

## CLEVERDASH

**Designed and Developed by**

1. SUBHAPREET PATRO 2211CS010547
2. T. SANTHOSHI 2211CS010557 3.MUNIMANDA SNEHA 2211CS010390

4. DUBBAKA PRASANNA 2211CS010620

**Guided by**

**Mrs. P.SABITHA**

**Assistant Professor**

## Department of Computer Science & Engineering

**MALLA REDDY UNIVERSITY, HYDERABAD**

**2023-2024**



## 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 2023-24. The results embodied in this report have not been submitted to any other university or institute for the award of any degree or diploma.

**Internal Guide HOD-CSE**

**Mrs. P.Sabitha Dr. Shaik Meeravali**

**External Examiner**



## 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.P.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.P.Sabitha** for her constant guidance throughout our mini project work.

We are grateful to **Mr M. Rakesh** for his 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 Department of Computer Science and Engineering,** for providing seamless support and right Suggestions in the development of App.

First of all, we are highly indebted to **Dr. V. S. K Reddy,** **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**

# 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.

## TABLE OF CONTENTS

## DESCRIPTION PAGE NUMBER

[**CERTIFICATE ii**](#_bookmark0)

[**DECLARATION iii**](#_bookmark1)

**ACKNOWLEDGEMENTS iv**

**ABSTRACT v**

[**LIST OF FIGURES 2**](#_bookmark2)

[**Chapter 1 Introduction 3**](#_bookmark4)

* 1. **Introduction 3**
  2. [**Problem Statement 3**](#_bookmark5)
  3. **Objective of Project 4**
  4. **Goal of Project 5**

**Chapter 2 Problem Identification 6**

* 1. **Existing System 6**

**7**

* 1. **Proposed System Chapter 3 Requirements**

**9**

**3.1 Software Requirements**

**9**

**9**

**3. Hardware Requirements**

**Chapter 4 Design and Implementation**

**11**

* 1. **Design 11**

**12**

* 1. **Implementation Chapter 5 Code**

**14**

* 1. **Source Code 14**
  2. **Screenshots of Application 50**

**Chapter 6 Results & Conclusion**

**59**

* 1. **Results 59**

**59**

* 1. **Conclusion REFERENCES**

**60**

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE NO** | **NAME OF THE FIGURE** | **PAGE NO** |
| 5.1 | Design of Clever-Dash | 11 |
| 5.2.1 | Login Page | 50 |
| 5.2.2 | Register Page | 50 |
| 5.2.3 | Dashboard Page | 51 |
| 5.2.4 | Teams Page | 51 |
| 5.2.5 | Contacts Page | 52 |
| 5.2.6 | Invoices Page | 52 |
| 5.2.7 | Products Page | 53 |
| 5.2.8 | Calendar Page | 53 |
| 5.2.9 | Bar Chart Page | 54 |
| 5.2.10 | Pie Chart Page | 54 |
| 5.2.11 | Line Chart Page | 55 |
| 5.2.12 | Geography Page | 55 |
| 5.2.13 | Add/Edit Contact | 56 |
| 5.2.14 | Add/Edit Member | 56 |
| 5.2.15 | Add/Edit Invoice | 57 |
| 5.2.16 | Add/Edit Product | 57 |
| 5.2.17 | Add Event | 58 |

**CHAPTER - 1**

# INTRODUCTION

### Introduction to application

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 navbar , ensures 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. **Problem Statement**

Clever-Dash addresses the challenge of efficiently visualizing and managing large volume data

Cross various domains. It aims to streamline data interpretation and decision-making processes

for businesses, organizations, and individuals who grapple with complex datasets and seek

intuitive solutions for data analysis and presentation .By providing a unified platform with

advanced visualization tools and seamless data management capabilities, Clever-Dash aims

to empower users to extract valuable insights, identify trends, and make informed decisions

based on their data.

### Objectives of project

**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

### 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.

**CHAPTER-2**

**PROBLEM IDENTIFICATION**

### 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.

**2.2Proposed System**

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.

**CHAPTER – 3**

**REQUIREMENTS**

**3.1 Software Requirements**

**1. ReactJS:** A JavaScript library for building user interfaces with a component-based architecture.

**2. Material UI:** A popular React component library that implements Google's Material Design guidelines.

**3. Nivo Charts:** A collection of dataviz components built on top of D3.js for React applications.

**4. React-Toastify:** A library for displaying customizable and easy-to-use toast notifications in React apps.

**5. Formik:** A React library that helps in managing form state, validation, and handling submission.

**6. Yup:** A JavaScript schema builder for value parsing and validation, often used with Formik.

**7. NodeJS:** A JavaScript runtime built on Chrome's V8 engine for executing JavaScript code server-side.

**8. ExpressJS:** A minimal and flexible Node.js web application framework providing robust features for web and mobile applications.

**9. Mongoose:** An ODM (Object Data Modeling) library for MongoDB and Node.js, providing a schema-based solution to model data.

