Relatório do trabalho da disciplina de Armazenamento e Acesso a Dados

Stand AutoIPCA

João Castro - 21143

João Cunha - 23519

Paulo Costa - 29851

Engenharia de Sistemas Informáticos

Setembro de 2024

|  |
| --- |
| Afirmo por minha honra que não recebi qualquer apoio não autorizado na realização deste trabalho prático. Afirmo igualmente que não copiei qualquer material de livro, artigo, documento web ou de qualquer outra fonte exceto onde a origem estiver expressamente citada. |

João Castro - 21143

João Cunha - 23519

Paulo Costa - 29851

Índice

[Stand AutoIPCA 1](#_Toc187506177)

[Descrição do negócio 1](#_Toc187506178)

[Âmbito do negócio 1](#_Toc187506179)

[Objetivos empresariais 1](#_Toc187506180)

[Caraterísticas do serviço 2](#_Toc187506181)

[Business rules 2](#_Toc187506182)

[Diagrama ER 3](#_Toc187506183)

[Descrição de tabelas 4](#_Toc187506184)

[SQL 5](#_Toc187506185)

[Criação da database e das respetivas tabelas 5](#_Toc187506186)

[Resolução de problemas 8](#_Toc187506187)

[Criação de Index’s e Constraint’s 9](#_Toc187506188)

[Triggers 10](#_Toc187506189)

[Views 11](#_Toc187506190)

[Stored Procedures 12](#_Toc187506191)

[Desenvolvimento da Aplicação 14](#_Toc187506192)

[Outros Ficheiros 16](#_Toc187506193)

[Conclusão 17](#_Toc187506194)

**Lista de Figuras**

[Figura 1 - Diagrama ER 3](#_Toc187505512)

[Figura 2- Criação da tabela Utilizador 14](#_Toc187505513)

[Figura 3- Inserção de dados na tabela utilizador 14](#_Toc187505514)

[Figura 4- Código desenvolvido em C# para a implementação da funcionalidade de login 14](#_Toc187505515)

[Figura 5- Login na app 15](#_Toc187505516)

[Figura 6- Menu da app 15](#_Toc187505517)

[Figura 7-Inserir Cliente 16](#_Toc187505518)

[Figura 8- Confirmação do cliente inserido 16](#_Toc187505519)

# Stand AutoIPCA

## Descrição do negócio

Somos um serviço de aluguer de automóveis exclusivamente para estudantes universitários. O nosso objetivo é oferecer uma solução acessível e conveniente para que os estudantes tenham mobilidade durante todo o ano académico. Compreendemos que muitos estudantes precisam de um transporte flexível para se deslocarem entre casa, a universidade, os estágios e as atividades diárias, e é exatamente isso que oferecemos.

A nossa frota é composta por uma variedade de veículos, desde carros compactos a modelos maiores, todos cuidadosamente mantidos para garantir a segurança e o conforto dos nossos clientes. Operamos dentro do campus, facilitando o acesso dos estudantes aos nossos veículos, e oferecemos tarifas especiais adaptadas aos orçamentos dos estudantes, com opções de aluguer de curto e longo prazo. Para além disso, a nossa plataforma online garante que o processo de reserva e pagamento é simples e rápido, sem papelada desnecessária.

## Âmbito do negócio

O nosso serviço destina-se a estudantes universitários, o que significa que damos prioridade à simplicidade e eficiência em todas as fases do processo de aluguer. Os estudantes podem efetuar reservas através de uma plataforma digital, verificar a disponibilidade dos veículos em tempo real e efetuar pagamentos diretamente online. Este sistema permite-nos gerir eficazmente a nossa frota, desde a manutenção dos veículos até ao acompanhamento das reservas e das devoluções.

Oferecemos também opções de aluguer flexíveis - diárias, semanais ou mesmo para todo o ano letivo - de acordo com as necessidades de cada estudante. Estamos constantemente sintonizados com as exigências do nosso público, ajustando o nosso serviço para o tornar o mais conveniente possível.

