

2ª entrega **Entrega Final**

Integração de Sistemas de Informação

Aluno/os:

21140 - Pedro Vieira Simões
21145 – Gonçalo Moreira da Cunha
21152 – João Carlos da Costa Apresentação

Professor/es: Óscar Ribeiro

Licenciatura em Engenharia de Sistemas Informáticos

Barcelos, dezembro de 2022

IPCA GYM

Resumo

Este trabalho prático, relativo à unidade curricular de **Integração de sistemas de informação**, propende a melhorar a performance de trabalho em equipa num desafio que irá explorar as necessidades de um smart campus, no IPCA e demonstrar técnicas e conceitos abordados inter e extracurricular.

A ideia principal do projeto será um sistema para um ginásio e uma aplicação para os utilizadores. Nesta unidade curricular em específico iremos abordar a construção e interação com a API que irá sustentar o nosso projeto.

Conteúdo

Resumo	4
Índice de figuras	6
1. Introdução	7
1.1. Contextualização	7
1.2. Motivação e Objetivos	7
1.3. Estrutura do Documento	7
2. Produto	8
2.1. Visão do Produto	8
3. Diagramas	9
3.1. Diagrama Entidade-Relação	9
4. Código	11
4.1. Programação por Camadas	11
4.1.1. Backend_IPCA_Gym	11
.....	12
.....	12
.....	12
.....	12
.....	12
4.1.2. LayerBLL	13
.....	14
4.1.3. LayerBOL	15
4.1.4. LayerDAL	16
5. Dependências	22
6. WebServices em SOAP	23
6.1. WebMethod GetAllClientesService() [GET]	23
6.2. WebMethod LoginFuncionarioService() [POST]	26
6.3. WebMethod PatchGinasioService() [PATCH]	29
6.4. WebMethod DeleteClassificacaoService() [DELETE]	31
7. Testes Unitários	33
7.1. Teste GetAllClienteTest	34
7.2. Teste LoginClienteTest	35
7.3. Teste DeleteClassificaçãoTest	36
8. Documentação OpenAPI (SwaggerUI)	37
9. Conclusão	38
10. Bibliografia	38

Índice de figuras

Figura 1 - Diagrama de Entidade-Relação.....	9
Figura 2 - Camadas.....	11
Figura 3 - Lista dos controladores.....	11
Figura 4 - Main	11
Figura 5 - Get dos clientes todos	12
Figura 6 - Get de um cliente pelo ID	12
Figura 7 - Adicionar Cliente.....	12
Figura 8 - Remover Cliente	12
Figura 9 - Alteração dados de um cliente	12
Figura 10 - Camada BLL	13
Figura 11 - Exemplo Utils	13
Figura 12 - Logics Amostrar cliente por ID.....	14
Figura 13 - Logics lista de Clientes	14
Figura 14 - Logics Adicionar Cliente	14
Figura 15 - Logics Remover Cliente.....	14
Figura 16 - Logics Alterar dados do Cliente	14
Figura 17 - Propriedades Ginásio DB	15
Figura 18 - Propriedades Ginásio	15
Figura 19 - Ginasio Service.....	16
Figura 20 - Ginasio Service - GetAll.....	17
Figura 21 - Ginasio Service – By ID	17
Figura 23 - Ginasio Service - Post	18
Figura 22 - Ginasio Service - Patch	18
Figura 24 - Ginasio Service - Delete	19
Figura 25 - Dependência API Layer	22
Figura 26 - Dependência BLL	Erro! Marcador não definido.
Figura 27 - Dependência BOL.....	22
Figura 28 - Modelo de dados auxiliar (Cliente)	23
Figura 29 - WebMethod GetAllClientesService()	24
Figura 30 - XML Result de GetAllClientesService()	25
Figura 31 - Modelo de dados auxiliar (LoginModel)	26
Figura 32 - WebMethod LoginFuncionarioService()	27
Figura 33 - Método auxiliar VerifyPasswordHash().....	27
Figura 34 - Input de dados para LoginFuncionarioService().....	28
Figura 35 - XML Result de LoginFuncionarioService()	28
Figura 36 - XML Result de LoginFuncionarioService() no caso de erro	28
Figura 37 - WebMethod PatchGinásioService().....	29
Figura 38 - Input de dados para PatchGinasioService()	30
Figura 39 - XML Result de PatchGinasioService()	30
Figura 40 - Resultado do PatchGinasioService() na Base de Dados (SQLServer).....	30
Figura 41 - WebMethod DeleteClassificacaoService()	31
Figura 42 - Base de dados antes da execução do Webservice	32
Figura 43 - Input de dados para DeleteClassificacaoService().....	32
Figura 44 - XML Result de DeleteClassificacaoService()	32
Figura 45 - Base de Dados após a execução do DeleteClassificacaoService().....	32
Figura 46 - Execução dos Testes Unitários	33
Figura 47 - Carregamento de informação do appsettings.json para uma configuração	33
Figura 48 - Método teste GetAllClientes()	34
Figura 49 - Método Teste LoginCliente()	35
Figura 50 - Método Teste DeleteClassificacao().....	36

1. Introdução

1.1. Contextualização

Provindo da ideia do projeto inicial, esta unidade curricular tem como propósito a construção da arquitetura do sistema, implementando a API do projeto que irá executar serviços web.

1.2. Motivação e Objetivos

A ideia de um sistema para o ginásio foi originada pela ideia de futuramente o IPCA vir a ter mais instalações à medida que este vai crescendo e desta forma existir uma forma de gerir o mesmo e ainda ajudar os clientes.

Temos por objetivos pessoais:

- Cimentar conhecimentos obtidos ao longo do percurso académico;

Objetivos do projeto:

- Construir a arquitetura do sistema
- Montagem de uma API que suporte serviços web, incluído:
 - Swagger;
 - Base de dados;
 - Autenticação;

1.3. Estrutura do Documento

O documento está estruturado de forma que seja de simples leitura. Existe recurso a referências de material fornecido pelo professor Óscar Ribeiro e/ou referências a excertos de Web grafia.

Este trabalho encontra-se também dividido em grupos, de forma a facilitar a procura e associação face ao material fornecido pelo docente.

2. Produto

2.1. Visão do Produto

Dentro dos subtópicos possíveis encaixados no Smart Campus vai ser abordado a Saúde. Foi decidido toda uma construção em torno do desenvolvimento android que visa à nossa universidade acompanhar a vida saudável e atlética dos estudantes.

O IPKA GYM nasce após notar-se a necessidade desse mesmo acompanhamento e a falta de um setor que permita a atividade aos jovens, no sentido de incentivar aos estudantes a realizar um estilo de vida saudável.

Será então possível aos estudantes terem um acompanhamento mobile da sua atividade física, tal como os diferentes exercícios que pode fazer ao longo do seu treino.

Os gestores do ginásio conseguirão fazer uma monitorização de todas as pessoas inscritas no ginásio, já que, em conjunto com outra unidade curricular, irá ser implementado um sistema externo para gestão de acesso através de um chip/cartão eletrónico.

Este projeto visa alcançar este objetivo através da implementação de uma aplicação Mobile e de hardware de gestão de acessos para que se torne mais cómoda a utilização da mesma.

3. Diagramas

3.1. Diagrama Entidade-Relação

Segue-se abaixo o diagrama de entidade-relação da base de dados do IPCA GYM:

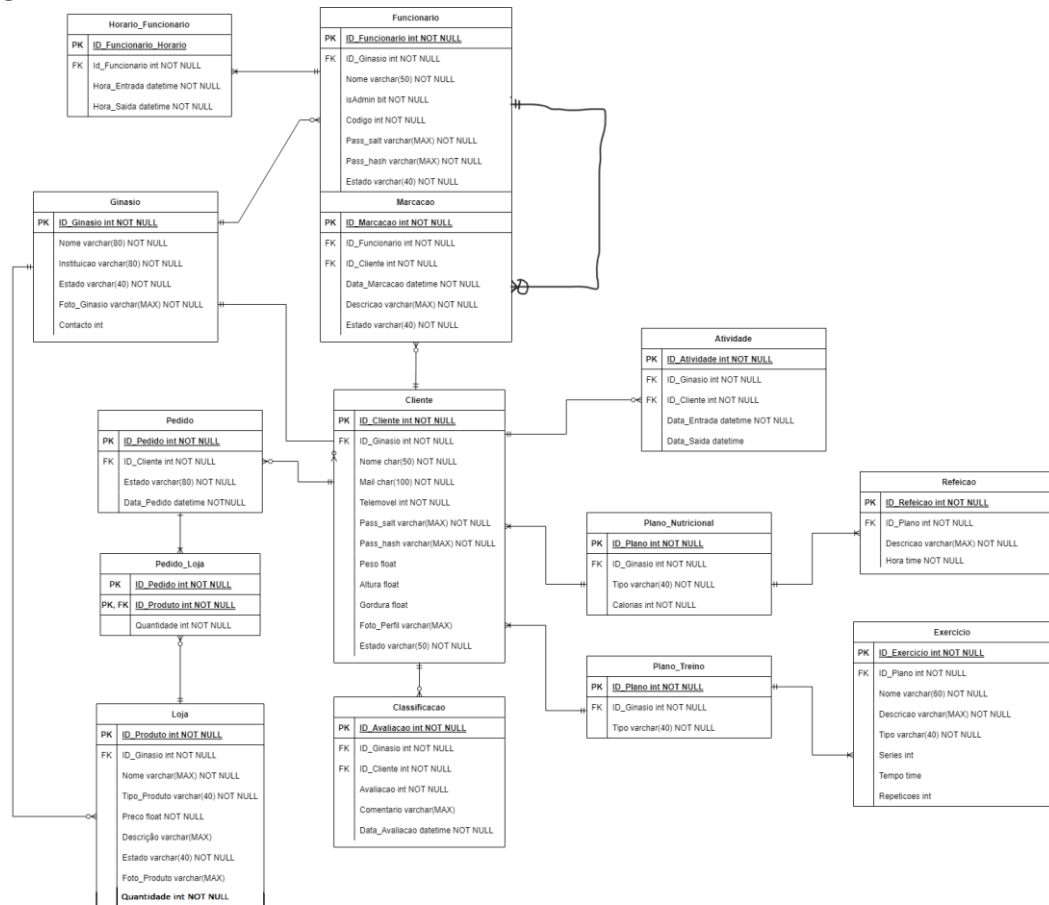


Figura 1 - Diagrama de Entidade-Relação

Como entidades principais este diagrama possui:

