**Real time Dashboard Notes**

1. Material UI

Here is theme.js

import { createContext,useState, useMemo } from "react";

import { createTheme } from '@mui/material/styles'

// import { colors, Typography } from "@mui/material";

// import { light } from "@mui/material/styles/createPalette";

export const tokens = (mode) => ({

    ...( mode === 'dark'

        ?

        {

            greay : {

                900: "#141414",

                800: "#292929",

                700: "#3d3d3d",

                600: "#525252",

                500: "#666666",

                400: "#858585",

                300: "#a3a3a3",

                200: "#c2c2c2",

                100: "#e0e0e0",

            },

            primary: {

                900: "#040509",

                800: "#080b12",

                700: "#0c101b",

                600: "#101624",

                500: "#141b2d",

                400: "#434957",

                300: "#727681",

                200: "#a1a4ab",

                100: "#d0d1d5",

            },

            greenAccent: {

                900: "#0f2922",

                800: "#1e5245",

                700: "#2e7c67",

                600: "#3da58a",

                500: "#4cceac",

                400: "#70d8bd",

                300: "#94e2cd",

                200: "#b7ebde",

                100: "#dbf5ee",

            },

            redAccent: {

                900: "#2c100f",

                800: "#58201e",

                700: "#832f2c",

                600: "#af3f3b",

                500: "#db4f4a",

                400: "#e2726e",

                300: "#e99592",

                200: "#f1b9b7",

                100: "#f8dcdb",

            },

            blueAccent : {

                900: "#151632",

                800: "#2a2d64",

                700: "#3e4396",

                600: "#535ac8",

                500: "#6870fa",

                400: "#868dfb",

                300: "#a4a9fc",

                200: "#c3c6fd",

                100: "#e1e2fe",

            },

        }

        :

        {

            greay : {

                100: "#141414",

                200: "#292929",

                300: "#3d3d3d",

                400: "#525252",

                500: "#666666",

                600: "#858585",

                700: "#a3a3a3",

                800: "#c2c2c2",

                900: "#e0e0e0",

            },

            primary: {

                100: "#040509",

                200: "#080b12",

                300: "#0c101b",

                400: "#101624",

                500: "#141b2d",

                600: "#434957",

                700: "#727681",

                800: "#a1a4ab",

                900: "#d0d1d5",

            },

            greenAccent: {

                100: "#0f2922",

                200: "#1e5245",

                300: "#2e7c67",

                400: "#3da58a",

                500: "#4cceac",

                600: "#70d8bd",

                700: "#94e2cd",

                800: "#b7ebde",

                900: "#dbf5ee",

            },

            redAccent: {

                100: "#2c100f",

                200: "#58201e",

                300: "#832f2c",

                400: "#af3f3b",

                500: "#db4f4a",

                600: "#e2726e",

                700: "#e99592",

                800: "#f1b9b7",

                900: "#f8dcdb",

            },

            blueAccent : {

                100: "#151632",

                200: "#2a2d64",

                300: "#3e4396",

                400: "#535ac8",

                500: "#6870fa",

                600: "#868dfb",

                700: "#a4a9fc",

                800: "#c3c6fd",

                900: "#e1e2fe",

            },

        }

    )

})

// mui thesme settings

export const themeSettings = (mode ) => {

    const colors = tokens(mode);

    return {

        palette : {

            mode : mode,

            ...(mode === 'dark'

                ? {

                    primary : {

                        main : colors.primary[500],

                    },

                    secondary : {

                        main : colors.greenAccent[500]

                    },

                    neutral : {

                        dark : colors.greay[700],

                        main : colors.greay[500],

                        light : colors.greay[100]

                    },

                    background : {

                        default : colors.primary[500]

                    }

                }

                :

                {

                    primary : {

                        main : colors.primary[100],

                    },

                    secondary : {

                        main : colors.greenAccent[500]

                    },

                    neutral : {

                        dark : colors.greay[700],

                        main : colors.greay[500],

                        light : colors.greay[100]

                    },

                    background : {

                        default : "#fcfcfc"

                    }

                }

            )

        },

        Typography : {

            fontFamily : ['source Sans Pro','sans-serif'].join(','),

            fontSize : 12,

            h1 : {

                fontFamily : ['source Sans Pro','sans-serif'].join(','),

                fontSize : 40,

            },

            h2 : {

                fontFamily : ['source Sans Pro','sans-serif'].join(','),

                fontSize : 32,

            },

            h3 : {

                fontFamily : ['source Sans Pro','sans-serif'].join(','),

                fontSize : 24,

            },

            h4 : {

                fontFamily : ['source Sans Pro','sans-serif'].join(','),

                fontSize : 20,

            },

            h5 : {

                fontFamily : ['source Sans Pro','sans-serif'].join(','),

                fontSize : 16,

            }

            ,

            h6 : {

                fontFamily : ['source Sans Pro','sans-serif'].join(','),

                fontSize : 14,

            },

        }

    }

}

// context for the color mode

export const ColorModeContext = createContext({

    toggleColorMode : () => {},

})

export const useMode = () => {

    const [mode,setMode] = useState('dark');

    const ColorMode = useMemo(

        () => {

            toggleColorMode : () => {

                setMode((prev)=> (

                    prev === "light" ?'dark': 'light'

                ))

            }

        },

        []

    )

    const theme = useMemo(

        () => createTheme(themeSettings(mode)),

        [mode]

    )

    return [ theme , ColorMode]

}

This code creates a **dark/light theme toggler** in a React app using Material UI. It utilizes React’s Context API for state management, along with useState, useMemo, and custom color palettes for dark and light modes.

Let’s break it down section by section:

**1. Import Statements**

javascript

Copy code

import { createContext, useState, useMemo } from "react";

import { createTheme, ThemeProvider } from "@mui/material/styles";

import { light } from "@mui/material/styles/createPalette";

* createContext, useState, and useMemo are React hooks that help manage and memoize state.
* createTheme and ThemeProvider are from Material UI to create and apply a custom theme.

