Несколько примеров контроллеров. Эти классы служат для приёма и обработки WEB запросов.

Контроллер авторизации AccountController

namespace JustInMindApp.Controllers

{

[Route("[controller]")]

[Produces("application/json")]

[ApiController]

public class AccountController : ControllerBase

{

private readonly IUserService \_userService;

public AccountController(IUserService userService)

{

\_userService = userService;

}

[HttpPost("signUp")]

public async Task<IActionResult> SignUp([FromBody] SignUpRequest request)

{

var user = await \_userService.GetByEmailAsync(request.Email);

if (user != null)

{

return BadRequest("User with entered email already exists!");

}

var newUser = new User

{

Name = request.Name,

Surname = request.Surname,

Email = request.Email,

Password = request.Password,

};

await \_userService.InsertAsync(newUser);

return Ok();

}

[HttpPost("signIn")]

public async Task<IActionResult> SignIn([FromBody] SignInRequest request)

{

var user = await \_userService.GetByEmailAndPasswordAsync(request.Email, request.Password);

if (user == null)

{

return NotFound("User is not found!");

}

var token = TokenCreater.CreateToken(user);

var encodedToken = new JwtSecurityTokenHandler().WriteToken(token);

var response = new

{

token = encodedToken,

userName = token.Claims.FirstOrDefault(c => c.Type == ClaimsIdentity.DefaultNameClaimType).Value,

userId = token.Claims.FirstOrDefault(c => c.Type == nameof(JustInMind.Shared.Models.User.Id).ToLower()).Value,

};

return new ObjectResult(response);

}

}

}

Пример класса-сервиса. Он выступает прослойкой между WEB-частью и частью доступа к базе данных.

public class ProjectService : IProjectService

{

private readonly IProjectRepository \_projectRepository;

public ProjectService(IProjectRepository projectRepository)

{

\_projectRepository = projectRepository;

}

public async Task<Project> GetAsync(int id)

{

return await \_projectRepository.GetAsync(id);

}

public async Task<IEnumerable<Project>> GetAllAsync(int userId)

{

return await \_projectRepository.GetAllAsync(userId);

}

public async System.Threading.Tasks.Task InsertAsync(CreateProjectRequest request)

{

await \_projectRepository.InsertAsync(request);

}

public async Task<bool> UpdateAsync(Project entity)

{

return await \_projectRepository.UpdateAsync(entity);

}

public async Task<bool> DeleteAsync(Project entity)

{

return await \_projectRepository.DeleteAsync(entity);

}

public async Task<bool> LeaveAsync(UsersToProjects entity)

{

return await \_projectRepository.LeaveAsync(entity);

}

}

Пример репозитория. Репозитории выступают классами, которые предоставляют доступ к данным в хранилище. В нашем случае это MS SQl Server база.

public class ProjectRepository : IProjectRepository

{

private readonly string connectionString;

public ProjectRepository(string connectionString)

{

this.connectionString = connectionString;

}

public async Task<Project> GetAsync(int id)

{

using var db = new SqlConnection(connectionString);

var project = await db.GetAsync<Project>(id);

return project;

}

public async Task<IEnumerable<Project>> GetAllAsync(int userId)

{

string sql = "SELECT p.Id, p.Name, up.UserId, up.UserRoleId FROM UsersToProjects up " +

"LEFT JOIN Projects p ON p.Id = up.ProjectId " +

"WHERE up.UserId = @UserId";

using var db = new SqlConnection(connectionString);

var projects = await db.QueryAsync<Project>(sql, new { UserId = userId });

return projects;

}

public async Task<int> InsertAsync(CreateProjectRequest request)

{

using var db = new SqlConnection(connectionString);

string sql = "INSERT INTO Projects (Name) " +

"OUTPUT INSERTED.Id " +

$"VALUES ('{request.Name}')";

var projectId = await db.QuerySingleAsync<int>(sql);

sql = "INSERT INTO UsersToProjects (ProjectId, UserId, UserRoleId) " +

"OUTPUT INSERTED.Id " +

$"VALUES ({projectId}, {request.OwnerId}, 5)";

return await db.QuerySingleAsync<int>(sql);

}

public async Task<bool> UpdateAsync(Project entity)

{

using var db = new SqlConnection(connectionString);

return await db.UpdateAsync(entity);

}

public async Task<bool> DeleteAsync(Project entity)

{

using var db = new SqlConnection(connectionString);

return await db.DeleteAsync(entity);

}

public async Task<bool> LeaveAsync(UsersToProjects entity)

{

using var db = new SqlConnection(connectionString);

return await db.DeleteAsync(entity);

}

}

Пример модели сущности в базе данных.

public class Project

{

[Key]

public int Id { get; set; }

public string Name { get; set; }

public int UserId { get; set; }

public int UserRoleId { get; set; }

}