- **Cliente** – dados de um cliente que está a utilizar a aplicação;
- **Funcionário** – dados de um funcionário do ginásio em causa, possui o atributo “isAdmin” para determinar se este tem como role Gerente ou não;
- **Ginásio** – dados do ginásio em causa, entidade criada de forma que o projeto, mais tarde, tenha suporte para várias instituições académicas
- **Loja** – possui dados de todos os produtos disponíveis e indisponíveis na loja de cada ginásio
- **Atividade** – entidade criada com o propósito de analisar as entradas e saídas de cada cliente no ginásio (recebe informação do Arduino)

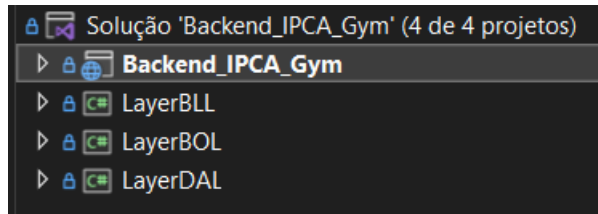
De forma que fosse possível suportar alguns dados sobre outras funcionalidades, foram adicionadas as seguintes entidades:

- **Plano_Nutricional** e Refeição – entidades que possuem dados sobre diferentes refeições e seus horários, cada ginásio define o seu plano nutricional;
- **Plano_Treino** e Exercício – entidades que possuem dados sobre diferentes exercícios e suas descrições, cada ginásio define o seu plano de treino;
- Pedido e **Pedido_Loja** – entidade que possui dados de cada encomenda feita pelo utilizador na loja do ginásio no qual este está inscrito;
- **Horario_Funcionario** – regista o horário de cada funcionário, de forma a verificar a sua disponibilidade para as diferentes marcações;
- **Marcação** – possui a informação de todas as marcações marcadas pelo cliente com o funcionário, associadas a cada ginásio;
- **Classificação** – contém todas as avaliações feitas pelos clientes a cada ginásio.

4. Código

4.1. Programação por Camadas

Neste projeto, como forma de organizar o nosso código, decidimos programar em 4 camadas, isto para que o código fique mais organizado, com melhor performance e mais seguro. A nós permite-nos também detetar anomalias e corrigir problemas de forma mais simples e direta, tudo isto porque é possível substituir partes das camadas (ou a camada toda) sem que o sistema fique todo ele comprometido.



- 1ª Camada - Backend API
- 2ª Camada - BLL
- 3ª Camada - BOL
- 4ª Camada - DAL

Figura 2 - Camadas

4.1.1. Backend_IPCA_Gym

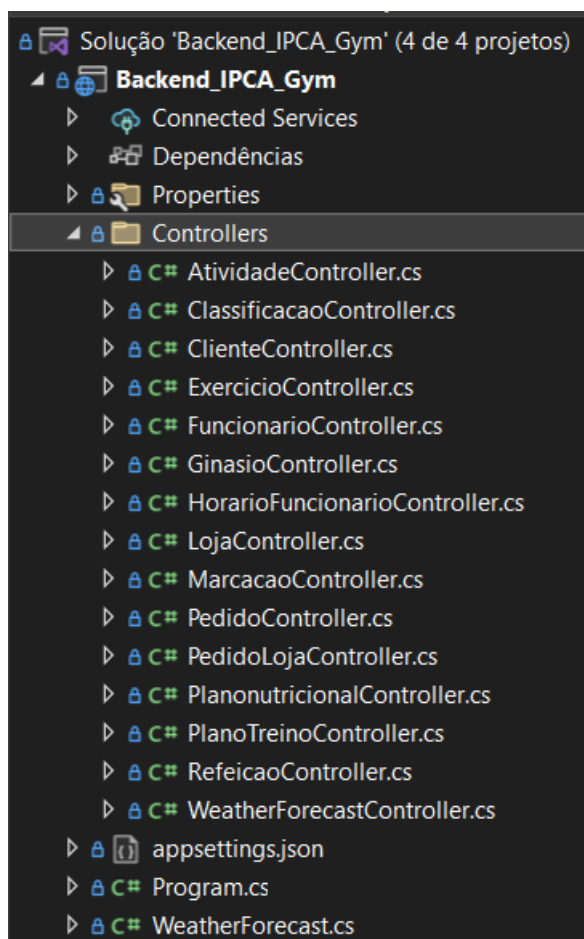


Figura 3 - Lista dos controladores

Na camada "Backend_IPCA_Gym" estão os controllers respetivos para todas as entidades do nosso sistema. Esta camada utiliza funções logic, sendo estas funções providas da camada DAL. Aqui é onde são executadas as chamadas à API. Também é nesta camada que está o nosso main, que é chamado quando executamos o

```
//-----
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = false,
            ValidateIssuerSigningKey = true,

            ValidIssuer = builder.Configuration["Jwt:Issuer"],
            ValidAudience = builder.Configuration["Jwt:Audience"],
            IssuerSigningKey = new SymmetricSecurityKey
            (Encoding.UTF8.GetBytes(builder.Configuration["Jwt:Key"]))
        };
    });

//-----
var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

//app CORS
//app.UseCors("corsapp");

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

Figura 4 - Main

Como referido na página anterior, em cada controlador corresponde a cada parte do nosso projeto e é onde são chamadas as funções que irão fazer a conexão com a bases de dados. Dentro dos controladores temos que colocar (antes de executarmos a chamada), o tipo de request que pretendemos fazer (httpget, httppost, httpdelete...).

- Listagem de todos os clientes

```
public class ClienteController : Controller
{
    private readonly IConfiguration _configuration;
    0 referências
    public ClienteController(IConfiguration configuration)
    {
        _configuration = configuration;
    }

    [HttpGet]
    0 referências
    public async Task<IActionResult> GetAll()
    {
        string sqlDataSource = _configuration.GetConnectionString("DatabaseLink");
        Response response = await ClienteLogic.GetAllLogic(sqlDataSource);

        if (response.StatusCode != LayerBLL.Utills.StatusCodes.SUCCESS) return StatusCode((int)response.StatusCode);

        return new JsonResult(response);
    }
}
```

Figura 5 - Get dos clientes todos

através do seu ID

```
[HttpGet("{targetID}")]
0 referências
public async Task<IActionResult> GetByID(int targetID)
{
    string sqlDataSource = _configuration.GetConnectionString("DatabaseLink");
    Response response = await ClienteLogic.GetByIDLogic(sqlDataSource, targetID);

    if (response.StatusCode != LayerBLL.Utills.StatusCodes.SUCCESS) return StatusCode((int)response.StatusCode);

    return new JsonResult(response);
}
```

Figura 6 - Get de um cliente pelo ID

- Criação de um novo cliente

```
[HttpPost]
0 referências
public async Task<IActionResult> Post([FromBody] Cliente newCliente)
{
    string sqlDataSource = _configuration.GetConnectionString("DatabaseLink");
    Response response = await ClienteLogic.PostLogic(sqlDataSource, newCliente);

    if (response.StatusCode != LayerBLL.Utills.StatusCodes.SUCCESS) return StatusCode((int)response.StatusCode);

    return new JsonResult(response);
}
```

Figura 7 - Adicionar Cliente

- Remoção de um cliente

```
[HttpDelete("{targetID}")]
0 referências
public async Task<IActionResult> Delete(int targetID)
{
    string sqlDataSource = _configuration.GetConnectionString("DatabaseLink");
    Response response = await ClienteLogic.DeleteLogic(sqlDataSource, targetID);

    if (response.StatusCode != LayerBLL.Utills.StatusCodes.SUCCESS) return StatusCode((int)response.StatusCode);

    return new JsonResult(response);
}
```

Figura 8 - Remover Cliente

relativos a um cliente

```
[HttpPatch("{targetID}")]
0 referências
public async Task<IActionResult> Patch([FromBody] Cliente cliente, int targetID)
{
    string sqlDataSource = _configuration.GetConnectionString("DatabaseLink");
    Response response = await ClienteLogic.PatchLogic(sqlDataSource, cliente, targetID);

    if (response.StatusCode != LayerBLL.Utills.StatusCodes.SUCCESS) return StatusCode((int)response.StatusCode);

    return new JsonResult(response);
}
```

Figura 9 - Alteração dados de um cliente

- Listagem de um cliente

- Alteração dos dados

4.1.2. LayerBLL

"BLL" significa "Business Logic Layer" e é outro termo comum usado no desenvolvimento de software para se referir a uma camada na arquitetura de uma aplicação. A camada BLL normalmente é responsável por implementar a lógica de negócios. Isso pode incluir tarefas como validar a entrada do utilizador, realizar cálculos e interagir com a camada de acesso a dados (DAL) para recuperar e armazenar dados e muito mais.

No C#, a camada de lógica de negócios foi implementada como um conjunto de classes que contém os métodos que executam a lógica de negócios da aplicação. Esses métodos são chamados pela camada de apresentação (como uma interface de utilizador) ou por outros componentes na aplicação para executar determinadas tarefas.

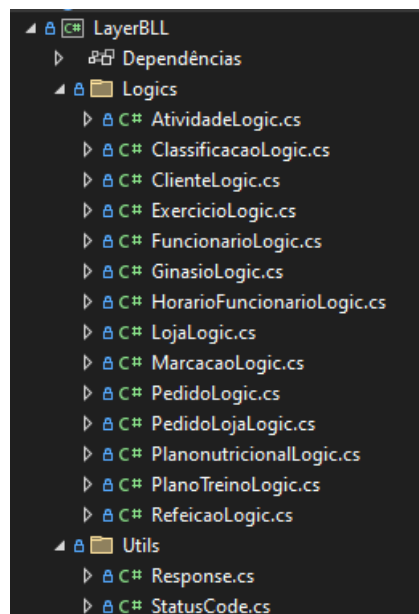


Figura 10 - Camada BLL

- Na parte *Logics* temos as funções logic (funções referidas anteriormente chamadas na camada Backend).
- Na parte *Utils* temos o código que nos dá informação request está ou não a funcionar corretamente.

