ChatGPT

# Relatório de Inconsistências de Estilo

## Objetivo

Este relatório analisa o **padrão de estilos** do repositório `ProjetoHubx` para verificar se os templates de listas (e outras páginas) estão alinhados com o layout desejado. O objetivo final é alcançar um **layout unificado**, conforme a ilustração abaixo, com barra lateral fixa, cabeçalho superior, seção *hero* e grade de cards responsiva.

As análises consideraram os arquivos CSS e templates HTML da aplicação, com foco em:

- **Uso de cores e classes utilitárias** do Tailwind e do design system (`hubx.css`);
- **Componentes reutilizáveis** (cards, botões, modais, formulários);
- **Padronização de listas** (grade de cards vs tabelas);
- **Suporte a tema claro/escuro**;
- **Acessibilidade** e uso de semântica HTML.

## Resumo do layout esperado

O layout almejado segue as diretrizes definidas para Hubx.Space:

- **Navegação lateral fixa e colapsável**: menu à esquerda, com ícones e etiquetas traduzidas; cores e hover baseados em variáveis do tema;
- **Cabeçalho superior**: logotipo, campo de busca, seletor de tema, notificações e avatar do usuário;
- **Seção *Hero***: gradiente azul (de `#3b82f6` para `#1e40af`), título/subtítulo e ações contextuais;
- **Grade de cards**: lista de itens dispostos em `grid` responsivo (`card-grid`) com espaçamento consistente e cartões de canto arredondado;

- **Suporte a tema claro/escuro** e **acessibilidade** (foco visível, `aria-*`);
- **Parciais reutilizáveis** para busca, paginação, KPIs e cards de entidades (empresa, núcleo, evento, associado).

## Mapeamento de estilos existentes

### Design system (`hubx.css`)

O arquivo `app/static/css/hubx.css` concentra a base do design:

- **Variáveis CSS para cores**, tipografia, espaçamento, bordas e sombras, com variantes de tema claro e escuro;
- **Reset e estilos base** aplicados via `@layer base` com Tailwind (`m-0 p-0 box-border`, tipografia, links);
- **Componentes utilitários** em `@layer components` (p.ex. `.card`, `.btn`, `.alert`, `.badge`, `.modal`, `.dropdown`), usando `@apply` para combinar classes Tailwind e variáveis personalizadas;
- Suporte a **modais**, **dropdowns**, **botões** e **alertas** com variações (primário, secundário, sucesso, aviso, erro);
- Definição de classes utilitárias para contêineres (`.container`, `.container-sm`, `.container-lg`) e estados (`#sidebar a[aria-current='page']`).

### Base layout (`templates/base.html`)

O layout base implementa:

- **Carga dinâmica de tema** via `localStorage` (`claro`, `escuro` ou `automático`) com classes `dark` no `<html>`;
- Inclusão do **Tailwind** compilado (`static/tailwind.css`) e do design system (`hubx.css` via `@tailwind`);
- **Barra lateral** e **header** definidos em parciais (`nav_sidebar.html` e `hero.html`), inseridos conforme necessidade;
- **Estrutura flexível** para aplicar margem esquerda dependendo da visibilidade da sidebar (`ml-64` ou `ml-0`) e transições na colapsação.

### Componentes

- `hero.html`: seção com gradiente azul e títulos; permite breadcrumb e ações. Usa classes hardcoded (`from-[#3b82f6] to-[#1e40af]`) e `text-secondary` (dependendo do design system para cor secundária).
- `nav_sidebar.html`: menu vertical com links e ícones; atualmente utiliza classes Tailwind diretas (`hover:bg-slate-100`, `dark:hover:bg-slate-700`, `bg-slate-200`, etc.) em vez de variáveis do design system, causando discrepância de cores.
- `pagination.html` e `search_form.html`: fornecem paginação padrão e campo de busca com HTMX.
- **Cards** (`templates/partials/cards/*.html`): definem componentes visuais para empresas, eventos, núcleos e totais. Variam entre usar classes utilitárias padrão (`bg-white dark:bg-slate-800 p-6 rounded-lg shadow`) e as classes definidas no design system (`card`, `card-body`).

### Uso de Tailwind e classes customizadas

O projeto faz uso extensivo de classes Tailwind (`p-6`, `bg-white`, `rounded-lg`, etc.) e, em muitas páginas, **repete sequências de classes** que poderiam ser substituídas por componentes (`.card`, `.btn-primary`). Isto leva a inconsistências visuais e a um CSS mais difícil de manter.