**2. tokens Function: Color Palette**

javascript

Copy code

export const tokens = (mode) => ({

...(mode === "dark"

? {

greay: { /\* colors for dark mode \*/ },

primary: { /\* colors for primary section \*/ },

greenAccent: { /\* green accent colors \*/ },

redAccent: { /\* red accent colors \*/ },

blueAccent: { /\* blue accent colors \*/ },

}

: {

greay: { /\* colors for light mode \*/ },

primary: { /\* colors for primary section \*/ },

greenAccent: { /\* green accent colors \*/ },

redAccent: { /\* red accent colors \*/ },

blueAccent: { /\* blue accent colors \*/ },

}),

});

* This function returns an object with color tokens based on the mode (dark or light).
* It defines custom color shades (900, 800, 700, etc.) for each mode. These tokens are later accessed in the theme settings to customize the app’s look and feel.

**3. themeSettings Function: Theme Customization**

javascript

Copy code

export const themeSettings = (mode) => {

const colors = tokens(mode);

return {

palette: {

mode: mode,

...(mode === "dark"

? {

primary: { main: colors.primary[500] },

secondary: { main: colors.greenAccent[500] },

neutral: { dark: colors.greay[700], main: colors.greay[500], light: colors.greay[100] },

background: { default: colors.primary[500] },

}

: {

primary: { main: colors.primary[100] },

secondary: { main: colors.greenAccent[500] },

neutral: { dark: colors.greay[700], main: colors.greay[500], light: colors.greay[100] },

background: { default: "#fcfcfc" },

}),

},

typography: {

fontFamily: ["source Sans Pro", "sans-serif"].join(","),

fontSize: 12,

h1: { fontSize: 40 },

h2: { fontSize: 32 },

h3: { fontSize: 24 },

h4: { fontSize: 20 },

h5: { fontSize: 16 },

h6: { fontSize: 14 },

},

};

};

* themeSettings creates the theme object using Material UI’s structure.
* The palette key applies colors from tokens based on whether mode is "dark" or "light".
* typography defines font styles for headings and base font size, ensuring consistent typography across dark and light modes.

**4. ColorModeContext: Context for Theme Toggle**

javascript

Copy code

export const ColorModeContext = createContext({

toggleColorMode: () => {},

});

* ColorModeContext is a Context object to store and provide access to the toggleColorMode function across the app.
* It initializes an empty toggle function (toggleColorMode: () => {}) that’s later set in useMode.

**5. useMode Hook: Custom Hook for Theme Management**

javascript

Copy code

export const useMode = () => {

const [mode, setMode] = useState("dark");

const colorMode = useMemo(() => ({

toggleColorMode: () => {

setMode((prev) => (prev === 'light' ? 'dark' : 'light'));

},

}), []);

const theme = useMemo(() => createTheme(themeSettings(mode)), [mode]);

return [theme, colorMode];

};

* useMode initializes the theme mode with useState, defaulting to "dark".
* colorMode provides toggleColorMode, which switches between dark and light modes by updating mode.
* theme uses createTheme to dynamically apply themeSettings(mode), ensuring theme changes are applied immediately.
* useMemo is used to optimize performance by caching theme and colorMode, only recalculating them if mode changes.

**Usage**: useMode returns both the theme and the toggle function, making it easy to integrate the theme toggle into a component with ThemeProvider.

This structure provides a flexible theme that can be changed by calling colorMode.toggleColorMode(), which you can connect to UI controls for a responsive dark/light mode switch.

**Material UI (MUI) components**

Material UI (MUI) offers a range of components (or tags, as you might call them) that help streamline building a responsive, consistent, and modern user interface. Here’s a breakdown of some commonly used MUI components to get you started:

### 1. ****Container****

* **<Container>**: Centers your content and applies default padding. Great for wrapping content in a responsive layout.

### 2. ****Grid****

* **<Grid container>**: Defines a container for grid items and sets up a responsive grid layout.
* **<Grid item>**: Represents an item within a grid container. Can specify column width, spacing, and breakpoints.

### 3. ****Typography****

* **<Typography>**: Used for displaying text with consistent styling. Can take variants like "h1", "body1", "caption", etc., to adjust size, weight, and other styles.

### 4. ****Buttons****

* **<Button>**: The basic button component. Can have different variants like "contained", "outlined", and "text". Also supports colors like "primary", "secondary", etc.
* **<IconButton>**: Displays an icon within a button, often used for actions like close or edit.

### 5. ****AppBar and Toolbar****

* **<AppBar>**: A top-level navigation component that typically holds a title, logo, or actions.
* **<Toolbar>**: Serves as a flexible container for grouping content inside the AppBar.

### 6. ****Card****

* **<Card>**: A container for holding information or media with an elevated look.
* **<CardContent>**: Holds the main content inside a card.
* **<CardMedia>**: Used to display an image or media within the card.

### 7. ****Form Controls****

* **<TextField>**: A form field for user input, commonly used for both text and number inputs.
* **<Checkbox>, <Radio>, <Switch>**: Form elements for selecting options.
* **<Select>**: A dropdown menu for selecting from a list of options.

### 8. ****Modal****

* **<Dialog>**: A modal dialog component that displays over other content. Contains sub-components like <DialogTitle>, <DialogContent>, and <DialogActions> for organizing content.
* **<Popover> and <Tooltip>**: Used to display information or actions in floating boxes triggered by user interaction.

### 9. ****Icons****

* Material UI provides a wide variety of icons in the **@mui/icons-material** package that you can easily include using the **<Icon>** component or directly importing individual icons.

### 10. ****Paper****

* **<Paper>**: A basic surface for other components that provides an elevated, shadowed look similar to cards.

You can combine these tags to create layouts and structures quickly. Let me know if you’d like to dive into any specific component or example code!

### 1. ****Box****

* **<Box>** is a versatile component in Material UI for layout and styling. It’s essentially a wrapper with powerful styling props for creating responsive and flexible layouts.

Jsx

import React from 'react';

import { Box } from '@mui/material';

function ExampleBox() {

return (

<Box

sx={{

width: 300,

height: 200,

backgroundColor: 'primary.main',

color: 'white',

padding: 2,

borderRadius: 2,

}}

>

This is a Box component!

</Box>

);

}

export default ExampleBox;

* **Key props**:
  + **sx**: A powerful prop for applying custom CSS.
  + **padding, margin, display, width, height**: Controls layout and styling.

### 2. ****Icons****

* Material UI includes a large library of icons that can be imported and customized.