```
public class Response
{
    99+ referências
    public StatusCodes StatusCode { get; set; }
    72 referências
    public string Message { get; set; }
    72 referências
    public object Data { get; set; }

    /// <summary>
    /// Construtor com dados inicializados
    /// </summary>
    /// <param name="statusCode">Código do estado do request</param>
    /// <param name="message">Mensagem do request</param>
    /// <param name="data">Dados que o request envia</param>
    0 referências
    public Response(StatusCodes statusCode, string message, object data)
    {
        StatusCode = statusCode;
        Message = message;
        Data = data;
    }
}
```

Figura 11 - Exemplo Utils

Apresento agora um exemplo, para um cliente do nosso sistema, do que é necessário para que seja permitido à camada da apresentação fornecer métodos para a camada de negócios.

```
5 referências
public class ClienteLogic
{
    1 referência
    public static async Task<Response> GetAllLogic(string sqlDataSource)
    {
        Response response = new Response();
        List<Cliente> clienteList = await ClienteService.GetAllService(sqlDataSource);

        if (clienteList.Count != 0)
        {
            response.StatusCode = StatusCodes.SUCCESS;
            response.Message = "Lista de clientes obtidos com sucesso";
            response.Data = new JsonResult(clienteList);
        }

        return response;
    }
}
```

Figura 13 - Logics lista de Clientes

```
1 referência
public static async Task<Response> GetByIDLogic(string sqlDataSource, int targetID)
{
    Response response = new Response();
    Cliente cliente = await ClienteService.GetByIDService(sqlDataSource, targetID);

    if (cliente != null)
    {
        response.StatusCode = StatusCodes.SUCCESS;
        response.Message = "Cliente obtido com sucesso";
        response.Data = new JsonResult(cliente);
    }

    return response;
}
```

Figura 12 - Logics Amostrando cliente por ID

```
1 referência
public static async Task<Response> PatchLogic(string sqlDataSource, Cliente cliente, int targetID)
{
    Response response = new Response();
    bool updateResult = await ClienteService.PatchService(sqlDataSource, cliente, targetID);

    if (updateResult)
    {
        response.StatusCode = StatusCodes.SUCCESS;
        response.Message = "Success!";
        response.Data = new JsonResult("Cliente alterado com sucesso");
    }

    return response;
}
```

Figura 16 - Logics Alterar dados do Cliente

```
1 referência
public static async Task<Response> PostLogic(string sqlDataSource, Cliente newCliente)
{
    Response response = new Response();
    bool creationResult = await ClienteService.PostService(sqlDataSource, newCliente);

    if (creationResult)
    {
        response.StatusCode = StatusCodes.SUCCESS;
        response.Message = "Success!";
        response.Data = new JsonResult("Cliente adicionado com sucesso");
    }

    return response;
}
```

Figura 14 - Logics Adicionar Cliente

```
1 referência
public static async Task<Response> DeleteLogic(string sqlDataSource, int targetID)
{
    Response response = new Response();
    bool deleteResult = await ClienteService.DeleteService(sqlDataSource, targetID);

    if (deleteResult)
    {
        response.StatusCode = StatusCodes.SUCCESS;
        response.Message = "Success!";
        response.Data = new JsonResult("Cliente removido com sucesso");
    }

    return response;
}
```

Figura 15 - Logics Remover Cliente

Nestas imagens podemos verificar que em todos os requests (neste caso para o cliente) necessitamos de comunicar com a camada BOL (ClienteService).

4.1.3. LayerBOL

A camada "BOL" significa "Business Object Layer" e é um termo comum usado no desenvolvimento de software referir-se a uma camada que é responsável por representar entidades de negócios e os seus relacionamentos.

O "BOL" normalmente fica entre a camada de apresentação (como uma interface de utilizador) e a camada de acesso a dados (que é responsável pela comunicação com a base de dados ou outro armazenamento de dados).

No C#, a camada do objeto de negócios foi também implementada como um conjunto de classes respetivos a cada entidade do negócio e os relacionamentos entre elas. Nestas classes podemos observar também as propriedades que correspondem aos atributos das entidades de negócios e métodos que executam a lógica de negócios.

Em baixo, e com o código já devidamente comentado, podemos observar as propriedades que a entidade Ginásio tem e estão a ser chamados, ao lado da figura podemos também observar na base de dados do projeto que os campos são os mesmos da tabela.

```
public class Ginasio
{
    /// <summary>
    /// Id do ginásio
    /// </summary>
    /// <example>1</example>
    5 referências
    public int id_ginasio { get; set; }
    /// <summary>
    /// Nome da instituição/estabelecimento
    /// </summary>
    /// <example>UMinho</example>
    6 referências
    public string instituicao { get; set; }
    /// <summary>
    /// Estado do ginásio (Ativo ou Inativo)
    /// </summary>
    /// <example>Ativo</example>
    6 referências
    public string estado { get; set; }
    /// <summary>
    /// Foto do ginásio que poderá ser nula
    /// </summary>
    /// <example>C:\OneDrive\ginasio.png</example>
    9 referências
    public string? foto_ginasio { get; set; }
    /// <summary>
    /// Contacto do ginásio
    /// </summary>
    /// <example>911922933</example>
    6 referências
    public int contacto { get; set; }
}
```

Figura 18 - Propriedades Ginásio

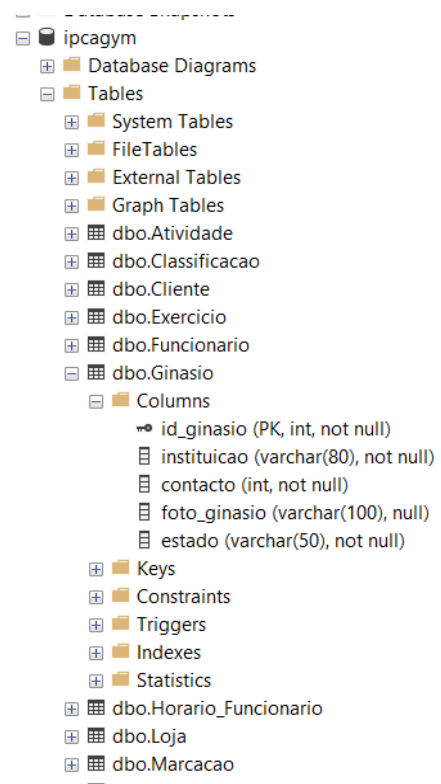


Figura 17 - Propriedades Ginásio DB

4.1.4. LayerDAL

"DAL" significa "Data Access Layer" e é um termo comum usado no desenvolvimento de software para referir-se a uma camada na arquitetura de uma aplicação que é responsável pela comunicação com uma base de dados ou outro armazenamento de dados. A DAL normalmente é responsável por tarefas como executar consultas SQL e interagir com a base de dados para recuperar e armazenar dados.

No C#, a camada de acesso a dados foi, como as restantes, implementada como um conjunto de classes que contém os métodos que executam as interações da base de dados. Esses métodos são chamados pela Business Logic Layer (BLL) e outros componentes na aplicação para recuperar e armazenar dados na base de dados.

```

1 referência
public class GinasioService
{
    /// <summary>
    /// Leitura dos dados de todos os ginásios da base de dados
    /// </summary>
    /// <param name="sqlDataSource">String de conexão à base de dados</param>
    /// <returns>Lista de ginásios se uma leitura bem sucedida, null em caso de erro</returns>
    /// <exception cref="SqlException">Ocorre quando há um erro na conexão com a base de dados.</exception>
    /// <exception cref="InvalidCastException">Ocorre quando há um erro na conversão de dados.</exception>
    /// <exception cref="InvalidOperationException">Trata o caso em que ocorreu um erro de leitura dos dados</exception>
    /// <exception cref="FormatException">Ocorre quando há um erro de tipo de dados.</exception>
    /// <exception cref="IndexOutOfRangeException">Trata o caso em que o índice da coluna da base de dados acessado é inválido</exception>
    /// <exception cref="Exception">Ocorre quando ocorre qualquer outro erro.</exception>
    1 referência
    public static async Task<List<Ginasio>> GetAllService(string sqlDataSource)...

    /// <summary>
    /// Leitura dos dados de um ginásio através do seu id na base de dados
    /// </summary>
    /// <param name="sqlDataSource">String de conexão à base de dados</param>
    /// <param name="targetID">ID do ginásio a ser lido</param>
    /// <returns>Atividade se uma leitura bem sucedida, ou null em caso de erro</returns>
    /// <exception cref="SqlException">Ocorre quando há um erro na conexão com a base de dados.</exception>
    /// <exception cref="InvalidCastException">Ocorre quando há um erro na conversão de dados.</exception>
    /// <exception cref="InvalidOperationException">Trata o caso em que ocorreu um erro de leitura dos dados</exception>
    /// <exception cref="FormatException">Ocorre quando há um erro de tipo de dados.</exception>
    /// <exception cref="IndexOutOfRangeException">Trata o caso em que o índice da coluna da base de dados acessado é inválido</exception>
    /// <exception cref="ArgumentNullException">Ocorre quando um parâmetro é nulo.</exception>
    /// <exception cref="Exception">Ocorre quando ocorre qualquer outro erro.</exception>
    2 referências
    public static async Task<Ginasio> GetByIDService(string sqlDataSource, int targetID)...

    /// <summary>
    /// Inserção dos dados de um novo ginásio na base de dados
    /// </summary>
    /// <param name="sqlDataSource">String de conexão à base de dados</param>
    /// <param name="newGinasio">Objeto com os dados do novo ginásio</param>
    /// <returns>true se a escrita dos dados foi bem sucedida, false em caso de erro.</returns>
    /// <exception cref="SqlException">Ocorre quando há um erro na conexão com a base de dados.</exception>
    /// <exception cref="InvalidCastException">Ocorre quando há um erro na conversão de dados.</exception>
    /// <exception cref="FormatException">Ocorre quando há um erro de tipo de dados.</exception>
    /// <exception cref="ArgumentNullException">Ocorre quando um parâmetro é nulo.</exception>
    /// <exception cref="Exception">Ocorre quando ocorre qualquer outro erro.</exception>
    1 referência
    public static async Task<bool> PostService(string sqlDataSource, Ginasio newGinasio)...
    
