

<LAB365>

ATOMIC DESIGN

<LAB365>



AGENDA | M1S10 - A1

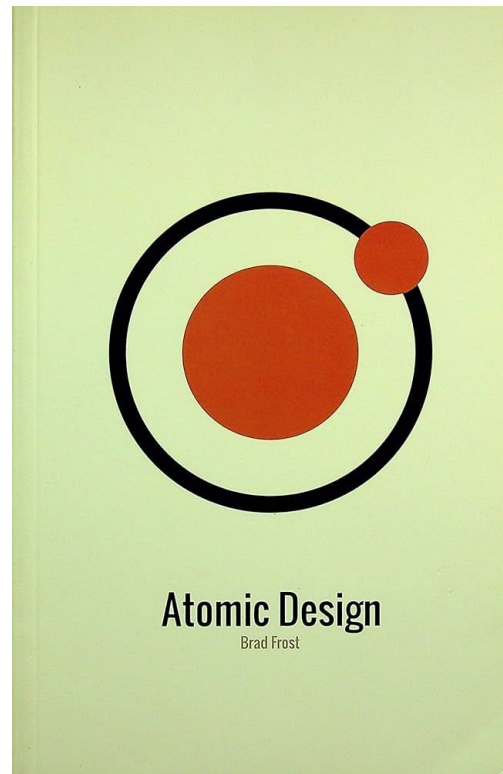
- Atomic Design

ATOMIC DESIGN

O Atomic Design é uma **metodologia** criada por Brad Frost que revolucionou a maneira como organizamos e estruturamos componentes em interfaces web, especialmente em projetos React. Inspirado na química, ele divide a UI em níveis hierárquicos, facilitando a construção de sistemas de design escaláveis e reutilizáveis.

O Atomic Design é composto por cinco estágios, cada um representando um nível de complexidade:

- Átomos
- Moléculas
- Organismos
- Templates
- Páginas



ATOMIC DESIGN

Átomos

Os menores blocos de construção da interface.

Exemplos: botões, inputs, ícones, labels, etc...

Características:

- Não podem ser divididos em partes menores sem perder sua função.
- Altamente reutilizáveis em todo o projeto.
- Pouca ou nenhuma lógica (são componentes puramente visuais).



ATOMS

ATOMIC DESIGN

```
type ButtonProps = {
  text: string;
  onClick?: () => void;
  variant?: 'primary' | 'secondary' | 'danger';
  size?: 'sm' | 'lg';
  className?: string;
};

export const Button = ({ text, onClick, variant = 'primary', size, className = '' }: ButtonProps) => {

  const variantClasses = {
    primary: 'btn-primary',
    secondary: 'btn-secondary',
    danger: 'btn-danger'
  };

  const sizeClasses = {
    sm: 'btn-sm',
    lg: 'btn-lg'
  };

  return (
    <button
      className={`btn ${variantClasses[variant]} ${size ? sizeClasses[size] : ''} ${className}`}
      onClick={onClick}
    >
      {text}
    </button>
  );
};
```

ATOMIC DESIGN

```
type TypographyProps = {
  variant?: 'h1' | 'h2' | 'h3' | 'body' | 'caption';
  text: string;
  className?: string;
  bold?: boolean;
};

export const Typography = ({ variant = 'body', text, className = '', bold = false }: TypographyProps) => {

  const Tag = variant === 'body' || variant === 'caption' ? 'p' : variant;

  const classes = {
    h1: 'display-4',
    h2: 'display-5',
    h3: 'display-6',
    body: '',
    caption: 'small text-muted'
  };

  return (
    <Tag
      className={`${classes[variant]} ${bold ? 'fw-bold' : ''} ${className}`}
    >
      {text}
    </Tag>
  );
};
```

ATOMIC DESIGN

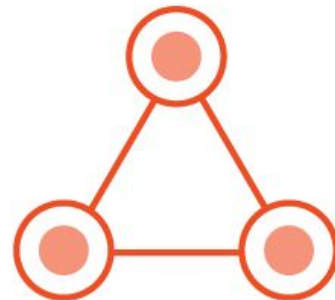
Moléculas

Combinações de átomos que formam componentes funcionais simples.

Exemplos: Formulário de busca (Input + Button), Card de produto (Imagem + Título + Preço) e Item de menu (Ícone + Texto).

Características:

- Começam a ter lógica interna (ex: validação de formulário).
- Ainda são reutilizáveis, mas menos genéricas que átomos.
- Não dependem de contexto externo (são autocontidas).



MOLECULES

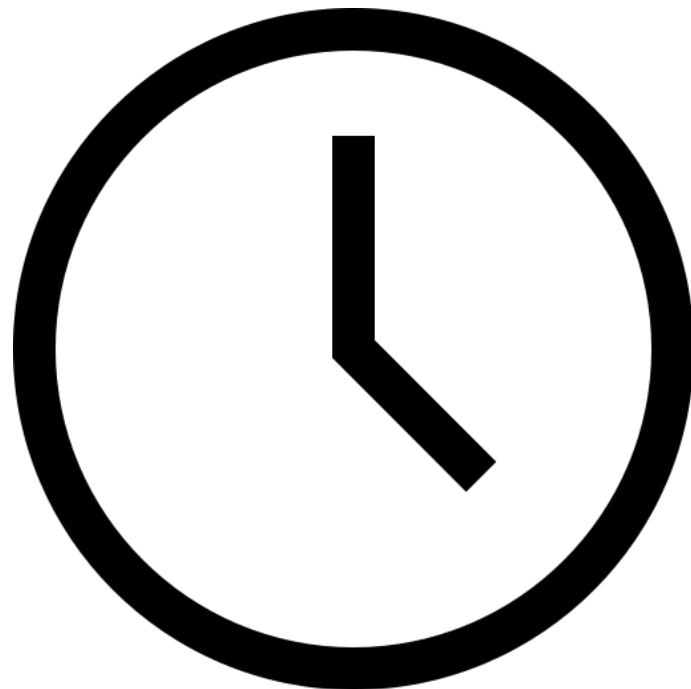
INTERVALO!

