**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, janeiro de 2023

IPCA GYM

# Resumo

Este projeto prático visou melhorar a performance de trabalho em equipa e explorar as necessidades de um smart campus no IPCA através da criação de um sistema para um ginásio e uma aplicação para os seus utilizadores.

Como parte da unidade curricular de **Integração de sistemas de informação**, o projeto incluiu a construção e interação com a API que sustentou o nosso sistema. Além de demonstrar técnicas e conceitos abordados tanto inter quanto extracurricular, o projeto teve como objetivo ajudar os ginásios a gerenciar o seu negócio de forma mais eficiente e proporcionar uma experiência excecional aos seus clientes.

Conteúdo

[Resumo 4](#_Toc123915277)

[Índice de figuras 6](#_Toc123915278)

[1. Introdução 7](#_Toc123915279)

[1.1. Contextualização 7](#_Toc123915280)

[1.2. Motivação e Objetivos 7](#_Toc123915281)

[1.2.1. Motivações: 7](#_Toc123915282)

[1.2.2. Objetivos pessoais: 7](#_Toc123915283)

[1.2.3. Objetivos do projeto: 7](#_Toc123915284)

[1.3. Estrutura do Documento 7](#_Toc123915285)

[2. Produto 8](#_Toc123915286)

[2.1. Visão do Produto 8](#_Toc123915287)

[3. Diagramas 9](#_Toc123915288)

[3.1. Diagrama Entidade-Relação 9](#_Toc123915289)

[4. Código 11](#_Toc123915290)

[4.1. Programação por Camadas 11](#_Toc123915291)

[4.1.1. Camada Backend\_IPCA\_Gym 12](#_Toc123915292)

[4.1.2. Camada BLL 14](#_Toc123915298)

[4.1.3. Camada BOL 16](#_Toc123915300)

[4.1.4. Camada DAL 17](#_Toc123915301)

[5. JWT Tokens 22](#_Toc123915302)

[5.1. Implementação. 22](#_Toc123915303)

[5.2. Utilização de Authorize nos requests 23](#_Toc123915304)

[6. Dependências das Camadas 24](#_Toc123915305)

[7. WebServices em SOAP 25](#_Toc123915306)

[7.1. WebMethod GetAllClientesService() [GET] 25](#_Toc123915307)

[7.2. WebMethod LoginFuncionarioService() [POST] 28](#_Toc123915308)

[7.3. WebMethod PatchGinasioService() [PATCH] 31](#_Toc123915309)

[7.4. WebMethod DeleteClassificacaoService() [DELETE] 33](#_Toc123915310)

[8. Testes Unitários 35](#_Toc123915311)

[8.1. Teste GetAllClienteTest 36](#_Toc123915312)

[8.2. Teste LoginClienteTest 37](#_Toc123915313)

[8.3. Teste DeleteClassificaçãoTest 38](#_Toc123915314)

[9. Documentação OpenAPI (SwaggerUI) 39](#_Toc123915315)

[10. Packages (Nuggets) 41](#_Toc123915316)

[11. Conclusão 42](#_Toc123915317)

[12. Bibliografia 42](#_Toc123915318)

# Índice de figuras

[Figura 1 - Diagrama de Entidade-Relação 9](#_Toc123915355)

[Figura 2 - Camadas 11](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915356)

[Figura 3 - Lista dos controladores 12](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915357)

[Figura 4 - Main 12](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915358)

[Figura 5 - Get dos clientes todos 13](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915359)

[Figura 6 - Get de um cliente pelo ID 13](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915360)

[Figura 7 - Adicionar Cliente 13](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915361)

[Figura 8 - Remover Cliente 13](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915362)

[Figura 9 - Alteração dados de um cliente 13](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915363)

[Figura 10 - Camada BLL 14](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915364)

[Figura 11 - Exemplo Utils 14](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915365)

[Figura 12 - Logics Amostrar cliente por ID 15](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915366)

[Figura 13 - Logics lista de Clientes 15](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915367)

