Documento de Boas práticas de codificação do projeto LibOnline

1. Padrão de Notação e Estilo de Código

- Seguir as convenções de nomenclatura do ecossistema JavaScript/React:
- Nomes de variáveis, funções e componentes devem ser descritivos, claros e em inglês para manter um padrão universal.

2. Princípio da Responsabilidade Única (SRP – SOLID)

- Cada componente React ou função JavaScript deve ter uma única e bem definida responsabilidade.
- **No Frontend (React):** Evitar "componentes faz-tudo".
- **No Backend (JavaScript):** Separar as responsabilidades em camadas. Por exemplo, um arquivo de `rotas` apenas define os endpoints, um `controller` gerencia a requisição e a resposta (req/res), e um `serviço` contém a lógica de negócio.

3. Comentários e Documentação

- Todo método ou função complexa deve ter um comentário explicando sua responsabilidade e o **motivo** de sua implementação, caso não seja óbvia.
- Evitar comentários óbvios que apenas repetem o que o código já diz. O código deve ser a principal documentação.

4. Código Limpo (Clean Code)

- Funções e componentes devem ser curtos e focados (idealmente, até 30 linhas). Se um componente React se tornar muito grande, deve ser quebrado em componentes menores e mais especializados.
- Evitar código duplicado (Princípio DRY - Don't Repeat Yourself). Lógicas repetidas devem ser abstraídas em funções, serviços ou Hooks customizados.
- Nomes de funções devem revelar sua intenção e fazer apenas uma coisa. Ex: `fetchUserData()` é melhor que `getData()`.
- Evitar o uso de "números mágicos" ou strings literais. Declare-os como constantes com nomes significativos.

5. Tratamento de Erros e Validações

- **No Backend:** A API deve sempre validar os dados de entrada (payloads, params). Em caso de erro, deve retornar códigos de status HTTP apropriados (`400`, `401`, `404`, `500`) e um corpo de resposta JSON com uma mensagem de erro clara.
- **No Frontend (React):** Todas as chamadas de API devem ser envolvidas em blocos `try...catch` (ou usar o `.catch()` de Promises). O estado de erro deve ser gerenciado para fornecer feedback visual claro ao usuário (ex: "Usuário ou senha inválidos"), em vez de exibir erros técnicos no console.

- Validações de formulário (campos obrigatórios, formato de e-mail, etc.) devem ser feitas no frontend antes do envio, para uma melhor experiência do usuário.

6. Reutilização e Modularização de Código

- Sempre que possível, evitar código repetido.
- Criar funções/métodos reutilizáveis e quebrar lógicas grandes em partes menores.