**3.2 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)

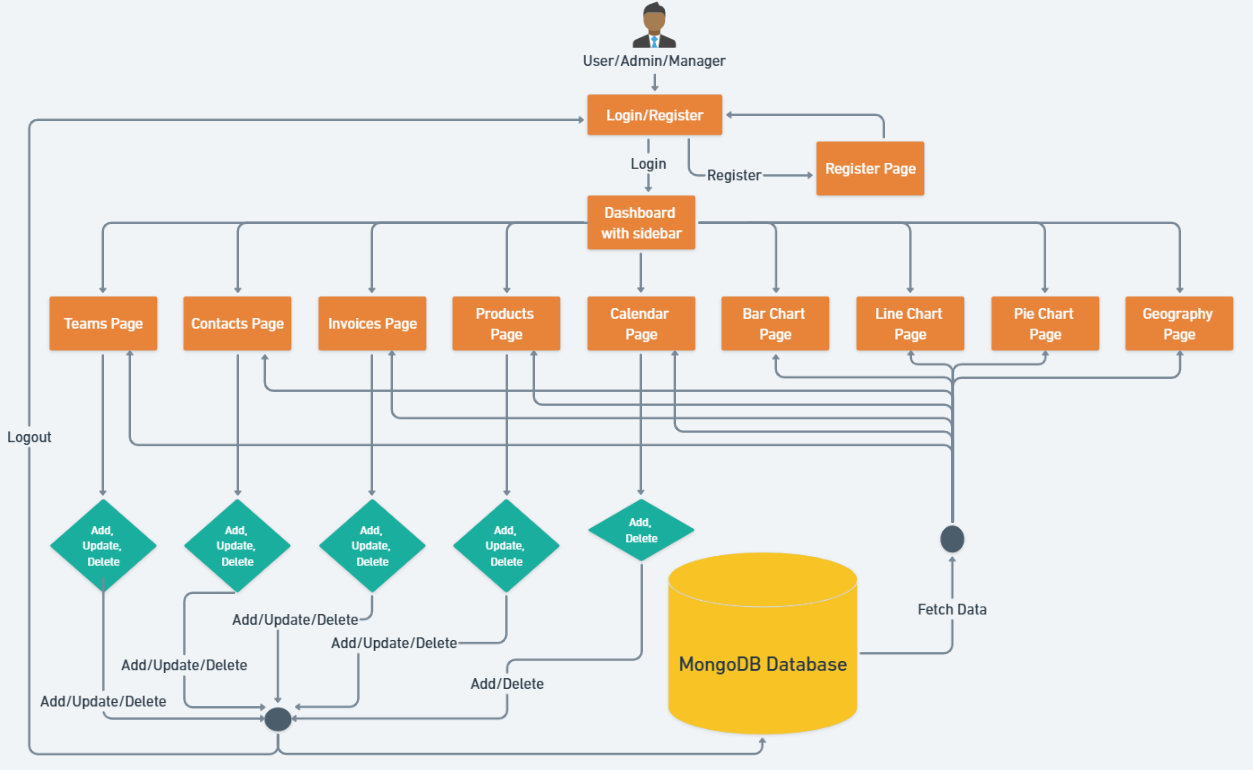
 **Other Requirements**:

* Modern web browser (Chrome, Firefox)
* Code editor/IDE (VS Code, WebStorm)
* Node.js installed
* MongoDB installed (or using a cloud service like MongoDB Atlas)

**CHAPTER – 4**

**DESIGN AND IMPLEMENTATION**

**4.1 Design**

****

**Fig: 5.1** Design of Clever-Dash

* 1. **Implementation**

CleverDash is a comprehensive business management application that provides various features to streamline operations and facilitate decision-making. Below are the implementation details for each page of CleverDash:

**1. Dashboard Page**

The dashboard serves as the central hub for accessing key metrics, insights, and summaries of business performance.

The features Displays customizable widgets and cards for monitoring important KPIs such as sales, revenue, and customer acquisition.

**2. Teams Page**

The teams page provides tools for managing team members, assignments ,and

collaboration within the organization.

It has User management functionalities such as adding, editing, and removing team

members.

**3. Contacts Page**

The contacts page enables users to manage customer and client information, facilitating relationship management and communication.

It has Contact databases for storing customer details including names, contact information, and interaction history.

**4. Invoices Page**

The invoices page allows users to create, send, and manageinvoices forbilling and payment processing.

It has Invoice generation with customizable templates and fields.

**5. Products Page**

The products page provides tools for managing inventory, product catalogs, and sales

tracking.

The Product catalog is for storing product details such as descriptions, prices, and

inventory levels.

**6. Calendar Page**

The calendar page allows users to schedule, manage, and track events, appointments, and

important dates.

Has Integration with FullCalendar library for interactive calendar functionality with various views (month, week, day, list).

**7. Bar Chart Page**

The bar chart page displays data visualization in the form of a bar chart, allowing users to

analyze sales quantity by product category.

It Utilizes the ResponsiveBar component from the Nivo library for responsive and

interactive bar chart rendering.

**8. Pie Chart Page**

The pie chart page visualizes product demand data using a pie chart, providing insights into

the distribution of demand across product categories.

It Implements the ResponsivePie component from the Nivo library for creating responsive

and interactive pie charts.

**9. Line Chart Page**

The line chart page visualizes revenuedata over time using a linechart, enabling users to

track revenue trends and performance.

ItUtilizes the ResponsiveLine component from the Nivo library for creating responsive and

interactive line charts.

**10. Geography Page**

The geography page displays geographical data using a choropleth map, allowing users to

visualize user distribution or other geographical metrics.

It Implements the ResponsiveChoropleth component from the Nivo library for creating

responsive and interactive choropleth maps.

CleverDash provides a user-friendly interface with robust functionalities tailored to meet the diverse needs of businesses, offering efficiency, insights, and control across various aspects of operations and management.

**CHAPTER - 5**

**CODE**

**5.1Source code**

**CLIENT: COMPONENTS**

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;

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;

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;

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;

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;

**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." });

}

});

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}`);

}

});