Finalizamos o nosso primeiro período de hoje. Que tal descansar um pouco?!

Nos vemos em 20 minutos.

Início: 20:20

Retorno: 20:40



ATOMIC DESIGN

```
import { Button } from '../atoms/Button';
import { Typography } from '../atoms/Typography';

type AlertMessageProps = {
  type: 'success' | 'warning' | 'error';
  title: string;
  message: string;
  onClose?: () => void;
};

export const AlertMessage = ({ type, title, message, onClose }: AlertMessageProps) => {
  const alertClasses = {
    success: 'alert-success',
    warning: 'alert-warning',
    error: 'alert-danger'
  };
};
```

```
return (
  <div className={`alert ${alertClasses[type]} justify-content-between`} >
    <div>
      <Typography variant="h3" bold text={title}></Typography>
      <Typography variant="body" text={message}></Typography>
    </div>
    {onClose && (
      <Button
        text="✕"
        variant="secondary"
        size="sm"
        onClick={onClose}
        className="align-self-start"
      >
      </Button>
    )}
  </div>
);
```

ATOMIC DESIGN

Organismos

Componentes complexos que combinam moléculas e átomos, formando seções significativas da UI.

Exemplos: Cabeçalho (Logo + Menu + SearchBar), Grade de produtos (Vários ProductCard) e Formulário de login (Inputs + Botão + Links).

Características:

- Podem ter estado e lógica mais complexa.
- São frequentemente vinculados a dados (API, contexto).
- Menos reutilizáveis, pois são mais específicos.



ORGANISMS

ATOMIC DESIGN

```
import { Button } from '../atoms/Button';
import { Typography } from '../atoms/Typography';

type Product = {
  id: string;
  name: string;
  price: number;
  image: string;
  rating: number;
};

type ProductCardProps = {
  product: Product;
  onAddToCart: (id: string) => void;
};

export const ProductCard = ({ product, onAddToCart }: ProductCardProps) => {

  const renderStars = () => {
    return Array(5).fill(0).map((_, i) => (
      <span
        key={i}
        className={`fa-star ${i < product.rating ? 'fas text-warning' : 'far'} `}
      />
    ));
  };

};
```

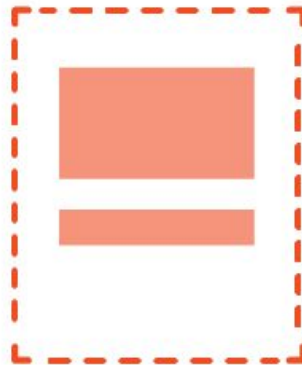

ATOMIC DESIGN

Templates

Estruturas de layout sem conteúdo real, definindo posicionamento de organismos, grids, espaçamentos e comportamentos responsivos.

Características:

- Não contém dados reais (placeholders).
- Focam em estrutura, não em estilo.
- Podem ter múltiplas variações (ex: Template com/sem sidebar).



TEMPLATES

ATOMIC DESIGN

```
type ProductPageTemplateProps = {
  products: {
    id: string;
    name: string;
    price: number;
    image: string;
    rating: number;
  }[];
  onAddToCart: (id: string) => void;
  footerText?: string;
};

export const ProductPageTemplate = ({
  products,
  onAddToCart,
  footerText,
}: ProductPageTemplateProps) => {
  return (
    <div className="container mx-auto p-4">
      <Typography variant="h1" className="mb-6 text-center" text='Seus produtos' />

      <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-6">
        {products.map((product) => (
          <ProductCard
            key={product.id}
            product={product}
            onAddToCart={onAddToCart}
          />
        ))}
      </div>
    </div>
  );
};
```

```
    <div className="mt-8 text-center">
      <Typography variant="body" className="text-gray-500" text={footerText ?? ""} />
    </div>
  </div>
);
};
```

ATOMIC DESIGN

Páginas

Implementações concretas dos templates, com dados reais.

Exemplos: Página inicial (HomePage = Header + Banner + Product) e Página de produto (ProductPage = Galeria + Descrição + Reviews).

Características:

- Podem ter estado e lógica mais complexa.
- São frequentemente vinculados a dados (API, contexto).
- Menos reutilizáveis, pois são mais específicos.



