

Centro de Alto Rendimento

Base de dados de gestão de um centro de alto rendimento

Universidade de Aveiro

Licenciatura em Engenharia de Computadores e Informática

Base de Dados P2G1

Regente: Prof. Carlos Costa

Professor Orientador: Prof. Tiago Marques Godinho

André Clérigo, 98485

Pedro da Rocha, 98256

26 de junho 2021

Índice

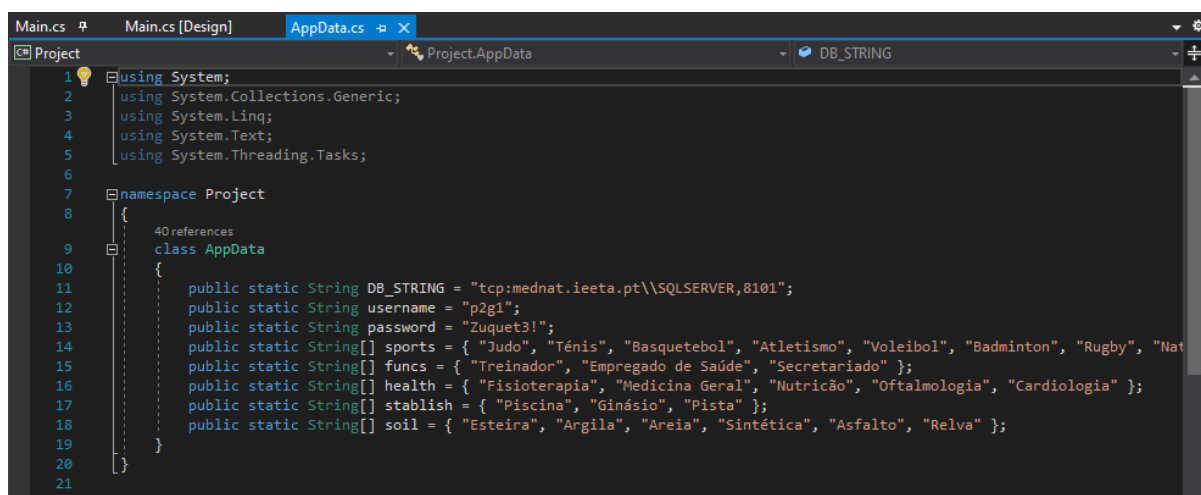
Introdução.....	3
Análise de Requisitos	4
Entidades e multiplicidade.....	4
Diagrama Entidade Relação (DER)	5
Esquema Relacional (ER).....	6
Stored Procedure (SP).....	6
View	7
User Defined Function (UDF)	8
Trigger	8
Index.....	10
Conclusão.....	10

Introdução

No âmbito da unidade curricular de Base de Dados, da Licenciatura em Engenharia de Computadores e Informática foi-nos proposto a criação de um projeto que fizesse a gestão de um sistema funcional e com complexidade razoável aplicável ao mundo real. O seguinte relatório irá descrever sucintamente o nosso projeto dando ênfase nas partes que achamos mais fulcrais para o funcionamento do mesmo.

O código enviado em anexo contém o código usado para criar a base de dados e utilizar a mesma no diretório **SQL-Code**, desenvolvemos um WFA em C# e o código relativo à mesma encontra-se no diretório **Aplicacao**.

Para que seja possível a testagem da nossa aplicação noutra base de dados para além daquela que foi usada no desenvolvimento nas aulas práticas é necessário aceder ao ficheiro **AppData.cs** e mudar os atributos **BD_STRING**, **username** e **password** para as credenciais desejadas.



```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace Project
8 {
9     class AppData
10     {
11         public static String DB_STRING = "tcp:mednat.ieeta.pt\\SQLSERVER,8101";
12         public static String username = "p2g1";
13         public static String password = "Zuquet3!";
14         public static String[] sports = { "Judo", "Ténis", "Basquetebol", "Atletismo", "Voleibol", "Badminton", "Rugby", "Nat
15         public static String[] funcs = { "Treinador", "Empregado de Saúde", "Secretariado" };
16         public static String[] health = { "Fisioterapia", "Medicina Geral", "Nutrição", "Oftalmologia", "Cardiologia" };
17         public static String[] stablish = { "Piscina", "Ginásio", "Pista" };
18         public static String[] soil = { "Esteira", "Argila", "Areia", "Sintética", "Asfalto", "Relva" };
19     }
20 }
21
```

Figura 1 - Ficheiro AppData.cs onde podemos alterar a conexão à base de dados utilizada

Segue também um conjunto de vídeos demonstrativos usados para a apresentação dentro do diretório de **vídeos**.