Jsx

import React from 'react';

import { Box } from '@mui/material';

import { Home, Favorite, Settings } from '@mui/icons-material';

function IconExample() {

return (

<Box display="flex" justifyContent="space-around" padding={2}>

<Home color="primary" fontSize="large" />

<Favorite color="secondary" fontSize="large" />

<Settings color="action" fontSize="large" />

</Box>

);

}

export default IconExample;

* **Customizing Icons**:
  + **color**: Choose from predefined color options like "primary", "secondary", "action", etc.
  + **fontSize**: Set sizes such as "small", "large", or "inherit".

### 3. ****Menu****

* **<Menu>** is a pop-up component that displays a list of options when triggered (usually by a button).

Jsx

import React, { useState } from 'react';

import { Button, Menu, MenuItem } from '@mui/material';

function MenuExample() {

const [anchorEl, setAnchorEl] = useState(null);

const handleClick = (event) => {

setAnchorEl(event.currentTarget);

};

const handleClose = () => {

setAnchorEl(null);

};

return (

<div>

<Button onClick={handleClick} variant="contained" color="primary">

Open Menu

</Button>

<Menu

anchorEl={anchorEl}

open={Boolean(anchorEl)}

onClose={handleClose}

>

<MenuItem onClick={handleClose}>Profile</MenuItem>

<MenuItem onClick={handleClose}>My Account</MenuItem>

<MenuItem onClick={handleClose}>Logout</MenuItem>

</Menu>

</div>

);

}

export default MenuExample;

* **Key props**:
  + **anchorEl**: The element that the menu is anchored to.
  + **open**: Controls if the menu is displayed.
  + **onClose**: Callback for when the menu closes.

These components allow for flexible layouts, simple icons, and intuitive menus that enhance the user experience. Let me know if you’d like to see more detailed customization or combined usage!

**ProSidebar**

To create a sidebar with **ProSidebar** in React, we can use the react-pro-sidebar library, which offers advanced sidebar features such as collapsibility, icon customization, and nested menu items.

Here’s how you can get started with **ProSidebar**:

**Step 1: Install react-pro-sidebar**

You’ll need to install the react-pro-sidebar library:

Bash

npm install react-pro-sidebar

**Step 2: Basic Setup with ProSidebar**

Import ProSidebar, Menu, and related components to build a sidebar with nested items and icons.

Jsx

import React, { useState } from 'react';

import { ProSidebar, Menu, MenuItem, SubMenu } from 'react-pro-sidebar';

import 'react-pro-sidebar/dist/css/styles.css'; // Import ProSidebar CSS

import { FaHome, FaInfo, FaEnvelope, FaCog } from 'react-icons/fa';

function ProSidebarExample() {

const [collapsed, setCollapsed] = useState(false);

const toggleSidebar = () => {

setCollapsed(!collapsed);

};

return (

<div style={{ display: 'flex' }}>

{/\* Sidebar Component \*/}

<ProSidebar collapsed={collapsed}>

<Menu iconShape="square">

<MenuItem icon={<FaHome />}>Home</MenuItem>

<MenuItem icon={<FaInfo />}>About</MenuItem>

<SubMenu title="Services" icon={<FaCog />}>

<MenuItem> Web Development </MenuItem>

<MenuItem> Mobile Development </MenuItem>

<MenuItem> Cloud Services </MenuItem>

</SubMenu>

<MenuItem icon={<FaEnvelope />}>Contact</MenuItem>

</Menu>

</ProSidebar>

{/\* Button to Collapse Sidebar \*/}

<button onClick={toggleSidebar} style={{ marginLeft: 10 }}>

Toggle Sidebar

</button>

</div>

);

}

export default ProSidebarExample;

**Explanation of Components and Props**

1. **ProSidebar**: The main sidebar container that accepts props like:
   * **collapsed**: A Boolean to toggle between collapsed and expanded views.
2. **Menu**: Wraps all MenuItem and SubMenu items, where:
   * **iconShape="square"** gives a square shape to icons by default.
3. **MenuItem**: Represents each menu item with an icon. Customization includes:
   * **icon**: Add custom icons from libraries like react-icons.
4. **SubMenu**: Allows for nested items, with a title and icon. Clicking on SubMenu expands or collapses the nested items.

**Additional Customization**

1. **Theming**: You can add custom CSS or use inline styles to theme the sidebar.
2. **Responsive Sidebar**: Control collapsed based on screen size to make the sidebar responsive.
3. **Active Item Styling**: Use className or inline styles to highlight the active item for better navigation context.

The **ProSidebar** component is feature-rich and highly customizable, making it ideal for complex, nested sidebars. Let me know if you’d like more examples on styling or advanced configurations!

**Pro SideBar**

import React , {useState} from 'react'

import { Menu, MenuItem,ProSidebar } from 'react-pro-sidebar'

import 'react-pro-sidebar/dist/css/styles.css';

import {Box,IconButton,Typography,useTheme} from '@mui/material'

import { Link } from 'react-router-dom'

import {tokens } from '../../theme.js'

import HomeOutlinedIcon from '@mui/icons-material/HomeOutlined'

import  LightModeOutlined from "@mui/icons-material/LightModeOutlined"

import  PeopleOutlinedIcion from "@mui/icons-material/PeopleOutlined"

import  ContactsOutlinedIcon from "@mui/icons-material/ContactsOutlined"

import  ReceiptOutlinedIcon from "@mui/icons-material/ReceiptOutlined"

import  PersonOutlinedIcon from "@mui/icons-material/PersonOutlined"

import  CalendarTodayOutlinedIcon from "@mui/icons-material/CalendarTodayOutlined"

import  HelpOutlinedIcon from "@mui/icons-material/HelpOutlined"

import  BarChartOutlinedIcon from "@mui/icons-material/BarChartOutlined"

import  PieChartOutlineOutlinedIcon from "@mui/icons-material/PieChartOutlineOutlined"

import PublicOutlinedIcon from '@mui/icons-material/PublicOutlined';

import  TimelineOutlinedIcon from "@mui/icons-material/TimelineOutlined"

import  MenuOutlinedIcon from "@mui/icons-material/MenuOutlined"

import  MapOutlinedIcon from "@mui/icons-material/MapOutlined"

const  Sidebar = () => {

  const theme = useTheme();

  const colors = tokens(theme.palette.mode)

  const [isCollapsed,setIsCollapsed] = useState(false);

  const [selected,setSelected ] = useState('Dashboard')

  const Item = ({ title,to,icon,selected,setSelected}) => {

    const theme = useTheme();

    const colors = tokens(theme.palette.mode)

    return (

      <MenuItem

      active={ selected === title }

      style={{color : colors.greay[100] }}

      onClick={()=>setSelected(title)}

      icon={icon}

      >

      <Typography>

        {title}

      </Typography>

      <Link to={to} />

      </MenuItem>

    )