### 5.2Screenshots of Application

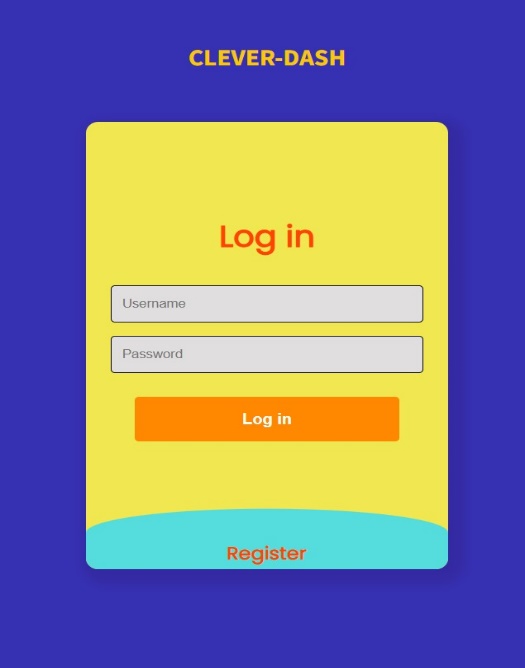


Fig 5.2.1 Login page

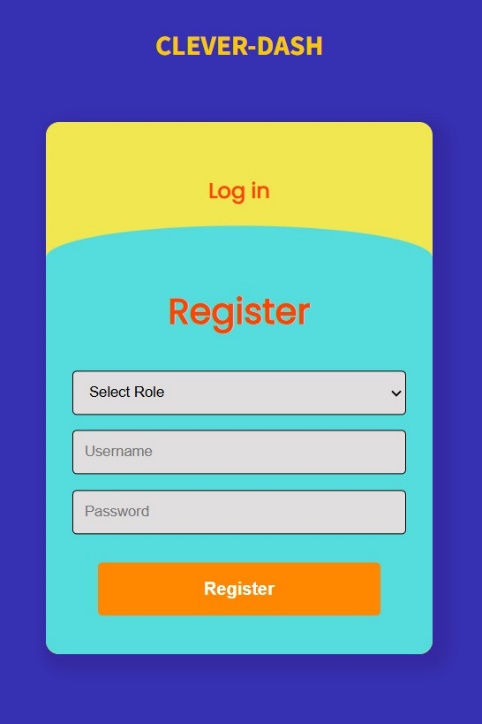


Fig 5.2.2 Registration page



Fig 5.2.3 Dashboard

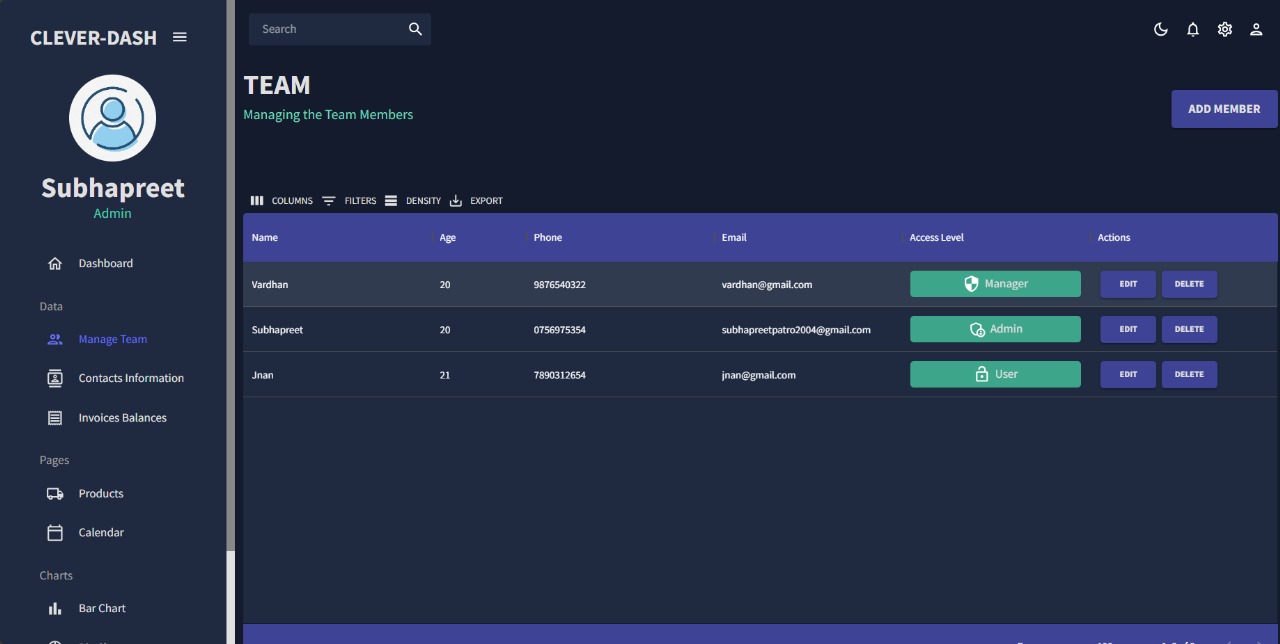


