```
===== /home/alissu/Desktop/nansen_Web/frontend/src/layout/Sidebar/components/MenuSideBar.ts
x =====
import React, { useState } from "react";
import {
  MenuFoldOutlined,
  MenuUnfoldOutlined,
  UserOutlined,
} from "@ant-design/icons";
import { Layout, Menu } from "antd";
import { Link } from "react-router-dom";

import Logo from "../../../assets/nansen_logo.svg";
import LogoMini from "../../../assets/logo_mini.png";

const { Sider } = Layout;

const MenuSideBar: React.FC = () => {
  const [collapsed, setCollapsed] = useState<boolean>(false);

  const toggleCollapsed = () => {
    setCollapsed(!collapsed);
  };

  const menuItems = [
    {
      key: "1",
      icon: <UserOutlined />,
      label: <Link to="/">Home</Link>,
    },
    {
      key: "2",
      icon: <UserOutlined />,
      label: <Link to="/create">Create</Link>,
    },
  ];

  return (
    <Layout style={{ height: "100vh" }}>
      <Sider
        trigger={null}
        collapsible
        collapsed={collapsed}
        style={{ padding: 0, background: "#FFF" }}
      >
        <div
          className="demo-logo-vertical"
          style={{
            display: "flex",
            justifyContent: "center",
            alignItems: "center",
            marginTop: 10,
          }}
        >
          <img
            src={collapsed ? LogoMini : Logo}
            alt="Logo"
            style={{
              height: collapsed ? "auto" : 70,
              marginTop: 30,
              marginLeft: 20,
            }}
          />
        </div>

        <Menu
          style={{ padding: 0, background: "#FFF", marginTop: 20 }}
          theme="dark"
          mode="inline"
```

```
          defaultSelectedKeys={["1"]}
          items={menuItems}
        />
      </Sider>
    </Layout>
  );
};

export default MenuSideBar;
===== /home/alissu/Desktop/nansen_Web/frontend/src/layout/Sidebar/components/DropDownMenu.t
sx =====
import React, { useState } from "react";
import { MenuOutlined } from "@ant-design/icons";
import { Dropdown, Space, MenuProps } from "antd";
import { Link } from "react-router-dom";

const menuItems: MenuProps["items"] = [
  { key: "/home", label: <Link to="/home">Home</Link> },
  { key: "/products", label: <Link to="/products">Produtos</Link> },
  { key: "/equipments", label: <Link to="/equipments">Equipamentos</Link> },
  { key: "/users", label: <Link to="/users">UsuÃ¡rios</Link> },
  { key: "/sectors", label: <Link to="/sectors">Setores</Link> },
  { key: "/production-lines", label: <Link to="/production-lines">Linhas de ProduÃ§Ã£o</Lin
k> },
  { key: "/iotdevices", label: <Link to="/iotdevices">Dispositivos IoT</Link> },
  { key: "/monitoring", label: <Link to="/monitoring">Monitoramentos</Link> },
  { key: "/quizzes", label: <Link to="/quizzes">Quizzes</Link> },
  { key: "/missions", label: <Link to="/missions">MissÃµes</Link> },
  {
    key: "logout",
    label: <span style={{ color: "red", fontWeight: "bold" }}>Sair</span>,
  },
];

const DropDownMenu: React.FC = () => {
  const [open, setOpen] = useState(false);

  return (
    <>
      {open && (
        <div
          onClick={() => setOpen(false)}
          style={{
            position: "fixed",
            top: 0,
            left: 0,
            width: "100%",
            height: "100vh",
            backgroundColor: "rgba(0, 0, 0, 0.4)",
            zIndex: 999,
          }}
        />
      )}

      <Dropdown
        menu={{ items: menuItems }}
        trigger={["click"]}
        open={open}
        onOpenChange={(flag) => setOpen(flag)}
        overlayStyle={{
          position: "fixed",
          top: 60,
          left: 0,
          width: "80%",
          backgroundColor: "rgba(255, 255, 255, 0.9)",
          zIndex: 1000,
        }}
      >
        <a onClick={(e) => e.preventDefault()} style={{ display: "block", width: "100%", cu
```

```
rsor: "pointer" }}>
          <Space style={{ justifyContent: "space-between", width: "100%", height: "50%" }}>
            <MenuOutlined />
          </Space>
        </a>
      </Dropdown>
    </>
  );
};

export default DropDownMenu;
===== /home/alissu/Desktop/nansen_Web/frontend/src/layout/Sidebar/components/Logo.tsx =====
import React from "react";
import { Link } from "react-router-dom";
import LogoLarge from "../../../assets/nansen_logo.svg";
import LogoMini from "../../../assets/logo_mini.svg";

interface LogoProps {
  collapsed: boolean;
}

const Logo: React.FC<LogoProps> = ({ collapsed }) => {
  return (
    <div
      style={{
        display: "flex",
        justifyContent: "center",
        alignItems: "center",
        background: "rgb(0 66 129)",
        padding: "10px 0",
        height: 64,
      }}
    >
      <Link to="/">
        <img
          src={collapsed ? LogoMini : LogoLarge}
          alt="Logo"
          style={{
            height: 50,
            width: "auto",
            maxWidth: "160px",
          }}
        />
      </Link>
    </div>
  );
};

export default Logo;
===== /home/alissu/Desktop/nansen_Web/frontend/src/layout/Sidebar/components/LogoInova.tsx
=====
import React from "react";
import { Link } from "react-router-dom";
import LogoInovaImage from "../../../assets/inova_logo.png";

interface LogoInovaProps {
  collapsed: boolean;
}

const LogoInova: React.FC<LogoInovaProps> = ({ collapsed }) => {
  return (
    <div
      style={{
        position: "relative",
        bottom: "0px",
        display: "flex",
        justifyContent: "center",
        alignItems: "center",
        width: "100%",
```

```
        padding: collapsed ? "5px" : "10px",
        transition: "all 0.3s ease-in-out",
      }}
    >
      <Link to="/">
        <img
          src={LogoInovaImage}
          alt="Logo Inova"
          style={{
            height: collapsed ? 20 : 40,
            width: collapsed ? 50 : 100,
            transition: "all 0.3s ease-in-out",
          }}
        />
      </Link>
    </div>
  );
};

export default LogoInova;
===== /home/alissu/Desktop/nansen_Web/frontend/src/layout/Sidebar/components/ItensMenu.tsx
=====
import React from "react";
import { Menu } from "antd";
import { useNavigate, useLocation } from "react-router-dom";
import { MenuProps } from "antd/es/menu";
import { useAuth } from "../../../contexts/auth/AuthContext";

// Ã\215cones
import HomeIcon from "@mui/icons-material/Home";
import LocalOfferIcon from "@mui/icons-material/LocalOffer";
import BuildIcon from "@mui/icons-material/Build";
import PeopleIcon from "@mui/icons-material/People";
import AccountTreeIcon from "@mui/icons-material/AccountTree";
import MemoryIcon from "@mui/icons-material/Memory";
import MonitorIcon from "@mui/icons-material/Monitor";
import QuizIcon from "@mui/icons-material/Quiz";
import ExploreIcon from "@mui/icons-material/Explore";
import FactoryIcon from "@mui/icons-material/Factory";
import ShoppingCartIcon from "@mui/icons-material/ShoppingCart";
import ExitToAppIcon from "@mui/icons-material/ExitToApp";

const ItensMenu: React.FC = () => {
  const navigate = useNavigate();
  const location = useLocation();
  const { logout } = useAuth();

  const items: MenuProps["items"] = [
    {
      key: "/home",
      icon: <HomeIcon fontSize="small" />,
      label: "Home",
      onClick: () => navigate("/home"),
    },
    {
      key: "/sectors",
      icon: <AccountTreeIcon fontSize="small" />,
      label: "Setores",
      onClick: () => navigate("/sectors"),
    },
    {
      key: "/production-lines",
      icon: <FactoryIcon fontSize="small" />,
      label: "Linhas de ProduÃ§Ã£o",
      onClick: () => navigate("/production-lines"),
    },
    {
      key: "/equipments",
      icon: <BuildIcon fontSize="small" />,
```

```
    label: "Equipamentos",
    onClick: () => navigate("/equipments"),
  },
  {
    key: "/products",
    icon: <LocalOfferIcon fontSize="small" />,
    label: "Produtos",
    onClick: () => navigate("/products"),
  },
  {
    key: "/users",
    icon: <PeopleIcon fontSize="small" />,
    label: "UsuÃ¡rios",
    onClick: () => navigate("/users"),
  },
  {
    key: "/iotdevices",
    icon: <MemoryIcon fontSize="small" />,
    label: "Dispositivos IoT",
    onClick: () => navigate("/iotdevices"),
  },
  {
    key: "monitoring-group",
    icon: <MonitorIcon fontSize="small" />,
    label: "Monitoramentos",
    children: [
      {
        key: "/monitoring",
        label: "NANSENic",
        onClick: () => navigate("/monitoring"),
      },
      {
        key: "/sensor-monitoring",
        label: "NANSENsor",
        onClick: () => navigate("/sensor-monitoring"),
      },
    ],
  },
  {
    key: "loja-group",
    icon: <ShoppingCartIcon fontSize="small" />,
    label: "Loja",
    children: [
      {
        key: "/loja",
        label: "Produtos Loja",
        onClick: () => navigate("/loja"),
      },
      {
        key: "/loja/register",
        label: "Cadastrar Produto",
        onClick: () => navigate("/loja/register"),
      },
    ],
  },
  {
    key: "/quizzes",
    icon: <QuizIcon fontSize="small" />,
    label: "Quizzes",
    onClick: () => navigate("/quizzes"),
  },
  {
    key: "/missions",
    icon: <ExploreIcon fontSize="small" />,
    label: "MissÃµes",
    onClick: () => navigate("/missions"),
  },
  { key: "divider", type: "divider" },
  {
```

```
      key: "logout",
      icon: <ExitToAppIcon fontSize="small" />,
      label: <span style={{ color: "red" }}>Sair</span>,
      onClick: logout,
    },
  ];

  return (
    <Menu
      className="custom-menu"
      mode="inline"
      selectedKeys={[location.pathname]}
      items={items}
    />
  );
};

export default ItensMenu;

===== /home/alissu/Desktop/nansen_Web/frontend/src/layout/Sidebar/ItemSideBar.tsx =====
import React, { useState, useEffect } from "react";
import { Layout, Button } from "antd";
import { MenuFoldOutlined, MenuUnfoldOutlined } from "@ant-design/icons";
import ItensMenu from "./components/ItensMenu";
import Logo from "./components/Logo";
import LogoInova from "./components/LogoInova";

const { Sider } = Layout;

const ItemSideBar: React.FC = () => {
  const [collapsed, setCollapsed] = useState(false);
  const [isMobile, setIsMobile] = useState<boolean>(window.innerWidth < 1024);

  useEffect(() => {
    const handleResize = () => {
      setIsMobile(window.innerWidth < 1024);
    };

    window.addEventListener("resize", handleResize);
    return () => {
      window.removeEventListener("resize", handleResize);
    };
  }, []);

  if (isMobile) return null;

  return (
    <Sider
      className="menu flex flex-column justify-content-between shadow-2 bg-white"
      trigger={null}
      collapsible
      collapsed={collapsed}
      width={220}
      style={{ minHeight: "100vh" }} // nÃ£o tem utilitÃ¡rio direto pra 100vh no antd, mas
se quiser tirar, use height: "100vh" em CSS externo
    >
      <div>
        <Logo collapsed={collapsed} />
        <div className="flex justify-content-center my-3">
          <Button
            type="text"
            icon={collapsed ? <MenuUnfoldOutlined /> : <MenuFoldOutlined />}
            onClick={() => setCollapsed(!collapsed)}
            className="p-0"
            style={{ fontSize: "24px", color: "#000" }}
          />
        </div>
        <ItensMenu />
      </div>
```

```
      <LogoInova collapsed={collapsed} />
    </Sider>
  );
};

export default ItemSideBar;

===== /home/alissu/Desktop/nansen_Web/frontend/src/layout/Header/components/DrawMenu.tsx ==
===
import React, { useState, useEffect } from "react";
import { Button, Drawer } from "antd";
import { MenuOutlined } from "@ant-design/icons";
import { Link } from "react-router-dom";

const menuItems = [
  { key: "/home", label: "Home" },
  { key: "/products", label: "Produtos" },
  { key: "/equipments", label: "Equipamentos" },
  { key: "/users", label: "UsuÃ¡rios" },
  { key: "/sectors", label: "Setores" },
  { key: "/production-lines", label: "Linhas de ProduÃ§Ã£o" },
  { key: "/iotdevices", label: "Dispositivos IoT" },
  { key: "/monitoring", label: "Monitoramentos" },
  { key: "/quizzes", label: "Quizzes" },
  { key: "/missions", label: "MissÃµes" },
  { key: "logout", label: "Sair", logout: true },
];

const DrawMenu: React.FC = () => {
  const [open, setOpen] = useState(false);
  const [isMobile, setIsMobile] = useState(window.innerWidth < 1024);

  useEffect(() => {
    const handleResize = () => setIsMobile(window.innerWidth < 1024);
    window.addEventListener("resize", handleResize);
    return () => window.removeEventListener("resize", handleResize);
  }, []);

  if (!isMobile) return null;

  return (
    <>
      <Button onClick={() => setOpen(true)} style={{ marginLeft: 10 }}>
        <MenuOutlined />
      </Button>
      <Drawer
        title="Acesso RÃ¡pido"
        onClose={() => setOpen(false)}
        open={open}
        bodyStyle={{ padding: "16px" }}
      >
        <div style={{ display: "flex", flexDirection: "column", gap: "10px" }}>
          {menuItems.map((item) =>
            item.logout ? (
              <span
                key={item.key}
                style={{
                  color: "red",
                  fontWeight: "bold",
                  padding: "10px 0",
                  cursor: "pointer",
                }}
              >
                {item.label}
              </span>
            ) : (
              <Link
                key={item.key}
```

```
                to={item.key}
                style={{ padding: "10px 0", fontSize: "16px" }}
              >
                {item.label}
              </Link>
            )
          )}
        </div>
      </Drawer>
    </>
  );
};

export default DrawMenu;

===== /home/alissu/Desktop/nansen_Web/frontend/src/layout/Header/components/ItemHeaderCabec
alho.tsx =====
import React from "react";
import { Typography, Space, Divider } from "antd";

const { Title, Text } = Typography;

interface ItemHeaderCabecalhoProps {
  title: string;
  subTitle?: string;
}

const ItemHeaderCabecalho: React.FC<ItemHeaderCabecalhoProps> = ({ title, subTitle }) => {
  return (
    <Space direction="vertical" style={{ width: "100%", marginBottom: "20px" }}>
      <Title level={2} style={{ color: "#0a2a66" }}>{title}</Title>
      {subTitle && <Text type="secondary">{subTitle}</Text>}
      <Divider />
    </Space>
  );
};

export default ItemHeaderCabecalho;
===== /home/alissu/Desktop/nansen_Web/frontend/src/layout/Header/ItemHeader.tsx =====
import { Layout } from "antd";
import DrawMenu from "./components/DrawMenu";

const { Header } = Layout;

export default function ItemHeader() {
  return (
    <Header
      style={{
        padding: "0 0px",
        background: "rgb(0 66 129)",
        borderBottom: "1px solid #e8e8e8",
        height: 65, // MantÃ©m altura fixa para alinhar com o Sidebar
        display: "flex",
        alignItems: "center",
        justifyContent: "space-between",
      }}
    >
      <div className="menu_hamburguer">
        <DrawMenu />
      </div>

      <div style={{ fontSize: "24px", fontWeight: "bold", color: "#FFF" }}>
        <h1></h1>
      </div>

      <div>
        {/* EspaÃ§o reservado para Ã-cones de usuÃ¡rio/notificaÃ§Ãµes */}
      </div>
    </Header>
```

```
  );
}

===== /home/alissu/Desktop/nansen_Web/frontend/src/types/IoTDevice.ts =====
export interface IoTDevice {
  id: number;
  name: string;
  deveui?: string | null;
  type_device?: string | null;
  equipement?: number | null;
}
===== /home/alissu/Desktop/nansen_Web/frontend/src/types/subsectionTypes.ts =====
export interface Subsection {
    id: number;
    description?: string | null;
    is_monitored: boolean;
    section: number | null;
    deviceIot?: number | null;
  }

===== /home/alissu/Desktop/nansen_Web/frontend/src/types/ProductionLinesTypes.ts =====
export interface ProductionLine {
  id: number;
  name: string;
  description?: string | null;
  value_mensuration_estimated: number;
  setor?: number | null;
  created_at: string;
}

export interface ProductionLineCreate {
  name: string;
  description?: string | null;
  value_mensuration_estimated: number;
  setor?: number | null;
}
===== /home/alissu/Desktop/nansen_Web/frontend/src/types/monitoringTypes.ts =====
export interface MonitoringItem {
    id: number;
    name: string;
    description: string;
    estimated_consumption: number;
    created_at: string;
  }
===== /home/alissu/Desktop/nansen_Web/frontend/src/types/missions.ts =====
// src/types/missions.ts
export interface MissionItem {
  id: number;
  users: number[];
  name: string;
  description: string;
  //quantity_na: number; // sempre vem como nÃºmero
  energy_meta: number;
  nansen_coins: number;
  quantity_xp: number;
  status: "Pendente" | "Em Andamento" | "Finalizada";
  date_start: string; // ISO date-time
  date_end: string | null; // ISO date-time ou null
  order_production: number | null;
  quantity_product: number | null;
  is_order_production: boolean;
  monitoring: number | null; // ID do sensor
  product: number | null;
}

===== /home/alissu/Desktop/nansen_Web/frontend/src/types/products.ts =====
export interface ProductItem {
    id: number;
    name: string;
```

```
    description?: string | null;
    created_at?: string;
    photo?: string | null;
  }


===== /home/alissu/Desktop/nansen_Web/frontend/src/types/lojaTypes.ts =====
export interface ProductLojaItem {
  id: number;
  name: string;
  description: string;
  price: number;
  quantity: number;
  image?: string;
  disponivel: boolean;
  created_at: string;
  updated_at: string;
}

===== /home/alissu/Desktop/nansen_Web/frontend/src/types/quizzes.ts =====
export interface QuizItem {
    id: number;
    name: string;
    description: string;
    hour_start: string;
    hour_end: string;
  }
===== /home/alissu/Desktop/nansen_Web/frontend/src/types/users.ts =====
export interface UserItem {
  id: number;
  username: string;
  email: string;
  name: string | null;
  role: "ADMIN" | "LIDER" | "GAME";
  avatar_url: string;
  avatar: string;
}

export interface UserRegister {
  username: string;
  name: string;
  email: string;
  password: string;
  role: "ADMIN" | "LIDER" | "GAME";
}

===== /home/alissu/Desktop/nansen_Web/frontend/src/types/sections.ts =====
// types/sections.ts
export interface SectionItem {
  id: number;
  name: string;
  description: string | null;
  is_monitored: boolean;
  monitoring: number | null;

  // AssociaÃ§Ãµes
  setor: number | null;
  productionLine: number | null;
  equipament: number | null;
  DeviceIot: number | null;

  // Tipo da seÃ§Ã£o ("SETOR", "LINHA", "EQUIPAMENTO")
  type_section: number | null;

  // ReferÃªncia Ã  seÃ§Ã£o pai
  secticon_parent: number | null;

  // Sub-seÃ§Ãµes aninhadas
  sections_filhas?: SectionItem[];
```

```
  // Campos adicionais
  estimated_consumption?: number;
  power?: number | null;
  tension?: number | null;
  min_consumption?: number | null;
  max_consumption?: number | null;

  // Tipo literal opcional para renderizaÃ§Ã£o
  type?: "sector" | "productionLine" | "equipment";
}
===== /home/alissu/Desktop/nansen_Web/frontend/src/types/sectors.ts =====
export interface Sector {
  id: number;
  name: string;
  description?: string;
  estimated_consumption: number;
  created_at: string;
}
===== /home/alissu/Desktop/nansen_Web/frontend/src/types/equipaments.ts =====
export interface EquipamentItem {
  id: number;
  name: string;
  description?: string | null;
  power: number | null;
  tension: number | null;
  energy_consumption: number | null;
  max_consumption: number | null;
  min_consumption: number | null;
  production_line: number | null;
  created_at: string;
}
===== /home/alissu/Desktop/nansen_Web/frontend/src/vite-env.d.ts =====
/// <reference types="vite/client" />

===== /home/alissu/Desktop/nansen_Web/frontend/src/styles/base/reset.css =====
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body, html {
    width: 100%;
    height: 100%;
}

ul, ol {
    list-style: none;
}

a {
    text-decoration: none;
    color: inherit;
}

button {
    cursor: pointer;
    font-family: inherit;
}

img {
    max-width: 100%;
    height: auto;
}
===== /home/alissu/Desktop/nansen_Web/frontend/src/styles/base/variables.css =====
:root {
    --primary-color: #004281;
    --primary-hover: #003366;
```

```
    --secondary-color: #6c757d;
    --danger-color: #d75c5d;
    --background-light: #f8f9fa;
    --background-dark: #242424;
    --text-light: rgba(2, 2, 2, 0.87);
    --text-dark: #213547;

    /* Fontes */
    --font-family: Inter, system-ui, Avenir, Helvetica, Arial, sans-serif;
    --font-size-title: 28px;
    --font-size-subtitle: 14px;

    /* Layout */
    --border-radius: 8px;
}
===== /home/alissu/Desktop/nansen_Web/frontend/src/styles/custom/custom.css =====
.card_style {
  border-left: 3px solid rgb(0, 66, 129);
}

.card_content_style {
  min-width: 100%;
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 0;
}
.card_content_title {
  font-size: 16px;
  font-weight: 500;
}

.card_conten_value {
  font-size: 40px;
  font-weight: 500;
  color: rgb(0, 66, 129);
}

/* Ajuste para telas entre 740px e 1220px */
@media (min-width: 750px) and (max-width: 1200px) {
  .card_content_title {
    font-size: 13px;
    font-weight: bold;
  }
  .card_conten_value {
    font-size: 26px; /* Diminui o nÃºmero */
  }
}

.card_style {
  border-left: 3px solid rgb(0, 66, 129);
}

.card_content_style {
  min-width: 100%;
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 0;
}
.card_content_title {
  font-size: 16px;
  font-weight: 500;
}

.card_conten_value {
  font-size: 40px;
  font-weight: 500;
  color: rgb(0, 66, 129);
```

```
}

/* Ajuste para telas entre 740px e 1220px */
@media (min-width: 750px) and (max-width: 1200px) {
  .card_content_title {
    font-size: 13px;
    font-weight: bold;
  }
  .card_conten_value {
    font-size: 26px; /* Diminui o nÃºmero */
  }
}

===== /home/alissu/Desktop/nansen_Web/frontend/src/styles/components/buttons.css =====
@import "../base/variables.css";

.primary-btn {
    background-color: var(--primary-color);
    border: none;
    color: white;
    padding: 10px 15px;
    border-radius: var(--border-radius);
    transition: background 0.3s ease-in-out;
}

.primary-btn:hover {
    background-color: var(--primary-hover);
}

.filter-btn {
    border-bottom: 2px solid #1890ff;
}

.edit-btn {
    background-color: var(--primary-color);
    border: none;
    color: white;
}

.edit-btn:hover {
    background-color: var(--primary-hover);
}

.delete-btn {
    background-color: var(--danger-color);
    border: none;
    color: white;
}

.delete-btn:hover {
    background-color: #b94647;
}
===== /home/alissu/Desktop/nansen_Web/frontend/src/styles/components/header.css =====

===== /home/alissu/Desktop/nansen_Web/frontend/src/styles/global/tables.css =====
.table-container {
    background: white;
    padding: 16px;
    border-radius: var(--border-radius);
    width: 100%;
    overflow-x: auto;
}

.table-header {
    font-size: var(--font-size-title);
    font-weight: bold;
    color: var(--text-dark);
    margin-bottom: 15px;
}
```

```
@media (max-width: 1024px) {
    .table-container {
        padding: 10px;
    }

    .table-header {
        font-size: 20px;
    }
}


.actions {
    display: flex;
    gap: 8px;
}
===== /home/alissu/Desktop/nansen_Web/frontend/src/styles/global/sidebar.css =====
/* ð\237\223\214 Sidebar (Menu lateral) */
.custom-menu {
    padding: 0;
    background: #FFF;
    margin-top: 20px;
    border: none;

}

.custom-menu .ant-menu-item {
    display: flex;
    align-items: center;
    font-size: 14px;
    font-weight: 500;
    color: var(--primary-color) !important;
    transition: background 0.3s;
}

.custom-menu .ant-menu-item:hover {
    background-color: #d9eaff !important;
}

.custom-menu .ant-menu-item a {
    color: var(--primary-color) !important;
    text-decoration: none;
}

.custom-menu .ant-menu-item-icon {
    color: var(--primary-color) !important;
}

/* ð\237\223\214 Sidebar (Menu estilo dark) */
.ant-menu-dark .ant-menu-item {
    display: flex;
    margin: 0 auto;
    background-color: #FFF;
    font-weight: 400;
    border: solid 0px #b6b3b3;
    line-height: 14px;
    padding: 25px;
    margin-top: 10px;
    color: #0d0d0d;
    font-size: 16px;
}

.ant-menu-dark .ant-menu-item:hover {
    background-color: rgb(250, 250, 250);
}

/* ð\237\223\214 Header (barra superior) */
.header-app {
```

```
    width: 100%;
    height: 100%;
    display: flex;
}

/* Menu lateral responsivo */
.item_menu_draw {
    display: block;
    background-color: #e8eaef70;
    margin-top: 10px;
    padding: 15px;
    border-radius: 5px;
    font-size: 14px;
    color: #555867;
}

.item_menu_draw:hover {
    color: #FFF;
}

.header-card .header-itens {
    display: flex;
    flex-direction: column;
    align-items: baseline;
    margin: 10px;
}

.header-itens-botton {
    display: flex;
    justify-content: baseline;
    align-items: center;
}

.header-itens-botton span {
    margin-left: 10px;
}
===== /home/alissu/Desktop/nansen_Web/frontend/src/styles/global/globals.css =====
@import "../base/reset.css";
@import "../base/variables.css";
@import "../components/buttons.css";
@import "../global/tables.css";

/* Layout geral */
.layout-container {
    display: flex;
    min-height: 100vh;
}

.content-container {
    flex: 1;
    display: flex;
    flex-direction: column;
    max-width: 100%;
    overflow-x: auto;
}

.content {
    padding: 20px;
    background-color: var(--background-light);
}

/* SeÃ§Ã£o de AÃ§Ãµes */
.actions-section {
    display: flex;
    gap: 10px;
    margin-bottom: 20px;
    align-items: center;
}
```

```
/* Ajuste para manter os botÃµes alinhados corretamente */
.actions {
    display: flex;
    gap: 8px;
    align-items: center;
}

/* Garantir que a tabela fique responsiva */
.table-wrapper {
    width: 100%;
    overflow-x: auto;
    margin-top: 10px;
}
===== /home/alissu/Desktop/nansen_Web/frontend/src/contexts/auth/AuthContext.tsx =====
import React, { createContext, useContext, useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";


interface AuthContextProps {
  token: string | null;
  login: (token: string) => void;
  logout: () => void;
}


const AuthContext = createContext<AuthContextProps | undefined>(undefined);

export const AuthProvider = ({ children }: { children: React.ReactNode }) => {
  const [token, setToken] = useState<string | null>(localStorage.getItem("userToken"));
  const navigate = useNavigate();

  useEffect(() => {
    const storedToken = localStorage.getItem("userToken");
    if (storedToken) {
      setToken(storedToken);
    }
  }, []);


  const login = (newToken: string) => {
    localStorage.setItem("userToken", newToken);
    setToken(newToken);
    navigate("/home");
  };


  const logout = () => {
    localStorage.removeItem("userToken");
    setToken(null);
    navigate("/login");
  };

  return (
    <AuthContext.Provider value={{ token, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
};


export const useAuth = () => {
  const context = useContext(AuthContext);
  if (!context) {
    throw new Error("useAuth deve ser usado dentro de um AuthProvider");
  }
  return context;
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/App.tsx =====
import Routers from "./router/Routers";
```

```
function App() {
  return <Routers />;
}

export default App;
===== /home/alissu/Desktop/nansen_Web/frontend/src/main.tsx =====
import React from "react";
import ReactDOM from "react-dom/client";
import { BrowserRouter } from "react-router-dom";
import { AuthProvider } from "./contexts/auth/AuthContext";
import App from "./App";
import "./index.css";

ReactDOM.createRoot(document.getElementById("root") as HTMLElement).render(
  <React.StrictMode>
    <BrowserRouter>
      <AuthProvider>
        <App />
      </AuthProvider>
    </BrowserRouter>
  </React.StrictMode>
);
===== /home/alissu/Desktop/nansen_Web/frontend/src/router/ProtectedRoute.tsx =====
import { useAuth } from "../contexts/auth/AuthContext";
import { ReactNode } from "react";
import { Navigate } from "react-router-dom";

interface ProtectedRouteProps {
  children: ReactNode;
}

const ProtectedRoute = ({ children }: ProtectedRouteProps) => {
  const { token } = useAuth();

  if (!token) {
    return <Navigate to="/login" replace />;
  }

  return children;
};

export default ProtectedRoute;
===== /home/alissu/Desktop/nansen_Web/frontend/src/router/Routers.tsx =====
// src/router/Routers.tsx
import { Routes, Route, Navigate, Outlet } from "react-router-dom";
import ProtectedRoute from "./ProtectedRoute";

// Home e Login
import HomePage from "../pages/home/Home";
import LoginPage from "../pages/login/Login";

// Produtos
import ProductsPage from "../pages/products/Products";
import ProductsRegister from "../pages/products/ProductsRegister/Register";
import EditProducts from "../pages/products/components/EditProducts";

// Equipamentos
import EquipmentsPage from "../pages/equipments/Equipments";
import EquipmentsRegister from "../pages/equipments/equipmentsregister/Register";
import EditEquipments from "../pages/equipments/components/EditEquipment";

// UsuÃ¡rios
import UsersPage from "../pages/users/Users";
import UsersRegister from "../pages/users/components/UsersRegister";

// Setores
import SectorsPage from "../pages/sectors/Sectors";
import SectorsRegister from "../pages/sectors/components/SectorsRegister";
```

```
import SectorsEdit from "../pages/sectors/components/SectorsEdit";

// Dispositivos IoT
import IoTDevice from "../pages/iotDevices/IoTDevices";
import IoTDeviceRegister from "../pages/iotDevices/components/IoTDevicesRegister";
import IoTDeviceEdit from "../pages/iotDevices/components/IoTDevicesEdit";

// Linhas de ProduÃ§Ã£o
import ProductionLinesPage from "../pages/productionLines/ProductionLines";
import ProductionLinesRegister from "../pages/productionLines/components/ProductionLinesReg
ister";
import ProductionLinesEdit from "../pages/productionLines/components/ProductionLinesEdit";

// Monitoramento (NansenIC)
import MonitoringPage from "../pages/monitoring/Monitoring";
import MonitoringRegister from "../pages/monitoring/components/MonitoringForm";
import MonitoringConfigure from "../pages/monitoring/components/MonitoringConfigure";
import MonitoringAddSection from "../pages/monitoring/components/MonitoringAddSection";
import MonitoringEdit from "../pages/monitoring/components/MonitoringEdit";
import SectionList from "../pages/monitoring/components/SectionList";
import SectionEdit from "../pages/monitoring/components/SectionEdit";

// Monitoramento (NansenSensor â\200\223 mock)
import MonitoringSensor from "../pages/monitoring-sensor/MonitoringSensor";
import MonitoringSensorForm from "../pages/monitoring-sensor/components/MonitoringForm";
import SectionListSensor from "../pages/monitoring-sensor/components/SectionList";
import MonitoringAddSectionSensor from "../pages/monitoring-sensor/components/MonitoringAdd
Section";
import MonitoringSensorEdit from "../pages/monitoring-sensor/components/MonitoringEdit";
import SectionEditSensor from "../pages/monitoring-sensor/components/SectionEdit";

// Loja
import LojaProductsPage from "../pages/loja/LojaProducts";
import LojaProductsRegister from "../pages/loja/LojaProductsRegister";
import LojaProductsEdit from "../pages/loja/LojaProductsEdit";

// Quizzes
import QuizzesPage from "../pages/quizzes/Quizzes";
import QuizRegister from "../pages/quizzes/quizregister/Register";
import EditQuiz from "../pages/quizzes/components/EditQuiz";

// MissÃµes
import MissionsPage from "../pages/missions/Missions";
import MissionRegister from "../pages/missions/missionregister/Register";
import EditMission from "../pages/missions/components/EditMission";
export default function Routers() {
  return (
    <Routes>
      {/* Rota pÃºblica */}
      <Route path="/login" element={<LoginPage />} />

      {/* Bloco de rotas protegidas */}
      <Route
        element={
          <ProtectedRoute>
            <Outlet />
          </ProtectedRoute>
        }
      >
        {/* Home */}
        <Route path="/home" element={<HomePage />} />

        {/* Produtos */}
        <Route path="/products" element={<ProductsPage />} />
        <Route path="/products/register" element={<ProductsRegister />} />
        <Route path="/products/edit/:id" element={<EditProducts />} />

        {/* Equipamentos */}
        <Route path="/equipments" element={<EquipmentsPage />} />
```

```
<Route path="/equipments/register" element={<EquipmentsRegister />} />
<Route path="/equipments/edit/:id" element={<EditEquipments />} />

{/* UsuÃ¡rios */}
<Route path="/users" element={<UsersPage />} />
<Route path="/users/register" element={<UsersRegister />} />

{/* Setores */}
<Route path="/sectors" element={<SectorsPage />} />
<Route path="/sectors/register" element={<SectorsRegister />} />
<Route path="/sectors/edit/:id" element={<SectorsEdit />} />

{/* Dispositivos IoT */}
<Route path="/iotdevices" element={<IoTDevice />} />
<Route path="/iotdevices/register" element={<IoTDeviceRegister />} />
<Route path="/iotdevices/edit/:id" element={<IoTDeviceEdit />} />

{/* Linhas de ProduÃ§Ã£o */}
<Route path="/production-lines" element={<ProductionLinesPage />} />
<Route
  path="/production-lines/register"
  element={<ProductionLinesRegister />}
/>
<Route
  path="/production-lines/edit/:id"
  element={<ProductionLinesEdit />}
/>

{/* Monitoramento (NansenIC) */}
<Route path="/monitoring" element={<MonitoringPage />} />
<Route path="/monitoring/register" element={<MonitoringRegister />} />
<Route
  path="/monitoring/configure/:id"
  element={<MonitoringConfigure />}
/>
<Route
  path="/monitoring/configure/:id/sections"
  element={<SectionList />}
/>
<Route
  path="/monitoring/add-section/:id"
  element={<MonitoringAddSection />}
/>
<Route path="/monitoring/edit/:id" element={<MonitoringEdit />} />
<Route path="/monitoring/edit-section/:id" element={<SectionEdit />} />

{/* Monitoramento (NansenSensor) */}
<Route path="/sensor-monitoring" element={<MonitoringSensor />} />
<Route
  path="/sensor-monitoring/register"
  element={<MonitoringSensorForm />}
/>
<Route
  path="/sensor-monitoring/configure/:id"
  element={<SectionListSensor />}
/>
<Route
  path="/sensor-monitoring/add-section/:id"
  element={<MonitoringAddSectionSensor />}
/>
<Route
  path="/sensor-monitoring/edit/:id"
  element={<MonitoringSensorEdit />}
/>
<Route
  path="/sensor-monitoring/edit-section/:id"
  element={<SectionEditSensor />}
/>
```

```
        {/* Loja */}
        <Route path="/loja" element={<LojaProductsPage />} />
        <Route path="/loja/register" element={<LojaProductsRegister />} />
        <Route path="/loja/edit/:id" element={<LojaProductsEdit />} />

        {/* Quizzes */}
        <Route path="/quizzes" element={<QuizzesPage />} />
        <Route path="/quizzes/register" element={<QuizRegister />} />
        <Route path="/quizzes/edit/:id" element={<EditQuiz />} />

        {/* MissÃµes */}
        <Route path="/missions" element={<MissionsPage />} />
        <Route path="/missions/register" element={<MissionRegister />} />
        <Route path="/missions/edit/:id" element={<EditMission />} />

        {/* Redirecionamentos */}
        <Route path="/" element={<Navigate to="/home" replace />} />
        <Route path="*" element={<Navigate to="/home" replace />} />
      </Route>
    </Routes>
  );
}

===== /home/alissu/Desktop/nansen_Web/frontend/src/router/types.ts =====

===== /home/alissu/Desktop/nansen_Web/frontend/src/utils/constants.ts =====

===== /home/alissu/Desktop/nansen_Web/frontend/src/utils/validation.ts =====

===== /home/alissu/Desktop/nansen_Web/frontend/src/utils/format.ts =====

===== /home/alissu/Desktop/nansen_Web/frontend/src/utils/formatData.ts =====

===== /home/alissu/Desktop/nansen_Web/frontend/src/services/ClaimService.ts =====
// src/services/ClaimService.ts
import api from "./api";

export interface ClaimItem {
  id: number;
  data_claim: string;
  description: string;
  user_claim: number;
  reward: number;
}

// GET /claims/?user_claim={userId}
export function getUserClaims(userId: number): Promise<ClaimItem[]> {
  return api
    .get<ClaimItem[]>("/claims/", { params: { user_claim: userId } })
    .then((r) => r.data);
}

===== /home/alissu/Desktop/nansen_Web/frontend/src/services/subsectionService.ts =====
import api from "./api";
import { SubsectionItem } from "../types/subsectionTypes";

// Buscar todas as subseÃ§Ãµes
export const getSubsections = async (): Promise<SubsectionItem[]> => {
  const response = await api.get("/sub_sections/");
  return response.data;
};

// Buscar uma subseÃ§Ã£o especÃ-fica
export const getSubsectionById = async (id: number): Promise<SubsectionItem> => {
  const response = await api.get(`/sub_sections/${id}/`);
  return response.data;
};

// Criar uma nova subseÃ§Ã£o
```

```
export const createSubsection = async (subsection: Partial<SubsectionItem>): Promise<void>
=> {
  await api.post("/sub_sections/", subsection);
};

// Atualizar uma subseÃ§Ã£o (PUT)
export const updateSubsection = async (id: number, subsection: Partial<SubsectionItem>): Pr
omise<void> => {
  await api.put(`/sub_sections/${id}/`, subsection);
};

// Excluir uma subseÃ§Ã£o
export const deleteSubsection = async (id: number): Promise<void> => {
  await api.delete(`/sub_sections/${id}/`);
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/services/productsService.ts =====
import api from "./api";
import { ProductionLine, ProductionLineCreate } from "../types/ProductionLinesTypes";

// ð\237\224¹ Listar todas as linhas de produÃ§Ã£o
export const getProductionLines = async (): Promise<ProductionLine[]> => {
  const response = await api.get<ProductionLine[]>("/production_lines/");
  return response.data;
};

// ð\237\224¹ Obter uma linha de produÃ§Ã£o por ID
export const getProductionLineById = async (id: number): Promise<ProductionLine> => {
  const response = await api.get<ProductionLine>(`/production_lines/${id}/`);
  return response.data;
};

// ð\237\224¹ Criar uma nova linha de produÃ§Ã£o
export const createProductionLine = async (data: ProductionLineCreate): Promise<ProductionL
ine> => {
  const response = await api.post<ProductionLine>("/production_lines/", data);
  return response.data;
};

// ð\237\224¹ Atualizar uma linha de produÃ§Ã£o existente
export const updateProductionLine = async (id: number, data: ProductionLineCreate): Promise
<ProductionLine> => {
  const response = await api.put<ProductionLine>(`/production_lines/${id}/`, data);
  return response.data;
};

// ð\237\224¹ Deletar uma linha de produÃ§Ã£o
export const deleteProductionLine = async (id: number): Promise<void> => {
  await api.delete(`/production_lines/${id}/`);
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/services/api.ts =====
// src/services/api.ts
import axios from "axios";

export const BASE_URL = `http://200.129.168.197:20163/api`;

const api = axios.create({
  baseURL: BASE_URL,
  headers: { "Content-Type": "application/json" },
});

// Adicionar token automaticamente se existir
api.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem("userToken");
    if (token) config.headers.Authorization = `Bearer ${token}`;
    return config;
  },
  (error) => Promise.reject(error)
);
```

```
// Interceptor de resposta
api.interceptors.response.use(
  (response) => response,
  async (error) => {
    const status = error.response?.status;
    const url = error.config?.url || "";
    const method = error.config?.method || "";

    // 1) Se for 401 **na rota de login**, sÃ³ rejeita para
    // que o catch local exiba a notificaÃ§Ã£o
    if (
      status === 401 &&
      method.toLowerCase() === "post" &&
      url.endsWith("/login/")
    ) {
      return Promise.reject(error);
    }

    // 2) SenÃ£o, continua seu fluxo de refresh / logout habitual
    if (status === 401) {
      const refreshToken = localStorage.getItem("refreshToken");
      if (refreshToken) {
        try {
          const r = await axios.post(`${BASE_URL}/auth/refresh/`, {
            refresh: refreshToken,
          });
          const newAccess = r.data.access;
          localStorage.setItem("userToken", newAccess);
          error.config.headers["Authorization"] = `Bearer ${newAccess}`;
          return axios(error.config);
        } catch {
          localStorage.removeItem("userToken");
          localStorage.removeItem("refreshToken");
          window.location.href = "/login";
        }
      } else {
        localStorage.removeItem("userToken");
        window.location.href = "/login";
      }
    }

    return Promise.reject(error);
  }
);

export default api;

===== /home/alissu/Desktop/nansen_Web/frontend/src/services/lojaService.ts =====
import api from "./api";
import { ProductLojaItem } from "../types/lojaTypes";

// ð\237\224¹ Listar produtos da loja
export const getStoreProducts = async (): Promise<ProductLojaItem[]> =>
  (await api.get("/products_loja/")).data;

// ð\237\224¹ Criar novo produto da loja
export const createStoreProduct = async (
  data: FormData
): Promise<ProductLojaItem> => (await api.post("/products_loja/", data)).data;

// ð\237\224¹ Atualizar produto existente
export const updateStoreProduct = async (
  id: number,
  data: FormData
): Promise<ProductLojaItem> =>
  (await api.put(`/products_loja/${id}/`, data)).data;

// ð\237\224¹ Deletar produto
```

```
export const deleteStoreProduct = async (id: number): Promise<void> =>
  await api.delete(`/products_loja/${id}/`);

===== /home/alissu/Desktop/nansen_Web/frontend/src/services/IoTDevicesService.ts =====
import api from "./api";
import { IoTDevice } from "../types/IoTDevice";

// Buscar todos os dispositivos IoT
export const getIoTDevices = async (): Promise<IoTDevice[]> => {
  const response = await api.get("/device_iots/");
  return response.data;
};

// Buscar um dispositivo IoT especÃ-fico
export const getIoTDeviceById = async (id: number): Promise<IoTDevice> => {
  const response = await api.get(`/device_iots/${id}/`);
  return response.data;
};

// Criar um novo dispositivo IoT
export const createIoTDevice = async (device: Partial<IoTDevice>): Promise<void> => {
  await api.post("/device_iots/", device);
};

// Atualizar um dispositivo IoT (PUT)
export const updateIoTDevice = async (id: number, device: Partial<IoTDevice>): Promise<void
> => {
  await api.put(`/device_iots/${id}/`, device);
};

// Atualizar parcialmente um dispositivo IoT (PATCH)
export const patchIoTDevice = async (id: number, device: Partial<IoTDevice>): Promise<void>
 => {
  await api.patch(`/device_iots/${id}/`, device);
};

// Excluir um dispositivo IoT
export const deleteIoTDevice = async (id: number): Promise<void> => {
  await api.delete(`/device_iots/${id}/`);
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/services/UsersService.ts =====
// src/services/UsersService.ts
import api from "./api";
import { UserItem, UserRegister } from "../types/users";

// GET /users/
export async function getUsers(): Promise<UserItem[]> {
  const resp = await api.get<UserItem[]>("/users/");
  return resp.data;
}

// POST /register/  â\206\220 endpoint correto para cadastro de usuÃ¡rio
export async function createUser(data: UserRegister): Promise<UserItem> {
  const resp = await api.post<UserItem>("/register/", data);
  return resp.data;
}

// PUT /users/{id}/
export async function updateUser(
  id: number,
  data: Partial<UserItem>
): Promise<UserItem> {
  const resp = await api.put<UserItem>(`/users/${id}/`, data);
  return resp.data;
}

// DELETE /users/{id}/
export async function deleteUser(id: number): Promise<void> {
  await api.delete(`/users/${id}/`);
```

```
}

===== /home/alissu/Desktop/nansen_Web/frontend/src/services/SectionsService.ts =====
import api from "./api";

// Buscar todas as seÃ§Ãµes
export const getSections = async (): Promise<any[]> => {
  const response = await api.get("/sections/");
  return response.data;
};

// Buscar uma seÃ§Ã£o especÃfica
export const getSectionById = async (id: number): Promise<any> => {
  const response = await api.get(`/sections/${id}/`);
  return response.data;
};

// Criar uma nova seÃ§Ã£o
export const createSection = async (section: Partial<any>): Promise<void> => {
  await api.post("/sections/", section);
};

// Atualizar uma seÃ§Ã£o (PUT)
export const updateSection = async (
  id: number,
  section: Partial<any>
): Promise<void> => {
  await api.put(`/sections/${id}/`, section);
};

// Excluir uma seÃ§Ã£o
export const deleteSection = async (id: number): Promise<void> => {
  await api.delete(`/sections/${id}/`);
};

// Buscar tipos de seÃ§Ã£o (SETOR, LINHA, EQUIPAMENTO)
export const getTypeSections = async (): Promise<
  { id: number; name: string }[]
> => {
  const response = await api.get("/typesection/");