Análise de Requisitos

Podem registar-se atletas, identificados distintamente pelo seu número de inscrição nas seguintes modalidades: natação, judo, ténis, basquetebol, atletismo, voleibol, badminton, rugby.

- Existem 3 tipos de empregados: treinador, secretariado e empregados de saúde, conhecidos pelos seus números de funcionários.
- Para um treinador registar a avaliação de uma sessão de treino é necessário o atleta ter reservado anteriormente o estabelecimento onde pretendia treinar.
- Os estabelecimentos dividem-se entre piscinas, pistas e ginásios.
- De maneira a reservar espaços, os atletas têm que ter um diagnóstico 'apto' para treinar, realizado numa consulta pelos empregados de saúde, qualquer que seja a sua especialidade fisioterapia, medicina geral, cardiologia, oftalmologia, nutrição.

Entidades e multiplicidade

- **Atleta** é identificado por um `n_inscricao` e tem atributos não únicos como por exemplo, nome, modalidade, `estado_saude`, `data_nasc` e idade (atributo calculado a partir da `data_nasc`).
- **Empregado** é identificado por um `n_funcionario` e `nif` (sendo `n_funcionario` a chave primária) e tem atributos não únicos como o salário e nome.
- **Treinador, Secretariado e Empregado de saúde** que são entidades que se relacionam com o Empregado (is-a *disjoint*) e tem atributos específicos de modalidade, `id_estabelecimento` e especialidade respetivamente.
- **Reserva** é identificada por um `num_reserva` e tem atributos não únicos como por exemplo, `data_reserva`, `hora_inicio`, duração e `hora_fim` (calculado a partir da `hora_inicio` e `duracao`). Os atributos `n_insc_atleta`, `n_func_treinador` e `id_estabelecimento` são as chaves estrangeiras utilizadas.
- **Consulta** é identificada por um `id_consulta` e tem atributos não únicos como por exemplo, `data_consulta` e `estado_saude` (enum de "apto" ou "inapto"). Usada `n_func_emp` e `n_insc_atleta` como chaves estrangeiras.
- **Sessão de Treino** é identificada por um `num_reserva` usado como chave estrangeira e tem uma avaliação de 0 a 20.

- **Estabelecimento** identificado por um id e tem atributos não únicos como por exemplo, data_manutencao, data_aberto (após às 05h00), data_fecado (antes das 23h30) e uma lotação para espectadores que deve ser maior que 0.
- **Pista, Piscina e Ginásio** que são entidades que se relacionam com o Estabelecimento (is-a disjoint) e tem atributos específicos de tipo_solo, temp_agua e qtd_aparelhos respetivamente.

Diagrama Entidade Relação (DER)

