Normativa de Desenvolvimento: **Gerenciador de Contatos**

Este documento estabelece as normas técnicas para o desenvolvimento do projeto. O seguimento destas regras é **obrigatório** e será critério de avaliação nos Pull Requests.

1. Tecnologias e Arquitetura

- **Linguagem:** Java 21+
- **Interface:** JavaFX com FXML
- **Build:** Maven
- **Base de Dados:** SQLite (Local) e MySQL (Remoto)
- **Arquitetura:** MVC Adaptado com Camada de Serviço e Repositório.

- Regras de Arquitetura (Onde codificar?)

- **Model (`model/`):** Apenas classes POJO (Getters, Setters, Construtores). Sem lógica de negócio.
- **Views (`resources/`):** Arquivos `.fxml` e `.css`.
- **Controllers (`controller/`):** Para capturar eventos da tela. **Não** devem fazer chamadas SQL diretas, pois devem chamar o `Service` para isso.
- **Services (`service/`):** Contêm as regra de negócio, as ações com banco ou outros serviços e decidem qual repositório (MysqlRepository ou SqlieteRespository com apenas código SQL (JDBC)).

2. Fluxo de Desenvolvimento (Passo a Passo)

Como proceder para cada tarefa (issue):

Passo 1: Início

1. Visualizar `Issue` que lhe foi atribuída no GitHub.
2. Mover a tarefa de `Ready` para `**In progress**` Em Projects / Gerenciador de Contatos SN

- No computador local, atualize sua branch `main` com o remoto:

```
git checkout main
git pull origin main
```

- Cria uma nova branch para trabalhar a partir da `main` de acordo com a nomenclatura da regra.

Passo 2: Desenvolvimento e Commits

- O aluno trabalha na sua branch local, fazendo as alterações necessárias (ex: editar `ContatoFormController.java` ou `FormatterUtil.java`).
- Faz **commits** frequentes usando o padrão de mensagens definido neste guia.
- Quando termina, envia a branch para o GitHub:

```
git push origin nome-da-sua-branch
```

Passo 3: Pull Request (PR)

- **Ação:** Assim que enviar a branch, o GitHub sugere a criação de um **Pull Request (PR)**.
- **Como:** Abra o PR comparando a sua branch com a `main`.
- **Vínculo Mágico:** Na descrição do PR, escreva: `Closes #NumeroDaIssue` (ex: `Closes #12`). Isso fechará a Issue automaticamente quando aprovado.
- **Revisão:** Adicione o instrutor como "Reviewer" na lateral direita.

Passo 4: Revisão e Merge (Pelo Instrutor)

- O professor recebe o PR e analisa os "Files changed".
- O professor comenta linha por linha, sugerindo melhorias ou pedindo correções.
- **Se houver correções:** O aluno faz novos commits na mesma branch e dá push (o PR atualiza sozinho).
- **Aprovação:** Se o trabalho estiver bom, o "Merge Pull Request" é realizado e a funcionalidade entra no projeto oficial.

3. Padrões de Git e GitHub

🌳 Padrão de Nomes para Branchs

Atualmente é padrão industrial nomear a branch com base no **trabalho**.

Estrutura: `tipo/descricao-curta-kebab-case`

- **Tipos aceites:**
  - `feature/`: Para uma nova funcionalidade.
  - `fix/`: Para a correção de um erro/bug.
  - `docs/`: Para alterações na documentação.
  - `refactor/`: Melhoria de código sem mudar o comportamento visual.

Exemplos:

- ✅ `feature/adicionar-campo-endereco`
- ✅ `fix/validacao-telefone-9-digitos`
- ❌ `joao-trabalho` (Não descreve o que foi feito)

💬 Padrão de Mensagens de Commit (Conventional Commits)

Usaremos este padrão para forçar o pensamento sobre *o quê* está a ser feito.

Estrutura: `tipo(escopo): mensagem no imperativo`

- **`tipo`**:
  - `feat`: Nova funcionalidade.
  - `fix`: Correção de bug.
  - `style`: CSS, formatação, espaços em branco.
  - `refactor`: Melhoria de código.
  - `docs`: Alteração em README ou documentação.
- **`(escopo)`** (Opcional): Onde mexeu (`form`, `service`, `db`, `list`).
- **`mensagem`**: Curta, letra minúscula, no imperativo (como uma ordem).

Exemplos de bons commits:

- `feat(form): adiciona campo 'endereco' ao contato-form-view`
- `fix(validation): corrige regex de email em ValidationUtil`
- `style(theme): ajusta cor dos botões no tema escuro`
- `refactor(service): otimiza método de salvar contato`

4. Padrões de Nomenclatura no Código e Clean Code

Mantenha a consistência com o código existente.

1. **Classes:** PascalCase (ex: `ContatoService`, `MainController`).
2. **Métodos e Variáveis:** camelCase (ex: `listarContatos`, `telefoneField`).
3. **Constantes:** UPPER_SNAKE_CASE (ex: `COLOR_BG_DARK` em `ConfigService.java`).
4. **FXML IDs:** Devem ter o sufixo do tipo do componente para fácil identificação no Controller.
   - `btnSalvar` (Button)
   - `lblTitulo` (Label)
   - `txtNome` (TextField)
   - `colTelefone` (TableColumn)

Antes de enviar o PR:

1. Remova *imports* não utilizados.
2. Formate o código (na IDE: CTRL+ALT+L ou equivalente).
3. Verifique se não deixou `System.out.println` de debug esquecidos.

> **Observação:** Em caso de dúvida, consulte o código existente como referência ou pergunte na Issue antes de começar a codificar.