  }

  return (

    <Box

    sx={{

        "& .pro-sidebar-inner":{

          background : `${colors.primary[900]} !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 : "#6870"

        }

    }}

    >

      <ProSidebar

      collapsed={isCollapsed}

      >

    <Menu

    iconShape='square'

    >

    <MenuItem

    onClick={()=>setIsCollapsed(!isCollapsed)}

    icon={isCollapsed ? <MenuOutlinedIcon /> : undefined}

    style={{

      margin : '10px 0 20px 0',

      color : colors.greay[100]

    }}

    sx={{

    }}

    >

      {

        !isCollapsed && (

          <Box

          display="flex"

          justifyContent="space-between"

          alignItems="center"

          ml="15px"

          >

            <Typography variant='h3' color={colors.greay[100]} >

              ADMINIS

            </Typography>

            <IconButton onClick={()=> setIsCollapsed(!isCollapsed)} >

              <MenuOutlinedIcon/>

            </IconButton>

            </Box>

        )

      }

</MenuItem>

{/\* user  \*/}

{ !isCollapsed && (

  <Box mb='25px'>

<Box display='flex' justifyContent='center' alightItem='center' >

  <img

  alt='profile-user'

  width='100px'

  height='100px'

  src={`../../assets/user.jpg`}

  style={{cursor:'pointer',borderRadius:'50%'}}

  />

</Box>

  <Box textAlign='center'>

    <Typography variant='h2' color={colors.greay[100]} fontweight='bold' sx={{m:'10px 0 0 0'}} > Pooja ( Pillu ) </Typography>

    <Typography

    variant='h5'

    color={colors.greenAccent[400]}

    > Dr. Babasaheb Ambedkar Marathwada University</Typography>

  </Box>

  </Box>

)}

 {/\* MenuItem  \*/}

<Box

paddingLeft={isCollapsed ? undefined: '10%'}

>

  <Item

  title='Dashboard'

  to='/'

  icon={<HomeOutlinedIcon/>}

  selected={selected}

  setSelected={setSelected}

  />

  <Item

  title='Manage Team'

  to='/team'

  icon={<PeopleOutlinedIcion/>}

  selected={selected}

  setSelected={setSelected}

  />

  <Item

  title='Contact Information'

  to='/contacts'

  icon={<ContactsOutlinedIcon/>}

  selected={selected}

  setSelected={setSelected}

  />

  <Item

  title='Invoices Balances'

  to='/invoices'

  icon={<ReceiptOutlinedIcon/>}

  selected={selected}

  setSelected={setSelected}

  />

  <Item

  title='Profile Form'

  to='/form'

  icon={<PersonOutlinedIcon/>}

  selected={selected}

  setSelected={setSelected}

  />

  <Item

  title='Calendar'

  to='/calendar'

  icon={<CalendarTodayOutlinedIcon/>}

  selected={selected}

  setSelected={setSelected}

  />

  <Item

  title='FAQ Page'

  to='/faq'

  icon={<HelpOutlinedIcon/>}

  selected={selected}

  setSelected={setSelected}

  />

  <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'

  to='/geography'

  icon={<PublicOutlinedIcon/>}

  selected={selected}

  setSelected={setSelected}

  />

</Box>

</Menu>

</ProSidebar>

    </Box>

  )

}

export default Sidebar

This code sets up a **collapsible sidebar** component using react-pro-sidebar and **Material UI** in React. Let's break down each section to understand how it all comes together:

### 1. Imports

* **React and useState**: For managing component state, especially the sidebar’s collapsed state and selected items.
* **ProSidebar Components**: Menu, MenuItem, and ProSidebar are from react-pro-sidebar, used to build and style the sidebar.
* **Material UI Components**: Box, IconButton, Typography, useTheme for layout, buttons, and styling.
* **Icons**: Several icons from @mui/icons-material for menu items.
* **Router Link**: Link from react-router-dom for navigation within the app.
* **Theme Tokens**: tokens (presumably defined in theme.js) is used for color settings based on the theme.

### 2. Sidebar Component Setup

The Sidebar component manages two states:

* **isCollapsed**: To toggle between collapsed and expanded sidebar views.
* **selected**: To track which menu item is currently selected.

#### Nested Item Component

The Item component defines individual sidebar menu items with props:

* **title, to, icon**: Passed in to customize the menu item.
* **selected, setSelected**: Used to handle the active state of the menu item.

When clicked, each item updates the selected state to mark itself as active.

### 3. Sidebar Styling with sx

The Box component wraps ProSidebar and applies custom CSS with sx to style specific elements within the sidebar:

* **.pro-sidebar-inner**: Background color of the sidebar.
* **.pro-icon-wrapper**: Makes the icon backgrounds transparent.
* **.pro-inner-item**: Adjusts padding for each menu item.
* **.pro-inner-item:hover**: Changes text color on hover.
* **.pro-menu-item.active**: Highlights the active menu item.

### 4. Sidebar Collapse Toggle

The sidebar can toggle between collapsed and expanded states:

* The **collapse button** is the MenuOutlinedIcon in the top MenuItem. When clicked, isCollapsed toggles between true and false, expanding or collapsing the sidebar.
* **Conditional rendering**: When isCollapsed is false, the sidebar displays additional content, like the title, profile image, and other information.

### 5. Profile Section

When isCollapsed is false, this section displays:

* A **profile picture** and user details in a centered layout.
* Typography components for displaying the user's name and title.

### 6. Sidebar Menu Items

Menu items are added using the Item component, with icons and links:

* **Dynamic Styling**: active={selected === title} in each MenuItem applies active styling if the item’s title matches the selected state.
* **Navigation**: to and Link enable in-app navigation for each item.