Fig 5.2.4 Teams page

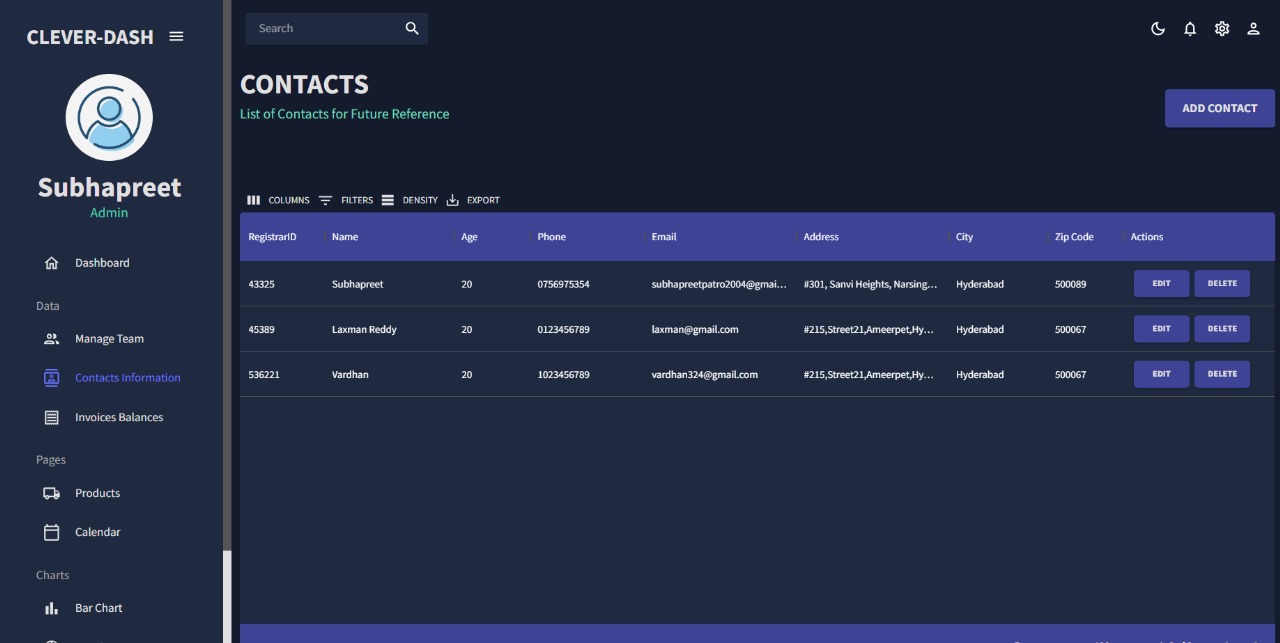


Fig 5.2.5 Contact page

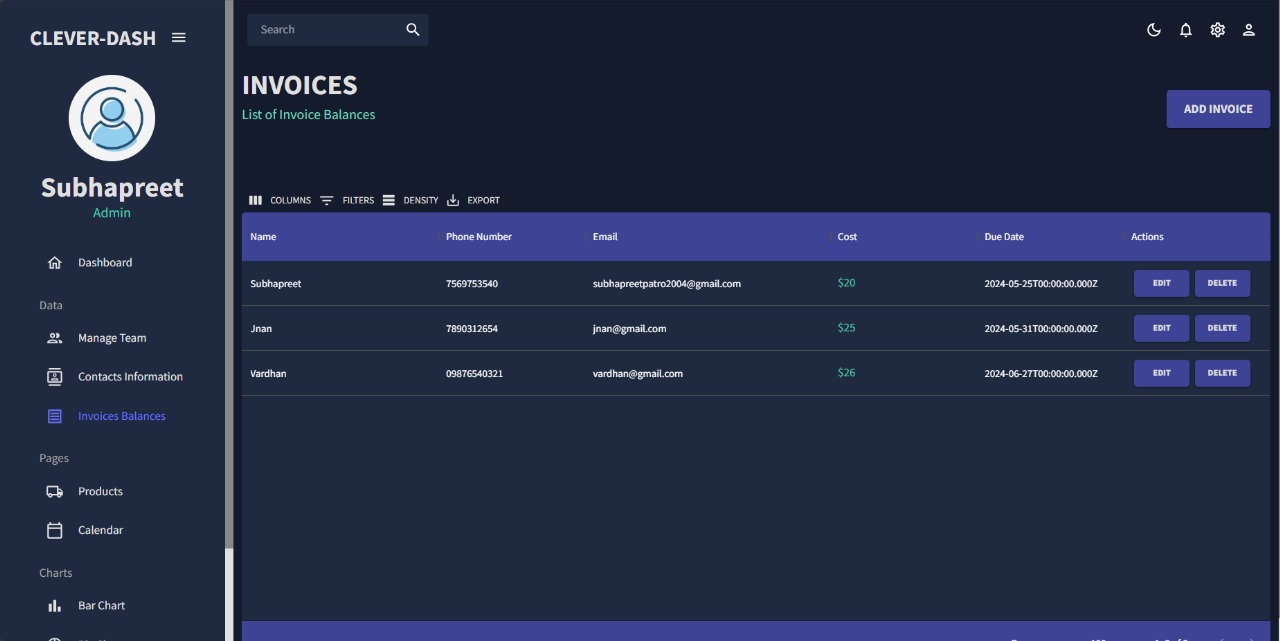


Fig 5.2.6 Invoices pages

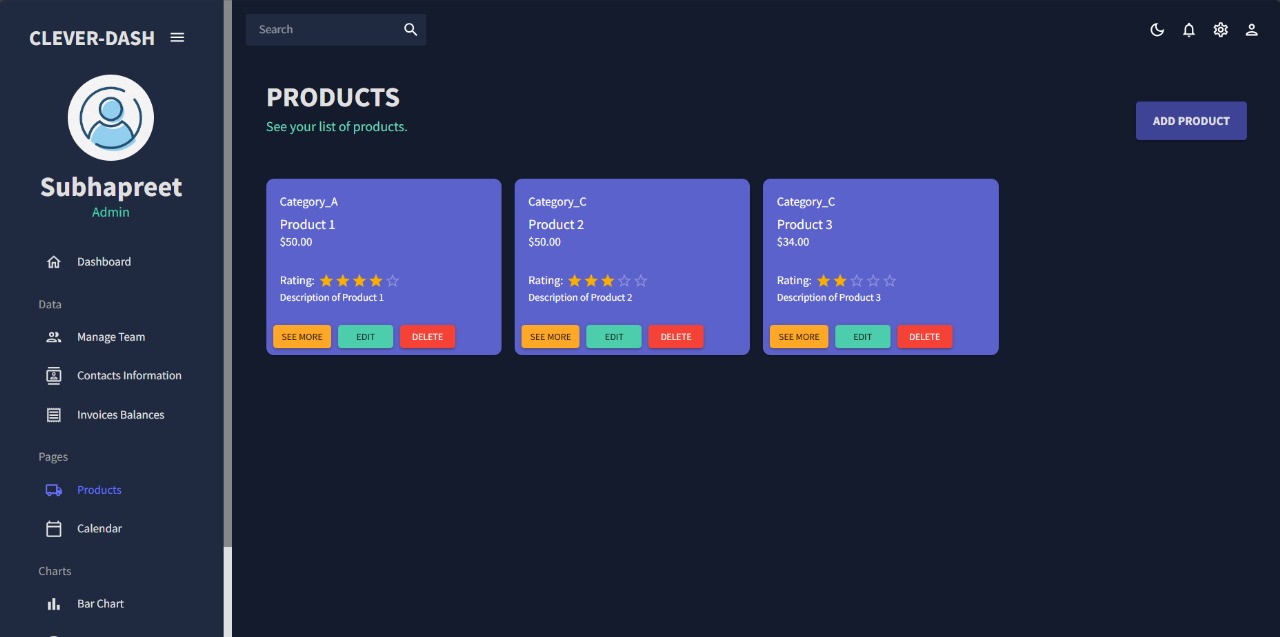


Fig 5.2.7 Products page

