Documento de Padrão de Codificação
Dream Audiovisual

Durante o desenvolvimento deste projeto, adotaremos as seguintes práticas de codificação para garantir legibilidade, organização e facilidade de manutenção do código. As diretrizes a seguir foram escolhidas com base nos princípios de **Clean Code**, **SOLID** e outras práticas recomendadas em engenharia de software.

---

1. Documentação e Comentários no Código (Obrigatório)

- O código deve ser comentado de forma que outros desenvolvedores consigam entender facilmente o que está sendo feito, especialmente em funções com lógica mais complexa. O objetivo é facilitar a reutilização e manutenção do código. Comentários devem ser objetivos e explicativos, sem descrever o óbvio.

Exemplo:

- `// Calcula o total de fotos selecionadas pelo cliente`

---

2. Padrão de Notação (Obrigatório)

- Para manter consistência entre os membros do grupo, adotaremos os seguintes padrões de nomenclatura:

- `camelCase` para métodos e funções
- `snake_case` para variáveis e atributos
- `PascalCase` para classes
- `UPPER_SNAKE_CASE` para constantes

Exemplo:

- `class EnsaioFotografico { ... }`
- `const MAX_FOTOS = 100`
- `let nome_cliente = 'João'`
- `function criarEnsaio() { ... }`

---

3. Princípio da Responsabilidade Única (SRP - SOLID)

- Cada classe ou função será projetada para ter apenas **uma única responsabilidade**. Ou seja, cada módulo deverá cumprir um único papel no sistema. Isso facilita a testabilidade, manutenção e entendimento do código, além de promover maior coesão entre os componentes.

---

4. Modularização do Código

- O projeto será dividido em **módulos organizados por domínio**, como por exemplo: `fotografo/`, `cliente/`, `ensaio/`, `album/`. Cada módulo conterá seus próprios arquivos. Essa estrutura modular melhora a escalabilidade do sistema e facilita a organização do código.

---

5. Evitar Código Duplicado (DRY - Don't Repeat Yourself)

- Evitar duplicação de código será uma prioridade. Sempre que for identificado um trecho de lógica repetido, ele será extraído para uma função ou classe reutilizável, promovendo reaproveitamento e facilitando a manutenção futura.

---

6. Nomes Significativos (Clean Code)

- Os nomes de variáveis, funções, classes e arquivos devem refletir claramente seu propósito. Evitaremos nomes genéricos como `data1`, `temp`, `var1`, etc. Bons nomes tornam o código mais legível e reduzem a necessidade de comentários explicativos.

Exemplo:
✔️ `criarAlbumParaCliente()`
✖️ `processarDados1()`