Alguns exemplos encontrados:

- Em diversas páginas de cadastro e perfil (por exemplo, `accounts/templates/perfil/disable_2fa.html`, `register/termos.html`, etc.) foram definidos contêineres com `class="bg-white dark:bg-slate-800 p-6 rounded-lg shadow"` em vez de utilizar o componente `.card`. A cor `bg-white` e a sombra `shadow` são duplicadas em múltiplos locais.
- O **menu lateral** usa classes Tailwind com cores fixas (`hover:bg-slate-100 dark:hover:bg-slate-700`, `bg-slate-200 dark:bg-slate-800`), ignorando as variáveis `--bg-tertiary` e `--border`, o que impede atualização automática de cores pelo design system.
- Algumas listas utilizam tabelas com classes artesanais (`table-auto w-full divide-y divide-gray-200 dark:divide-gray-700`). Outras usam grade de cards com classes como `grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-6` mas sem encapsular num componente (`.card-grid`).
- Formulários ainda usam classes isoladas (`bg-white dark:bg-slate-800 p-6 rounded-lg`) sem aproveitar o plugin `@tailwindcss/forms` e os estilos centralizados.

## Inconsistências identificadas

### 1. Mistura de cores fixas e variáveis do design system

- Várias páginas aplicam **cores fixas** do Tailwind (`bg-white`, `bg-slate-100`, `bg-slate-200`, `dark:bg-slate-800`) ao invés de utilizar as variáveis CSS (`--bg-secondary`, `--bg-tertiary`) definidas em `hubx.css`. Isso quebra a uniformidade entre tema claro e escuro e dificulta ajustes globais.
- O menu lateral (`nav_sidebar.html`) e alguns cards utilizam `hover:bg-slate-100` e `bg-slate-200`, que destoam do design system. O correto seria usar classes como `bg-[var(--bg-tertiary)]`, `hover:bg-[var(--bg-accent)]` ou definições centralizadas para estados ativos/inativos.

### 2. Repetição de utilitários Tailwind sem componentes reutilizáveis

- Muitos templates especificam manualmente combinações de utilitários para definir um "card" (`bg-white dark:bg-slate-800 p-6 rounded-lg shadow`) ao invés de usar a classe `.card` presente em `hubx.css`. Isso leva a divergências no tamanho do padding, altura das bordas e sombreamento de um módulo para outro.
- A ausência de um **componente de grade** centralizado faz com que diferentes telas definam diferentes grades (por exemplo, alguns usam `grid grid-cols-3 gap-4`, outros `grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-6`).

### 3. Menu lateral e Hero inconsistentes

- O componente **Hero** utiliza cores inline (`from-[#3b82f6] to-[#1e40af]`), enquanto outras partes da interface usam variáveis. Se o gradiente mudar, é necessário editar o template

manualmente. Deveria estar parametrizado em `hubx.css` ou em variáveis (e.g., `--hero-from`, `--hero-to`).
- O **menu lateral** não utiliza os tokens de cor do design system e não aplica as classes de estado definidas para links ativos (`#sidebar a[aria-current='page']` no CSS). Cada item aplica sua própria classe `hover:bg-slate-100` etc., quebrando a consistência.

### 4. Tabelas versus cards

- Há heterogeneidade entre o uso de **tabelas** e **grades de cards**. Para listas simples (nome, status, avatar) recomenda-se sempre usar cards; contudo, alguns templates menores (por exemplo, `empresas/tags_list.html`) usam tabelas, aumentando a complexidade visual. Já outras listas com muitos atributos (inscrições, materiais) continuam usando tabelas, mas sem consistência de estilo (cores de linhas, sombreamento e fontes).
- Tabelas em dark mode utilizam divisores fixos (`divide-gray-700`) que não aproveitam as variáveis do design system.

### 5. Formulários e inputs

- Diversos formulários utilizam classes base do Tailwind (`border`, `rounded-lg`, `p-4`) sem incluir o plugin de formulários da Tailwind (`@tailwindcss/forms`), que já está configurado no projeto. Isso gera campos de tamanhos diferentes e bordas irregulares.
- Em alguns formulários, as mensagens de erro são renderizadas sem as classes de estilo definidas em `hubx.css` (`.errorlist`), resultando em fontes e cores distintas.

### 6. Dark mode implementado parcialmente

- Embora `base.html` configure o modo escuro adicionando a classe `dark` no elemento `<html>`, vários componentes esquecem de utilizar as variantes `dark:` ou as variáveis CSS para cores. Por exemplo, muitos cards definem apenas `bg-white` sem `dark:bg-slate-800`, ou utilizam bordas com cores fixas (`border-gray-200`), tornando a leitura difícil em dark mode.