  return response.data;
};

// Buscar mediÃ§Ãµes de energia para uma seÃ§Ã£o
// Use hÃfen ASCII normal (U+002D) em "section-measurements"
export async function getSectionMeasurements(
  sectionId: number
): Promise<{ interval: number; energia_ativa_kWh: number }[]> {
  const { data } = await api.get(
    `/section-measurements/?section_id=${sectionId}`
  );
  return data;
}

===== /home/alissu/Desktop/nansen_Web/frontend/src/services/MissionService.ts =====
// src/services/MissionService.ts
import api from "./api";
import { MissionItem } from "../types/missions";

export async function getMissions(): Promise<MissionItem[]> {
  const response = await api.get<MissionItem[]>("/missions/");
  return response.data;
}

export async function createMission(
  payload: Omit<MissionItem, "id">
): Promise<MissionItem> {
  const response = await api.post<MissionItem>("/missions/", payload);
  return response.data;
```

```
}

export async function updateMission(
  id: number,
  payload: Partial<Omit<MissionItem, "id">>
): Promise<MissionItem> {
  const response = await api.put<MissionItem>(`/missions/${id}/`, payload);
  return response.data;
}

export async function deleteMission(id: number): Promise<void> {
  await api.delete(`/missions/${id}/`);
}

export async function associateMissionToMonitoring(
  missionId: number,
  monitoringId: number
): Promise<MissionItem> {
  // Reutiliza o endpoint de atualizaÃ§Ã£o parcial para alterar sÃ³ o campo "monitoring"
  const response = await api.patch<MissionItem>(`/missions/${missionId}/`, {
    monitoring: monitoringId,
  });
  return response.data;
}

===== /home/alissu/Desktop/nansen_Web/frontend/src/services/equipmentsService.ts =====
import api from "./api";

// FunÃ§Ã£o para listar todos os equipamentos
export const getEquipments = async () => {
  try {
    const response = await api.get("/equipaments/");
    return response.data;
  } catch (error) {
    console.error("Erro ao buscar equipamentos:", error);
    throw error;
  }
};

// FunÃ§Ã£o para criar um novo equipamento
export const createEquipment = async (data: any) => {
  try {
    const response = await api.post("/equipaments/", data);
    return response.data;
  } catch (error) {
    console.error("Erro ao criar equipamento:", error);
    throw error;
  }
};

// FunÃ§Ã£o para editar um equipamento
export const updateEquipment = async (id: number, data: any) => {
  try {
    const response = await api.put(`/equipaments/${id}/`, data);
    return response.data;
  } catch (error) {
    console.error("Erro ao editar equipamento:", error);
    throw error;
  }
};

// FunÃ§Ã£o para excluir um equipamento
export const deleteEquipment = async (id: number) => {
  try {
    await api.delete(`/equipaments/${id}/`);
  } catch (error) {
    console.error("Erro ao excluir equipamento:", error);
    throw error;
  }
```

```
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/services/ProductionLinesService.ts =====
import api from "./api";
import { ProductionLine, ProductionLineCreate } from "../types/ProductionLinesTypes";

// ð\237\224¹ Buscar todas as linhas de produÃ§Ã£o
export const getProductionLines = async (): Promise<ProductionLine[]> => {
  try {
    const response = await api.get<ProductionLine[]>("/production_lines/");
    return response.data;
  } catch (error) {
    console.error("Erro ao buscar linhas de produÃ§Ã£o:", error);
    throw error;
  }
};

// ð\237\224¹ Buscar uma linha de produÃ§Ã£o pelo ID
export const getProductionLineById = async (id: number): Promise<ProductionLine> => {
  try {
    const response = await api.get<ProductionLine>(`/production_lines/${id}/`);
    return response.data;
  } catch (error) {
    console.error(`Erro ao buscar a linha de produÃ§Ã£o com ID ${id}:`, error);
    throw error;
  }
};

// ð\237\224¹ Criar uma nova linha de produÃ§Ã£o
export const createProductionLine = async (data: ProductionLineCreate): Promise<ProductionL
ine> => {
  try {
    const response = await api.post<ProductionLine>("/production_lines/", data);
    return response.data;
  } catch (error) {
    console.error("Erro ao criar linha de produÃ§Ã£o:", error);
    throw error;
  }
};

// ð\237\224¹ Atualizar uma linha de produÃ§Ã£o existente
export const updateProductionLine = async (id: number, data: ProductionLineCreate): Promise
<ProductionLine> => {
  try {
    const response = await api.put<ProductionLine>(`/production_lines/${id}/`, data);
    return response.data;
  } catch (error) {
    console.error(`Erro ao atualizar a linha de produÃ§Ã£o com ID ${id}:`, error);
    throw error;
  }
};

// ð\237\224¹ Deletar uma linha de produÃ§Ã£o
export const deleteProductionLine = async (id: number): Promise<void> => {
  try {
    await api.delete(`/production_lines/${id}/`);
  } catch (error) {
    console.error(`Erro ao deletar a linha de produÃ§Ã£o com ID ${id}:`, error);
    throw error;
  }
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/services/RewardService.ts =====
// src/services/RewardService.ts
import api from "./api";

export interface RewardItem {
  id: number;
  description: string;
  points: number;
  type_reward: "TIPO_REWARD_A" | "TIPO_REWARD_B";
```

```
  mission: number | null;
}

// named export â\234\205
export function getRewardById(id: number): Promise<RewardItem> {
  return api.get<RewardItem>(`/rewards/${id}/`).then((r) => r.data);
}

===== /home/alissu/Desktop/nansen_Web/frontend/src/services/AchievementService.ts =====
// src/services/AchievementService.ts
import api from "./api";

export interface AchievementItem {
  id: number;
  description: string;
  nansen_coins: number;
  quantity_xp: number;
  nivel: number;
  created_at: string;
  updated_at: string;
  user_achievement: number;
  mission: number | null;
}

// GET /achivements/?user_achievement={userId}
export function getUserAchievements(
  userId: number
): Promise<AchievementItem[]> {
  return api
    .get<AchievementItem[]>("/achivements/", {
      params: { user_achievement: userId },
    })
    .then((r) => r.data);
}

===== /home/alissu/Desktop/nansen_Web/frontend/src/services/QuizService.ts =====
import api from "../services/api";
import { QuizItem } from "../types/quizzes";

// ð\237\224¹ Listar todos os quizzes
export const getQuizzes = async (): Promise<QuizItem[]> => {
  const response = await api.get("/quizzes/");
  return response.data;
};

// ð\237\224¹ Criar um novo quiz
export const createQuiz = async (quizData: Partial<QuizItem>) => {
  return await api.post("/quizz/create/", quizData);
};

// ð\237\224¹ Atualizar um quiz existente
export const updateQuiz = async (id: number, quizData: Partial<QuizItem>) => {
  const response = await api.put(`/quizs/${id}/`, quizData);
  return response.data;
};

// ð\237\224¹ Deletar um quiz
export const deleteQuiz = async (id: number) => {
  await api.delete(`/quizs/${id}/`);
};

===== /home/alissu/Desktop/nansen_Web/frontend/src/services/SectorsService.ts =====
import api from "./api";
import { Sector } from "../types/sectors";

export const getSectors = async (): Promise<Sector[]> => {
  const response = await api.get<Sector[]>("/setors/");
  return response.data;
};
```

```
export const getSectorById = async (id: number): Promise<Sector> => {
  const response = await api.get<Sector>(`/setors/${id}/`);
  return response.data;
};

export const createSector = async (data: Omit<Sector, "id" | "created_at">) => {
  const response = await api.post("/setors/", data);
  return response.data;
};


export const updateSector = async (id: number, data: Omit<Sector, "id" | "created_at">) =>
{
  const response = await api.put(`/setors/${id}/`, data);
  return response.data;
};


export const deleteSector = async (id: number) => {
  await api.delete(`/setors/${id}/`);
};

===== /home/alissu/Desktop/nansen_Web/frontend/src/services/monitoringService.ts =====
import api from "./api";
import { MonitoringItem } from "../types/monitoringTypes";

// ð\237\224¹ Buscar todos os monitoramentos
export const fetchMonitorings = async (): Promise<MonitoringItem[]> => {
  const response = await api.get("/monitorings/");
  return response.data;
};

// ð\237\224¹ Buscar monitoramento por ID
export const fetchMonitoringById = async (
  id: number
): Promise<MonitoringItem> => {
  const response = await api.get(`/monitorings/${id}/`);
  return response.data;
};

// ð\237\224¹ Contar monitoramentos ativos
export const fetchActiveMonitoringCount = async (): Promise<number> => {
  const response = await api.get("/monitoring-active-count/");
  return response.data;
};

// ð\237\224¹ Criar monitoramento
export const createMonitoring = async (
  data: Partial<MonitoringItem>
): Promise<MonitoringItem> => {
  const response = await api.post("/monitorings/", data);
  return response.data;
};

// ð\237\224¹ Atualizar monitoramento
export const updateMonitoring = async (
  id: number,
  data: Partial<MonitoringItem>
): Promise<MonitoringItem> => {
  const response = await api.put(`/monitorings/${id}/`, data);
  return response.data;
};

// ð\237\224¹ Excluir monitoramento
export const deleteMonitoring = async (id: number): Promise<void> => {
  await api.delete(`/monitorings/${id}/`);
};
```

```
===== /home/alissu/Desktop/nansen_Web/frontend/src/components/actions/Actions.tsx =====
import React from "react";
import { Button, Tooltip, Popconfirm } from "antd";
import {
  EyeOutlined,
  EditOutlined,
  DeleteOutlined,
  SettingOutlined,
} from "@ant-design/icons";

interface ActionsProps {
  onView?: () => void;
  onEdit?: () => void;
  onDelete?: () => void;
  onConfigure?: () => void;
  onSubmit?: () => void;
  onCancel?: () => void;
}

const Actions: React.FC<ActionsProps> = ({
  onView,
  onEdit,
  onDelete,
  onConfigure,
  onSubmit,
  onCancel,
}) => {
  return (
    <div
      style={{
        display: "flex",
        gap: "10px",
        justifyContent: "flex-end",
        marginTop: "20px",
      }}
    >
      {onView && (
        <Tooltip title="Visualizar">
          <Button icon={<EyeOutlined />} className="default-btn" onClick={onView} />
        </Tooltip>
      )}
      {onEdit && (
        <Tooltip title="Editar">
          <Button
            type="primary"
            icon={<EditOutlined />}
            className="primary-btn"
            onClick={onEdit}
          >
            Editar
          </Button>
        </Tooltip>
      )}
      {onConfigure && (
        <Tooltip title="Configurar">
          <Button
            icon={<SettingOutlined />}
            className="default-btn"
            onClick={onConfigure}
          >
            Configurar
          </Button>
        </Tooltip>
      )}
      {onDelete && (
        <Popconfirm
          title="Deseja excluir este item?"
          onConfirm={onDelete}
          okText="Sim"
```

```
          cancelText="NÃ£o"
        >
          <Tooltip title="Excluir">
            <Button
              type="primary"
              icon={<DeleteOutlined />}
              className="danger-btn"
              danger
            >
              Excluir
            </Button>
          </Tooltip>
        </Popconfirm>
      )}
      {onCancel && (
        <Button type="default" onClick={onCancel}>
          Cancelar
        </Button>
      )}
      {onSubmit && (
        <Button type="primary" className="primary-btn" onClick={onSubmit}>
          Enviar
        </Button>
      )}
    </div>
  );
};

export default Actions;
===== /home/alissu/Desktop/nansen_Web/frontend/src/components/selects/SelectField.tsx =====
// src/components/selects/SelectField.tsx
import React, { useEffect, useState } from "react";
import { Select, Button, Spin } from "antd";

interface SelectFieldProps {
  name: string;
  label: string;
  value?: number;
  onChange: (name: string, value: number) => void;
  options?: { value: number; label: string }[];
  fetchOptions?: () => Promise<{ value: number; label: string }[]>;
  placeholder?: string;
  onCreateNew?: () => void;
}

const SelectField: React.FC<SelectFieldProps> = ({
  name,
  label,
  value,
  onChange,
  options = [],
  fetchOptions,
  placeholder,
  onCreateNew,
}) => {
  const [loading, setLoading] = useState(false);
  const [selectOptions, setSelectOptions] = useState<{ value: number; label: string }[]>(op
tions);

  useEffect(() => {
    if (fetchOptions) {
      setLoading(true);
      fetchOptions()
        .then((data) =>
          setSelectOptions(
            data.map((option) => ({
              value: Number(option.value), // ð\237\224¹ Garante que ‘value‘ seja sempre um
 nÃºmero
              label: option.label,
```

```
            }))
          )
        )
        .catch((error) => console.error(`Erro ao buscar opÃ§Ãµes para ${name}:`, error))
        .finally(() => setLoading(false));
    }
  }, [fetchOptions]);

  return (
    <div>
      <label style={{ fontWeight: "bold", marginBottom: "5px", display: "block" }}>{label}<
/label>
      {loading ? (
        <Spin />
      ) : (
        <Select
          style={{ width: "100%" }}
          value={value}
          onChange={(val) => onChange(name, Number(val))} // ð\237\224¹ Converte o valor pa
ra nÃºmero ao selecionar
          placeholder={placeholder || "Selecione uma opÃ§Ã£o"}
          options={selectOptions}
          allowClear
        />
      )}
      {onCreateNew && (
        <Button type="dashed" style={{ marginTop: "10px" }} onClick={onCreateNew}>
          Cadastrar {label}
        </Button>
      )}
    </div>
  );
};

export default SelectField;
===== /home/alissu/Desktop/nansen_Web/frontend/src/components/ItemLoaging/ItemLoaging.tsx =
====
import React from "react";
import { Alert, Flex, Spin } from "antd";

const contentStyle: React.CSSProperties = {
  padding: 50,
  background: "rgba(0, 0, 0, 0.05)",
  borderRadius: 4,
};

const ItemLoading: React.FC = () => (
  <Flex
    gap="middle"
    vertical
    style={{
      display: "flex",
      width: "100%",
    }}
  >
    <Spin tip="Carregando dados do grÃ¡fico...">
      <Alert
        type="info"
        style={{
          minHeight: "350px",
        }}
      />
    </Spin>
  </Flex>
);

export default ItemLoading;
===== /home/alissu/Desktop/nansen_Web/frontend/src/components/Table/Table.tsx =====
import React from "react";
```

```
import { Table } from "antd";
import type { TableProps as AntTableProps, ColumnsType } from "antd/es/table";

interface TableProps<T> {
  columns: ColumnsType<T>;
  data: T[];
  loading?: boolean;
  rowKey?: string;
}

/**
 * Componente reutilizÃ¡vel de tabela
 * @param columns - DefiniÃ§Ã£o das colunas da tabela
 * @param data - Dados a serem exibidos na tabela
 * @param loading - Define se a tabela estÃ¡ carregando
 * @param rowKey - Chave Ãºnica de cada linha da tabela
 */
const CustomTable = <T extends object>({
  columns,
  data,
  loading = false,
  rowKey = "id",
}: TableProps<T>) => {
  return (
    <div className="table-container">
      <Table<T>
  columns={columns}
  dataSource={data}
  loading={loading}
  rowKey={rowKey}
  pagination={{ pageSize: 5 }}
  scroll={{ x: true }}
/>

    </div>
  );
};

export default CustomTable;
===== /home/alissu/Desktop/nansen_Web/frontend/src/components/Button/Button.tsx =====
import React from "react";
import { Button as AntButton } from "antd";

interface ButtonProps {
  type?: "primary" | "default" | "dashed" | "link" | "text";
  icon?: React.ReactNode;
  children: React.ReactNode;
  onClick?: () => void;
  className?: string;
  danger?: boolean;
}

const Button: React.FC<ButtonProps> = ({
  type = "default",
  icon,
  children,
  onClick,
  className,
  danger = false,
}) => {
  return (
    <AntButton
      type={type}
      icon={icon}
      onClick={onClick}
      className={`custom-btn ${className}`}
      danger={danger}
    >
      {children}
```

```
    </AntButton>
  );
};

export default Button;
===== /home/alissu/Desktop/nansen_Web/frontend/src/components/productionLinesTransfer/Produ
ctionLinesTransfer.tsx =====
import React from "react";
import { Transfer } from "antd";
import type { TransferDirection } from "antd/es/transfer";
import type { Key } from "react";
import { ProductionLine } from "../../types/ProductionLinesTypes"; // ð\237\224¹ Importando
 corretamente

interface TransferItem {
  key: string;
  title: string;
  value_mensuration_estimated: number;
}

interface ProductionLinesTransferProps {
  availableLines: ProductionLine[];
  selectedKeys: Key[];
  onChange: (keys: Key[], direction: TransferDirection, moveKeys: Key[]) => void;
}

const ProductionLinesTransfer: React.FC<ProductionLinesTransferProps> = ({
  availableLines,
  selectedKeys,
  onChange,
}) => {
  // ð\237\224¹ Transformando para o formato aceito pelo Transfer do Ant Design
  const dataSource: TransferItem[] = availableLines.map((line) => ({
    key: String(line.id),
    title: `${line.name} - ${line.value_mensuration_estimated} kWh`,
    value_mensuration_estimated: line.value_mensuration_estimated,
  }));

  // ð\237\224¹ Ajustando para o formato correto esperado pelo `onChange`
  const handleChange = (targetKeys: Key[], direction: TransferDirection, moveKeys: Key[]) =
> {
    onChange(targetKeys, direction, moveKeys);
  };

  return (
    <Transfer
      dataSource={dataSource}
      titles={["Linhas DisponÃ-veis", "Linhas Associadas"]}
      targetKeys={selectedKeys.map(String)} // â\234\205 Converte para string, jÃ¡ que Ant
Design usa `string[]`
      onChange={handleChange}
      render={(item) => item.title}
      showSearch
      rowKey={(item) => item.key}
      style={{ width: "100%" }}
    />
  );
};

export default ProductionLinesTransfer;
===== /home/alissu/Desktop/nansen_Web/frontend/src/components/form/FormField.tsx =====
import React, { useState } from "react";
import { Input, Select, Upload, Form, Button, InputNumber, Switch } from "antd";
import {
  UploadOutlined,
  ThunderboltOutlined,
  PoweroffOutlined,
  BulbOutlined,
} from "@ant-design/icons";
```

```
import type { FormField } from "./formTypes";
import { RcFile } from "antd/lib/upload";

interface FormFieldProps {
  field: FormField;
  value?: string | number | boolean | RcFile | undefined;
  onChange: (name: string, value: string | number | boolean | RcFile | null) => void;
}

const FormField: React.FC<FormFieldProps> = ({ field, value, onChange }) => {
  const [error, setError] = useState<string | null>(null);

  const validateNumber = (name: string, val: any) => {
    if (val === "" || val === null) {
      setError(null);
      onChange(name, val);
      return;
    }
    if (isNaN(val)) {
      setError("Deve ser um nÃºmero");
    } else {
      setError(null);
      onChange(name, val);
    }
  };

  const getIcon = () => {
    switch (field.name) {
      case "power":
        return <ThunderboltOutlined />;
      case "tension":
        return <PoweroffOutlined />;
      case "energy_consumption":
        return <BulbOutlined />;
      default:
        return null;
    }
  };

  const renderField = () => {
    switch (field.type) {
      case "input":
        return (
          <Input
            placeholder={field.placeholder || `Digite ${field.label.toLowerCase()}`}
            value={value as string}
            onChange={(e) => onChange(field.name, e.target.value)}
            style={{ width: "100%", height: "40px", padding: "4px 11px" }}
          />
        );

      case "readonly":
        return (
          <Input
            value={value as string}
            readOnly
            style={{
              width: "100%",
              padding: "8px",
              border: "1px solid #d9d9d9",
              borderRadius: "4px",
              backgroundColor: "#f5f5f5",
              color: "#000",
              cursor: "not-allowed",
            }}
          />
        );

      case "password":
```

```
  return (
    <Input.Password
      placeholder={field.placeholder || `Digite ${field.label.toLowerCase()}`}
      value={value as string}
      onChange={(e) => onChange(field.name, e.target.value)}
      style={{
        width: "100%",
        height: "40px",
        padding: "4px 11px",
        lineHeight: "normal",
        borderRadius: "6px",
      }}
    />
  );

case "number":
  return (
    <Form.Item validateStatus={error ? "error" : ""} help={error}>
      <InputNumber
        placeholder={field.placeholder || `Digite ${field.label.toLowerCase()}`}
        value={value as number}
        onChange={(val) => validateNumber(field.name, val)}
        style={{ width: "100%" }}
        addonBefore={getIcon()}
      />
    </Form.Item>
  );

case "textarea":
  return (
    <Input.TextArea
      placeholder={field.placeholder || `Digite ${field.label.toLowerCase()}`}
      value={value as string}
      onChange={(e) => onChange(field.name, e.target.value)}
      rows={4}
    />
  );

case "select":
  return (
    <Select
      placeholder={field.placeholder || "Selecione uma opÃ§Ã£o"}
      value={value as string | number | undefined}
      onChange={(val) => onChange(field.name, val)}
      style={{ width: "100%" }}
      disabled={field.disabled}
    >
      {field.options?.map((option) => (
        <Select.Option key={option.value} value={option.value}>
          {option.label}
        </Select.Option>
      ))}
    </Select>
  );

case "upload":
  return (
    <Upload
      beforeUpload={(file: RcFile) => {
        onChange(field.name, file);
        return false;
      }}
      showUploadList={true}
      maxCount={1}
      accept=".png, .jpg, .jpeg"
    >
      <Button type="primary" icon={<UploadOutlined />}>
        Upload da Imagem
      </Button>
```

```
          </Upload>
        );

      case "switch":
        return (
          <Switch
            checked={value as boolean}
            onChange={(val) => onChange(field.name, val)}
          />
        );

      default:
        return null;
    }
  };

  return (
    <Form.Item label={field.label} required={field.required}>
      {renderField()}
    </Form.Item>
  );
};

export default FormField;
===== /home/alissu/Desktop/nansen_Web/frontend/src/components/form/formTypes.ts =====
// Definindo tipos permitidos para o campo 'type' do FormField
export type FormFieldType =
  "text" |
  "input" |
  "select" |
  "number" |
  "upload" |
  "textarea" |
  "file" |
  "password" |
  "transfer" |
  "checkbox" |
  "switch" |
  "readonly"; // novo tipo para campos somente leitura

// A interface que descreve o campo do formulÃ¡rio
export interface FormField {
  name: string; // Nome do campo
  label: string; // RÃ³tulo do campo
  type: FormFieldType; // Tipo do campo
  options?: { value: string | number; label: string }[]; // OpÃ§Ãµes para campos do tipo 's
elect', 'transfer', etc
  fetchOptions?: () => Promise<{ value: string | number; label: string }[]>; // FunÃ§Ã£o as
sÃncrona para buscar as opÃ§Ãµes
  required?: boolean; // Se o campo Ã© obrigatÃ³rio
  placeholder?: string; // Texto de espaÃ§o reservado
  disabled?: boolean; // Se o campo estÃ¡ desabilitado
  onChange?: (value: any) => void; // FunÃ§Ã£o de retorno de mudanÃ§a (para atualizar o val
or do campo)
  value?: string | number | boolean | null; // O valor atual do campo, pode ser qualquer um
 dos tipos
}
// Interface para o componente de formulÃ¡rio dinÃ¢mico
export interface DynamicFormProps {
  fields: FormField[]; // Lista de campos do formulÃ¡rio
  initialValues?: Record<string, any>; // Valores iniciais para os campos do formulÃ¡rio
  onSubmit: (values: Record<string, any>) => void; // FunÃ§Ã£o de envio com os valores do f
ormulÃ¡rio
}
===== /home/alissu/Desktop/nansen_Web/frontend/src/components/form/useFormHandlers.ts =====
//to vendo se isso Ã© mesmo necessario
===== /home/alissu/Desktop/nansen_Web/frontend/src/components/form/DynamicForm.tsx =====
import React from "react";
import { Form, Input } from "antd";
```

```
import FormField from "./FormField";
import type { FormField as FormFieldType } from "./formTypes";
import Actions from "../actions/Actions";

interface DynamicFormProps {
  fields: FormFieldType[];
  values: Record<string, any>;
  loading?: boolean;
  onChange: (name: string, value: any) => void;
  onSubmit?: () => void; // Agora opcional
  onCancel?: () => void;
}

const DynamicForm: React.FC<DynamicFormProps> = ({
  fields,
  values,
  onChange,
  onSubmit,
  onCancel,
}) => {
  return (
    <Form layout="vertical" className="dynamic-form">
      {fields.map((field) => {
        if (field.type === "readonly") {
          return (
            <Form.Item key={field.name} label={field.label}>
              <Input
                value={values[field.name]}
                readOnly
                style={{
                  backgroundColor: "#f5f5f5",
                  color: "#000",
                  cursor: "not-allowed",
                }}
              />
            </Form.Item>
          );
        }

        return (
          <FormField
            key={field.name}
            field={field}
            value={values[field.name]}
            onChange={onChange}
          />
        );
      })}

      {(onSubmit || onCancel) && (
        <div className="form-actions">
          <Actions onSubmit={onSubmit} onCancel={onCancel} />
        </div>
      )}
    </Form>
  );
};

export default DynamicForm;
===== /home/alissu/Desktop/nansen_Web/frontend/src/store/subsectionStore.ts =====
import { create } from "zustand";
import { SubsectionItem } from "@/types/subsectionTypes";
import api from "@/services/api";

interface SubsectionState {
  subsections: SubsectionItem[];
  loading: boolean;
  fetchSubsections: () => Promise<void>;
  createSubsection: (data: Partial<SubsectionItem>) => Promise<void>;
```

```
  updateSubsection: (id: number, data: Partial<SubsectionItem>) => Promise<void>;
  deleteSubsection: (id: number) => Promise<void>;
}

export const useSubsectionStore = create<SubsectionState>((set) => ({
  subsections: [],
  loading: false,

  fetchSubsections: async () => {
    set({ loading: true });
    try {
      const response = await api.get<SubsectionItem[]>("/sub_sections/");
      set({ subsections: response.data });
    } catch (error) {
      console.error("Erro ao buscar subseÃ§Ãµes:", error);
    } finally {
      set({ loading: false });
    }
  },

  createSubsection: async (data) => {
    try {
      const response = await api.post("/sub_sections/", data);
      set((state) => ({
        subsections: [...state.subsections, response.data],
      }));
    } catch (error) {
      console.error("Erro ao criar subseÃ§Ã£o:", error);
    }
  },

  updateSubsection: async (id, data) => {
    try {
      await api.put(`/sub_sections/${id}/`, data);
      set((state) => ({
        subsections: state.subsections.map((s) =>
          s.id === id ? { ...s, ...data } : s
        ),
      }));
    } catch (error) {
      console.error("Erro ao atualizar subseÃ§Ã£o:", error);
    }
  },

  deleteSubsection: async (id) => {
    try {
      await api.delete(`/sub_sections/${id}/`);
      set((state) => ({
        subsections: state.subsections.filter((s) => s.id !== id),
      }));
    } catch (error) {
      console.error("Erro ao excluir subseÃ§Ã£o:", error);
    }
  },
}));

===== /home/alissu/Desktop/nansen_Web/frontend/src/store/sensorMonitoringStore.ts =====
// src/store/sensorMonitoringStore.ts
import { create } from "zustand";
import { MonitoringItem } from "@/types/monitoringTypes";

const mockData: MonitoringItem[] = [
  {
    id: 1,
    name: "Sensor A",
    description: "sensor A",
    estimated_consumption: 0,
    created_at: new Date().toISOString(),
  },
```

```
  {
    id: 2,
    name: "Sensor B",
    description: "sensor B",
    estimated_consumption: 0,
    created_at: new Date().toISOString(),
  },
];

type State = {
  sensorMonitorings: MonitoringItem[];
  fetchSensorMonitorings: () => Promise<void>;
  deleteSensorMonitoring: (id: number) => Promise<void>;
};

export const useSensorMonitoringStore = create<State>((set) => ({
  sensorMonitorings: [],
  fetchSensorMonitorings: async () => {
    await new Promise((r) => setTimeout(r, 200));
    set({ sensorMonitorings: mockData });
  },
  deleteSensorMonitoring: async (id) => {
    set((state) => ({
      sensorMonitorings: state.sensorMonitorings.filter((m) => m.id !== id),
    }));
  },
}));

===== /home/alissu/Desktop/nansen_Web/frontend/src/store/sectionStore.ts =====
import { create } from "zustand";
import { SectionItem } from "@/types/sections";
import api from "@/services/api";

interface SectionState {
  sections: SectionItem[];
  loading: boolean;
  fetchSections: () => Promise<void>;
  addSection: (data: Partial<SectionItem>) => Promise<void>;
  updateSection: (id: number, data: Partial<SectionItem>) => Promise<void>;
  deleteSection: (id: number) => Promise<void>;
}

// FunÃ§Ã£o auxiliar para tratar erros do Axios
function isAxiosError(error: unknown): error is { response: { data: any } } {
  return typeof error === "object" && error !== null && "response" in error;
}

export const useSectionStore = create<SectionState>((set, get) => ({
  sections: [],
  loading: false,

  // FunÃ§Ã£o para buscar todas as seÃ§Ãµes e construir a hierarquia
  fetchSections: async () => {
    set({ loading: true });

    try {
      const response = await api.get<SectionItem[]>("/sections/");
      const allSections = response.data;

      // CriaÃ§Ã£o do mapa de seÃ§Ãµes com `sections_filhas` vazias
      const sectionMap: Record<number, SectionItem> = {};
      allSections.forEach((section) => {
        sectionMap[section.id] = { ...section, sections_filhas: [] };
      });

      // ConstruÃ§Ã£o da hierarquia pai â\206\222 filhos
      const rootSections: SectionItem[] = [];
      allSections.forEach((section) => {
        if (section.secticon_parent) {
```

```
        const parent = sectionMap[section.secticon_parent];
        if (parent) {
          parent.sections_filhas?.push(sectionMap[section.id]);
        }
      } else {
        rootSections.push(sectionMap[section.id]);
      }
    });

    set({ sections: rootSections });
    console.log("â\234\205 SeÃ§Ãµes carregadas com hierarquia:", rootSections);
  } catch (error) {
    if (isAxiosError(error)) {
      console.error("â\235\214 Erro ao buscar seÃ§Ãµes:", error.response.data);
    } else {
      console.error("â\235\214 Erro ao buscar seÃ§Ãµes:", String(error));
    }
  } finally {
    set({ loading: false });
  }
},

// FunÃ§Ã£o para adicionar uma nova seÃ§Ã£o
addSection: async (data) => {
  try {
    if (!data.type_section || typeof data.type_section !== "number") {
      throw new Error("`type_section` deve ser um ID numÃ©rico.");
    }

    const response = await api.post("/sections/", data, {
      headers: {
        "Content-Type": "application/json",
      },
    });
    console.log("â\234\205 SeÃ§Ã£o criada:", response.data);
    await get().fetchSections();
  } catch (error) {
    if (isAxiosError(error)) {
      console.error("â\235\214 Erro ao adicionar seÃ§Ã£o:", error.response.data);
    } else {
      console.error("â\235\214 Erro ao adicionar seÃ§Ã£o:", String(error));
    }
  }
},

// FunÃ§Ã£o para atualizar uma seÃ§Ã£o existente
updateSection: async (id, data) => {
  try {
    // Garantindo que o campo description tenha um valor padrÃ£o
    if (typeof data.description !== "string") {
      data.description = data.description || "";
    }

    const response = await api.patch(`/sections/${id}/`, data, {
      headers: {
        "Content-Type": "application/json",
      },
    });
    console.log(`â\234\205 SeÃ§Ã£o ${id} atualizada com sucesso:`, response.data);
    await get().fetchSections();
  } catch (error) {
    if (isAxiosError(error)) {
      console.error("â\235\214 Erro ao atualizar seÃ§Ã£o:", error.response.data);
    } else {
      console.error("â\235\214 Erro ao atualizar seÃ§Ã£o:", String(error));
    }
  }
},
```

```
  // FunÃ§Ã£o para deletar uma seÃ§Ã£o
  deleteSection: async (id) => {
    try {
      await api.delete(`/sections/${id}/`);
      console.log(`ð\237\227\221ï¸\217 SeÃ§Ã£o ${id} excluÃ-da.`);
      await get().fetchSections();
    } catch (error) {
      if (isAxiosError(error)) {
        console.error("â\235\214 Erro ao excluir seÃ§Ã£o:", error.response.data);
      } else {
        console.error("â\235\214 Erro ao excluir seÃ§Ã£o:", String(error));
      }
    }
  },
}));
===== /home/alissu/Desktop/nansen_Web/frontend/src/store/iotDevices.ts =====
import { create } from "zustand";
import { message } from "antd";
import { IoTDevice } from "../types/IoTDevice";
import {
  getIoTDevices,
  getIoTDeviceById,
  createIoTDevice,
  updateIoTDevice,
  deleteIoTDevice,
} from "../services/IoTDevicesService";

interface IoTDevicesState {
  devices: IoTDevice[];
  loading: boolean;
  fetchDevices: () => Promise<void>;
  getDeviceById: (id: number) => Promise<IoTDevice | null>;
  addDevice: (device: Partial<IoTDevice>) => Promise<void>;
  editDevice: (id: number, device: Partial<IoTDevice>) => Promise<void>;
  removeDevice: (id: number) => Promise<void>;
}

export const useIoTDevicesStore = create<IoTDevicesState>((set) => ({
  devices: [],
  loading: false,

  fetchDevices: async () => {
    set({ loading: true });
    try {
      const data = await getIoTDevices();
      set({ devices: data });
    } catch (error) {
      console.error("Erro ao buscar dispositivos IoT:", error);
      message.error("Erro ao carregar dispositivos IoT!");
    } finally {
      set({ loading: false });
    }
  },

  getDeviceById: async (id: number) => {
    try {
      return await getIoTDeviceById(id);
    } catch (error) {
      console.error("Erro ao buscar dispositivo IoT:", error);
      message.error("Erro ao carregar o dispositivo!");
      return null;
    }
  },

  addDevice: async (device: Partial<IoTDevice>) => {
    try {
      await createIoTDevice(device);
      message.success("Dispositivo IoT cadastrado com sucesso!");
      await useIoTDevicesStore.getState().fetchDevices();
```

```
    } catch (error) {
      console.error("Erro ao cadastrar dispositivo:", error);
      message.error("Erro ao cadastrar dispositivo!");
    }
  },

  editDevice: async (id: number, device: Partial<IoTDevice>) => {
    try {
      await updateIoTDevice(id, device);
      message.success("Dispositivo IoT atualizado com sucesso!");
      await useIoTDevicesStore.getState().fetchDevices();
    } catch (error) {
      console.error("Erro ao atualizar dispositivo:", error);
      message.error("Erro ao atualizar dispositivo!");
    }
  },

  removeDevice: async (id: number) => {
    try {
      await deleteIoTDevice(id);
      message.success("Dispositivo IoT removido com sucesso!");
      set((state) => ({
        devices: state.devices.filter((device) => device.id !== id),
      }));
    } catch (error) {
      console.error("Erro ao excluir dispositivo:", error);
      message.error("Erro ao excluir dispositivo!");
    }
  },
}));
===== /home/alissu/Desktop/nansen_Web/frontend/src/store/useLojaStore.ts =====
import { create } from "zustand";
import { message } from "antd";
import {
  getStoreProducts,
  createStoreProduct,
  updateStoreProduct,
  deleteStoreProduct,
} from "../services/lojaService";
import { ProductLojaItem } from "../types/lojaTypes";

interface LojaState {
  products: ProductLojaItem[];
  fetchProducts: () => Promise<void>;
  fetchProductById: (id: number) => Promise<ProductLojaItem>;
  createProduct: (data: FormData) => Promise<void>;
  editProduct: (id: number, data: FormData) => Promise<void>;
  deleteProduct: (id: number) => Promise<void>;
}

export const useLojaStore = create<LojaState>((set, get) => ({
  products: [],

  fetchProducts: async () => {
    try {
      const products = await getStoreProducts();
      set({ products });
    } catch (error) {
      console.error(error);
      message.error("Erro ao carregar produtos!");
    }
  },

  fetchProductById: async (id) => {
    const product = get().products.find((p) => p.id === id);
    if (product) return product;
    throw new Error("Produto nÃ£o encontrado");
  },
```

```
  createProduct: async (data) => {
    try {
      const product = await createStoreProduct(data);
      set({ products: [...get().products, product] });
    } catch (error) {
      console.error(error);
      message.error("Erro ao criar produto!");
      throw error;
    }
  },

  editProduct: async (id, data) => {
    try {
      const updated = await updateStoreProduct(id, data);
      set({
        products: get().products.map((p) => (p.id === id ? updated : p)),
      });
    } catch (error) {
      console.error(error);
      message.error("Erro ao editar produto!");
      throw error;
    }
  },

  deleteProduct: async (id) => {
    try {
      await deleteStoreProduct(id);
      set({ products: get().products.filter((p) => p.id !== id) });
      message.success("Produto excluÃ-do!");
    } catch (error) {
      console.error(error);
      message.error("Erro ao excluir produto!");
      throw error;
    }
  },
}));

===== /home/alissu/Desktop/nansen_Web/frontend/src/store/missions.ts =====
// src/store/missions.ts
import { create } from "zustand";
import { MissionItem } from "../types/missions";
import {
  getMissions,
  createMission as svcCreateMission,
  updateMission as svcUpdateMission,
  deleteMission as svcDeleteMission,
  associateMissionToMonitoring as svcAssociateMission,
} from "../services/MissionService";

interface MissionStore {
  missions: MissionItem[];
  fetchMissions: () => Promise<void>;
  createMission: (missionData: Omit<MissionItem, "id">) => Promise<void>;
  updateMission: (
    id: number,
    missionData: Partial<Omit<MissionItem, "id">>
  ) => Promise<void>;
  deleteMission: (id: number) => Promise<void>;
  associateMissionToMonitoring: (
    missionId: number,
    monitoringId: number
  ) => Promise<void>;
}

export const useMissionStore = create<MissionStore>((set) => ({
  missions: [],

  fetchMissions: async () => {
    try {
```

```
      const data = await getMissions();
      set({ missions: data });
    } catch (error) {
      console.error("Erro ao buscar missÃµes", error);
    }
  },

  createMission: async (missionData) => {
    try {
      const newMission = await svcCreateMission(missionData);
      set((state) => ({
        missions: [...state.missions, newMission],
      }));
    } catch (error) {
      console.error("Erro ao criar missÃ£o", error);
    }
  },

  updateMission: async (id, missionData) => {
    try {
      const updated = await svcUpdateMission(id, missionData);
      set((state) => ({
        missions: state.missions.map((m) => (m.id === id ? updated : m)),
      }));
    } catch (error) {
      console.error("Erro ao atualizar missÃ£o", error);
    }
  },

  deleteMission: async (id) => {
    try {
      await svcDeleteMission(id);
      set((state) => ({
        missions: state.missions.filter((m) => m.id !== id),
      }));
    } catch (error) {
      console.error("Erro ao deletar missÃ£o", error);
    }
  },

  associateMissionToMonitoring: async (missionId, monitoringId) => {
    try {
      const updated = await svcAssociateMission(missionId, monitoringId);
      set((state) => ({
        missions: state.missions.map((m) => (m.id === missionId ? updated : m)),
      }));
    } catch (error) {
      console.error("Erro ao associar missÃ£o ao monitoramento", error);
    }
  },
}));

===== /home/alissu/Desktop/nansen_Web/frontend/src/store/ProductionLinesStore.ts =====
import { create } from "zustand";
import { getProductionLines, createProductionLine, updateProductionLine, deleteProductionLi
ne } from "../services/ProductionLinesService";
import { ProductionLine } from "../types/ProductionLinesTypes";

interface ProductionLinesState {
  productionLines: ProductionLine[];
  loading: boolean;
  fetchProductionLines: () => Promise<void>;
  createProductionLine: (data: Omit<ProductionLine, "id" | "created_at">) => Promise<void>;
  updateProductionLine: (id: number, data: Omit<ProductionLine, "id" | "created_at">) => Pr
omise<void>;
  removeProductionLine: (id: number) => Promise<void>;
}

export const useProductionLinesStore = create<ProductionLinesState>((set) => ({
```

```
  productionLines: [],
  loading: false,

  fetchProductionLines: async () => {
    set({ loading: true });
    try {
      const data = await getProductionLines();
      set({ productionLines: data });
    } catch (error) {
      console.error("Erro ao buscar linhas de produÃ§Ã£o:", error);
    } finally {
      set({ loading: false });
    }
  },

  createProductionLine: async (data) => {
    set({ loading: true });
    try {
      await createProductionLine(data);
      await useProductionLinesStore.getState().fetchProductionLines(); // ð\237\224¹ Atuali
za os dados
    } catch (error) {
      console.error("Erro ao criar linha de produÃ§Ã£o:", error);
    } finally {
      set({ loading: false });
    }
  },

  updateProductionLine: async (id, data) => {
    set({ loading: true });
    try {
      await updateProductionLine(id, data);
      await useProductionLinesStore.getState().fetchProductionLines();
    } catch (error) {
      console.error("Erro ao atualizar linha de produÃ§Ã£o:", error);
    } finally {
      set({ loading: false });
    }
  },

  removeProductionLine: async (id) => {
    set({ loading: true });
    try {
      await deleteProductionLine(id);
      await useProductionLinesStore.getState().fetchProductionLines();
    } catch (error) {
      console.error("Erro ao excluir linha de produÃ§Ã£o:", error);
    } finally {
      set({ loading: false });
    }
  },
}));
===== /home/alissu/Desktop/nansen_Web/frontend/src/store/monitoringStore.ts =====
// src/store/monitoringStore.ts
import { create } from "zustand";
import { MonitoringItem } from "@/types/monitoringTypes";
import api from "@/services/api";
import { fetchActiveMonitoringCount } from "@/services/monitoringService";

interface MonitoringState {
  monitorings: MonitoringItem[];
  loading: boolean;
  activeCount: number;
  fetchMonitorings: () => Promise<void>;
  fetchActiveCount: () => Promise<void>;
  createMonitoring: (data: Partial<MonitoringItem>) => Promise<void>;
  updateMonitoring: (
    id: number,
    data: Partial<MonitoringItem>
```

```
  ) => Promise<void>;
  deleteMonitoring: (id: number) => Promise<void>;
}

export const useMonitoringStore = create<MonitoringState>((set) => ({
  monitorings: [],
  loading: false,
  activeCount: 0,

  // Busca lista de monitoramentos
  fetchMonitorings: async () => {
    set({ loading: true });
    try {
      const response = await api.get<MonitoringItem[]>("/monitorings/");
      set({ monitorings: response.data });
    } catch (error) {
      console.error("Erro ao buscar monitoramentos:", error);
    } finally {
      set({ loading: false });
    }
  },

  // Busca a contagem de monitoramentos ativos
  fetchActiveCount: async () => {
    try {
      const count = await fetchActiveMonitoringCount();
      set({ activeCount: count });
    } catch (error) {
      console.error("Erro ao buscar contagem de monitoramentos ativos:", error);
    }
  },

  createMonitoring: async (data) => {
    try {
      const response = await api.post<MonitoringItem>("/monitorings/", data);
      set((state) => ({
        monitorings: [...state.monitorings, response.data],
      }));
    } catch (error) {
      console.error("Erro ao criar monitoramento:", error);
    }
  },

  updateMonitoring: async (id, data) => {
    try {
      const response = await api.put<MonitoringItem>(
        `/monitorings/${id}/`,
        data
      );
      set((state) => ({
        monitorings: state.monitorings.map((m) =>
          m.id === id ? response.data : m
        ),
      }));
    } catch (error) {
      console.error("Erro ao atualizar monitoramento:", error);
    }
  },

  deleteMonitoring: async (id) => {
    try {
      await api.delete(`/monitorings/${id}/`);
      set((state) => ({
        monitorings: state.monitorings.filter((m) => m.id !== id),
      }));
    } catch (error) {
      console.error("Erro ao excluir monitoramento:", error);
    }
  },
```

```
}));

===== /home/alissu/Desktop/nansen_Web/frontend/src/store/products.ts =====
import { create } from "zustand";
import { message } from "antd";
import { ProductItem } from "@/types/products";
import api from "@/services/api";

// DefiniÃ§Ã£o do estado do store
interface ProductsState {
  products: ProductItem[];
  loading: boolean;
  fetchProducts: () => Promise<void>;
  fetchProductById: (id: number) => Promise<ProductItem>; // ð\237\224¹ Busca produto por I
D
  createProduct: (data: FormData) => Promise<void>;
  updateProduct: (id: number, data: FormData) => Promise<void>; // ð\237\224¹ Atualiza prod
uto
  deleteProduct: (id: number) => Promise<void>;
