12 — Tema e estilo

12 — Tema e estilo

> Um `createTheme` central, locale pt-BR, e a regra de quando usar `sx` versus `styled`. Estilo repetido em duas telas vira tema; estilo repetido entre 2+ **componentes** vira `theme/commonStyles.ts`. Sem dark mode na referência — se o projeto precisar, é decisão do início.

O tema

```
// src/theme/index.ts
import { createTheme } from "@mui/material/styles";
import { ptBR } from "@mui/material/locale";
const palette = {  primary: { main: "#1976d2", light: "#42a5f5", dark: "#1565c0", contrastText:
"#ffffff" },  secondary: { main: "#dc004e", light: "#ff5983", dark: "#9a0036", contrastText:
"#ffffff" },  error: { main: "#f44336", light: "#e57373", dark: "#d32f2f", contrastText:
"#ffffff" },  warning: { main: "#ff9800", light: "#ffb74d", dark: "#f57c00", contrastText:
"#000000" },  info: { main: "#2196f3", light: "#64b5f6", dark: "#1976d2", contrastText:
"#ffffff" },  success: { main: "#4caf50", light: "#81c784", dark: "#388e3c", contrastText:
"#ffffff" }
};
export const theme = createTheme(
  {
    palette,
    typography: {
      fontFamily: '"Roboto", "Helvetica", "Arial", sans-serif'      // escala h1..overline
definida explicitamente — ver arquivo de referencia
```

```
    },
    shape: { borderRadius: 8 },
    spacing: 8,
    components: {
      MuiButton: {
        styleOverrides: {          root: { textTransform: "none", borderRadius: 8, fontWeight:
500, padding: "8px 16px" },
          contained: {
            boxShadow: "0 2px 4px rgba(0,0,0,0.1)",            "&:hover": { boxShadow: "0 4px
8px rgba(0,0,0,0.15)" }
          }
        }
      },
      MuiCard: {        styleOverrides: { root: { boxShadow: "0 2px 8px rgba(0,0,0,0.1)",
borderRadius: 12 } }
      },
      MuiTextField: {        styleOverrides: { root: { "& .MuiOutlinedInput-root": {
borderRadius: 8 } } }
      },
      MuiPaper: {
        // remove gradiente de elevacao do MUI        styleOverrides: { root: { backgroundImage:
"none" } }
      }
    }
  },
  ptBR // locale dos componentes MUI
);
export default theme;
```

O segundo argumento `ptBR` traduz textos internos do MUI (paginação, date pickers). É separado do `adapterLocale={ptBR}` de `date-fns` em `main.tsx` — **os dois são necessários** e vêm de pacotes diferentes:

```
import { ptBR } from "@mui/material/locale"; // textos dos componentesimport { ptBR } from "date-
fns/locale"; // formatação de datas
```

`sx` vs `styled` vs arquivo de estilo

Situação	Use
1–2 propriedades pontuais	`sx` inline
Bloco de estilo de um componente	Constante nomeada em `Componente.styles.ts`
Estilo que depende de prop/estado	Função em `.styles.ts` que recebe o valor
Componente novo com estilo próprio e complexo	`styled()` em `.styles.ts`
Estilo repetido em 2+ **componentes**	`src/theme/commonStyles.ts`
Estilo repetido em 2+ **telas via MUI slot**	Tema (`components.styleOverrides`)

`sx` inline — pouco e óbvio

```
<Typography variant="subtitle2" sx={{ mr: 2 }}>
```

Aceitável até ~2 propriedades. Acima disso, vai para o arquivo de estilos.

`.styles.ts` — uma constante nomeada por estilo

Convenção do padrão: cada estilo é uma **constante exportada e nomeada**, não uma propriedade de um objeto `styles`. O nome descreve o que o estilo faz, sufixado com `Style`.

```
// src/pages/Cadastro/Fornecedor/FornecedorPage.styles.tsimport { SxProps, Theme } from
"@mui/material";
export const outerBoxStyle: SxProps<Theme> = {
  flexGrow: 1,
  p: 3,
  height: "calc(100vh - 64px)",
  overflow: "hidden"
};
```

```
export const listLoadingStyle: SxProps<Theme> = {
  display: "flex",
  justifyContent: "center",
  alignItems: "center",
  height: "100%"
};
export const unselectedItemStyle: SxProps<Theme> = {
  display: "flex",
  justifyContent: "center",
  alignItems: "center",
  height: "100%"
};
```

```
import { listLoadingStyle, outerBoxStyle, unselectedItemStyle } from "./FornecedorPage.styles";
<Box sx={outerBoxStyle}>
  <Box sx={listLoadingStyle}>...</Box>
</Box>;
```

Por que constantes nomeadas e não um objeto `styles = { root, header, ... }`:

- **O import fica explícito** — `import { outerBoxStyle }` mostra exatamente o que a tela usa; um objeto `styles` obriga a abrir o arquivo para saber quais chaves existem.
- **Renomear é seguro.** Renomear `styles.root` para outra coisa exige revisar todo uso de `styles.` no arquivo; renomear `outerBoxStyle` é um rename de símbolo, com suporte de qualquer editor.
- **Facilita promover para `commonStyles.ts`** — mover uma constante exportada para outro arquivo é um corte-e-cola; extrair uma chave de dentro de um objeto exige reescrever os dois lados.

Tipar como `SxProps<Theme>` dá autocomplete e valida os tokens.