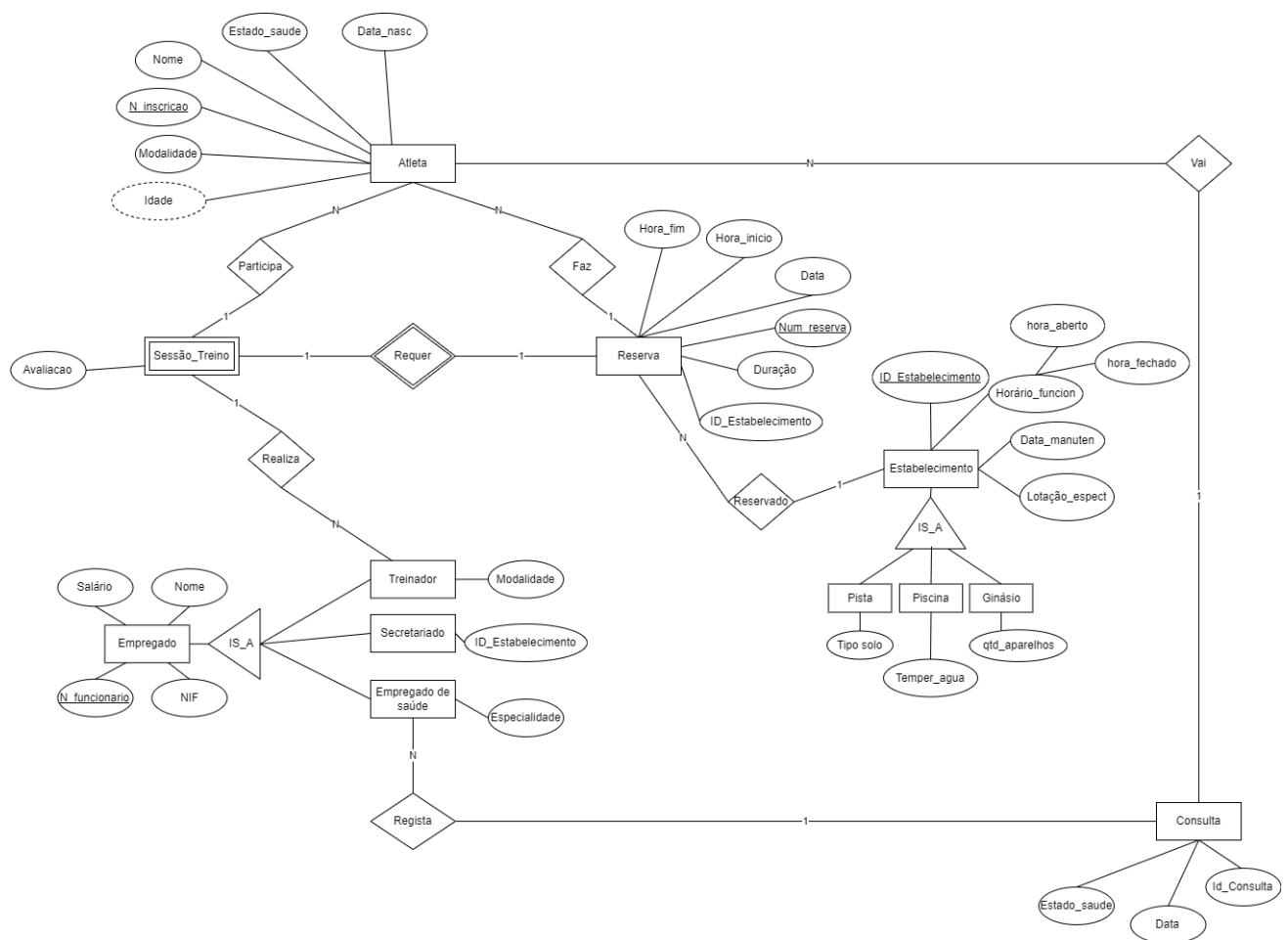


Figura 2 - Diagrama Entidade Relação

Esquema Relacional (ER)

