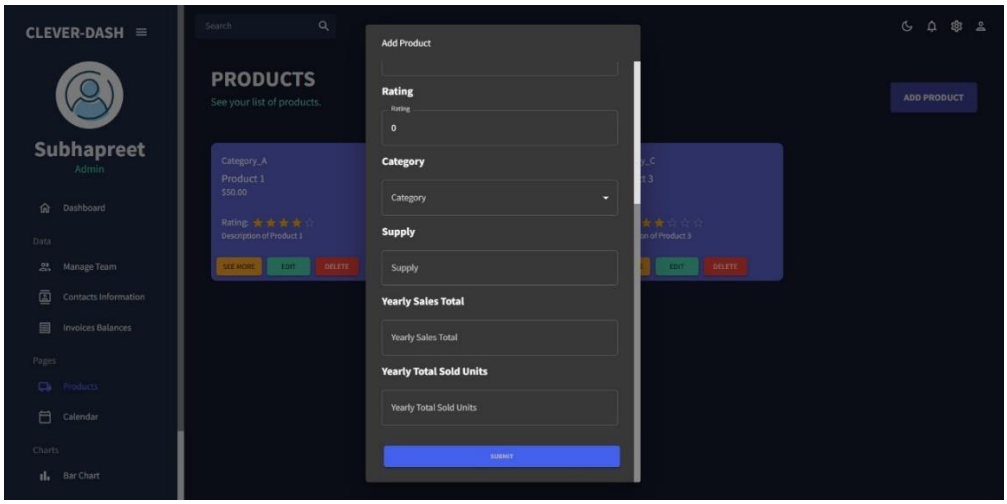

Fig 5.17 Add/Edit product

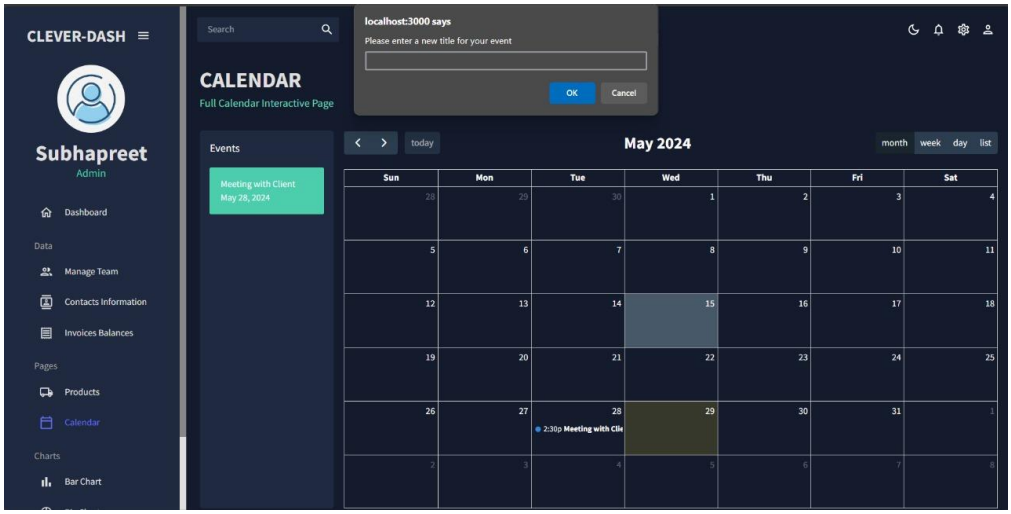

Fig 5.18 Add event

6. RESULT

6.1 Result and Discussions

Implementing the navbar in the CleverDash app using the MERN stack yields a responsive, scalable, and maintainable solution. The combination of React's efficient rendering, Node.js's robust backend capabilities, Express.js's flexible routing, and MongoDB's scalable database infrastructure ensures that the navbar performs well under varying loads while providing a great user experience. The results of developing and implementing the CleverDash app's navbar using the MERN stack (MongoDB, Express.js, React.js, Node.js) can be assessed across various dimensions such as performance, user experience, scalability, and maintainability.

7. CONCLUSION & FUTURE ENHANCEMENTS

7.1 Conclusion:

In conclusion, Clever-Dash offers a user-friendly platform for data visualization and management. With its advanced features and seamless integration capabilities, Clever-Dash empowers users to unlock insights and make informed decisions. It serves as a tool that can be intuitive and also is minimal in complexity that makes it easy for a wider range of users to use.

CleverDash can deliver a robust, scalable, and efficient application with a well-implemented navbar. This approach ensures a high-quality user experience and the ability to scale as the user base grows.

It serves as a great application for store owners so as to be able to track their inventory, sales and overall performance of their shop.

7.2 Future enhancements

1. Dynamic User Personalization

User-specific Navigation: Customize the navbar based on user roles (admin, regular user, guest). Display different options and links accordingly.

Profile Information: Show user profile details like avatar, name, and notifications directly on the navbar.

2. Notifications and Alerts

Real-time Notifications: Integrate real-time notifications using WebSockets (Socket.io with Node.js) or a service like Firebase.

Badge Alerts: Show alert badges on icons within the navbar (e.g., messages, notifications) to draw user attention.

3. Responsive Design Improvements

Mobile-friendly Navbar: Enhance the navbar to be more responsive with a collapsible menu or a hamburger menu for mobile devices.

Touch-friendly Elements: Ensure that navbar elements are easy to interact with on touchscreens.

4. Theming and Customization

User-selectable Themes: Allow users to switch between different themes (dark mode, light mode) from the navbar.

REFERENCES:

•**Material UI Documentation** (https://material-ui.com/): MongoDB is a popular NoSQL database that provides flexibility and scalability for storing and managing data. Its document-oriented nature and rich query capabilities make it suitable for MERN Applications.

•**Nivo Charts Documentation** (https://nivo.rocks/): Nivo is a rich collection of customizable React components for building data visualization charts.

•**Node.js Documentation** (https://nodejs.org/en/docs/): Node.js is a popular runtime environment for building server-side applications in JavaScript

•**Express.js Documentation** (https://expressjs.com/): Express.js is a minimalist web framework for Node.js, designed for building web applications and APIs.

•**MongoDB Documentation** (https://docs.mongodb.com/): MongoDB is a popular NoSQL database that provides flexibility and scalability for storing and managing data. Its document-oriented nature and rich query capabilities make it suitable for MERN Applications.