}

export const useProductsStore = create<ProductsState>((set) => ({
  products: [],
  loading: false,

  // Buscar produtos da API
  fetchProducts: async () => {
    set({ loading: true });
    try {
      const response = await api.get<ProductItem[]>("/products/");
      set(() => ({ products: response.data }));
    } catch (error) {
      console.error("Erro ao buscar produtos:", error);
      message.error("Erro ao carregar os produtos!");
    } finally {
      set(() => ({ loading: false }));
    }
  },

  // ð\237\224¹ Buscar produto por ID para ediÃ§Ã£o
  fetchProductById: async (id: number) => {
    try {
      const response = await api.get<ProductItem>(`/products/${id}/`);
      return response.data;
    } catch (error) {
      console.error("Erro ao buscar produto:", error);
      message.error("Erro ao carregar os dados do produto!");
      throw error; // Retorna erro para ser tratado no componente
    }
  },

  // Criar produto
  createProduct: async (data: FormData) => {
    try {
      const response = await api.post("/products/", data, {
        headers: { "Content-Type": "multipart/form-data" },
      });


      set((state) => ({
        products: [...state.products, response.data],
      }));
    } catch (error) {
      console.error("Erro ao cadastrar produto:", error);
      message.error("Erro ao cadastrar produto!");
    }
  },

  // ð\237\224¹ Atualizar produto
```

```
  updateProduct: async (id: number, data: FormData) => {
    try {
      await api.put(`/products/${id}/`, data, {
        headers: { "Content-Type": "multipart/form-data" },
      });


      set((state) => ({
        products: state.products.map((product) =>
          product.id === id ? { ...product, ...Object.fromEntries(data) } : product
        ),
      }));
    } catch (error) {
      console.error("Erro ao atualizar produto:", error);
      message.error("Erro ao atualizar produto!");
    }
  },


  // Excluir produto
  deleteProduct: async (id: number) => {
    try {
      await api.delete(`/products/${id}/`);
      set((state) => ({
        products: state.products.filter((product) => product.id !== id),
      }));
      message.success("Produto excluÃ-do com sucesso!");
    } catch (error) {
      console.error("Erro ao excluir produto:", error);
      message.error("Erro ao excluir o produto!");
    }
  },
}));
===== /home/alissu/Desktop/nansen_Web/frontend/src/store/quizzes.ts =====
import { create } from "zustand";
import { QuizItem } from "../types/quizzes";
import { getQuizzes, createQuiz as createQuizAPI, updateQuiz as updateQuizAPI, deleteQuiz a
s deleteQuizAPI } from "../services/QuizService";

interface QuizStore {
  quizzes: QuizItem[];
  fetchQuizzes: () => Promise<void>;
  createQuiz: (quizData: Partial<QuizItem>) => Promise<void>;
  updateQuiz: (id: number, quizData: Partial<QuizItem>) => Promise<void>;
  deleteQuiz: (id: number) => Promise<void>;
}

export const useQuizStore = create<QuizStore>((set) => ({
  quizzes: [],

  fetchQuizzes: async () => {
    try {
      const data = await getQuizzes();
      set({ quizzes: data });
    } catch (error) {
      console.error("Erro ao buscar quizzes", error);
    }
  },

  createQuiz: async (quizData) => {
    try {
      const response = await createQuizAPI(quizData);
      set((state) => ({
        quizzes: [...state.quizzes, response.data],
      }));
    } catch (error) {
      console.error("Erro ao criar quizz", error);
    }
  },
```

```
  updateQuiz: async (id, quizData) => {
    try {
      const updatedQuiz = await updateQuizAPI(id, quizData);
      set((state) => ({
        quizzes: state.quizzes.map((quiz) => (quiz.id === id ? updatedQuiz : quiz)),
      }));
    } catch (error) {
      console.error("Erro ao atualizar quizz", error);
    }
  },

  deleteQuiz: async (id) => {
    try {
      await deleteQuizAPI(id);
      set((state) => ({
        quizzes: state.quizzes.filter((quiz) => quiz.id !== id),
      }));
    } catch (error) {
      console.error("Erro ao excluir quizz", error);
    }
  },
}));
===== /home/alissu/Desktop/nansen_Web/frontend/src/store/users.ts =====
import { create } from "zustand";
import * as UsersService from "../services/UsersService";
import { UserItem, UserRegister } from "../types/users";

interface UsersStore {
  users: UserItem[];
  fetchUsers: () => Promise<void>;
  addUser: (data: UserRegister) => Promise<void>;
  updateUser: (id: number, data: Partial<UserItem>) => Promise<void>;
  deleteUser: (id: number) => Promise<void>;
}

export const useUsersStore = create<UsersStore>((set) => ({
  users: [],

  fetchUsers: async () => {
    try {
      const users = await UsersService.getUsers();
      set({ users });
    } catch (err) {
      console.error("Erro ao buscar usuÃ¡rios:", err);
    }
  },

  addUser: async (data) => {
    try {
      const newUser = await UsersService.createUser(data);
      set((state) => ({ users: [...state.users, newUser] }));
    } catch (err) {
      console.error("Erro ao cadastrar usuÃ¡rio:", err);
    }
  },

  updateUser: async (id, data) => {
    try {
      const updated = await UsersService.updateUser(id, data);
      set((state) => ({
        users: state.users.map((u) => (u.id === id ? updated : u)),
      }));
    } catch (err) {
      console.error("Erro ao atualizar usuÃ¡rio:", err);
    }
  },

  deleteUser: async (id) => {
```

```
    try {
      await UsersService.deleteUser(id);
      set((state) => ({
        users: state.users.filter((u) => u.id !== id),
      }));
    } catch (err) {
      console.error("Erro ao excluir usuÃ¡rio:", err);
    }
  },
}));

===== /home/alissu/Desktop/nansen_Web/frontend/src/store/typeSectionStore.ts =====
import { create } from "zustand";
import api from "@/services/api";

export interface TypeSection {
  id: number;
  name: "SETOR" | "LINHA" | "EQUIPAMENTO";
}

interface TypeSectionState {
  types: TypeSection[];
  loading: boolean;
  fetchTypes: () => Promise<void>;
  getTypeIdByName: (name: TypeSection["name"]) => number | null;
}

export const useTypeSectionStore = create<TypeSectionState>((set, get) => ({
  types: [],
  loading: false,

  fetchTypes: async () => {
    set({ loading: true });
    try {
      const response = await api.get<TypeSection[]>("/type_sections/"); // â\234\205 URL co
rrigida
      set({ types: response.data });
      console.log("ð\237\223¦ Tipos de seÃ§Ã£o carregados:", response.data);
    } catch (error: any) {
      console.error("â\235\214 Erro ao buscar tipos de seÃ§Ã£o:", error?.response?.data ||
error);
    } finally {
      set({ loading: false });
    }
  },

  getTypeIdByName: (name) => {
    const { types } = get();
    const match = types.find((t) => t.name === name);
    return match ? match.id : null;
  },
}));
===== /home/alissu/Desktop/nansen_Web/frontend/src/store/index.ts =====
export { useProductsStore } from "./products";
// Caso tenha outros stores no futuro, basta adicionar aqui
// export { useAuthStore } from "./auth";
// export { useUserStore } from "./user";
===== /home/alissu/Desktop/nansen_Web/frontend/src/store/sectors.ts =====
import { create } from "zustand";
import { message } from "antd";
import { getSectors, createSector, updateSector, deleteSector } from "../services/SectorsSe
rvice";
import { getProductionLines } from "../services/ProductionLinesService"; // â\234\205 Corri
gido import
import { Sector } from "../types/sectors";
import { ProductionLine } from "../types/ProductionLinesTypes"; // â\234\205 Corrigido impo
rt

interface SectorsState {
```

```
  sectors: Sector[];
  loading: boolean;
  fetchSectors: () => Promise<void>;
  createSector: (data: Omit<Sector, "id" | "created_at">) => Promise<Sector | null>;
  updateSector: (id: number, data: Omit<Sector, "id" | "created_at">) => Promise<boolean>;
  deleteSector: (id: number) => Promise<boolean>;
  fetchProductionLines: () => Promise<void>; // â\234\205 Adicionado para buscar linhas dis
ponÃ-veis
  availableProductionLines: ProductionLine[]; // â\234\205 Adicionado para armazenar linhas
 disponÃ-veis
}

export const useSectorsStore = create<SectorsState>((set) => ({
  sectors: [],
  loading: false,
  availableProductionLines: [], // â\234\205 Inicializa a lista de linhas disponÃ-veis

  // Buscar setores da API
  fetchSectors: async () => {
    set({ loading: true });
    try {
      const response = await getSectors();
      set({ sectors: response || [] });
    } catch (error) {
      console.error("Erro ao buscar setores:", error);
      message.error("Erro ao carregar os setores!");
    } finally {
      set({ loading: false });
    }
  },

  // Criar setor
  createSector: async (data) => {
    try {
      const newSector = await createSector(data);
      if (!newSector?.id) throw new Error("ID do setor nÃ£o foi retornado pela API.");

      set((state) => ({
        sectors: [...state.sectors, newSector],
      }));

      // â\234\205 Atualiza a lista de linhas disponÃ-veis apÃ³s a criaÃ§Ã£o do setor
      await useSectorsStore.getState().fetchProductionLines();

      message.success("Setor cadastrado com sucesso!");
      return newSector;
    } catch (error) {
      console.error("Erro ao cadastrar setor:", error);
      message.error("Erro ao cadastrar setor!");
      return null;
    }
  },

  // Atualizar setor
  updateSector: async (id, data) => {
    if (!id) {
      message.error("ID invÃ¡lido para atualizaÃ§Ã£o!");
      return false;
    }

    try {
      await updateSector(id, data);
      set((state) => ({
        sectors: state.sectors.map((sector) =>
          sector.id === id ? { ...sector, ...data } : sector
        ),
      }));
      message.success("Setor atualizado com sucesso!");
      return true;
```

```
    } catch (error) {
      console.error("Erro ao atualizar setor:", error);
      message.error("Erro ao atualizar setor!");
      return false;
    }
  },

  // Buscar linhas de produÃ§Ã£o disponÃveis (nÃ£o associadas a setores)
  fetchProductionLines: async () => {
    set({ loading: true });
    try {
      const data = await getProductionLines();
      const filteredLines = data.filter((line: ProductionLine) => !line.setor); // â\234
\205 Filtra apenas linhas sem setor
      set({ availableProductionLines: filteredLines });
    } catch (error) {
      console.error("Erro ao buscar linhas de produÃ§Ã£o:", error);
    } finally {
      set({ loading: false });
    }
  },

  // Excluir setor
  deleteSector: async (id) => {
    if (!id) {
      message.error("ID invÃ¡lido para exclusÃ£o!");
      return false;
    }

    try {
      await deleteSector(id);
      set((state) => ({
        sectors: state.sectors.filter((sector) => sector.id !== id),
      }));

      // â\234\205 Atualiza a lista de linhas disponÃveis apÃ³s exclusÃ£o do setor
      await useSectorsStore.getState().fetchProductionLines();

      message.success("Setor excluÃdo com sucesso!");
      return true;
    } catch (error) {
      console.error("Erro ao excluir setor:", error);
      message.error("Erro ao excluir setor!");
      return false;
    }
  },
}));
===== /home/alissu/Desktop/nansen_Web/frontend/src/store/equipaments.ts =====
import { create } from "zustand";
import { message } from "antd";
import { EquipamentItem } from "@/types/equipaments";
import api from "@/services/api";

interface EquipamentsState {
  equipaments: EquipamentItem[];
  loading: boolean;
  fetchEquipaments: () => Promise<void>;
  fetchEquipamentById: (id: number) => Promise<EquipamentItem>;
  createEquipament: (data: any) => Promise<void>;
  updateEquipament: (id: number, data: any) => Promise<void>;
  deleteEquipament: (id: number) => Promise<void>;
}

export const useEquipamentsStore = create<EquipamentsState>((set) => ({
  equipaments: [],
  loading: false,

  fetchEquipaments: async () => {
    set({ loading: true });
```

```
    try {
      const response = await api.get<EquipamentItem[]>("/equipaments/");
      set(() => ({ equipaments: response.data }));
    } catch (error) {
      console.error("Erro ao buscar equipamentos:", error);
      message.error("Erro ao carregar os equipamentos!");
    } finally {
      set(() => ({ loading: false }));
    }
  },

  fetchEquipamentById: async (id: number) => {
    try {
      const response = await api.get<EquipamentItem>(`/equipaments/${id}/`);
      return response.data;
    } catch (error) {
      console.error("Erro ao buscar equipamento:", error);
      message.error("Erro ao carregar os dados do equipamento!");
      throw error;
    }
  },

  createEquipament: async (data: any) => {
    try {
      const response = await api.post("/equipaments/", data);
      set((state) => ({
        equipaments: [...state.equipaments, response.data],
      }));
    } catch (error) {
      console.error("Erro ao cadastrar equipamento:", error);
      message.error("Erro ao cadastrar equipamento!");
    }
  },

  updateEquipament: async (id: number, data: any) => {
    try {
      await api.put(`/equipaments/${id}/`, data);
      set((state) => ({
        equipaments: state.equipaments.map((equipament) =>
          equipament.id === id ? { ...equipament, ...data } : equipament
        ),
      }));
      message.success("Equipamento atualizado com sucesso!");
    } catch (error) {
      console.error("Erro ao atualizar equipamento:", error);
      message.error("Erro ao atualizar equipamento!");
    }
  },

  deleteEquipament: async (id: number) => {
    try {
      await api.delete(`/equipaments/${id}/`);
      set((state) => ({
        equipaments: state.equipaments.filter((equipament) => equipament.id !== id),
      }));
      message.success("Equipamento excluÃ-do com sucesso!");
    } catch (error) {
      console.error("Erro ao excluir equipamento:", error);
      message.error("Erro ao excluir equipamento!");
    }
  },
}));
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/equipments/Equipments.tsx =====
// src/pages/equipments/Equipments.tsx
import React from "react";
import { PlusOutlined, FilterOutlined } from "@ant-design/icons";
import { useNavigate } from "react-router-dom";
import ItemSideBar from "../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../layout/Header/ItemHeader";
```

```
import Button from "../../components/Button/Button";
import CustomTable from "../../components/Table/Table";
import ItemHeaderCabecalho from "../../layout/Header/components/ItemHeaderCabecalho";
import { useEquipamentsTable } from "./hooks/useEquipamentsTable";

const Equipments: React.FC = () => {
  const navigate = useNavigate();
  const { columns, equipaments, loading } = useEquipamentsTable();

  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          {/* CabeÃ§alho da pÃ¡gina */}
          <ItemHeaderCabecalho
            title="Equipamentos"
            subTitle="Lista de equipamentos jÃ¡ cadastrados"
          />

          {/* BotÃµes de aÃ§Ã£o */}
          <section className="actions-section">
            <Button
              type="primary"
              className="primary-btn"
              icon={<PlusOutlined />}
              onClick={() => navigate("/equipments/register")}
            >
              Cadastrar equipamento
            </Button>
            <Button type="link" className="filter-btn" icon={<FilterOutlined />}>
              Filtros
            </Button>
          </section>

          {/* Tabela utilizando o hook de Equipamentos */}
          <section className="table-container">
            <CustomTable columns={columns} data={equipaments} loading={loading} />
          </section>
        </main>
      </div>
    </div>
  );
};

export default Equipments;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/equipments/components/EditEquipmen
t.tsx =====
import React, { useEffect, useState } from "react";
import { useNavigate, useParams } from "react-router-dom";
import { message } from "antd";
import ItemSideBar from "../../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../../layout/Header/components/ItemHeaderCabecalho";
import DynamicForm from "../../../components/form/DynamicForm";
import { useEquipamentsStore } from "../../../store/equipaments";
import { getProductionLines } from "../../../services/ProductionLinesService";

const EditEquipment: React.FC = () => {
  const navigate = useNavigate();
  const { id } = useParams<{ id: string }>();
  const { fetchEquipamentById, updateEquipament } = useEquipamentsStore();
  const [loading, setLoading] = useState(false);
  const [loadingOptions, setLoadingOptions] = useState(true);
  const [formValues, setFormValues] = useState<{
    name: string;
    description: string;
    power?: number;
```

```
    tension?: number;
    energy_consumption?: number;
    max_consumption?: number;
    min_consumption?: number;
    production_line?: number;
  }>({
    name: "",
    description: "",
    power: undefined,
    tension: undefined,
    energy_consumption: undefined,
    max_consumption: undefined,
    min_consumption: undefined,
    production_line: undefined,
  });

  const [productionLinesOptions, setProductionLinesOptions] = useState<{ value: number; lab
el: string }[]>([]);

  useEffect(() => {
    const fetchData = async () => {
      if (!id || isNaN(Number(id))) {
        message.error("ID invÃ¡lido para ediÃ§Ã£o.");
        return;
      }

      setLoading(true);
      try {
        const data = await fetchEquipamentById(Number(id));
        setFormValues({
          name: data.name || "",
          description: data.description || "",
          power: data.power ?? undefined,
          tension: data.tension ?? undefined,
          energy_consumption: data.energy_consumption ?? undefined,
          max_consumption: data.max_consumption ?? undefined,
          min_consumption: data.min_consumption ?? undefined,
          production_line: data.production_line ?? undefined,
        });

        const productionLinesData = await getProductionLines();
        setProductionLinesOptions(
          productionLinesData.map((line) => ({ value: line.id, label: line.name }))
        );
      } catch (error) {
        message.error("Erro ao carregar os dados do equipamento.");
        console.error("Erro ao buscar equipamento:", error);
      } finally {
        setLoading(false);
        setLoadingOptions(false);
      }
    };

    fetchData();
  }, [id]);

  const handleChange = (name: string, value: any) => {
    setFormValues((prev) => ({ ...prev, [name]: value }));
  };

  const handleSubmit = async () => {
    if (!formValues.name.trim()) {
      message.error("O nome do equipamento Ã© obrigatÃ³rio!");
      return;
    }

    setLoading(true);
    try {
      await updateEquipament(Number(id), {
```

```
        name: formValues.name,
        description: formValues.description,
        power: formValues.power !== undefined ? Number(formValues.power) : null,
        tension: formValues.tension !== undefined ? Number(formValues.tension) : null,
        energy_consumption: formValues.energy_consumption !== undefined ? Number(formValues
.energy_consumption) : null,
        max_consumption: formValues.max_consumption !== undefined ? Number(formValues.max_c
onsumption) : null,
        min_consumption: formValues.min_consumption !== undefined ? Number(formValues.min_c
onsumption) : null,
        production_line: formValues.production_line !== undefined ? Number(formValues.produ
ction_line) : null,
      });

      message.success("Equipamento atualizado com sucesso!");
      navigate("/equipments");
    } catch (error) {
      message.error("Erro ao atualizar equipamento.");
      console.error("Erro ao atualizar equipamento:", error);
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          <ItemHeaderCabecalho title="Editar Equipamento" subTitle="Atualize os dados do eq
uipamento" />
          <DynamicForm
            fields={[
              { name: "name", label: "Nome", type: "input", required: true },
              { name: "description", label: "DescriÃ§Ã£o", type: "textarea", required: true
 },
              { name: "power", label: "PotÃªncia (W)", type: "number" },
              { name: "tension", label: "TensÃ£o (V)", type: "number" },
              { name: "energy_consumption", label: "Consumo de Energia (kWh)", type: "numbe
r" },
              { name: "max_consumption", label: "Consumo MÃ¡ximo (kWh)", type: "number" },
              { name: "min_consumption", label: "Consumo MÃnimo (kWh)", type: "number" },
              {
                name: "production_line",
                label: "Linha de ProduÃ§Ã£o",
                type: "select",
                options: productionLinesOptions,
                disabled: loadingOptions,
              },
            ]}
            values={formValues}
            onChange={handleChange}
            onSubmit={handleSubmit}
            loading={loading}
            onCancel={() => navigate("/equipments")}
          />
        </main>
      </div>
    </div>
  );
};

export default EditEquipment;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/equipments/hooks/useEquipamentsTab
le.tsx =====
// src/pages/equipments/hooks/useEquipamentsTable.tsx
import { useEffect, useState } from "react";
import { message } from "antd";
```

```
import { useNavigate } from "react-router-dom";
import { SortOrder } from "antd/es/table/interface";
import { useEquipamentsStore } from "../../../store/equipaments";
import Actions from "../../../components/actions/Actions";
import { EquipamentItem } from "../../../types/equipaments";

export const useEquipamentsTable = () => {
  const navigate = useNavigate();
  const { equipaments, fetchEquipaments, deleteEquipament } = useEquipamentsStore();
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchEquipamentsFromAPI = async () => {
      try {
        setLoading(true);
        await fetchEquipaments();
      } catch (error) {
        console.error("Erro ao buscar equipamentos:", error);
        message.error("Erro ao carregar os equipamentos!");
      } finally {
        setLoading(false);
      }
    };

    fetchEquipamentsFromAPI();
  }, []);

  // DefiniÃ§Ã£o das colunas da tabela
  const columns = [
    {
      title: "Nome",
      dataIndex: "name",
      key: "name",
      sorter: (a: EquipamentItem, b: EquipamentItem) => a.name.localeCompare(b.name),
      sortDirections: ["ascend", "descend"] as SortOrder[],
      render: (text: string | undefined) => <strong>{text ?? "Sem nome"}</strong>,
    },
    {
      title: "DescriÃ§Ã£o",
      dataIndex: "description",
      key: "description",
      render: (text: string | undefined) => <span>{text ?? "Sem descriÃ§Ã£o"}</span>,
    },
    {
      title: "PotÃªncia (W)",
      dataIndex: "power",
      key: "power",
      render: (text: number | undefined) => <span>{text ?? "NÃ£o informado"}</span>,
    },
    {
      title: "TensÃ£o (V)",
      dataIndex: "tension",
      key: "tension",
      render: (text: number | undefined) => <span>{text ?? "NÃ£o informado"}</span>,
    },
    {
      title: "Consumo de Energia (kWh)",
      dataIndex: "energy_consumption",
      key: "energy_consumption",
      render: (text: number | undefined) => <span>{text ?? "NÃ£o informado"}</span>,
    },
    {
      title: "Setor",
      dataIndex: "setor",
      key: "setor",
      render: (text: any) => <span>{text ?? "NÃ£o informado"}</span>,
    },
    {
      title: "SeÃ§Ã£o",
```

```
      dataIndex: "section",
      key: "section",
      render: (text: any) => <span>{text ?? "NÃ£o informado"}</span>,
    },
    {
      title: "AÃ§Ãµes",
      key: "actions",
      render: (_: any, record: EquipamentItem) => (
        <Actions
          onEdit={() => navigate(`/equipments/edit/${record.id}`)}
          onDelete={async () => {
            if (record.id) {
              await deleteEquipament(Number(record.id));
            }
          }}
        />
      ),
    },
  ];

  return { columns, equipaments, loading };
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/equipments/equipmentsregister/Regi
ster.tsx =====
import React, { useState, useEffect } from "react";
import { message } from "antd";
import { useNavigate } from "react-router-dom";
import ItemSideBar from "../../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../../layout/Header/components/ItemHeaderCabecalho";
import DynamicForm from "../../../components/form/DynamicForm";
import { useEquipamentsStore } from "../../../store/equipaments";
import { getProductionLines } from "../../../services/ProductionLinesService"; // ð\237\224
¹ Nova API para buscar linhas de produÃ§Ã£o

const Register: React.FC = () => {
  const navigate = useNavigate();
  const { createEquipament } = useEquipamentsStore();
  const [loading, setLoading] = useState(false);
  const [loadingOptions, setLoadingOptions] = useState(true);
  const [formValues, setFormValues] = useState({
    name: "",
    description: "",
    power: "",
    tension: "",
    energy_consumption: "",
    max_consumption: "",
    min_consumption: "",
    production_line: null,
  });

  const [productionLinesOptions, setProductionLinesOptions] = useState<{ value: number; lab
el: string }[]>([]);

  useEffect(() => {
    const fetchOptions = async () => {
      try {
        const productionLinesData = await getProductionLines();
        setProductionLinesOptions(
          productionLinesData.map((line) => ({ value: line.id, label: line.name }))
        );
      } catch (error) {
        message.error("Erro ao carregar linhas de produÃ§Ã£o!");
        console.error(error);
      } finally {
        setLoadingOptions(false);
      }
    };
```

```
    fetchOptions();
  }, []);

  const handleChange = (name: string, value: any) => {
    setFormValues((prev) => ({ ...prev, [name]: value }));
  };

  const handleSubmit = async () => {
    if (!formValues.name.trim()) {
      message.error("O nome do equipamento Ã© obrigatÃ³rio!");
      return;
    }

    if (!formValues.description.trim()) {
      message.error("A descriÃ§Ã£o do equipamento Ã© obrigatÃ³ria!");
      return;
    }

    if (loading) return;

    setLoading(true);
    try {
      await createEquipament({
        ...formValues,
        power: Number(formValues.power) || null,
        tension: Number(formValues.tension) || null,
        energy_consumption: Number(formValues.energy_consumption) || null,
        max_consumption: Number(formValues.max_consumption) || null,
        min_consumption: Number(formValues.min_consumption) || null,
        production_line: formValues.production_line ? Number(formValues.production_line) :
null,
      });

      message.success("Equipamento cadastrado com sucesso!");
      navigate("/equipments");
    } catch (error) {
      message.error("Erro ao cadastrar equipamento!");
      console.error(error);
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          <ItemHeaderCabecalho
            title="Cadastro de Equipamentos"
            subTitle="Preencha os campos abaixo para cadastrar um equipamento"
          />
          <DynamicForm
            fields={[
              { name: "name", label: "Nome", type: "input", required: true },
              { name: "description", label: "DescriÃ§Ã£o", type: "textarea", required: true
 },
              { name: "power", label: "PotÃªncia (W)", type: "number" },
              { name: "tension", label: "TensÃ£o (V)", type: "number" },
              { name: "energy_consumption", label: "Consumo de Energia (kWh)", type: "numbe
r" },
              { name: "max_consumption", label: "Consumo MÃ¡ximo (kWh)", type: "number" },
              { name: "min_consumption", label: "Consumo MÃnimo (kWh)", type: "number" },
              {
                name: "production_line",
                label: "Linha de ProduÃ§Ã£o",
                type: "select",
                options: productionLinesOptions,
```

```
                disabled: loadingOptions,
              },
            ]}
            values={formValues}
            onChange={handleChange}
            onSubmit={handleSubmit}
            loading={loading}
            onCancel={() => navigate("/equipments")}
          />
        </main>
      </div>
    </div>
  );
};

export default Register;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/users/Users.tsx =====
import React from "react";
import { PlusOutlined, FilterOutlined } from "@ant-design/icons";
import { useNavigate } from "react-router-dom";
import ItemSideBar from "../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../layout/Header/ItemHeader";
import Button from "../../components/Button/Button";

import ItemHeaderCabecalho from "../../layout/Header/components/ItemHeaderCabecalho";
import UsersList from "./components/UsersList";

const Users: React.FC = () => {
  const navigate = useNavigate();

  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          {/* CabeÃ§alho da pÃ¡gina */}
          <ItemHeaderCabecalho
            title="UsuÃ¡rios"
            subTitle="Lista de usuÃ¡rios cadastrados"
          />

          {/* BotÃµes de aÃ§Ã£o */}
          <section className="actions-section">
            <Button
              type="primary"
              className="primary-btn"
              icon={<PlusOutlined />}
              onClick={() => navigate("/users/register")}
            >
              Cadastrar UsuÃ¡rio
            </Button>
            <Button type="link" className="filter-btn" icon={<FilterOutlined />}>
              Filtros
            </Button>
          </section>

          {/* Tabela de usuÃ¡rios */}
          <section className="table-container">
            <UsersList />
          </section>
        </main>
      </div>
    </div>
  );
};

export default Users;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/users/components/UsersList.tsx ===
```

```
==
// src/pages/users/components/UsersList.tsx
import React, { useEffect, useState } from "react";
import { Table, Input, Button, Avatar, Popconfirm, message, Tag } from "antd";
import { EditOutlined, DeleteOutlined } from "@ant-design/icons";
import { useUsers } from "../hooks/useUsers";
import { useUserDetails } from "../../../hooks/useUserDetails";
import { useNavigate } from "react-router-dom";

const { Search } = Input;

const UsersList: React.FC = () => {
  const { users, fetchUsers, deleteUser } = useUsers();
  const { cache, load } = useUserDetails();
  const [filtered, setFiltered] = useState(users);
  const navigate = useNavigate();

  useEffect(() => {
    fetchUsers();
  }, [fetchUsers]);
  useEffect(() => {
    setFiltered(users);
  }, [users]);

  const onSearch = (q: string) => {
    const low = q.toLowerCase();
    setFiltered(
      users.filter((u) =>
        [u.username, u.name || "", u.email, u.role].some((f) =>
          f.toLowerCase().includes(low)
        )
      )
    );
  };

  const handleDelete = async (id: number) => {
    await deleteUser(id);
    message.success("UsuÃ¡rio excluÃ-do!");
  };

  const columns = [
    {
      title: "Avatar",
      dataIndex: "avatar_url",
      key: "avatar",
      render: (url: string) => <Avatar src={url} />,
    },
    { title: "Username", dataIndex: "username", key: "username" },
    {
      title: "Nome",
      dataIndex: "name",
      key: "name",
      render: (name: string, rec: any) => name || rec.username,
    },
    { title: "Email", dataIndex: "email", key: "email" },
    { title: "PermissÃ£o", dataIndex: "role", key: "role" },

    {
      title: "Conquistas",
      key: "achievements",
      render: (_: any, rec: any) => {
        const d = cache[rec.id];
        if (d) return d.achievements.length;
        return <a onClick={() => load(rec.id)}>Carregar</a>;
      },
    },
    {
      title: "Recompensas",
      key: "rewards",
```

```
    render: (_: any, rec: any) => {
      const d = cache[rec.id];
      if (d) return d.rewards.length;
      return <a onClick={() => load(rec.id)}>Carregar</a>;
    },
  },

  {
    title: "AÃ§Ãµes",
    key: "actions",
    render: (_: any, rec: any) => (
      <>
        <Button
          icon={<EditOutlined />}
          onClick={() => navigate(`/users/edit/${rec.id}`)}
          style={{ marginRight: 8 }}
        />
        <Popconfirm
          title="Confirma exclusÃ£o?"
          onConfirm={() => handleDelete(rec.id)}
        >
          <Button danger icon={<DeleteOutlined />} />
        </Popconfirm>
      </>
    ),
  },
];

return (
  <>
    <Search
      placeholder="Buscar usuÃ¡rio..."
      onSearch={onSearch}
      enterButton
      style={{ marginBottom: 16, width: 300 }}
    />

    <Table
      dataSource={filtered}
      columns={columns}
      rowKey="id"
      pagination={{ pageSize: 5 }}
      expandable={{
        expandedRowRender: (rec) => {
          const d = cache[rec.id];
          if (!d) return <p>Carregandoâ\200¦</p>;
          return (
            <div style={{ padding: 16 }}>
              <h4>Conquistas</h4>
              {d.achievements.map((a) => (
                <Tag key={a.id}>
                  {a.description} (+{a.nansen_coins} coins, +{a.quantity_xp}{" "}
                  XP)
                </Tag>
              ))}
              <h4 style={{ marginTop: 16 }}>Recompensas</h4>
              {d.rewards.map((r) => (
                <Tag key={r.id}>
                  {r.description} ({r.points} pts)
                </Tag>
              ))}
            </div>
          );
        },
        onExpand: (_expanded, rec) => load(rec.id),
      }}
    />
  </>
);
```

```
};

export default UsersList;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/users/components/UsersEdit.tsx ===
==
import React, { useEffect, useState } from "react";
import {
  Form,
  Input,
  Select,
  Switch,
  Button,
  Card,
  Row,
  Col,
  message,
} from "antd";
import { useNavigate, useParams } from "react-router-dom";
import dayjs, { Dayjs } from "dayjs";
import { useUsersStore } from "../../../store/users";
import ItemSideBar from "../../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../../layout/Header/components/ItemHeaderCabecalho";

const { Option } = Select;

const UsersEdit: React.FC = () => {
  const [form] = Form.useForm();
  const navigate = useNavigate();
  const { id } = useParams<{ id: string }>();
  const { users, fetchUsers, updateUser } = useUsersStore();
  const [loading, setLoading] = useState(false);

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

  useEffect(() => {
    const uid = Number(id);
    const u = users.find((x) => x.id === uid);
    if (u) {
      form.setFieldsValue({
        username: u.username,
        name: u.name,
        email: u.email,
        role: u.role,
      });
    }
  }, [users]);

  const onFinish = async (vals: any) => {
    setLoading(true);
    try {
      await updateUser(Number(id), {
        name: vals.name,
        role: vals.role,
      });
      message.success("UsuÃ¡rio atualizado!");
      navigate("/users");
    } catch {
      message.error("Erro ao atualizar!");
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="layout-container">
```

```
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          <ItemHeaderCabecalho
            title="Editar UsuÃ¡rio"
            subTitle="Atualize os dados"
          />
          <Card>
            <Form form={form} layout="vertical" onFinish={onFinish}>
              <Row gutter={16}>
                <Col span={12}>
                  <Form.Item label="Username" name="username">
                    <Input disabled />
                  </Form.Item>
                </Col>
                <Col span={12}>
                  <Form.Item
                    name="name"
                    label="Nome"
                    rules={[{ required: true, message: "Informe o nome" }]}
                  >
                    <Input />
                  </Form.Item>
                </Col>
              </Row>
              <Row gutter={16}>
                <Col span={12}>
                  <Form.Item label="Email" name="email">
                    <Input disabled />
                  </Form.Item>
                </Col>
                <Col span={12}>
                  <Form.Item
                    name="role"
                    label="PermissÃ£o"
                    rules={[
                      { required: true, message: "Selecione a permissÃ£o" },
                    ]}
                  >
                    <Select>
                      <Option value="ADMIN">Admin</Option>
                      <Option value="LIDER">LÃ-der</Option>
                      <Option value="GAME">Game</Option>
                    </Select>
                  </Form.Item>
                </Col>
              </Row>
              <Form.Item style={{ textAlign: "right", marginTop: 16 }}>
                <Button
                  onClick={() => navigate("/users")}
                  style={{ marginRight: 8 }}
                >
                  Cancelar
                </Button>
                <Button type="primary" htmlType="submit" loading={loading}>
                  Salvar
                </Button>
              </Form.Item>
            </Form>
          </Card>
        </main>
      </div>
    </div>
  );
};

export default UsersEdit;
```

```
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/users/components/UsersRegister.tsx
 =====
import React, { useEffect, useState } from "react";
import { Form, Input, Select, Button, Card, message } from "antd";
import { useNavigate } from "react-router-dom";
import { useUsersStore } from "../../../store/users";
import ItemSideBar from "../../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../../layout/Header/components/ItemHeaderCabecalho";
import { UserRegister } from "../../../types/users";

const { Option } = Select;

const UsersRegister: React.FC = () => {
  const [form] = Form.useForm<UserRegister>();
  const navigate = useNavigate();
  const { addUser } = useUsersStore();
  const [loading, setLoading] = useState(false);

  const onFinish = async (vals: UserRegister) => {
    setLoading(true);
    try {
      await addUser(vals);
      message.success("UsuÃ¡rio cadastrado!");
      navigate("/users");
    } catch {
      message.error("Erro ao cadastrar!");
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          <ItemHeaderCabecalho
            title="Cadastro de UsuÃ¡rio"
            subTitle="Preencha os dados"
          />
          <Card>
            <Form
              form={form}
              layout="vertical"
              onFinish={onFinish}
              initialValues={{ role: "ADMIN" }}
            >
              <Form.Item
                name="username"
                label="Username"
                rules={[{ required: true, message: "Informe o username" }]}
              >
                <Input />
              </Form.Item>
              <Form.Item
                name="name"
                label="Nome"
                rules={[{ required: true, message: "Informe o nome" }]}
              >
                <Input />
              </Form.Item>
              <Form.Item
                name="email"
                label="Email"
                rules={[
                  { required: true, message: "Informe o email" },
                  { type: "email", message: "Email invÃ¡lido" },
```

```
              ]}
            >
              <Input />
            </Form.Item>
            <Form.Item
              name="password"
              label="Senha"
              rules={[{ required: true, message: "Informe a senha" }]}
            >
              <Input.Password />
            </Form.Item>
            <Form.Item
              name="role"
              label="PermissÃ£o"
              rules={[{ required: true, message: "Selecione a permissÃ£o" }]}
            >
              <Select>
                <Option value="ADMIN">Admin</Option>
                <Option value="LIDER">LÃ-der</Option>
                <Option value="GAME">Game</Option>
              </Select>
            </Form.Item>
            <Form.Item style={{ textAlign: "right" }}>
              <Button
                onClick={() => navigate("/users")}
                style={{ marginRight: 8 }}
              >
                Cancelar
              </Button>
              <Button type="primary" htmlType="submit" loading={loading}>
                Cadastrar
              </Button>
            </Form.Item>
          </Form>
        </Card>
      </main>
    </div>
  </div>
 );
};

export default UsersRegister;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/users/hooks/useUsers.ts =====
import { useUsersStore } from "../../../store/users";

export const useUsers = () => {
  const { users, fetchUsers, addUser, updateUser, deleteUser } =
    useUsersStore();
  return { users, fetchUsers, addUser, updateUser, deleteUser };
};

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring/Monitoring.tsx =====
import React from "react";
import { PlusOutlined, FilterOutlined } from "@ant-design/icons";
import { useNavigate } from "react-router-dom";
import ItemSideBar from "../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../layout/Header/ItemHeader";
import Button from "../../components/Button/Button";
import CustomTable from "../../components/Table/Table";
import ItemHeaderCabecalho from "../../layout/Header/components/ItemHeaderCabecalho";
import { useMonitoringTable } from "./hooks/useMonitoringTable";

const Monitoring: React.FC = () => {
  const navigate = useNavigate();
  const { columns, monitorings, loading } = useMonitoringTable();

  return (
    <div className="layout-container">
```

```
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          {/* CabeÃ§alho da pÃ¡gina */}
          <ItemHeaderCabecalho
            title="Monitoramento de Energia"
            subTitle="Lista de Monitoramento de Energia jÃ¡ cadastrados"
          />

          {/* BotÃµes de aÃ§Ã£o */}
          <section className="actions-section">
            <Button
              type="primary"
              className="primary-btn"
              icon={<PlusOutlined />}
              onClick={() => navigate("/monitoring/register")}
            >
              Cadastrar Monitoramento
            </Button>
            <Button type="link" className="filter-btn" icon={<FilterOutlined />}>
              Filtros
            </Button>
          </section>

          {/* Tabela de monitoramentos */}
          <section className="table-container">
            <CustomTable columns={columns} data={monitorings} loading={loading} />
          </section>
        </main>
      </div>
    </div>
  );
};

export default Monitoring;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring/components/MonitoringSe
ctions.tsx =====

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring/components/MonitoringDe
tails.tsx =====
import React from "react";
import { MonitoringItem } from "../../../types/monitoringTypes";

interface Props {
  monitoring: MonitoringItem;
}

const MonitoringDetails: React.FC<Props> = ({ monitoring }) => {
  return (
    <section className="monitoring-info">
      <div className="monitoring-table">
        <div className="monitoring-row">
          <div className="monitoring-cell">
            <span className="info-label">Nome do Monitoramento:</span>
            <span className="info-value">{monitoring.name}</span>
          </div>
          <div className="monitoring-cell">
            <span className="info-label">DescriÃ§Ã£o:</span>
            <span className="info-value">{monitoring.description}</span>
          </div>
          <div className="monitoring-cell">
            <span className="info-label">Consumo Estimado:</span>
            <span className="info-value">{monitoring.estimated_consumption} kWh</span>
          </div>
        </div>
      </div>
    </section>
```

```
  );
};

export default MonitoringDetails;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring/components/columnsWithA
ctions.tsx =====
import { SectionItem } from "@/types/sections";
import { ColumnsType } from "antd/es/table";
import Actions from "@/components/actions/Actions"; // <-- Import corrigido

type Handlers = {
  onEdit: (id: number) => void;
  onDelete: (id: number) => void;
  onConfigure: (section: SectionItem) => void;
};

export const columnsWithActions = (
  baseColumns: ColumnsType<SectionItem>,
  { onEdit, onDelete, onConfigure }: Handlers
): ColumnsType<SectionItem> => {
  return [
    ...baseColumns,
    {
      title: "AÃ§Ãµes",
      key: "actions",
      render: (_: unknown, record: SectionItem) => (
        <Actions
          onEdit={() => onEdit(record.id)}
          onDelete={() => onDelete(record.id)}
          onConfigure={() => onConfigure(record)}
        />
      ),
    },
  ];
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring/components/SectionList.
