**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](#_Toc123809239)

[Índice de figuras 6](#_Toc123809240)

[1. Introdução 7](#_Toc123809241)

[1.1. Contextualização 7](#_Toc123809242)

[1.2. Motivação e Objetivos 7](#_Toc123809243)

[1.3. Estrutura do Documento 7](#_Toc123809244)

[2. Produto 8](#_Toc123809245)

[2.1. Visão do Produto 8](#_Toc123809246)

[3. Diagramas 9](#_Toc123809247)

[3.1. Diagrama Entidade-Relação 9](#_Toc123809248)

[4. Código 11](#_Toc123809249)

[4.1. Programação por Camadas 11](#_Toc123809250)

[4.1.1. Backend\_IPCA\_Gym 11](#_Toc123809251)

[12](#_Toc123809252)

[12](#_Toc123809253)

[12](#_Toc123809254)

[12](#_Toc123809255)

[12](#_Toc123809256)

[4.1.2. LayerBLL 13](#_Toc123809257)

[14](#_Toc123809258)

[4.1.3. LayerBOL 15](#_Toc123809259)

[4.1.4. LayerDAL 16](#_Toc123809260)

[5. Dependências 20](#_Toc123809261)

[6. WebServices em SOAP 21](#_Toc123809262)

[6.1. WebMethod GetAllClientesService() [GET] 21](#_Toc123809263)

[6.2. WebMethod LoginFuncionarioService() [POST] 24](#_Toc123809264)

[6.3. WebMethod PatchGinasioService() [PATCH] 27](#_Toc123809265)

[6.4. WebMethod DeleteClassificacaoService() [DELETE] 29](#_Toc123809266)

[7. Testes Unitários 31](#_Toc123809267)

[7.1. Teste GetAllClienteTest 32](#_Toc123809268)

[7.2. Teste LoginClienteTest 33](#_Toc123809269)

[7.3. Teste DeleteClassificaçãoTest 34](#_Toc123809270)

[8. Documentação OpenAPI (SwaggerUI) 35](#_Toc123809271)

[9. Conclusão 36](#_Toc123809272)

[10. Bibliografia 36](#_Toc123809273)

# Índice de figuras

[Figura 1 - Diagrama de Entidade-Relação 9](#_Toc123809274)

[Figura 2 - Camadas 11](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809275)

[Figura 3 - Lista dos controladores 11](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809276)

[Figura 4 - Main 11](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809277)

[Figura 5 - Get dos clientes todos 12](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809278)

[Figura 6 - Get de um cliente pelo ID 12](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809279)

[Figura 7 - Adicionar Cliente 12](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809280)

[Figura 8 - Remover Cliente 12](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809281)

[Figura 9 - Alteração dados de um cliente 12](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809282)

[Figura 10 - Camada BLL 13](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809283)

[Figura 11 - Exemplo Utils 13](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809284)

[Figura 12 - Logics Amostrar cliente por ID 14](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809285)

[Figura 13 - Logics lista de Clientes 14](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809286)

[Figura 14 - Logics Adicionar Cliente 14](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809287)

[Figura 15 - Logics Remover Cliente 14](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809288)

[Figura 16 - Logics Alterar dados do Cliente 14](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809289)

[Figura 17 - Propriedades Ginásio DB 15](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809290)

[Figura 18 - Propriedades Ginásio 15](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809291)

[Figura 19 - Ginasio Service 16](#_Toc123809292)

[Figura 20 - Ginasio Service - GetAll 17](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809293)

[Figura 21 - Ginasio Service – By ID 17](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809294)

[Figura 23 - Ginasio Service - Post 18](#_Toc123809295)

[Figura 22 - Ginasio Service - Patch 18](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809296)

[Figura 24 - Ginasio Service - Delete 19](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809297)

[Figura 25 - Dependêcia API Layer 20](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809298)

[Figura 26 - Dependêcia BLL 20](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809299)

[Figura 27 - Dependêcia BOL 20](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809300)

[Figura 28 - Modelo de dados auxiliar (Cliente) 21](#_Toc123809301)

[Figura 29 - WebMethod GetAllClientesService() 22](#_Toc123809302)

[Figura 30 - XML Result de GetAllClientesService() 23](#_Toc123809303)

[Figura 31 - Modelo de dados auxiliar (LoginModel) 24](#_Toc123809304)

[Figura 32 - WebMethod LoginFuncionarioService() 25](#_Toc123809305)

[Figura 33 - Método auxiliar VerifyPasswordHash() 25](#_Toc123809306)