## Objetivos empresariais

Os nossos principais objetivos são:

* **Facilitar a mobilidade dos estudantes**: O nosso objetivo é garantir que os estudantes tenham uma solução de transporte acessível e sem complicações, permitindo-lhes concentrar-se nos seus estudos e atividades.
* **Maximizar a utilização da frota:** Esforçamo-nos por manter uma elevada taxa de ocupação dos nossos veículos, garantindo que estão sempre disponíveis e em boas condições de funcionamento.
* **Prestar um serviço prático e acessível:** Através da nossa plataforma digital, tornamos todo o processo de aluguer rápido e simples, desde a reserva até à devolução do veículo.
* **Criar confiança:** Acreditamos que a confiança é fundamental. Por isso, garantimos que todos os veículos são submetidos a manutenção regular e que o nosso apoio ao cliente é reativo e eficiente.

## Caraterísticas do serviço

O nosso serviço oferece várias caraterísticas para garantir uma ótima experiência ao cliente:

* **Plataforma de reservas online:** Os alunos podem ver os carros disponíveis e fazer reservas a partir de qualquer lugar, a qualquer momento.
* **Gestão simplificada:** A nossa equipa pode gerir eficazmente a frota, monitorizar a utilização e o estado dos veículos e programar a manutenção.
* **Processo de aluguer sem falhas:** Desde a reserva até à devolução do veículo, os alunos experimentam um processo tranquilo, com apoio disponível para quaisquer questões ou problemas.
* **Pagamentos flexíveis:** Para além de aceitarem vários métodos de pagamento, os estudantes podem optar por dividir os custos do aluguer ao longo do período de aluguer.
* **Preços acessíveis:** Oferecemos preços adaptados aos orçamentos dos estudantes, bem como descontos para alugueres de longa duração.
* **Apoio e manutenção:** A nossa equipa efetua inspeções regulares à frota, garantindo que os veículos estão sempre em perfeitas condições.

## Business rules

Para garantirmos um processo de aluguer transparente e sem problemas, aplicam-se as seguintes regras comerciais:

* **Devoluções antecipadas:** Se um estudante decidir devolver um veículo antes do fim do período de aluguer acordado, não será emitido qualquer reembolso pelo tempo não utilizado. Isto garante que o planeamento e a disponibilidade do aluguer não são perturbados.
* **Danos no veículo:** Em caso de danos no veículo alugado, o estudante será responsável pelos custos de reparação até um montante de franquia pré-determinado. A extensão da responsabilidade será delineada no contrato de aluguer. No caso de danos menores, poderá ser aplicada uma taxa fixa, enquanto os danos maiores exigirão uma avaliação completa.
* **Devoluções tardias:** Se o veículo não for devolvido a tempo, será aplicada uma taxa de atraso por cada dia de atraso. A não devolução do veículo dentro de um determinado período de carência (por exemplo, 48 horas após a data de vencimento) pode resultar em penalizações adicionais, incluindo a suspensão temporária do aluguer de veículos no futuro.
* **Infrações de trânsito e multas:** Quaisquer multas ou penalizações incorridas durante o período de aluguer, tais como multas de estacionamento ou infrações de trânsito, são da responsabilidade do estudante que aluga o veículo. Estas devem ser pagas diretamente pelo estudante, ou o montante será debitado na sua conta.

## Diagrama ER

Figura - Diagrama ER

### Descrição de tabelas

* **Cliente** - Armazena informações sobre os clientes, como ID, nome, data de nascimento e código postal;
* **Código Postal** - Contém códigos postais e localidades, facilitando o armazenamento da localização dos clientes.
* **Contato** - Representa o contato (telefone, e-mail, etc.) que um cliente pode ter.
* **Tipos de Contacto** - Define os tipos de contato (ex.: telefone, e-mail) usados na tabela Contato.
* **ClienteVendedor** - Relação entre clientes e vendedores, indicando quais clientes estão associados a quais vendedores.
* **Vendedor** - Contém dados sobre os vendedores, incluindo ID e nome.
* **Tipo de Veículo** - Define o tipo de veículo (carro, moto, etc.) e relaciona-os com os veículos específicos.
* **Marca** - Armazena as marcas dos veículos.
* **Veículo** - Contém informações sobre cada veículo, incluindo ID, marca, tipo e outros detalhes.
* **Aluguer** - Regista informações de aluguer de veículos, associando clientes e vendedores a veículos alugados.
* **Pagamento** - Regista informações sobre pagamentos de reservas, incluindo ID de pagamento e valor.
* **Fatura** - Contém informações sobre as faturas de reservas emitidas.
* **Manutenção** - Armazena registos de manutenção para os veículos, incluindo tipo de manutenção, custo, entre outros detalhes.
* **TipoMan** - Define os tipos de manutenção realizados nos veículos.
* **Multa** - Armazena informações sobre multas aplicadas a clientes.
* **TipoMulta** - Define os tipos de multas aplicáveis.