The items cover a range of navigation options like “Dashboard,” “Manage Team,” “Contact Information,” and various chart and form routes.

### Summary of Sidebar Features

* **Collapsible** with a toggle button.
* **Custom styling** via Material UI’s sx and theme-based colors.
* **Dynamic menu items** with icons and active state handling.
* **Conditional rendering** for expanded content.

This setup is flexible and scalable for applications requiring a sidebar with nested items, dynamic styling, and theme adaptation. Let me know if you'd like additional customizations or further details on any section!

**Data Grid**

import { Typography, useTheme,   Box } from '@mui/material'

import { DataGrid } from '@mui/x-data-grid'

import { tokens } from '../../theme'

import { mockDataTeam } from '../../data/MockData'

import AdminPanelSettingsOutlinedIcon from '@mui/icons-material/AdminPanelSettingsOutlined'

import LockOpenOutlinedIcon from '@mui/icons-material/LockOpenOutlined'

import LockOpeneOutlinedIcon from '@mui/icons-material/LockOpenOutlined'

import Header from '../../components/Header'

import SecurityOutlinedIcon from '@mui/icons-material/SecurityOutlined';

const Team = () => {

    const theme = useTheme()

    const colors = tokens(theme.palette.mode);

    const columns = [

        {

            field : "id",

            headerName : "ID"

        },

        {

            field : "name",

            headerName : "NAME",

            flex : 1 ,

            cellClassName : "name-column--cell",

        },

        {

            field : "email",

            headerName : "EMAIL",

            flex : 1,

        },

        {

            field : "age",

            headerName : "AGE",

            type : "number",

            headerAlign : "left",

            align : "left",

        },

        {

            field : "phone",

            headerName : "Phone Number",

            flex : 1,

        },

        {

            field : "access",

            headerName : "ACCESS LEVEL",

            flex : 1,

            renderCell : ({ row : { access }}) => {

                return (

                    <Box

                    width='60%'

                    m='0 auto'

                    p='5px'

                    display='flex'

                    justifyContent='center'

                    backgroundColor={

                        access === 'admin' ?

                        colors.greenAccent[600] :

                        colors.greenAccent[700]

                    }

                    borderRadius='4px'

                    >

                    {

                        access === 'admin'

                        &&

                        <AdminPanelSettingsOutlinedIcon/>

                    }

                    {

                        access === 'manager'

                        &&

                        <SecurityOutlinedIcon/>

                    }

                    {

                        access === 'user'

                        &&

                        <LockOpeneOutlinedIcon/>

                    }

                    <Typography color={colors.greay[100]} sx={{ ml:'5px'}} >

                            { access }

                    </Typography>

                    </Box>

                )

            }

        }

    ]

  return (

    <Box m='20px'

    width='75vw'

    >

         <Header

      title='TEAM'

      subtitle='Managing the Team Members'

      />

        <Box

        m='40px'

        height='50vh'

        sx={{

            "& .MuiDataGrid-root": {

                border : "none"

            },

            "& .MuiDataGrid-cell":{

                borderBottom : "none"

            },

            "& .name-column--cell":{

                color : colors.greenAccent[300]

            },

            "& .MuiDataGrid-columnHeader":{

                backgroundColor : colors.blueAccent[700],

                borderBottom : "none"

            },

            "& .MuiDataGrid-virtualScroller" : {

                backgroundColor : colors.primary[400]

            },

            "& .MuiDataGrid-footerContainer":{

                 borderTop : "none",

                 backgroundColor : colors.blueAccent[700]

            }

        }}

        >

            <DataGrid

            rows={mockDataTeam}

            columns={columns}

            />

        </Box>

    </Box>

  )

}

export default Team

This code defines a **Team** component, which displays a **DataGrid** table using Material UI and renders data on team members, including attributes like their access level. Let’s go through each part of the code in detail:

### Imports

* **Material UI Components**: Typography, useTheme, and Box are used for typography, theming, and layout.
* **DataGrid**: A powerful table component from Material UI’s x-data-grid for displaying rows and columns.
* **Theme Tokens**: tokens from a theme.js file allows for dynamic theming.
* **Mock Data**: mockDataTeam is a mock data array of team members for the table.
* **Icons**: Icons like AdminPanelSettingsOutlinedIcon, SecurityOutlinedIcon, and LockOpenOutlinedIcon visually represent access levels.
* **Header Component**: This custom Header component renders the main title and subtitle of the page.

### Team Component Structure

The Team component uses hooks, theme-based colors, and conditional rendering to display team data with dynamic styles based on access levels.

#### 1. **Theme and Colors Setup**

Javascript

const theme = useTheme();

const colors = tokens(theme.palette.mode);

* useTheme(): Retrieves the current theme, allowing dynamic changes based on light/dark mode.
* tokens: Maps theme colors to specific palette modes (e.g., light or dark).

#### 2. **DataGrid Column Definitions**

The columns array defines each column of the DataGrid, including header names, field keys, and additional properties:

1. **ID Column**
   * Basic setup with field: "id" and headerName: "ID".
2. **Name Column**
   * flex: 1: Makes the column responsive, expanding to fill available space.
   * cellClassName: "name-column--cell": Adds a class for custom styling on name cells.
3. **Email Column**
   * Also uses flex: 1 for flexible sizing.
4. **Age Column**
   * type: "number": Specifies that this column is numeric.
   * headerAlign and align: Aligns the header and cells to the left.
5. **Phone Column**
   * Basic column setup with headerName: "Phone Number" and flexible sizing.
6. **Access Level Column**
   * This column customizes each cell based on the access level (e.g., admin, manager, user).
   * renderCell: Defines a custom cell renderer using a function that takes row as input.
   * Based on access, a **Box** component with different background colors and icons is rendered, showing the access type with a label.

### 3. Return JSX

The component renders two main sections:

1. **Header Section**

Javascript

<Header title="TEAM" subtitle="Managing the Team Members" />

* + Displays the Header component with a main title and subtitle.