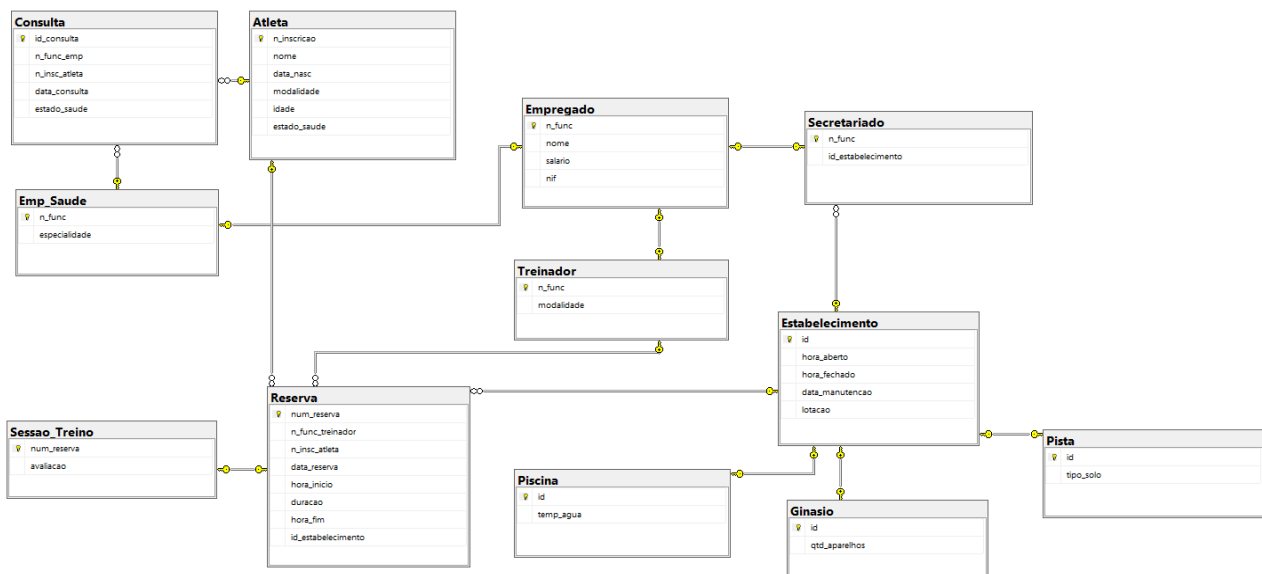


Figura 3 - Esquema Relacional

Stored Procedure (SP)

Para garantir uma relação is-a *disjoint* nas entidades Pista, Piscina, Ginásio, Secretariado, Empregado de Saúde e Treinador fizemos SPs que criam uma entidade mãe e logo de seguida criam a respetiva entidade pretendida.

```
-- Quando um treinador é adicionado, ele é adicionado à tabela de empregados
GO
CREATE PROC sp_add_treinador    @nome VARCHAR(50),
                                @salario FLOAT,
                                @nif INTEGER,
                                @modalidade VARCHAR(15)
AS
BEGIN
    INSERT INTO Empregado(nome, salario, nif) VALUES (@nome, @salario, @nif);
    INSERT INTO Treinador(n_func, modalidade) VALUES (SCOPE_IDENTITY(), @modalidade);
END
```

Figura 4 - Stored Procedure para adicionar Treinador

```

-- Quando uma piscina é adicionada, ela é adicionada à tabela de estabelecimentos
GO
CREATE PROC sp_add_piscina
    @hora_aberto TIME,
    @hora_fechado TIME,
    @data_manutencao DATE,
    @lotacao INTEGER,
    @temp_agua FLOAT
AS
BEGIN
    INSERT INTO Estabelecimento(hora_aberto, hora_fechado, data_manutencao, lotacao) VALUES (@hora_aberto, @hora_fechado, @data_manutencao, @lotacao);
    INSERT INTO Piscina(id, temp_agua) VALUES (SCOPE_IDENTITY(), @temp_agua);
END

```

Figura 5 - Stored Procedure para adicionar Piscina

View

Como os atletas têm a opção de verificar as estatísticas da sua avaliação e de outros por modalidade praticada, é criada uma tabela auxiliar (view) para ser mostrada ao utilizador da interface de maneira a comparar as suas avaliações com o panorama geral.

```

-- melhores avaliacoes por modalidade
CREATE VIEW melhores_avaliacoes_por_modalidade AS
SELECT modalidade, Atleta.nome, MAX(avaliacao) AS melhor_avaliacao
FROM Atleta
INNER JOIN Reserva ON Atleta.n_inscricao = Reserva.n_insc_atleta
INNER JOIN Sessao_treino ON Reserva.num_reserva = Sessao_treino.num_reserva
GROUP BY modalidade, nome;

```

Figura 6 - View melhor avaliação por modalidade

User Defined Function (UDF)