tsx =====
import { Table, Badge, Button, Tooltip } from "antd";
import { ThunderboltOutlined } from "@ant-design/icons";
import { useNavigate, useParams } from "react-router-dom";
import { useSectionTable } from "../hooks/useSectionTable";
import { useSectionHierarchy } from "../hooks/useSectionHierarchy";
import { useSectionActions } from "../hooks/useSectionActions";
import { columnsWithActions } from "./columnsWithActions";
import SectionExpandedTree from "./SectionExpandedTree";
import MonitoringConfigureSectionModal from "./MonitoringConfigureSectionModal";
import { SectionItem } from "@/types/sections";

const SectionList = () => {
  const { id } = useParams<{ id: string }>();
  const navigate = useNavigate();

  const {
    columns,
    sections,
    loading,
    sectionToConfigure,
    isConfigureModalVisible,
    handleOpenConfigure,
    handleCloseConfigure,
  } = useSectionTable();

  const { handleDelete } = useSectionActions();
  const { setoresPrincipais } = useSectionHierarchy(sections);

  // Verifica se uma seÃ§Ã£o ou qualquer descendente possui IoT (LED Verde)
  const hasIotDeviceRecursive = (section: SectionItem, allSections: SectionItem[]): boolean
 => {
    if (section.DeviceIot) return true;
```

```
    const children = allSections.filter((s) => s.secticon_parent === section.id);
    return children.some((child) => hasIotDeviceRecursive(child, allSections));
  };

  // Verifica se a seÃ§Ã£o tem um dispositivo IoT diretamente associado (Ã\215cone de Monit
oramento)
  const hasDirectIotDevice = (section: SectionItem): boolean => {
    return !!section.DeviceIot;
  };

  // FunÃ§Ã£o para lidar com o clique no Ã-cone de monitoramento
  const handleMonitorClick = (section: SectionItem) => {
    console.log("ð\237\224\215 Monitoramento da seÃ§Ã£o:", section.name);
    // Aqui vamos abrir o mini modal futuramente
  };

  // Substitui a coluna "Nome" para adicionar LED verde e Ã-cone de monitoramento
  const enhancedColumns = columns.map((col) => {
    if (col.key === "name") {
      return {
        ...col,
        render: (_: unknown, record: SectionItem) => {
          const hasIot = hasIotDeviceRecursive(record, sections);
          const hasDirectIot = hasDirectIotDevice(record);

          return (
            <span style={{ display: "flex", alignItems: "center" }}>
              {/* LED Verde para toda a Ã¡rvore */}
              {hasIot && <Badge status="success" style={{ marginRight: 6 }} />}
              {record.name}
              {/* Ã\215cone de Monitoramento apenas para a seÃ§Ã£o diretamente associada */
}
              {hasDirectIot && (
                <Tooltip title="Monitoramento Ativo">
                  <Button
                    type="text"
                    icon={<ThunderboltOutlined />}
                    onClick={() => handleMonitorClick(record)}
                    style={{ marginLeft: 8, color: "#faad14" }}
                  />
                </Tooltip>
              )}
            </span>
          );
        },
      };
    }
    return col;
  });

  const actionColumns = columnsWithActions(enhancedColumns, {
    onEdit: (sectionId) => navigate(`/monitoring/edit-section/${sectionId}`),
    onDelete: handleDelete,
    onConfigure: handleOpenConfigure,
  });

  return (
    <>
      <Table
        columns={actionColumns}
        dataSource={setoresPrincipais.filter((s) => s.monitoring === Number(id))}
        loading={loading}
        rowKey="id"
        expandable={{
          expandedRowRender: (record: SectionItem) => (
            <SectionExpandedTree
              section={record}
              allSections={sections}
              onConfigure={handleOpenConfigure}
```

```
              onDelete={handleDelete}
              onMonitor={handleMonitorClick}
            />
          ),
        }}
        pagination={{ pageSize: 10 }}
      />

      {sectionToConfigure && (
        <MonitoringConfigureSectionModal
          section={sectionToConfigure}
          open={isConfigureModalVisible}
          onClose={handleCloseConfigure}
        />
      )}
    </>
  );
};

export default SectionList;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring/components/MonitoringCo
nfigureSectionModal.tsx =====
import React, { useEffect, useState } from "react";
import { Modal, Switch, Select, message } from "antd";
import { useIoTDevices } from "@/hooks/useIoTDevices";
import { SectionItem } from "@/types/sections";
import { useSectionStore } from "@/store/sectionStore";

interface Props {
  section: SectionItem;
  open: boolean;
  onClose: () => void;
}

const MonitoringConfigureSectionModal: React.FC<Props> = ({ section, open, onClose }) => {
  const { devices, fetchDevices } = useIoTDevices();
  const { updateSection } = useSectionStore();

  const [form, setForm] = useState({
    is_monitored: section.is_monitored,
    deviceIot: section.DeviceIot ?? null,
  });

  const [loading, setLoading] = useState(false);

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

  const handleChange = (name: string, value: any) => {
    setForm((prev) => ({
      ...prev,
      [name]: value,
    }));
  };

  const handleSave = async () => {
    try {
      setLoading(true);
      await updateSection(section.id, {
        name: section.name,
        description: section.description,
        is_monitored: form.is_monitored,
        DeviceIot: form.is_monitored ? form.deviceIot : null,
        monitoring: section.monitoring,
        setor: section.setor,
        productionLine: section.productionLine,
        equipament: section.equipament,
        type_section: section.type_section,
```

```
        secticon_parent: section.secticon_parent,
      });

      message.success("SeÃ§Ã£o atualizada com sucesso!");
      onClose();
    } catch (error) {
      console.error("Erro ao configurar seÃ§Ã£o:", error);
      message.error("Erro ao configurar seÃ§Ã£o.");
    } finally {
      setLoading(false);
    }
  };

  return (
    <Modal
      title="Configurar SeÃ§Ã£o"
      open={open}
      onCancel={onClose}
      onOk={handleSave}
      confirmLoading={loading}
    >
      <div style={{ marginBottom: 20 }}>
        <p>Monitorado?</p>
        <Switch
          checked={form.is_monitored}
          onChange={(value) => handleChange("is_monitored", value)}
        />
      </div>

      {form.is_monitored && (
        <div>
          <p>Dispositivo IoT</p>
          <Select
            style={{ width: "100%" }}
            value={form.deviceIot}
            onChange={(value) => handleChange("deviceIot", value)}
            options={devices.map((device) => ({
              value: device.id,
              label: device.name,
            }))}
          />
        </div>
      )}
    </Modal>
  );
};

export default MonitoringConfigureSectionModal;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring/components/MonitoringCo
nfigure.tsx =====
import React from "react";
import { PlusOutlined } from "@ant-design/icons";
import { useNavigate } from "react-router-dom";
import ItemSideBar from "../../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../../layout/Header/components/ItemHeaderCabecalho";
import Button from "../../../components/Button/Button";
import MonitoringDetails from "./MonitoringDetails";
import { useMonitoringById } from "../hooks/useMonitoringById";
import SectionList from "./SectionList";

const MonitoringConfigure: React.FC = () => {
  const { monitoring, loading } = useMonitoringById();
  const navigate = useNavigate();

  if (loading) return <p>Carregando...</p>;
  if (!monitoring) return <p>Monitoramento nÃ£o encontrado.</p>;

  return (
```

```
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          <ItemHeaderCabecalho
            title="Configurar Monitoramento de Energia"
            subTitle="Monitoramento de Energia"
          />

          {/* InformaÃ§Ãµes do Monitoramento */}
          <MonitoringDetails monitoring={monitoring} />

          {/* BotÃ£o Adicionar SeÃ§Ã£o */}
          <section className="actions-section">
            <Button
              type="primary"
              className="add-section-btn"
              icon={<PlusOutlined />}
              onClick={() => navigate(`/monitoring/add-section/${monitoring.id}`)}
            >
              Adicionar SeÃ§Ã£o
            </Button>
          </section>

          {/* Lista de SeÃ§Ãµes */}
          <section className="table-container">
            <p>Lista de Consumo (SeÃ§Ãµes associadas)</p>
            <SectionList /> {/* Adiciona o componente da lista de seÃ§Ãµes */}
          </section>
        </main>
      </div>
    </div>
  );
};

export default MonitoringConfigure;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring/components/SectionForm.
tsx =====
import React from "react";
import DynamicForm from "@/components/form/DynamicForm";
import { SectionFormValues } from "@/pages/monitoring/hooks/useSectionForm";
import { TypeSection } from "@/store/typeSectionStore";

interface SectionFormProps {
  values: SectionFormValues;
  onChange: (name: keyof SectionFormValues, value: any) => void;
  onCancel: () => void;
  onSubmit: () => void;
  loading: boolean;
  availableSections: { value: number; label: string }[];
  devices: { id: number; name: string }[];
  typeSections: TypeSection[]; // â\234\205 tipos reais
  isEdit?: boolean;
}

const SectionForm: React.FC<SectionFormProps> = ({
  values,
  onChange,
  onCancel,
  onSubmit,
  loading,
  availableSections,
  devices,
  typeSections,
  isEdit = false,
}) => {
  return (
    <DynamicForm
```

```
      fields={[

        {
          name: "is_monitored",
          label: "Monitorado?",
          type: "switch",
        },
        {
          name: "type_section",
          label: "Tipo da SeÃ§Ã£o",
          type: "select",
          options: typeSections.map((t) => ({
            value: t.name, // "SETOR", "LINHA", "EQUIPAMENTO"
            label:
              t.name === "SETOR"
                ? "Setor"
                : t.name === "LINHA"
                ? "Linha de ProduÃ§Ã£o"
                : "Equipamento",
          })),
          disabled: isEdit,
        },
        {
          name: "section_consume",
          label: "SeÃ§Ã£o de Consumo",
          type: "select",
          options: availableSections,
          disabled: !values.type_section,
        },
        {
          name: "deviceIot",
          label: "Dispositivo IoT",
          type: "select",
          options: devices.map((device) => ({
            value: device.id,
            label: device.name,
          })),
          disabled: !values.is_monitored,
        },
      ]}
      values={values}
      onChange={(name, value) => onChange(name as keyof SectionFormValues, value)}
      loading={loading}
      onCancel={onCancel}
      onSubmit={onSubmit}
    />
  );
};

export default SectionForm;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring/components/SectionEdit.
tsx =====
import React, { useEffect, useState } from "react";
import { Modal, message } from "antd";
import { useParams, useNavigate } from "react-router-dom";
import { useSectionStore } from "@/store/sectionStore";
import { useSectionForm } from "@/pages/monitoring/hooks/useSectionForm";
import { useTypeSectionStore, TypeSection } from "@/store/typeSectionStore";
import SectionForm from "./SectionForm";

const SectionEdit: React.FC = () => {
  const { id } = useParams<{ id: string }>();
  const navigate = useNavigate();

  const { sections, fetchSections, updateSection } = useSectionStore();
  const { fetchTypes, types } = useTypeSectionStore();

  const {
    formValues,
```

```
    setFormValues,
    handleChange,
    getAvailableSections,
    devices,
  } = useSectionForm(true); // â\234\205 modo ediÃ§Ã£o ativado

  const [loading, setLoading] = useState(false);
  const [monitoringId, setMonitoringId] = useState<number | null>(null);
  const [loaded, setLoaded] = useState(false); // â\234\205 impede reexecuÃ§Ã£o do setFormV
alues

  useEffect(() => {
    fetchSections();
    fetchTypes();
  }, []);

  useEffect(() => {
    if (id && sections.length > 0 && types.length > 0 && !loaded) {
      const section = sections.find((s) => s.id === Number(id));
      if (!section) {
        message.error("SeÃ§Ã£o nÃ£o encontrada.");
        navigate("/monitoring");
        return;
      }

      const typeMatch = types.find((t: TypeSection) => t.id === section.type_section);
      if (!typeMatch) {
        message.error("Tipo da seÃ§Ã£o invÃ¡lido.");
        return;
      }

      setMonitoringId(section.monitoring || null);

      setFormValues({
        name: section.name || "",
        is_monitored: !!section.is_monitored,
        type_section: typeMatch.name as "SETOR" | "LINHA" | "EQUIPAMENTO",
        section_consume:
          section.setor ?? section.productionLine ?? section.equipament ?? null,
        deviceIot: section.DeviceIot ?? null,
      });

      setLoaded(true); // â\234\205 garante que isso sÃ³ roda uma vez
    }
  }, [id, sections, types, loaded]);

  const handleSubmit = async () => {
    if (!formValues.name.trim()) {
      message.error("O nome da seÃ§Ã£o Ã© obrigatÃ³rio!");
      return;
    }

    const typeId = types.find((t: TypeSection) => t.name === formValues.type_section)?.id;

    if (!typeId) {
      message.error("Tipo da seÃ§Ã£o invÃ¡lido!");
      return;
    }

    setLoading(true);
    try {
      await updateSection(Number(id), {
        name: formValues.name,
        is_monitored: formValues.is_monitored,
        type_section: typeId,
        DeviceIot: formValues.is_monitored ? formValues.deviceIot : null,
        setor: formValues.type_section === "SETOR" ? formValues.section_consume : null,
        productionLine: formValues.type_section === "LINHA" ? formValues.section_consume :
null,
```

```
        equipament: formValues.type_section === "EQUIPAMENTO" ? formValues.section_consume
: null,
      });

      message.success("SeÃ§Ã£o atualizada com sucesso!");
      navigate(`/monitoring/configure/${monitoringId}`);
    } catch (error) {
      console.error("â\235\214 Erro ao atualizar:", error);
      message.error("Erro ao atualizar seÃ§Ã£o!");
    } finally {
      setLoading(false);
    }
  };

  return (
    <Modal
      title="Editar SeÃ§Ã£o"
      open={true}
      footer={null}
      onCancel={() => navigate(`/monitoring/configure/${monitoringId}`)}
    >
      <SectionForm
        values={formValues}
        onChange={handleChange}
        onCancel={() => navigate(`/monitoring/configure/${monitoringId}`)}
        onSubmit={handleSubmit}
        loading={loading}
        availableSections={getAvailableSections()}
        devices={devices}
        typeSections={types}
        isEdit={true}
      />
    </Modal>
  );
};

export default SectionEdit;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring/components/SectionExpan
dedTree.tsx =====
import { FC } from "react";
import { Tree, Collapse, Button, Popconfirm, Badge, Tooltip } from "antd";
import { SettingOutlined, DeleteOutlined, ThunderboltOutlined } from "@ant-design/icons";
import { SectionItem } from "@/types/sections";

type Props = {
  section: SectionItem;
  allSections: SectionItem[];
  onConfigure: (section: SectionItem) => void;
  onDelete: (sectionId: number) => void;
  onMonitor: (section: SectionItem) => void;
};

// Verifica recursivamente se a seÃ§Ã£o ou alguma de suas filhas tem DeviceIot
const hasIotDeviceRecursive = (section: SectionItem, allSections: SectionItem[]): boolean =
> {
  if (section.DeviceIot) return true;
  const children = allSections.filter((s) => s.secticon_parent === section.id);
  return children.some((child) => hasIotDeviceRecursive(child, allSections));
};

// Verifica se a seÃ§Ã£o tem um dispositivo IoT diretamente associado
const hasDirectIotDevice = (section: SectionItem): boolean => {
  return !!section.DeviceIot;
};

const SectionExpandedTree: FC<Props> = ({ section, allSections, onConfigure, onDelete, onMo
nitor }) => {
  const buildTree = (parent: SectionItem): any => {
    const children = allSections.filter((s) => s.secticon_parent === parent.id);
```

```
    const hasIot = hasIotDeviceRecursive(parent, allSections);
    const hasDirectIot = hasDirectIotDevice(parent);

    return {
      title: (
        <span>
          {/* LED de status se houver IoT em qualquer nÃ-vel */}
          {hasIot && <Badge status="success" style={{ marginRight: 6 }} />}
          {parent.name}

          {/* Ã\215cone de Monitoramento, apenas se houver IoT diretamente associado */}
          {hasDirectIot && (
            <Tooltip title="Monitoramento Ativo">
              <Button
                type="text"
                icon={<ThunderboltOutlined />}
                onClick={() => onMonitor(parent)}
                style={{ marginLeft: 6, color: "#faad14" }} // Cor amarela
              />
            </Tooltip>
          )}

          <Button
            type="text"
            icon={<SettingOutlined />}
            onClick={() => onConfigure(parent)}
            style={{ marginLeft: 8 }}
          />
          <Popconfirm
            title="Deseja excluir esta subseÃ§Ã£o?"
            onConfirm={() => onDelete(parent.id)}
            okText="Sim"
            cancelText="NÃ£o"
          >
            <Button
              type="text"
              icon={<DeleteOutlined />}
              danger
              style={{ marginLeft: 4 }}
            />
          </Popconfirm>
        </span>
      ),
      key: `section-${parent.id}`,
      children: children.map(buildTree),
    };
  };

  const rootChildren = allSections
    .filter((s) => s.secticon_parent === section.id)
    .map(buildTree);

  return (
    <Collapse
      items={[
        {
          key: `panel-${section.id}`,
          label: "SubseÃ§Ãµes",
          children: rootChildren.length > 0 ? (
            <Tree treeData={rootChildren} />
          ) : (
            <p>NÃ£o hÃ¡ subseÃ§Ãµes associadas.</p>
          ),
        },
      ]}
    />
  );
};
```

```
export default SectionExpandedTree;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring/components/MonitoringFo
rm.tsx =====
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import { message } from "antd";
import ItemSideBar from "../../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../../layout/Header/components/ItemHeaderCabecalho";
import DynamicForm from "../../../components/form/DynamicForm";
import { useMonitoring } from "../hooks/useMonitoring";

const MonitoringForm: React.FC = () => {
  const navigate = useNavigate();
  const { addMonitoring } = useMonitoring();
  const [loading, setLoading] = useState(false);
  const [formValues, setFormValues] = useState({
    name: "",
    description: "",
    estimated_consumption: "",
  });

  const handleChange = (name: string, value: any) => {
    setFormValues((prev) => ({ ...prev, [name]: value }));
  };

  const handleSubmit = async () => {
    if (!formValues.name.trim()) {
      message.error("O nome do monitoramento Ã© obrigatÃ³rio!");
      return;
    }

    if (!formValues.description.trim()) {
      message.error("A descriÃ§Ã£o do monitoramento Ã© obrigatÃ³ria!");
      return;
    }

    if (loading) return;

    setLoading(true);
    try {
      await addMonitoring({
        name: formValues.name,
        description: formValues.description,
        estimated_consumption: Number(formValues.estimated_consumption) || 0,
      });

      message.success("Cadastro de monitoramento realizado com sucesso!");
      navigate("/monitoring");
    } catch (error) {
      message.error("Erro ao cadastrar monitoramento!");
      console.error(error);
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          <ItemHeaderCabecalho
            title="Cadastro Monitoramento de Energia"
            subTitle="FormulÃ¡rio para cadastro de Monitoramento de Energia"
          />

          <DynamicForm
```

```
            fields={[
              { name: "name", label: "Nome do Monitoramento", type: "input", required: true
 },
              { name: "description", label: "DescriÃ§Ã£o", type: "textarea", required: true
 },
              { name: "estimated_consumption", label: "Consumo Estimado (Kw/H)", type: "num
ber" },
            ]}
            values={formValues}
            onChange={handleChange}
            onSubmit={handleSubmit}
            loading={loading}
            onCancel={() => navigate("/monitoring")}
          />
        </main>
      </div>
    </div>
  );
};

export default MonitoringForm;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring/components/MonitoringAd
dSection.tsx =====
import React, { useEffect, useState } from "react";
import { Modal, message } from "antd";
import { useParams, useNavigate } from "react-router-dom";
import { useSectionStore } from "@/store/sectionStore";
import { useSectionForm } from "@/pages/monitoring/hooks/useSectionForm";
import SectionForm from "./SectionForm";

const MonitoringAddSection: React.FC = () => {
  const { id } = useParams<{ id: string }>();
  const navigate = useNavigate();
  const { addSection } = useSectionStore();

  const {
    formValues,
    handleChange,
    getAvailableSections,
    devices,
    typeSections, // â\234\205 Tipos de seÃ§Ã£o da API
  } = useSectionForm();

  const [loading, setLoading] = useState(false);

  const handleSubmit = async () => {
    if (!formValues.name.trim()) {
      message.error("O nome da seÃ§Ã£o Ã© obrigatÃ³rio!");
      return;
    }

    const typeId = typeSections.find((t) => t.name === formValues.type_section)?.id;

    if (!typeId) {
      message.error("ID do tipo de seÃ§Ã£o nÃ£o encontrado!");
      return;
    }

    console.log("ð\237\221\211 Enviando tipo_section ID:", typeId);

    setLoading(true);
    try {
      await addSection({
        name: formValues.name,
        is_monitored: formValues.is_monitored,
        monitoring: Number(id),
        type_section: typeId,
        DeviceIot: formValues.is_monitored ? formValues.deviceIot : null,
```

```
        setor: formValues.type_section === "SETOR" ? formValues.section_consume : null,
        productionLine: formValues.type_section === "LINHA" ? formValues.section_consume :
null,
        equipament: formValues.type_section === "EQUIPAMENTO" ? formValues.section_consume
: null,
      });

      message.success("SeÃ§Ã£o adicionada com sucesso!");
      navigate(`/monitoring/configure/${id}`);
    } catch (error) {
      console.error("â\235\214 Erro real ao adicionar:", error);
      message.error("Erro ao adicionar seÃ§Ã£o!");
    } finally {
      setLoading(false);
    }
  };

  return (
    <Modal
      title="Adicionar Nova SeÃ§Ã£o"
      open={true}
      footer={null}
      onCancel={() => navigate(`/monitoring/configure/${id}`)}
    >
      <SectionForm
        values={formValues}
        onChange={handleChange}
        onCancel={() => navigate(`/monitoring/configure/${id}`)}
        onSubmit={handleSubmit}
        loading={loading}
        availableSections={getAvailableSections()}
        devices={devices}
        typeSections={typeSections} // â\234\205 Passa os tipos pro form
      />
    </Modal>
  );
};

export default MonitoringAddSection;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring/components/MonitoringEd
it.tsx =====
// src/pages/monitoring/components/MonitoringEdit.tsx

import React, { useEffect, useState } from "react";
import { useParams, useNavigate } from "react-router-dom";
import { message } from "antd";
import ItemSideBar from "../../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../../layout/Header/components/ItemHeaderCabecalho";
import DynamicForm from "../../../components/form/DynamicForm";
import { useMonitoring } from "../hooks/useMonitoring";

const MonitoringEdit: React.FC = () => {
  const { id } = useParams<{ id: string }>();
  const navigate = useNavigate();
  const { monitorings, editMonitoring } = useMonitoring();

  const [loading, setLoading] = useState(false);
  const [formValues, setFormValues] = useState({
    name: "",
    description: "",
    estimated_consumption: "",
  });

  useEffect(() => {
    if (id) {
      const monitoring = monitorings.find((item) => item.id === Number(id));
      if (monitoring) {
        setFormValues({
```

```
          name: monitoring.name,
          description: monitoring.description,
          estimated_consumption: monitoring.estimated_consumption.toString(),
        });
      } else {
        message.error("Monitoramento nÃ£o encontrado.");
        navigate("/monitoring");
      }
    }
  }, [id, monitorings]);

  const handleChange = (name: string, value: any) => {
    setFormValues((prev) => ({ ...prev, [name]: value }));
  };

  const handleSubmit = async () => {
    if (!formValues.name.trim()) {
      message.error("O nome do monitoramento Ã© obrigatÃ³rio!");
      return;
    }

    if (!formValues.description.trim()) {
      message.error("A descriÃ§Ã£o do monitoramento Ã© obrigatÃ³ria!");
      return;
    }

    setLoading(true);
    try {
      await editMonitoring(Number(id), {
        name: formValues.name,
        description: formValues.description,
        estimated_consumption: Number(formValues.estimated_consumption) || 0,
      });

      message.success("Monitoramento atualizado com sucesso!");
      navigate("/monitoring");
    } catch (error) {
      message.error("Erro ao atualizar monitoramento!");
      console.error(error);
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          <ItemHeaderCabecalho
            title="Editar Monitoramento de Energia"
            subTitle="FormulÃ¡rio para editar dados do monitoramento"
          />
          <DynamicForm
            fields={[
              { name: "name", label: "Nome do Monitoramento", type: "input", required: true
 },
              { name: "description", label: "DescriÃ§Ã£o", type: "textarea", required: true
 },
              { name: "estimated_consumption", label: "Consumo Estimado (Kw/H)", type: "num
ber" },
            ]}
            values={formValues}
            onChange={handleChange}
            onSubmit={handleSubmit}
            loading={loading}
            onCancel={() => navigate("/monitoring")}
          />
```

```
        </main>
      </div>
    </div>
  );
};

export default MonitoringEdit;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring/hooks/useMonitoringTabl
e.tsx =====
import { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
import { useMonitoringStore } from "@/store/monitoringStore";
import { message } from "antd";
import { MonitoringItem } from "@/types/monitoringTypes";
import Actions from "@/components/actions/Actions";

export const useMonitoringTable = () => {
  const navigate = useNavigate();
  const { monitorings, fetchMonitorings, deleteMonitoring } = useMonitoringStore();
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchData = async () => {
      try {
        setLoading(true);
        await fetchMonitorings();
      } catch (error) {
        message.error("Erro ao carregar monitoramentos.");
      } finally {
        setLoading(false);
      }
    };

    fetchData();
  }, []);

  const columns = [
    {
      title: "Nome",
      dataIndex: "name",
      key: "name",
      sorter: (a: MonitoringItem, b: MonitoringItem) => a.name.localeCompare(b.name),
    },
    {
      title: "DescriÃ§Ã£o",
      dataIndex: "description",
      key: "description",
    },
    {
      title: "Consumo Estimado (kWh)",
      dataIndex: "estimated_consumption",
      key: "estimated_consumption",
    },
    {
      title: "AÃ§Ãµes",
      key: "actions",
      render: (_: any, record: MonitoringItem) => (
        <Actions
          onEdit={() => navigate(`/monitoring/edit/${record.id}`)} // Edita monitoramento
          onConfigure={() => navigate(`/monitoring/configure/${record.id}`)} // ð\237\224§
Vai pra configuraÃ§Ã£o de seÃ§Ãµes
          onDelete={async () => {
            if (record.id) {
              await deleteMonitoring(record.id);
              message.success("Monitoramento excluÃ-do.");
            }
          }}
        />
      ),
```

```
    }
  ];

  return { columns, monitorings, loading };
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring/hooks/useMonitoring.ts
=====
import { useEffect } from "react";
import { useMonitoringStore } from "@/store/monitoringStore";
import { getMonitorings, createMonitoring, updateMonitoring, deleteMonitoring } from "@/ser
vices/monitoringService";
import { MonitoringItem } from "@/types/monitoringTypes";

export const useMonitoring = () => {
  const { monitorings, fetchMonitorings, loading } = useMonitoringStore();

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

  const addMonitoring = async (data: Partial<MonitoringItem>) => {
    await createMonitoring(data);
    fetchMonitorings(); // Atualiza os dados apÃ³s a criaÃ§Ã£o
  };

  const editMonitoring = async (id: number, data: Partial<MonitoringItem>) => {
    await updateMonitoring(id, data);
    fetchMonitorings();
  };

  const removeMonitoring = async (id: number) => {
    await deleteMonitoring(id);
    fetchMonitorings();
  };

  return { monitorings, loading, addMonitoring, editMonitoring, removeMonitoring };
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring/hooks/useSection.ts ===
==
import { useEffect } from "react";
import { useSectionStore } from "@/store/sectionStore";

export const useSection = () => {
  const {
    sections,
    fetchSections,
    addSection,
    updateSection,
    deleteSection,
    loading,
  } = useSectionStore();

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

  return {
    sections,
    loading,
    addSection,
    editSection: updateSection,
    removeSection: deleteSection,
  };
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring/hooks/useSectionActions
.ts =====
import { useState } from "react";
import { message } from "antd";
import { SectionItem } from "@/types/sections";
```

```
import { useSectionStore } from "@/store/sectionStore";

export const useSectionActions = () => {
  const [sectionToConfigure, setSectionToConfigure] = useState<SectionItem | null>(null);
  const { deleteSection, fetchSections } = useSectionStore();

  const handleOpenConfig = (section: SectionItem) => {
    setSectionToConfigure(section);
  };

  const handleCloseConfig = () => {
    setSectionToConfigure(null);
  };

  const handleDelete = async (id: number) => {
    try {
      await deleteSection(id);
      await fetchSections(); // garante atualizaÃ§Ã£o visual apÃ³s excluir hierarquias
      message.success("SeÃ§Ã£o excluÃ-da com sucesso.");
    } catch (err) {
      console.error(err);
      message.error("Erro ao excluir seÃ§Ã£o.");
    }
  };

  const handleEdit = (id: number) => {
    // Aqui vocÃª pode redirecionar ou setar um estado global de ediÃ§Ã£o
    console.log("Editar seÃ§Ã£o com ID:", id);
  };

  return {
    sectionToConfigure,
    handleOpenConfig,
    handleCloseConfig,
    handleDelete,
    handleEdit,
  };
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring/hooks/useSectionTable.t
sx =====
import { useEffect, useState } from "react";
import { message } from "antd";
import { useSectionStore } from "@/store/sectionStore";
import { useIoTDevices } from "@/hooks/useIoTDevices";
import { SectionItem } from "@/types/sections";
import { useSectionHierarchy } from "./useSectionHierarchy";

export const useSectionTable = () => {
  // Stores
  const {
    sections,
    fetchSections,
    deleteSection,
    updateSection,
    loading,
  } = useSectionStore();

  const { devices, fetchDevices } = useIoTDevices();

  // Estados locais
  const [sectionToConfigure, setSectionToConfigure] = useState<SectionItem | null>(null);
  const [isConfigureModalVisible, setIsConfigureModalVisible] = useState(false);

  // Carregamento inicial
  useEffect(() => {
    const loadData = async () => {
      try {
        await Promise.all([fetchSections(), fetchDevices()]);
      } catch (error) {
```

```
      message.error("Erro ao carregar seÃ§Ãµes ou dispositivos.");
    }
  };

  loadData();
}, [fetchSections, fetchDevices]);

// Processa hierarquia das seÃ§Ãµes
const { setoresPrincipais, filteredSections } = useSectionHierarchy(sections);

// Abrir modal de configuraÃ§Ã£o
const handleOpenConfigure = (section: SectionItem) => {
  setSectionToConfigure(section);
  setIsConfigureModalVisible(true);
};

// Fechar modal
const handleCloseConfigure = () => {
  setIsConfigureModalVisible(false);
  setSectionToConfigure(null);
};

// Salvar configuraÃ§Ã£o de uma seÃ§Ã£o
const handleSaveConfigure = async (data: Partial<SectionItem>) => {
  if (!sectionToConfigure) return;

  try {
    const updatedData: Partial<SectionItem> = {
      ...sectionToConfigure,
      is_monitored: data.is_monitored,
      DeviceIot: data.is_monitored ? data.DeviceIot : null,
    };

    await updateSection(sectionToConfigure.id, updatedData);
    message.success("SeÃ§Ã£o configurada com sucesso.");
    handleCloseConfigure();
  } catch (error) {
    console.error(error);
    message.error("Erro ao configurar seÃ§Ã£o.");
  }
};

// Colunas da tabela
const baseColumns = [
  {
    title: "SeÃ§Ã£o",
    dataIndex: "name",
    key: "name",
    sorter: (a: SectionItem, b: SectionItem) => a.name.localeCompare(b.name),
  },
  {
    title: "DescriÃ§Ã£o",
    dataIndex: "description",
    key: "description",
  },
  {
    title: "Consumo Estimado (kWh)",
    dataIndex: "estimated_consumption",
    key: "estimated_consumption",
  },
];

return {
  columns: baseColumns,
  sections: filteredSections,
  setoresPrincipais,
  loading,
  sectionToConfigure,
  isConfigureModalVisible,
```

```
    setIsConfigureModalVisible,
    handleOpenConfigure,
    handleCloseConfigure,
    handleSaveConfigure,
    deleteSection,
    devices,
  };
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring/hooks/useSectionHierarc
hy.ts =====
import { useParams } from "react-router-dom";
import { SectionItem } from "@/types/sections";

/**
 * FunÃ§Ã£o recursiva que achata uma hierarquia de seÃ§Ãµes em uma lista Ãºnica.