PAGES

ATOMIC DESIGN

```
type Product = {
  id: string;
  name: string;
  price: number;
  image: string;
  rating: number;
};

// Dados Fakes, simulação de API
const fetchProducts = async (): Promise<Product[]> => {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve([
        {
          id: '1',
          name: 'Smartphone Premium',
          price: 2999.90,
          image: 'https://via.placeholder.com/300?text=Smartphone',
          rating: 2
        },
        {
          id: '2',
          name: 'Notebook Pro',
          price: 5899.90,
          image: 'https://via.placeholder.com/300?text=Notebook',
          rating: 3
        }
      ]);
    }, 500);
  });
};
```

```
    {
      id: '3',
      name: 'Fone sem Fio',
      price: 799.90,
      image: 'https://via.placeholder.com/300?text=Fone',
      rating: 5
    }
  ]);
}, 500);
});
});

export const ProductPage = () => {
  const [products, setProducts] = useState<Product[]>([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);

  // Carrega os produtos
  useEffect(() => {
    const loadProducts = async () => {
      try {
        const data = await fetchProducts();
        setProducts(data);
      } catch (err) {
        setError('Falha ao carregar produtos: ' + err);
      } finally {
        setLoading(false);
      }
    };
  });
};
```


ATOMIC DESIGN

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

  // Lógica do carrinho
  const handleAddToCart = (id: string) => {
    alert(`Produto ${id} adicionado ao carrinho!`);
    // Em uma app real: dispatch(adicionarAoCarrinho(id))
  };

  // Renderização condicional
  if (loading) return <div>Carregando...</div>;
  if (error) return <div>{error}</div>;
  if (products.length === 0) return <div>Nenhum produto encontrado</div>;