# SQL

## Criação da database e das respetivas tabelas

-- Criação da Database

CREATE DATABASE autoIPCA;

-- Utiliza a Database criada "autoIPCA"

USE autoIPCA;

-- Criação da tabela "CP"

CREATE TABLE CP(

CPid INTEGER PRIMARY KEY,

Localidade VARCHAR(255) NOT NULL

);

-- Criação da tabela "TipoMulta"

CREATE TABLE TipoMulta(

TMultaID INTEGER PRIMARY KEY,

Tipo VARCHAR(255) NOT NULL

);

-- Criação da tabela "Cliente"

CREATE TABLE Cliente(

ClienteID INTEGER PRIMARY KEY,

Nome VARCHAR(255) NOT NULL,

Email VARCHAR(255),

DataNasc DATE,

NumeroDeAlugures INTEGER,

CPid INTEGER,

FOREIGN KEY (CPid) REFERENCES CP(CPid)

);

-- Criação da tabela "Multa"

CREATE TABLE Multa(

MultaID INTEGER PRIMARY KEY,

Valor INTEGER NOT NULL,

StatusPag VARCHAR(255) NOT NULL,

DataMulta DATE,

ClienteID INTEGER,

CPid INTEGER,

TMultaID INTEGER,

FOREIGN KEY (ClienteID) REFERENCES Cliente(ClienteID),

FOREIGN KEY (CPid) REFERENCES CP(CPid),

FOREIGN KEY (TMultaID) REFERENCES TipoMulta(TMultaID)

);

-- Criação da tabela "Vendedor"

CREATE TABLE Vendedor (

VendedorID INTEGER PRIMARY KEY,

Nome VARCHAR(255) NOT NULL,

Email VARCHAR(255),

NumeroDeVendas INTEGER NOT NULL,

DataNasc DATE

);

-- Criação da tabela "ClienteVendedor"

CREATE TABLE ClienteVendedor (

VendedorID INTEGER,

ClienteID INTEGER,

PRIMARY KEY (VendedorID, ClienteID),

FOREIGN KEY (VendedorID) REFERENCES Vendedor(VendedorID),

FOREIGN KEY (ClienteID) REFERENCES Cliente(ClienteID)

);

-- Criação da tabela "Marca"

CREATE TABLE Marca (

MarcaID INTEGER PRIMARY KEY,

Nome VARCHAR(255) NOT NULL

);

-- Criação da tabela "TipoDeVeiculo"

CREATE TABLE TipoDeVeiculo (

TipoVeilID INTEGER PRIMARY KEY,

Tipo VARCHAR(255),

MarcaID INTEGER,

FOREIGN KEY (MarcaID) REFERENCES Marca(MarcaID)

);

-- Criação da tabela "Veiculo"

CREATE TABLE Veiculo (

VeicID INTEGER PRIMARY KEY,

Matricula VARCHAR(255) NOT NULL,

Lugares INTEGER,

Preco INTEGER,

Ano INTEGER,

Cor VARCHAR(255),

MarcaID INTEGER,

TipoVeilID INTEGER,

FOREIGN KEY (MarcaID) REFERENCES Marca(MarcaID),

FOREIGN KEY (TipoVeilID) REFERENCES TipoDeVeiculo(TipoVeilID)

);

-- Criação da tabela "Pagamento"

CREATE TABLE Pagamento (

PagID INTEGER PRIMARY KEY,

Valor INTEGER NOT NULL

);