 */
const flattenHierarchy = (sections: SectionItem[]): SectionItem[] => {
  const result: SectionItem[] = [];

  for (const section of sections) {
    result.push(section);
    if (section.sections_filhas && section.sections_filhas.length > 0) {
      result.push(...flattenHierarchy(section.sections_filhas));
    }
  }

  return result;
};

export const useSectionHierarchy = (
  sections: SectionItem[]
): {
  setoresPrincipais: SectionItem[];
  filteredSections: SectionItem[];
} => {
  const { id } = useParams<{ id: string }>();
  const monitoringId = Number(id);

  // Se id nÃ£o for vÃ¡lido, retorna arrays vazios
  if (isNaN(monitoringId)) {
    return {
      setoresPrincipais: [],
      filteredSections: [],
    };
  }

  // Setores principais (sem seÃ§Ã£o pai e pertencentes ao monitoramento)
  const setoresPrincipais = sections.filter(
    (s) => s.monitoring === monitoringId && !s.secticon_parent
  );

  // Achata a hierarquia a partir dos setores principais
  const filteredSections = flattenHierarchy(setoresPrincipais);

  return {
    setoresPrincipais,
    filteredSections,
  };
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring/hooks/useMonitoringById
.tsx =====
import { useEffect, useState } from "react";
import { useParams, useNavigate } from "react-router-dom";
import { useMonitoringStore } from "@/store/monitoringStore";
import { MonitoringItem } from "@/types/monitoringTypes";
import { message } from "antd";

export const useMonitoringById = () => {
  const { id } = useParams<{ id: string }>();
```

```
  const navigate = useNavigate();

  const {
    monitorings,
    fetchMonitorings,
    loading: storeLoading,
  } = useMonitoringStore();

  const [monitoring, setMonitoring] = useState<MonitoringItem | null>(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const loadMonitoring = async () => {
      setLoading(true);
      await fetchMonitorings();
    };

    loadMonitoring();
  }, [fetchMonitorings]);

  useEffect(() => {
    if (monitorings.length === 0) return;

    const found = monitorings.find((m) => m.id === Number(id));
    if (found) {
      setMonitoring(found);
    } else {
      message.error("Monitoramento nÃ£o encontrado.");
      navigate("/monitoring");
    }

    setLoading(false);
  }, [monitorings, id]);

  return { monitoring, loading: loading || storeLoading };
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring/hooks/useSectionForm.ts
 =====
import { useEffect, useState } from "react";
import { useSectorsStore } from "@/store/sectors";
import { useProductionLinesStore } from "@/store/ProductionLinesStore";
import { useEquipamentsStore } from "@/store/equipaments";
import { useIoTDevices } from "@/hooks/useIoTDevices";
import { useTypeSectionStore } from "@/store/typeSectionStore";
import { useSectionStore } from "@/store/sectionStore";

export type SectionFormValues = {
  name: string;
  is_monitored: boolean;
  type_section: "SETOR" | "LINHA" | "EQUIPAMENTO" | null;
  section_consume: number | null;
  deviceIot: number | null;
};

export const useSectionForm = (isEdit = false) => {
  const [formValues, setFormValues] = useState<SectionFormValues>({
    name: "",
    is_monitored: false,
    type_section: null,
    section_consume: null,
    deviceIot: null,
  });

  const { devices, fetchDevices } = useIoTDevices();
  const { sectors, fetchSectors } = useSectorsStore();
  const { productionLines, fetchProductionLines } = useProductionLinesStore();
  const { equipaments, fetchEquipaments } = useEquipamentsStore();
  const { types, fetchTypes } = useTypeSectionStore();
  const { fetchSections } = useSectionStore();
```

```
  useEffect(() => {
    fetchDevices();
    fetchSectors();
    fetchProductionLines();
    fetchEquipaments();
    fetchTypes();
  }, []);

  const filteredTypeSections = types.filter((t) => [1, 2, 3].includes(t.id)); // Apenas os
fixos

  const getSelectedLabelFromId = (
    type: SectionFormValues["type_section"],
    id: number | null
  ) => {
    if (!id) return "";
    switch (type) {
      case "SETOR":
        return sectors.find((s) => s.id === id)?.name ?? "";
      case "LINHA":
        return productionLines.find((l) => l.id === id)?.name ?? "";
      case "EQUIPAMENTO":
        return equipaments.find((e) => e.id === id)?.name ?? "";
      default:
        return "";
    }
  };

  const handleChange = <K extends keyof SectionFormValues>(
    name: K,
    value: SectionFormValues[K]
  ) => {
    setFormValues((prev) => {
      if (name === "is_monitored") {
        return {
          ...prev,
          is_monitored: Boolean(value),
          deviceIot: value ? prev.deviceIot : null,
        };
      }

      if (name === "type_section") {
        return {
          ...prev,
          type_section: value as SectionFormValues["type_section"],
          section_consume: null,
          name: "", // Limpa o nome para ser atualizado com a seÃ§Ã£o de consumo
          deviceIot: null,
        };
      }

      if (name === "section_consume") {
        const label = getSelectedLabelFromId(formValues.type_section, value as number);

        if (isEdit) fetchSections();

        return {
          ...prev,
          section_consume: value as number,
          name: label ? label : prev.name, // Atualiza o nome automaticamente
          deviceIot: null,
        };
      }

      return {
        ...prev,
        [name]: value,
      };
```

```
    });
  };


  const getAvailableSections = () => {
    switch (formValues.type_section) {
      case "SETOR":
        return sectors.map((s) => ({ value: s.id, label: s.name }));
      case "LINHA":
        return productionLines.map((l) => ({ value: l.id, label: l.name }));
      case "EQUIPAMENTO":
        return equipaments.map((e) => ({ value: e.id, label: e.name }));
      default:
        return [];
    }
  };

  return {
    formValues,
    setFormValues,
    handleChange,
    getAvailableSections,
    devices,
    typeSections: filteredTypeSections,
  };
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/productionLines/components/Product
ionLinesRegister.tsx =====
import React, { useState, useEffect } from "react";
import { message } from "antd";
import { useNavigate } from "react-router-dom";
import ItemSideBar from "../../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../../layout/Header/components/ItemHeaderCabecalho";
import DynamicForm from "../../../components/form/DynamicForm";
import { useProductionLinesStore } from "../../../store/ProductionLinesStore";
import { getSectors } from "../../../services/SectorsService";

const ProductionLinesRegister: React.FC = () => {
  const navigate = useNavigate();
  const { createProductionLine } = useProductionLinesStore();
  const [loading, setLoading] = useState(false);
  const [loadingOptions, setLoadingOptions] = useState(true);
  const [sectorsOptions, setSectorsOptions] = useState<{ value: number; label: string }[]>(
[]);

  const [formValues, setFormValues] = useState({
    name: "",
    description: "",
    value_mensuration_estimated: "",
    setor: null,
  });

  useEffect(() => {
    const fetchSectors = async () => {
      try {
        const sectorsData = await getSectors();
        setSectorsOptions(
          sectorsData.map((sector) => ({ value: sector.id, label: sector.name }))
        );
        setLoadingOptions(false);
      } catch (error) {
        message.error("Erro ao carregar setores. Tente novamente.");
        console.error("Erro ao buscar setores:", error);
        setLoadingOptions(false);
      }
    };

    fetchSectors();
```

```
  }, []);

  const handleChange = (name: string, value: any) => {
    setFormValues((prev) => ({ ...prev, [name]: value }));
  };

  const handleSubmit = async () => {
    if (!formValues.name.trim()) {
      message.error("O nome da linha de produÃ§Ã£o Ã© obrigatÃ³rio!");
      return;
    }

    setLoading(true);
    try {
      await createProductionLine({
        name: formValues.name,
        description: formValues.description || "",
        value_mensuration_estimated: formValues.value_mensuration_estimated
          ? Number(formValues.value_mensuration_estimated)
          : 0,
        setor: formValues.setor ? Number(formValues.setor) : null,
      });

      message.success("Linha de produÃ§Ã£o cadastrada com sucesso!");
      navigate("/production-lines"); // ð\237\224¹ Ajustado para a URL correta
    } catch (error) {
      message.error("Erro ao cadastrar linha de produÃ§Ã£o. Verifique os dados e tente nova
mente.");
      console.error("Erro ao cadastrar linha de produÃ§Ã£o:", error);
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          <ItemHeaderCabecalho
            title="Cadastro de Linha de ProduÃ§Ã£o"
            subTitle="Preencha os campos abaixo para cadastrar uma nova linha de produÃ§Ã£o
"
          />
          <DynamicForm
            fields={[
              { name: "name", label: "Nome", type: "input", required: true },
              { name: "description", label: "DescriÃ§Ã£o", type: "textarea" },
              { name: "value_mensuration_estimated", label: "Valor Estimado", type: "number
" },
              {
                name: "setor",
                label: "Setor",
                type: "select",
                options: sectorsOptions,
                disabled: loadingOptions,
              },
            ]}
            values={formValues}
            onChange={handleChange}
            onSubmit={handleSubmit}
            loading={loading}
            onCancel={() => navigate("/production-lines")}
          />
        </main>
      </div>
    </div>
  );
```

```
};

export default ProductionLinesRegister;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/productionLines/components/Product
ionLinesEdit.tsx =====
import React, { useEffect, useState } from "react";
import { useNavigate, useParams } from "react-router-dom";
import { message } from "antd";
import ItemSideBar from "../../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../../layout/Header/components/ItemHeaderCabecalho";
import DynamicForm from "../../../components/form/DynamicForm";
import { useProductionLinesStore } from "../../../store/ProductionLinesStore";
import { ProductionLine } from "../../../types/ProductionLinesTypes";
import { getProductionLineById } from "../../../services/ProductionLinesService";
import { getSectors } from "../../../services/SectorsService";

const ProductionLinesEdit: React.FC = () => {
  const navigate = useNavigate();
  const { id } = useParams<{ id: string }>();
  const { updateProductionLine } = useProductionLinesStore();
  const [loading, setLoading] = useState(false);
  const [loadingOptions, setLoadingOptions] = useState(true);
  const [formValues, setFormValues] = useState<Partial<ProductionLine>>({
    name: "",
    description: "",
    value_mensuration_estimated: undefined, // ð\237\224¹ Corrigido para evitar erro de tip
o
    setor: undefined, // ð\237\224¹ Corrigido para evitar erro de tipo
  });

  const [sectorsOptions, setSectorsOptions] = useState<{ value: number; label: string }[]>(
[]);

  // ð\237\224¹ Busca a linha de produÃ§Ã£o e os setores ao carregar a pÃ¡gina
  useEffect(() => {
    const fetchData = async () => {
      setLoading(true);
      try {
        if (id) {
          const data = await getProductionLineById(Number(id));
          setFormValues({
            name: data.name,
            description: data.description,
            value_mensuration_estimated: data.value_mensuration_estimated ?? undefined, //
ð\237\224¹ Garante tipo correto
            setor: data.setor ?? undefined, // ð\237\224¹ Garante tipo correto
          });
        }

        // ð\237\224¹ Carrega setores para o select
        const sectorsData = await getSectors();
        setSectorsOptions(
          sectorsData.map((sector) => ({ value: sector.id, label: sector.name }))
        );
      } catch (error) {
        message.error("Erro ao carregar os dados!");
        console.error(error);
      } finally {
        setLoading(false);
        setLoadingOptions(false);
      }
    };

    fetchData();
  }, [id]);

  const handleChange = (name: string, value: any) => {
    setFormValues((prev) => ({ ...prev, [name]: value }));
```

```
  };

  const handleSubmit = async () => {
    if (!formValues.name?.trim()) {
      message.error("O nome da linha de produÃ§Ã£o Ã© obrigatÃ³rio!");
      return;
    }

    setLoading(true);
    try {
      await updateProductionLine(Number(id), {
        name: formValues.name,
        description: formValues.description,
        value_mensuration_estimated: formValues.value_mensuration_estimated ? Number(formVa
lues.value_mensuration_estimated) : 0, // ð\237\224¹ Corrigido
        setor: formValues.setor ? Number(formValues.setor) : undefined, // ð\237\224¹ Corri
gido
      });

      message.success("Linha de produÃ§Ã£o atualizada com sucesso!");
      navigate("/productionLines");
    } catch (error) {
      message.error("Erro ao atualizar linha de produÃ§Ã£o!");
      console.error(error);
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          <ItemHeaderCabecalho
            title="Editar Linha de ProduÃ§Ã£o"
            subTitle="Atualize os dados da linha de produÃ§Ã£o"
          />
          <DynamicForm
            fields={[
              { name: "name", label: "Nome", type: "input", required: true },
              { name: "description", label: "DescriÃ§Ã£o", type: "textarea" },
              { name: "value_mensuration_estimated", label: "Valor Estimado", type: "number
" },
              {
                name: "setor",
                label: "Setor",
                type: "select",
                options: sectorsOptions,
                disabled: loadingOptions, // ð\237\224¹ Desativa o select enquanto carrega
              },
            ]}
            values={formValues}
            onChange={handleChange}
            onSubmit={handleSubmit}
            loading={loading}
            onCancel={() => navigate("/productionLines")}
          />
        </main>
      </div>
    </div>
  );
};

export default ProductionLinesEdit;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/productionLines/ProductionLines.ts
x =====
import React from "react";
```

```
import { PlusOutlined, FilterOutlined } from "@ant-design/icons";
import { useNavigate } from "react-router-dom";
import ItemSideBar from "../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../layout/Header/ItemHeader";
import Button from "../../components/Button/Button";
import CustomTable from "../../components/Table/Table";
import ItemHeaderCabecalho from "../../layout/Header/components/ItemHeaderCabecalho";
import { useProductionLinesTable } from "./hooks/useProductionLinesTable.tsx"; // ð\237\224
¹ Hook especÃ-fico para tabela

const ProductionLines: React.FC = () => {
  const navigate = useNavigate();
  const { columns, productionLines, loading } = useProductionLinesTable();

  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          {/* CabeÃ§alho da pÃ¡gina */}
          <ItemHeaderCabecalho
            title="Linhas de ProduÃ§Ã£o"
            subTitle="Gerencie as linhas de produÃ§Ã£o cadastradas"
          />

          {/* BotÃµes de aÃ§Ã£o */}
          <section className="actions-section">
            <Button
              type="primary"
              className="primary-btn"
              icon={<PlusOutlined />}
              onClick={() => navigate("/production-lines/register")}
            >
              Cadastrar Linha de ProduÃ§Ã£o
            </Button>
            <Button type="link" className="filter-btn" icon={<FilterOutlined />}>
              Filtros
            </Button>
          </section>

          {/* Tabela utilizando o hook de Linhas de ProduÃ§Ã£o */}
          <section className="table-container">
            <CustomTable columns={columns} data={productionLines} loading={loading} />
          </section>
        </main>
      </div>
    </div>
  );
};

export default ProductionLines;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/productionLines/hooks/useProductio
nLinesTable.tsx =====
import { useEffect, useState } from "react";
import { message } from "antd";
import { useNavigate } from "react-router-dom";
import { SortOrder } from "antd/es/table/interface";
import { useProductionLinesStore } from "../../../store/ProductionLinesStore";
import { getSectors } from "../../../services/SectorsService"; // ð\237\224¹ Importa serviÃ
§o de setores
import Actions from "../../../components/actions/Actions";
import { ProductionLine } from "../../../types/ProductionLinesTypes";

export const useProductionLinesTable = () => {
  const navigate = useNavigate();
  const { productionLines, fetchProductionLines, removeProductionLine } = useProductionLine
sStore();
  const [loading, setLoading] = useState(true);
```

```
  const [sectors, setSectors] = useState<{ [key: number]: string }>({});

  useEffect(() => {
    const fetchProductionLinesFromAPI = async () => {
      try {
        setLoading(true);
        await fetchProductionLines();

        // ð\237\224¹ Buscar setores e mapear ID -> Nome
        const sectorsData = await getSectors();
        const mappedSectors: { [key: number]: string } = {};
        sectorsData.forEach((sector: any) => {
          mappedSectors[sector.id] = sector.name;
        });

        setSectors(mappedSectors);
      } catch (error) {
        console.error("Erro ao buscar linhas de produÃ§Ã£o ou setores:", error);
        message.error("Erro ao carregar os dados!");
      } finally {
        setLoading(false);
      }
    };

    fetchProductionLinesFromAPI();
  }, []);

  // DefiniÃ§Ã£o das colunas da tabela
  const columns = [
    {
      title: "Nome",
      dataIndex: "name",
      key: "name",
      sorter: (a: ProductionLine, b: ProductionLine) => a.name.localeCompare(b.name),
      sortDirections: ["ascend", "descend"] as SortOrder[],
      render: (text: string | undefined) => <strong>{text ?? "Sem nome"}</strong>,
    },
    {
      title: "DescriÃ§Ã£o",
      dataIndex: "description",
      key: "description",
      render: (text: string | undefined) => <span>{text ?? "Sem descriÃ§Ã£o"}</span>,
    },
    {
      title: "Setor",
      dataIndex: "setor",
      key: "setor",
      render: (id: number | null) => <span>{id ? sectors[id] ?? "NÃ£o informado" : "NÃ£o in
formado"}</span>,
    },
    {
      title: "Valor Mensurado",
      dataIndex: "value_mensuration_estimated",
      key: "value_mensuration_estimated",
      render: (text: number | undefined) => <span>{text ?? "NÃ£o informado"}</span>,
    },
    {
      title: "AÃ§Ãµes",
      key: "actions",
      render: (_: any, record: ProductionLine) => (
        <Actions
          onEdit={() => navigate(`/production-lines/edit/${record.id}`)}
          onDelete={async () => {
            if (record.id) {
              await removeProductionLine(Number(record.id));
            }
          }}
        />
      ),
```

```
    },
  ];

  return { columns, productionLines, loading };
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/sectors/components/SectorsEdit.tsx
 =====
import React, { useEffect, useState } from "react";
import { useNavigate, useParams } from "react-router-dom";
import { Button, Input, Row, Col, message, Form } from "antd";
import { PlusOutlined } from "@ant-design/icons";
import ItemSideBar from "../../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../../layout/Header/components/ItemHeaderCabecalho";
import ProductionLinesTransfer from "../../../components/productionLinesTransfer/Production
LinesTransfer";
import { useSectorsStore } from "../../../store/sectors";
import { useProductionLinesStore } from "../../../store/ProductionLinesStore";
import { getSectorById, updateSector } from "../../../services/SectorsService";
import { updateProductionLine } from "../../../services/ProductionLinesService";
import { Sector } from "../../../types/sectors";

const SectorsEdit: React.FC = () => {
  const navigate = useNavigate();
  const { id } = useParams<{ id: string }>();
  const { fetchSectors } = useSectorsStore();
  const { productionLines, fetchProductionLines } = useProductionLinesStore();

  const [loading, setLoading] = useState(false);
  const [form] = Form.useForm();
  const [selectedLines, setSelectedLines] = useState<number[]>([]);
  const [estimatedConsumption, setEstimatedConsumption] = useState<number>(0);

  useEffect(() => {
    const fetchData = async () => {
      try {
        if (id) {
          const sectorData = await getSectorById(Number(id));
          form.setFieldsValue({
            name: sectorData.name,
            description: sectorData.description,
            estimated_consumption: sectorData.estimated_consumption,
          });

          const associatedLines = productionLines
            .filter((line) => line.setor === Number(id))
            .map((line) => line.id);

          setSelectedLines(associatedLines);
          setEstimatedConsumption(sectorData.estimated_consumption);
        }
      } catch (error) {
        message.error("Erro ao carregar os dados do setor!");
        console.error(error);
      }
    };

    fetchData();
    fetchProductionLines();
  }, [id]);

  useEffect(() => {
    const totalConsumption = selectedLines.reduce((sum, lineId) => {
      const line = productionLines.find((l) => l.id === lineId);
      return sum + (line?.value_mensuration_estimated ?? 0);
    }, 0);

    setEstimatedConsumption(totalConsumption);
    form.setFieldsValue({ estimated_consumption: totalConsumption });
```

```
}, [selectedLines, productionLines, form]);

const handleSubmit = async (values: Omit<Sector, "id" | "created_at">) => {
  try {
    setLoading(true);

    await updateSector(Number(id), {
      ...values,
      estimated_consumption: estimatedConsumption,
    });

    await Promise.all(
      selectedLines.map(async (lineId) => {
        const line = productionLines.find((l) => l.id === lineId);
        if (line) {
          await updateProductionLine(lineId, {
            ...line,
            setor: Number(id),
          });
        }
      })
    );

    message.success("Setor atualizado com sucesso!");

    await fetchSectors();
    await fetchProductionLines();
    navigate("/sectors");
  } catch (error) {
    console.error("Erro ao atualizar setor:", error);
    message.error("Erro ao atualizar setor!");
  } finally {
    setLoading(false);
  }
};

return (
  <div className="layout-container">
    <ItemSideBar />
    <div className="content-container">
      <ItemHeader />
      <main className="content">
        <ItemHeaderCabecalho
          title="Editar Setor"
          subTitle="Altere os dados do setor abaixo"
        />

        <Form form={form} layout="vertical" onFinish={handleSubmit}>
          <Row gutter={24}>
            <Col span={12}>
              <Form.Item
                name="name"
                label="Nome do Setor"
                rules={[{ required: true, message: "ObrigatÃ³rio" }]}
              >
                <Input placeholder="Digite nome do setor" />
              </Form.Item>
            </Col>
            <Col span={12}>
              <Form.Item name="description" label="DescriÃ§Ã£o">
                <Input.TextArea placeholder="Digite a descriÃ§Ã£o do setor" />
              </Form.Item>
            </Col>
          </Row>

          <Row gutter={24}>
            <Col span={12}>
              <Form.Item name="estimated_consumption" label="Consumo Estimado (kWh)">
                <Input value={estimatedConsumption} disabled />
```

```
              </Form.Item>
            </Col>
          </Row>

          <h3>Linhas de ProduÃ§Ã£o</h3>
          <ProductionLinesTransfer
            availableLines={productionLines}
            selectedKeys={selectedLines.map(String)}
            onChange={(keys) => setSelectedLines(keys.map(Number))}
          />

          <div className="button-group">
            <Button danger onClick={() => navigate("/sectors")}>
              Cancelar
            </Button>
            <Button type="primary" icon={<PlusOutlined />} htmlType="submit" loading={loa
ding}>
              Atualizar
            </Button>
          </div>
        </Form>
      </main>
    </div>
  </div>
 );
};

export default SectorsEdit;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/sectors/components/SectorsList.tsx
 =====
import React, { useEffect, useState } from "react";
import { Table, Button, message } from "antd";
import { EditOutlined, DeleteOutlined } from "@ant-design/icons";
import { useSectors } from "../../../contexts/sectors/SectorsContext";
import { useNavigate } from "react-router-dom";

const SectorsList: React.FC = () => {
  const { sectors, fetchSectors, removeSector } = useSectors(); // ð\237\224¹ Usando `remov
eSector` ao invÃ©s de `deleteSector`
  const navigate = useNavigate();
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const loadSectors = async () => {
      try {
        setLoading(true);
        await fetchSectors();
      } catch (error) {
        console.error("Erro ao buscar setores:", error);
        message.error("Erro ao carregar setores!");
      } finally {
        setLoading(false);
      }
    };

    loadSectors();
  }, []);

  // ConfiguraÃ§Ã£o das colunas da tabela
  const columns = [
    {
      title: "Nome",
      dataIndex: "name",
      key: "name",
      render: (text: string) => <strong>{text}</strong>,
    },
    {
      title: "DescriÃ§Ã£o",
      dataIndex: "description",
```

```
      key: "description",
      render: (text: string | null) => text || "NÃ£o informado",
    },
    {
      title: "Consumo Estimado",
      dataIndex: "estimated_consumption",
      key: "estimated_consumption",
      render: (value: number) => `${value} kWh`,
    },
  ];

  return (
    <Table
      dataSource={sectors}
      columns={columns}
      rowKey="id"
      loading={loading}
      pagination={{ pageSize: 5 }}
    />
  );
};

export default SectorsList;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/sectors/components/SectorsRegister
.tsx =====
import React, { useState, useEffect } from "react";
import { useNavigate } from "react-router-dom";
import { Button, Input, Row, Col, message, Form } from "antd";
import { PlusOutlined } from "@ant-design/icons";
import ItemSideBar from "../../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../../layout/Header/ItemHeader";
import ProductionLinesTransfer from "../../../components/productionLinesTransfer/Production
LinesTransfer";
import { useSectorsStore } from "../../../store/sectors";
import { useProductionLinesStore } from "../../../store/ProductionLinesStore";
import { Sector } from "../../../types/sectors";
import { updateProductionLine } from "../../../services/ProductionLinesService";

const SectorsRegister: React.FC = () => {
  const navigate = useNavigate();
  const { createSector } = useSectorsStore();
  const { productionLines, fetchProductionLines } = useProductionLinesStore();

  const [loading, setLoading] = useState(false);
  const [form] = Form.useForm();
  const [selectedLines, setSelectedLines] = useState<number[]>([]);
  const [estimatedConsumption, setEstimatedConsumption] = useState<number>(0);

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

  useEffect(() => {
    const totalConsumption = selectedLines.reduce((sum, lineId) => {
      const numericId = Number(lineId);
      const line = productionLines.find((l) => l.id === numericId);
      return sum + (line?.value_mensuration_estimated ?? 0);
    }, 0);

    setEstimatedConsumption(totalConsumption);
    form.setFieldsValue({ estimated_consumption: totalConsumption });
  }, [selectedLines, productionLines, form]);

  const handleSubmit = async (values: Omit<Sector, "id" | "created_at">) => {
    try {
      setLoading(true);

      const newSector = await createSector({
        ...values,
```

```
        estimated_consumption: estimatedConsumption,
      });

      if (!newSector) throw new Error("Erro ao criar setor.");

      await Promise.all(
        selectedLines.map(async (lineId) => {
          const line = productionLines.find((l) => l.id === lineId);
          if (line) {
            await updateProductionLine(lineId, {
              ...line,
              setor: newSector.id,
            });
          }
        })
      );

      message.success("Setor cadastrado com sucesso!");
      await fetchProductionLines();
      navigate("/sectors");
    } catch (error) {
      console.error("Erro ao cadastrar setor:", error);
      message.error("Erro ao cadastrar setor!");
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          <h1 className="title">Cadastro de Setor</h1>
          <p className="subtitle">Preencha os campos abaixo para cadastrar um setor</p>

          <Form form={form} layout="vertical" onFinish={handleSubmit}>
            <Row gutter={24}>
              <Col span={12}>
                <Form.Item name="name" label="Nome do Setor" rules={[{ required: true, mess
age: "ObrigatÃ³rio" }]}>
                  <Input placeholder="Digite nome do setor" />
                </Form.Item>
              </Col>
              <Col span={12}>
                <Form.Item name="description" label="DescriÃ§Ã£o">
                  <Input.TextArea placeholder="Digite a descriÃ§Ã£o do setor" />
                </Form.Item>
              </Col>
            </Row>

            <Row gutter={24}>
              <Col span={12}>
                <Form.Item name="estimated_consumption" label="Consumo Estimado (kWh)">
                  <Input value={estimatedConsumption} disabled />
                </Form.Item>
              </Col>
            </Row>

            <h3>Linhas de ProduÃ§Ã£o</h3>
            <ProductionLinesTransfer
              availableLines={productionLines.filter((line) => !line.setor)}
              selectedKeys={selectedLines.map(String)}
              onChange={(keys) => setSelectedLines(keys.map(Number))}
            />

            {/* â\234\205 Adicionando espaÃ§amento entre os botÃµes e o transfer */}
            <div style={{ marginTop: "20px", display: "flex", gap: "10px" }}>
```

```
              <Button danger onClick={() => navigate("/sectors")}>
                Cancelar
              </Button>
              <Button type="primary" icon={<PlusOutlined />} htmlType="submit" loading={loa
ding}>
                Enviar
              </Button>
            </div>
          </Form>
        </main>
      </div>
    </div>
  );
};

export default SectorsRegister;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/sectors/Sectors.tsx =====
import React from "react";
import { PlusOutlined, FilterOutlined } from "@ant-design/icons";
import { useNavigate } from "react-router-dom";
import ItemSideBar from "../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../layout/Header/ItemHeader";
import Button from "../../components/Button/Button";
import CustomTable from "../../components/Table/Table";
import ItemHeaderCabecalho from "../../layout/Header/components/ItemHeaderCabecalho";
import { useSectorsTable } from "./hooks/useSectorsTable"; // ð\237\224¹ Hook de setores

const Sectors: React.FC = () => {
  const navigate = useNavigate();
  const { columns, sectors, loading } = useSectorsTable();

  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          {/* CabeÃ§alho da pÃ¡gina */}
          <ItemHeaderCabecalho
            title="Setores"
            subTitle="Lista de setores jÃ¡ cadastrados"
          />

          {/* BotÃµes de aÃ§Ã£o */}
          <section className="actions-section">
            <Button
              type="primary"
              className="primary-btn"
              icon={<PlusOutlined />}
              onClick={() => navigate("/sectors/register")}
            >
              Cadastrar Setor
            </Button>
            <Button type="link" className="filter-btn" icon={<FilterOutlined />}>
              Filtros
            </Button>
          </section>

          {/* Tabela utilizando o hook de Setores */}
          <section className="table-container">
            <CustomTable columns={columns} data={sectors} loading={loading} />
          </section>
        </main>
      </div>
    </div>
  );
};

export default Sectors;
```

```
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/sectors/hooks/useSectorsTable.tsx
=====
import { useEffect, useState } from "react";
import { message } from "antd";
import { useNavigate } from "react-router-dom";
import { SortOrder } from "antd/es/table/interface";
import { useSectorsStore } from "../../../store/sectors";
import Actions from "../../../components/actions/Actions";
import { Sector } from "../../../types/sectors";

export const useSectorsTable = () => {
  const navigate = useNavigate();
  const { sectors, fetchSectors, deleteSector } = useSectorsStore();
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchSectorsFromAPI = async () => {
      try {
        setLoading(true);
        await fetchSectors();
      } catch (error) {
        console.error("Erro ao buscar setores:", error);
        message.error("Erro ao carregar setores!");
      } finally {
        setLoading(false);
      }
    };

    fetchSectorsFromAPI();
  }, []);

  // DefiniÃ§Ã£o das colunas da tabela
  const columns = [
    {
      title: "Nome",
      dataIndex: "name",
      key: "name",
      sorter: (a: Sector, b: Sector) => a.name.localeCompare(b.name),
      sortDirections: ["ascend", "descend"] as SortOrder[],
      render: (text: string | undefined) => <strong>{text ?? "Sem nome"}</strong>,
    },
    {
      title: "DescriÃ§Ã£o",
      dataIndex: "description",
      key: "description",
      render: (text: string | undefined) => <span>{text ?? "Sem descriÃ§Ã£o"}</span>,
    },
    {
      title: "Consumo Estimado",
      dataIndex: "estimated_consumption",
      key: "estimated_consumption",
      render: (text: number | undefined) => <span>{text ? `${text} kWh` : "0 kWh"}</span>,
    },
    {
      title: "AÃ§Ãµes",
      key: "actions",
      render: (_: any, record: Sector) => (
        <Actions
          onEdit={() => navigate(`/sectors/edit/${record.id}`)} // â\234\205 Direciona para
 ediÃ§Ã£o do setor
          onDelete={async () => {
            if (record.id) {
              await deleteSector(Number(record.id));
              await fetchSectors(); // â\234\205 Atualiza a tabela apÃ³s a exclusÃ£o
            }
          }}
        />
      ),
    },
```

```
  ];

  return { columns, sectors, loading };
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/login/Login.tsx =====
// src/pages/auth/Login.tsx

import { useState } from "react";
import { useNavigate } from "react-router-dom";
import { Input, Button, notification } from "antd";
import { Eye, EyeClosed, User } from "lucide-react";
import { useAuth } from "../../contexts/auth/AuthContext";
import api from "../../services/api";

import "primeflex/primeflex.css";

import BackgroundImage from "../../assets/background.jpg";
import ImageLogoLogin from "../../assets/logo-footer1.png";
import EmbrapiLogo from "../../assets/embrapi_logo.png";
import InovaLogo from "../../assets/inova_logo.png";
import ImageLogin from "../../assets/OperadorNansenRedimensionada.jpg";

interface LoginDTO {
  username: string;
  password: string;
}

export type NotificationType = "success" | "info" | "warning" | "error";

export default function Login() {
  const { login } = useAuth();
  const navigate = useNavigate();
  const [loginObject, setLoginObject] = useState<LoginDTO>({
    username: "",
    password: "",
  });

  const [apiNotification, contextHolder] = notification.useNotification();

  const openNotificationWithIcon = (
    type: NotificationType,
    message: string,
    description: string
  ) => {
    apiNotification[type]({ message, description });
  };

  const loginMethod = async () => {
    // 1) validaÃ§Ã£o de campos
    if (!loginObject.username || !loginObject.password) {
      openNotificationWithIcon(
        "warning",
        "Campos vazios",
        "Preencha todos os campos!"
      );
      return;
    }

    try {
      // 2) chamada ao back
      const response = await api.post("/login/", loginObject);

      // 3) back pode retornar 200 ou 201
      if (response.status === 200 || response.status === 201) {
        login(response.data.access);
        // login() jÃ¡ faz navigate("/home")
      } else {
        openNotificationWithIcon(
          "error",
```

```
          "Erro de login",
          "UsuÃ¡rio ou senha invÃ¡lidos."
        );
      }
    } catch (err: any) {
      // 4) tratamento de erro
      if (err.response) {
        const status = err.response.status;
        if (status === 400 || status === 401) {
          openNotificationWithIcon(
            "error",
            "Credenciais invÃ¡lidas",
            "UsuÃ¡rio ou senha incorretos."
          );
        } else {
          openNotificationWithIcon(
            "error",
            "Erro de login",
            err.response.data?.detail || "Algo deu errado."
          );
        }
      } else {
        openNotificationWithIcon(
          "error",
          "Erro de conexÃ£o",
          "NÃ£o foi possÃvel conectar Ã  API."
        );
      }
    }
  };

  return (
    <div
      className="flex items-center justify-center min-h-screen bg-cover bg-center px-4"
      style={{ backgroundImage: `url(${BackgroundImage})` }}
    >
      {contextHolder}

      <div className="w-full max-w-[850px] bg-white rounded-lg shadow-lg flex flex-col md:f
lex-row align-items-stretch overflow-hidden h-auto md:h-[500px]">
        {/* Lado Esquerdo â\200\223 FormulÃ¡rio */}
        <div className="w-full md:w-1/2 p-6 flex flex-col items-center text-center justify-
center max-w-md mx-auto">
          <img src={ImageLogoLogin} alt="Logo" className="mb-4 w-10" />
          <h1 className="text-xl font-semibold text-gray-800 mb-4">
            Acesse sua conta
          </h1>

          {/* Form captura Enter */}
          <form
            className="flex flex-col items-center"
            onSubmit={(e) => {
              e.preventDefault();
              loginMethod();
            }}
          >
            <Input
              placeholder="Nome de usuÃ¡rio"
              prefix={<User color="#4892D7" size={20} />}
              className="w-full max-w-[300px] h-12 text-lg border border-gray-300 rounded-m
d px-4 py-2 focus:ring-2 focus:ring-blue-500 focus:border-blue-500"
              value={loginObject.username}
              onChange={(e) =>
                setLoginObject({ ...loginObject, username: e.target.value })
              }
            />
            <Input.Password
              placeholder="Senha"
              className="w-full max-w-[300px] h-12 text-lg border border-gray-300 rounded-m
```

```
d px-4 py-2 mt-3 focus:ring-2 focus:ring-blue-500 focus:border-blue-500"
              value={loginObject.password}
              onChange={(e) =>
                setLoginObject({ ...loginObject, password: e.target.value })
              }
              iconRender={(visible) =>
                visible ? <Eye size={20} /> : <EyeClosed size={20} />
              }
            />
            <Button
              type="primary"
              htmlType="submit"
              block
              className="w-full max-w-[300px] h-12 text-lg mt-4 bg-[#0057B8] hover:bg-[#004
A99] text-white font-semibold rounded-md transition-all"
            >
              Entrar
            </Button>
          </form>

          {/* Logos inferiores */}
          <div className="flex justify-center items-center mt-6 space-x-6">
            <img src={InovaLogo} alt="Inova" className="w-20 h-auto" />
            <img src={EmbrapiLogo} alt="Embrapi" className="w-24 h-auto" />
          </div>
        </div>

        {/* Lado Direito â\200\223 Imagem */}
        <div className="hidden md:flex w-full md:w-1/2 h-auto md:h-[500px] relative">
          <img
            src={ImageLogin}
            alt="Operador"
            className="w-full h-full object-cover"
          />
          <div className="absolute inset-0 bg-[#112A42] opacity-50"></div>
        </div>
      </div>
    </div>
  );
}

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/loja/LojaProducts.tsx =====
import React from "react";
import { PlusOutlined } from "@ant-design/icons";
import { useNavigate } from "react-router-dom";
import ItemSideBar from "../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../layout/Header/ItemHeader";
import Button from "../../components/Button/Button";
import CustomTable from "../../components/Table/Table";
import ItemHeaderCabecalho from "../../layout/Header/components/ItemHeaderCabecalho";
import { useLojaProductsTable } from "./hooks/useLojaProductsTable";

const LojaProductsPage: React.FC = () => {
  const navigate = useNavigate();
  const { columns, products, loading } = useLojaProductsTable();

  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          <ItemHeaderCabecalho
            title="Produtos Loja"
            subTitle="Listagem dos produtos da loja"
          />
          <Button
            icon={<PlusOutlined />}
            onClick={() => navigate("/loja/register")}
```

```
          >
            Novo Produto
          </Button>
          <CustomTable columns={columns} data={products} loading={loading} />
        </main>
      </div>
    </div>
  );
};

export default LojaProductsPage;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/loja/LojaProductsRegister.tsx ====
=
import React, { useState } from "react";
import { message } from "antd";
import { useNavigate } from "react-router-dom";
import ItemSideBar from "../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../layout/Header/components/ItemHeaderCabecalho";
import DynamicForm from "../../components/form/DynamicForm";
import { useLojaStore } from "../../store/useLojaStore";

const LojaProductsRegister: React.FC = () => {
  const navigate = useNavigate();
  const { createProduct } = useLojaStore();
  const [loading, setLoading] = useState(false);
  const [formValues, setFormValues] = useState<{
    name: string;
    description: string;
    price: number;
    quantity: number;
    image: File | null;
  }>({
    name: "",
    description: "",
    price: 0,
    quantity: 0,
    image: null,
  });

  const handleChange = (name: string, value: any) => {
    setFormValues((prev) => ({ ...prev, [name]: value }));
  };

  const handleSubmit = async () => {
    if (!formValues.name.trim() || !formValues.description.trim()) {
      message.error("Campos obrigatÃ³rios precisam ser preenchidos!");
      return;
    }
    setLoading(true);
    try {
      const formData = new FormData();
      Object.entries(formValues).forEach(([key, val]) => {
        if (val !== null)
          formData.append(key, val instanceof File ? val : String(val));
      });
      await createProduct(formData);
      message.success("Produto cadastrado!");
      navigate("/loja");
    } catch (error) {
      message.error("Erro ao cadastrar produto!");
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="layout-container">
```

```
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          <ItemHeaderCabecalho
            title="Cadastro de Produto"
            subTitle="Cadastre um novo produto"
          />
          <DynamicForm
            fields={[
              { name: "name", label: "Nome", type: "input", required: true },
              {
                name: "description",
                label: "DescriÃ§Ã£o",
                type: "textarea",
                required: true,
              },
              { name: "price", label: "PreÃ§o", type: "number", required: true },
              {
                name: "quantity",
                label: "Quantidade",
                type: "number",
                required: true,
              },
              { name: "image", label: "Imagem", type: "upload" },
            ]}
            values={formValues}
            onChange={handleChange}
            onSubmit={handleSubmit}
            loading={loading}
            onCancel={() => navigate("/loja")}
          />
        </main>
      </div>
    </div>
  );
};

export default LojaProductsRegister;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/loja/hooks/useLojaProductsTable.ts
x =====
import { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
import Actions from "../../../components/actions/Actions";
import { useLojaStore } from "../../../store/useLojaStore";

export const useLojaProductsTable = () => {
  const navigate = useNavigate();
  const { products, fetchProducts, deleteProduct } = useLojaStore();
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetchProducts().finally(() => setLoading(false));
  }, []);

  const columns = [
    {
      title: "Imagem",
      dataIndex: "image",
      key: "image",
      render: (image: string | null) =>
        image ? (
          <img
            src={
              image.startsWith("http")
                ? image
                : `http://inova-sistemas.ddns.net:20163${image}`
            }
```

```
            alt="Produto"
            style={{
              width: "50px",
              height: "50px",
              objectFit: "cover",
              borderRadius: "5px",
            }}
          />
        ) : (
          <span>Sem imagem</span>
        ),
    },
    {
      title: "Nome",
      dataIndex: "name",
      key: "name",
    },
    {
      title: "PreÃ§o",
      dataIndex: "price",
      key: "price",
    },
    {
      title: "Quantidade",
      dataIndex: "quantity",
      key: "quantity",
    },
    {
      title: "AÃ§Ãµes",
      key: "actions",
      render: (_: any, record: any) => (
        <Actions
          onEdit={() => navigate(`/loja/edit/${record.id}`)}
          onDelete={() => deleteProduct(record.id)}
        />
      ),
    },
  ];

  return { columns, products, loading };
};

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/loja/LojaProductsEdit.tsx =====
import React, { useEffect, useState } from "react";
import { useNavigate, useParams } from "react-router-dom";
import { message } from "antd";
import ItemSideBar from "../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../layout/Header/components/ItemHeaderCabecalho";
import DynamicForm from "../../components/form/DynamicForm";
import { useLojaStore } from "../../store/useLojaStore";

const LojaProductsEdit: React.FC = () => {
  const navigate = useNavigate();
  const { id } = useParams<{ id: string }>();
  const { fetchProductById, editProduct } = useLojaStore();
  const [loading, setLoading] = useState(false);
  const [formValues, setFormValues] = useState<{
    name: string;
    description: string;
    price: number;
    quantity: number;
    image: File | null;
  }>({
    name: "",
    description: "",
    price: 0,
    quantity: 0,
    image: null,
```

```
});

useEffect(() => {
  const loadProduct = async () => {
    setLoading(true);
    try {
      const data = await fetchProductById(Number(id));
      setFormValues({
        name: data.name,
        description: data.description,
        price: data.price,
        quantity: data.quantity,
        image: null, // Corrigido: nÃ£o atribuir diretamente a string
      });
    } catch (error) {
      message.error("Erro ao carregar produto!");
    } finally {
      setLoading(false);
    }
  };
  if (id) loadProduct();
}, [id, fetchProductById]);

const handleChange = (name: string, value: any) => {
  setFormValues((prev) => ({ ...prev, [name]: value }));
};

const handleSubmit = async () => {
  if (!formValues.name.trim()) {
    message.error("O nome do produto Ã© obrigatÃ³rio!");
    return;
  }
  setLoading(true);
  try {
    const formData = new FormData();
    Object.entries(formValues).forEach(([key, val]) => {
      if (val !== null)
        formData.append(key, val instanceof File ? val : String(val));
    });
    await editProduct(Number(id), formData);
    message.success("Produto atualizado!");
    navigate("/loja");
  } catch (error) {
    message.error("Erro ao atualizar produto!");
  } finally {
    setLoading(false);
  }
};

return (
  <div className="layout-container">
    <ItemSideBar />
    <div className="content-container">
      <ItemHeader />
      <main className="content">
        <ItemHeaderCabecalho
          title="Editar Produto"
          subTitle="Edite os dados do produto"
        />
        <DynamicForm
          fields={[
            { name: "name", label: "Nome", type: "input", required: true },
            {
              name: "description",
              label: "DescriÃ§Ã£o",
              type: "textarea",
              required: true,
            },
            { name: "price", label: "PreÃ§o", type: "number", required: true },
```

```
              {
                name: "quantity",
                label: "Quantidade",
                type: "number",