1. **DataGrid Section**

<Box m="40px" height="50vh" sx={{ /\* Styles \*/ }}>

<DataGrid rows={mockDataTeam} columns={columns} />

</Box>

* + **Outer Box**: Provides margin and sets a fixed height for the DataGrid component.
  + **Custom Styling with sx**: Customizes the DataGrid using Material UI’s sx property:
    - .MuiDataGrid-root: Removes border around the entire grid.
    - .MuiDataGrid-cell: Removes the bottom border of each cell.
    - .name-column--cell: Sets the color of the "name" column cells.
    - .MuiDataGrid-columnHeader: Adds a blue background color to column headers.
    - .MuiDataGrid-virtualScroller: Sets the main grid background.
    - .MuiDataGrid-footerContainer: Styles the footer background color.

### 4. DataGrid Component

The DataGrid component is passed:

* **rows**: Uses mockDataTeam as the data source.
* **columns**: Uses the columns array, where each column is defined with specific properties and cell rendering logic.

### Summary

This Team component provides a clean, well-styled table that showcases team members’ details with a dynamically styled access level column. The combination of custom styling with sx and conditional rendering in renderCell makes the component both visually appealing and responsive to different themes and data inputs. Let me know if you want further detail on any part of this!

**useMediaQuery**

**useMediaQuery** is a custom React hook provided by libraries like Material-UI (now known as MUI) and others to help manage responsive designs by allowing you to respond to changes in media queries directly within your components. This hook simplifies the process of determining whether a particular media query matches the current viewport size, enabling you to conditionally render components or apply styles based on the device's screen size.

### Key Features:

1. **Responsive Design**: Easily implement responsive layouts by checking if the current screen size matches specific media queries.
2. **Dynamic Re-renders**: Components automatically re-render when the viewport changes, allowing for real-time responsiveness.
3. **Custom Breakpoints**: You can define custom breakpoints to fit your design requirements.

### Basic Usage:

Here's how to use useMediaQuery in a React component:

#### 1. Importing the Hook

If you're using Material-UI, you can import it like this:

Javascript

import { useMediaQuery } from '@mui/material';

#### 2. Using useMediaQuery

Here's a simple example of how to use useMediaQuery to conditionally render components based on the screen size:

Javascript

import React from 'react';

import { useMediaQuery } from '@mui/material';

const MyComponent = () => {

// Using useMediaQuery to check if the screen size is at least 600px wide

const isMobile = useMediaQuery('(max-width:600px)');

return (

<div>

{isMobile ? (

<h2>This is a mobile view!</h2>

) : (

<h2>This is a desktop view!</h2>

)}

</div>

);

};

export default MyComponent;

### Explanation of the Example:

* **Media Query**: In this example, the media query '(max-width:600px)' checks if the viewport width is 600 pixels or less.
* **Conditional Rendering**: Depending on whether the query matches, the component renders different content for mobile and desktop views.

### Using Custom Breakpoints

You can also define custom breakpoints if you need specific conditions:

Javascript

const isTablet = useMediaQuery('(min-width:600px) and (max-width:960px)');

### Combining with Other Libraries

useMediaQuery can also be combined with other styling approaches like styled-components or Emotion for more complex responsive designs.

### Conclusion

The useMediaQuery hook is a powerful tool for creating responsive React applications that adapt to different screen sizes. It enhances user experience by providing appropriate layouts and components based on the user's device. If you have any specific use cases or questions about useMediaQuery, feel free to ask!

**Yup**

The **Yup** library in JavaScript is used for object schema validation. It is particularly useful for validating data structures, ensuring that the data conforms to specific formats and rules. Here are some key features and uses of Yup:

### Key Features:

1. **Schema Creation**: Yup allows you to create schemas for validating objects. You can define the shape of an object, specifying the required fields and their types (e.g., strings, numbers, dates).
2. **Type Validation**: You can specify the expected data types for each field and validate whether the data conforms to those types.
3. **Asynchronous Validation**: Yup supports asynchronous validation, which is useful for cases like checking if an email is already in use (e.g., through an API call).
4. **Custom Validation**: You can create custom validation methods to enforce specific business rules that may not be covered by built-in validations.
5. **Error Messages**: Yup allows you to specify custom error messages for validation failures, making it easier to provide feedback to users.

### Common Use Cases:

* **Form Validation**: Yup is often used with form libraries like Formik to validate user inputs in web applications, ensuring that the data collected is correct before submission.
* **Data Parsing and Validation**: Yup can validate and parse data from APIs or other sources to ensure that it meets the expected structure.
* **Type Safety**: Yup enhances the type safety of JavaScript applications, especially when dealing with complex data structures.

### Example Usage:

Here's a simple example of how Yup can be used to validate a user registration form:

import \* as Yup from 'yup';

const validationSchema = Yup.object().shape({

username: Yup.string().required('Username is required'),

email: Yup.string().email('Invalid email format').required('Email is required'),

password: Yup.string().min(6, 'Password must be at least 6 characters long').required('Password is required'),

});

// Example of validating an object

const userInput = {

username: 'user123',

email: 'user@example.com',

password: 'password123',

};

validationSchema.validate(userInput)

.then(() => {

console.log('Validation succeeded!');

})

.catch(err => {

console.error('Validation error:', err.errors);

});

In this example, Yup defines a schema for a user registration form and validates a user input object against that schema. If the validation fails, it returns an error message.

Let me know if you have specific questions about using Yup or need more examples!

**Formik**

**Formik** is a popular library for managing forms in React applications. It simplifies the process of handling form state, validation, and submission, making it easier to build robust and user-friendly forms. Here are the key features and components of Formik:

**Key Features of Formik:**

1. **Form State Management**: Formik provides a structured way to manage form state, including values, errors, and touched fields, reducing the need for manual state management.
2. **Validation**: Formik integrates seamlessly with validation libraries like Yup to validate form data. You can define validation schemas and provide error messages based on user input.
3. **Field-Level Validation**: Formik allows you to validate individual fields as the user interacts with them, providing immediate feedback.
4. **Form Submission Handling**: It simplifies handling form submissions, including managing loading states and handling submission errors.
5. **Easy Integration**: Formik is easy to integrate with existing form libraries and custom components, allowing for flexibility in your form designs.

