



Projeto: Blog Simples

Em: 27/11/2024



Autor: Herbert Bruchmann Junior

1. Introdução / Visão Geral

Este projeto é um sistema básico de blog que permite visualizar, criar, editar e excluir postagens. Ele foi desenvolvido com foco nos princípios de orientação a objetos, aplicando **SOLID** e utilizando o **Entity Framework** para manipulação de dados. Além disso, possui notificações em tempo real via [WebSockets](#).

1.1-Objetivo: Demonstrar habilidades técnicas na implementação de soluções seguindo boas práticas de desenvolvimento, como organização de código, uso de padrões arquiteturais e atenção a requisitos específicos.

- **Tecnologias Utilizadas:**
 - **Linguagem:** C# (.NET)
 - **Banco de Dados:** SQL Server (via Entity Framework)
 - **Comunicação em Tempo Real:** WebSockets
 - **Ferramentas de Desenvolvimento:** Visual Studio, Git

2. Arquitetura

- **Tipo de Arquitetura:** Monolítica
- **Organização de Responsabilidades:**
 - **Camada de Domínio:** Lógica de negócios e entidades.
 - **Camada de Aplicação:** Contém os controladores e serviços que conectam a lógica de negócios à interface (API).
 - **Camada de Dados:** Manipulação e persistência de dados utilizando o Entity Framework.
- **Diagrama da Arquitetura:** *(Adicione um diagrama simples para ilustrar as camadas do projeto).*

3. Funcionalidades

- **Autenticação:**
 - Registro de usuários com validação de senha.

- Login e persistência de sessão.
- **Gerenciamento de Postagens:**
 - Criar, editar e excluir postagens (restrito a usuários autenticados).
 - Validação de permissões para edição/exclusão de postagens.
- **Visualização de Postagens:**
 - Qualquer visitante pode visualizar as postagens publicadas.
- **Notificações em Tempo Real:**
 - Envio de mensagens via WebSockets para todos os usuários conectados, informando sobre novas postagens.

4. Configuração do Ambiente

Pré-requisitos

- .NET SDK 7.0 ou superior
- SQL Server (ou outro banco compatível com o Entity Framework)
- Cliente Git
- Postman (opcional, para testar APIs)

Instruções para Configuração

1. Clone o repositório:
git clone <URL_DO_REPOSITORIO>
2. Acesse o diretório do projeto:
cd simple-blog
3. Configure o banco de dados no arquivo appsettings.json:
6. {
7. "ConnectionStrings": {
8. "DefaultConnection": "Server=SEU_SERVIDOR;Database=BlogDB;User
Id=USUARIO;Password=SENHA;"
9. }
10. }
11. Execute as migrações para criar o banco de dados:
12. dotnet ef database update
13. Inicie o servidor:
14. dotnet run

5. Estrutura do Código

- **Diretório Principal:**
- /src
- /Domain -> Entidades e lógica de negócios
- /Application -> Controladores e serviços
- /Infrastructure -> Configuração do EF, WebSockets e banco de dados
- Program.cs -> Configuração inicial da aplicação

- **Principais Classes:**

- **User.cs:** Representa os usuários registrados.
- **Post.cs:** Representa as postagens do blog.
- **PostService.cs:** Contém a lógica de negócios para o gerenciamento de postagens.
- **WebSocketHandler.cs:** Gerencia as conexões WebSocket e envia notificações.

6. Boas Práticas e Princípios Adotados

- **SRP (Princípio da Responsabilidade Única):**
 - Cada classe e método possui uma única responsabilidade, como PostService para lógica de postagens e AuthenticationService para autenticação.
- **DIP (Princípio da Inversão de Dependência):**
 - Interfaces como IPostRepository foram usadas para abstrair a lógica de persistência.
- **Injeção de Dependência:**
 - Configurado no Program.cs, garantindo que os serviços sejam testáveis e desacoplados.

7. Funcionalidade com WebSockets

- **Como Funciona:**
 1. O cliente se conecta via WebSocket ao endpoint configurado (/notifications).
 2. Sempre que uma nova postagem é criada, o servidor envia uma mensagem para todos os clientes conectados.
- **Código de Exemplo:**

```
public async Task NotifyAllAsync(string message)
```

```
{  
    foreach (var socket in _connectedSockets)  
    {  
        if (socket.State == WebSocketState.Open)  
        {  
            await socket.SendAsync(  
                Encoding.UTF8.GetBytes(message),  
                WebSocketMessageType.Text,  
                true,  
                CancellationToken.None  
            );  
        }  
    }  
}
```

8. Testes

- **Testes Unitários:** Foram escritos testes unitários para as classes de serviço, utilizando o framework **xUnit**.
- **Cobertura de Testes:**
 - **PostService:** Cobertura de 90%.
 - **AuthService:** Cobertura de 85%.
- **Como Executar Testes:**
- dotnet test

9. Diferenciais do Projeto

 **Segurança:** Autenticação segura com tokens **JWT**.

 **Alta Performance:** Consultas otimizadas no SQL Server.

 **Portabilidade:** Compatível com Windows, Linux e macOS.

10. Conclusão

Este projeto demonstra a aplicação prática dos princípios SOLID e boas práticas de desenvolvimento em um sistema funcional e escalável.