                required: true,
              },
              { name: "image", label: "Imagem", type: "upload" },
            ]}
            values={formValues}
            onChange={handleChange}
            onSubmit={handleSubmit}
            loading={loading}
            onCancel={() => navigate("/loja")}
          />
        </main>
      </div>
    </div>
  );
};

export default LojaProductsEdit;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/home/Home.tsx =====
// src/pages/home/HomePage.tsx
import React, { useEffect, useState } from "react";
import { Layout, Row, Col, Spin } from "antd";
import {
  DashboardOutlined,
  PartitionOutlined,
  HddOutlined,
  NodeExpandOutlined,
} from "@ant-design/icons";
import ItemHeader from "../../layout/Header/ItemHeader";
import ItemSideBar from "../../layout/Sidebar/ItemSideBar";
import { useMonitoringStore } from "@/store/monitoringStore";

import DashboardMetrics from "./components/dashboard/DashboardMetrics";
import MonitoringSelector from "./components/dashboard/MonitoringSelector";
import EnergyTrendCard from "./components/dashboard/EnergyTrendCard";
import BarBySectionCard from "./components/dashboard/BarBySectionCard";
import EnvironmentalBreakdownCard from "./components/dashboard/EnvironmentalBreakdownCard";
import TotalConsumptionOverview from "./components/dashboard/TotalConsumptionOverview";

import { useDashboardMetrics } from "./hooks/useDashboardMetrics";
import { useEnergyTrend } from "./hooks/useEnergyTrend";
import { useSectionLoads } from "./hooks/useSectionLoads";
import { useEnvironmentalData } from "./hooks/useEnvironmentalData";
import { useTotalConsumption } from "./hooks/useTotalConsumption";

const { Content } = Layout;

const HomePage: React.FC = () => {
  // selector for Monitoramentos dropdown
  const [selectedMonitoringId, setSelectedMonitoringId] = useState<
    number | null
  >(null);
  const { monitorings, fetchMonitorings } = useMonitoringStore();

  // fetch monitorings once
  useEffect(() => {
    fetchMonitorings();
  }, [fetchMonitorings]);

  // top metrics hook
  const {
    activeMonitoringCount,
    totalSectionsCount,
    totalSectorsCount,
    totalDevicesCount,
```

```
} = useDashboardMetrics();

// data hooks
const { data: trendData, loading: trendLoading } = useEnergyTrend(
  selectedMonitoringId ?? undefined
);
const { data: barData, loading: barLoading } = useSectionLoads(
  selectedMonitoringId ?? undefined
);
const { data: envData, loading: envLoading } = useEnvironmentalData(
  selectedMonitoringId ?? undefined
);
const { data: totalData, loading: totalLoading } = useTotalConsumption(
  selectedMonitoringId ?? undefined
);

// assemble metrics items
const metricsItems = [
  {
    icon: <DashboardOutlined style={{ fontSize: 16, color: "#004281" }} />,
    title: "Monitoramentos Ativos",
    value: activeMonitoringCount,
  },
  {
    icon: <PartitionOutlined style={{ fontSize: 16, color: "#004281" }} />,
    title: "SeÃ§Ãµes Ativas",
    value: totalSectionsCount,
  },
  {
    icon: <HddOutlined style={{ fontSize: 16, color: "#004281" }} />,
    title: "Setores Monitorados",
    value: totalSectorsCount,
  },
  {
    icon: <NodeExpandOutlined style={{ fontSize: 16, color: "#004281" }} />,
    title: "Dispositivos Monitorados",
    value: totalDevicesCount,
  },
];

const loadingAny = trendLoading || barLoading || envLoading || totalLoading;

return (
  <Layout style={{ height: "100vh", overflow: "hidden" }}>
    <ItemSideBar />
    <Layout>
      <ItemHeader />
      <Content style={{ padding: 8, height: "100%" }}>
        {/* mÃ©tricas + selector */}
        <Row gutter={8} wrap={false} align="middle">
          <Col flex="auto">
            <DashboardMetrics items={metricsItems} />
          </Col>
          <Col flex="0 0 240px">
            <MonitoringSelector
              value={selectedMonitoringId}
              onChange={setSelectedMonitoringId}
            />
          </Col>
        </Row>

        {loadingAny ? (
          <div style={{ textAlign: "center", marginTop: 50 }}>
            <Spin size="large" />
          </div>
        ) : (
          <>
            {/* 2Ã\2272 mini-grÃ¡ficos */}
            <Row gutter={[8, 8]} style={{ marginTop: 12 }}>
```

```
                <Col span={12}>
                  <EnergyTrendCard
                    title={`TendÃªncia de Energia â\200\224 ${
                      selectedMonitoringId ?? "Geral"
                    }`}
                    data={trendData}
                  />
                </Col>
                <Col span={12}>
                  <EnergyTrendCard
                    title="TendÃªncia de Energia â\200\224 Geral"
                    data={trendData}
                    stroke="#82ca9d"
                  />
                </Col>
                <Col span={12}>
                  <BarBySectionCard title="Carga por SeÃ§Ã£o" data={barData} />
                </Col>
                <Col span={12}>
                  <EnvironmentalBreakdownCard
                    title="DistribuiÃ§Ã£o Ambiental"
                    data={envData}
                    colors={["#0088FE", "#00C49F", "#FFBB28"]}
                  />
                </Col>
              </Row>

              {/* visÃ£o geral full-width */}
              <Row style={{ marginTop: 12 }}>
                <Col span={24}>
                  <TotalConsumptionOverview
                    data={totalData.map((d) => ({
                      name: `${d.time}m`,
                      value: d.value,
                    }))}
                  />
                </Col>
              </Row>
            </>
          )}
        </Content>
      </Layout>
    </Layout>
  );
};

export default HomePage;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/home/components/dashboard/Dashboar
dMetrics.tsx =====
// src/pages/home/components/dashboard/DashboardMetrics.tsx
import React from "react";
import { Row, Col } from "antd";
import MetricCard, { MetricCardProps } from "./MetricCard";

export interface DashboardMetricsProps {
  items: MetricCardProps[];
}

const DashboardMetrics: React.FC<DashboardMetricsProps> = ({ items }) => (
  <Row gutter={8} wrap={false} align="middle">
    {items.map((item, idx) => (
      <Col flex="1 1 0" key={idx}>
        <MetricCard icon={item.icon} title={item.title} value={item.value} />
      </Col>
    ))}
  </Row>
);
```

```
export default DashboardMetrics;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/home/components/dashboard/Monitori
ngSelector.tsx =====
// src/pages/home/components/dashboard/MonitoringSelector.tsx
import React, { useEffect } from "react";
import { Card, Select } from "antd";
import { useMonitoringStore } from "@/store/monitoringStore";
import { MonitoringItem } from "@/types/monitoringTypes";

const { Option } = Select;

export interface MonitoringSelectorProps {
  value?: number | null;
  onChange: (value: number | null) => void;
}

const MonitoringSelector: React.FC<MonitoringSelectorProps> = ({
  value,
  onChange,
}) => {
  const { monitorings, fetchMonitorings } = useMonitoringStore();

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

  return (
    <Card
      size="small"
      title="Selecionar Monitoramento"
      bodyStyle={{ padding: 4 }}
    >
      <Select
        value={value ?? undefined}
        onChange={(v: number) => onChange(v)}
        placeholder="Selecione o monitoramento"
        style={{ width: "100%" }}
        size="small"
      >
        {monitorings.map((m: MonitoringItem) => (
          <Option key={m.id} value={m.id}>
            {m.name}
          </Option>
        ))}
      </Select>
    </Card>
  );
};

export default MonitoringSelector;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/home/components/dashboard/TotalCon
sumptionOverview.tsx =====
// src/pages/home/components/dashboard/TotalConsumptionOverview.tsx
import React from "react";
import { Card, Spin } from "antd";
import MonitoringChart from "./MonitoringChart";

export interface TotalConsumptionOverviewProps {
  data: { name: string; value: number }[];
  stroke?: string;
  loading?: boolean;
}

const TotalConsumptionOverview: React.FC<TotalConsumptionOverviewProps> = ({
  data,
  stroke = "#ffc658",
  loading = false,
```

```
}) => {
  const title = "VisÃ£o Geral do Consumo Total (kW ao longo do tempo)";

  if (loading) {
    return (
      <Card
        size="small"
        title={title}
        bodyStyle={{ padding: 24, textAlign: "center" }}
      >
        <Spin />
      </Card>
    );
  }

  if (!data || data.length === 0) {
    return (
      <Card
        size="small"
        title={title}
        bodyStyle={{ padding: 16, textAlign: "center" }}
      >
        Sem dados disponÃ-veis
      </Card>
    );
  }

  return <MonitoringChart title={title} data={data} stroke={stroke} />;
};

export default TotalConsumptionOverview;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/home/components/dashboard/Environm
entalBreakdownCard.tsx =====
// src/pages/home/components/dashboard/EnvironmentalBreakdownCard.tsx
import React from "react";
import { Card, Spin } from "antd";
import { ResponsiveContainer, PieChart, Pie, Cell, Tooltip } from "recharts";

export interface EnvironmentalBreakdownCardProps {
  title: string;
  data: { name: string; value: number }[];
  colors: string[];
  loading?: boolean;
}

const EnvironmentalBreakdownCard: React.FC<EnvironmentalBreakdownCardProps> = ({
  title,
  data,
  colors,
  loading = false,
}) => {
  if (loading) {
    return (
      <Card
        size="small"
        title={title}
        bodyStyle={{ padding: 24, textAlign: "center" }}
      >
        <Spin />
      </Card>
    );
  }

  if (!data || data.length === 0) {
    return (
      <Card
        size="small"
        title={title}
```

```
        bodyStyle={{ padding: 16, textAlign: "center" }}
      >
        Sem dados disponÃ-veis
      </Card>
    );
  }

  return (
    <Card size="small" title={title} bodyStyle={{ padding: 4 }}>
      <ResponsiveContainer width="100%" height={100}>
        <PieChart>
          <Pie
            data={data}
            dataKey="value"
            nameKey="name"
            cx="50%"
            cy="50%"
            innerRadius={15}
            outerRadius={40}
            label={false}
          >
            {data.map((_, idx) => (
              <Cell key={idx} fill={colors[idx % colors.length]} />
            ))}
          </Pie>
          <Tooltip formatter={(value: number) => `${value}`} />
        </PieChart>
      </ResponsiveContainer>
    </Card>
  );
};

export default EnvironmentalBreakdownCard;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/home/components/dashboard/EnergyTr
endCard.tsx =====
// src/pages/home/components/dashboard/EnergyTrendCard.tsx
import React from "react";
import { Card, Spin } from "antd";
import {
  ResponsiveContainer,
  AreaChart,
  XAxis,
  YAxis,
  Tooltip,
  Area,
} from "recharts";

export interface EnergyTrendCardProps {
  title: string;
  data: { time: number; kW: number }[];
  stroke?: string;
  loading?: boolean;
}

const EnergyTrendCard: React.FC<EnergyTrendCardProps> = ({
  title,
  data,
  stroke = "#8884d8",
  loading = false,
}) => {
  if (loading) {
    return (
      <Card
        size="small"
        title={title}
        bodyStyle={{ padding: 24, textAlign: "center" }}
      >
        <Spin />
```

```
      </Card>
    );
  }

  if (!data || data.length === 0) {
    return (
      <Card
        size="small"
        title={title}
        bodyStyle={{ padding: 16, textAlign: "center" }}
      >
        Sem dados disponÃ-veis
      </Card>
    );
  }

  const fillColor = `${stroke}aa`;

  return (
    <Card size="small" title={title} bodyStyle={{ padding: 4 }}>
      <ResponsiveContainer width="100%" height={100}>
        <AreaChart data={data}>
          <XAxis
            dataKey="time"
            tickFormatter={(t) => `${t}m`}
            tick={{ fontSize: 10 }}
          />
          <YAxis unit="kW" tick={{ fontSize: 10 }} />
          <Tooltip formatter={(value: number) => `${value} kW`} />
          <Area type="monotone" dataKey="kW" stroke={stroke} fill={fillColor} />
        </AreaChart>
      </ResponsiveContainer>
    </Card>
  );
};

export default EnergyTrendCard;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/home/components/dashboard/Monitori
ngChart.tsx =====
import React from "react";
import { Card } from "antd";
import {
  LineChart,
  Line,
  XAxis,
  YAxis,
  CartesianGrid,
  Tooltip,
  Legend,
  ResponsiveContainer,
} from "recharts";

interface Props {
  title: string;
  data: { name: string; value: number }[];
  stroke?: string;
}

const MonitoringChart: React.FC<Props> = ({
  title,
  data,
  stroke = "#8884d8",
}) => {
  return (
    <Card title={title} style={{ marginBottom: "20px" }}>
      <ResponsiveContainer width="100%" height={200}>
        <LineChart data={data}>
          <CartesianGrid strokeDasharray="3 3" />
```

```
          <XAxis dataKey="name" />
          <YAxis />
          <Tooltip />
          <Legend />
          <Line type="monotone" dataKey="value" stroke={stroke} />
        </LineChart>
      </ResponsiveContainer>
    </Card>
  );
};

export default MonitoringChart;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/home/components/dashboard/BarBySec
tionCard.tsx =====
// src/pages/home/components/dashboard/BarBySectionCard.tsx
import React from "react";
import { Card, Spin } from "antd";
import {
  ResponsiveContainer,
  BarChart,
  Bar,
  XAxis,
  YAxis,
  Tooltip,
} from "recharts";

export interface BarBySectionCardProps {
  title: string;
  data: { section: string; load: number }[];
  loading?: boolean;
}

const BarBySectionCard: React.FC<BarBySectionCardProps> = ({
  title,
  data,
  loading = false,
}) => {
  if (loading) {
    return (
      <Card
        size="small"
        title={title}
        bodyStyle={{ padding: 24, textAlign: "center" }}
      >
        <Spin />
      </Card>
    );
  }

  if (!data || data.length === 0) {
    return (
      <Card
        size="small"
        title={title}
        bodyStyle={{ padding: 16, textAlign: "center" }}
      >
        Sem dados disponÃ-veis
      </Card>
    );
  }

  return (
    <Card size="small" title={title} bodyStyle={{ padding: 4 }}>
      <ResponsiveContainer width="100%" height={100}>
        <BarChart data={data}>
          <XAxis dataKey="section" tick={{ fontSize: 10 }} />
          <YAxis hide unit="%" />
          <Tooltip formatter={(value: number) => `${value}%`} />
```

```
          <Bar dataKey="load" barSize={12} fill="#8884d8" />
        </BarChart>
      </ResponsiveContainer>
    </Card>
  );
};

export default BarBySectionCard;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/home/components/dashboard/MetricCa
rd.tsx =====
// src/pages/home/components/dashboard/MetricCard.tsx
import React from "react";
import { Card } from "antd";

export interface MetricCardProps {
  icon: React.ReactNode;
  title: string;
  value: number | string;
}

const MetricCard: React.FC<MetricCardProps> = ({ icon, title, value }) => (
  <Card className="card_style">
    <div className="card_content_style">
      <div>
        {icon}
        <h1 className="card_content_title">{title}</h1>
      </div>
      <span className="card_conten_value">{value}</span>
    </div>
  </Card>
);

export default MetricCard;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/home/hooks/useSectionsEnergyData.t
s =====
import { useEffect, useState, useMemo } from "react";
import api from "@/services/api";

export interface EnergyMeasurement {
  id: number;
  energia_ativa_kWh: number;
  interval: number;
  section: number;
}

export function useSectionsEnergyData(sectionIds: number[]) {
  // memoiza o array de IDs para nÃ£o disparar efeito sem necessidade
  const memoIds = useMemo(() => [...sectionIds], [sectionIds.join(",")]);

  const [dataMap, setDataMap] = useState<Record<number, EnergyMeasurement[]>>(
    {}
  );
  const [loading, setLoading] = useState(false);

  useEffect(() => {
    if (memoIds.length === 0) {
      setDataMap({});
      return;
    }
    let mounted = true;
    setLoading(true);

    Promise.all(
      memoIds.map((id) =>
        api
          .get<EnergyMeasurement[]>(`/section-measurements/?section_id=${id}`)
          .then((res) => {
```

```
            // limpa leituras invÃ¡lidas e pega sÃ³ Ãºltimas 60
            const valid = res.data.filter(
              (m) =>
                typeof m.energia_ativa_kWh === "number" &&
                !Number.isNaN(m.energia_ativa_kWh)
            );
            const slice = valid.length > 60 ? valid.slice(-60) : valid;
            return { id, data: slice };
          })
          .catch(() => ({ id, data: [] }))
      )
    )
      .then((results) => {
        if (!mounted) return;
        const map: Record<number, EnergyMeasurement[]> = {};
        results.forEach(({ id, data }) => {
          map[id] = data;
        });
        setDataMap(map);
      })
      .finally(() => {
        if (mounted) setLoading(false);
      });

    return () => {
      mounted = false;
    };
  }, [memoIds]);

  return { dataMap, loading };
}

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/home/hooks/useTotalConsumption.ts
=====
// src/pages/home/hooks/useTotalConsumption.ts
import { useEffect, useState } from "react";
import { useMonitoringSections } from "./useMonitoringSections";
import { getSectionMeasurements } from "@/services/SectionsService";

export interface TotalPoint {
  time: number;
  value: number; // kW
}

export function useTotalConsumption(monitoringId?: number) {
  const { monitoringSections, loading: sectionsLoading } =
    useMonitoringSections(monitoringId);
  const [data, setData] = useState<TotalPoint[]>([]);
  const [loading, setLoading] = useState(false);

  useEffect(() => {
    if (!monitoringId) {
      setData([]);
      return;
    }

    const fetchTotal = async () => {
      setLoading(true);
      try {
        // Pegar mediÃ§Ãµes de todas as seÃ§Ãµes
        const allMeasurements = await Promise.all(
          monitoringSections.map((sec) => getSectionMeasurements(sec.id))
        );
        // Achatar tudo
        const flat = allMeasurements.flat();
        // Agrupar por tempo, somando todos
        const groups: Record<number, number> = {};
        flat.forEach((m) => {
          const t = m.interval;
```

```
          groups[t] = (groups[t] || 0) + (m.energia_ativa_kWh || 0);
        });
        // Mapear para sÃ©rie
        const points: TotalPoint[] = Object.entries(groups).map(
          ([interval, sum]) => ({ time: Number(interval), value: sum })
        );
        // Ordenar
        points.sort((a, b) => a.time - b.time);
        setData(points);
      } catch (err) {
        console.error("Erro ao buscar consumo total:", err);
      } finally {
        setLoading(false);
      }
    };

    fetchTotal();
  }, [monitoringSections, monitoringId]);

  return { data, loading: loading || sectionsLoading };
}

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/home/hooks/useMonitoringSections.t
s =====
// src/pages/home/hooks/useMonitoringSections.ts
import { useEffect, useState } from "react";
import { useSectionStore } from "@/store/sectionStore";
import { SectionItem } from "@/types/sections";

/**
 * Hook que retorna as seÃ§Ãµes relacionadas a um monitoramento especÃfico.
 * Considera a hierarquia de setores â\206\222 linhas â\206\222 equipamentos.
 */
export function useMonitoringSections(monitoringId?: number) {
  const { sections, fetchSections, loading } = useSectionStore();
  const [filtered, setFiltered] = useState<SectionItem[]>([]);

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

  useEffect(() => {
    if (!monitoringId || loading) return;

    const filterByMonitoring = (sectionList: SectionItem[]): SectionItem[] => {
      return sectionList
        .filter((s) => s.monitoring === monitoringId)
        .map((s) => ({
          ...s,
          sections_filhas: filterByMonitoring(s.sections_filhas || []),
        }));
    };

    const result = filterByMonitoring(sections);
    setFiltered(result);
  }, [monitoringId, sections, loading]);

  return { monitoringSections: filtered, loading };
}

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/home/hooks/useEnergyTrend.ts =====
import { useEffect, useState } from "react";
import { useMonitoringSections } from "./useMonitoringSections";
import { getSectionMeasurements } from "@/services/SectionsService";

export interface TrendPoint {
  time: number;
  kW: number;
}
```

```
export function useEnergyTrend(monitoringId?: number) {
  const { monitoringSections, loading: sectionsLoading } =
    useMonitoringSections(monitoringId);
  const [data, setData] = useState<TrendPoint[]>([]);
  const [loading, setLoading] = useState(false);

  useEffect(() => {
    if (!monitoringId) {
      setData([]);
      return;
    }

    const fetchTrend = async () => {
      setLoading(true);
      try {
        // 1) busca mediÃ§Ãµes de cada seÃ§Ã£o
        const allArrays = await Promise.all(
          monitoringSections.map((sec) =>
            getSectionMeasurements(sec.id)
              .then((arr) =>
                // filtra e ordena as Ãºltimas 60 mediÃ§Ãµes
                arr
                  .filter(
                    (m) =>
                      typeof m.energia_ativa_kWh === "number" &&
                      !Number.isNaN(m.energia_ativa_kWh)
                  )
                  .slice(-60)
              )
              .catch(() => [])
          )
        );
        // 2) achata tudo num sÃ³ array
        const flat = allArrays.flat();
        // 3) agrupa por 'interval' e calcula mÃ©dia
        const groups: Record<number, number[]> = {};
        flat.forEach((m) => {
          const t = m.interval;
          groups[t] = groups[t] || [];
          groups[t].push(m.energia_ativa_kWh);
        });
        const points: TrendPoint[] = Object.entries(groups).map(
          ([interval, values]) => ({
            time: Number(interval),
            kW: values.reduce((sum, v) => sum + v, 0) / values.length,
          })
        );
        // 4) ordena por tempo
        points.sort((a, b) => a.time - b.time);
        setData(points);
      } catch (err) {
        console.error("Erro ao buscar tendÃªncia de energia:", err);
        setData([]);
      } finally {
        setLoading(false);
      }
    };

    fetchTrend();
  }, [monitoringSections, monitoringId]);

  return { data, loading: loading || sectionsLoading };
}

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/home/hooks/useEnergyData.ts =====
import { useEffect, useState } from "react";
import api from "@/services/api";
```

```
export interface EnergyMeasurement {
  id: number;
  energia_ativa_kWh: number;
  interval: number;
  section: number;
}

export const useEnergyData = (sectionId?: number) => {
  const [data, setData] = useState<EnergyMeasurement[]>([]);
  const [loading, setLoading] = useState(false);

  useEffect(() => {
    if (!sectionId) {
      setData([]);
      return;
    }

    const fetchData = async () => {
      setLoading(true);
      try {
        const response = await api.get<EnergyMeasurement[]>(
          `/section-measurements/?section_id=${sectionId}`
        );
        // filtra apenas leituras vÃ¡lidas (nÃ£o NaN) e mantÃ©m ordem original
        const clean = response.data.filter(
          (m) =>
            typeof m.energia_ativa_kWh === "number" &&
            !Number.isNaN(m.energia_ativa_kWh)
        );
        setData(clean);
      } catch (error) {
        console.error("Erro ao buscar dados de energia:", error);
        setData([]);
      } finally {
        setLoading(false);
      }
    };

    fetchData();
  }, [sectionId]);

  return { data, loading };
};

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/home/hooks/useSectionLoads.ts ====
=
// src/pages/home/hooks/useSectionLoads.ts
import { useEffect, useState } from "react";
import { useMonitoringSections } from "./useMonitoringSections";
import { getSectionMeasurements } from "@/services/SectionsService";
import { SectionItem } from "@/types/sections";

export interface SectionLoad {
  section: string;
  load: number;
}

export function useSectionLoads(monitoringId?: number) {
  const { monitoringSections, loading: sectionsLoading } =
    useMonitoringSections(monitoringId);
  const [data, setData] = useState<SectionLoad[]>([]);
  const [loading, setLoading] = useState(false);

  useEffect(() => {
    if (!monitoringId) {
      setData([]);
      return;
    }
```

```
    const fetchLoads = async () => {
      setLoading(true);
      try {
        const loads = await Promise.all(
          monitoringSections.map(async (section: SectionItem) => {
            const measurements = await getSectionMeasurements(section.id);
            const total = measurements.reduce(
              (sum, m) => sum + (m.energia_ativa_kWh || 0),
              0
            );
            return { section: section.name, load: total };
          })
        );
        setData(loads);
      } catch (error) {
        console.error("Erro ao buscar carga por seÃ§Ã£o:", error);
      } finally {
        setLoading(false);
      }
    };

    fetchLoads();
  }, [monitoringSections, monitoringId]);

  return { data, loading: loading || sectionsLoading };
}

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/home/hooks/useEnvironmentalData.ts
 =====
// src/pages/home/hooks/useEnvironmentalData.ts
import { useEffect, useState } from "react";

export interface EnvPoint {
  name: string;
  value: number;
}

export function useEnvironmentalData(monitoringId?: number) {
  const [data, setData] = useState<EnvPoint[]>([]);
  const [loading, setLoading] = useState(false);

  useEffect(() => {
    if (!monitoringId) {
      setData([]);
      return;
    }

    setLoading(true);
    // mock temporÃ¡rio
    const mock = [
      { name: "Temperatura", value: 22 },
      { name: "Umidade", value: 60 },
      { name: "Luminosidade", value: 350 },
    ];
    // simula atraso de fetch
    const timer = setTimeout(() => {
      setData(mock);
      setLoading(false);
    }, 500);

    return () => clearTimeout(timer);
  }, [monitoringId]);

  return { data, loading };
}

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/home/hooks/useDashboardMetrics.ts
=====
// src/pages/home/hooks/useDashboardMetrics.ts
```

```
import { useEffect } from "react";
import { useMonitoringStore } from "@/store/monitoringStore";
import { useSectionStore } from "@/store/sectionStore";
import { SectionItem } from "@/types/sections";

export interface DashboardMetrics {
  activeMonitoringCount: number;
  totalSectionsCount: number;
  totalSectorsCount: number;
  totalDevicesCount: number;
}

export function useDashboardMetrics(): DashboardMetrics {
  // Seletores do Zustand
  const activeMonitoringCount = useMonitoringStore((s) => s.activeCount);
  const fetchActiveCount = useMonitoringStore((s) => s.fetchActiveCount);
  const sections = useSectionStore((s) => s.sections);
  const fetchSections = useSectionStore((s) => s.fetchSections);

  // Carregar dados uma vez ao montar
  useEffect(() => {
    fetchSections();
    fetchActiveCount();
  }, [fetchSections, fetchActiveCount]);

  // Quantidade de setores (nÃvel raiz)
  const totalSectorsCount = sections.length;

  // Quantidade total de seÃ§Ãµes (inclui subseÃ§Ãµes recursivamente)
  const countAllSections = (list: SectionItem[]): number => {
    return list.reduce(
      (sum, s) => sum + 1 + countAllSections(s.sections_filhas || []),
      0
    );
  };
  const totalSectionsCount = countAllSections(sections);

  // Somar dispositivos IoT em todas as seÃ§Ãµes de primeiro nÃvel
  const totalDevicesCount = sections.reduce((sum, s) => {
    const devices = Array.isArray((s as any).DeviceIot)
      ? (s as any).DeviceIot.length
      : 0;
    return sum + devices;
  }, 0);

  return {
    activeMonitoringCount,
    totalSectionsCount,
    totalSectorsCount,
    totalDevicesCount,
  };
}

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring-sensor/MonitoringSensor
.tsx =====
// src/pages/monitoring-sensor/MonitoringSensor.tsx
import React from "react";
import { PlusOutlined, FilterOutlined } from "@ant-design/icons";
import { useNavigate } from "react-router-dom";
import ItemSideBar from "../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../layout/Header/ItemHeader";
import Button from "../../components/Button/Button";
import CustomTable from "../../components/Table/Table";
import ItemHeaderCabecalho from "../../layout/Header/components/ItemHeaderCabecalho";
import { useSensorTable } from "./hooks/useSensorTable";

const MonitoringSensor: React.FC = () => {
  const navigate = useNavigate();
  const { columns, monitorings, loading } = useSensorTable();
```

```
  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          <ItemHeaderCabecalho
            title="Nansensor"
            subTitle="Lista de NansenSensor"
          />

          <section className="actions-section">
            <Button
              type="primary"
              icon={<PlusOutlined />}
              onClick={() => navigate("/sensor-monitoring/register")}
            >
              Cadastrar Sensor
            </Button>
            <Button type="link" icon={<FilterOutlined />}>
              Filtros
            </Button>
          </section>

          <section className="table-container">
            <CustomTable
              columns={columns}
              data={monitorings}
              loading={loading}
            />
          </section>
        </main>
      </div>
    </div>
  );
};

export default MonitoringSensor;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring-sensor/components/Monit
oringSections.tsx =====

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring-sensor/components/Monit
oringDetails.tsx =====
import React from "react";
import { MonitoringItem } from "../../../types/monitoringTypes";

interface Props {
  monitoring: MonitoringItem;
}

const MonitoringDetails: React.FC<Props> = ({ monitoring }) => {
  return (
    <section className="monitoring-info">
      <div className="monitoring-table">
        <div className="monitoring-row">
          <div className="monitoring-cell">
            <span className="info-label">Nome do Monitoramento:</span>
            <span className="info-value">{monitoring.name}</span>
          </div>
          <div className="monitoring-cell">
            <span className="info-label">DescriÃ§Ã£o:</span>
            <span className="info-value">{monitoring.description}</span>
          </div>
        </div>
      </div>
    </section>
  );
```

```
};

export default MonitoringDetails;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring-sensor/components/colum
nsWithActions.tsx =====
import { SectionItem } from "@/types/sections";
import { ColumnsType } from "antd/es/table";
import Actions from "@/components/actions/Actions"; // <-- Import corrigido

type Handlers = {
  onEdit: (id: number) => void;
  onDelete: (id: number) => void;
  onConfigure: (section: SectionItem) => void;
};

export const columnsWithActions = (
  baseColumns: ColumnsType<SectionItem>,
  { onEdit, onDelete, onConfigure }: Handlers
): ColumnsType<SectionItem> => {
  return [
    ...baseColumns,
    {
      title: "AÃ§Ãµes",
      key: "actions",
      render: (_: unknown, record: SectionItem) => (
        <Actions
          onEdit={() => onEdit(record.id)}
          onDelete={() => onDelete(record.id)}
          onConfigure={() => onConfigure(record)}
        />
      ),
    },
  ];
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring-sensor/components/Secti
onList.tsx =====
import { Table, Badge, Button, Tooltip } from "antd";
import { ThunderboltOutlined } from "@ant-design/icons";
import { useNavigate, useParams } from "react-router-dom";
import { useSectionTable } from "../hooks/useSectionTable";
import { useSectionHierarchy } from "../hooks/useSectionHierarchy";
import { useSectionActions } from "../hooks/useSectionActions";
import { columnsWithActions } from "./columnsWithActions";
import SectionExpandedTree from "./SectionExpandedTree";
import MonitoringConfigureSectionModal from "./MonitoringConfigureSectionModal";
import { SectionItem } from "@/types/sections";

const SectionList = () => {
  const { id } = useParams<{ id: string }>();
  const navigate = useNavigate();

  const {
    columns,
    sections,
    loading,
    sectionToConfigure,
    isConfigureModalVisible,
    handleOpenConfigure,
    handleCloseConfigure,
  } = useSectionTable();

  const { handleDelete } = useSectionActions();
  const { setoresPrincipais } = useSectionHierarchy(sections);

  // Verifica se uma seÃ§Ã£o ou qualquer descendente possui IoT (LED Verde)
  const hasIotDeviceRecursive = (section: SectionItem, allSections: SectionItem[]): boolean
 => {
    if (section.DeviceIot) return true;
```

```
    const children = allSections.filter((s) => s.secticon_parent === section.id);
    return children.some((child) => hasIotDeviceRecursive(child, allSections));
  };

  // Verifica se a seÃ§Ã£o tem um dispositivo IoT diretamente associado (Ã\215cone de Monit
oramento)
  const hasDirectIotDevice = (section: SectionItem): boolean => {
    return !!section.DeviceIot;
  };

  // FunÃ§Ã£o para lidar com o clique no Ã-cone de monitoramento
  const handleMonitorClick = (section: SectionItem) => {
    console.log("ð\237\224\215 Monitoramento da seÃ§Ã£o:", section.name);
    // Aqui vamos abrir o mini modal futuramente
  };

  // Substitui a coluna "Nome" para adicionar LED verde e Ã-cone de monitoramento
  const enhancedColumns = columns.map((col) => {
    if (col.key === "name") {
      return {
        ...col,
        render: (_: unknown, record: SectionItem) => {
          const hasIot = hasIotDeviceRecursive(record, sections);
          const hasDirectIot = hasDirectIotDevice(record);

          return (
            <span style={{ display: "flex", alignItems: "center" }}>
              {/* LED Verde para toda a Ã¡rvore */}
              {hasIot && <Badge status="success" style={{ marginRight: 6 }} />}
              {record.name}
              {/* Ã\215cone de Monitoramento apenas para a seÃ§Ã£o diretamente associada */
}
              {hasDirectIot && (
                <Tooltip title="Monitoramento Ativo">
                  <Button
                    type="text"
                    icon={<ThunderboltOutlined />}
                    onClick={() => handleMonitorClick(record)}
                    style={{ marginLeft: 8, color: "#faad14" }}
                  />
                </Tooltip>
              )}
            </span>
          );
        },
      };
    }
    return col;
  });

  const actionColumns = columnsWithActions(enhancedColumns, {
    onEdit: (sectionId) => navigate(`/monitoring/edit-section/${sectionId}`),
    onDelete: handleDelete,
    onConfigure: handleOpenConfigure,
  });

  return (
    <>
      <Table
        columns={actionColumns}
        dataSource={setoresPrincipais.filter((s) => s.monitoring === Number(id))}
        loading={loading}
        rowKey="id"
        expandable={{
          expandedRowRender: (record: SectionItem) => (
            <SectionExpandedTree
              section={record}
              allSections={sections}
              onConfigure={handleOpenConfigure}
```

```
              onDelete={handleDelete}
              onMonitor={handleMonitorClick}
            />
          ),
        }}
        pagination={{ pageSize: 10 }}
      />

      {sectionToConfigure && (
        <MonitoringConfigureSectionModal
          section={sectionToConfigure}
          open={isConfigureModalVisible}
          onClose={handleCloseConfigure}
        />
      )}
    </>
  );
};

export default SectionList;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring-sensor/components/Monit
oringConfigureSectionModal.tsx =====
import React, { useEffect, useState } from "react";
import { Modal, Switch, Select, message } from "antd";
import { useIoTDevices } from "@/hooks/useIoTDevices";
import { SectionItem } from "@/types/sections";
import { useSectionStore } from "@/store/sectionStore";

interface Props {
  section: SectionItem;
  open: boolean;
  onClose: () => void;
}

const MonitoringConfigureSectionModal: React.FC<Props> = ({ section, open, onClose }) => {
  const { devices, fetchDevices } = useIoTDevices();
  const { updateSection } = useSectionStore();

  const [form, setForm] = useState({
    is_monitored: section.is_monitored,
    deviceIot: section.DeviceIot ?? null,
  });

  const [loading, setLoading] = useState(false);

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

  const handleChange = (name: string, value: any) => {
    setForm((prev) => ({
      ...prev,
      [name]: value,
    }));
  };

  const handleSave = async () => {
    try {
      setLoading(true);
      await updateSection(section.id, {
        name: section.name,
        description: section.description,
        is_monitored: form.is_monitored,
        DeviceIot: form.is_monitored ? form.deviceIot : null,
        monitoring: section.monitoring,
        setor: section.setor,
        productionLine: section.productionLine,
        equipament: section.equipament,
        type_section: section.type_section,
```

```
        secticon_parent: section.secticon_parent,
      });

      message.success("SeÃ§Ã£o atualizada com sucesso!");
      onClose();
    } catch (error) {
      console.error("Erro ao configurar seÃ§Ã£o:", error);
      message.error("Erro ao configurar seÃ§Ã£o.");
    } finally {
      setLoading(false);
    }
  };

  return (
    <Modal
      title="Configurar SeÃ§Ã£o"
      open={open}
      onCancel={onClose}
      onOk={handleSave}
      confirmLoading={loading}
    >
      <div style={{ marginBottom: 20 }}>
        <p>Monitorado?</p>
        <Switch
          checked={form.is_monitored}
          onChange={(value) => handleChange("is_monitored", value)}
        />
      </div>

      {form.is_monitored && (
        <div>
          <p>Dispositivo IoT</p>
          <Select
            style={{ width: "100%" }}
            value={form.deviceIot}
            onChange={(value) => handleChange("deviceIot", value)}
            options={devices.map((device) => ({
              value: device.id,
              label: device.name,
            }))}
          />
        </div>
      )}
    </Modal>
  );
};

export default MonitoringConfigureSectionModal;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring-sensor/components/Monit
oringConfigure.tsx =====
import React from "react";
import { PlusOutlined } from "@ant-design/icons";
import { useNavigate } from "react-router-dom";
import ItemSideBar from "../../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../../layout/Header/components/ItemHeaderCabecalho";
import Button from "../../../components/Button/Button";
import MonitoringDetails from "./MonitoringDetails";
import { useSensorById } from "../hooks/useSensorById";
import SectionList from "./SectionList";

const MonitoringConfigure: React.FC = () => {
  const { monitoring, loading } = useSensorById();
  const navigate = useNavigate();

  if (loading) return <p>Carregando...</p>;
  if (!monitoring) return <p>Monitoramento nÃ£o encontrado.</p>;

  return (
```

```
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          <ItemHeaderCabecalho
            title="Configurar Monitoramento de Energia"
            subTitle="Monitoramento de Energia"
          />

          {/* InformaÃ§Ãµes do Monitoramento */}
          <MonitoringDetails monitoring={monitoring} />

          {/* BotÃ£o Adicionar SeÃ§Ã£o */}
          <section className="actions-section">
            <Button
              type="primary"
              className="add-section-btn"
              icon={<PlusOutlined />}
              onClick={() => navigate(`/monitoring/add-section/${monitoring.id}`)}
            >
              Adicionar SeÃ§Ã£o
            </Button>
          </section>

          {/* Lista de SeÃ§Ãµes */}
          <section className="table-container">
            <p>Lista de Consumo (SeÃ§Ãµes associadas)</p>
            <SectionList /> {/* Adiciona o componente da lista de seÃ§Ãµes */}
          </section>
        </main>
      </div>
    </div>
  );
};

export default MonitoringConfigure;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring-sensor/components/Secti
onForm.tsx =====
import React from "react";
import DynamicForm from "@/components/form/DynamicForm";
import { SectionFormValues } from "@/pages/monitoring/hooks/useSectionForm";
import { TypeSection } from "@/store/typeSectionStore";

interface SectionFormProps {
  values: SectionFormValues;
  onChange: (name: keyof SectionFormValues, value: any) => void;
  onCancel: () => void;
  onSubmit: () => void;
  loading: boolean;
  availableSections: { value: number; label: string }[];
  devices: { id: number; name: string }[];
  typeSections: TypeSection[]; // â\234\205 tipos reais
  isEdit?: boolean;
}

const SectionForm: React.FC<SectionFormProps> = ({
  values,
  onChange,
  onCancel,
  onSubmit,
  loading,
  availableSections,
  devices,
  typeSections,
  isEdit = false,
}) => {
  return (
    <DynamicForm
```

```
      fields={[

        {
          name: "is_monitored",
          label: "Monitorado?",
          type: "switch",
        },
        {
          name: "type_section",
          label: "Tipo da SeÃ§Ã£o",
          type: "select",
          options: typeSections.map((t) => ({
            value: t.name, // "SETOR", "LINHA", "EQUIPAMENTO"
            label:
              t.name === "SETOR"
                ? "Setor"
                : t.name === "LINHA"
                ? "Linha de ProduÃ§Ã£o"
                : "Equipamento",
          })),
          disabled: isEdit,
        },
        {
          name: "section_consume",
          label: "SeÃ§Ã£o de Consumo",
          type: "select",
          options: availableSections,
          disabled: !values.type_section,
        },
        {
          name: "deviceIot",
          label: "Dispositivo IoT",
          type: "select",
          options: devices.map((device) => ({
            value: device.id,
            label: device.name,
          })),
          disabled: !values.is_monitored,
        },
      ]}
      values={values}
      onChange={(name, value) => onChange(name as keyof SectionFormValues, value)}
      loading={loading}
      onCancel={onCancel}
      onSubmit={onSubmit}
    />
  );
};

export default SectionForm;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring-sensor/components/Secti
onEdit.tsx =====
import React, { useEffect, useState } from "react";
import { Modal, message } from "antd";
import { useParams, useNavigate } from "react-router-dom";
import { useSectionStore } from "@/store/sectionStore";
import { useSectionForm } from "@/pages/monitoring/hooks/useSectionForm";
import { useTypeSectionStore, TypeSection } from "@/store/typeSectionStore";
import SectionForm from "./SectionForm";

const SectionEdit: React.FC = () => {
  const { id } = useParams<{ id: string }>();
  const navigate = useNavigate();

  const { sections, fetchSections, updateSection } = useSectionStore();
  const { fetchTypes, types } = useTypeSectionStore();

  const {
    formValues,
```

```
    setFormValues,
    handleChange,
    getAvailableSections,
    devices,
  } = useSectionForm(true); // â\234\205 modo ediÃ§Ã£o ativado

  const [loading, setLoading] = useState(false);
  const [monitoringId, setMonitoringId] = useState<number | null>(null);
  const [loaded, setLoaded] = useState(false); // â\234\205 impede reexecuÃ§Ã£o do setFormV
alues

  useEffect(() => {
    fetchSections();
    fetchTypes();
  }, []);

  useEffect(() => {
    if (id && sections.length > 0 && types.length > 0 && !loaded) {
      const section = sections.find((s) => s.id === Number(id));
      if (!section) {
        message.error("SeÃ§Ã£o nÃ£o encontrada.");
        navigate("/sensor-monitoring");
        return;
      }

      const typeMatch = types.find((t: TypeSection) => t.id === section.type_section);
      if (!typeMatch) {
        message.error("Tipo da seÃ§Ã£o invÃ¡lido.");
        return;
      }

      setMonitoringId(section.monitoring || null);

      setFormValues({
        name: section.name || "",
        is_monitored: !!section.is_monitored,
        type_section: typeMatch.name as "SETOR" | "LINHA" | "EQUIPAMENTO",
        section_consume:
          section.setor ?? section.productionLine ?? section.equipament ?? null,
        deviceIot: section.DeviceIot ?? null,
      });

      setLoaded(true); // â\234\205 garante que isso sÃ³ roda uma vez
    }
  }, [id, sections, types, loaded]);

  const handleSubmit = async () => {
    if (!formValues.name.trim()) {
      message.error("O nome da seÃ§Ã£o Ã© obrigatÃ³rio!");
      return;
    }

    const typeId = types.find((t: TypeSection) => t.name === formValues.type_section)?.id;

    if (!typeId) {
      message.error("Tipo da seÃ§Ã£o invÃ¡lido!");
      return;
    }

    setLoading(true);
    try {
      await updateSection(Number(id), {
        name: formValues.name,
        is_monitored: formValues.is_monitored,
        type_section: typeId,
        DeviceIot: formValues.is_monitored ? formValues.deviceIot : null,
        setor: formValues.type_section === "SETOR" ? formValues.section_consume : null,
        productionLine: formValues.type_section === "LINHA" ? formValues.section_consume :
null,
```

```
        equipament: formValues.type_section === "EQUIPAMENTO" ? formValues.section_consume
: null,
      });

      message.success("SeÃ§Ã£o atualizada com sucesso!");
      navigate(`/monitoring/configure/${monitoringId}`);
    } catch (error) {
      console.error("â\235\214 Erro ao atualizar:", error);
      message.error("Erro ao atualizar seÃ§Ã£o!");
    } finally {
      setLoading(false);
    }
  };

  return (
    <Modal
      title="Editar SeÃ§Ã£o"
      open={true}
      footer={null}
      onCancel={() => navigate(`/monitoring/configure/${monitoringId}`)}
    >
      <SectionForm
        values={formValues}
        onChange={handleChange}
        onCancel={() => navigate(`/monitoring/configure/${monitoringId}`)}
        onSubmit={handleSubmit}
        loading={loading}
        availableSections={getAvailableSections()}
        devices={devices}
        typeSections={types}
        isEdit={true}
      />
    </Modal>
  );
};

export default SectionEdit;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring-sensor/components/Secti
onExpandedTree.tsx =====
import { FC } from "react";
import { Tree, Collapse, Button, Popconfirm, Badge, Tooltip } from "antd";
import { SettingOutlined, DeleteOutlined, ThunderboltOutlined } from "@ant-design/icons";
import { SectionItem } from "@/types/sections";

type Props = {
  section: SectionItem;
  allSections: SectionItem[];
  onConfigure: (section: SectionItem) => void;
  onDelete: (sectionId: number) => void;
  onMonitor: (section: SectionItem) => void;
};

// Verifica recursivamente se a seÃ§Ã£o ou alguma de suas filhas tem DeviceIot
const hasIotDeviceRecursive = (section: SectionItem, allSections: SectionItem[]): boolean =
> {
  if (section.DeviceIot) return true;
  const children = allSections.filter((s) => s.secticon_parent === section.id);
  return children.some((child) => hasIotDeviceRecursive(child, allSections));
};

// Verifica se a seÃ§Ã£o tem um dispositivo IoT diretamente associado
const hasDirectIotDevice = (section: SectionItem): boolean => {
  return !!section.DeviceIot;
};

const SectionExpandedTree: FC<Props> = ({ section, allSections, onConfigure, onDelete, onMo
nitor }) => {
  const buildTree = (parent: SectionItem): any => {
    const children = allSections.filter((s) => s.secticon_parent === parent.id);
```

```
    const hasIot = hasIotDeviceRecursive(parent, allSections);
    const hasDirectIot = hasDirectIotDevice(parent);

    return {
      title: (
        <span>
          {/* LED de status se houver IoT em qualquer nÃ-vel */}
          {hasIot && <Badge status="success" style={{ marginRight: 6 }} />}
          {parent.name}

          {/* Ã\215cone de Monitoramento, apenas se houver IoT diretamente associado */}
          {hasDirectIot && (
            <Tooltip title="Monitoramento Ativo">
              <Button
                type="text"
                icon={<ThunderboltOutlined />}
                onClick={() => onMonitor(parent)}
                style={{ marginLeft: 6, color: "#faad14" }} // Cor amarela
              />
            </Tooltip>
          )}

          <Button
            type="text"
            icon={<SettingOutlined />}
            onClick={() => onConfigure(parent)}
            style={{ marginLeft: 8 }}
          />
          <Popconfirm
            title="Deseja excluir esta subseÃ§Ã£o?"