  return (
    <ProductPageTemplate
      products={products}
      onAddToCart={handleAddToCart}
    />
  );
};
```

ATOMIC DESIGN

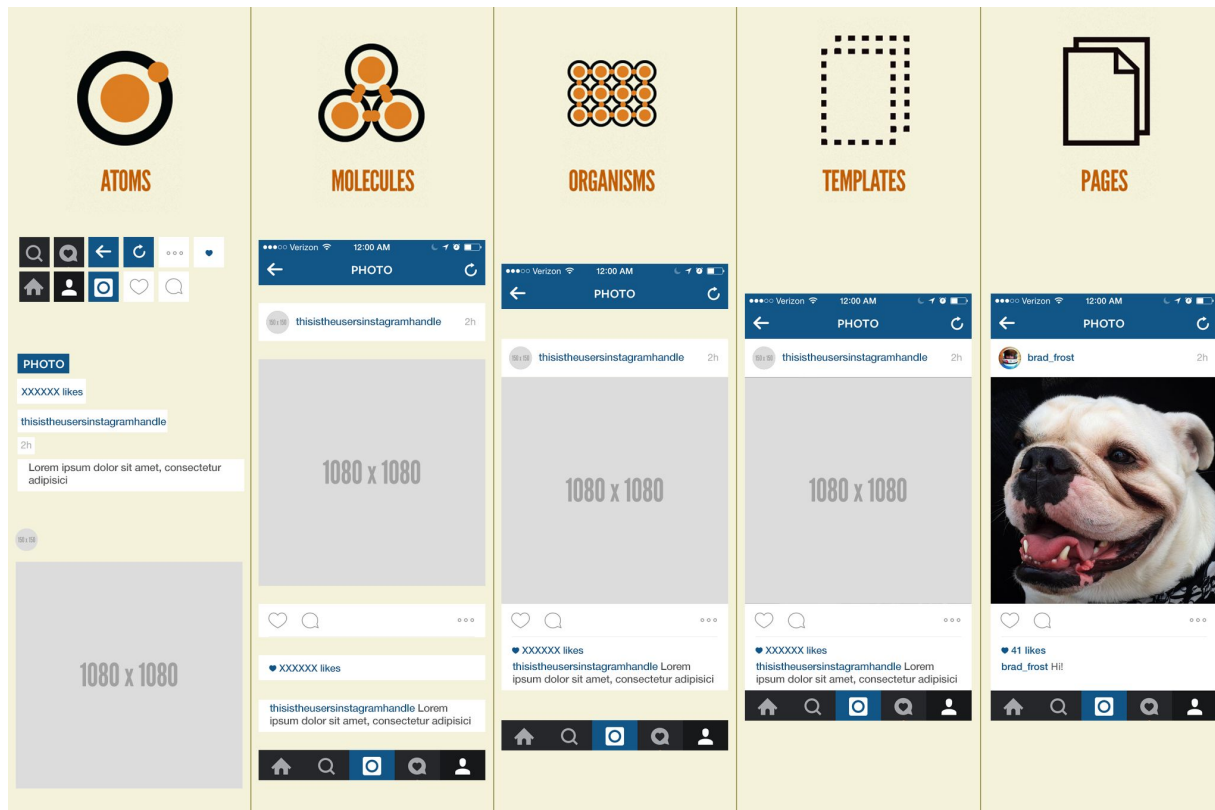
Vantagens

- Escalabilidade: Facilita o crescimento do projeto.
- Consistência: Componentes reutilizáveis garantem um design uniforme.
- Manutenção: Problemas são isolados e fáceis de corrigir.
- Colaboração: Designers e devs trabalham com a mesma linguagem.
- Documentação natural: A estrutura já serve como guia.

Quando Usar (e Quando Não Usar) Atomic Design

| Ótimo | Pode ser excessivo |
|-------------------------------|---------------------|
| Design Systems | Aplicações pequenas |
| Aplicações grande e complexas | Times muito enxutos |
| Equipes com múltiplos devs | |

ATOMIC DESIGN



<LAB365>

<LAB365>

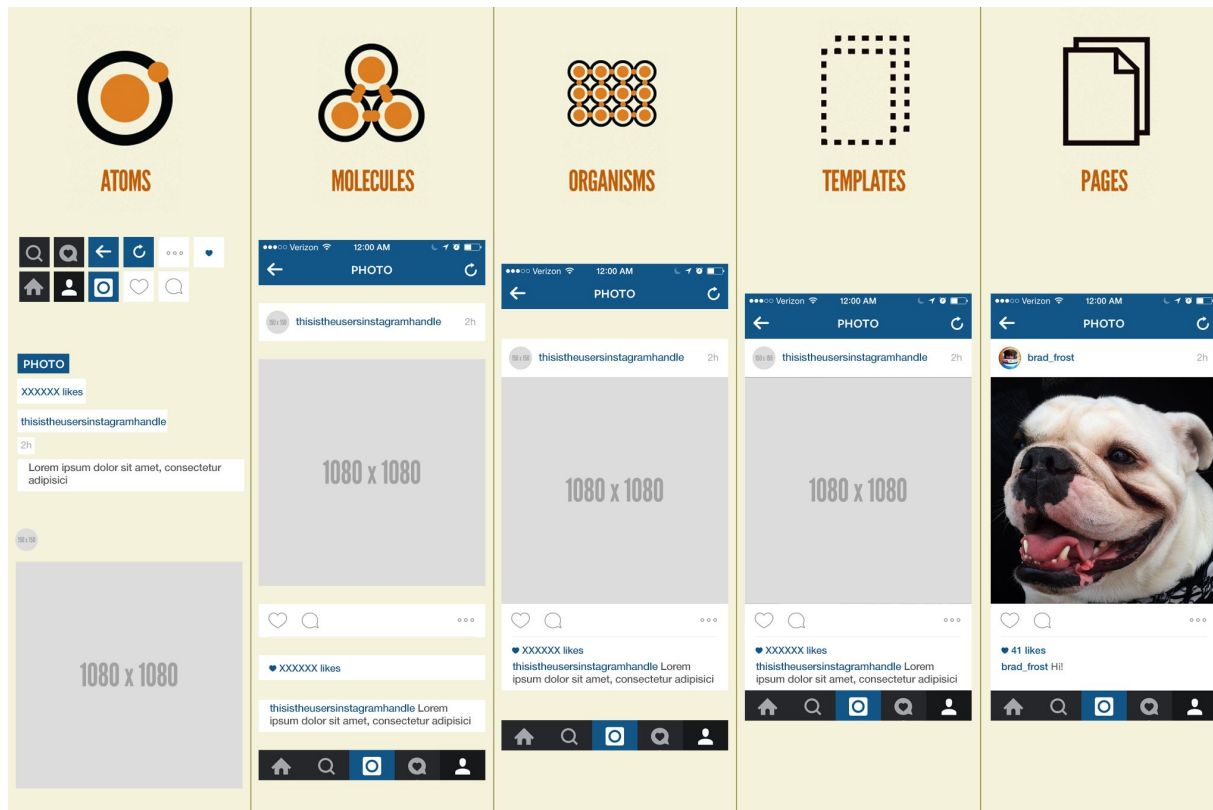
CSS MODULES

MATERIAL UI

AGENDA | M1S010 - A2

- Finalização Atomic Design
- CSS Modules
- Uso de bibliotecas externas
- Material UI

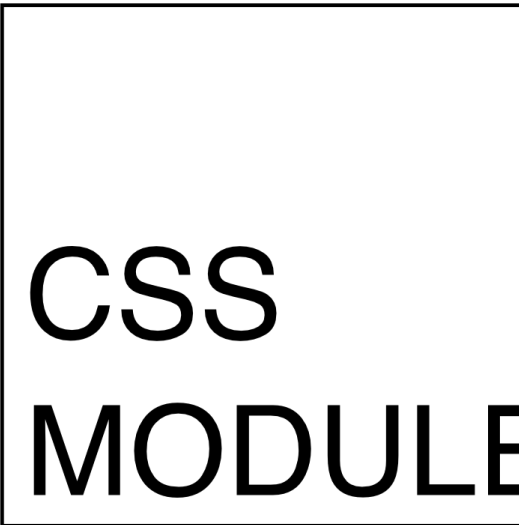
ATOMIC DESIGN



CSS MODULES

CSS Modules é uma técnica de escopamento de CSS que permite escrever estilos locais para componentes React/Vue/Angular sem vazamentos para outras partes da aplicação. Ele transforma nomes de classes em identificadores únicos automaticamente.

- Surgiu como alternativa ao CSS global (que causa conflitos)
- Popularizou-se com a ascensão dos componentes React
- É suportado nativamente por Webpack, Vite e outros bundlers



CSS
MODULES

CSS MODULES

Antes do CSS Modules, os estilos em aplicações React eram frequentemente escritos de três maneiras:

1. CSS Global – Classes aplicadas em todo o projeto, levando a conflitos.
2. Inline Styles – Estilos diretamente nos elementos, limitando reuso e manutenção.
3. CSS-in-JS (Styled Components, Emotion) – Poderoso, mas com overhead de performance.

CSS Modules surgiu como uma solução intermediária

- Escopo local automático (evita conflitos de classes).
- Mantém a sintaxe CSS pura (familiar para devs front-end).
- Integração simples com bundlers (Webpack, Vite).
- Performance melhor que CSS-in-JS (sem runtime JavaScript).

CSS MODULES

Como Funcionam os CSS Modules?

1. Você cria um arquivo .module.css
2. Importa como um objeto JavaScript
3. O bundler (Webpack/Vite) transforma