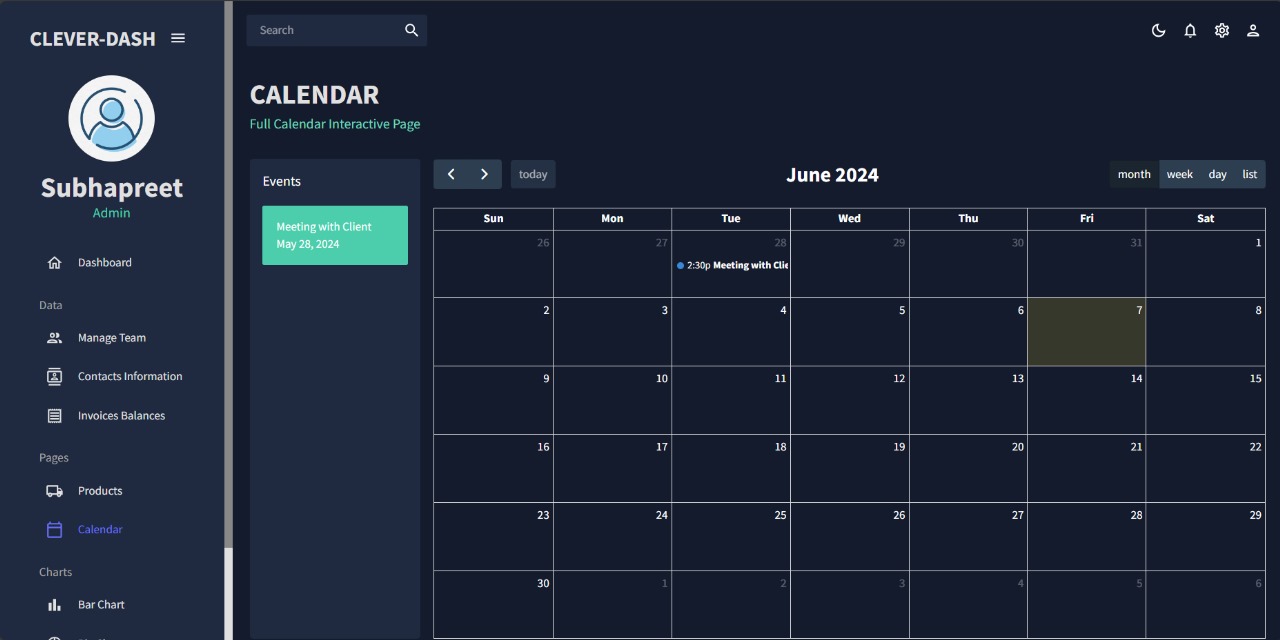


Fig 5.2.8 Calendar page

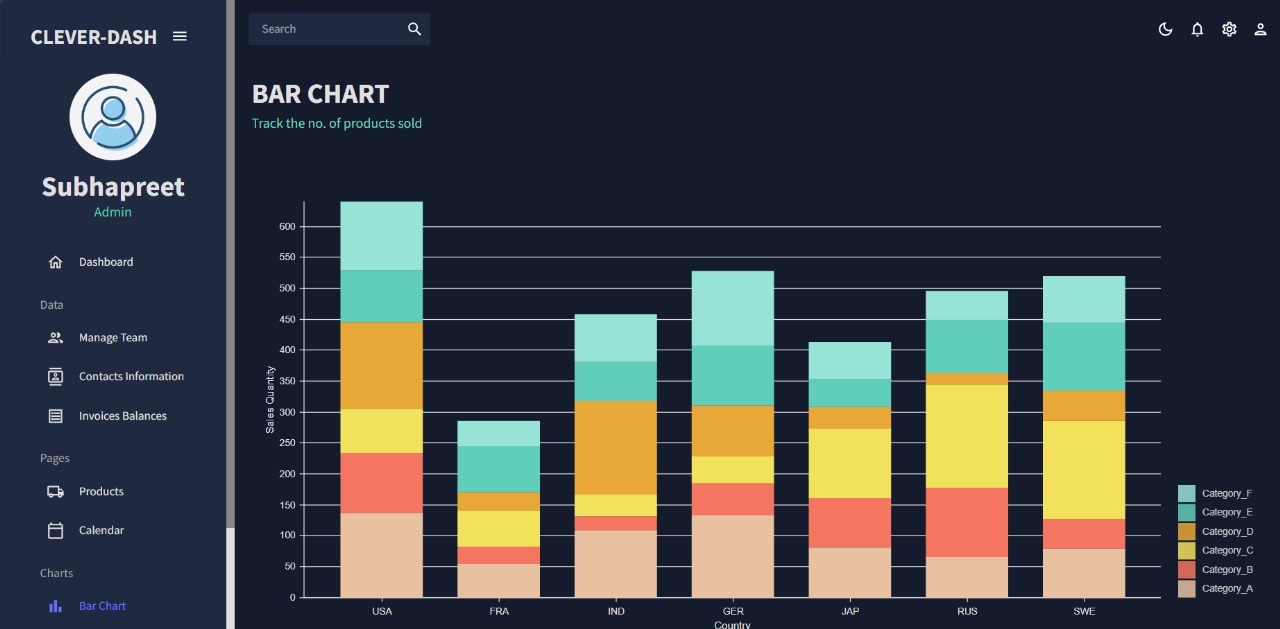


Fig 5.2.9 Bar chart page

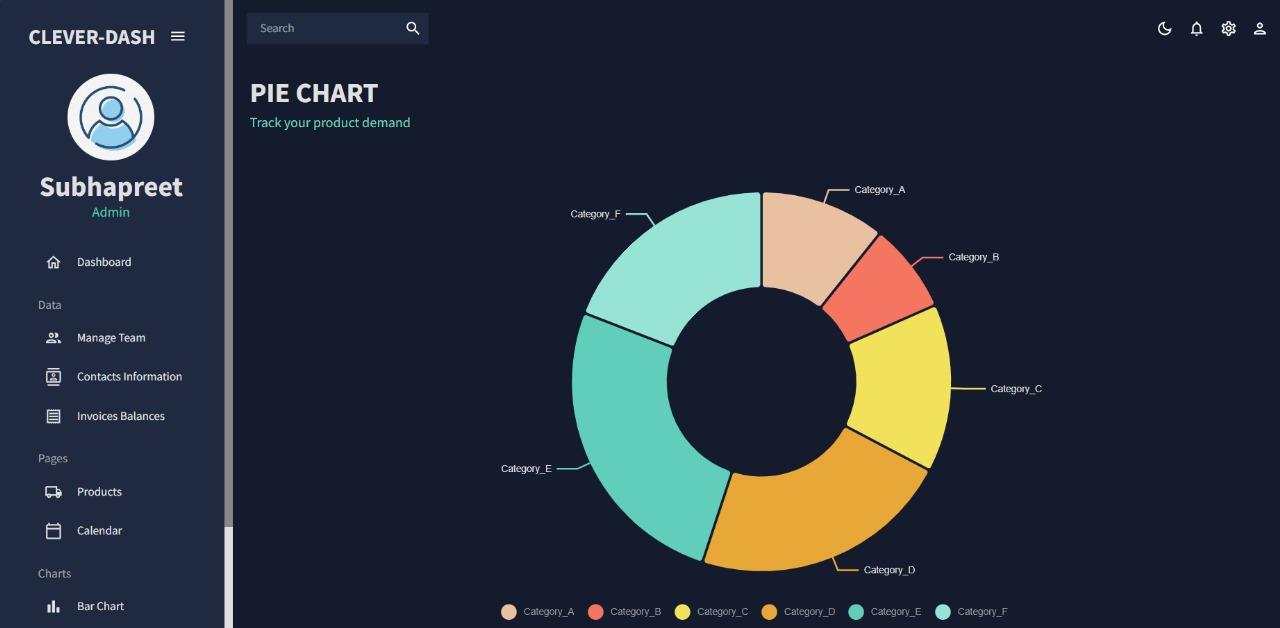


Fig 5.2.10 Pie chart page

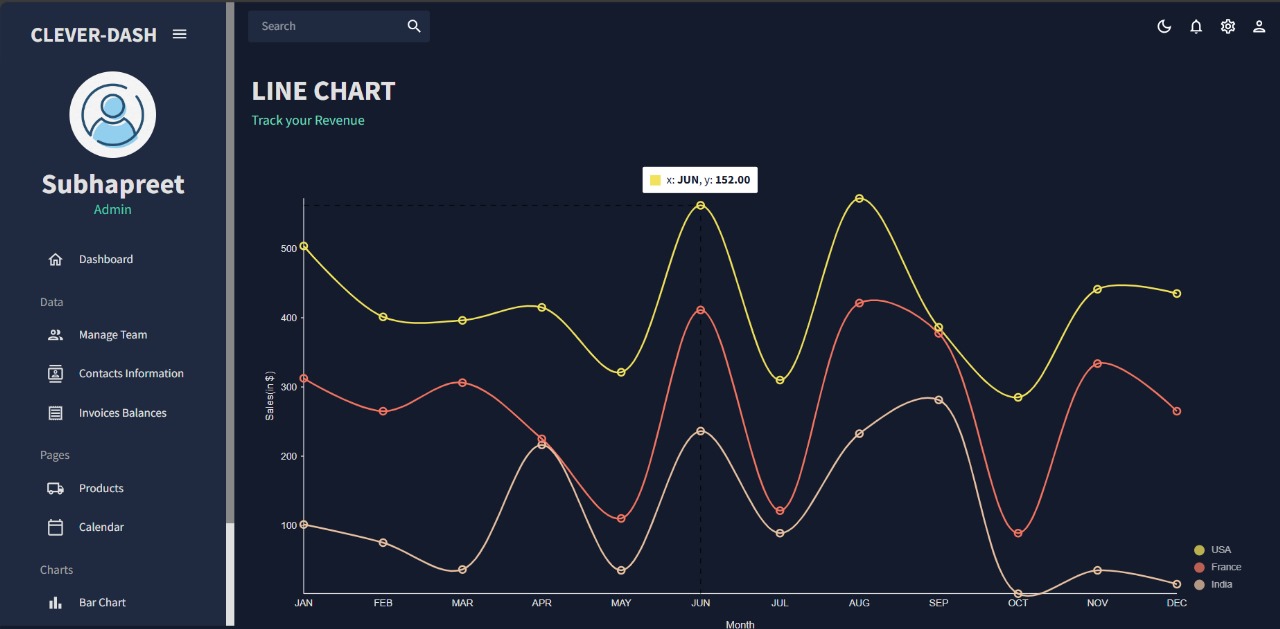