[Figura 14 - Logics Adicionar Cliente 15](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915368)

[Figura 15 - Logics Remover Cliente 15](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915369)

[Figura 16 - Logics Alterar dados do Cliente 15](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915370)

[Figura 17 - Propriedades Ginásio DB 16](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915371)

[Figura 18 - Propriedades Ginásio 16](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915372)

[Figura 19 - Ginasio Service 17](#_Toc123915373)

[Figura 20 – Horario\_Funcionario Service - GetAll 18](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915374)

[Figura 21 - Horario\_Funcionario Service – By ID 19](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915375)

[Figura 23 - Horario\_Funcionario Service – Post 20](#_Toc123915376)

[Figura 22 - Horario\_Funcionario Service - Patch 20](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915377)

[Figura 24 - Horario\_Funcionario Service - Delete 21](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915378)

[Figura 25 - AddAuthentication() no Program.cs 22](#_Toc123915379)

[Figura 26 - app function calls no Program.cs 22](#_Toc123915380)

[Figura 27 - Criação de uma token para um Funcionário 22](#_Toc123915381)

[Figura 28 - Request com autorizações 23](#_Toc123915382)

[Figura 29 - Request para obter informação de um Cliente a partir da Token de Sessão 23](#_Toc123915383)

[Figura 30 - Dependência API Layer 24](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915384)

[Figura 31 - Dependência BLL 24](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915385)

[Figura 32 - Dependência BOL 24](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915386)

[Figura 33 - Modelo de dados auxiliar (Cliente) 25](#_Toc123915387)

[Figura 34 - WebMethod GetAllClientesService() 26](#_Toc123915388)

[Figura 35 - XML Result de GetAllClientesService() 27](#_Toc123915389)

[Figura 36 - Modelo de dados auxiliar (LoginModel) 28](#_Toc123915390)

[Figura 37 - WebMethod LoginFuncionarioService() 29](#_Toc123915391)

[Figura 38 - Método auxiliar VerifyPasswordHash() 29](#_Toc123915392)

[Figura 39 - Input de dados para LoginFuncionarioService() 30](#_Toc123915393)

[Figura 40 - XML Result de LoginFuncionarioService() 30](#_Toc123915394)

[Figura 41 - XML Result de LoginFuncionarioService() no caso de erro 30](#_Toc123915395)

[Figura 42 - WebMethod PatchGinásioService() 31](#_Toc123915396)

[Figura 43 - Input de dados para PatchGinasioService() 32](#_Toc123915397)

[Figura 44 - XML Result de PatchGinasioService() 32](#_Toc123915398)

[Figura 45 - Resultado do PatchGinasioService() na Base de Dados (SQLServer) 32](#_Toc123915399)

[Figura 46 - WebMethod DeleteClassificacaoService() 33](#_Toc123915400)

[Figura 47 - Base de dados antes da execução do WebService 34](#_Toc123915401)

[Figura 48 - Input de dados para DeleteClassificacaoService() 34](#_Toc123915402)

[Figura 49 - XML Result de DeleteClassificacaoService() 34](#_Toc123915403)

[Figura 50 - Base de Dados após a execução do DeleteClassificacaoService() 34](#_Toc123915404)

[Figura 51 - Execução dos Testes Unitários 35](file:///C:\Users\joaoc\OneDrive\Ambiente%20de%20Trabalho\Ipca_Gym\Organização\Entrega%20ISI\Entrega_21140_21145_21152.docx#_Toc123915405)

[Figura 52 - Carregamento de informação do appsetings.json para uma configuração 35](#_Toc123915406)

[Figura 53 - Método teste GetAllClientes() 36](#_Toc123915407)

[Figura 54 - Método Teste LoginCliente() 37](#_Toc123915408)

[Figura 55 - Método Teste DeleteClassificacao() 38](#_Toc123915409)

[Figura 56 - Geração documentação no Swagger 39](#_Toc123915410)

[Figura 57 - Propriedades da Solução do projeto 39](#_Toc123915411)

[Figura 58 - Documentação de métodos 39](#_Toc123915412)

[Figura 59 - Documentação de modelos de dados 40](#_Toc123915413)

[Figura 60 - Resultado documentação no Swagger 40](#_Toc123915414)

# Introdução

## Contextualização

A unidade curricular insere-se na construção da arquitetura do sistema proposto no projeto inicial, implementando a API que irá executar os serviços web. A implementação da API é uma parte importante do projeto, pois ela irá fornecer a base para a comunicação entre o sistema e os utilizadores.