            onConfirm={() => onDelete(parent.id)}
            okText="Sim"
            cancelText="NÃ£o"
          >
            <Button
              type="text"
              icon={<DeleteOutlined />}
              danger
              style={{ marginLeft: 4 }}
            />
          </Popconfirm>
        </span>
      ),
      key: `section-${parent.id}`,
      children: children.map(buildTree),
    };
  };

  const rootChildren = allSections
    .filter((s) => s.secticon_parent === section.id)
    .map(buildTree);

  return (
    <Collapse
      items={[
        {
          key: `panel-${section.id}`,
          label: "SubseÃ§Ãµes",
          children: rootChildren.length > 0 ? (
            <Tree treeData={rootChildren} />
          ) : (
            <p>NÃ£o hÃ¡ subseÃ§Ãµes associadas.</p>
          ),
        },
      ]}
    />
  );
};
```

```
export default SectionExpandedTree;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring-sensor/components/Monit
oringForm.tsx =====
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import { message } from "antd";
import ItemSideBar from "../../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../../layout/Header/components/ItemHeaderCabecalho";
import DynamicForm from "../../../components/form/DynamicForm";
import { useSensor } from "../hooks/useSensor";

const MonitoringForm: React.FC = () => {
  const navigate = useNavigate();
  const { addMonitoring } = useSensor();
  const [loading, setLoading] = useState(false);
  const [formValues, setFormValues] = useState({
    name: "",
    description: "",
    estimated_consumption: "",
  });

  const handleChange = (name: string, value: any) => {
    setFormValues((prev) => ({ ...prev, [name]: value }));
  };

  const handleSubmit = async () => {
    if (!formValues.name.trim()) {
      message.error("O nome do sensor Ã© obrigatÃ³rio!");
      return;
    }

    if (!formValues.description.trim()) {
      message.error("A descriÃ§Ã£o do sensor Ã© obrigatÃ³ria!");
      return;
    }

    if (loading) return;
    setLoading(true);

    try {
      await addMonitoring({
        name: formValues.name,
        description: formValues.description,
        estimated_consumption: Number(formValues.estimated_consumption) || 0,
      });
      message.success("Sensor cadastrado com sucesso!");
      navigate("/sensor-monitoring");
    } catch (error) {
      console.error(error);
      message.error("Erro ao cadastrar sensor!");
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          <ItemHeaderCabecalho
            title="Cadastro de Sensor"
            subTitle="FormulÃ¡rio para cadastro de sensor no NansenSensor"
          />

          <DynamicForm
            fields={[
```

```
              {
                name: "name",
                label: "Nome do Sensor",
                type: "input",
                required: true,
              },
              {
                name: "description",
                label: "DescriÃ§Ã£o",
                type: "textarea",
                required: true,
              },
              {
                name: "estimated_consumption",
                label: "Consumo Estimado (kWh)",
                type: "number",
              },
            ]}
            values={formValues}
            onChange={handleChange}
            onSubmit={handleSubmit}
            loading={loading}
            onCancel={() => navigate("/sensor-monitoring")}
          />
        </main>
      </div>
    </div>
  );
};

export default MonitoringForm;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring-sensor/components/Monit
oringAddSection.tsx =====
import React, { useEffect, useState } from "react";
import { Modal, message } from "antd";
import { useParams, useNavigate } from "react-router-dom";
import { useSectionStore } from "@/store/sectionStore";
import { useSectionForm } from "@/pages/monitoring/hooks/useSectionForm";
import SectionForm from "./SectionForm";

const MonitoringAddSection: React.FC = () => {
  const { id } = useParams<{ id: string }>();
  const navigate = useNavigate();
  const { addSection } = useSectionStore();

  const {
    formValues,
    handleChange,
    getAvailableSections,
    devices,
    typeSections, // â\234\205 Tipos de seÃ§Ã£o da API
  } = useSectionForm();

  const [loading, setLoading] = useState(false);

  const handleSubmit = async () => {
    if (!formValues.name.trim()) {
      message.error("O nome da seÃ§Ã£o Ã© obrigatÃ³rio!");
      return;
    }

    const typeId = typeSections.find((t) => t.name === formValues.type_section)?.id;

    if (!typeId) {
      message.error("ID do tipo de seÃ§Ã£o nÃ£o encontrado!");
      return;
    }
```

```
    console.log("ð\237\221\211 Enviando tipo_section ID:", typeId);

    setLoading(true);
    try {
      await addSection({
        name: formValues.name,
        is_monitored: formValues.is_monitored,
        monitoring: Number(id),
        type_section: typeId,
        DeviceIot: formValues.is_monitored ? formValues.deviceIot : null,
        setor: formValues.type_section === "SETOR" ? formValues.section_consume : null,
        productionLine: formValues.type_section === "LINHA" ? formValues.section_consume :
null,
        equipament: formValues.type_section === "EQUIPAMENTO" ? formValues.section_consume
: null,
      });

      message.success("SeÃ§Ã£o adicionada com sucesso!");
      navigate(`/monitoring/configure/${id}`);
    } catch (error) {
      console.error("â\235\214 Erro real ao adicionar:", error);
      message.error("Erro ao adicionar seÃ§Ã£o!");
    } finally {
      setLoading(false);
    }
  };

  return (
    <Modal
      title="Adicionar Nova SeÃ§Ã£o"
      open={true}
      footer={null}
      onCancel={() => navigate(`/monitoring/configure/${id}`)}
    >
      <SectionForm
        values={formValues}
        onChange={handleChange}
        onCancel={() => navigate(`/monitoring/configure/${id}`)}
        onSubmit={handleSubmit}
        loading={loading}
        availableSections={getAvailableSections()}
        devices={devices}
        typeSections={typeSections} // â\234\205 Passa os tipos pro form
      />
    </Modal>
  );
};

export default MonitoringAddSection;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring-sensor/components/Monit
oringEdit.tsx =====
import React, { useEffect, useState } from "react";
import { useParams, useNavigate } from "react-router-dom";
import { message } from "antd";
import ItemSideBar from "../../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../../layout/Header/components/ItemHeaderCabecalho";
import DynamicForm from "../../../components/form/DynamicForm";
import { useSensor } from "../hooks/useSensor";
import { MonitoringItem } from "@/types/monitoringTypes";

const MonitoringEdit: React.FC = () => {
  const { id } = useParams<{ id: string }>();
  const navigate = useNavigate();
  const { sensorMonitorings, editMonitoring } = useSensor();

  const [loading, setLoading] = useState(false);
  const [formValues, setFormValues] = useState({
    name: "",
```

```
  description: "",
  estimated_consumption: "",
});

useEffect(() => {
  if (id) {
    const monitoring = sensorMonitorings.find(
      (item: MonitoringItem) => item.id === Number(id)
    );
    if (monitoring) {
      setFormValues({
        name: monitoring.name,
        description: monitoring.description,
        estimated_consumption: monitoring.estimated_consumption.toString(),
      });
    } else {
      message.error("Sensor nÃ£o encontrado.");
      navigate("/sensor-monitoring");
    }
  }
}, [id, sensorMonitorings, navigate]);

const handleChange = (name: string, value: any) => {
  setFormValues((prev) => ({ ...prev, [name]: value }));
};

const handleSubmit = async () => {
  if (!formValues.name.trim()) {
    message.error("O nome do sensor Ã© obrigatÃ³rio!");
    return;
  }

  if (!formValues.description.trim()) {
    message.error("A descriÃ§Ã£o do sensor Ã© obrigatÃ³ria!");
    return;
  }

  setLoading(true);
  try {
    await editMonitoring(Number(id), {
      name: formValues.name,
      description: formValues.description,
      estimated_consumption: Number(formValues.estimated_consumption) || 0,
    });
    message.success("Sensor atualizado com sucesso!");
    navigate("/sensor-monitoring");
  } catch (error) {
    console.error(error);
    message.error("Erro ao atualizar sensor!");
  } finally {
    setLoading(false);
  }
};

return (
  <div className="layout-container">
    <ItemSideBar />
    <div className="content-container">
      <ItemHeader />
      <main className="content">
        <ItemHeaderCabecalho
          title="Editar Sensor"
          subTitle="FormulÃ¡rio para ediÃ§Ã£o de sensor no NansenSensor"
        />
        <DynamicForm
          fields={[
            {
              name: "name",
              label: "Nome do Sensor",
```

```
                type: "input",
                required: true,
              },
              {
                name: "description",
                label: "DescriÃ§Ã£o",
                type: "textarea",
                required: true,
              },
              {
                name: "estimated_consumption",
                label: "Consumo Estimado (kWh)",
                type: "number",
              },
            ]}
            values={formValues}
            onChange={handleChange}
            onSubmit={handleSubmit}
            loading={loading}
            onCancel={() => navigate("/sensor-monitoring")}
          />
        </main>
      </div>
    </div>
  );
};

export default MonitoringEdit;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring-sensor/hooks/useSensorT
able.tsx =====
// src/pages/monitoring-sensor/hooks/useSensorTable.tsx
import { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
import { message } from "antd";
import Actions from "@/components/actions/Actions";
import { useSensorMonitoringStore } from "@/store/sensorMonitoringStore";
import type { MonitoringItem } from "@/types/monitoringTypes";

export const useSensorTable = () => {
  const navigate = useNavigate();
  const { sensorMonitorings, fetchSensorMonitorings, deleteSensorMonitoring } =
    useSensorMonitoringStore();

  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const load = async () => {
      setLoading(true);
      try {
        await fetchSensorMonitorings();
      } catch {
        message.error("Erro ao carregar sensores.");
      } finally {
        setLoading(false);
      }
    };
    load();
  }, [fetchSensorMonitorings]);

  const columns = [
    {
      title: "Nome",
      dataIndex: "name",
      key: "name",
      sorter: (a: MonitoringItem, b: MonitoringItem) =>
        a.name.localeCompare(b.name),
    },
    {
```

```
      title: "DescriÃ§Ã£o",
      dataIndex: "description",
      key: "description",
    },
    {
      title: "AÃ§Ãµes",
      key: "actions",
      align: "center",
      render: (_: any, record: MonitoringItem) => (
        <div style={{ display: "flex", justifyContent: "center", gap: 8 }}>
          <Actions
            onEdit={() => navigate(`/sensor-monitoring/edit/${record.id}`)}
            onConfigure={() =>
              navigate(`/sensor-monitoring/configure/${record.id}`)
            }
            onDelete={async () => {
              await deleteSensorMonitoring(record.id);
              message.success("Sensor excluÃ-do.");
            }}
          />
        </div>
      ),
    },
  ];

  return {
    columns,
    monitorings: sensorMonitorings,
    loading,
  };
};

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring-sensor/hooks/useSection
.ts =====
import { useEffect } from "react";
import { useSectionStore } from "@/store/sectionStore";

export const useSection = () => {
  const {
    sections,
    fetchSections,
    addSection,
    updateSection,
    deleteSection,
    loading,
  } = useSectionStore();

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

  return {
    sections,
    loading,
    addSection,
    editSection: updateSection,
    removeSection: deleteSection,
  };
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring-sensor/hooks/useSection
Actions.ts =====
import { useState } from "react";
import { message } from "antd";
import { SectionItem } from "@/types/sections";
import { useSectionStore } from "@/store/sectionStore";

export const useSectionActions = () => {
  const [sectionToConfigure, setSectionToConfigure] = useState<SectionItem | null>(null);
  const { deleteSection, fetchSections } = useSectionStore();
```

```
  const handleOpenConfig = (section: SectionItem) => {
    setSectionToConfigure(section);
  };

  const handleCloseConfig = () => {
    setSectionToConfigure(null);
  };

  const handleDelete = async (id: number) => {
    try {
      await deleteSection(id);
      await fetchSections(); // garante atualizaÃ§Ã£o visual apÃ³s excluir hierarquias
      message.success("SeÃ§Ã£o excluÃ-da com sucesso.");
    } catch (err) {
      console.error(err);
      message.error("Erro ao excluir seÃ§Ã£o.");
    }
  };

  const handleEdit = (id: number) => {
    // Aqui vocÃª pode redirecionar ou setar um estado global de ediÃ§Ã£o
    console.log("Editar seÃ§Ã£o com ID:", id);
  };

  return {
    sectionToConfigure,
    handleOpenConfig,
    handleCloseConfig,
    handleDelete,
    handleEdit,
  };
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring-sensor/hooks/useSection
Table.tsx =====
import { useEffect, useState } from "react";
import { message } from "antd";
import { useSectionStore } from "@/store/sectionStore";
import { useIoTDevices } from "@/hooks/useIoTDevices";
import { SectionItem } from "@/types/sections";
import { useSectionHierarchy } from "./useSectionHierarchy";

export const useSectionTable = () => {
  // Stores
  const {
    sections,
    fetchSections,
    deleteSection,
    updateSection,
    loading,
  } = useSectionStore();

  const { devices, fetchDevices } = useIoTDevices();

  // Estados locais
  const [sectionToConfigure, setSectionToConfigure] = useState<SectionItem | null>(null);
  const [isConfigureModalVisible, setIsConfigureModalVisible] = useState(false);

  // Carregamento inicial
  useEffect(() => {
    const loadData = async () => {
      try {
        await Promise.all([fetchSections(), fetchDevices()]);
      } catch (error) {
        message.error("Erro ao carregar seÃ§Ãµes ou dispositivos.");
      }
    };

    loadData();
```

```
}, [fetchSections, fetchDevices]);

// Processa hierarquia das seÃ§Ãµes
const { setoresPrincipais, filteredSections } = useSectionHierarchy(sections);

// Abrir modal de configuraÃ§Ã£o
const handleOpenConfigure = (section: SectionItem) => {
  setSectionToConfigure(section);
  setIsConfigureModalVisible(true);
};

// Fechar modal
const handleCloseConfigure = () => {
  setIsConfigureModalVisible(false);
  setSectionToConfigure(null);
};

// Salvar configuraÃ§Ã£o de uma seÃ§Ã£o
const handleSaveConfigure = async (data: Partial<SectionItem>) => {
  if (!sectionToConfigure) return;

  try {
    const updatedData: Partial<SectionItem> = {
      ...sectionToConfigure,
      is_monitored: data.is_monitored,
      DeviceIot: data.is_monitored ? data.DeviceIot : null,
    };

    await updateSection(sectionToConfigure.id, updatedData);
    message.success("SeÃ§Ã£o configurada com sucesso.");
    handleCloseConfigure();
  } catch (error) {
    console.error(error);
    message.error("Erro ao configurar seÃ§Ã£o.");
  }
};

// Colunas da tabela
const baseColumns = [
  {
    title: "SeÃ§Ã£o",
    dataIndex: "name",
    key: "name",
    sorter: (a: SectionItem, b: SectionItem) => a.name.localeCompare(b.name),
  },
  {
    title: "DescriÃ§Ã£o",
    dataIndex: "description",
    key: "description",
  },
  {
    title: "Consumo Estimado (kWh)",
    dataIndex: "estimated_consumption",
    key: "estimated_consumption",
  },
];

return {
  columns: baseColumns,
  sections: filteredSections,
  setoresPrincipais,
  loading,
  sectionToConfigure,
  isConfigureModalVisible,
  setIsConfigureModalVisible,
  handleOpenConfigure,
  handleCloseConfigure,
  handleSaveConfigure,
  deleteSection,
```

```
    devices,
  };
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring-sensor/hooks/useSection
Hierarchy.ts =====
import { useParams } from "react-router-dom";
import { SectionItem } from "@/types/sections";

/**
 * FunÃ§Ã£o recursiva que achata uma hierarquia de seÃ§Ãµes em uma lista Ãºnica.
 */
const flattenHierarchy = (sections: SectionItem[]): SectionItem[] => {
  const result: SectionItem[] = [];

  for (const section of sections) {
    result.push(section);
    if (section.sections_filhas && section.sections_filhas.length > 0) {
      result.push(...flattenHierarchy(section.sections_filhas));
    }
  }

  return result;
};

export const useSectionHierarchy = (
  sections: SectionItem[]
): {
  setoresPrincipais: SectionItem[];
  filteredSections: SectionItem[];
} => {
  const { id } = useParams<{ id: string }>();
  const monitoringId = Number(id);

  // Se id nÃ£o for vÃ¡lido, retorna arrays vazios
  if (isNaN(monitoringId)) {
    return {
      setoresPrincipais: [],
      filteredSections: [],
    };
  }

  // Setores principais (sem seÃ§Ã£o pai e pertencentes ao monitoramento)
  const setoresPrincipais = sections.filter(
    (s) => s.monitoring === monitoringId && !s.secticon_parent
  );

  // Achata a hierarquia a partir dos setores principais
  const filteredSections = flattenHierarchy(setoresPrincipais);

  return {
    setoresPrincipais,
    filteredSections,
  };
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring-sensor/hooks/useSensor.
ts =====
// src/pages/monitoring-sensor/hooks/useSensor.ts
import { useEffect } from "react";
import { useSensorMonitoringStore } from "@/store/sensorMonitoringStore";
import { MonitoringItem } from "@/types/monitoringTypes";

export const useSensor = () => {
  const { sensorMonitorings, fetchSensorMonitorings, deleteSensorMonitoring } =
    useSensorMonitoringStore();

  useEffect(() => {
    fetchSensorMonitorings();
  }, [fetchSensorMonitorings]);
```

```
  const addMonitoring = async (data: Partial<MonitoringItem>) => {
    await fetchSensorMonitorings();
  };

  const editMonitoring = async (id: number, data: Partial<MonitoringItem>) => {
    await fetchSensorMonitorings();
  };

  const removeMonitoring = async (id: number) => {
    await deleteSensorMonitoring(id);
  };

  return {
    sensorMonitorings,
    addMonitoring,
    editMonitoring,
    removeMonitoring,
  };
};

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring-sensor/hooks/useSensorB
yId.tsx =====
import { useEffect, useState } from "react";
import { useParams, useNavigate } from "react-router-dom";
import { useSensorMonitoringStore } from "@/store/sensorMonitoringStore";
import { MonitoringItem } from "@/types/monitoringTypes";
import { message } from "antd";

export const useSensorById = () => {
  const { id } = useParams<{ id: string }>();
  const navigate = useNavigate();

  const { sensorMonitorings, fetchSensorMonitorings } =
    useSensorMonitoringStore();

  const [monitoring, setMonitoring] = useState<MonitoringItem | null>(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const load = async () => {
      setLoading(true);
      await fetchSensorMonitorings();
      setLoading(false);
    };
    load();
  }, [fetchSensorMonitorings]);

  useEffect(() => {
    if (!loading) {
      const found = sensorMonitorings.find((m) => m.id === Number(id));
      if (found) {
        setMonitoring(found);
      } else {
        message.error("Sensor nÃ£o encontrado.");
        navigate("/sensor-monitoring");
      }
    }
  }, [loading, sensorMonitorings, id, navigate]);

  return { monitoring, loading };
};

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/monitoring-sensor/hooks/useSection
Form.ts =====
import { useEffect, useState } from "react";
import { useSectorsStore } from "@/store/sectors";
import { useProductionLinesStore } from "@/store/ProductionLinesStore";
import { useEquipamentsStore } from "@/store/equipaments";
import { useIoTDevices } from "@/hooks/useIoTDevices";
```

```
import { useTypeSectionStore } from "@/store/typeSectionStore";
import { useSectionStore } from "@/store/sectionStore";

export type SectionFormValues = {
  name: string;
  is_monitored: boolean;
  type_section: "SETOR" | "LINHA" | "EQUIPAMENTO" | null;
  section_consume: number | null;
  deviceIot: number | null;
};

export const useSectionForm = (isEdit = false) => {
  const [formValues, setFormValues] = useState<SectionFormValues>({
    name: "",
    is_monitored: false,
    type_section: null,
    section_consume: null,
    deviceIot: null,
  });

  const { devices, fetchDevices } = useIoTDevices();
  const { sectors, fetchSectors } = useSectorsStore();
  const { productionLines, fetchProductionLines } = useProductionLinesStore();
  const { equipaments, fetchEquipaments } = useEquipamentsStore();
  const { types, fetchTypes } = useTypeSectionStore();
  const { fetchSections } = useSectionStore();

  useEffect(() => {
    fetchDevices();
    fetchSectors();
    fetchProductionLines();
    fetchEquipaments();
    fetchTypes();
  }, []);

  const filteredTypeSections = types.filter((t) => [1, 2, 3].includes(t.id)); // Apenas os
fixos

  const getSelectedLabelFromId = (
    type: SectionFormValues["type_section"],
    id: number | null
  ) => {
    if (!id) return "";
    switch (type) {
      case "SETOR":
        return sectors.find((s) => s.id === id)?.name ?? "";
      case "LINHA":
        return productionLines.find((l) => l.id === id)?.name ?? "";
      case "EQUIPAMENTO":
        return equipaments.find((e) => e.id === id)?.name ?? "";
      default:
        return "";
    }
  };

  const handleChange = <K extends keyof SectionFormValues>(
    name: K,
    value: SectionFormValues[K]
  ) => {
    setFormValues((prev) => {
      if (name === "is_monitored") {
        return {
          ...prev,
          is_monitored: Boolean(value),
          deviceIot: value ? prev.deviceIot : null,
        };
      }

      if (name === "type_section") {
```

```
        return {
          ...prev,
          type_section: value as SectionFormValues["type_section"],
          section_consume: null,
          name: "", // Limpa o nome para ser atualizado com a seÃ§Ã£o de consumo
          deviceIot: null,
        };
      }

      if (name === "section_consume") {
        const label = getSelectedLabelFromId(formValues.type_section, value as number);

        if (isEdit) fetchSections();

        return {
          ...prev,
          section_consume: value as number,
          name: label ? label : prev.name, // Atualiza o nome automaticamente
          deviceIot: null,
        };
      }

      return {
        ...prev,
        [name]: value,
      };
    });
  };


  const getAvailableSections = () => {
    switch (formValues.type_section) {
      case "SETOR":
        return sectors.map((s) => ({ value: s.id, label: s.name }));
      case "LINHA":
        return productionLines.map((l) => ({ value: l.id, label: l.name }));
      case "EQUIPAMENTO":
        return equipaments.map((e) => ({ value: e.id, label: e.name }));
      default:
        return [];
    }
  };

  return {
    formValues,
    setFormValues,
    handleChange,
    getAvailableSections,
    devices,
    typeSections: filteredTypeSections,
  };
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/iotDevices/components/IoTDevicesLi
st.tsx =====
import React from "react";
import { Table, Button, message } from "antd";
import { EditOutlined, DeleteOutlined } from "@ant-design/icons";
import { useNavigate } from "react-router-dom";
import { useIoTDevicesStore } from "@/store/iotDevices";

const IoTDevicesList: React.FC = () => {
  const navigate = useNavigate();
  const { devices, fetchDevices, removeDevice } = useIoTDevicesStore();

  const handleDelete = async (id: number) => {
    try {
      await removeDevice(id);
      message.success("Dispositivo removido com sucesso!");
      fetchDevices(); // Atualiza a lista apÃ³s exclusÃ£o
```

```
    } catch (error) {
      console.error("Erro ao remover dispositivo:", error);
      message.error("Erro ao remover dispositivo.");
    }
  };

  const columns = [
    {
      title: "Nome",
      dataIndex: "name",
      key: "name",
      render: (text: string) => <strong>{text || "Sem nome"}</strong>,
    },
    {
      title: "Tipo de Dispositivo",
      dataIndex: "type_device",
      key: "type_device",
      render: (text: string) => text || "NÃ£o informado",
    },
    {
      title: "Equipamento Associado",
      dataIndex: "equipement",
      key: "equipement",
      render: (equipement: number | null) => equipement || "Nenhum",
    },
    {
      title: "AÃ§Ãµes",
      key: "actions",
      render: (_: any, record: any) => (
        <div className="actions">
          <Button
            type="primary"
            icon={<EditOutlined />}
            onClick={() => navigate(`/iotdevices/edit/${record.id}`)}
          >
            Editar
          </Button>
          <Button
            danger
            icon={<DeleteOutlined />}
            onClick={() => handleDelete(record.id)}
          >
            Excluir
          </Button>
        </div>
      ),
    },
  ];

  return <Table dataSource={devices} columns={columns} rowKey="id" pagination={{ pageSize:
5 }} />;
};

export default IoTDevicesList;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/iotDevices/components/IoTDevicesEd
it.tsx =====
import React, { useEffect, useState } from "react";
import { useNavigate, useParams } from "react-router-dom";
import { message } from "antd";
import ItemSideBar from "../../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../../layout/Header/components/ItemHeaderCabecalho";
import DynamicForm from "../../../components/form/DynamicForm";
import { useIoTDevicesStore } from "../../../store/iotDevices";
import { getIoTDeviceById } from "../../../services/IoTDevicesService";
import { getEquipments } from "../../../services/equipmentsService";
import type { FormField as FormFieldType } from "../../../components/form/formTypes";
import type { IoTDevice } from "../../../types/IoTDevice";
```

```
const IoTDevicesEdit: React.FC = () => {
  const navigate = useNavigate();
  const { id } = useParams<{ id: string }>();
  const { editDevice } = useIoTDevicesStore();
  const [loading, setLoading] = useState(false);
  const [loadingOptions, setLoadingOptions] = useState(true);
  const [equipments, setEquipments] = useState<{ value: number; label: string }[]>([]);
  const [formValues, setFormValues] = useState<Partial<IoTDevice>>({
    name: "",
    deveui: "", // â\234\205 Agora Ã© tratado como "deveui" no front
    equipement: null,
  });

  const fields: FormFieldType[] = [
    {
      name: "name",
      label: "Nome do Dispositivo",
      type: "input",
      required: true,
      placeholder: "Digite o nome do dispositivo",
    },
    {
      name: "deveui",
      label: "DevEUI", // â\234\205 Substitui "Tipo do Dispositivo"
      type: "input",
      required: true,
      placeholder: "Digite o DevEUI do dispositivo",
    },
    {
      name: "equipement",
      label: "Equipamento Vinculado (Opcional)",
      type: "select",
      options: equipments,
      disabled: loadingOptions || equipments.length === 0,
      placeholder: equipments.length > 0 ? "Selecione um equipamento" : "Carregando opÃ§Ãµe
s...",
    },
  ];

  useEffect(() => {
    const fetchData = async () => {
      if (!id || isNaN(Number(id))) {
        message.error("ID invÃ¡lido para ediÃ§Ã£o.");
        return;
      }

      setLoading(true);
      try {
        const deviceData = await getIoTDeviceById(Number(id));

        setFormValues({
          name: deviceData.name ?? "",
          deveui: deviceData.type_device ?? "", // â\234\205 Pegamos `type_device` da API e
 usamos como `deveui`
          equipement: deviceData.equipement ?? null,
        });

        const equipmentList = await getEquipments();
        setEquipments(
          equipmentList.map((eq: { id: number; name: string }) => ({
            value: eq.id,
            label: eq.name,
          }))
        );
      } catch (error) {
        message.error("Erro ao carregar os dados do dispositivo IoT.");
        console.error("Erro ao buscar dispositivo:", error);
      } finally {
        setLoading(false);
```

```
        setLoadingOptions(false);
      }
    };

    fetchData();
  }, [id]);

  const handleChange = (name: string, value: any) => {
    setFormValues((prevValues) => ({ ...prevValues, [name]: value }));
  };

  const handleSubmit = async () => {
    if (!formValues.name?.trim()) {
      message.error("O nome do dispositivo Ã© obrigatÃ³rio!");
      return;
    }

    if (!formValues.deveui?.trim()) {
      message.error("O DevEUI do dispositivo Ã© obrigatÃ³rio!");
      return;
    }

    setLoading(true);
    try {
      await editDevice(Number(id), {
        name: formValues.name,
        type_device: formValues.deveui, // â\234\205 Agora o DevEUI Ã© enviado como `type_d
evice`
        equipement: formValues.equipement !== null ? Number(formValues.equipement) : null,
      });

      message.success("Dispositivo atualizado com sucesso!");
      navigate("/iotdevices");
    } catch (error) {
      message.error("Erro ao atualizar dispositivo IoT.");
      console.error("Erro ao atualizar dispositivo:", error);
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          <ItemHeaderCabecalho
            title="Editar Dispositivo IoT"
            subTitle="Altere os dados do dispositivo abaixo"
          />
          <DynamicForm
            fields={fields}
            values={formValues}
            loading={loading}
            onChange={handleChange}
            onSubmit={handleSubmit}
            onCancel={() => navigate("/iotdevices")}
          />
        </main>
      </div>
    </div>
  );
};

export default IoTDevicesEdit;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/iotDevices/components/IoTDevicesRe
gister.tsx =====
import React, { useState, useEffect } from "react";
```

```
import { message } from "antd";
import { useNavigate } from "react-router-dom";
import ItemSideBar from "../../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../../layout/Header/components/ItemHeaderCabecalho";
import DynamicForm from "../../../components/form/DynamicForm";
import { useIoTDevicesStore } from "../../../store/iotDevices";
import { getEquipments } from "../../../services/equipmentsService";
import type { IoTDevice } from "../../../types/IoTDevice";

const IoTDevicesRegister: React.FC = () => {
  const navigate = useNavigate();
  const { addDevice } = useIoTDevicesStore();
  const [loading, setLoading] = useState(false);
  const [loadingOptions, setLoadingOptions] = useState(true);
  const [equipments, setEquipments] = useState<{ value: number; label: string }[]>([]);

  const [formValues, setFormValues] = useState<Partial<IoTDevice>>({
    name: "",
    deveui: "",
    equipement: null,
  });

  useEffect(() => {
    const fetchEquipments = async () => {
      try {
        const equipmentList = await getEquipments();
        setEquipments(
          equipmentList.map((eq: { id: number; name: string }) => ({
            value: eq.id,
            label: eq.name,
          }))
        );
      } catch (error) {
        message.error("Erro ao carregar equipamentos!");
        console.error(error);
      } finally {
        setLoadingOptions(false);
      }
    };

    fetchEquipments();
  }, []);

  const handleChange = (name: string, value: any) => {
    setFormValues((prev) => ({ ...prev, [name]: value }));
  };

  const handleSubmit = async () => {
    if (!formValues.name?.trim()) {
      message.error("O nome do dispositivo Ã© obrigatÃ³rio!");
      return;
    }

    if (!formValues.deveui?.trim()) {
      message.error("O DevEUI do dispositivo Ã© obrigatÃ³rio!");
      return;
    }

    if (loading) return;

    setLoading(true);
    try {
      await addDevice({
        name: formValues.name,
        type_device: formValues.deveui, // â\234\205 Aqui convertemos para `type_device` pa
ra enviar Ã  API
        equipement: formValues.equipement !== null ? Number(formValues.equipement) : null,
      });
```

```
      message.success("Dispositivo cadastrado com sucesso!");
      navigate("/iotdevices");
    } catch (error) {
      message.error("Erro ao cadastrar dispositivo!");
      console.error(error);
    } finally {
      setLoading(false);
    }
  };


  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          <ItemHeaderCabecalho
            title="Cadastro de Dispositivo IoT"
            subTitle="Preencha os campos abaixo para cadastrar um dispositivo IoT"
          />
          <DynamicForm
  fields={[
    { name: "name", label: "Nome do Dispositivo", type: "input", required: true },
    { name: "deveui", label: "DevEUI", type: "input", required: true }, // â\234\205 ExibiÃ
§Ã£o correta
    {
      name: "equipement",
      label: "Equipamento Vinculado (Opcional)",
      type: "select",
      options: equipments,
      disabled: loadingOptions,
    },
  ]}
  values={formValues}
  onChange={handleChange}
  onSubmit={handleSubmit}
  loading={loading}
  onCancel={() => navigate("/iotdevices")}
/>
        </main>
      </div>
    </div>
  );
};

export default IoTDevicesRegister;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/iotDevices/IoTDevices.tsx =====
import React from "react";
import { PlusOutlined, FilterOutlined } from "@ant-design/icons";
import { useNavigate } from "react-router-dom";
import ItemSideBar from "../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../layout/Header/ItemHeader";
import Button from "../../components/Button/Button";
import CustomTable from "../../components/Table/Table";
import ItemHeaderCabecalho from "../../layout/Header/components/ItemHeaderCabecalho";
import { useIoTDevicesTable } from "./hooks/useIoTDevicesTable";

const IoTDevices: React.FC = () => {
  const navigate = useNavigate();
  const { columns, devices, loading } = useIoTDevicesTable();

  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
```

```
          <ItemHeaderCabecalho
            title="Dispositivos IoT"
            subTitle="Lista de dispositivos IoT cadastrados"
          />

          <section className="actions-section">
            <Button
              type="primary"
              className="primary-btn"
              icon={<PlusOutlined />}
              onClick={() => navigate("/iotdevices/register")}
            >
              Cadastrar Dispositivo
            </Button>
            <Button type="link" className="filter-btn" icon={<FilterOutlined />}>
              Filtros
            </Button>
          </section>

          <section className="table-container">
            <CustomTable columns={columns} data={devices} loading={loading} />
          </section>
        </main>
      </div>
    </div>
  );
};

export default IoTDevices;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/iotDevices/hooks/useIoTDevicesTabl
e.tsx =====
import { useEffect, useState } from "react";
import { message } from "antd";
import { useNavigate } from "react-router-dom";
import { SortOrder } from "antd/es/table/interface";
import { useIoTDevicesStore } from "../../../store/iotDevices";
import { getEquipments } from "../../../services/equipmentsService"; // ð\237\224¹ Buscar e
quipamentos
import Actions from "../../../components/actions/Actions";
import { IoTDevice } from "../../../types/IoTDevice";

export const useIoTDevicesTable = () => {
  const navigate = useNavigate();
  const { devices, fetchDevices, removeDevice } = useIoTDevicesStore();
  const [loading, setLoading] = useState(true);
  const [equipments, setEquipments] = useState<{ id: number; name: string }[]>([]);

  // ð\237\224¹ Buscar dispositivos IoT e equipamentos
  useEffect(() => {
    const fetchData = async () => {
      try {
        setLoading(true);
        await fetchDevices();

        // Buscar equipamentos para mapear o nome
        const equipmentList = await getEquipments();
        setEquipments(equipmentList);
      } catch (error) {
        console.error("Erro ao buscar dispositivos IoT:", error);
        message.error("Erro ao carregar dispositivos IoT!");
      } finally {
        setLoading(false);
      }
    };

    fetchData();
  }, []);

  // ð\237\224¹ Mapear o nome do equipamento vinculado
```

```
  const getEquipmentName = (equipmentId: number | null) => {
    if (!equipmentId) return "Nenhum";
    const equipment = equipments.find((eq) => eq.id === equipmentId);
    return equipment ? equipment.name : "Equipamento nÃ£o encontrado";
  };

  const columns = [
    {
      title: "Nome",
      dataIndex: "name",
      key: "name",
      sorter: (a: IoTDevice, b: IoTDevice) => a.name?.localeCompare(b.name || "") || 0,
      sortDirections: ["ascend", "descend"] as SortOrder[],
      render: (text: string | undefined) => <strong>{text ?? "Sem nome"}</strong>,
    },
    {
      title: "DevEUI",
      dataIndex: "type_device", // â\234\205 Pegamos da API, mas mostramos como DevEUI no f
ront
      key: "type_device",
      render: (text: string | undefined) => text ?? "NÃ£o informado",
    },
    {
      title: "Equipamento Associado",
      dataIndex: "equipement",
      key: "equipement",
      render: (equipement: number | null) => getEquipmentName(equipement),
    },
    {
      title: "AÃ§Ãµes",
      key: "actions",
      render: (_: any, record: IoTDevice) => (
        <Actions
          onEdit={() => navigate(`/iotdevices/edit/${record.id}`)}
          onDelete={async () => {
            if (record.id) {
              await removeDevice(record.id);
              await fetchDevices();
            }
          }}
        />
      ),
    },
  ];


  return { columns, devices, loading };
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/sections/Sections.tsx =====
import React from "react";
import { PlusOutlined, FilterOutlined } from "@ant-design/icons";
import { useNavigate } from "react-router-dom";
import ItemSideBar from "../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../layout/Header/ItemHeader";
import Button from "../../components/Button/Button";
import CustomTable from "../../components/Table/Table";
import ItemHeaderCabecalho from "../../layout/Header/components/ItemHeaderCabecalho";
import { useSectionsTable } from "./hooks/useSectionsTable";

const Sections: React.FC = () => {
  const navigate = useNavigate();
  const { columns, sections, loading } = useSectionsTable();

  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
```

```
          <ItemHeaderCabecalho
            title="SeÃ§Ãµes"
            subTitle="Lista de seÃ§Ãµes cadastradas"
          />

          <section className="actions-section">
            <Button
              type="primary"
              className="primary-btn"
              icon={<PlusOutlined />}
              onClick={() => navigate("/sections/register")}
            >
              Cadastrar SeÃ§Ã£o
            </Button>
            <Button type="link" className="filter-btn" icon={<FilterOutlined />}>
              Filtros
            </Button>
          </section>

          <section className="table-container">
            <CustomTable columns={columns} data={sections} loading={loading} />
          </section>
        </main>
      </div>
    </div>
  );
};

export default Sections;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/sections/hooks/useSectionsTable.ts
x =====
import { useEffect, useState } from "react";
import { message } from "antd";
import { useNavigate } from "react-router-dom";
import { SortOrder } from "antd/es/table/interface";
import { useSectionsStore } from "../../../store/sections";
import Actions from "../../../components/actions/Actions";
import { SectionItem } from "../../../types/sections";

export const useSectionsTable = () => {
  const navigate = useNavigate();
  const { sections, fetchSections, removeSection } = useSectionsStore();
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchData = async () => {
      try {
        setLoading(true);
        await fetchSections();
      } catch (error) {
        console.error("Erro ao buscar seÃ§Ãµes:", error);
        message.error("Erro ao carregar seÃ§Ãµes!");
      } finally {
        setLoading(false);
      }
    };

    fetchData();
  }, []);

  const columns = [
    {
      title: "Nome",
      dataIndex: "name",
      key: "name",
      sorter: (a: SectionItem, b: SectionItem) => a.name.localeCompare(b.name),
      sortDirections: ["ascend", "descend"] as SortOrder[],
      render: (text: string) => <strong>{text}</strong>,
    },
```

```
    {
      title: "DescriÃ§Ã£o",
      dataIndex: "description",
      key: "description",
      render: (text: string | null) => text || "NÃ£o informado",
    },
    {
      title: "Consumo Estimado",
      dataIndex: "estimated_consumption",
      key: "estimated_consumption",
      render: (value: number) => `${value} kWh`,
    },
    {
      title: "AÃ§Ãµes",
      key: "actions",
      render: (_: any, record: SectionItem) => (
        <Actions
          onEdit={() => navigate(`/sections/edit/${record.id}`)}
          onDelete={async () => {
            if (record.id) {
              await removeSection(record.id);
              await fetchSections();
            }
          }}
        />
      ),
    },
  ];

  return { columns, sections, loading };
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/products/ProductsRegister/Register
.tsx =====
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import { message } from "antd";
import ItemSideBar from "../../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../../layout/Header/components/ItemHeaderCabecalho";
import { useProductsStore } from "../../../store/products";
import DynamicForm from "../../../components/form/DynamicForm";
import type { FormField } from "../../../components/form/formTypes";

const ProductsRegister: React.FC = () => {
  const navigate = useNavigate();
  const { createProduct } = useProductsStore();
  const [loading, setLoading] = useState(false);
  const [values, setValues] = useState<Record<string, any>>({
    name: "",
    description: "",
    photo: null,
  });

  // ConfiguraÃ§Ã£o dos campos do formulÃ¡rio
  const formFields: FormField[] = [
    { name: "name", label: "Nome do Produto", type: "input", required: true },
    { name: "description", label: "DescriÃ§Ã£o", type: "textarea", required: true },
    { name: "photo", label: "Imagem do Produto", type: "upload", required: false },
  ];

  // Atualiza valores do formulÃ¡rio
  const handleChange = (name: string, value: any) => {
    setValues((prev) => ({ ...prev, [name]: value }));
  };

  // Envia os dados para a API
  const handleSubmit = async () => {
    if (!values.name || !values.description) {
      message.error("Preencha os campos obrigatÃ³rios!");
```

```
      return;
    }

    try {
      setLoading(true);
      const formData = new FormData();
      formData.append("name", values.name);
      formData.append("description", values.description);

      if (values.photo instanceof File) {
        formData.append("photo", values.photo);
      }

      await createProduct(formData);


      if (!loading) {
        message.success("Produto cadastrado com sucesso!");
      }

      navigate("/products");
    } catch (error) {
      message.error("Erro ao cadastrar produto!");
      console.error(error);
    } finally {
      setLoading(false);
    }
  };


  const handleCancel = () => {
    navigate("/products");
  };

  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          <ItemHeaderCabecalho
            title="Cadastro de Produtos"
            subTitle="Preencha os campos abaixo para cadastrar um produto"
          />

          {/* ð\237\224¹ Passando `onCancel` corretamente */}
          <DynamicForm
            fields={formFields}
            values={values}
            onChange={handleChange}
            onSubmit={handleSubmit}
            onCancel={handleCancel}
            loading={loading}
          />
        </main>
      </div>
    </div>
  );
};

export default ProductsRegister;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/products/Products.tsx =====
import React from "react";
import { PlusOutlined, FilterOutlined } from "@ant-design/icons";
import { useNavigate } from "react-router-dom";
import ItemSideBar from "../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../layout/Header/ItemHeader";
import Button from "../../components/Button/Button";
import CustomTable from "../../components/Table/Table";
```

```
import ItemHeaderCabecalho from "../../layout/Header/components/ItemHeaderCabecalho";
import { useProductsTable } from "./hooks/useProductsTable";

const Products: React.FC = () => {
  const navigate = useNavigate();
  const { columns, products, loading } = useProductsTable();

  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">

          <ItemHeaderCabecalho
            title="Produtos"
            subTitle="Lista de produtos jÃ¡ cadastrados"
          />


          <section className="actions-section">
            <Button
              type="primary"
              className="primary-btn"
              icon={<PlusOutlined />}
              onClick={() => navigate("/products/register")}
            >
              Cadastrar produto
            </Button>
            <Button type="link" className="filter-btn" icon={<FilterOutlined />}>
              Filtros
            </Button>
          </section>


          <section className="table-container">
            <CustomTable columns={columns} data={products} loading={loading} />
          </section>
        </main>
      </div>
    </div>
  );
};

export default Products;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/products/components/EditProducts.t
sx =====
import React, { useEffect, useState } from "react";
import { useNavigate, useParams } from "react-router-dom";
import { message } from "antd";
import ItemSideBar from "../../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../../layout/Header/components/ItemHeaderCabecalho";
import { useProductsStore } from "../../../store/products";
import DynamicForm from "../../../components/form/DynamicForm";
import type { FormField } from "../../../components/form/formTypes";

const EditProducts: React.FC = () => {
  const navigate = useNavigate();
  const { id } = useParams(); // Recebe o ID do produto para editar
  const { updateProduct, fetchProductById } = useProductsStore();
  const [loading, setLoading] = useState(false);
  const [values, setValues] = useState<Record<string, any>>({
    name: "",
    description: "",
    photo: null,
  });

  useEffect(() => {
```

```
  if (id) {
    const loadProduct = async () => {
      setLoading(true);
      try {
        const product = await fetchProductById(Number(id));
        setValues({
          name: product.name,
          description: product.description,
          photo: product.photo || null,
        });
      } catch (error) {
        message.error("Erro ao carregar o produto!");
      } finally {
        setLoading(false);
      }
    };
    loadProduct();
  }
}, [id]);

const formFields: FormField[] = [
  { name: "name", label: "Nome do Produto", type: "input", required: true },
  { name: "description", label: "DescriÃ§Ã£o", type: "textarea", required: true },
  { name: "photo", label: "Imagem do Produto", type: "upload", required: false },
];

const handleChange = (name: string, value: any) => {
  setValues((prev) => ({ ...prev, [name]: value }));
};

const handleSubmit = async () => {
  if (!values.name || !values.description) {
    message.error("Preencha os campos obrigatÃ³rios!");
    return;
  }

  try {
    setLoading(true);
    const formData = new FormData();
    formData.append("name", values.name);
    formData.append("description", values.description);

    if (values.photo instanceof File) {
      formData.append("photo", values.photo);
    }

    await updateProduct(Number(id), formData);
    message.success("Produto editado com sucesso!");
    navigate("/products");
  } catch (error) {
    message.error("Erro ao editar produto!");
    console.error(error);
  } finally {
    setLoading(false);
  }
};

const handleCancel = () => {
  navigate("/products"); // Redireciona para a listagem de produtos ao cancelar
};

return (
  <div className="layout-container">
    <ItemSideBar />
    <div className="content-container">
      <ItemHeader />
      <main className="content">
        <ItemHeaderCabecalho
          title="Editar Produto"
```

```
            subTitle="Atualize os dados do produto abaixo"
          />

          <DynamicForm
            fields={formFields}
            values={values}
            onChange={handleChange}
            onSubmit={handleSubmit}
            onCancel={handleCancel}  // ð\237\224¹ Agora o botÃ£o de cancelar serÃ¡ renderi
zado
            loading={loading}
          />
        </main>
      </div>
    </div>
  );
};

export default EditProducts;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/products/hooks/useProductsTable.ts
x =====
import { useEffect, useState } from "react";
import { message } from "antd";
import { useNavigate } from "react-router-dom";
import { SortOrder } from "antd/es/table/interface";
import { useProductsStore } from "../../../store/products";
import Actions from "../../../components/actions/Actions";
import { ProductItem } from "../../../types/products";

export const useProductsTable = () => {
  const navigate = useNavigate();
  const { products, fetchProducts, deleteProduct } = useProductsStore();
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchProductsFromAPI = async () => {
      try {
        setLoading(true);
        await fetchProducts();
      } catch (error) {
        console.error("Erro ao buscar produtos:", error);
        message.error("Erro ao carregar os produtos!");
      } finally {
        setLoading(false);
      }
    };

    fetchProductsFromAPI();
  }, []);

  const columns = [
    {
      title: "Foto",
      dataIndex: "photo",
      key: "photo",
      render: (photo: any) =>
        typeof photo === "string" && photo.startsWith("http") ? (
          <img
            src={photo}
            alt="Produto"
            style={{
              width: "50px",
              height: "50px",
              objectFit: "cover",
              borderRadius: "5px",
            }}
          />
        ) : photo ? (
          <img
```

```
            src={`http://inova-sistemas.ddns.net:20163${photo}`}
            alt="Produto"
            style={{
              width: "540px",
              height: "540px",
              objectFit: "cover",
              borderRadius: "5px",
            }}
          />
        ) : (
          <span>Sem imagem</span>
        ),
    },
    {
      title: "Nome",
      dataIndex: "name",
      key: "name",
      sorter: (a: ProductItem, b: ProductItem) => a.name.localeCompare(b.name),
      sortDirections: ["ascend", "descend"] as SortOrder[],
      render: (text: string | undefined) => (
        <strong>{text ?? "Sem nome"}</strong>
      ),
    },
    {
      title: "DescriÃ§Ã£o",
      dataIndex: "description",
      key: "description",
      render: (text: string | undefined) => (
        <span>{text ?? "Sem descriÃ§Ã£o"}</span>
      ),
    },
    {
      title: "AÃ§Ãµes",
      key: "actions",
      render: (_: any, record: ProductItem) => (
        <Actions
          onEdit={() => navigate(`/products/edit/${record.id}`)}
          onDelete={async () => {
            if (record.id) {
              await deleteProduct(Number(record.id));
            }
          }}
        />
      ),
    },
  ];

  return { columns, products, loading };
};

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/missions/components/EditMission.ts
x =====
// src/pages/home/components/mission/MissionEdit.tsx
import React, { useEffect, useState } from "react";
import { useNavigate, useParams } from "react-router-dom";
import {
  Form,
  Input,
  InputNumber,
  Select,
  DatePicker,
  Switch,
  Button,
  Row,
  Col,
  Card,
  message,
} from "antd";
import dayjs, { Dayjs } from "dayjs";
```

```
import { useMissionStore } from "../../../store/missions";
import { useMonitoringStore } from "../../../store/monitoringStore";
import { useProductsStore } from "../../../store/products";
import ItemSideBar from "../../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../../layout/Header/components/ItemHeaderCabecalho";

const { Option } = Select;

const MissionEdit: React.FC = () => {
  const [form] = Form.useForm();
  const navigate = useNavigate();
  const { id } = useParams<{ id: string }>();
  const { missions, fetchMissions, updateMission } = useMissionStore();
  const { monitorings, fetchMonitorings } = useMonitoringStore();
  const { products, fetchProducts } = useProductsStore();
  const [loading, setLoading] = useState(false);

  // carrega dados
  useEffect(() => {
    fetchMissions();
    fetchMonitorings();
    fetchProducts();
  }, []);

  // preenche o formulÃ¡rio quando missions estiver disponÃvel
  useEffect(() => {
    const mid = Number(id);
    const mission = missions.find((m) => m.id === mid);
    if (mission) {
      form.setFieldsValue({
        name: mission.name,
        description: mission.description,
        energy_meta: mission.energy_meta,
        nansen_coins: mission.nansen_coins,
        quantity_xp: mission.quantity_xp,
        status: mission.status,
        date_start: mission.date_start ? dayjs(mission.date_start) : null,
        date_end: mission.date_end ? dayjs(mission.date_end) : null,
        monitoring: mission.monitoring ?? undefined,
        is_order_production: mission.is_order_production,
        product: mission.product ?? undefined,
        order_production: mission.order_production,
        quantity_product: mission.quantity_product,
      });
    }
  }, [missions, form, id]);

  const onFinish = async (values: any) => {
    setLoading(true);
    try {
      await updateMission(Number(id), {
        name: values.name,
        description: values.description,
        energy_meta: Number(values.energy_meta),
        nansen_coins: Number(values.nansen_coins),
        quantity_xp: Number(values.quantity_xp),
        status: values.status,
        date_start: values.date_start
          ? (values.date_start as Dayjs).toISOString()
          : null,
        date_end: values.date_end
          ? (values.date_end as Dayjs).toISOString()
          : null,
        is_order_production: values.is_order_production,
        product: values.is_order_production ? values.product : null,
        order_production: values.is_order_production
          ? Number(values.order_production)
          : 0,
```

```
      quantity_product: values.is_order_production
        ? Number(values.quantity_product)
        : 0,
      monitoring: values.monitoring ?? null,
    });
    message.success("MissÃ£o atualizada com sucesso!");
    navigate("/missions");
  } catch (err) {
    console.error(err);
    message.error("Erro ao atualizar missÃ£o.");
  } finally {
    setLoading(false);
  }
};

return (
  <div className="layout-container">
    <ItemSideBar />
    <div className="content-container">
      <ItemHeader />
      <main className="content">
        <ItemHeaderCabecalho
          title="Editar MissÃ£o"
          subTitle="Atualize os campos necessÃ¡rios"
        />

        <Card>
          <Form
            form={form}
            layout="vertical"
            onFinish={onFinish}
            initialValues={{ status: "Pendente", is_order_production: false }}
          >
            {/* Nome e DescriÃ§Ã£o */}
            <Row gutter={16}>
              <Col span={12}>
                <Form.Item
                  name="name"
                  label="Nome"
                  rules={[{ required: true, message: "Informe o nome" }]}
                >
                  <Input />
                </Form.Item>
              </Col>
              <Col span={12}>
                <Form.Item
                  name="description"
                  label="DescriÃ§Ã£o"
                  rules={[{ required: true, message: "Informe a descriÃ§Ã£o" }]}
                >
                  <Input.TextArea />
                </Form.Item>
              </Col>
            </Row>

            {/* Valores numÃ©ricos */}
            <Row gutter={16}>
              <Col span={8}>
                <Form.Item
                  name="energy_meta"
                  label="Meta de Energia"
                  rules={[
                    { required: true, message: "Informe a meta de energia" },
                  ]}
                >
                  <InputNumber style={{ width: "100%" }} />
                </Form.Item>
              </Col>
              <Col span={8}>
```

```
    <Form.Item
      name="nansen_coins"
      label="Nansen Coins"
      rules={[
        { required: true, message: "Informe os Nansen Coins" },
      ]}
    >
      <InputNumber style={{ width: "100%" }} />
    </Form.Item>
  </Col>
  <Col span={8}>
    <Form.Item
      name="quantity_xp"
      label="Quantidade XP"
      rules={[
        { required: true, message: "Informe a quantidade XP" },
      ]}
    >
      <InputNumber style={{ width: "100%" }} />
    </Form.Item>
  </Col>
</Row>

{/* Status e Datas */}
<Row gutter={16}>
  <Col span={8}>
    <Form.Item
      name="status"
      label="Status"
      rules={[{ required: true, message: "Selecione o status" }]}
    >
      <Select>
        <Option value="Pendente">Pendente</Option>
        <Option value="Em Andamento">Em andamento</Option>
        <Option value="Finalizada">Finalizada</Option>
      </Select>
    </Form.Item>
  </Col>
  <Col span={8}>
    <Form.Item name="date_start" label="Data de InÃ-cio">
      <DatePicker style={{ width: "100%" }} showTime />
    </Form.Item>
  </Col>
  <Col span={8}>
    <Form.Item name="date_end" label="Data de Fim">
      <DatePicker style={{ width: "100%" }} showTime />
    </Form.Item>
  </Col>
</Row>

{/* Monitoramento */}
<Row gutter={16}>
  <Col span={12}>
    <Form.Item name="monitoring" label="Monitoramento">
      <Select allowClear placeholder="Selecione">
        {monitorings.map((m) => (
          <Option key={m.id} value={m.id}>
            {m.name}
          </Option>
        ))}
      </Select>
    </Form.Item>
  </Col>
</Row>

{/* Ã\211 Ordem de ProduÃ§Ã£o? */}
<Row gutter={16}>
  <Col span={12}>
    <Form.Item
```

```
      name="is_order_production"
      label="Ã\211 Ordem de ProduÃ§Ã£o?"