-- Criação da tabela "Aluguer"

CREATE TABLE Aluguer (

AluguerID INTEGER PRIMARY KEY,

Custo INTEGER,

Comissao INTEGER,

Data DATE,

ClientID INTEGER,

VendedorID INTEGER,

PagID INTEGER,

VeicID INTEGER,

TipoVeilID INTEGER,

CPid INTEGER,

FOREIGN KEY (ClientID) REFERENCES Cliente(ClienteID),

FOREIGN KEY (VendedorID) REFERENCES Vendedor(VendedorID),

FOREIGN KEY (PagID) REFERENCES Pagamento(PagID),

FOREIGN KEY (VeicID) REFERENCES Veiculo(VeicID),

FOREIGN KEY (TipoVeilID) REFERENCES TipoDeVeiculo(TipoVeilID),

FOREIGN KEY (CPid) REFERENCES CP(CPid)

);

-- Criação da tabela "Fatura"

CREATE TABLE Fatura (

FATID INTEGER PRIMARY KEY,

Nif INTEGER,

PagID INTEGER,

FOREIGN KEY (PagID) REFERENCES Pagamento(PagID)

);

-- Criação da tabela "TiposDeContacto"

CREATE TABLE TiposDeContacto (

TPID INTEGER PRIMARY KEY,

Tipo VARCHAR(255)

);

-- Criação da tabela "Contacto"

CREATE TABLE Contacto (

CID INTEGER PRIMARY KEY,

Email VARCHAR(255),

Telefone INTEGER,

TPID INTEGER,

ClientID INTEGER,

VendedorID INTEGER,

FOREIGN KEY (TPID) REFERENCES TiposDeContacto(TPID),

FOREIGN KEY (ClientID) REFERENCES Cliente(ClienteID),

FOREIGN KEY (VendedorID) REFERENCES Vendedor(VendedorID)

);

-- Criação da tabela "TipoMan"

CREATE TABLE TipoMan (

TManID INTEGER PRIMARY KEY,

Tipo VARCHAR(255)

);

-- Criação da tabela "Manutencao"

CREATE TABLE Manutencao (

ManID INTEGER PRIMARY KEY,

DataMan DATE,

Descricao VARCHAR(255),

Custo INTEGER,

VeicID INTEGER,

MarcaID INTEGER,

TipoVeilID INTEGER,

TManID INTEGER,

FOREIGN KEY (VeicID) REFERENCES Veiculo(VeicID),

FOREIGN KEY (MarcaID) REFERENCES Marca(MarcaID),

FOREIGN KEY (TipoVeilID) REFERENCES TipoDeVeiculo(TipoVeilID),

FOREIGN KEY (TManID) REFERENCES TipoMan(TManID)

);

## Resolução de problemas

Criamos uma série de 10 problemas para respondermos usando query’s.

1. Quais os vendedores sem aluguéis atribuidos?

SELECT Vendedor.Nome, COUNT(AluguerID) AS NumAlugueis

FROM Vendedor LEFT JOIN Aluguer on Vendedor.VendedorID=Aluguer.VendedorID

GROUP BY Vendedor.Nome, Aluguer.AluguerID

HAVING COUNT(Aluguer.AluguerID) = 0;

1. Qual o nome, morada e idade do cliente que menos aluguéis realizou?

SELECT Cliente.Nome, CP.Localidade, DATEDIFF(YEAR, Cliente.DataNasc, GETDATE()) AS Idade

FROM Cliente JOIN CP ON Cliente.CPid=CP.CPid

LEFT JOIN Aluguer ON Aluguer.ClientID=Cliente.ClienteID

GROUP BY Cliente.Nome, CP.Localidade, Cliente.DataNasc

HAVING COUNT(Aluguer.AluguerID) =

(SELECT MIN(contagem)

FROM (SELECT COUNT(Aluguer.AluguerID) AS contagem

FROM Aluguer LEFT JOIN Cliente ON Aluguer.ClientID=Cliente.ClienteID

GROUP BY Cliente.ClienteID

) as subconsulta

);