## Motivação e Objetivos

### Motivações:

Criar um sistema que possa ajudar a gerir o ginásio e fornecer uma experiência excecional aos seus clientes, á medida que o IPCA cresce e adquire mais instalações.

### Objetivos pessoais:

Cimentar os conhecimentos adquiridos durante o percurso académico.

### Objetivos do projeto:

Construir a arquitetura do sistema de gestão do ginásio e da aplicação para os clientes.

Desenvolver uma API que suporte serviços web, incluindo:

* Swagger para documentação da API.
* Uma base de dados para armazenar informações relevantes.
* Mecanismo de autenticação para garantir a segurança dos dados.

## Estrutura do Documento

Este documento foi estruturado de forma a ser de fácil leitura, com a inclusão de referências ao material fornecido pelo professor Óscar Ribeiro e/ou excertos de Web grafia. Além disso, o documento foi dividido em tópicos e subseções, de forma a facilitar a navegação e a associação do conteúdo com o material fornecido pelo docente. Acreditamos que essa estrutura tornará mais fácil para o leitor encontrar e compreender as informações relevantes.

# Produto

## 2.1. Visão do Produto

O IPCA GYM é uma aplicação mobile e conjunto de hardware de gestão de acessos que se enquadra no subtópico Smart Campus da Saúde, criado para acompanhar a vida saudável e atlética dos estudantes do IPCA e incentivar um estilo de vida saudável.

A aplicação permite aos estudantes acompanhar a sua atividade física e os exercícios que podem fazer durante o treino, enquanto os gestores do ginásio podem monitorizar as pessoas inscritas no ginásio através do sistema de gestão de acesso com chip/cartão eletrónico.

Com o IPCA GYM, os estudantes podem ter um acompanhamento móvel da sua atividade física e os gestores do ginásio podem maximizar a eficiência e oferecer um valor excecional aos seus clientes.