```
/* Input: Button.module.css */  
.primary { background: blue; }  
.secondary { background: red; }
```

```
import styles from './ButtonCSS.module.css';  
  
type ButtonCSSProps = {  
  text: string;  
  onClick?: () => void;  
};  
  
export const ButtonCSS = ({ text, onClick}: ButtonCSSProps) => {  
  return (  
    <button  
      className={styles.primary}  
      onClick={onClick}  
    >  
      {text}  
    </button>  
  );  
};
```

CSS MODULES

Vantagens

- Escopo Local
- Nomenclatura simples
- Manutenção facilitada
- Performance
- Compatibilidade

Quando usar o CSS Modules?

| Ótimo | Pode ser excessivo |
|---|---------------------------------|
| Projetos React/Next.js médios a grandes | Protótipos rápidos |
| Equipes que preferem CSS tradicional | Projetos que já usam CSS-in-JS. |
| Aplicações onde performance é crítica | |

CSS MODULES

```
import styles from './ButtonCSS.module.css';

type ButtonCSSProps = {
  text: string;
  onClick?: () => void;
  moduleCSS?: 'primary' | 'secondary';
};

export const ButtonCSS = ({ text, moduleCSS = 'primary', onClick }: ButtonCSSProps) => {
  return (
    <button className={` ${styles.button} ${styles[moduleCSS]} `} onClick={onClick}>
      {text}
    </button>
  );
};
```

CSS MODULES

```
.button {  
  padding: 8px 16px;  
  border: none;  
  border-radius: 4px;  
  cursor: pointer;  
  transition: opacity 0.2s;  
}  
  
.primary {  
  background: #3498db;  
  color: white;  
}  
  
.secondary {  
  background: #e0e0e0;  
  color: #333;  
}  
  
.button:hover {  
  opacity: 0.9;  
}
```

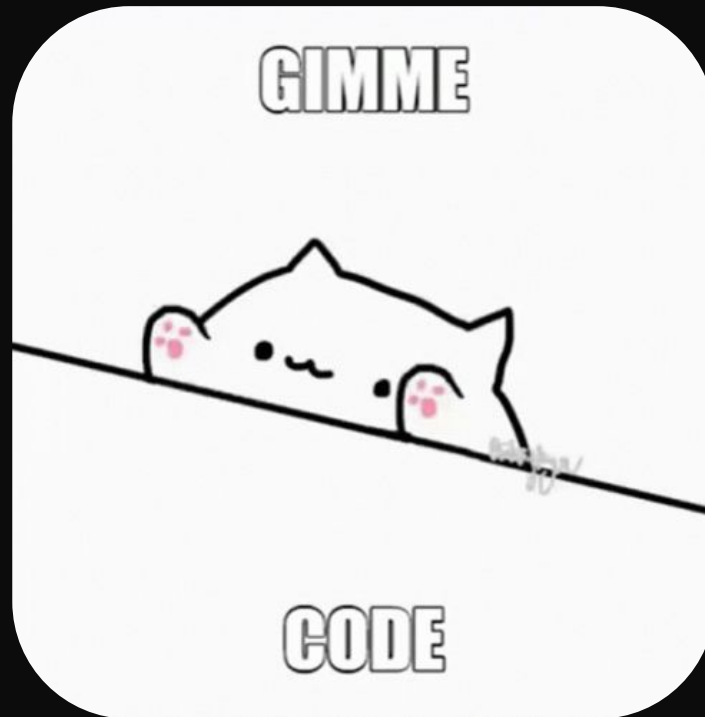
TREINANDO NOSSAS HABILIDADES!

Nós temos a missão agora de criar um ecommerce de ebooks!

Usaremos Atomic design e CSS modules para o desenvolvimento.

- Átomos: Tags (Exibe tags como "Novo", "Promoção"), Price e Rating.
- Moléculas: BookCard, FilterPanel (por gênero).
- Organismos: BookGrid (Exibição de varios BookCards).
- Template: Header (topo), FilterPanel (a esquerda), BookGrid (a direita) Footer.
- Page: BookPage.

Crie um novo projeto para estar desenvolvendo essa atividade.



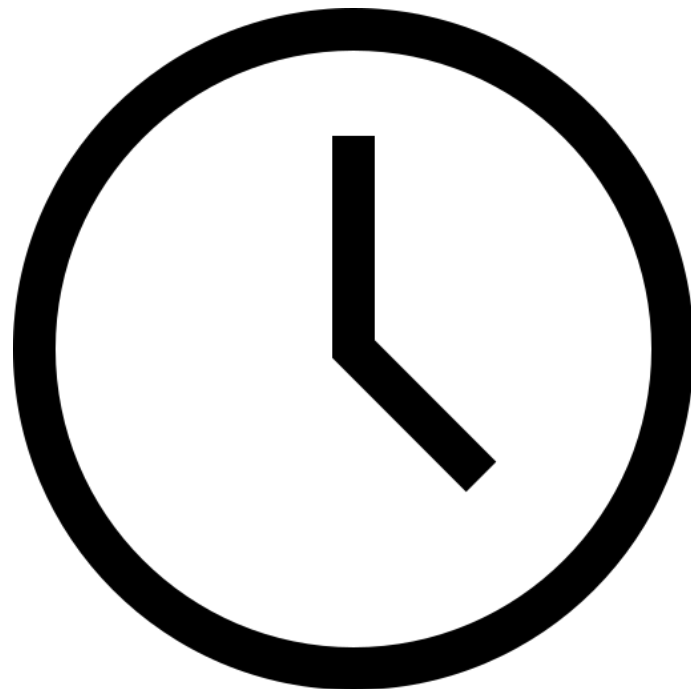
INTERVALO!

Finalizamos o nosso primeiro período de hoje. Que tal descansar um pouco?!

Nos vemos em 20 minutos.

Início: 20:30

Retorno: 20:50



BIBLIOTECAS EXTERNAS

Bibliotecas externas são **pacotes de código pré-escrito** que ajudam a adicionar funcionalidades ao seu projeto React sem precisar desenvolver tudo do zero. Elas podem oferecer:

- Componentes UI prontos (Material UI, Ant Design)
- Gerenciamento de estado (Redux, Zustand)
- Roteamento (React Router)
- Formulários (Formik, React Hook Form)
- Animações (Framer Motion)
- Requisições HTTP (Axios, SWR)

Vantagens

- Produtividade: Acelera o desenvolvimento
- Confiabilidade: Soluções testadas pela comunidade
- Manutenção: Atualizações feitas pelos mantenedores
- Performance: Otimizações especializadas

MATERIAL UI

O Material UI (MUI) é uma das bibliotecas de componentes React mais populares, implementando o Material Design do Google. Ele oferece:

- +80 componentes prontos
- Sistema de temas customizável
- Acessibilidade integrada
- Suporte a SSR (Server-Side Rendering)
- Ótima documentação

Quando Usar?

- Aplicações que precisam de UI profissional rapidamente
- Projetos que seguem Material Design
- Times que valorizam consistência visual

MATERIAL UI

Para utilizar essa biblioteca o primeiro passo é seguirmos com a instalação e configuração.

1. npm install @mui/material
@emotion/react
@emotion/styled
2. npm install
@mui/icons-material

E agora precisamos construir nosso ThemeProvider.