[Figura 34 - Input de dados para LoginFuncionarioService() 26](#_Toc123809307)

[Figura 35 - XML Result de LoginFuncionarioService() 26](#_Toc123809308)

[Figura 36 - XML Result de LoginFuncionarioService() no caso de erro 26](#_Toc123809309)

[Figura 37 - WebMethod PatchGinásioService() 27](#_Toc123809310)

[Figura 38 - Input de dados para PatchGinasioService() 28](#_Toc123809311)

[Figura 39 - XML Result de PatchGinasioService() 28](#_Toc123809312)

[Figura 40 - Resultado do PatchGinasioService() na Base de Dados (SQLServer) 28](#_Toc123809313)

[Figura 41 - WebMethod DeleteClassificacaoService() 29](#_Toc123809314)

[Figura 42 - Base de dados antes da execução do WebService 30](#_Toc123809315)

[Figura 43 - Input de dados para DeleteClassificacaoService() 30](#_Toc123809316)

[Figura 44 - XML Result de DeleteClassificacaoService() 30](#_Toc123809317)

[Figura 45 - Base de Dados após a execução do DeleteClassificacaoService() 30](#_Toc123809318)

[Figura 46 - Execução dos Testes Unitários 31](file:///C:\TrabalhosPraticos\Projeto_Aplicado\ipca_gym\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123809319)

[Figura 47 - Carregamento de informação do appsetings.json para uma configuração 31](#_Toc123809320)

[Figura 48 - Método teste GetAllClientes() 32](#_Toc123809321)

[Figura 49 - Método Teste LoginCliente() 33](#_Toc123809322)

[Figura 50 - Método Teste DeleteClassificacao() 34](#_Toc123809323)