1. Quantos carros foram alugados no ano passado?

SELECT COUNT(DISTINCT Veiculo.VeicID) as alugueis

FROM Veiculo JOIN Aluguer ON Aluguer.VeicID=Veiculo.VeicID

WHERE YEAR(Aluguer.Data) = YEAR(GETDATE())-1;

1. Quais os clientes com mais do que 1 aluguer?

SELECT Cliente.ClienteID, Cliente.Nome

FROM Cliente

JOIN Aluguer ON Cliente.ClienteID = Aluguer.ClientID

GROUP BY Cliente.ClienteID, Cliente.Nome

HAVING COUNT(Aluguer.AluguerID) > 1;

1. Qual o vendedor que realizou mais aluguéis?

SELECT Vendedor.VendedorID, Vendedor.Nome

FROM Vendedor

JOIN Aluguer ON Vendedor.VendedorID=Aluguer.VendedorID

GROUP BY Vendedor.VendedorID, Vendedor.Nome

HAVING COUNT(Aluguer.AluguerID) = (SELECT MAX(contagem)

FROM (SELECT COUNT(Aluguer.AluguerID) AS contagem

FROM Aluguer GROUP BY Aluguer.VendedorID) AS subconsulta

);

1. Qual a média do aluguer de um carro?

SELECT AVG(Aluguer.Custo) AS MediaCusto

FROM Aluguer;

1. Qual é o carro mais caro?

SELECT DISTINCT Veiculo.VeicID, Veiculo.Matricula

FROM Veiculo

JOIN Aluguer ON Aluguer.VeicID=Veiculo.VeicID

WHERE Aluguer.Custo = (SELECT MAX(Aluguer.Custo)

FROM Aluguer);

1. Qual é a marca de carro que foi mais alugada?

SELECT Marca.MarcaID, Marca.Nome

FROM Marca

JOIN Veiculo ON Marca.MarcaID=Veiculo.VeicID

JOIN Aluguer ON Aluguer.VeicID=Veiculo.VeicID

GROUP BY Marca.MarcaID, Marca.Nome

HAVING COUNT(Aluguer.AluguerID) = (SELECT MAX(contagem)

FROM (SELECT COUNT(Aluguer.AluguerID) AS contagem

FROM Aluguer

JOIN Veiculo ON Aluguer.VeicID = Veiculo.VeicID

GROUP BY Veiculo.MarcaID)

AS subconsulta);

1. Qual o nome do vendedor que recebeu maior comissão no aluguer de um carro?

SELECT DISTINCT Vendedor.VendedorID, Vendedor.Nome

FROM Vendedor

JOIN Aluguer ON Aluguer.VendedorID=Vendedor.VendedorID

WHERE Aluguer.Comissao=(SELECT MAX(Aluguer.Comissao) FROM Aluguer);

1. Qual a localidade com maior numero de clientes?

SELECT CP.CPid, CP.Localidade

FROM CP

JOIN Cliente ON Cliente.CPid=CP.CPid

GROUP BY CP.CPid, CP.Localidade

HAVING COUNT(Cliente.CPid)=(SELECT MAX(contagem)

FROM (SELECT COUNT(Cliente.CPid) AS contagem

FROM Cliente

GROUP BY Cliente.CPid)

AS subconsulta);

## Criação de Index’s e Constraint’s

Índices são estruturas adicionais associadas a tabelas ou colunas que permitem acelerar a recuperação de dados enquanto constraints são restrições aplicadas a colunas ou tabelas para garantir a integridade e consistência dos dados.

Indices para que servem?

* Melhoraram o desempenho das consultas.
* Reduzem o tempo de pesquisa.
* Facilitam a ordenação e agrupamento eficiente de dados.

Constraints para que servem?