Fig 5.2.11 Line chart page

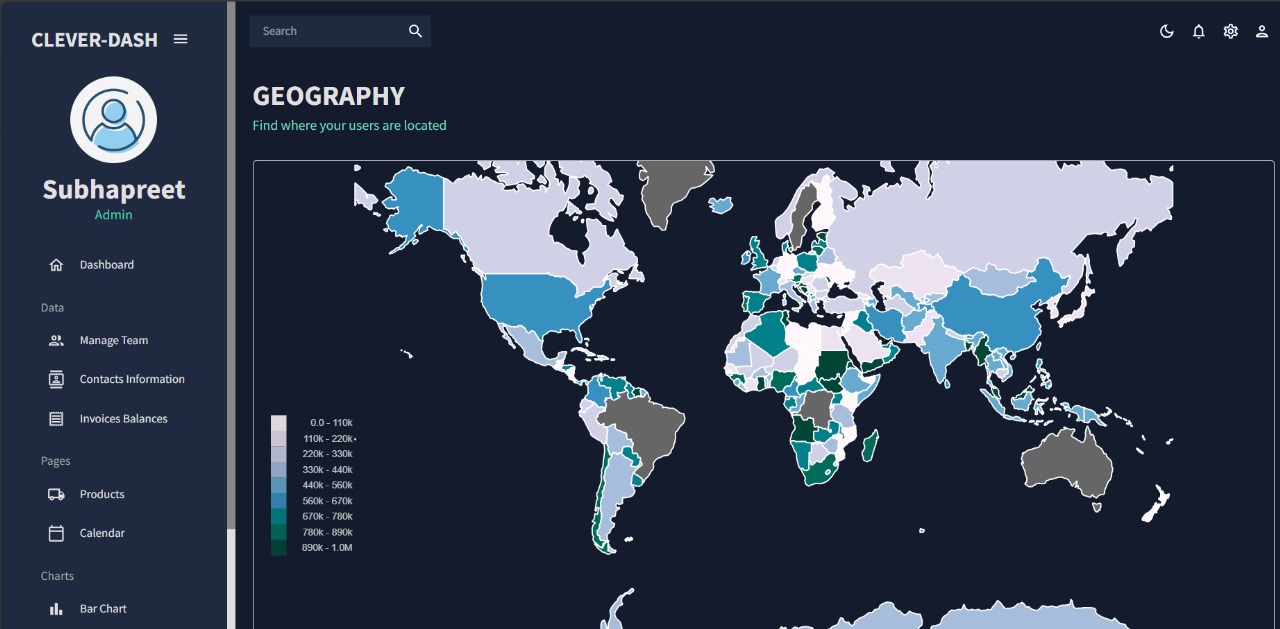


Fig 5.2.12 Geography page

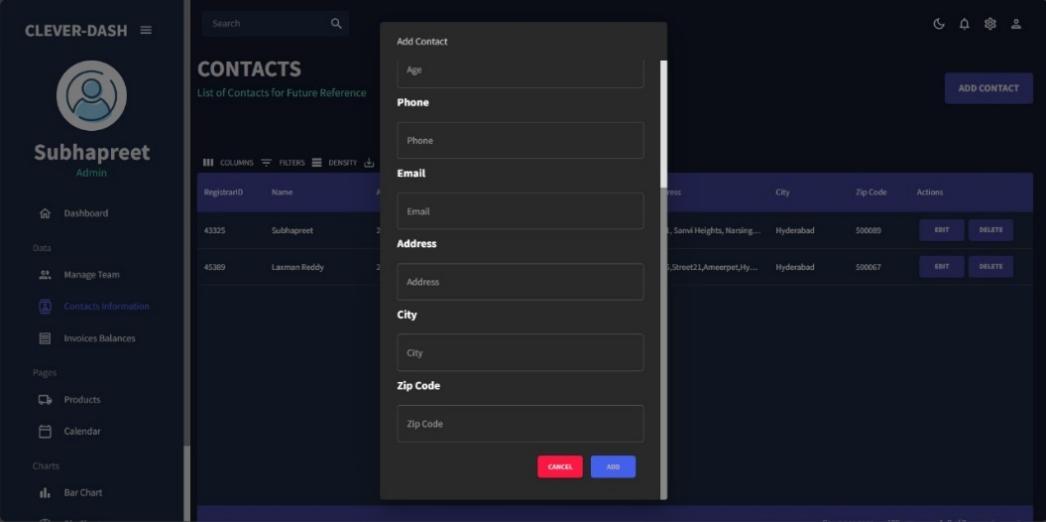


Fig 5.2.13 Add/edit contact

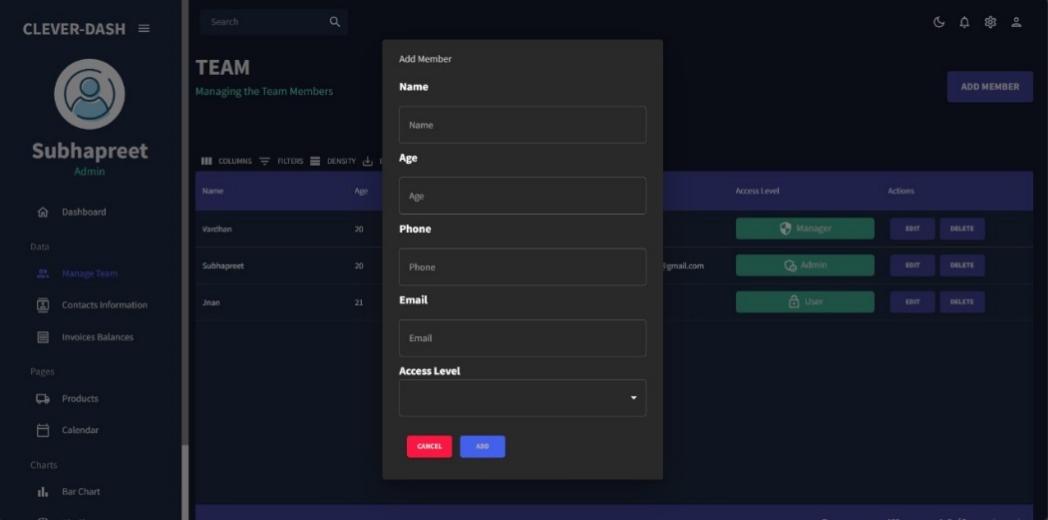