# Introdução

## 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.

## 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;

## 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.

# 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 IPCA 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.

# Diagramas

## Diagrama Entidade-Relação

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

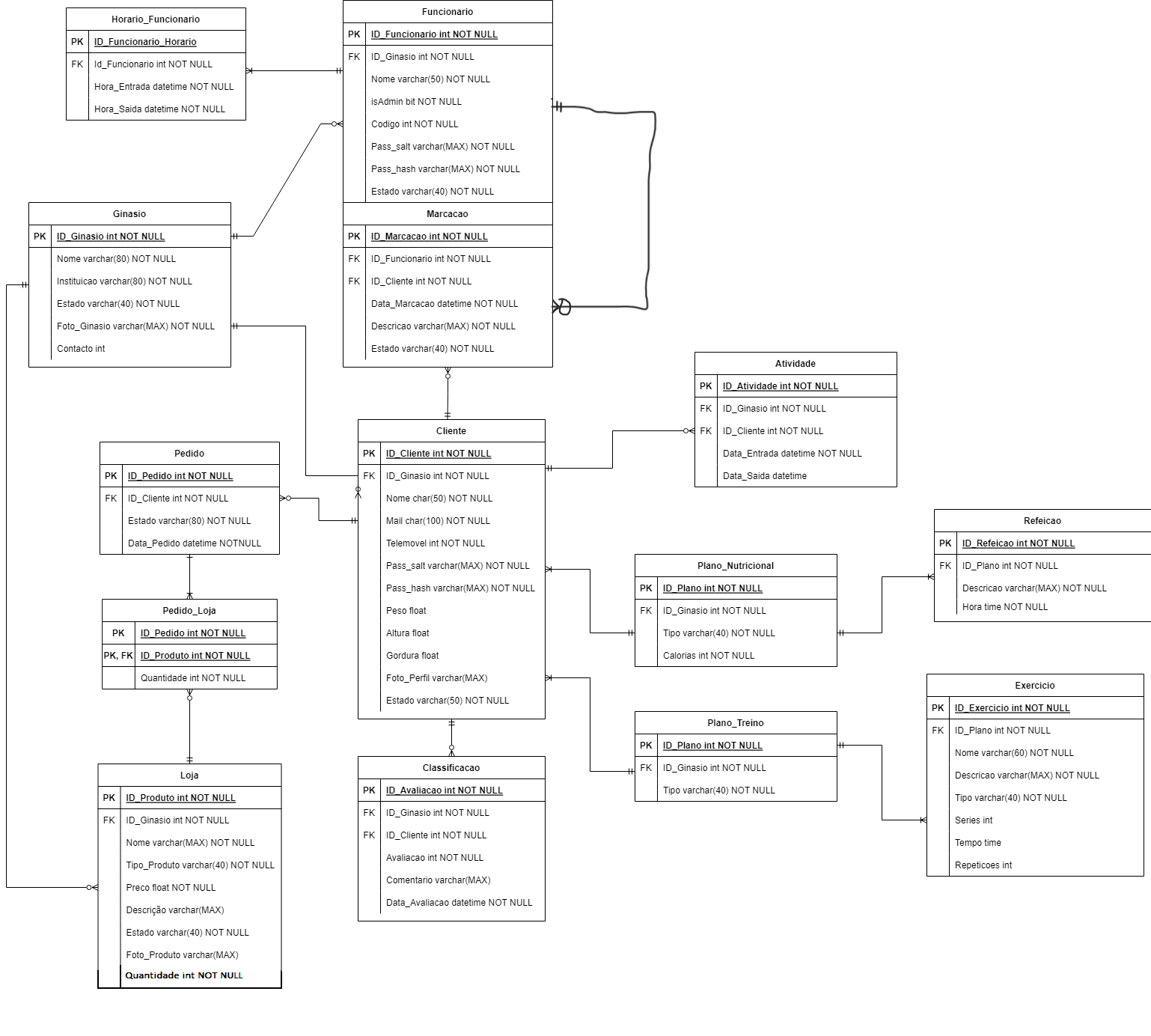


Figura 1 - Diagrama de Entidade-Relação

Como entidades principais este diagrama possuí:

* **­Cliente** – dados de um cliente que está a utilizar a aplicação;
* **Funcionário** – dados de um funcionário do ginásio em causa, possuí 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 Arduíno)

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.

# Código

## Programação por Camadas

Neste projeto, como forma de organizar o nosso código, decidimos programar em 4 camadas, isto para que 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.

Uma imagem com texto

Descrição gerada automaticamente

1ª Camada - Backend API

2ª Camada - BLL

3ª Camada - BOL

4ª Camada - DAL

Figura - Camadas

Na camada “Backend\_IPCA\_Gym” estão os controllers respetivos para todas as entidades do nosso sistema. Esta camada utiliza funções logic, sendo estas funções provindas 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 nosso sistema.

### Uma imagem com texto Descrição gerada automaticamenteBackend\_IPCA\_Gym

Figura - Lista dos controladores

Uma imagem com texto

Descrição gerada automaticamente

Figura - 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,hettpdelete…).

* Listagem de todos os clientes

### Uma imagem com texto Descrição gerada automaticamente

### 

Figura - Get dos clientes todos

* Listagem de um cliente através do seu ID

Uma imagem com texto

Descrição gerada automaticamente

Figura - Get de um cliente pelo ID

* Criação de um novo cliente

### Uma imagem com texto Descrição gerada automaticamente

### 

Figura - Adicionar Cliente

* Remoção de um cliente

Uma imagem com texto

Descrição gerada automaticamente

Figura - Remover Cliente

* Alteração dos dados relativos a um cliente

### Uma imagem com texto Descrição gerada automaticamente

Figura - Alteração dados de um cliente

### 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.

* Uma imagem com texto, eletrónica

  Descrição gerada automaticamenteNa parte *Logics* temos as funções logic (funções referidas anteriormente chamadas na camada Backend.

Figura - Camada BLL

* Uma imagem com texto

  Descrição gerada automaticamenteNa parte *Utils* temos o código que nos dá informação request está ou não a funcionar corretamente.

Figura - 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.

### Uma imagem com texto, captura de ecrã, ecrã, prateado Descrição gerada automaticamenteUma imagem com texto Descrição gerada automaticamente

Figura - Logics Amostrar cliente por ID

Figura - Logics lista de Clientes

Uma imagem com texto

Descrição gerada automaticamenteUma imagem com texto

Descrição gerada automaticamenteUma imagem com texto

Descrição gerada automaticamente

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

Figura - Logics Adicionar Cliente

Figura - Logics Remover Cliente

Figura - Logics Alterar dados do Cliente

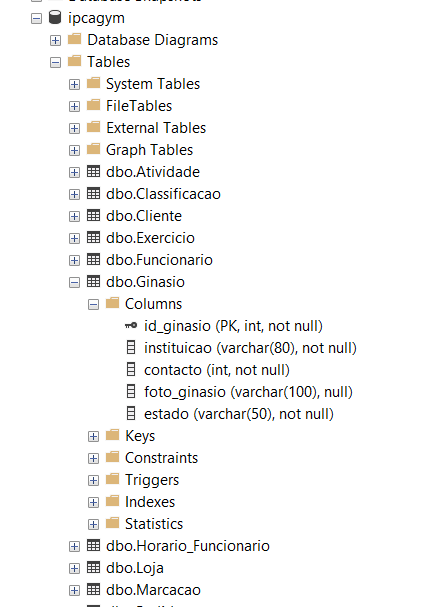
### LayerBOL

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.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Propriedades Ginásio DB

Figura - Propriedades Ginásio

### 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.

Uma imagem com texto

Descrição gerada automaticamente

Figura - 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.