```

Figura 19 - Ginasio Service

Resumidamente, nesta camada é onde os dados estão a ser comunicados diretamente com a base de dados. Como se pode observar, todas as funções estão devidamente comentadas do que executam, ou seja, todas as entidades do nosso sistema têm os serviços específicos e necessários para a correta conexão com a base de dados, seja para inserir, remover, listar e editar dados.

Nas figuras seguintes iremos apresentar a forma de como está a ser realizada cada chamada à base de dados.

Começando com a listagem de entidades, neste caso ginásio, construímos 2 funções diferentes, uma para listar todos os ginásios existentes e outra para apresentar apenas um ginásio específico através do seu ID.

```
1 referência
public static async Task<List<HorarioFuncionario>> GetAllService(string sqlDataSource)
{
    string query = @"select * from dbo.Horario_Funcionario";

    try
    {
        List<HorarioFuncionario> horarios = new List<HorarioFuncionario>();
        SqlDataReader dataReader;

        using (SqlConnection databaseConnection = new SqlConnection(sqlDataSource))
        {
            databaseConnection.Open();
            using (SqlCommand myCommand = new SqlCommand(query, databaseConnection))
            {
                dataReader = myCommand.ExecuteReader();
                while (dataReader.Read())
                {
                    HorarioFuncionario dia = new HorarioFuncionario();

                    dia.id_funcionario_horario = Convert.ToInt32(dataReader["id_funcionario_horario"]);
                    dia.id_funcionario = Convert.ToInt32(dataReader["id_funcionario"]);
                    dia.hora_entrada = (TimeSpan)dataReader["hora_entrada"];
                    dia.hora_saida = (TimeSpan)dataReader["hora_saida"];
                    dia.dia_semana = dataReader["dia_semana"].ToString();

                    horarios.Add(dia);
                }

                dataReader.Close();
                databaseConnection.Close();
            }
        }

        return horarios;
    }
}
```

Figura 20 - Ginasio Service - GetAll

```
public static async Task<HorarioFuncionario> GetByIDService(string sqlDataSource, int targetID)
{
    string query = @"select * from dbo.Horario_Funcionario where id_funcionario_horario = @id_funcionario_horario";

    try
    {
        using (SqlConnection databaseConnection = new SqlConnection(sqlDataSource))
        {
            databaseConnection.Open();
            using (SqlCommand myCommand = new SqlCommand(query, databaseConnection))
            {
                Console.WriteLine(targetID);
                myCommand.Parameters.AddWithValue("id_funcionario_horario", targetID);

                using (SqlDataReader reader = myCommand.ExecuteReader())
                {
                    reader.Read();

                    HorarioFuncionario targetHorarioFuncionario = new HorarioFuncionario();
                    targetHorarioFuncionario.id_funcionario_horario = reader.GetInt32(0);
                    targetHorarioFuncionario.id_funcionario = reader.GetInt32(1);
                    targetHorarioFuncionario.hora_entrada = reader.GetTimeSpan(2);
                    targetHorarioFuncionario.hora_saida = reader.GetTimeSpan(3);
                    targetHorarioFuncionario.dia_semana = reader.GetString(4);

                    reader.Close();
                    databaseConnection.Close();
                }

                return targetHorarioFuncionario;
            }
        }
    }
}
```

Figura 21 - Ginasio Service – By ID

Nas duas funções vemos que a forma como estão implementadas é bastante diferente.

Apesar da forma como as queries estão implementadas serem praticamente iguais, vemos que na primeira função temos a necessidade de colocar especificamente que campos desejamos que apareçam, enquanto na segunda função apenas temos de enviar o ID e a ordem de como queremos que sejam apresentadas as colunas da tabela, essa ordem é definida através do “target...”, onde colocamos no reader a numeração que indica a ordem de como as colunas são apresentadas.

Quanto ao método Post (inserção de valores na base de dados), podemos observar na query que temos a necessidade de colocar todos os campos que vamos adicionar a uma entidade específica. Após definidos os campos basta apenas fazer a inserção dos campos através de comandos predefinidos no C#.

```
1 referencia
public static async Task<bool> PostService(string sqlDataSource, HorarioFuncionario newHorarioFuncionario)
{
    string query = @"
        insert into dbo.Horario_Funcionario (id_funcionario, hora_entrada, hora_saida, dia_semana)
        values (@id_funcionario, @hora_entrada, @hora_saida, @dia_semana)";

    try
    {
        SqlDataReader dataReader;
        using (SqlConnection databaseConnection = new SqlConnection(sqlDataSource))
        {
            databaseConnection.Open();
            using (SqlCommand myCommand = new SqlCommand(query, databaseConnection))
            {
                myCommand.Parameters.AddWithValue("id_funcionario", newHorarioFuncionario.id_funcionario);
                myCommand.Parameters.AddWithValue("hora_entrada", newHorarioFuncionario.hora_entrada);
                myCommand.Parameters.AddWithValue("hora_saida", newHorarioFuncionario.hora_saida);
                myCommand.Parameters.AddWithValue("dia_semana", newHorarioFuncionario.dia_semana);

                dataReader = myCommand.ExecuteReader();

                dataReader.Close();
                databaseConnection.Close();
            }
        }

        return true;
    }
}
```

Figura 23 - Ginasio Service - Post

```
public static async Task<bool> PatchService(string sqlDataSource, HorarioFuncionario horarioFuncionario, int targetID)
{
    string query = @"
        update dbo.Horario_Funcionario
        set id_funcionario = @id_funcionario,
        hora_entrada = @hora_entrada,
        hora_saida = @hora_saida,
        dia_semana = @dia_semana
        where id_funcionario_horario = @id_funcionario_horario";

    try
    {
        HorarioFuncionario horarioFuncionarioAtual = await GetByIDService(sqlDataSource, targetID);
        SqlDataReader dataReader;

        using (SqlConnection databaseConnection = new SqlConnection(sqlDataSource))
        {
            databaseConnection.Open();
            using (SqlCommand myCommand = new SqlCommand(query, databaseConnection))
            {
                myCommand.Parameters.AddWithValue("id_funcionario_horario", horarioFuncionario.id_funcionario_horario != 0 ? horarioFuncionario.id_funcionario_horario : horarioFuncionarioAtual.id_funcionario_horario);
                myCommand.Parameters.AddWithValue("id_funcionario", horarioFuncionario.id_funcionario != 0 ? horarioFuncionario.id_funcionario : horarioFuncionarioAtual.id_funcionario);
                myCommand.Parameters.AddWithValue("hora_entrada", horarioFuncionario.hora_entrada != TimeSpan.Zero ? horarioFuncionario.hora_entrada : horarioFuncionarioAtual.hora_entrada);
                myCommand.Parameters.AddWithValue("hora_saida", horarioFuncionario.hora_saida != TimeSpan.Zero ? horarioFuncionario.hora_saida : horarioFuncionarioAtual.hora_saida);
                myCommand.Parameters.AddWithValue("dia_semana", !string.IsNullOrEmpty(horarioFuncionario.dia_semana) ? horarioFuncionario.dia_semana : horarioFuncionarioAtual.dia_semana);

                dataReader = myCommand.ExecuteReader();

                dataReader.Close();
                databaseConnection.Close();
            }
        }

        return true;
    }
}
```

Figura 22 - Ginasio Service - Patch

Na imagem acima vemos como está implementada a função do Patch (alteração de dados relativos a uma entidade). Começamos por criar um objeto que irá ficar temporariamente a ser utilizado, esse objeto é atribuído a um específico através do seu ID onde serão alterados os dados.

Após a alteração de dados é feita a comunicação com a base de dados e alterados os dados correspondentes ao mesmo ID.

```
public static async Task<bool> DeleteService(string sqlDataSource, int targetID)
{
    string query = @"
        delete from dbo.Horario_Funcionario
        where id_funcionario_horario = @id_funcionario_horario";

    try
    {
        SqlDataReader dataReader;

        using (SqlConnection databaseConnection = new SqlConnection(sqlDataSource))
        {
            databaseConnection.Open();
            using (SqlCommand myCommand = new SqlCommand(query, databaseConnection))
            {
                myCommand.Parameters.AddWithValue("id_funcionario_horario", targetID);
                dataReader = myCommand.ExecuteReader();

                dataReader.Close();
                databaseConnection.Close();
            }
        }

        return true;
    }
}
```

Figura 24 - Ginasio Service - Delete

Para a remoção de um registo relativo a uma entidade estamos a utilizar apenas o ID.

5. JWT Tokens

5.1. Implementação

Os requests foram implementados de forma segura e com autenticação, desta forma é possível saber quem é que está a efetuar o request e se o mesmo possui as permissões necessárias para obter os dados pedidos.

Inicialmente foi configurado o Program.cs na implementação de JWT Tokens

```
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme).AddJwtBearer(options =>
{
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8
            .GetBytes(builder.Configuration.GetSection("AppSettings:Token").Value)),
        ValidateIssuer = false,
        ValidateAudience = false
    };
});
```

Figura 25 - AddAuthentication() no Program.cs

```
app.UseAuthentication();

app.UseRouting();

app.UseAuthorization();
```

Figura 26 - app function calls no Program.cs

Para atribuir as diferentes Roles, foi feito num ficheiro à Parte na camada de DAL (Token.cs), para diferentes tipos de utilizadores, neste caso Clientes e Funcionários

```
public static string CreateTokenFuncionario(Funcionario funcionario, IConfiguration _configuration)
{
    string role = string.Empty;

    if (funcionario.is_admin) role = "Gerente";
    else role = "Funcionario";

    if (funcionario.nome == "adminaccount") role = "Admin";

    List<Claim> claims = new List<Claim>
    {
        new Claim(ClaimTypes.Email, funcionario.codigo.ToString()),
        new Claim(ClaimTypes.Role, role),
        new Claim(ClaimTypes.SerialNumber, funcionario.id_ginasio.ToString())
    };

