**1. Estrutura do Projeto (Camadas e Pastas)**

- SimpleBlog/

- SimpleBlog.API/ --> Camada de apresentação (controllers e endpoints)

- SimpleBlog.Application/ --> Camada de lógica de aplicação (serviços e DTOs)

- SimpleBlog.Domain/ --> Camada de domínio (modelos, interfaces e regras de negócio)

- SimpleBlog.Infrastructure/ --> Infraestrutura (acesso a dados e repositórios)

Texto

Descrição gerada automaticamente

**2. Organização das Responsabilidades**

**2.1. Camada API**

* **Responsabilidade:** Gerenciar a interação com o cliente (endpoints HTTP). Essa camada deve ser o ponto de entrada do sistema.
* **Definição:**
  + *Controllers:*
    - AuthController.cs (autenticação)
    - PostsController.cs (gerenciamento de postagens)
    - NotificationsController.cs (notificações)

**2.2. Camada Application**

* **Responsabilidade:** Contém a lógica da aplicação, aplicando o **SRP**.
* **Definição:**
  + *Serviços:*
    - AuthService.cs (autenticação)
    - PostManagementService.cs (gerenciar postagens)
    - NotificationService.cs (notificações em tempo real)
  + *DTOs:*
    - UserLoginDTO.cs
    - PostCreationDTO.cs

**2.3. Camada Domain**

* **Responsabilidade:** Modelar as entidades do domínio e abstrair as regras de negócio, seguindo o **DIP**.
* **Definição:**
  + *Entidades:*
    - User.cs
    - Post.cs
    - Notification.cs
  + *Interfaces:*
    - IAuthRepository.cs (repositório de autenticação)
    - IPostRepository.cs (repositório de postagens)

**2.4. Camada Infrastructure**

* **Responsabilidade:** Implementar o acesso a dados (ORM: Entity Framework).
* **Definição:**
  + *Repositórios:*
    - AuthRepository.cs
    - PostRepository.cs
  + *DbContext:*
    - SimpleBlogDbContext.cs

**3. Configuração de Dependências**

1. **Inversão de Dependência (DIP):**
   * Utilizo **interfaces** para abstrair repositórios e serviços.
   * Exemplos:
   * public interface IAuthRepository { Task<User> GetUserAsync(string username); }
   * public class AuthRepository : IAuthRepository { /\* implementação \*/ }

Interface gráfica do usuário, Texto, Site

Descrição gerada automaticamente

1. **Injeção de Dependências:**
   * Configure as dependências no Startup.cs ou Program.cs:
   * services.AddScoped<IAuthRepository, AuthRepository>();
   * services.AddScoped<IPostRepository, PostRepository>();
   * services.AddScoped<IAuthService, AuthService>();

**4. Configuração do Entity Framework**

1. **Definição do DbContext:**
2. public class SimpleBlogDbContext : DbContext
3. {
4. public SimpleBlogDbContext(DbContextOptions<SimpleBlogDbContext> options) : base(options) { }
5. public DbSet<User> Users { get; set; }
6. public DbSet<Post> Posts { get; set; }
7. public DbSet<Notification> Notifications { get; set; }
8. }
9. **Models (Domain):**
10. public class User
11. {
12. public int Id { get; set; }
13. public string Username { get; set; }
14. public string PasswordHash { get; set; }
15. public ICollection<Post> Posts { get; set; }
16. }
17. public class Post
18. {
19. public int Id { get; set; }
20. public string Title { get; set; }
21. public string Content { get; set; }
22. public DateTime CreatedAt { get; set; }
23. public int UserId { get; set; }
24. public User User { get; set; }
25. }
26. public class Notification
27. {
28. public int Id { get; set; }
29. public string Message { get; set; }
30. public DateTime CreatedAt { get; set; }
31. }
32. **Configuração do Banco no appsettings.json:**
33. {
34. "ConnectionStrings": {
35. "DefaultConnection": "Server=localhost;Database=SimpleBlogDb;Trusted\_Connection=True;"
36. }
37. }
38. **Iniciar o EF Core:**
39. services.AddDbContext<SimpleBlogDbContext>(options =>
40. options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

**5. Exemplo de Funcionalidades**

**Autenticação (API + Serviço)**

**AuthController.cs:**

[ApiController]

[Route("api/[controller]")]

public class AuthController : ControllerBase

{

private readonly IAuthService \_authService;

public AuthController(IAuthService authService)

{

\_authService = authService;

}

[HttpPost("login")]

public async Task<IActionResult> Login([FromBody] UserLoginDTO loginDTO)

{

var token = await \_authService.AuthenticateAsync(loginDTO);

if (token == null) return Unauthorized();

return Ok(new { Token = token });

}

}

**AuthService.cs:**

public class AuthService : IAuthService

{

private readonly IAuthRepository \_authRepository;

public AuthService(IAuthRepository authRepository)

{

\_authRepository = authRepository;

}

public async Task<string> AuthenticateAsync(UserLoginDTO loginDTO)

{

var user = await \_authRepository.GetUserAsync(loginDTO.Username);

if (user == null || !VerifyPassword(user.PasswordHash, loginDTO.Password)) return null;

return GenerateToken(user);

}

private bool VerifyPassword(string passwordHash, string password)

{

// Lógica para verificar a senha (Hashing, etc.)

}

private string GenerateToken(User user)

{

// Gerar token JWT

}

}