Uma imagem com texto

Descrição gerada automaticamenteComeçando 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.

Figura - Ginasio Service - GetAll

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

Apesar da forma como as querys estão implementadas serem praticamente iguais, vemos que na primeira função temos a necessidade de colocar especificamente que campos desejamos que aparecem, 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.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Ginasio Service – By ID

Uma imagem com texto

Descrição gerada automaticamenteQuanto 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 especifica. Após definidos os campos basta apenas fazer a inserção dos campos através de comandos predefinidos no C#.

Figura - Ginasio Service - Post

Figura - 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.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Ginasio Service - Delete

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

# JWT Tokens

## 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 possuí as permissões necessárias para obter os dados pedidos.

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

Uma imagem com texto

Descrição gerada automaticamente

Figura - AddAuthentication() no Program.cs



Figura - 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

Uma imagem com texto

Descrição gerada automaticamente

Figura - Criação de uma token para um Funcionário

## Utilização de Authorize nos requests

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

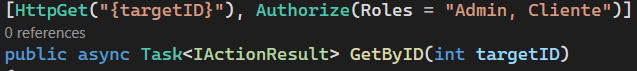


Figura - Request com autorizações

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

Uma imagem com texto

Descrição gerada automaticamente

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

# 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)

Uma imagem com texto

Descrição gerada automaticamente

Figura - Dependêcia API Layer

Uma imagem com texto

Descrição gerada automaticamente

Figura - Dependêcia BLL

Uma imagem com texto

Descrição gerada automaticamente

Figura - Dependêcia BOL

# 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

## WebMethod GetAllClientesService() [GET]

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

Uma imagem com texto

Descrição gerada automaticamente

Figura - Modelo de dados auxiliar (Cliente)

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



Figura - WebMethod GetAllClientesService()

WebService em funcionamento:

Uma imagem com texto

Descrição gerada automaticamente

Figura - XML Result de GetAllClientesService()

## 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:

Uma imagem com texto

Descrição gerada automaticamente

Figura - Modelo de dados auxiliar (LoginModel)

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



Figura - WebMethod LoginFuncionarioService()

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

Uma imagem com texto, captura de ecrã, ecrã, prateado

Descrição gerada automaticamente

Figura - Método auxiliar VerifyPasswordHash()

WebService em funcionamento:

Uma imagem com texto

Descrição gerada automaticamente

Figura - Input de dados para LoginFuncionarioService()

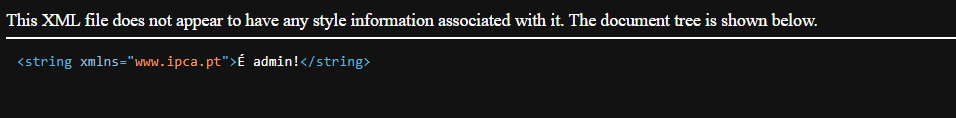


Figura - XML Result de LoginFuncionarioService()

Uma imagem com texto

Descrição gerada automaticamente

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

## 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:



Figura - WebMethod PatchGinásioService()

WebService em funcionamento:

Uma imagem com texto

Descrição gerada automaticamente

Figura - Input de dados para PatchGinasioService()

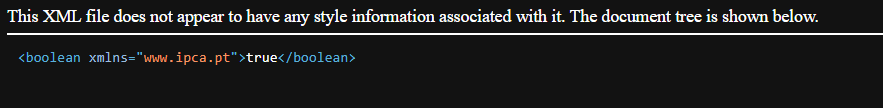


Figura - XML Result de PatchGinasioService()