    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration.GetSection("AppSettings:Token").Value));
    var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha512Signature);

    var token = new JwtSecurityToken(
        claims: claims,
        expires: DateTime.Now.AddDays(1),
        signingCredentials: creds);

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

    return jwt;
}
```

Figura 27 - Criação de uma token para um Funcionário

5.2. Utilização de Authorize nos requests

Para utilizar autenticação nos requests e decidir quem tem acesso, é criado um header nos controladores dos modelos:

```
[HttpGet("{targetID}"), Authorize(Roles = "Admin, Cliente")]  
0 references  
public async Task<IActionResult> GetByID(int targetID)
```

Figura 28 - Request com autorizações

Para ser obtida informação a partir da token de sessão que fez o request, é utilizado o `User.HasClaim()` ou `User.FindFirstValue()`

```
[HttpGet("getbytoken"), Authorize(Roles = "Admin, Cliente")]  
0 references  
public async Task<IActionResult> GetClienteByToken()  
{  
    string sqlDataSource = _configuration.GetConnectionString("DatabaseLink");  
    string emailCliente = User.FindFirstValue(ClaimTypes.Email);  
  
    Response response = await ClienteLogic.GetClienteByTokenLogic(sqlDataSource, emailCliente);  
  
    if (response.StatusCode != LayerBLL.Utils.StatusCodes.SUCCESS) return StatusCode((int)response.StatusCode);  
  
    return new JsonResult(response);  
}
```

Figura 29 - Request para obter informação de um Cliente a partir da Token de Sessão

6. Dependências

Tendo em conta que este projeto está dividido em camadas, foi necessário atribuir as dependências a cada camada.

A atribuição de dependências foi a seguinte:

- Camada do Backend_IPCA_Gym (camada da API) depende da camada de Business Logic (BLL)
- Camada do Business Logic depende da camada de Data Access (DAL)
- Camada de Data Access depende da camada de Business Object (BOL)

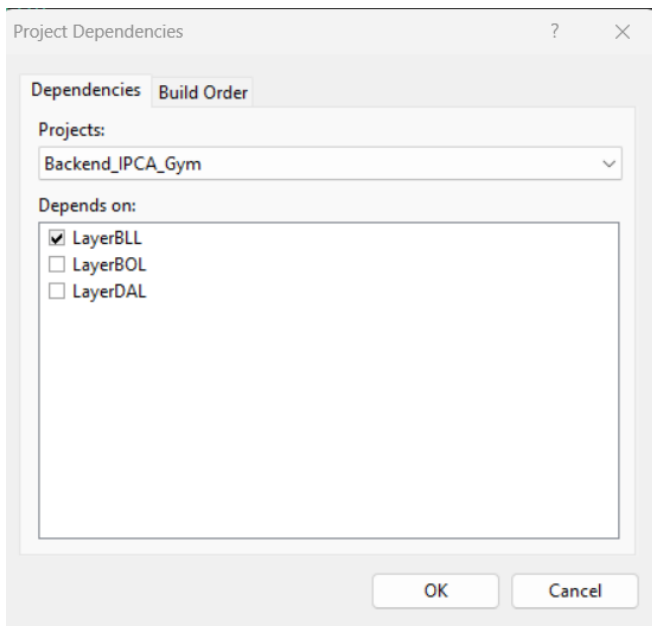


Figura 30 - Dependência API Layer

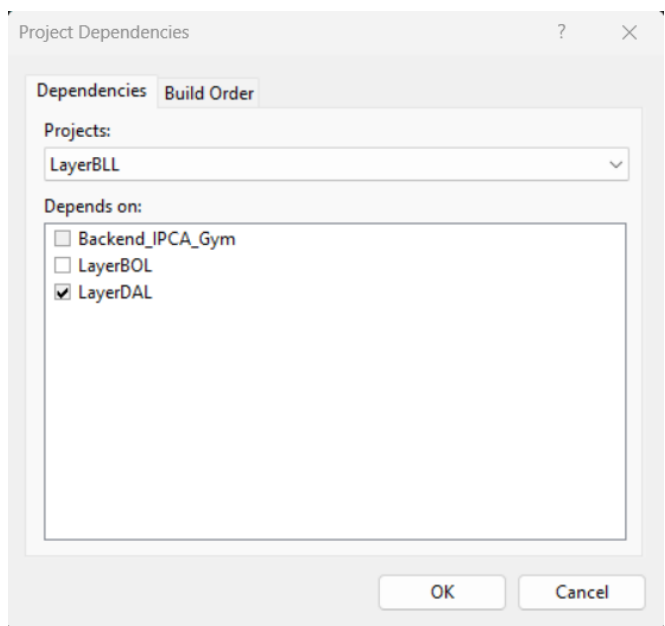


Figura 31 - Dependência BLL

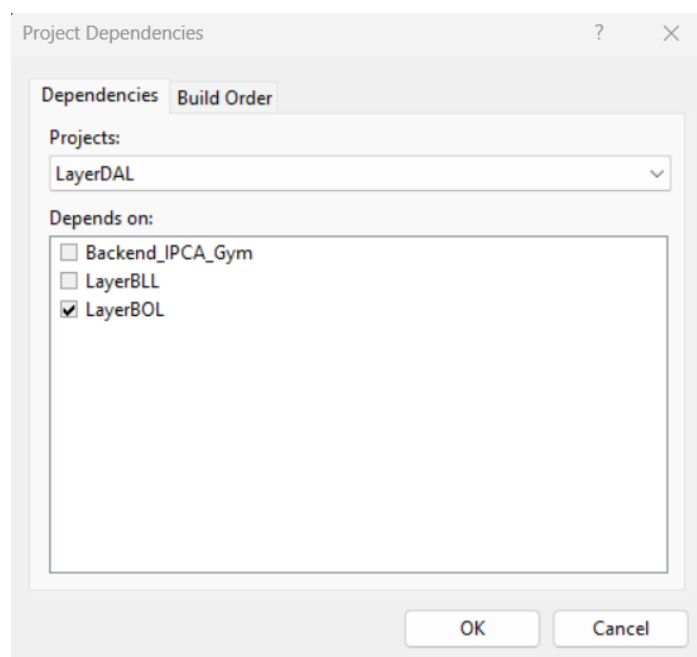


Figura 32 - Dependência BOL

7. WebServices em SOAP

De forma a ser possível alcançar os objetivos que foram propostos no enunciado desta entrega foi implementado um Webservice, neste casos foi feito um WebMethod para cada tipo de request (GET, POST, PATCH, DELETE), dos quais foram escolhidos os seguintes:

- GET – obter uma lista de todos os clientes na base de dados
- POST – efetuar um login de um funcionário e responde com o tipo de conta que o funcionário fez login (Admin, Gerente, Funcionário)
- PATCH – fazer uma edição de um ginásio na base de dados
- DELETE – fazer uma remoção de um comentário da base de dados

7.1. WebMethod GetAllClientesService() [GET]

Neste WebMethod é feita uma busca à base de dados de uma lista de objetos do tipo Cliente:

```
5 references
public class Cliente
{
    1 reference
    public int id_cliente { get; set; }
    1 reference
    public int id_ginasio { get; set; }
    2 references
    public int id_plano_nutricional { get; set; }
    1 reference
    public string nome { get; set; }
    1 reference
    public string mail { get; set; }
    1 reference
    public int telemovel { get; set; }
    1 reference
    public string pass_salt { get; set; }
    1 reference
    public string pass_hash { get; set; }
    2 references
    public double peso { get; set; }
    2 references
    public int altura { get; set; }
    2 references
    public double gordura { get; set; }
    2 references
    public string foto_perfil { get; set; }
    1 reference
    public string estado { get; set; }
}
```

Figura 33 - Modelo de dados auxiliar (Cliente)

Segue-se agora o código desenvolvido para o WebMethod de obter uma lista de todos os clientes:

```
[WebMethod]
public List<Cliente> GetAllClientesService()
{
    string query = @"select * from dbo.Cliente";
    string connectionString = "data source=DESKTOP-8AIBMTG;initial catalog=ipcagym;trusted_connection=true";
    List<Cliente> clientes = new List<Cliente>();

    try
    {
        SqlDataReader dataReader;
        using (SqlConnection databaseConnection = new SqlConnection(connectionString))
        {
            databaseConnection.Open();
            using (SqlCommand myCommand = new SqlCommand(query, databaseConnection))
            {
                dataReader = myCommand.ExecuteReader();
                while (dataReader.Read())
                {
                    Cliente cliente = new Cliente();

                    cliente.id_cliente = Convert.ToInt32(dataReader["id_cliente"]);
                    cliente.id_ginasio = Convert.ToInt32(dataReader["id_ginasio"]);
                    if (!Convert.IsDBNull(dataReader["id_plano_nutricional"]))
                    {
                        cliente.id_plano_nutricional = Convert.ToInt32(dataReader["id_plano_nutricional"]);
                    }
                    else
                    {
                        cliente.id_plano_nutricional = -1;
                    }
                    cliente.nome = dataReader["nome"].ToString();
                    cliente.mail = dataReader["mail"].ToString();
                    cliente.telemovel = Convert.ToInt32(dataReader["telemovel"]);
                    cliente.pass_salt = "It's a secret!";
                    cliente.pass_hash = "Already told you that it's a secret!";
                    if (!Convert.IsDBNull(dataReader["peso"]))
                    {
                        cliente.peso = Convert.ToDouble(dataReader["peso"]);
                    }
                    else
                    {
                        cliente.peso = -1;
                    }
                    if (!Convert.IsDBNull(dataReader["altura"]))
                    {
                        cliente.altura = Convert.ToInt32(dataReader["altura"]);
                    }
                    else
                    {
                        cliente.altura = -1;
                    }
                    if (!Convert.IsDBNull(dataReader["gordura"]))
                    {
                        cliente.gordura = Convert.ToDouble(dataReader["gordura"]);
                    }
                    else
                    {
                        cliente.gordura = -1;
                    }
                    if (!Convert.IsDBNull(dataReader["foto_perfil"]))
                    {
                        cliente.foto_perfil = dataReader["foto_perfil"].ToString();
                    }
                    else
                    {
                        cliente.foto_perfil = null;
                    }
                    cliente.estado = dataReader["estado"].ToString();