* Impedem a inserção de dados inválidos.
* Eliminam a necessidade de validações complexas no lado da aplicação.
* Evitam erros humanos ou automáticos durante inserções ou atualizações.

|  |
| --- |
| -- Índices para melhorar desempenho em consultas  CREATE INDEX idx\_cliente\_cpid ON Cliente(CPid);  CREATE INDEX idx\_aluguer\_clientid ON Aluguer(ClientID);  CREATE INDEX idx\_veiculo\_marcaid ON Veiculo(MarcaID);  CREATE INDEX idx\_multa\_clienteid ON Multa(ClienteID);  -- Datas coerentes  ALTER TABLE Cliente ADD CONSTRAINT chk\_data\_nasc CHECK (DataNasc < GETDATE());  ALTER TABLE Aluguer ADD CONSTRAINT chk\_data\_aluguer CHECK (Data < GETDATE());  -- Valores positivos  ALTER TABLE Veiculo ADD CONSTRAINT chk\_preco\_positivo CHECK (Preco > 0);  ALTER TABLE Aluguer ADD CONSTRAINT chk\_custo\_positivo CHECK (Custo >= 0);  ALTER TABLE Manutencao ADD CONSTRAINT chk\_custo\_manutencao CHECK (Custo >= 0);  -- Validação de emails  ALTER TABLE Cliente ADD CONSTRAINT chk\_email\_cliente CHECK (Email LIKE '%@%.%');  ALTER TABLE Vendedor ADD CONSTRAINT chk\_email\_vendedor CHECK (Email LIKE '%@%.%'); |

## Triggers

Os Triggers são instruções SQL que são executadas automaticamente em resposta a eventos como INSERT, UPDATE ou DELETE numa tabela.

Para que servem:

* Automatizam regras de negócio (ex.: calcular penalizações).
* Mantêm a integridade dos dados (ex.: atualizar colunas dependentes).
* Executam tarefas administrativas automaticamente.

|  |
| --- |
| -- Penalização por atrasos  CREATE TRIGGER PenalizarAtrasos  ON Aluguer  AFTER UPDATE  AS  BEGIN  IF EXISTS (SELECT 1 FROM inserted WHERE Data > GETDATE())  BEGIN  INSERT INTO Multa (MultaID, Valor, StatusPag, DataMulta, ClienteID, CPid, TMultaID)  VALUES (DEFAULT, 50, 'Por pagar', GETDATE(), (SELECT ClientID FROM inserted), (SELECT CPid FROM inserted), 1);  END  END;  -- Atualizar número de alugueres  CREATE TRIGGER AtualizarNumeroAlugueres  ON Aluguer  AFTER INSERT  AS  BEGIN  UPDATE Cliente  SET NumeroDeAlugures = NumeroDeAlugures + 1  WHERE ClienteID = (SELECT ClientID FROM inserted);  END; |

## Views

As Views são consultas SQL armazenadas na base de dados como objetos. Funcionam como uma tabela virtual, mostrando os resultados de uma query.

Para que servem:

* Simplificam consultas complexas, evitando a necessidade de reescrevê-las.
* Melhoram a segurança, pois permitem exibir apenas colunas específicas sem expor dados confidenciais.
* Facilitam a integração com aplicações, tornando o acesso aos dados mais direto.

Para executar as Views basta escrever SELECT \* FROM vw\_nomedaview;

|  |
| --- |
| -- Relatório de multas  CREATE VIEW vw\_multas\_cliente AS  SELECT  Cliente.Nome AS Cliente,  Multa.Valor AS Multa,  Multa.StatusPag AS EstadoPagamento,  Multa.DataMulta AS DataMulta,  TipoMulta.Tipo AS TipoMulta  FROM Multa  JOIN Cliente ON Multa.ClienteID = Cliente.ClienteID  JOIN TipoMulta ON Multa.TMultaID = TipoMulta.TMultaID;  -- Resumo de alugueres  CREATE VIEW vw\_resumo\_alugueres AS  SELECT  Cliente.Nome AS Cliente,  Veiculo.Matricula AS Veiculo,  Aluguer.Custo AS Custo,  Aluguer.Data AS DataAluguer  FROM Aluguer  JOIN Cliente ON Aluguer.ClientID = Cliente.ClienteID  JOIN Veiculo ON Aluguer.VeicID = Veiculo.VeicID; |

## Stored Procedures

As *Stored Procedures* são blocos de código SQL pré-compilados que podem ser executados por comandos e ajudam em:

* Encapsular a logica sql.
* Melhorar desempenho.
* Permitem reutilizar e organizar código.