Estilo dependente de estado — função nomeada

```
// src/components/layout/Main/Main.styles.ts
```

```
import { SxProps, Theme } from "@mui/material";
const DRAWER_WIDTH = 260;
export const mainRootStyle: SxProps<Theme> = {
  display: "flex",
  minHeight: "100vh"
};
// depende do estado do drawer — por isso funcao, nao constanteexport const drawerStyle = (open:
boolean): SxProps<Theme> => ({
  width: open ? DRAWER_WIDTH : 0,
  flexShrink: 0,
  "& .MuiDrawer-paper": { width: DRAWER_WIDTH, boxSizing: "border-box" }
});
```

```
<Drawer sx={drawerStyle(drawerOpen)} />
```

Mesma regra: nome exportado, não uma chave de objeto. A única diferença é que o valor é uma função.

`styled()`

```
import { styled } from "@mui/material/styles";
import { Box } from "@mui/material";
export const PainelDestacado = styled(Box)(({ theme }) => ({
  padding: theme.spacing(2),
  borderRadius: theme.shape.borderRadius,
  backgroundColor: theme.palette.grey[100]
}));
```

Use quando o resultado é um **componente** reutilizável, não um conjunto de props de estilo.

Sempre use tokens do tema

```
// ERRADO — valor solto
```

```
<Box sx={{ padding: "16px", color: "#1976d2", borderRadius: "8px" }} />
// CERTO — tokens
<Box sx={{ p: 2, color: "primary.main", borderRadius: 1 }} />
```

`spacing: 8` significa que `p: 2` = 16px. Mudar o espaçamento base reajusta o app inteiro; valores soltos não acompanham.

Atalhos comuns: `p` / `m` (padding/margin), `px` / `py`, `mt` / `mb` / `ml` / `mr`, `gap`.

Estilo repetido vira `commonStyles.ts` ou tema

Uma constante de `.styles.ts` nasce local ao componente. Quando a **segunda** tela precisar do mesmo estilo, promova-a — não copie a definição.

```
// src/theme/commonStyles.ts
import { SxProps, Theme } from "@mui/material";
// Estilos reaproveitados por mais de uma tela.// Regra: nasce em <Componente>.styles.ts; move
pra ca quando a 2a tela precisar.
/** Centraliza o conteudo nos dois eixos ocupando toda a altura disponivel. */export const
centeredFillStyle: SxProps<Theme> = {
  display: "flex",
  justifyContent: "center",
  alignItems: "center",
  height: "100%"
};
/** Area de conteudo de uma tela dentro do shell — desconta a AppBar. */export const
outerBoxStyle: SxProps<Theme> = {
  flexGrow: 1,
  p: 3,
  height: "calc(100vh - 64px)",
  overflow: "hidden"
};
```

Quando o repetido é uma propriedade de um **slot do MUI** (todo `TextField` da aplicação, todo `Button`

), o lugar certo é o tema, não `commonStyles.ts`:

```
// src/theme/index.ts
components: {
  MuiTextField: {    styleOverrides: { root: { "& .MuiOutlinedInput-root": { borderRadius: 8 } }
}
  }
}
```

O mesmo vale para `defaultProps`:

```
components: {
  MuiTextField: { defaultProps: { size: "small", fullWidth: true } }
}
```

Isso elimina `size="small" fullWidth` repetido em cada campo. **Defina `defaultProps` desde o início** — evita repetir as mesmas props em toda tela.

Regra de decisão:

O que se repete	**Vai para**
Um layout específico (centralizar, área de conteúdo)	`theme/commonStyles.ts`
Uma propriedade de todo componente MUI de um tipo	`theme/index.ts`

Tipografia

Use `variant`, não `fontSize`:

```
// ERRADO
<Typography sx={{ fontSize: "1.25rem", fontWeight: 400 }}>Título</Typography>
// CERTO
<Typography variant="h3">Título</Typography>
```

A escala está definida no tema. Se um tamanho não existe lá, ou você quer a variante errada, ou falta uma variante no tema.

Cores

Só da paleta:

```
<Typography color="text.secondary" /><Box sx={{ bgcolor: "background.paper", borderColor:
"divider" }} />
<Button color="primary" />
```

Nenhum hex fora de `src/theme/index.ts`. Se precisar de uma cor nova, adicione à paleta.

Dark mode

A referência não tem. Se o projeto novo precisar:

1. Extraia a paleta para `light` e `dark`.
2. Crie o tema com `mode` vindo de estado (`useMediaQuery("(prefers-color-scheme: dark)")` + preferência do usuário em `localStorage`).
3. Envolva com `useMemo` para não recriar a cada render.
4. **Auditar todo `sx`** — qualquer hex solto quebra no modo escuro.

Adotar depois é caro exatamente pelo passo 4. Decida no início.

Responsividade

Breakpoints do MUI dentro do `sx`:

```
<Box sx={{ display: { xs: "none", md: "block" }, p: { xs: 1, md: 3 } }} />
```

Se o projeto precisar de suporte a mobile, trate desde o começo — retrofitar layout responsivo é reescrever telas.

> **Este documento adota constantes nomeadas** como padrão — uma constante por estilo, em vez de um objeto `styles = { root, drawer, ... }` por arquivo — ver a comparação em "`.styles.ts` — uma constante nomeada por estilo" acima.

Revision #2

Created Thu, Aug 20, 2026 5:01 PM by Geraldo Barbosa

Updated Tue, Aug 25, 2026 4:54 PM by Geraldo Barbosa