Fig 5.2.14 Add/edit member

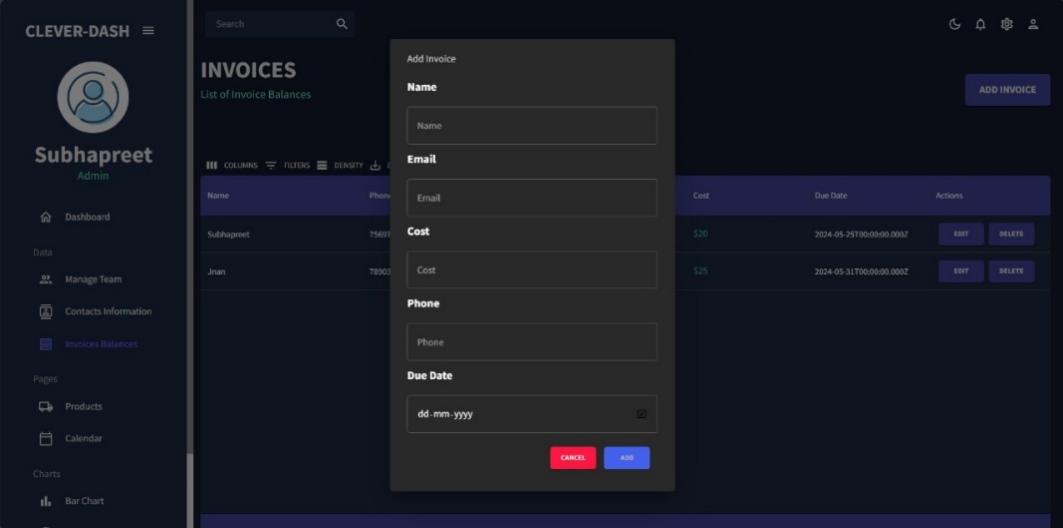


Fig 5.2.15 Add/edit invoice

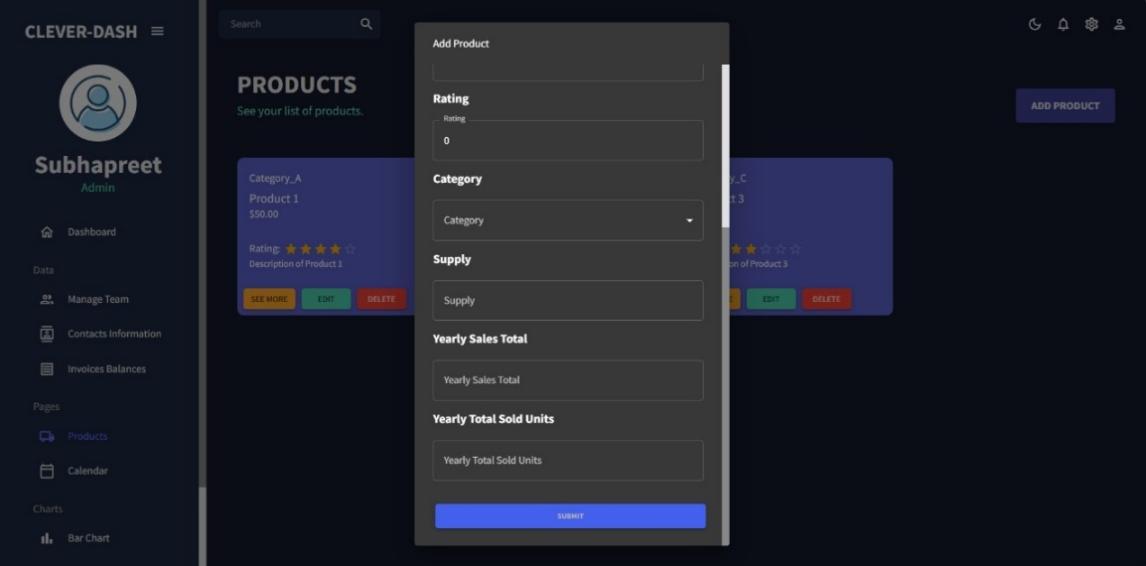


Fig 5.2.16 Add/edit product

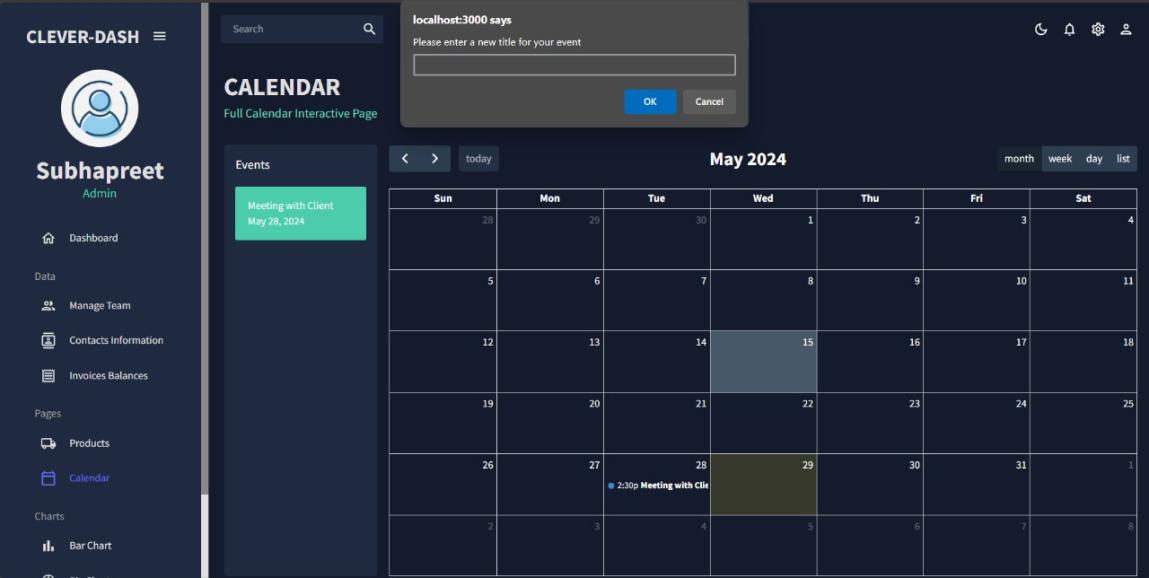


Fig 5.2.17 Add Event

**CHAPTER -6**

**RESULTS & CONCLUSION**

### Result

### 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.

### 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.

# 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.