Criámos algumas *Stored Procedures são elas:*

* ProcurarCliente: Encontra o cliente através do ClienteID.
* InserirCliente: Cria um novo cliente.
* ProcurarAlugueres: Encontra todos os alugueres realizados por um certo cliente.
* EleminarCliente: Elemina um cliente de todas as tabelas.

Para executar estas stored procedures basta utilizar o comando EXEC .

CREATE PROCEDURE ProcurarCliente

@ClienteID INT

AS

BEGIN

SELECT \* FROM Cliente WHERE Cliente.ClienteID=@ClienteID;

END;

|  |
| --- |
| CREATE PROCEDURE ProcurarAlugueres  @ClienteID INT  AS  BEGIN  SELECT \* FROM Aluguer WHERE ClientID = @ClienteID;  END;  CREATE PROCEDURE InserirCliente  @ClienteID INT,  @Nome NVARCHAR(255),  @Email NVARCHAR(255),  @DataNasc DATE,  @NumeroDeAlugueres INT,  @CPid INT  AS  BEGIN  INSERT INTO Cliente (ClienteID, Nome, Email, DataNasc, NumeroDeAlugures, CPid)  VALUES (@ClienteID, @Nome, @Email, @DataNasc, 0, @CPid);  END;  CREATE PROCEDURE EliminarCliente  @ClienteID INT  AS  BEGIN  -- Iniciar uma transação para garantir consistência  BEGIN TRANSACTION;  BEGIN TRY  -- Eliminar alugueres associados ao cliente  DELETE FROM Aluguer WHERE ClientID = @ClienteID;  -- Eliminar multas associadas ao cliente  DELETE FROM Multa WHERE ClienteID = @ClienteID;  -- Eliminar contactos associados ao cliente  DELETE FROM Contacto WHERE ClientID = @ClienteID;  -- Eliminar relações com vendedores  DELETE FROM ClienteVendedor WHERE ClienteID = @ClienteID;  -- Eliminar o cliente  DELETE FROM Cliente WHERE ClienteID = @ClienteID;  -- Confirmar a transação  COMMIT TRANSACTION;  -- Mensagem de sucesso  PRINT 'Cliente eliminado com sucesso.';  END TRY  BEGIN CATCH  -- Reverter a transação em caso de erro  ROLLBACK TRANSACTION;  -- Mostrar mensagem de erro  PRINT 'Erro ao eliminar o cliente.';  END CATCH;  END; |

## Desenvolvimento da Aplicação

Implementação da funcionalidade de login:

Uma imagem com texto, captura de ecrã, Tipo de letra, file

Descrição gerada automaticamente

Figura - Criação da tabela Utilizador

Uma imagem com texto, Tipo de letra, captura de ecrã, file

Descrição gerada automaticamente

Figura - Inserção de dados na tabela utilizador

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura - Código desenvolvido em C# para a implementação da funcionalidade de login

Uma imagem com texto, Tipo de letra, captura de ecrã, preto

Descrição gerada automaticamente

Figura - Login na app

Utilização do menu da app para inserção de um novo cliente:

Uma imagem com texto, Tipo de letra, captura de ecrã, tipografia

Descrição gerada automaticamente

Figura - Menu da app

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura -Inserir Cliente

Uma imagem com texto, captura de ecrã, Tipo de letra, preto

Descrição gerada automaticamente

Figura - Confirmação do cliente inserido

## Outros Ficheiros

Outros ficheiros como todas as querys efetuadas sobre a base de dados podem ser encontradas na pasta querys dentro do trabalho enviado.

## Conclusão

O desenvolvimento deste sistema para uma stand estudantil representou uma aplicação prática da matéria da cadeira de Armazenamento e Acesso a Dados.

Com este projeto conseguimos entender melhor a importância da normalização de tabelas da utilização de Stored Procedures, Triggers, Views, Indexes e contraints para otimizar o acesso e a manipulação de dados, como também o desenvolvimento de uma aplicação de manipulação de dados . Em suma conseguimos aplicar os conceitos desta cadeira nos problemas criados por nós próprios promovendo a investigação e inovação.