                    clientes.Add(cliente);
                }
            }
        }
        dataReader.Close();
        databaseConnection.Close();
    }

    return clientes;
}

catch (SqlException ex)
{
    Console.WriteLine("Erro na conexão com a base de dados: " + ex.Message);
    return null;
}
catch (InvalidCastException ex)
{
    Console.WriteLine("Erro na conversão de dados: " + ex.Message);
    return null;
}
catch (InvalidOperationException ex)
{
    Console.WriteLine("Erro de leitura dos dados: " + ex.Message);
    return null;
}
catch (FormatException ex)
{
    Console.WriteLine("Erro de tipo de dados: " + ex.Message);
    return null;
}
catch (IndexOutOfRangeException ex)
{
    Console.WriteLine("Erro de acesso a uma coluna da base de dados: " + ex.Message);
    return null;
}
catch (Exception ex)
{
    Console.WriteLine(ex.ToString());
    return null;
}
```

Figura 34 - WebMethod GetAllClientesService()

WebService em funcionamento:

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<ArrayOfCliente xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="www.ipca.pt">
  <Cliente>
    <id_cliente>1</id_cliente>
    <id_ginasio>2</id_ginasio>
    <id_plano_nutricional>1</id_plano_nutricional>
    <nome>John Gillman</nome>
    <mail>homelenderisbetter@vought.com</mail>
    <telemovel>922345543</telemovel>
    <pass_salt>It's a secret!</pass_salt>
    <pass_hash>Already told you that it's a secret!</pass_hash>
    <peso>1</peso>
    <altura>1</altura>
    <gordura>1</gordura>
    <estado>Ativo</estado>
  </Cliente>
  <Cliente>
    <id_cliente>2</id_cliente>
    <id_ginasio>2</id_ginasio>
    <id_plano_nutricional>1</id_plano_nutricional>
    <nome>William Butcher</nome>
    <mail>theboys@gmail.com</mail>
    <telemovel>923365547</telemovel>
    <pass_salt>It's a secret!</pass_salt>
    <pass_hash>Already told you that it's a secret!</pass_hash>
    <peso>1</peso>
    <altura>1</altura>
    <gordura>1</gordura>
    <estado>Ativo</estado>
  </Cliente>
  <Cliente>
    ...
  </Cliente>
  <Cliente>
    ...
  </Cliente>
  <Cliente>
    ...
  </Cliente>
  <Cliente>
    ...
  </Cliente>
  <Cliente>
    ...
  </Cliente>
  <Cliente>
    ...
  </Cliente>
  <Cliente>
    ...
  </Cliente>
  <Cliente>
    ...
  </Cliente>
  <Cliente>
    ...
  </Cliente>
  <Cliente>
    ...
  </Cliente>
  <ArrayOfCliente>
```

Figura 35 - XML Result de GetAllClientesService()

7.2. WebMethod LoginFuncionarioService() [POST]

Neste WebMethod é feito um login de funcionário, retornando o tipo de Role que este possui.

Foi escolhido este request de forma a variar os objetivos de cada tipo de request, sendo assim o Login não insere dados na base de dados, todavia, este não convém ser um GET pois recebe um request body.

De forma a dar suporte a esta funcionalidade, foi criada uma class para servir de modelo de dados:

```
/// <summary> Class auxiliar para a recepção de dados de login
4 references
class LoginModel
{
    public bool role;
    public int code;
    public string password;
    public string hash_pass;

    1 reference
    public LoginModel() { }

    0 references
    public LoginModel(bool role, int code, string password, string hash_pass)
    {
        this.role = role;
        this.code = code;
        this.password = password;
        this.hash_pass = hash_pass;
    }
}
```

Figura 36 - Modelo de dados auxiliar (LoginModel)

Segue-se agora o código desenvolvido para efetuar o login de um funcionário:

```
[WebMethod]
1 reference
public string LoginFuncionarioService(string codeInput, string passwordInput)
{
    string query = @"
        select * from dbo.Funcionario
        where codigo = @codigo and estado != 'Inativo';

    string connectionString = "data source=DESKTOP-8AIBMT;initial catalog=ipcagym;trusted_connection=true";
    try
    {
        using (SqlConnection databaseConnection = new SqlConnection(connectionString))
        {
            databaseConnection.Open();
            using (SqlCommand myCommand = new SqlCommand(query, databaseConnection))
            {
                myCommand.Parameters.AddWithValue("codigo", codeInput);
                using (SqlDataReader reader = myCommand.ExecuteReader())
                {
                    reader.Read();

                    string result = string.Empty;
                    LoginModel account = new LoginModel();

                    account.role = reader.GetBoolean(3);
                    account.code = reader.GetInt32(4);
                    account.password = reader.GetString(5);
                    account.hash_pass = reader.GetString(6);
                    string auxForAdmin = reader.GetString(2);

                    reader.Close();
                    databaseConnection.Close();

                    if (!VerifyPasswordHash(passwordInput, Convert.FromBase64String(account.hash_pass), Convert.FromBase64String(account.password)))
                    {
                        throw new ArgumentException("Password Errada.", "conta");
                    }
                    else
                    {
                        if (account.role == true)
                        {
                            if (auxForAdmin == "adminaccount") result = "É admin!";
                            else result = "É gerente!";
                        }
                        else
                        {
                            result = "É funcionário";
                        }
                    }

                    Console.WriteLine(result);
                    return result;
                }
            }
        }
    }
    catch (InvalidOperationException ex)
    {
        Console.WriteLine("Funcionário não existe\n" + ex.Message);
        return "Erro na autenticação";
    }
    catch (SqlException ex)
    {
        Console.WriteLine("Erro na conexão com a base de dados: " + ex.Message);
        return "Erro na autenticação";
    }
    catch (ArgumentNullException ex)
    {
        Console.WriteLine("Erro de parametro inserido nulo: " + ex.Message);
        return "Erro na autenticação";
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
        return "Erro na autenticação";
    }
}
```

Figura 37 - WebMethod LoginFuncionarioService()

Foi ainda utilizado um método auxiliar para fazer a descriptação da palavra-passe:

```
/// <summary> Método auxiliar para verificar se a password está correta
1 reference
public static bool VerifyPasswordHash(string password, byte[] passwordHash, byte[] passwordSalt)
{
    using (var hmac = new HMACSHA512(passwordSalt))
    {
        var computeHash = hmac.ComputeHash(Encoding.UTF8.GetBytes(password));
        return computeHash.SequenceEqual(passwordHash);
    }
}
```

Figura 38 - Método auxiliar VerifyPasswordHash()

WebService em funcionamento:

IPCAGym

Clique [aqui](#) para obter uma lista completa de operações.

LoginFuncionarioService

Testar

Para testar a operação utilizando o protocolo HTTP POST, clique no botão 'Invocar'.

Parâmetro	Valor
codeInput:	<input type="text" value="1"/>
passwordInput:	<input type="text" value="admin"/>
<input type="button" value="Invocar"/>	

SOAP 1.1

Segue-se um exemplo de pedido e resposta SOAP 1.1. É necessário substituir os marcadores de posição mostrados por valores reais.

```
POST /Services/IPCAGymWS.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "www.ipca.pt/LoginFuncionarioService"

<?xml version="1.0" encoding="utf-8">
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <LoginFuncionarioService xmlns="www.ipca.pt">
      <codeInput>string</codeInput>
      <passwordInput>string</passwordInput>
    </LoginFuncionarioService>
  </soap:Body>
</soap:Envelope>

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8">
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <LoginFuncionarioServiceResponse xmlns="www.ipca.pt">
      <LoginFuncionarioServiceResult>string</LoginFuncionarioServiceResult>
    </LoginFuncionarioServiceResponse>
  </soap:Body>
</soap:Envelope>
```

Figura 39 - Input de dados para LoginFuncionarioService()

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<string xmlns="www.ipca.pt">É admin!</string>
```

Figura 40 - XML Result de LoginFuncionarioService()

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<string xmlns="www.ipca.pt">Erro na autenticação</string>
```

Figura 41 - XML Result de LoginFuncionarioService() no caso de erro

7.3. WebMethod PatchGinasioService() [PATCH]

Neste WebMethod é feito um request para fazer a edição de um ginásio.

