Normativa de Desenvolvimento: **Gerenciador de Contatos**

Este documento estabelece as normas técnicas e o fluxo de trabalho 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 (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; devem chamar o `Service`.
- **Services (`service/`):** Contêm as regras de negócio, validações e decidem qual repositório usar.

---

2. Organização das Equipes (Branches de Integração)

Para manter a organização, trabalharemos com **Branches de Equipe**. Você **não** deve criar a sua branch a partir da `main`, mas sim a partir da branch da sua equipe.

- ■ **Equipe 1:** `feat/data-iex-management` (Dados, Importação, Exportação)
- ■ **Equipe 2:** `feat/search-features` (Busca, Filtros)
- ■ **Equipe 3:** `refactor/ui-refactor` (UI, Refatoração, Configuração)

---

3. Fluxo de Desenvolvimento (Passo a Passo)

Siga este roteiro rigorosamente para cada tarefa (issue):

Passo 1: Início e Preparação

1. Visualize a `Issue` atribuída a você no GitHub e mova-a para **In Progress** no Project Board.
2. Identifique qual é a sua **Equipe** (1, 2 ou 3).
3. No seu terminal, atualize o repositório e **mude para a branch da sua equipe**:

```
git fetch origin
git checkout tipo/branch-da-sua-equipe
git pull origin tipo/branch-da-sua-equipe
```

(Exemplo: se você é da Equipe 1, fará `git checkout feat/data-iex-management`)

Passo 2: Criação da Branch de Trabalho

Crie a sua branch de desenvolvimento **a partir da branch da equipe**:

```
git checkout -b tipo/nome-da-sua-tarefa
```

(Verifique a secção "Padrões de Git" abaixo para saber como nomear).

Passo 3: Desenvolvimento e Commits

- Codifique a solução na sua IDE.
- Faça commits atômicos (pequenos) e frequentes seguindo o padrão Conventional Commits.

Exemplo: Criando um botão de exportar

1. *Adicionou o botão no arquivo visual (.fxml):*

```
git add .
git commit -m "feat(ui): adiciona botao exportar na tela principal"
```

2. *Programou a função do botão no Java (.java):*

```
git add .
git commit -m "feat(controller): implementa logica de clique do botao
exportar"
```

Passo 4: Envio (Push) e Pull Request

1. Envie a sua branch para o GitHub:

```
git push origin tipo/nome-da-sua-tarefa
```

2. Vá ao GitHub e abra um **Pull Request (PR)**.
3. ⚠ **IMPORTANTE:** Na seleção de branches do PR:
   - **Base:** Selecione a branch da sua Equipe (ex: `feat/data-iex-management`). **NÃO selecione `main`**.
   - **Compare:** Selecione a sua branch (ex: `feat/botao-importar`).
4. **Descrição:** Escreva `Closes #NumeroDaIssue` no corpo do texto.
5. **Reviewer:** Adicione o **Tech Lead (o Líder)** da sua equipe como revisor.

Passo 5: Revisão e Merge

- O Tech Lead revisa o código. Se houver pedidos de alteração, faça-os localmente e dê `push` novamente na mesma branch.
- Após aprovação do Tech Lead, o PR é mesclado na branch da Equipe.
- *(Apenas o Professor fará o merge final das branches de Equipe para a `main`).*

---

4. Padrões de Git e GitHub

Padrão de Nomes para Branches

Nomeie a branch com base no **trabalho** a ser realizado.

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

- **Tipos aceites:**
  - `feat/`: Para uma nova funcionalidade.
  - `fix/`: Para a correção de um erro/bug.
  - `docs/`: Para alterações na documentação.
  - `refactor/`: Melhoria de código técnica.

Exemplos:

- ✅ `feat/adicionar-campo-endereco`
- ✅ `fix/validacao-telefone-9-digitos`

Padrão de Mensagens de Commit

Use o padrão **Conventional Commits**.

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

- **Tipos:** `feat`, `fix`, `style`, `refactor`, `docs`.
- **Mensagem:** Curta, letra minúscula.

Exemplos:

- `feat(form): adiciona campo 'endereco' ao contato-form-view`
- `refactor(service): otimiza método de salvar contato`

---

5. Exemplo Prático Completo

Cenário: O aluno *João* (da Equipe 1) vai criar o botão de exportar contatos (Issue #10).

1. **Preparar:**

```
git fetch origin
git checkout feat/data-management
git pull origin feat/data-management
```

2. **Criar Branch:**

```
git checkout -b feat/botao-exportar
```

3. **Codificar e Commitar:**
   (João edita os arquivos...)

```
git add .
git commit -m "feat(ui): adiciona botao exportar na tela principal"
```

4. **Enviar:**

```
git push origin feat/botao-exportar
```

5. **No GitHub (Pull Request):**
   - João seleciona **Base:** `feat/data-management` <- **Compare:** `feat/botao-exportar`.
   - Adiciona o Tech Lead como Reviewer.
   - Escreve na descrição: `Closes #10`.

6. **Aprovação:**
    - O Tech Lead aprova. O código de João entra na branch `feat/data-management`.
    - A Issue #10 é fechada.