```
import Button from '@mui/material/Button';

function AtmButton() {
  return <Button variant="contained">Clique Aqui</Button>;
}

export default AtmButton;
```

```
import { createTheme, ThemeProvider } from '@mui/material/styles';
import AtmButton from './atoms/AtmButton';

const theme = createTheme({
  palette: {
    primary: {
      main: '#1976d2',
    },
  },
});

function App() {
  return (
    <ThemeProvider theme={theme}>
      <AtmButton />
    </ThemeProvider>
  );
}

export default App;
```

MATERIAL UI: HEADER

```
import { AppBar, Toolbar, IconButton, Typography } from '@mui/material';
import MenuIcon from '@mui/icons-material/Menu';

function MlcHeader() {
  return (
    <AppBar position="static">
      <Toolbar>
        <IconButton edge="start" color="inherit">
          <MenuIcon />
        </IconButton>
        <Typography variant="h6">Minha App</Typography>
      </Toolbar>
    </AppBar>
  );
}

export default MlcHeader;
```

MATERIAL UI : FORM

```
import { TextField, Checkbox, FormControlLabel, Button } from '@mui/material';

function MlcForm() {
  return (
    <form>
      <TextField label="Email" fullWidth margin="normal" />
      <TextField label="Senha" type="password" fullWidth margin="normal" />
      <FormControlLabel control={<Checkbox />} label="Lembrar-me" />
      <Button type="submit" variant="contained">Entrar</Button>
    </form>
  );
}

export default MlcForm;
```

MATERIAL UI: DINAMIC THEME

```
import { useState } from 'react';
import { ThemeProvider, createTheme, CssBaseline, Button, } from '@mui/material';
import MlcForm from './molecules/MlcForm';

function App() {
  // Estado para controlar o tema
  const [isDark, setIsDark] = useState(false);

  // Cria o tema com base no estado
  const theme = createTheme({
    palette: {
      mode: isDark ? 'dark' : 'light',
      background: {
        default: isDark ? '#121212' : '#f5f5f5',
        paper: isDark ? '#1e1e1e' : '#ffffff',
      },
      primary: {
        main: isDark ? '#90caf9' : '#1976d2',
      },
    },
  });
}
```

MATERIAL UI : DINAMIC THEME

```
// Função para alternar entre temas
const toggleTheme = () => {
  setIsDark(!isDark);
};

return (
  <ThemeProvider theme={theme}>
    <CssBaseline /> {/* Aplica o background global */}

    {/* Botão para trocar o tema */}
    <Button
      variant="contained"
      onClick={toggleTheme}
    >
      {isDark ? 'Tema Claro' : 'Tema Escuro'}
    </Button>