# Diagramas

## Diagrama Entidade-Relação

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

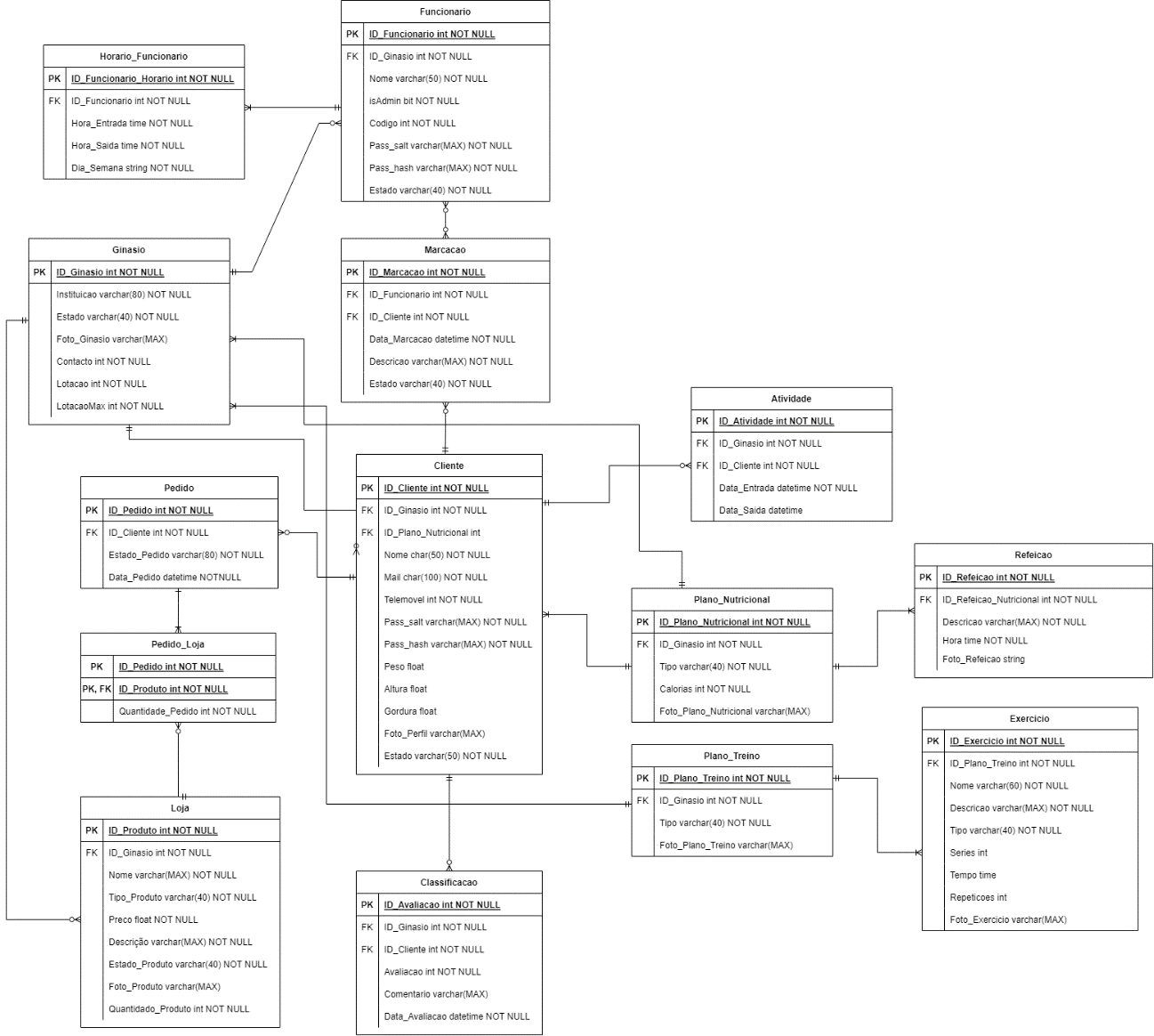


Figura 1 - Diagrama de Entidade-Relação

Este diagrama de entidade-relação possui as seguintes entidades principais:

* **Cliente**: armazena os dados de um cliente que está utilizando a aplicação.
* **Funcionário**: armazena os dados de um funcionário do ginásio, incluindo o atributo "isAdmin" para determinar se ele tem a função de gerente ou não.
* **Ginásio**: armazena os dados do ginásio em questão, de forma que o projeto possa suportar várias instituições acadêmicas no futuro.
* **Loja**: armazena os dados de todos os produtos disponíveis e indisponíveis na loja de cada ginásio.
* **Atividade**: armazena informações sobre as entradas e saídas de cada cliente no ginásio (recebe informação do Arduíno).

Para suportar outras funcionalidades, foram adicionadas as seguintes entidades:

* **Plano\_Nutricional** e **Refeição** – armazenam dados sobre diferentes refeições e seus horários, cada ginásio define os seus planos nutricionais.
* **Plano\_Treino** e **Exercício** – armazenam dados sobre diferentes exercícios e suas descrições, cada ginásio define os seus planos de treino;
* **Pedido** e **Pedido\_Loja**: armazenam informações sobre cada encomenda feita pelo usuário na loja do ginásio em que ele está inscrito.
* **Horario\_Funcionario**: regista o horário diário de cada funcionário, para verificar sua disponibilidade para as diferentes marcações.
* **Marcação**: armazena informações sobre as marcações feitas pelo cliente com o funcionário, associadas a cada ginásio.
* **Classificação**: armazena todas as avaliações feitas pelos clientes a cada ginásio.

