BookShelf (Parte 3)

Visão Geral

Nesta terceira parte do projeto BookShelf, você realizará uma migração fundamental na arquitetura da aplicação: **substituir o armazenamento em arquivos JSON por um banco de dados SQLite usando Prisma ORM**.

Esta migração proporcionará:

- **Performance superior** com consultas otimizadas e indexadas
- **Integridade de dados** com validações e relacionamentos
- **Escalabilidade** para futuras expansões
- **Type safety** completa com tipos gerados automaticamente
- **Funcionalidades avançadas** de rastreamento de leitura

Objetivos da Migração

1. Preservação de Dados

- **Manter 100% dos dados existentes** durante a migração
- Preservar todos os livros já cadastrados
- Manter todas as categorias/gêneros existentes
- Não perder nenhuma informação ou configuração

2. Expansão do Modelo de Dados

- **Implementar relacionamentos** adequados entre entidades

3. Modernização da Arquitetura

- **Substituir operações de arquivo** por queries de banco
- **Implementar ORM** para operações type-safe
- **Manter compatibilidade** com toda a interface existente

Parte 1: Configuração do Banco de Dados

Requisitos de Instalação

1.1. Dependências do Prisma

Instalar e configurar as dependências necessárias para:

- **Prisma Client** para operações de banco
- **Prisma CLI** para gerenciamento
- **ts-node** para execução de scripts TypeScript

1.2. Inicialização do Projeto

- Configurar Prisma com SQLite como provider
- Criar estrutura de arquivos adequada
- Definir variáveis de ambiente necessárias

1.3. Configuração do Ambiente

- Configurar arquivo .env com URL do banco
- Definir localização do banco SQLite
- Preparar ambiente para desenvolvimento e produção

Requisitos de Modelagem

1.4. Modelo de Livro Expandido

Expandir o modelo atual de livro com os seguintes campos adicionais:

Campos Existentes (preservar):

- id, title, author, genre, year, pages, rating, synopsis, cover

Novos Campos Obrigatórios:

- **status**: Enum com opções de status de leitura
- **currentPage**: Página atual da leitura (número)
- **createdAt**: Data de criação do registro
- **updatedAt**: Data da última atualização

Novos Campos Opcionais:

- **isbn**: Código ISBN do livro (string opcional)
- **notes**: Anotações pessoais sobre a leitura (texto longo)

1.5. Modelo de Gêneros

Criar modelo separado para gerenciamento dinâmico de gêneros:

- Permitir criação de novos gêneros
- Garantir unicidade dos nomes
- Facilitar consultas e relatórios

1.6. Status de Leitura como Enum

Implementar enum type-safe para status:

- QUERO_LER (padrão)
- LENDO
- LIDO
- PAUSADO
- ABANDONADO

Requisitos de Migração

1.7. Script de Migração de Dados

Criar script que:

- Leia todos os dados dos arquivos JSON existentes
- Insira no banco de dados preservando IDs originais
- Adicione valores padrão para novos campos
- Execute validações de integridade

1.8. Estratégia de Backup

- Manter arquivos JSON originais como backup
- Renomear arquivos antigos com sufixo "-DEPRECATED"
- Permitir reversão da migração se necessário

Parte 2: Camada de Acesso aos Dados

2.1. Cliente Prisma

Implementar padrão singleton para o cliente Prisma:

- Evitar múltiplas conexões desnecessárias
- Otimizar para desenvolvimento e produção
- Configurar logging adequado para debugging

2.2. Funções de Banco de Dados

Criar camada de abstração com funções para:

Operações de Livros:

- `getBooks()`: Listar todos os livros
- `getBook(id)`: Buscar livro específico
- `createBook(data)`: Criar novo livro
- `updateBook(id, data)`: Atualizar livro existente
- `deleteBook(id)`: Remover livro

Operações de Gêneros:

- `getGenres()`: Listar todos os gêneros
- `createGenre(name)`: Criar novo gênero

Operações de Status:

- `getReadingStatusOptions()`: Obter opções de status disponíveis

2.3. Interfaces TypeScript

Definir interfaces robustas para:

- `CreateBookData`: Dados para criação de livro
- `UpdateBookData`: Dados para atualização (campos opcionais)
- Tipos gerados automaticamente pelo Prisma

Requisitos de Performance

2.4. Otimizações de Query

- Implementar ordenação padrão por data de criação
- Configurar índices apropriados
- Otimizar consultas frequentes

2.5. Tratamento de Erros

- Implementar tratamento robusto de erros
- Fornecer mensagens de erro informativas
- Garantir rollback em operações que falharem

Parte 3: Atualização da Aplicação

Requisitos de Server Actions

3.1. Migração das Server Actions

Atualizar todas as server actions existentes para usar banco de dados:

- Substituir chamadas de API por funções de banco diretas
- Manter mesma funcionalidade e interface
- Preservar validações e tratamento de erros

3.2. Operações CRUD

Garantir que todas as operações funcionem corretamente:

- **Create**: Criação de novos livros com campos expandidos
- **Read**: Listagem e busca de livros
- **Update**: Atualização com novos campos
- **Delete**: Remoção segura de livros

Requisitos de Componentes

3.3. Atualização de Tipos

- Substituir tipos antigos por tipos gerados pelo Prisma

- Atualizar todas as importações
- Garantir type safety completa

3.4. Formulários Expandidos

Adicionar campos aos formulários:

- **Status de leitura**: Dropdown com opções do enum
- **Página atual**: Campo numérico para progresso
- **ISBN**: Campo de texto opcional
- **Notas**: Textarea para anotações pessoais

3.5. Exibição de Dados

Atualizar componentes para exibir novos campos:

- Mostrar status de leitura como badge
- Exibir progresso de leitura quando aplicável
- Incluir notas pessoais na visualização de detalhes

Requisitos de Páginas

3.6. Server Components

Converter páginas para buscar dados diretamente do banco:

- Eliminar uso desnecessário de estado local
- Aproveitar server-side rendering
- Otimizar performance de carregamento

3.7. Compatibilidade de Interface

- Manter toda a interface existente funcionando
- Preservar experiência do usuário
- Não quebrar funcionalidades existentes

Parte 4: API Routes

4.1. Manter API Routes Existentes

- **Não remover** as API routes implementadas na Parte 2

- **Atualizar** para usar banco de dados ao invés de arquivos
- **Preservar** como referência e documentação

4.2. Compatibilidade com Banco

Atualizar todas as API routes para:

- Usar funções do banco de dados
- Manter mesma interface HTTP
- Suportar novos campos do modelo expandido
- Preservar tratamento de erros existente

4.3. Next.js 15 Compatibility

Garantir compatibilidade com Next.js 15:

- Atualizar tipos de parâmetros para async
- Manter funcionamento correto de rotas dinâmicas

Parte 5: Configuração e Deploy

Requisitos de Configuração

5.1. Arquivo .gitignore

Configurar adequadamente para ignorar:

- Arquivos de banco de dados SQLite
- Diretório de tipos gerados pelo Prisma
- Arquivos temporários de desenvolvimento

5.2. Scripts de Package.json

Adicionar scripts úteis para:

- Geração de tipos do Prisma
- Execução de migrações
- Reset do banco de dados
- Execução de seeds

5.3. ESLint Configuration

Configurar ESLint para ignorar:

- Arquivos gerados automaticamente pelo Prisma
- Diretório de migrações
- Evitar warnings desnecessários

5.4. Variáveis de Ambiente

Configurar adequadamente para:

- Desenvolvimento local
- Build de produção
- Deploy em diferentes ambientes