    <MlcForm />
  </ThemeProvider>
);

export default App;
```

<LAB365>

<LAB365>

RESPONSIVIDADE REVISÃO

AGENDA | M1S10 - A3

- Responsividade
- Media Queries
- useMediaQuery
- Revisão

RESPONSIVIDADE

Responsividade em aplicações React modernas envolve estratégias para adaptar layouts a diferentes tamanhos de tela.

No vite já temos algumas configurações pré-definidas de responsividade, como o Viewport Básico (index.html).

Mas é interessante usarmos um reset CSS, que é responsável por limpar o CSS nativo dos navegadores, usaremos: `npm install modern-css-reset`.

E no nosso [main.ts](#) adicionaremos: `import 'modern-css-reset';`.

MEDIA QUERIES

Media queries são uma funcionalidade fundamental do CSS que permitem aplicar estilos condicionais baseados nas características do dispositivo ou viewport. Elas são o alicerce do design responsivo moderno.

Media queries são blocos de regras CSS que só são aplicados quando determinadas condições são satisfeitas, como:

- Largura da tela (width)
- Altura da tela (height)
- Orientação (orientation)
- Densidade de pixels (resolution)
- Tipo de dispositivo (screen, print)

MEDIA QUERIES

```
@media (min-width: 768px) {  
| /* Estilo por largura minima */  
}  
  
@media (orientation: portrait) {  
| /* Estilos para modo retrato (celular) */  
}  
  
@media (orientation: landscape) {  
| /* Estilos para modo paisagem (tablet/desktop) */  
}  
  
@media print {  
| /* Estilos para impressão */  
}  
  
@media screen {  
| /* Estilos para telas */  
}
```

MEDIA QUERIES

Valores comuns para dispositivos atuais:

```
/* Mobile (default) */  
body { font-size: 14px; }  
  
/* Small devices (landscape phones, 576px and up) */  
@media (min-width: 576px) { /*ESTILOS*/ }  
  
/* Medium devices (tablets, 768px and up) */  
@media (min-width: 768px) { /*ESTILOS*/ }  
  
/* Large devices (desktops, 992px and up) */  
@media (min-width: 992px) { /*ESTILOS*/ }  
  
/* Extra large devices (large desktops, 1200px and up) */  
@media (min-width: 1200px) { /*ESTILOS*/ }
```

USEMEDIAQUERY

O **useMediaQuery** é um **hook** poderoso do Material UI que permite criar componentes responsivos com base em condições de mídia CSS diretamente no JavaScript/React.

O que faz: Detecta condições de mídia CSS (como largura da tela) e retorna um valor booleano.

Vantagens:

- Integração perfeita com o sistema de temas do MUI
- Sintaxe mais limpa que media queries CSS tradicionais
- Ideal para lógica condicional em JSX

Com a instalação do Material UI que fizemos anteriormente, já temos disponível o useMediaQuery para utilização.

USEMEDIAQUERY

```
import { useTheme } from '@mui/material/styles';
import useMediaQuery from '@mui/material/useMediaQuery';

function Component(){
  const theme = useTheme();
  const isMobile = useMediaQuery(theme.breakpoints.down('sm'));

  return (
    <div>
      {isMobile ? 'Modo Mobile' : 'Modo Desktop'}
    </div>
  );
}

export default Component;
```

USEMEDIAQUERY

O Material UI fornece breakpoints pré-definidos:

| Chave | Largura (px) | Dispositivo |
|-------|--------------|---------------|
| xs | 0-599 | Celular |
| sm | 600-899 | Tablet |
| md | 900-1199 | Laptop |
| lg | 1200-1535 | Desktop |
| xl | 1536+ | Telas Grandes |

USEMEDIAQUERY

```
// Telas menores que 'sm' (mobile)
useMediaQuery(theme.breakpoints.down('sm'));

// Telas maiores que 'md' (desktop)
useMediaQuery(theme.breakpoints.up('md'));

// Intervalo específico
useMediaQuery(theme.breakpoints.between('sm', 'md'));

// Valor exato (não recomendado para responsividade)
useMediaQuery(theme.breakpoints.only('md'));
```

```
//Consultas personalizadas
const prefersDarkMode = useMediaQuery('(prefers-color-scheme: dark)');
const isPortrait = useMediaQuery('(orientation: portrait)');
const highResolution = useMediaQuery('(min-resolution: 2dppx)');
```

USEMEDIAQUERY

```
import { Grid, Container, Typography, Paper, Box, useTheme, useMediaQuery } from '@mui/material';

export const ResponsiveGrid = () => {
  const theme = useTheme();
  const isMobile = useMediaQuery(theme.breakpoints.down('sm'));

  // Dados de exemplo para o grid
  const items = [
    { id: 1, title: 'Item 1', content: 'Conteúdo do item 1' },
    { id: 2, title: 'Item 2', content: 'Conteúdo do item 2' },
    { id: 3, title: 'Item 3', content: 'Conteúdo do item 3' },
    { id: 4, title: 'Item 4', content: 'Conteúdo do item 4' },
    { id: 5, title: 'Item 5', content: 'Conteúdo do item 5' },
    { id: 6, title: 'Item 6', content: 'Conteúdo do item 6' },
  ];

  return (
    <Container maxWidth="lg" sx={{ py: 4 }}>
      <Typography variant="h4" component="h1" gutterBottom>
        Grid Responsivo
      </Typography>
    </Container>
  );
}
```

USEMEDIAQUERY