**Basic Components of Formik:**

1. **Formik**: The main component that wraps your form. It provides context to form fields and handles form state.
2. **Field**: A component that connects your input fields to Formik’s state. It automatically hooks into Formik's context to manage value and validation.
3. **ErrorMessage**: A component that displays error messages for a specific field based on the validation results.
4. **Form**: A wrapper component that handles form submission and prevents the default form behavior.

**Example Usage:**

Here’s a simple example of how to use Formik in a React application:

import React from 'react';

import { Formik, Form, Field, ErrorMessage } from 'formik';

import \* as Yup from 'yup';

const validationSchema = Yup.object().shape({

username: Yup.string().required('Username is required'),

email: Yup.string().email('Invalid email format').required('Email is required'),

password: Yup.string().min(6, 'Password must be at least 6 characters long').required('Password is required'),

});

const MyForm = () => {

return (

<Formik

initialValues={{ username: '', email: '', password: '' }}

validationSchema={validationSchema}

onSubmit={(values, { setSubmitting }) => {

// Handle form submission

console.log('Form data', values);

setSubmitting(false); // Set submitting to false after submission

}}

>

{({ isSubmitting }) => (

<Form>

<div>

<label htmlFor="username">Username:</label>

<Field type="text" name="username" />

<ErrorMessage name="username" component="div" />

</div>

<div>

<label htmlFor="email">Email:</label>

<Field type="email" name="email" />

<ErrorMessage name="email" component="div" />

</div>

<div>

<label htmlFor="password">Password:</label>

<Field type="password" name="password" />

<ErrorMessage name="password" component="div" />

</div>

<button type="submit" disabled={isSubmitting}>

Submit

</button>

</Form>

)}

</Formik>

);

};

export default MyForm;

**Explanation of the Example:**

1. **Initial Values**: initialValues prop sets the starting values for the form fields.
2. **Validation Schema**: Uses Yup to define validation rules for each field.
3. **onSubmit**: A function that handles what happens when the form is submitted, such as sending data to an API.
4. **Field Component**: Each input field is wrapped with a <Field> component to connect it to Formik’s state.
5. **ErrorMessage Component**: Displays error messages related to validation.

**Conclusion:**

Formik simplifies form handling in React by managing state, validation, and submission seamlessly. When used in conjunction with Yup, it allows for powerful and easy-to-use form validation. If you have specific questions or want to see more advanced usage, feel free to ask!

The whole code is below with their explanation >>>

Source code >>>

import { Box,Button,TextField } from '@mui/material'

import { Formik , Form  } from 'formik'

import \* as yup from 'yup'

import useMediaQuery from '@mui/material/useMediaQuery'

import Header from '../../components/Header'

Header

function FormComponent() {

    const isNonMobile = useMediaQuery("(min-width:600px)")

    const initialValues = {

        firstName : '',

        lastName : '',

        email : '',

        contact : '',

        address1 : '',

        address2  : '',

    }

    const userSchema = yup.object().shape({

        firstName : yup.string().required('required'),

        lastName : yup.string().required('required'),

        email : yup

        .string()

        .email('Invalid email')

        .required('required'),

        contact : yup

        .string()

        .required('required'),

        address1 : yup.string().required('required'),

        address2 : yup.string().required('required'),

    })

    const handleFormSubmit = (values) => {

        // values.preventDefault()

        console.log(values);

    }

  return (

    <Box

    m='20px'

    >

{/\* Header and Title  \*/}

        <Header

        title='CREATE USER'

        subtitle='Create a New User Profile'

        />

        {/\* Formik Form  \*/}

        <Formik

        onSubmit={handleFormSubmit}

        initialValues={initialValues}

        validationSchema={userSchema}

        >

          {({values,errors,touched,handleBlur,handleChange,handleSubmit}) => (

            <Form onSubmit={handleSubmit} >

                <Box

                display='grid'

                gap='30px'

                gridTemplateColumns='repeat(4,minmax(0,1fr))'

                sx={{

                  "& > div" : { gridColumn : isNonMobile ? undefined :

                    "span 4"

                  }

                }}

                >

                  <TextField

                  fullWidth

                  variant='filled'

                  type='text'

                  label='First Name'

                  onBlur={handleBlur}

                  onChange={handleChange}

                  value={values.firstName}

                  name='firstName'

                  error={!!touched.firstName && !!errors.firstName }

                  helperText={touched.firstName && errors.firstName}

                  sx={{gridColumn:'span 2'}}

                  />

                  <TextField

                  fullWidth

                  variant='filled'

                  type='text'

                  label='Last Name'

                  onBlur={handleBlur}

                  onChange={handleChange}

                  value={values.lastName}

                  name='lastName'

                  error={!!touched.lastName && !!errors.lastName }

                  helperText={touched.lastName && errors.lastName}

                  sx={{gridColumn:'span 2'}}

                  />

<TextField

                  fullWidth

                  variant='filled'

                  type='text'

                  label='Email'

                  onBlur={handleBlur}

                  onChange={handleChange}

                  value={values.email}

                  name='email'

                  error={!!touched.email && !!errors.email }

                  helperText={touched.email && errors.email}

                  sx={{gridColumn:'span 4'}}

                  />

                  <TextField

                  fullWidth

                  variant='filled'

                  type='text'

                  label='Contact Number'

                  onBlur={handleBlur}

                  onChange={handleChange}

                  value={values.contact}

                  name='contact'

                  error={!!touched.contact && !!errors.contact }

                  helperText={touched.contact && errors.contact}

                  sx={{gridColumn:'span 4'}}

                  />

                <TextField

                  fullWidth

                  variant='filled'

                  type='text'

                  label='Address 1'

                  onBlur={handleBlur}

                  onChange={handleChange}

                  value={values.address1}

                  name='address1'

                  error={!!touched.address1 && !!errors.address1 }

                  helperText={touched.address1 && errors.address1}

                  sx={{gridColumn:'span 4'}}

                  />

<TextField

                  fullWidth

                  variant='filled'

                  type='text'

                  label='Address 2'

                  onBlur={handleBlur}

                  onChange={handleChange}

                  value={values.address2}

                  name='address2'

                  error={!!touched.address2 && !!errors.address2}

                  helperText={touched.address2 && errors.address2}

                  sx={{gridColumn:'span 4'}}

                  />

                </Box>

                <Button type="submit" variant="contained" color="primary">

                            Submit

                        </Button>

              </Form>

          )}

        </Formik>

    </Box>

  )

}

export default FormComponent

Here's a detailed breakdown of each part of the code. This explanation will highlight potential points where issues might arise and provide insights to help avoid them.