      valuePropName="checked"
    >
      <Switch />
    </Form.Item>
  </Col>
</Row>

{/* Campos extras OP */}
<Form.Item noStyle shouldUpdate>
  {() =>
    form.getFieldValue("is_order_production") && (
      <Row gutter={16}>
        <Col span={8}>
          <Form.Item
            name="product"
            label="Produto"
            rules={[
              { required: true, message: "Selecione o produto" },
            ]}
          >
            <Select allowClear placeholder="Selecione">
              {products.map((p) => (
                <Option key={p.id} value={p.id}>
                  {p.name}
                </Option>
              ))}
            </Select>
          </Form.Item>
        </Col>
        <Col span={8}>
          <Form.Item
            name="order_production"
            label="NÂº da Ordem"
            rules={[
              {
                required: true,
                message: "Informe o nÂº da ordem",
              },
            ]}
          >
            <Input />
          </Form.Item>
        </Col>
        <Col span={8}>
          <Form.Item
            name="quantity_product"
            label="Quantidade"
            rules={[
              {
                required: true,
                message: "Informe a quantidade",
              },
            ]}
          >
            <InputNumber style={{ width: "100%" }} />
          </Form.Item>
        </Col>
      </Row>
    )
  }
</Form.Item>

{/* AÃ§Ãµes */}
<Form.Item style={{ textAlign: "right", marginTop: 16 }}>
  <Button
    style={{ marginRight: 8 }}
    onClick={() => navigate("/missions")}
```

```
                >
                  Cancelar
                </Button>
                <Button type="primary" htmlType="submit" loading={loading}>
                  Salvar
                </Button>
              </Form.Item>
            </Form>
          </Card>
        </main>
      </div>
    </div>
  );
};

export default MissionEdit;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/missions/missionregister/Register.
tsx =====
// src/pages/home/components/mission/MissionRegister.tsx
import React, { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
import {
  Form,
  Input,
  InputNumber,
  Select,
  DatePicker,
  Switch,
  Button,
  Row,
  Col,
  Card,
  message,
} from "antd";
import { Dayjs } from "dayjs";
import { useMissionStore } from "../../../store/missions";
import { useMonitoringStore } from "../../../store/monitoringStore";
import { useQuizStore } from "../../../store/quizzes";
import { useProductsStore } from "../../../store/products";
import ItemSideBar from "../../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../../layout/Header/components/ItemHeaderCabecalho";

const { Option } = Select;

const MissionRegister: React.FC = () => {
  const [form] = Form.useForm();
  const navigate = useNavigate();
  const { createMission } = useMissionStore();
  const { monitorings, fetchMonitorings } = useMonitoringStore();
  const { quizzes, fetchQuizzes } = useQuizStore();
  const { products, fetchProducts } = useProductsStore();
  const [loading, setLoading] = useState(false);

  useEffect(() => {
    fetchMonitorings();
    fetchQuizzes();
    fetchProducts();
  }, []);

  const onFinish = async (values: any) => {
    setLoading(true);
    try {
      await createMission({
        name: values.name,
        description: values.description,
        energy_meta: Number(values.energy_meta),
        nansen_coins: Number(values.nansen_coins),
```

```
      quantity_xp: Number(values.quantity_xp),
      status: values.status,
      date_start: values.date_start
        ? (values.date_start as Dayjs).toISOString()
        : null,
      date_end: values.date_end
        ? (values.date_end as Dayjs).toISOString()
        : null,
      is_order_production: values.is_order_production,
      order_production: values.is_order_production
        ? Number(values.order_production)
        : 0,
      product: values.is_order_production ? values.product : null,
      quantity_product: values.is_order_production
        ? Number(values.quantity_product)
        : 0,
      monitoring: values.monitoring ?? null,
      quiz: values.quiz ?? null,
    });
    message.success("MissÃ£o cadastrada com sucesso!");
    navigate("/missions");
  } catch (err) {
    console.error(err);
    message.error("Erro ao cadastrar missÃ£o.");
  } finally {
    setLoading(false);
  }
};

return (
  <div className="layout-container">
    <ItemSideBar />
    <div className="content-container">
      <ItemHeader />
      <main className="content">
        <ItemHeaderCabecalho
          title="Cadastro de MissÃ£o"
          subTitle="Preencha os campos abaixo para cadastrar uma missÃ£o"
        />

        <Card>
          <Form
            form={form}
            layout="vertical"
            onFinish={onFinish}
            initialValues={{
              status: "Pendente",
              is_order_production: false,
            }}
          >
            {/* Nome e DescriÃ§Ã£o */}
            <Row gutter={16}>
              <Col span={12}>
                <Form.Item
                  name="name"
                  label="Nome"
                  rules={[{ required: true, message: "Informe o nome" }]}
                >
                  <Input placeholder="Digite o nome da missÃ£o" />
                </Form.Item>
              </Col>
              <Col span={12}>
                <Form.Item
                  name="description"
                  label="DescriÃ§Ã£o"
                  rules={[{ required: true, message: "Informe a descriÃ§Ã£o" }]}
                >
                  <Input.TextArea placeholder="Digite a descriÃ§Ã£o" />
                </Form.Item>
```

```
  </Col>
</Row>

{/* Valores numÃ©ricos */}
<Row gutter={16}>
  <Col span={8}>
    <Form.Item
      name="energy_meta"
      label="Meta de Energia"
      rules={[
        { required: true, message: "Informe a meta de energia" },
      ]}
    >
      <InputNumber style={{ width: "100%" }} />
    </Form.Item>
  </Col>
  <Col span={8}>
    <Form.Item
      name="nansen_coins"
      label="Nansen Coins"
      rules={[
        { required: true, message: "Informe os Nansen Coins" },
      ]}
    >
      <InputNumber style={{ width: "100%" }} />
    </Form.Item>
  </Col>
  <Col span={8}>
    <Form.Item
      name="quantity_xp"
      label="Quantidade XP"
      rules={[
        { required: true, message: "Informe a quantidade XP" },
      ]}
    >
      <InputNumber style={{ width: "100%" }} />
    </Form.Item>
  </Col>
</Row>

{/* Status e Datas */}
<Row gutter={16}>
  <Col span={8}>
    <Form.Item
      name="status"
      label="Status"
      rules={[{ required: true, message: "Selecione o status" }]}
    >
      <Select>
        <Option value="Pendente">Pendente</Option>
        <Option value="Em Andamento">Em andamento</Option>
        <Option value="Finalizada">Finalizada</Option>
      </Select>
    </Form.Item>
  </Col>
  <Col span={8}>
    <Form.Item name="date_start" label="Data de InÃ-cio">
      <DatePicker style={{ width: "100%" }} showTime />
    </Form.Item>
  </Col>
  <Col span={8}>
    <Form.Item name="date_end" label="Data de Fim">
      <DatePicker style={{ width: "100%" }} showTime />
    </Form.Item>
  </Col>
</Row>

{/* Monitoramento e Quiz */}
<Row gutter={16}>
```

```
  <Col span={12}>
    <Form.Item name="monitoring" label="Monitoramento">
      <Select allowClear placeholder="Selecione">
        {monitorings.map((m) => (
          <Option key={m.id} value={m.id}>
            {m.name}
          </Option>
        ))}
      </Select>
    </Form.Item>
  </Col>
  <Col span={12}>
    <Form.Item name="quiz" label="Quiz">
      <Select allowClear placeholder="Selecione">
        {quizzes.map((q) => (
          <Option key={q.id} value={q.id}>
            {q.name}
          </Option>
        ))}
      </Select>
    </Form.Item>
  </Col>
</Row>

{/* Ã\211 Ordem de ProduÃ§Ã£o? */}
<Row gutter={16}>
  <Col span={12}>
    <Form.Item
      name="is_order_production"
      label="Ã\211 Ordem de ProduÃ§Ã£o?"
      valuePropName="checked"
    >
      <Switch />
    </Form.Item>
  </Col>
</Row>

{/* Campos extras OP */}
<Form.Item noStyle shouldUpdate>
  {() =>
    form.getFieldValue("is_order_production") && (
      <Row gutter={16}>
        <Col span={8}>
          <Form.Item
            name="product"
            label="Produto"
            rules={[
              { required: true, message: "Selecione o produto" },
            ]}
          >
            <Select allowClear placeholder="Selecione">
              {products.map((p) => (
                <Option key={p.id} value={p.id}>
                  {p.name}
                </Option>
              ))}
            </Select>
          </Form.Item>
        </Col>
        <Col span={8}>
          <Form.Item
            name="order_production"
            label="NÂº da Ordem"
            rules={[
              {
                required: true,
                message: "Informe o nÂº da ordem",
              },
            ]}
```

```
                      >
                        <Input />
                      </Form.Item>
                    </Col>
                    <Col span={8}>
                      <Form.Item
                        name="quantity_product"
                        label="Quantidade"
                        rules={[
                          {
                            required: true,
                            message: "Informe a quantidade",
                          },
                        ]}
                      >
                        <InputNumber style={{ width: "100%" }} />
                      </Form.Item>
                    </Col>
                  </Row>
                )
              }
            </Form.Item>

            {/* AÃ§Ãµes */}
            <Form.Item style={{ textAlign: "right", marginTop: 16 }}>
              <Button
                style={{ marginRight: 8 }}
                onClick={() => navigate("/missions")}
              >
                Cancelar
              </Button>
              <Button type="primary" htmlType="submit" loading={loading}>
                Cadastrar
              </Button>
            </Form.Item>
          </Form>
        </Card>
      </main>
    </div>
  </div>
  );
};

export default MissionRegister;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/missions/Missions.tsx =====
// src/pages/missions/Missions.tsx
import React, { useEffect } from "react";
import { Table, Button } from "antd";
import { PlusOutlined } from "@ant-design/icons";
import { useNavigate } from "react-router-dom";
import { useMissionStore } from "../../store/missions";
import { useMonitoringStore } from "../../store/monitoringStore";
import { useMissionsTable } from "./hooks/useMissionsTable";
import ItemSideBar from "../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../layout/Header/components/ItemHeaderCabecalho";

const Missions: React.FC = () => {
  const navigate = useNavigate();
  const { fetchMissions } = useMissionStore();
  const { fetchMonitorings } = useMonitoringStore();
  const { columns, missions, loading } = useMissionsTable();

  useEffect(() => {
    (async () => {
      await fetchMissions();
      await fetchMonitorings();
    })();
```

```
  }, [fetchMissions, fetchMonitorings]);

  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          <ItemHeaderCabecalho
            title="MissÃµes"
            subTitle="Lista de missÃµes existentes"
          />

          <section className="actions-section" style={{ marginBottom: 16 }}>
            <Button
              type="primary"
              icon={<PlusOutlined />}
              onClick={() => navigate("/missions/register")}
            >
              Criar MissÃ£o
            </Button>
          </section>

          <section className="table-container">
            <Table
              columns={columns}
              dataSource={missions}
              loading={loading}
              rowKey="id"
            />
          </section>
        </main>
      </div>
    </div>
  );
};

export default Missions;

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/missions/hooks/useMissionsTable.ts
x =====
// src/pages/missions/hooks/useMissionsTable.tsx
import { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
import { ColumnsType } from "antd/es/table";
import { Button, Select, message, Tag } from "antd";
import { EditOutlined, DeleteOutlined } from "@ant-design/icons";
import { useMissionStore } from "../../../store/missions";
import { useMonitoringStore } from "../../../store/monitoringStore";
import { MissionItem } from "../../../types/missions";

export const useMissionsTable = () => {
  const navigate = useNavigate();
  const {
    missions,
    fetchMissions,
    deleteMission,
    associateMissionToMonitoring,
  } = useMissionStore();
  const { monitorings, fetchMonitorings } = useMonitoringStore();
  const [loading, setLoading] = useState(false);

  useEffect(() => {
    (async () => {
      setLoading(true);
      await fetchMissions();
      await fetchMonitorings();
      setLoading(false);
    })();
```

```
}, [fetchMissions, fetchMonitorings]);

const handleDelete = async (id: number) => {
  try {
    await deleteMission(id);
    message.success("MissÃ£o excluÃ-da com sucesso!");
  } catch {
    message.error("Falha ao excluir a missÃ£o.");
  }
};

const handleAssociateMonitoring = async (
  missionId: number,
  monitoringId: number | null
) => {
  try {
    await associateMissionToMonitoring(missionId, monitoringId);
    message.success("Monitoramento atualizado com sucesso!");
    await fetchMissions();
  } catch {
    message.error("Falha ao atualizar o monitoramento.");
  }
};

const columns: ColumnsType<MissionItem> = [
  {
    title: "Nome da MissÃ£o",
    dataIndex: "name",
    key: "name",
  },
  {
    title: "DescriÃ§Ã£o",
    dataIndex: "description",
    key: "description",
  },
  {
    title: "Tipo",
    dataIndex: "is_order_production",
    key: "tipo",
    render: (isOp: boolean) =>
      isOp ? (
        <Tag color="blue">Ordem de ProduÃ§Ã£o</Tag>
      ) : (
        <Tag color="default">Normal</Tag>
      ),
  },
  {
    title: "Monitoramento Associado",
    dataIndex: "monitoring",
    key: "monitoring",
    render: (monitoringId: number | null) =>
      monitoringId
        ? monitorings.find((m) => m.id === monitoringId)?.name ??
          `#${monitoringId}`
        : "Nenhum",
  },
  {
    title: "AÃ§Ãµes",
    key: "actions",
    render: (_: any, record: MissionItem) => (
      <>
        <Button
          type="primary"
          icon={<EditOutlined />}
          onClick={() => navigate(`/missions/edit/${record.id}`)}
          style={{ marginRight: 8 }}
        >
          Editar
        </Button>
```

```
          <Select<number>
            placeholder="Associar Monitoramento"
            style={{ width: 200, marginRight: 8 }}
            allowClear
            value={record.monitoring ?? undefined}
            onChange={(value) =>
              handleAssociateMonitoring(record.id, value ?? null)
            }
          >
            {monitorings.map((m) => (
              <Select.Option key={m.id} value={m.id}>
                {m.name}
              </Select.Option>
            ))}
          </Select>

          <Button
            type="primary"
            danger
            icon={<DeleteOutlined />}
            onClick={() => handleDelete(record.id)}
          >
            Excluir
          </Button>
        </>
      ),
    },
  ];

  return { columns, missions, loading };
};

===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/quizzes/Quizzes.tsx =====
import React, { useEffect } from "react";
import { PlusOutlined, FilterOutlined } from "@ant-design/icons";
import { useNavigate } from "react-router-dom";
import ItemSideBar from "../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../layout/Header/ItemHeader";
import Button from "../../components/Button/Button";
import CustomTable from "../../components/Table/Table";
import ItemHeaderCabecalho from "../../layout/Header/components/ItemHeaderCabecalho";
import { useQuizzesTable } from "./hooks/useQuizzesTable";

const Quizzes: React.FC = () => {
  const navigate = useNavigate();
  const { columns, quizzes, loading, fetchQuizzes } = useQuizzesTable();

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

  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          {/* CabeÃ§alho da pÃ¡gina */}
          <ItemHeaderCabecalho
            title="Quizzes"
            subTitle="Lista de quizzes jÃ¡ cadastrados"
          />

          {/* BotÃµes de aÃ§Ã£o */}
          <section className="actions-section">
            <Button
              type="primary"
              className="primary-btn"
```

```
              icon={<PlusOutlined />}
              onClick={() => navigate("/quizzes/register")}
            >
              Criar Quizz
            </Button>
            <Button type="link" className="filter-btn" icon={<FilterOutlined />}>
              Filtros
            </Button>
          </section>

          {/* Tabela utilizando o hook de Quizzes */}
          <section className="table-container">
            <CustomTable columns={columns} data={quizzes} loading={loading} />
          </section>
        </main>
      </div>
    </div>
  );
};

export default Quizzes;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/quizzes/quizregister/Register.tsx
=====
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import { message, Card, Input, Radio, Button, Divider } from "antd";
import { useQuizStore } from "../../../store/quizzes";
import ItemSideBar from "../../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../../layout/Header/components/ItemHeaderCabecalho";

const Register: React.FC = () => {
  const navigate = useNavigate();
  const { createQuiz } = useQuizStore();
  const [loading, setLoading] = useState(false);
  const [formValues, setFormValues] = useState({
    name: "",
    description: "",
    questions: [
      {
        text: "",
        responses: [
          { text: "", is_correct: false },
          { text: "", is_correct: false },
          { text: "", is_correct: false },
          { text: "", is_correct: false },
        ],
      },
    ],
  });

  const handleChange = (name: string, value: any) => {
    setFormValues((prev) => ({ ...prev, [name]: value }));
  };

  const handleQuestionChange = (index: number, field: string, value: any) => {
    setFormValues((prev) => {
      const updatedQuestions = [...prev.questions];
      updatedQuestions[index] = { ...updatedQuestions[index], [field]: value };
      return { ...prev, questions: updatedQuestions };
    });
  };

  const handleResponseChange = (qIndex: number, rIndex: number, field: string, value: any)
=> {
    setFormValues((prev) => {
      const  updatedQuestions = [...prev.questions];

      if (field === "is_correct" && value === true) {
```

```
        // Garante que apenas um radio button serÃ¡ marcado como correto
        updatedQuestions[qIndex].responses = updatedQuestions[qIndex].responses.map((resp,
i) => ({
          ...resp,
          is_correct: i === rIndex, // Apenas o item clicado serÃ¡ verdadeiro
        }));
      } else {
        updatedQuestions[qIndex].responses[rIndex] = {
          ...updatedQuestions[qIndex].responses[rIndex],
          [field]: value,
        };
      }

      return { ...prev, questions: updatedQuestions };
    });
  };

  const addQuestion = () => {
    setFormValues((prev) => ({
      ...prev,
      questions: [
        ...prev.questions,
        {
          text: "",
          responses: [
            { text: "", is_correct: false },
            { text: "", is_correct: false },
            { text: "", is_correct: false },
            { text: "", is_correct: false },
          ],
        },
      ],
    }));
  };

  // FunÃ§Ã£o para remover uma pergunta
  const removeQuestion = (index: number) => {
    setFormValues((prev) => {
      const updatedQuestions = prev.questions.filter((_, i) => i !== index);
      return { ...prev, questions: updatedQuestions };
    });
  };

  const handleSubmit = async () => {
    if (!formValues.name.trim() || !formValues.description.trim()) {
      message.error("Todos os campos sÃ£o obrigatÃ³rios!");
      return;
    }

    if (formValues.questions.some(q => !q.text.trim() || q.responses.some(r => !r.text.trim
()))) {
      message.error("Preencha todas as perguntas e respostas!");
      return;
    }

    if (loading) return;

    setLoading(true);
    try {
      await createQuiz(formValues);
      message.success("Quiz cadastrado com sucesso!");
      navigate("/quizzes");
    } catch (error) {
      message.error("Erro ao cadastrar quiz!");
      console.error(error);
    } finally {
      setLoading(false);
    }
  };
```

```
  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content" style={{ display: "flex", flexDirection: "column", justif
yContent: "center", alignItems: "center" }}>
          <ItemHeaderCabecalho title="Cadastro de Quiz" subTitle="Preencha os campos abaixo
 para cadastrar um Quiz" />

          <Card style={{ minWidth: "100%" }}>
            <label>Nome do Quiz</label>
            <Input
              placeholder="Digite o nome do quiz"
              value={formValues.name}
              onChange={(e) => handleChange("name", e.target.value)}
            />

            <label>DescriÃ§Ã£o</label>
            <Input.TextArea
              placeholder="Digite a descriÃ§Ã£o"
              value={formValues.description}
              onChange={(e) => handleChange("description", e.target.value)}
            />

            <label>Perguntas</label>
            {formValues.questions.map((question, qIndex) => (
              <div key={qIndex} style={{ marginBottom: 16 }}>
                {qIndex > 0 && <Divider dashed />}
                <hr />
                <div style={{ display: "flex", justifyContent: "space-between", alignItems:
 "center" }}>
                  <h3>Pergunta {qIndex + 1}</h3>
                  <Button
                    type="link"
                    danger
                    onClick={() => removeQuestion(qIndex)}
                  >
                    Excluir Pergunta
                  </Button>
                </div>
                <Input
                  placeholder={`Pergunta ${qIndex + 1}`}
                  value={question.text}
                  onChange={(e) => handleQuestionChange(qIndex, "text", e.target.value)}
                  style={{ marginBottom: 8, width: '100%' }} // Input da pergunta com largu
ra total
                />

                <label style={{ display: "block", minWidth: "100%" }}>Alternativas</label>
                <Radio.Group
                  value={question.responses.findIndex((resp) => resp.is_correct)}
                  onChange={(e) => handleResponseChange(qIndex, e.target.value, "is_correct
", true)}
                  style={{ width: '100%' }}
                >
                  {question.responses.map((resp, rIndex) => (
                    <div
                      key={rIndex}
                      style={{
                        minWidth: '100%',
                        display: 'flex',
                        alignItems: 'center',
                        justifyContent: 'space-between',
                        gap: '10px',
                        marginBottom: 8,
                        borderBottom: '1px solid #FFF',
                      }}
```

```
                    >
                      <Input
                        placeholder={`Alternativa ${String.fromCharCode(65 + rIndex)}`}
                        value={resp.text}
                        onChange={(e) => handleResponseChange(qIndex, rIndex, "text", e.tar
get.value)}
                        style={{ marginRight: 8, flex: 1, width: '100%' }} // Input da resp
osta com largura total
                      />
                      <Radio value={rIndex}>Correta</Radio>
                    </div>
                  ))}
                </Radio.Group>
              </div>
            ))}

            <Button type="dashed" onClick={addQuestion} style={{ marginTop: 8 }}>
              Adicionar Pergunta
            </Button>

            <div style={{ marginTop: 16 }}>
              <Button type="primary" onClick={handleSubmit} loading={loading}>Cadastrar</Bu
tton>
              <Button style={{ marginLeft: 8 }} onClick={() => navigate("/quizzes")}>Cancel
ar</Button>
            </div>
          </Card>
        </main>
      </div>
    </div>
  );
};

export default Register;
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/quizzes/components/EditQuiz.tsx ==
===
import React, { useEffect, useState } from "react";
import { message } from "antd";
import { useNavigate, useParams } from "react-router-dom";
import ItemSideBar from "../../../layout/Sidebar/ItemSideBar";
import ItemHeader from "../../../layout/Header/ItemHeader";
import ItemHeaderCabecalho from "../../../layout/Header/components/ItemHeaderCabecalho";
import DynamicForm from "../../../components/form/DynamicForm";
import { useQuizStore } from "../../../store/quizzes";
import { QuizItem } from "../../../types/quizzes";

const EditQuiz: React.FC = () => {
  const { id } = useParams<{ id: string }>();
  const navigate = useNavigate();
  const { quizzes, updateQuiz } = useQuizStore();
  const [loading, setLoading] = useState(false);
  const [formValues, setFormValues] = useState<Partial<QuizItem>>({
    name: "",
    description: "",
    hour_start: "",
    hour_end: "",
  });

  // Carregar os dados do Quizz ao abrir a tela
  useEffect(() => {
    const quizToEdit = quizzes.find((quiz) => quiz.id === Number(id));
    if (quizToEdit) {
      setFormValues({
        name: quizToEdit.name,
        description: quizToEdit.description,
        hour_start: quizToEdit.hour_start,
        hour_end: quizToEdit.hour_end,
      });
    }
```

```
  }, [id, quizzes]);

  const handleChange = (name: string, value: any) => {
    setFormValues((prev) => ({ ...prev, [name]: value }));
  };

  const handleSubmit = async () => {
    if (!formValues.name?.trim()) {
      message.error("O nome do Quizz Ã© obrigatÃ³rio!");
      return;
    }

    if (!formValues.description?.trim()) {
      message.error("A descriÃ§Ã£o do Quizz Ã© obrigatÃ³ria!");
      return;
    }

    if (!formValues.hour_start || !formValues.hour_end) {
      message.error("Os horÃ¡rios de inÃcio e fim sÃ£o obrigatÃ³rios!");
      return;
    }

    if (loading) return;
    setLoading(true);

    try {
      await updateQuiz(Number(id), formValues);
      message.success("Quizz atualizado com sucesso!");
      navigate("/quizzes");
    } catch (error) {
      message.error("Erro ao atualizar Quizz!");
      console.error(error);
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="layout-container">
      <ItemSideBar />
      <div className="content-container">
        <ItemHeader />
        <main className="content">
          <ItemHeaderCabecalho
            title="Editar Quizz"
            subTitle="Altere os campos desejados e salve as mudanÃ§as"
          />
          <DynamicForm
            fields={[
              { name: "name", label: "Nome", type: "input", required: true },
              { name: "description", label: "DescriÃ§Ã£o", type: "textarea", required: true
 },
              { name: "hour_start", label: "Hora de InÃcio", type: "text", required: true
},
              { name: "hour_end", label: "Hora de Fim", type: "text", required: true },
            ]}
            values={formValues}
            onChange={handleChange}
            onSubmit={handleSubmit}
            loading={loading}
            onCancel={() => navigate("/quizzes")}
          />
        </main>
      </div>
    </div>
  );
};

export default EditQuiz;
```

```
===== /home/alissu/Desktop/nansen_Web/frontend/src/pages/quizzes/hooks/useQuizzesTable.tsx
=====
import { useEffect, useState } from "react";
import { message } from "antd";
import { useNavigate } from "react-router-dom";
import { SortOrder } from "antd/es/table/interface";
import { useQuizStore } from "../../../store/quizzes";
import Actions from "../../../components/actions/Actions";
import { QuizItem } from "../../../types/quizzes";

export const useQuizzesTable = () => {
  const navigate = useNavigate();
  const { quizzes, fetchQuizzes, deleteQuiz } = useQuizStore();
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchQuizzesFromAPI = async () => {
      try {
        setLoading(true);
        await fetchQuizzes();
      } catch (error) {
        console.error("Erro ao buscar quizzes:", error);
        message.error("Erro ao carregar os quizzes!");
      } finally {
        setLoading(false);
      }
    };

    fetchQuizzesFromAPI();
  }, []);

  // DefiniÃ§Ã£o das colunas da tabela
  const columns = [
    {
      title: "Nome",
      dataIndex: "name",
      key: "name",
      sorter: (a: QuizItem, b: QuizItem) => a.name.localeCompare(b.name),
      sortDirections: ["ascend", "descend"] as SortOrder[],
      render: (text: string | undefined) => <strong>{text ?? "Sem nome"}</strong>,
    },
    {
      title: "DescriÃ§Ã£o",
      dataIndex: "description",
      key: "description",
      render: (text: string | undefined) => <span>{text ?? "Sem descriÃ§Ã£o"}</span>,
    },
    {
      title: "Hora de InÃcio",
      dataIndex: "hour_start",
      key: "hour_start",
      render: (text: string | undefined) => <span>{text ?? "NÃ£o informado"}</span>,
    },
    {
      title: "Hora de Fim",
      dataIndex: "hour_end",
      key: "hour_end",
      render: (text: string | undefined) => <span>{text ?? "NÃ£o informado"}</span>,
    },
    {
      title: "AÃ§Ãµes",
      key: "actions",
      render: (_: any, record: QuizItem) => (
        <Actions
          onView={() => message.info(`Visualizando: ${record.name}`)}
          onEdit={() => navigate(`/quizzes/edit/${record.id}`)}
          onDelete={async () => {
            try {
              await deleteQuiz(record.id);
```

```
              message.success("Quiz excluÃ-do com sucesso!");
            } catch (error) {
              message.error("Erro ao excluir quiz.");
            }
          }}
        />
      ),
    },
  ];

  return { columns, quizzes, loading, fetchQuizzes };
};
===== /home/alissu/Desktop/nansen_Web/frontend/src/hooks/useIoTDevices.ts =====
import { useEffect, useState } from "react";
import { useIoTDevicesStore } from "../store/iotDevices";
import { IoTDevice } from "../types/IoTDevice";

export const useIoTDevices = () => {
  const { devices, fetchDevices } = useIoTDevicesStore();
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const loadDevices = async () => {
      setLoading(true);
      try {
        await fetchDevices();
      } catch (error) {
        console.error("Erro ao buscar dispositivos IoT:", error);
      } finally {
        setLoading(false);
      }
    };

    loadDevices();
  }, []);

  return { devices, loading, fetchDevices };
};

===== /home/alissu/Desktop/nansen_Web/frontend/src/hooks/useUserDetails.ts =====
// src/hooks/useUserDetails.ts
import { useState } from "react";
import {
  getUserAchievements,
  AchievementItem,
} from "../services/AchievementService";
import { getUserClaims, ClaimItem } from "../services/ClaimService";

// <-- this path must exactly match the service file name
import { getRewardById, RewardItem } from "../services/RewardService";

export function useUserDetails() {
  const [cache, setCache] = useState<
    Record<number, { achievements: AchievementItem[]; rewards: RewardItem[] }>
  >({});

  async function load(userId: number) {
    if (cache[userId]) return;
    const [ach, claims] = await Promise.all([
      getUserAchievements(userId),
      getUserClaims(userId),
    ]);
    // now fetch each reward by its ID
    const rewards = await Promise.all(
      claims.map((c) => getRewardById(c.reward))
    );
    setCache((c) => ({ ...c, [userId]: { achievements: ach, rewards } }));
  }
```

```
  return { cache, load };
}

===== /home/alissu/Desktop/nansen_Web/frontend/src/index.css =====
@import "./styles/base/reset.css";
@import "./styles/base/variables.css";
@import "./styles/components/buttons.css";

@import "./styles/global/globals.css";
@import "./styles/global/sidebar.css";
@import "./styles/global/tables.css";
@import "tailwindcss";

:root {
  font-family: Inter, system-ui, Avenir, Helvetica, Arial, sans-serif;
  line-height: 1.5;
  font-weight: 400;

  color-scheme: light dark;
  color: rgba(0, 0, 0, 0.87);

  font-synthesis: none;
  text-rendering: optimizeLegibility;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

body {
  margin: 0;

  min-width: 320px;
  min-height: 100vh;
  background-color: #ffffff;
}

/* Links */
a {
  font-weight: 500;
  color: #646cff;
  text-decoration: inherit;
}
a:hover {
  color: #535bf2;
}

/* BotÃµes */
button {
  border-radius: 8px;
  border: 1px solid transparent;
  padding: 0.6em 1.2em;
  font-size: 1em;
  font-weight: 500;
  font-family: inherit;
  background-color: #1a1a1a;
  cursor: pointer;
  transition: border-color 0.25s;
}
button:hover {
  border-color: #646cff;
}
:where(.css-dev-only-do-not-override-1d4w9r2).ant-menu-light .ant-menu-item,
:where(.css-dev-only-do-not-override-1d4w9r2).ant-menu-light
  > .ant-menu
  .ant-menu-item,
:where(.css-dev-only-do-not-override-1d4w9r2).ant-menu-light
  .ant-menu-submenu-title,
:where(.css-dev-only-do-not-override-1d4w9r2).ant-menu-light
  > .ant-menu
  .ant-menu-submenu-titl {
```

```
  color: #004281;
}
```