```
/* Grid Container */  
<Grid container spacing={isMobile ? 2 : 3}>  
  {items.map((item) => (  
    <Grid  
      item  
      key={item.id}  
      xs={12} // 1 coluna em mobile  
      sm={6}  // 2 colunas em tablet  
      md={4}  // 3 colunas em laptop  
      lg={3}  // 4 colunas em desktop  
    >  
      /* Componente de exemplo - pode ser substituído por qualquer conteúdo */  
      <Paper  
        sx={{  
          p: 2,  
          height: '100%',  
          display: 'flex',  
          flexDirection: 'column',  
          '&:hover': {  
            boxShadow: 4,  
          }  
        }}  
      >  
    </Paper>  
  )  
  )  
</Grid>
```

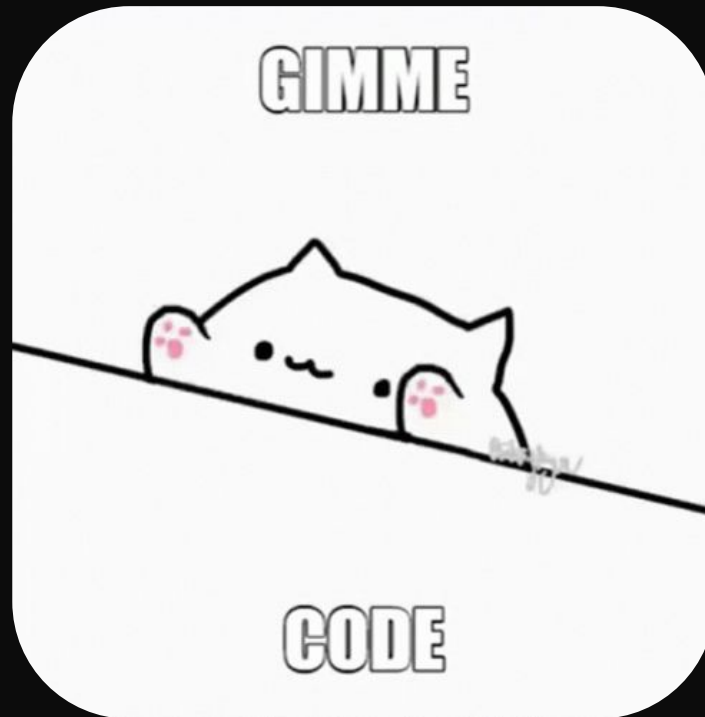
```
    <Typography variant="h6" gutterBottom>  
      {item.title}  
    </Typography>  
    <Typography variant="body1">  
      {item.content}  
    </Typography>  
    <Box sx={{ mt: 2, alignSelf: 'flex-end' }}>  
      <Typography variant="caption">  
        ID: {item.id}  
      </Typography>  
    </Box>  
  </Paper>  
</Grid>  
  )  
</Grid>  
</Container>  
</>  
);  
};
```

TREINANDO NOSSAS HABILIDADES!

Agora temos a missão de criar um layout responsivo para a aplicação que criamos na aula passada!

Você pode, se quiser, também usar o Material UI para adicionar ou modificar pontos que achar necessário na aplicação.

Foque em uma versão para mobile e uma para desktop!



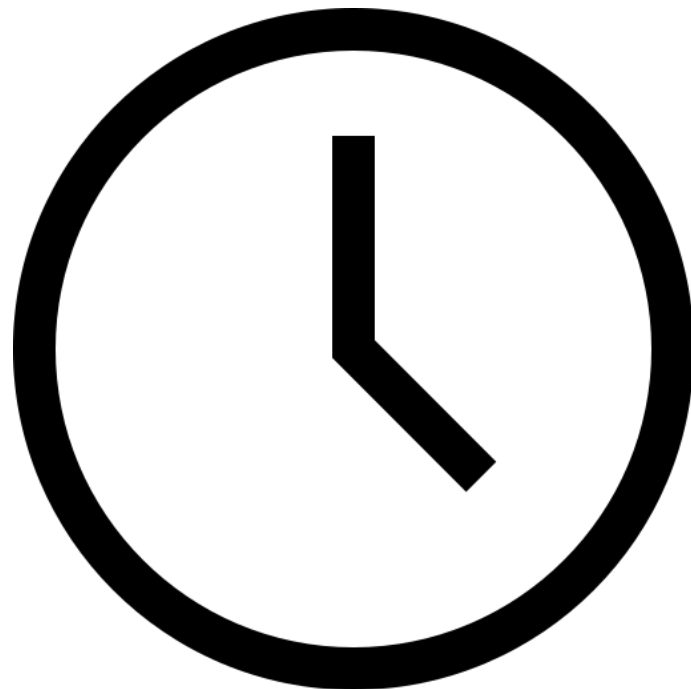
INTERVALO!

Finalizamos o nosso primeiro período de hoje. Que tal descansar um pouco?!

Nos vemos em 20 minutos.

Início: 20:20

Retorno: 20:40



REVISÃO

O que estamos aprendendo no React!

- Criação de Projetos
- Componentes Funcionais
- Props
- Tipagem com TypeScript
- Hooks
 - useState
 - useEffect
- Componentes controlados e não controlados
- Hooks Customizados
- Atomic Design
- CSS Modules
- Material UI
- useMediaQuery

VAMOS “COMPONENTIZAR” O MINI-PROJETO?



VAMOS CODAR!!!

AVALIAÇÃO DOCENTE

O que você está achando das minhas aulas neste conteúdo?

Clique [aqui](#) ou escaneie o QRCode ao lado para avaliar minha aula.

Sinta-se à vontade para fornecer uma avaliação sempre que achar necessário.



<LAB365>

<LAB365>

SENAI