### 7. Nomenclaturas e organização de classes

- Alguns templates ainda definem estilos embutidos (`style="background:#fff"`) ou classes customizadas (`.purple-bg`, `.light-border`) que não existem no design system. Estes estilos isolados devem ser removidos ou traduzidos para utilitários do Tailwind ou componentes do `hubx.css`.
- O padrão de nomes de parciais e componentes poderia ser mais consistente (`hero_action.html` vs `hero_action`, `card.html` em algumas pastas e `*_card.html` em outras). Essa falta de padronização complica a reutilização dos componentes de estilo.

## Recomendações para unificação de estilos

1. **Centralizar tokens de cor** – Substituir todas as cores fixas (`bg-white`, `bg-slate-100`, `bg-slate-200`, etc.) por variáveis definidas em `hubx.css` (p.ex. `bg-[var(--bg-secondary)]`, `text-[var(--text-primary)]`, `border-[var(--border)]`). Ajustar `nav_sidebar.html` e outros componentes para usar estas variáveis.

2. **Adotar componentes utilitários** – Utilizar consistentemente as classes já definidas em `hubx.css`:

3. `.card`, `.card-header`, `.card-body`, `.card-footer` para todos os contêineres;
4. `.btn`, `.btn-primary`, `.btn-secondary`, `.btn-outline`, etc., em vez de combinar utilitários manualmente;
5. `.container`, `.container-sm`, `.container-lg` para delimitar a largura das páginas;
6. `.card-grid` (ou criar caso ainda não exista) para a grade de cards usada em listas.

7. **Parametrizar o Hero** – Definir variáveis para o gradiente do *hero* (ex.: `--hero-from` e `--hero-to`) dentro do design system. Atualizar `hero.html` para usar `bg-gradient-to-r from-[var(--hero-from)] to-[var(--hero-to)]`. Isto permitirá ajustar cores globalmente.

8. **Unificar menu lateral** – Ajustar `nav_sidebar.html` para usar apenas classes definidas em `hubx.css`:

9. Definir classes como `.sidebar-item` e `.sidebar-item-active` no CSS, usando variáveis para cores e hovers;

10. Aplicar `aria-current="page"` nos itens ativos e centralizar o estilo em um único seletor em `hubx.css` ao invés de inline no template.

11. **Converter tabelas simples em cards** – Onde a lista exibe poucos campos (por exemplo, `empresas/tags_list.html`, `eventos/tarefa_list.html`), migrar para a grade de cards, utilizando um card parcial específico para cada entidade. Quando for necessário manter uma tabela (por exemplo, inscrições com várias colunas), aplicar um wrapper `<div class="overflow-x-auto">` e usar classes utilitárias que referenciem variáveis (`divide-[var(--border)]`).

12. **Uniformizar formulários** – Usar o plugin `@tailwindcss/forms` em todos os campos e centralizar estilos de erro (`.errorlist`) e labels flutuantes em componentes reutilizáveis. Definir um partial `components/form_field.html` ou utilizar macros Django que apliquem as classes certas.

13. **Cobertura do dark mode** – Revisar todos os templates para garantir que qualquer cor ou borda tenha sua variante `dark:` ou use variáveis de cor. Testar o site com `prefers-color-scheme: dark` para identificar campos que não mudam de cor.

14. **Padronizar nomes e organização** – Renomear parciais de forma clara (`card_empresa.html`, `card_nucleo.html` etc.), movendo os componentes para `templates/components/`. Evitar múltiplos arquivos com o mesmo propósito (por exemplo, consolidar `eventos/evento_list.html` e `eventos/eventos_lista.html`).

## Próximos passos

1. **Inventariar componentes** – Criar um catálogo dos componentes visuais disponíveis (cards, botões, formulários, modais) com exemplos de uso.
2. **Refatoração incremental** – Escolher um módulo (por exemplo, `empresas`) e refatorar as telas de lista para usar apenas componentes e variáveis do design system. Validar visualmente no modo claro e escuro.

3. **Remover CSS legado** – Buscar por classes não existentes no design system (por exemplo, `bg-white dark:bg-slate-800 p-6`) e substituí-las por `.card` e outros utilitários. Excluir arquivos CSS antigos ou desnecessários.
4. **Atualizar documentação** – Registrar no repositório (em arquivos Markdown) as diretrizes de estilo, incluindo exemplos de uso de cada componente. Isso ajudará a evitar regressões no futuro.

Ao seguir essas recomendações, o projeto alcançará **uniformidade visual**, facilitará a manutenção do código e aproximará todas as telas ao layout ideal representado na imagem.

---