# Código

## Programação por Camadas

Neste projeto, optamos por utilizar a arquitetura de programação em camadas para organizar o código de forma mais eficiente e segura. A utilização desta arquitetura permite uma melhor performance, facilita a deteção e correção de anomalias e permite substituir partes ou mesmo toda a camada sem comprometer o sistema inteiro. Esta abordagem de programação torna o código mais organizado e fácil de manter.

Níveis:

1ª Camada - Back-End API

2ª Camada - BLL

3ª Camada - BOL

4ª Camada - DAL

Uma imagem com texto

Descrição gerada automaticamente

Figura - Camadas

### Camada Backend\_IPCA\_Gym

A camada "Backend\_IPCA\_Gym" é responsável por gerenciar os Controllers de todas as entidades do nosso sistema. Nesta camada, são executadas as chamadas à API e utilizadas funções lógicas provenientes da camada de Acesso a Dados (DAL).

Uma imagem com texto

Descrição gerada automaticamente

Figura - Lista dos controladores

O Main do sistema também se encontra nesta camada e é invocado quando o sistema é inicializado.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Main

Cada controlador no projeto corresponde a uma parte específica do sistema e é responsável por fazer a conexão com a base de dados através de funções específicas. Antes de executar qualquer chamada, é necessário especificar o tipo de request desejado (como HTTP GET, POST, DELETE, etc.) dentro dos controladores. Isso permite que o sistema saiba como processar e lidar com as solicitações feitas pelos usuários.

* 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

### Camada BLL

"BLL" significa "Business Logic Layer" e é um termo comum no desenvolvimento de software para se referir a uma camada na arquitetura de uma aplicação. A camada BLL é responsável por implementar a lógica de negócios da aplicação, incluindo tarefas como validar a entrada do usuário, realizar cálculos e interagir com a camada de Acesso a Dados (DAL) para recuperar e armazenar dados. No C#, a camada BLL é implementada como um conjunto de classes que contêm os métodos que executam a lógica de negócios da aplicação. Esses métodos são chamados pela camada de Apresentação (como uma interface do usuário) ou por outros componentes da aplicação para realizar tarefas específicas.

Uma imagem com texto, eletrónica

Descrição gerada automaticamente

Figura - Camada BLL

* No package *Logics* temos as funções lógicas que são chamadas na camada Back-End para realizar tarefas específicas.
* No package *Utils* temos o código que nos fornece informações sobre o status da solicitação, permitindo verificar se a mesma está funcionando corretamente.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Exemplo Utils

Seguem-se exemplos, 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).

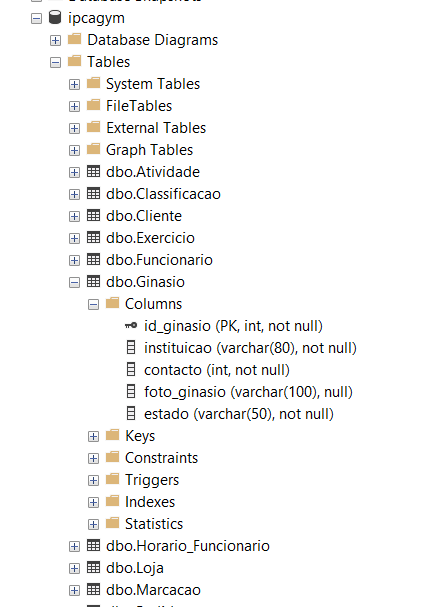
Figura - Logics Adicionar Cliente

Figura - Logics Remover Cliente

Figura - Logics Alterar dados do Cliente

### Camada BOL

A camada "BOL" (Business Object Layer) é um termo comummente usado no desenvolvimento de software para se referir a uma camada que é responsável por representar entidades de negócios e os seus relacionamentos. A camada BOL normalmente fica entre a camada de apresentação (como uma interface de usuário) e a camada de acesso a dados (que é responsável por comunicar com a base de dados ou outro armazenamento de dados). No C#, a camada de objeto de negócios foi implementada como um conjunto de classes, cada uma correspondendo a uma entidade de negócio e seus relacionamentos.