De forma a devolver uma tabela com todas as combinações disponíveis (isto é, nem o atleta nem ninguém tem alguma reserva a essa hora) para determinada data e hora de treino, podendo os estabelecimentos e o treinador serem indicados pelo utilizador ou considerar todos, utilizámos uma UDF que regista esses valores na tabela devolvida.

```
-- udf return table (record set) com estabelecimentos e treinadores nao reservados com os inputs data_reserva, hora_inicio, duracao
CREATE FUNCTION Reserva_disponivel(@data_reserva DATE, @hora_inicio TIME, @hora_fim TIME, @n_insc_atleta INT)
RETURNS @disponivel TABLE (estabelecimento INT, n_func_treinador INT, nome_treinador NVARCHAR(50))
AS
BEGIN
    IF NOT EXISTS (SELECT * FROM Reserva r WHERE r.n_insc_atleta = @n_insc_atleta
        AND (CAST(r.hora_inicio AS TIME) BETWEEN @hora_inicio AND @hora_fim)
        AND (CAST(r.hora_fim AS TIME) BETWEEN @hora_inicio AND @hora_fim))
    BEGIN
        INSERT @disponivel
        SELECT e.id, t.n_func, (SELECT nome FROM Empregado WHERE n_func = t.n_func)
        FROM Estabelecimento e
        CROSS JOIN Treinador t
        --estabelecimento nao esta reservado
        WHERE e.id NOT IN (SELECT e.id FROM Estabelecimento e
            INNER JOIN Reserva r ON r.id_estabelecimento = e.id
            WHERE r.data_reserva = @data_reserva AND r.id_estabelecimento = e.id
            AND (@hora_inicio < r.hora_inicio AND @hora_fim > r.hora_fim
                OR @hora_inicio < r.hora_inicio AND @hora_fim > r.hora_inicio
                OR @hora_inicio > r.hora_inicio AND @hora_fim < r.hora_fim))

        --treinador nao esta reservado
        AND t.n_func NOT IN (SELECT t.n_func FROM Treinador t
            INNER JOIN Reserva r ON r.n_func_treinador = t.n_func
            WHERE r.data_reserva = @data_reserva AND r.hora_inicio = @hora_inicio
            AND (@hora_inicio <= r.hora_inicio AND @hora_fim >= r.hora_fim
                OR @hora_inicio <= r.hora_inicio AND @hora_fim >= r.hora_inicio
                OR @hora_inicio >= r.hora_inicio AND @hora_fim <= r.hora_fim))

        --estabelecimento nao esta em manutencao
        AND e.id NOT IN (SELECT e.id FROM Estabelecimento e
            WHERE @data_reserva = e.data_manutencao)
        --hora_inicio nao pode ser antes da abertura do estabelecimento e nao pode ser depois da fechamento do estabelecimento
        AND e.id IN (SELECT e.id FROM Estabelecimento e WHERE @hora_inicio >= e.hora_aberto AND @hora_fim <= e.hora_fechado)
    END
    RETURN
END;
```