Uma imagem com mesa

Descrição gerada automaticamente

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

## 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:

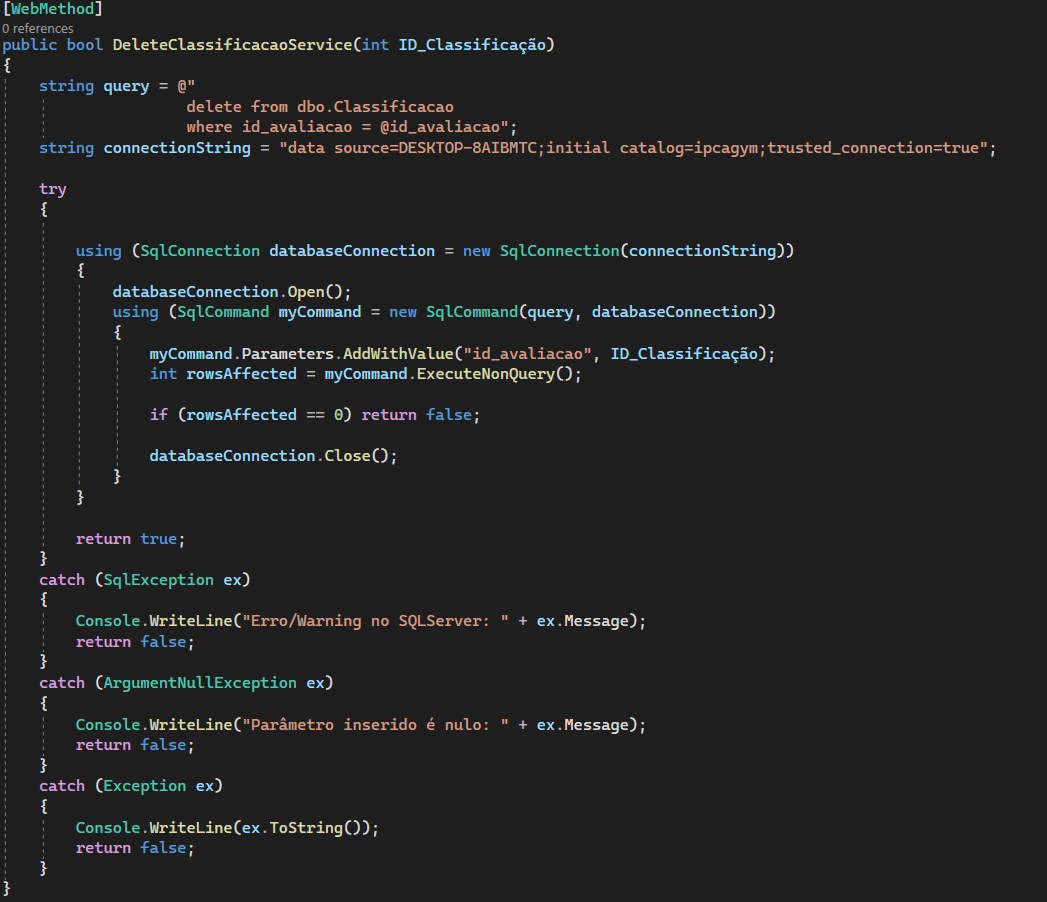


Figura - WebMethod DeleteClassificacaoService()

WebService em funcionamento:

Uma imagem com mesa

Descrição gerada automaticamente

Figura - Base de dados antes da execução do WebService

Uma imagem com texto

Descrição gerada automaticamente

Figura - Input de dados para DeleteClassificacaoService()

Uma imagem com texto

Descrição gerada automaticamente

Figura - XML Result de DeleteClassificacaoService()

Uma imagem com mesa

Descrição gerada automaticamente

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

# 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 respetivas funções de teste:

Uma imagem com texto, captura de ecrã, ecrã

Descrição gerada automaticamente

Figura - Execução dos Testes Unitários

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

Uma imagem com texto

Descrição gerada automaticamente

Figura - Carregamento de informação do appsetings.json para uma configuração

## Teste GetAllClienteTest

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

Uma imagem com texto

Descrição gerada automaticamente

Figura - Método teste GetAllClientes()

## 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.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Método Teste LoginCliente()

## 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.

Uma imagem com texto, captura de ecrã, telefone, telemóvel

Descrição gerada automaticamente

Figura - Método Teste DeleteClassificacao()

# 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):

Uma imagem com texto

Descrição gerada automaticamente

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

Uma imagem com texto

Descrição gerada automaticamente

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

Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente

Sendo este o resultado no Swagger:

Uma imagem com texto

Descrição gerada automaticamente

# 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.

# Bibliografia

**Repositório GitHub**

<https://github.com/Presentation12/Ipca_Gym>

**Figma**

<https://www.figma.com/file/Q4tM34gl91b9fhrGvUeXRs/MileriuPT's-teamlibrary?node-id=0%3A1&t=p9cWit1VJHUNwf2m-1>

**Material fornecido pelo docente:**

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

**Vídeo tutorial utilizado para conectar PostGreSQL a .NET:**

<https://www.youtube.com/watch?v=U_v1dSglNjE>