Uma imagem com texto

Descrição gerada automaticamenteEssas classes também incluem propriedades que representam os atributos das entidades de negócios e métodos que executam a lógica de negócios. Abaixo, podemos ver um exemplo da classe Ginásio, com seus atributos e métodos devidamente comentados. Ao lado da figura, podemos observar na base de dados do projeto que os campos são os mesmos da tabela Ginásio.

Figura - Propriedades Ginásio DB

Figura - Propriedades Ginásio

### Camada DAL

"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.

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

Na primeira função, utilizamos uma consulta SQL para selecionar todas as informações da tabela " Horario\_Funcionario" da nossa base de dados. Em seguida, utilizamos uma estrutura de repetição para percorrer todas as linhas retornadas pela consulta e criamos um objeto " dia" para cada linha, atribuindo os valores da linha aos atributos do objeto. Por fim, adicionamos cada objeto "dia" à nossa lista de horários e retornamos a lista.

Uma imagem com texto

Descrição gerada automaticamente

Figura – Horario\_Funcionario Service - GetAll

Na segunda função, utilizamos uma consulta SQL para selecionar todas as informações da tabela " Horario\_Funcionario" da nossa base de dados onde o ID do horário corresponde ao ID passado como parâmetro. Em seguida, criamos um objeto "targetHorarioFuncionario" e atribuímos os valores da linha retornada pela consulta aos atributos do objeto. Por fim, retornamos o objeto.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Horario\_Funcionario Service – By ID

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.

Na seguinte figura podemos ver um exemplo de como é realizado o processo de inserção (**POST**) de dados na base de dados para a entidade "Horario\_Funcionario". É necessário especificar todos os campos que serão adicionados e, em seguida, utilizar os comandos adequados para realizar a inserção. É importante notar que este processo pode variar de acordo com o tipo de base de dados que está sendo utilizada e com as especificidades de cada entidade.Uma imagem com texto

Descrição gerada automaticamente

Figura - Horario\_Funcionario Service – Post

Para uma função de alteração de dados na base de dados (Patch) foi implementada no exemplo de baixo, de forma a receber um objeto com os novos valores a serem atualizados. Essa é uma maneira comum de se fazer a atualização de dados em uma base de dados, pois permite que apenas os valores que forem modificados sejam atualizados e não todos os campos da entidade. Isso pode ser mais eficiente e evitar erros, já que não é necessário fornecer todos os valores novamente. Além disso, é possível observar que a função faz uso de comandos SQL para realizar a atualização dos dados na base de dados.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Horario\_Funcionario Service - Patch

Para remover um registo de uma entidade, estamos a utilizar apenas o seu ID para identificá-lo e, assim, realizar a operação de exclusão. A imagem abaixo ilustra este processo, onde é criada uma query com o comando DELETE para remover o registo correspondente ao ID especificado. Uma imagem com texto

Descrição gerada automaticamente

Figura - Horario\_Funcionario Service - Delete

# JWT Tokens

## Implementação.

Para garantir a autenticação de forma segura, foi implementado o uso de tokens JWT (JSON Web Token). Os tokens JWT são códigos criptografados que permitem a autenticação de usuários em sistemas. Dessa forma, é possível garantir que apenas usuários autenticados tenham acesso às informações solicitadas.

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

Uma imagem com texto

Descrição gerada automaticamente

Figura - AddAuthentication() no Program.cs



Figura 26 - app function calls no Program.cs

As roles de cada utilizador são definidas num ficheiro da camada DAL (Token.cs), garantindo assim que cada utilizador tenha as permissões adequadas para aceder aos dados solicitados (Cliente, Funcionario, Gerente e Admin).