Trigger

Para manter a integridade dos dados da reserva ao serem adicionadas e não existirem colisões de horários, criou-se um *trigger* que é ativado ao adicioná-la, que verifica se as horas de treino da reserva acontecem durante o horário do estabelecimento e que a data de reserva não colide com a data de manutenção. Certifica ainda que o atleta está apto de saúde para treinar e que a data da última consulta não foi há mais de 6 meses. Por último, confere que o atleta não tem já uma reserva nessa data e hora.


```

CREATE TRIGGER check_reserva ON Reserva
INSTEAD OF INSERT, UPDATE
AS
SET NOCOUNT ON;
DECLARE @hora_inicio TIME = (SELECT hora_inicio FROM inserted);
DECLARE @hora_fim TIME = (SELECT hora_fim FROM inserted);
DECLARE @duracao TIME = (SELECT duracao FROM inserted);
DECLARE @id_estabelecimento INT = (SELECT id_estabelecimento FROM inserted);
DECLARE @data_reserva DATE = (SELECT data_reserva FROM inserted);
DECLARE @hora_fechado TIME = (SELECT CAST(hora_fechado AS TIME) FROM Estabelecimento WHERE id=@id_estabelecimento);
DECLARE @hora_aberto TIME = (SELECT CAST(hora_aberto AS TIME) FROM Estabelecimento WHERE id=@id_estabelecimento);
DECLARE @data_ult_consulta DATE = (SELECT TOP 1 CAST(data_consulta AS DATE) FROM Consulta
WHERE n_insc_atleta = (SELECT n_insc_atleta FROM inserted) ORDER BY data_consulta DESC);
DECLARE @lotacao INT = (SELECT lotacao FROM Estabelecimento WHERE id=@id_estabelecimento);
DECLARE @num_reserva INT = (SELECT num_reserva FROM inserted);
DECLARE @n_insc_atleta INT = (SELECT n_insc_atleta FROM inserted);
DECLARE @n_func_treinador INT = (SELECT n_func_treinador FROM inserted);
BEGIN TRY
    IF (@hora_inicio < @hora_aberto)
    BEGIN
        RAISERROR ('A hora de inicio não pode ser antes da hora de abertura do estabelecimento', 1, 1);
        ROLLBACK TRAN;
    END
    IF (DATEADD(SECOND, DATEDIFF(SECOND, 0, @hora_inicio), @duracao) > @hora_fechado)
    BEGIN
        RAISERROR ('A hora de fim do treino não pode ser antes da hora de fecho do estabelecimento!', 1, 1);
        ROLLBACK TRAN;
    END
    IF @data_reserva = (SELECT CAST(data_manutencao AS DATE) FROM Estabelecimento WHERE id=@id_estabelecimento)
    BEGIN
        RAISERROR ('A data da reserva nao pode ser a mesma que a data de manutenção do estabelecimento!', 1, 1);
        ROLLBACK TRAN;
    END
    -- Verificar se a data da ultima consulta foi há mais de 6 meses
    IF (@data_ult_consulta IS NULL)
    BEGIN
        RAISERROR ('Não existem consultas anteriores a essa!', 1, 1);
        ROLLBACK TRAN;
    END
    IF (DATEDIFF(MONTH, @data_ult_consulta, @data_reserva) > 5)
    BEGIN
        RAISERROR ('A data da última consulta foi há mais de 6 meses, por favor, marque uma consulta!', 1, 1);
        ROLLBACK TRAN;
    END
    --verifica se o atleta está apto
    IF (SELECT estado_saude FROM Atleta WHERE n_inscricao = @n_insc_atleta) != 'apto'
    BEGIN
        RAISERROR ('O atleta não está apto para treinar!', 1, 1);
        ROLLBACK TRAN;
    END
    -- verifica que o atleta nao tem já uma reserva à mesma hora
    IF EXISTS (SELECT * FROM Reserva r WHERE n_insc_atleta = @n_insc_atleta AND data_reserva = @data_reserva
        AND (@hora_inicio = r.hora_inicio OR @hora_fim = r.hora_fim
        OR @hora_inicio < r.hora_inicio AND @hora_fim > r.hora_fim
        OR @hora_inicio < r.hora_inicio AND @hora_fim > r.hora_fim
        OR @hora_inicio BETWEEN r.hora_inicio AND r.hora_fim
        AND @hora_inicio NOT IN (r.hora_inicio, r.hora_fim)
        OR @hora_fim BETWEEN r.hora_inicio AND r.hora_fim
        AND @hora_fim NOT IN (r.hora_inicio, r.hora_fim)))
    BEGIN
        RAISERROR ('O atleta já tem uma reserva à mesma hora!', 1, 1);
        THROW 1, 'O atleta tem uma reserva à mesma hora!', 1;
    END
    -- inserir a reserva
    INSERT INTO Reserva(n_func_treinador, n_insc_atleta, data_reserva, hora_inicio, duracao, id_estabelecimento)
    VALUES (@n_func_treinador, @n_insc_atleta, @data_reserva, @hora_inicio, @duracao, @id_estabelecimento);
END TRY

BEGIN CATCH
    PRINT ('');
END CATCH

```

Figura 7 - Trigger check reserva

Index

Com vista a iterar mais eficazmente pelas reservas existentes, organizou-se a tabela de Reserva pelas datas, visto que o acesso mais complexo à mesma verifica os dados por esse mesmo atributo.

```
-- index pela data (pelos wheres que utilizamos na UDF)  
CREATE INDEX index_data_reserva ON Reserva(data_reserva);
```

Figura 8 - Index por data

Conclusão

Podemos concluir que os principais objetivos e as principais funcionalidades da aplicação para a gestão de um centro de alto rendimento foram cumpridas. O empregado do secretariado pode adicionar, remover Atletas, Empregados de Saúde, Treinadores, Secretariado, Piscinas, Pistas e Ginásios.

Os empregados de saúde podem registar consultas e verificar o histórico de consultas de um atleta.

Os treinadores podem registar uma sessão de treino dando uma avaliação (0-20) ao mesmo, podendo consultar as reservas feitas para si.

Os atletas podem consultar as suas reservas, fazer reservas novas, consultar as suas avaliações e as melhores avaliações de cada modalidade (leaderboard por modalidade).