**1. Imports:**

import { Box, Button, TextField } from '@mui/material';

import { Formik, Form } from 'formik';

import \* as yup from 'yup';

import useMediaQuery from '@mui/material/useMediaQuery';

import Header from '../../components/Header';

* **Box, Button, and TextField from @mui/material**: Material-UI components provide layout (Box), button functionality (Button), and text input fields (TextField).
* **Formik and Form from formik**: Formik is a library for handling form state, validation, and submission. The Form component replaces the HTML <form> tag and links Formik to form handling.
* **yup**: A library used to create validation schemas to validate form input data.
* **useMediaQuery**: This hook detects screen size changes and is used here to make layout adjustments for responsive design.
* **Header Component**: A custom component for displaying the header. This line (Header) appears to be mistakenly left in the code and can be removed as it's unused.

**2. Defining the Form Component and isNonMobile:**

function FormComponent() {

const isNonMobile = useMediaQuery("(min-width:600px)");

* **Function Component**: FormComponent is the main function that renders the form UI.
* **isNonMobile**: Uses useMediaQuery to check if the screen width is at least 600px, typically indicating a non-mobile device. This variable controls the grid layout, making the form responsive.

**3. initialValues Object:**

const initialValues = {

firstName: '',

lastName: '',

email: '',

contact: '',

address1: '',

address2: '',

};

* **Purpose**: Sets initial values for form fields (e.g., firstName, email). Without this, Formik wouldn't know the default values for each field.
* **Potential Issue**: If you forget to include any field in initialValues, Formik won’t recognize that field, and it won’t be managed properly.

**4. userSchema for Validation:**

const userSchema = yup.object().shape({

firstName: yup.string().required('required'),

lastName: yup.string().required('required'),

email: yup.string().email('Invalid email').required('required'),

contact: yup.string().required('required'),

address1: yup.string().required('required'),

address2: yup.string().required('required'),

});

* **Purpose**: Defines validation rules for each form field using yup. For example, email must be a valid email format, and all fields are required.
* **Potential Issue**: If the validation schema doesn’t match the form fields exactly (e.g., if you miss a field or make a typo), Formik will skip validation for that field.

**5. handleFormSubmit Function:**

const handleFormSubmit = (values) => {

console.log(values);

};

* **Purpose**: Function to handle form submission. values contains the form data when the form is submitted.
* **Potential Issue**: It’s easy to forget to add logic here. Right now, it just logs the form values, but in a real-world application, you might need to send this data to an API or handle it further.

**6. JSX Structure:**

return (

<Box m='20px'>

<Header title='CREATE USER' subtitle='Create a New User Profile' />

* **Box**: A Material-UI Box component used as a container for the form. m='20px' applies margin for spacing.
* **Header**: Displays a title and subtitle for the form.

**7. Formik Setup:**

<Formik

onSubmit={handleFormSubmit}

initialValues={initialValues}

validationSchema={userSchema}

>

* **Purpose**: Initializes Formik with properties:
  + **onSubmit**: Points to handleFormSubmit, which will be triggered on form submission.
  + **initialValues**: Sets the default values for the form fields.
  + **validationSchema**: Tells Formik to use yup to validate the form based on userSchema.
* **Potential Issue**: Make sure handleFormSubmit is defined, or the form won’t submit properly.

**8. Form Render and Handling with Formik:**

{({ values, errors, touched, handleBlur, handleChange, handleSubmit }) => (

<Form onSubmit={handleSubmit}>

* **Destructured Functions**:
  + **values**: Contains current form values.
  + **errors**: Holds validation errors after yup checks.
  + **touched**: Tracks if a field has been focused on (visited) to determine when to show errors.
  + **handleBlur, handleChange, handleSubmit**: Functions to manage field blur, input changes, and form submission.

**9. Form Layout with Box and TextField Components:**

<Box

display='grid'

gap='30px'

gridTemplateColumns='repeat(4,minmax(0,1fr))'

sx={{ "& > div": { gridColumn: isNonMobile ? undefined : "span 4" } }}

>

* **Grid Layout**: Sets up a CSS grid with 4 columns. When the screen width is below 600px, each TextField spans all 4 columns.
* **Responsive Styling with sx**: Applies responsive grid layout for mobile (single-column) vs. non-mobile (multi-column).

**10. TextField Components:**

Each TextField component (for firstName, lastName, email, etc.) is rendered with properties:

<TextField

fullWidth

variant='filled'

type='text'

label='First Name'

onBlur={handleBlur}

onChange={handleChange}

value={values.firstName}

name='firstName'

error={!!touched.firstName && !!errors.firstName}

helperText={touched.firstName && errors.firstName}

sx={{ gridColumn: 'span 2' }}

/>

* **Properties**:
  + **fullWidth**: Makes the input take the full width of its container.
  + **variant='filled'**: Uses the "filled" style variant for Material-UI text fields.
  + **type and label**: Sets the field type and placeholder label.
  + **onBlur and onChange**: Links form events to Formik’s functions to track user input.
  + **value and name**: Links each field to the respective Formik value and name, helping Formik recognize and handle each input.
  + **error and helperText**: Show error messages if the field is touched and contains an error.
* **Potential Issues**:
  + If name doesn’t match initialValues or validationSchema, Formik won’t track that field correctly.
  + The sx grid settings (like gridColumn: 'span 2') should align with the desired layout to avoid layout issues.

**11. Submit Button:**

<Button type="submit" variant="contained" color="primary">

Submit

</Button>

* **Purpose**: Submits the form. Setting type="submit" ensures the form triggers Formik’s handleSubmit when clicked.

**12. Closing Components and Exporting:**

export default FormComponent;

* **Purpose**: Exports FormComponent for use in other parts of the application.

**Summary**

This code sets up a responsive form using Formik for form management and yup for validation. To avoid errors:

* Ensure initialValues, validationSchema, and form field names match exactly.
* handleFormSubmit should be defined to handle form submission data.
* Ensure sx settings align with your layout design.