Relatório do Projeto - Sistema de Gerenciamento de Livros

1. Introdução

O **Sistema de Gerenciamento de Livros** é uma API RESTful que permite a criação, leitura, atualização e exclusão (CRUD) de informações sobre livros e autores. O objetivo principal deste projeto é fornecer uma interface simples e eficiente para gerenciar dados de livros, autores e editoras, facilitando o acesso e a manipulação dessas informações.

2. Tecnologias Usadas

- **.NET 6**: Framework utilizado para desenvolver a API.
- **Entity Framework Core**: ORM (Object-Relational Mapping) utilizado para interagir com o banco de dados SQL Server.
- **SQL Server**: Banco de dados relacional utilizado para armazenar as informações.
- **Swagger**: Utilizado para documentar e testar a API, proporcionando uma interface interativa para desenvolvedores.
- **Postman**: Ferramenta utilizada para testar a API durante o desenvolvimento.
- **xUnit e Moq**: Utilizados para implementar testes unitários.

3. Estrutura do Projeto

O projeto está organizado nas seguintes pastas:

- **Controllers**: Contém os controladores da API que gerenciam as requisições e respostas.
- **Models**: Contém as classes de modelo que representam as entidades do banco de dados.
- **Repositories**: Contém as interfaces e implementações do padrão Repository que abstraem o acesso ao banco de dados.
- **Data**: Contém o contexto do banco de dados (DbContext).
- **Tests**: Contém os testes unitários para verificar o funcionamento da API.

4. Funcionalidades

A API possui os seguintes endpoints principais:

Autores

- GET /api/authors: Obtém todos os autores.
- POST /api/authors: Adiciona um novo autor.
- PUT /api/authors/{id}: Atualiza um autor existente.
- DELETE /api/authors/{id}: Remove um autor.

Livros

- GET /api/books: Obtém todos os livros.
- POST /api/books: Adiciona um novo livro.
- PUT /api/books/{id}: Atualiza um livro existente.
- DELETE /api/books/{id}: Remove um livro.

5. Decisões de Design

Durante o desenvolvimento, algumas decisões importantes foram tomadas:

- **Padrão Repository**: Optou-se por usar o padrão Repository para desacoplar a lógica de acesso a dados da lógica de negócios. Isso facilita a manutenção e o teste da aplicação.
- **Injeção de Dependência**: A injeção de dependência foi utilizada para gerenciar as instâncias de repositórios, permitindo uma melhor modularidade e testabilidade do código.
- **Validação de Dados**: Utilizamos anotações de dados para validar as entradas de usuários, assegurando que campos obrigatórios sejam preenchidos corretamente.

6. Testes

A API foi testada extensivamente utilizando o xUnit e Moq. Os seguintes testes foram implementados:

- **BookRepositoryTests**: Testes para validar as operações CRUD do repositório de livros.
    - Adição de livros.
    - Recuperação de todos os livros.
    - Recuperação de um livro por ID.
    - Atualização de um livro.
    - Remoção de um livro.

- **BooksControllerTests**: Testes para verificar a funcionalidade dos controladores, incluindo validação de retornos corretos e tratamento de erros.

Os testes foram executados usando o Test Explorer no Visual Studio, garantindo que todos os métodos funcionassem como esperado.

7. Considerações Finais

Durante o desenvolvimento, alguns desafios foram encontrados, como a configuração da conexão ao banco de dados e a validação dos dados recebidos pela API. Todos esses problemas foram resolvidos através de revisão cuidadosa das configurações e testes.

Futuras melhorias podem incluir a implementação de autenticação e autorização para proteger os endpoints, bem como a adição de funcionalidades como pesquisa avançada de livros e autores.