Segue-se agora o código desenvolvido para efetuar a edição de um ginásio:

```
[WebMethod]
[references]
public bool PatchGinasioService(int targetID, string instituicao, int contacto, string foto, string estado, int lotacao, int lotacaoMax)
{
    string queryPatch = @"
        update dbo.Ginasio
        set instituicao = @instituicao,
        contacto = @contacto,
        foto_ginasio = @foto_ginasio,
        estado = @estado,
        lotacao = @lotacao,
        lotacaoMax = @lotacaoMax
        where id_ginasio = @id_ginasio";

    string queryGet = @"select * from dbo.Ginasio where id_ginasio = @id_ginasio";
    string connectionString = "data source=DESKTOP-8AIBMTG;initial catalog=ipcagyw;trusted_connection=true";

    string instituicaoAtual;
    int contactoAtual;
    string foto_ginasioAtual;
    string estadoAtual;
    int lotacaoAtual;
    int lotacaoMaxAtual;

    try
    {
        SqlDataReader dataReader;

        using (SqlConnection databaseConnection = new SqlConnection(connectionString))
        {
            databaseConnection.Open();
            using (SqlCommand myCommandAux = new SqlCommand(queryGet, databaseConnection))
            {
                myCommandAux.Parameters.AddWithValue("id_ginasio", targetID);

                using (SqlDataReader reader = myCommandAux.ExecuteReader())
                {
                    reader.Read();

                    instituicaoAtual = reader.GetString(1);
                    contactoAtual = reader.GetInt32(2);
                    foto_ginasioAtual = string.Empty;
                    if (!Convert.IsDBNull(reader["foto_ginasio"]))
                    {
                        foto_ginasioAtual = reader.GetString(3);
                    }
                    else
                    {
                        foto_ginasioAtual = null;
                    }

                    estadoAtual = reader.GetString(4);
                    lotacaoAtual = reader.GetInt32(5);
                    lotacaoMaxAtual = reader.GetInt32(6);

                    reader.Close();
                }
            }

            using (SqlCommand myCommand = new SqlCommand(queryPatch, databaseConnection))
            {
                myCommand.Parameters.AddWithValue("id_ginasio", targetID);
                myCommand.Parameters.AddWithValue("instituicao", !string.IsNullOrEmpty(instituicao) ? instituicao : instituicaoAtual);
                myCommand.Parameters.AddWithValue("contacto", contacto == null ? contacto : contactoAtual);

                if (!string.IsNullOrEmpty(foto) && !string.IsNullOrEmpty(foto_ginasioAtual))
                {
                    myCommand.Parameters.AddWithValue("foto_ginasio", !string.IsNullOrEmpty(foto) ? foto : foto_ginasioAtual);
                }
                else
                {
                    myCommand.Parameters.AddWithValue("foto_ginasio", DBNull.Value);
                }

                myCommand.Parameters.AddWithValue("estado", !string.IsNullOrEmpty(estado) ? estado : estadoAtual);
                myCommand.Parameters.AddWithValue("lotacao", lotacao == null ? lotacao : lotacaoAtual);
                myCommand.Parameters.AddWithValue("lotacaoMax", lotacaoMax == null ? lotacaoMax : lotacaoMaxAtual);

                dataReader = myCommand.ExecuteReader();

                dataReader.Close();
                databaseConnection.Close();
            }
        }

        return true;
    }
    catch (SqlException ex)
    {
        Console.WriteLine("Erro na conexao com a base de dados: " + ex.Message);
        return false;
    }
    catch (InvalidCastException ex)
    {
        Console.WriteLine("Erro na conversao de dados: " + ex.Message);
        return false;
    }
    catch (FormatException ex)
    {
        Console.WriteLine("Erro de tipo de dados: " + ex.Message);
        return false;
    }
    catch (ArgumentNullException ex)
    {
        Console.WriteLine("Erro de parametro inserido nulo: " + ex.Message);
        return false;
    }
    catch (ArgumentException ex)
    {
        Console.WriteLine("Valores em falta para ser inseridos: " + ex.Message);
        return false;
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
        return false;
    }
}
```

Figura 42 - WebMethod PatchGinasioService()

WebService em funcionamento:

PatchGinasioService

Testar

Para testar a operação utilizando o protocolo HTTP POST, clique no botão 'Invocar'.

Parâmetro	Valor
targetID:	<input type="text" value="3"/>
instituicao:	<input type="text" value="IPCA"/>
contacto:	<input type="text" value="253802190"/>
foto:	<input type="text" value="ipca.png"/>
estado:	<input type="text" value="Ativo"/>
lotacao:	<input type="text" value="30"/>
lotacaoMax:	<input type="text" value="50"/>
<input type="button" value="Invocar"/>	

SOAP 1.1

Segue-se um exemplo de pedido e resposta SOAP 1.1. É necessário substituir os marcadores de posição mostrados por valores reais.

```
POST /Services/IPCAgymWS.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "www.ipca.pt/PatchGinasioService"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <PatchGinasioService xmlns="www.ipca.pt">
      <targetID>int</targetID>
      <instituicao>string</instituicao>
      <contacto>int</contacto>
      <foto>string</foto>
      <estado>string</estado>
      <lotacao>int</lotacao>
      <lotacaoMax>int</lotacaoMax>
    </PatchGinasioService>
  </soap:Body>
</soap:Envelope>

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <PatchGinasioServiceResponse xmlns="www.ipca.pt">
      <PatchGinasioServiceResult>boolean</PatchGinasioServiceResult>
    </PatchGinasioServiceResponse>
  </soap:Body>
</soap:Envelope>
```

Figura 43 - Input de dados para PatchGinasioService()

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<boolean xmlns="www.ipca.pt">true</boolean>
```

Figura 44 - XML Result de PatchGinasioService()

	id_ginasio	instituicao	contacto	foto_ginasio	estado	lotacao	lotacaoMax
1	2	Boas	211345667	NULL	Ativo	32	55
2	3	IPCA	253802190	ipca.png	Ativo	30	50

Figura 45 - Resultado do PatchGinasioService() na Base de Dados (SQLServer)

7.4. WebMethod DeleteClassificacaoService() [DELETE]

Neste WebMethod é feito um request para fazer a remoção de um comentário.

Segue-se agora o código desenvolvido para efetuar a remoção de um comentário:

```
[WebMethod]
0 references
public bool DeleteClassificacaoService(int ID_Classificação)
{
    string query = @"
        delete from dbo.Classificacao
        where id_avaliacao = @id_avaliacao";
    string connectionString = "data source=DESKTOP-8AIBMTC;initial catalog=ipcagym;trusted_connection=true";

    try
    {
        using (SqlConnection databaseConnection = new SqlConnection(connectionString))
        {
            databaseConnection.Open();
            using (SqlCommand myCommand = new SqlCommand(query, databaseConnection))
            {
                myCommand.Parameters.AddWithValue("id_avaliacao", ID_Classificação);
                int rowsAffected = myCommand.ExecuteNonQuery();

                if (rowsAffected == 0) return false;

                databaseConnection.Close();
            }
        }

        return true;
    }
    catch (SqlException ex)
    {
        Console.WriteLine("Erro/Warning no SQLServer: " + ex.Message);
        return false;
    }
    catch (ArgumentNullException ex)
    {
        Console.WriteLine("Parâmetro inserido é nulo: " + ex.Message);
        return false;
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
        return false;
    }
}
```

Figura 46 - WebMethod DeleteClassificacaoService()

WebService em funcionamento:

	id_avaliacao	id_ginasio	id_cliente	avaliacao	comentario	data_avaliacao
1	1	2	1	5	Perfeito	2022-12-16 15:37:35.000
2	2	2	2	4	É bom	2022-12-16 17:37:35.000
3	3	2	3	3	Razoavel, melhorava o atendimento	2022-12-16 20:37:35.000
4	1010	2	1	5	Está ótimo	2022-11-23 12:56:49.000

Figura 47 - Base de dados antes da execução do Webservice

IPCAGym

Clique [aqui](#) para obter uma lista completa de operações.

DeleteClassificacaoService

Testar

Para testar a operação utilizando o protocolo HTTP POST, clique no botão 'Invocar'.

Parâmetro	Valor
ID_Classificação:	<input type="text" value="1010"/>
<input type="button" value="Invocar"/>	

Figura 48 - Input de dados para DeleteClassificacaoService()

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<boolean xmlns="www.ipca.pt">true</boolean>
```

Figura 49 - XML Result de DeleteClassificacaoService()

	id_avaliacao	id_ginasio	id_cliente	avaliacao	comentario	data_avaliacao
1	1	2	1	5	Perfeito	2022-12-16 15:37:35.000
2	2	2	2	4	É bom	2022-12-16 17:37:35.000
3	3	2	3	3	Razoavel, melhorava o atendimento	2022-12-16 20:37:35.000

Figura 50 - Base de Dados após a execução do DeleteClassificacaoService()

8. Testes Unitários

Foi pedido no enunciado deste trabalho prático para especificar um conjunto de testes para a API desenvolvida, de forma a obedecer a esse ponto proposto foi desenvolvido um conjunto de testes unitários, neste caso o grupo optou por fazer XUnit Tests.

A abordagem no desenvolvimento de todos os testes foi semelhante entre eles, focando no tipo de dados que os requests retornavam:

- No caso de retornar um JsonResult então espera-se um Code 200 e uma resposta não nula
- No caso de retornar um StatusCodeResult (caso o request não tenha sucesso nos dados que entrega) então espera-se um Code 204 e uma resposta não nula

Segue-se a execução dos testes da solução do projeto bem como as respectivas funções de teste:

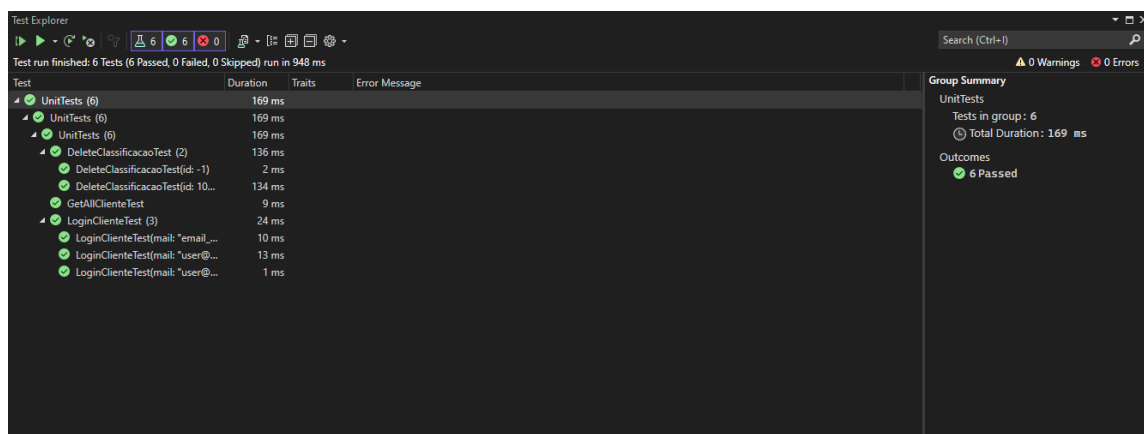


Figura 51 - Execução dos Testes Unitários

Para fazer a criação de Controladores da API para montar os dados e enviá-los para os respectivos requests, foi necessário fazer o carregamento do appsettings.json da API para ser utilizado como parâmetro do construtor de cada Controller:

```
/// <summary>
/// Configuração utilizada para a criação de controladores, para que seja possível fazer as function calls
/// </summary>
/// <returns></returns>
3 references | 6/6 passing
public static IConfiguration InitConfiguration()
{
    var config = new ConfigurationBuilder().AddJsonFile("C:\\TrabalhosPraticos\\Projeto_Aplicado\\ipca_gym" +
        "\\Backend_IPCA_Gym\\Backend_IPCA_Gym\\appsettings.json").AddEnvironmentVariables().Build();

    return config;
}
```