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:

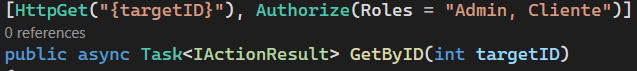


Figura - Request com autorizações

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 das Camadas

Tendo em conta que este projeto está dividido em camadas, foi necessário atribuir as dependências a cada camada da seguinte forma:

* 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 - Dependência API Layer

Uma imagem com texto

Descrição gerada automaticamente

Figura - Dependência BLL

Uma imagem com texto

Descrição gerada automaticamente

Figura - Dependência BOL

# WebServices em SOAP

Todos esses métodos foram implementados usando a tecnologia SOAP, que é um padrão para a troca de mensagens estruturadas entre sistemas distribuídos, para acessos mais seguros.

O projeto inclui este WebService com métodos GET, POST, PATCH e DELETE, dos quais foram selecionados 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]

O WebMethod para efetuar o login de funcionário retorna o tipo de Role do usuário. Ele foi escolhido para variar os objetivos de cada tipo de request. O Login não insere dados na base de dados, portanto nem um GET, pois precisa de um request body. Uma classe foi criada como modelo de dados para suportar esta funcionalidade:

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()

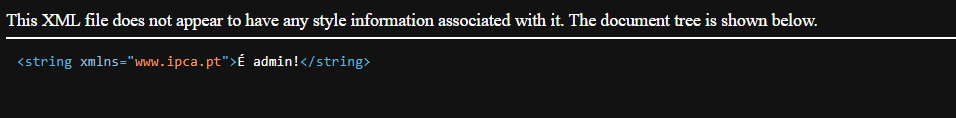


Figura - XML Result de LoginFuncionarioService()

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()

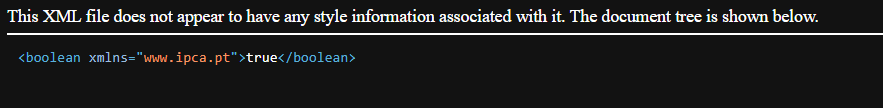


Figura - XML Result de PatchGinasioService()

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:

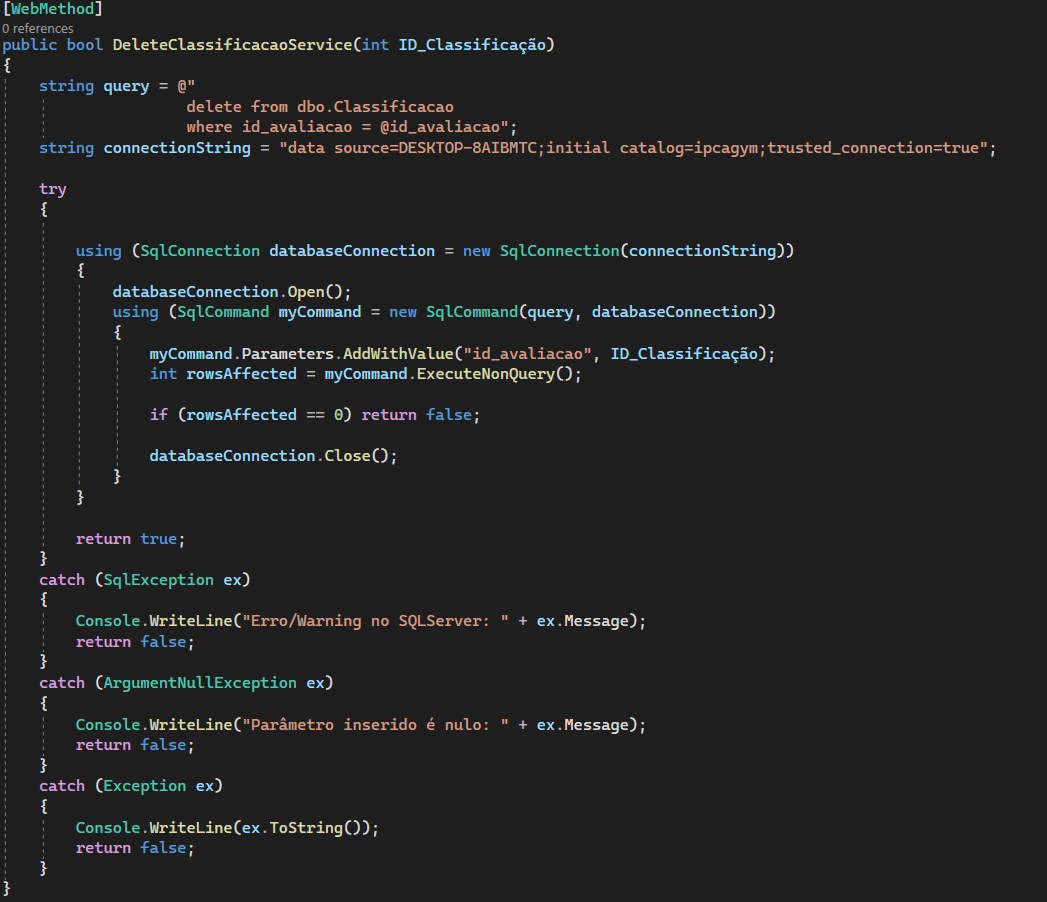


Figura - WebMethod DeleteClassificacaoService()

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

O objetivo dos testes unitários foi verificar se a API está a funcionar corretamente, validando a resposta de cada request. Usamos o XUnit para criar esses testes.

Verificamos o código de status retornado pelos requests e também se a resposta é não nula. Para requests que devem ter sucesso (retorna JsonResult), esperamos um código de status 200 e uma resposta não nula. Para requests que não devem ter sucesso (retorna StatusCodeResult), esperamos um código de status 204 e uma resposta não nula. Seguem os testes realizados e 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)

Para fazer a documentação da API com o SwaggerUI, foi necessário adicionar o pacote Swashbuckle.AspNetCore e configurá-lo no arquivo de inicialização da API (Program.cs). Em seguida, foi criada uma classe de configuração do Swagger, onde foram definidos os detalhes da API, como o título, descrição e versão.

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

Figura - Geração documentação no Swagger

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

Uma imagem com texto

Descrição gerada automaticamente

Figura - Propriedades da Solução do projeto

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

Uma imagem com texto

Descrição gerada automaticamente

Figura - Documentação de métodos

Uma imagem com texto

Descrição gerada automaticamente

Figura - Documentação de modelos de dados

Sendo este o resultado no Swagger:

Uma imagem com texto

Descrição gerada automaticamente

Figura - Resultado documentação no Swagger

# Packages (Nuggets)

Nesta lista de pacotes Nuget usados, podemos observar que as camadas Api, BLL, BOL e DAL utilizam o pacote Swashbuckle.AspNetCore, que é uma ferramenta para gerar documentação da API usando o OpenAPI (anteriormente conhecido como Swagger). Também podemos ver que as camadas Api e DAL utilizam o pacote Npgsql, que é um driver de banco de dados para conectar a aplicação ao PostgreSQL. A camada Api também usa o pacote Microsoft.AspNetCore.Authentication.JwtBearer para autenticação JWT e a camada BLL usa o pacote Microsoft.AspNetCore.Mvc.Formatters.Json para formatação de respostas Json. As camadas BOL e DAL também usam o pacote System.IdentityModel.Tokens.Jwt, que fornece suporte para a manipulação de tokens JWT. Por fim, a camada DAL usa o pacote System.Data.SqlClient para se conectar ao SQL Server.

# Conclusão

Conclui-se que este trabalho prático foi uma oportunidade de aprendizagem e melhoria de habilidades na implementação de uma API em C#, com segurança, organização e testes unitários. Foi também uma oportunidade para adquirir novos conhecimentos, como trabalhar com strings de query para o SQL e dividir o projeto em camadas de forma eficiente. Estas habilidades e conhecimentos serão valiosos em futuros projetos e desenvolvimentos.

# 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>