Figura 52 - Carregamento de informação do appsettings.json para uma configuração

8.1. Teste GetAllClienteTest

Teste para o request de obter todos os clientes da base de dados

```
/// <summary>
/// Teste no request de obter todos os clientes da Base de Dados
/// Esperado:
///     - No caso de retornar um JsonResult: Não ser nulo e ter code 200 ou 204
///     - No caso de retornar um StatusCodeResult: Deve ser code 204
/// </summary>
[Fact]
0 | 0 references
public async void GetAllClienteTest()
{
    var config = InitConfiguration();
    var clienteController = new ClienteController(config);

    IActionResult requestResult = await clienteController.GetAll();

    requestResult.Should().NotBeNull();

    var statusCodeResult = requestResult as StatusCodeResult;
    if (statusCodeResult != null)
    {
        statusCodeResult.Should().NotBeNull();
        statusCodeResult.StatusCode.Should().Be(204);
    }

    var jsonResponse = requestResult as JsonResult;
    if (jsonResponse != null)
    {
        var response = jsonResponse.Value as Response;
        if (response != null)
        {
            response.Should().NotBeNull();
            response.StatusCode.Should().Be(StatusCode.SUCCESS);
        }
    }
}
```

Figura 53 - Método teste GetAllClientes()

8.2. Teste LoginClienteTest

Teste para a funcionalidade da API de fazer Login numa conta de um cliente. Para este caso foi utilizado 3 conjuntos de dados distintos, de forma a testar todas as situações possíveis.

```
/// <summary>
/// Teste no request de fazer login
/// Esperado:
///     - No caso de retornar um JsonResult: Não ser nulo e ter code 200 ou 204
///     - No caso de retornar um StatusCodeResult: Deve ser code 204
/// Dados introduzidos:
///     - Primeiro InlineData com email errado
///     - Segundo InlineData com dados corretos
///     - Terceiro InlineData com password errada
/// </summary>
[Theory]
[InlineData("email_incorreto", "dm")]
[InlineData("user@gmail.com", "password")]
[InlineData("user@gmail.com", "pass_incorreta")]
| 0 references
public async void LoginClienteTest(string mail, string password)
{
    var config = InitConfiguration();
    var clienteController = new ClienteController(config);

    LoginCliente modelTest = new LoginCliente();

    modelTest.mail = mail;
    modelTest.password = password;

    IActionResult requestResult = await clienteController.Login(modelTest);

    requestResult.Should().NotBeNull();

    var statusCodeResult = requestResult as StatusCodeResult;
    if (statusCodeResult != null)
    {
        statusCodeResult.Should().NotBeNull();
        statusCodeResult.StatusCode.Should().Be(204);
    }

    var jsonResponse = requestResult as JsonResult;
    if (jsonResponse != null)
    {
        var response = jsonResponse.Value as Response;
        if (response != null)
        {
            response.Should().NotBeNull();
            response.StatusCode.Should().Be(StatusCode.SUCCESS);
        }
    }
}
```

Figura 54 - Método Teste LoginCliente()

8.3. Teste DeleteClassificaçãoTest

Teste para a funcionalidade da API de fazer remover uma classificação de um ginásio da base de dados.

Para este caso foi utilizado 2 conjuntos de dados distintos, de forma a testar todas as situações possíveis.

```
/// <summary>
/// Teste no request de fazer login
/// Esperado:
///     - No caso de retornar um JsonResult: Não ser nulo e ter code 200 ou 204
///     - No caso de retornar um StatusCodeResult: Deve ser code 204
/// Dados introduzidos:
///     - 1011 (ID anteriormente existente na Base de Dados)
///     - -1 (ID inexistente)
/// </summary>
[Theory]
[InlineData(1011)]
[InlineData(-1)]
0 references
public async void DeleteClassificacaoTest(int id)
{
    var config = InitConfiguration();
    var classificacaoController = new ClassificacaoController(config);

    IActionResult requestResult = await classificacaoController.Delete(id);

    requestResult.Should().NotBeNull();

    var statusCodeResult = requestResult as StatusCodeResult;
    if (statusCodeResult != null)
    {
        statusCodeResult.Should().NotBeNull();
        statusCodeResult.StatusCode.Should().Be(204);
    }

    var jsonResult = requestResult as JsonResult;
    if (jsonResult != null)
    {
        var response = jsonResult.Value as Response;
        if (response != null)
        {
            response.Should().NotBeNull();
            response.StatusCode.Should().Be(StatusCode.SUCCESS);
        }
    }
}
```

Figura 55 - Método Teste DeleteClassificacao()

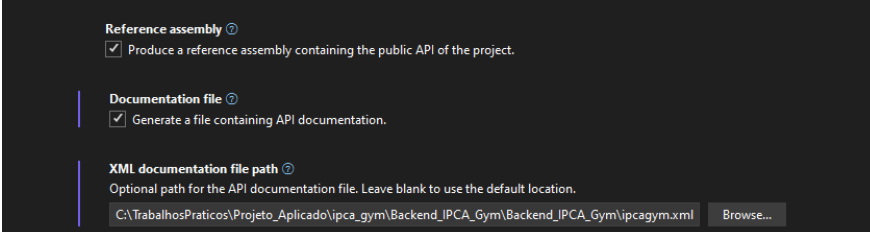
9. Documentação OpenAPI (SwaggerUI)

De forma a fazer a documentação correta da API, foi utilizado o SwaggerUI, ferramenta que foi sugerida pelo professor.

Começou-se por fazer a implementação do gerador do Swagger no Program.cs (que funciona como Startup da API):

```
c.SwaggerDoc("Alpha", new OpenApiInfo
{
    Title = "IPCAGym",
    Version = "Alpha",
    Description = "Web API para o aplicativo IPCAGym"
});
```

Foi também exportada toda a documentação para um ficheiro .xml externo que se encontra no seguinte diretório:



Reference assembly ①
☒ Produce a reference assembly containing the public API of the project.

Documentation file ②
☒ Generate a file containing API documentation.

XML documentation file path ③
Optional path for the API documentation file. Leave blank to use the default location.
C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Backend_IPCA_Gym\Backend_IPCA_Gym\ipcagym.xml Browse...

A forma que foi usada para a documentação de métodos e classes foi a seguinte:

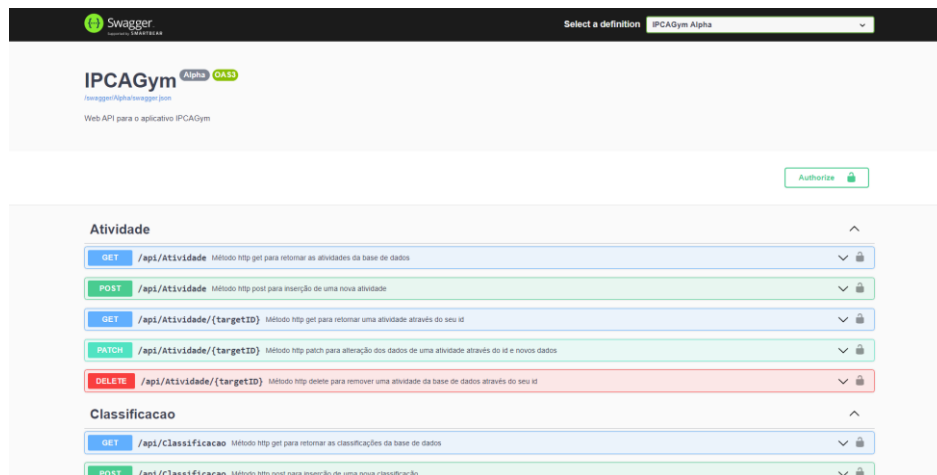
```
/// <summary>
/// Método http post para inserção de um novo ginásio
/// </summary>
/// <param name="newGinasio">Dados do novo ginásio a ser inserido</param>
/// <returns>Resposta do request que contém a sua mensagem e seu código em formato json</returns>
[HttpPost, Authorize(Roles = "Admin")]
public async Task<IActionResult> Post([FromBody] Ginasio newGinasio)
{
    string sqlDataSource = _configuration.GetConnectionString("DatabaseLink");
    Response response = await GinasioLogic.PostLogic(sqlDataSource, newGinasio);

    if (response.StatusCode != LayerBLL.Utills.StatusCodes.SUCCESS) return StatusCode((int)response.StatusCode);

    return new JsonResult(response);
}
```

```
namespace LayerBOL.Models
{
    19 references
    public class Ginasio
    {
        /// <summary>
        /// Id do ginásio
        /// </summary>
        /// <example>1</example>
        5 references
        public int id_ginasio { get; set; }
        /// <summary>
        /// Nome da instituição/estabelecimento
        /// </summary>
        /// <example>UMinho</example>
        6 references
        public string instituicao { get; set; }
        /// <summary>
        /// Estado do ginásio (Ativo ou Inativo)
        /// </summary>
        /// <example>Ativo</example>
        6 references
        public string estado { get; set; }
        /// <summary>
        /// Foto do ginásio que poderá ser nula
        /// </summary>
        /// <example>C:\OneDrive\ginasio.png</example>
        11 references
        public string? foto_ginasio { get; set; }
        /// <summary>
        /// Contacto do ginásio
        /// </summary>
        /// <example>911922933</example>
        10 references
        public int contacto { get; set; }
    }
}
```

Sendo este o resultado no Swagger:



10. Conclusão

Com este trabalho prático o grupo ficou apto de implementar uma API segura, organizada e com testes unitários em C#.

O grupo não tinha a experiência de fazer requests com a strings dos pedidos em SQL e divisão por camadas, que foi algo que cada membro aprendeu e conseguiu implementar nesta entrega.

A organização por camadas foi também um ponto de melhoria de cada um e será uma técnica que será reutilizada em futuros projetos.

11. Bibliografia

Repositório GitHub

https://github.com/Presentation12/Ipca_Gym

Figma

<https://www.figma.com/file/Q4tM34gl91b9fhrGvUeXRrs/MileriuPT's-teamlibrary?node-id=0%3A1&t=p9cWit1VJHUNwf2m-1>

Material fornecido pelo docente:

<https://elearning2.ipca.pt/2223/course/view.php?id=10611>