universidade de aveiro deti departamento de electrónica, telecomunicações e informática

Sistema de Gestão de Eventos

Base de Dados

Prof. Carlos Costa

Prof. Joaquim Sousa Pinto

JUNHO 2024

FRANCISCO ALBERGARIA (114646) | JOÃO OLIVEIRA (102631)

Índice

Introdução

O desenvolvimento de um Sistema de Gestão de Eventos é um projeto algo complexo e que envolve a integração de diversas funcionalidades para garantir a eficiência e a eficácia na organização e execução de eventos. Este relatório documenta todas as fases do projeto, desde a análise de requisitos até à implementação das diversas queries e ainda uma breve demonstração da interface final. O objetivo principal deste sistema é facilitar a gestão de eventos, proporcionando uma plataforma onde clientes e gestores possam interagir de forma integrada e segura.

Na fase inicial, foi realizada uma análise detalhada dos requisitos funcionais e não funcionais do sistema, garantindo que todas as necessidades dos utilizadores fossem concretizadas.

O projeto inclui a criação de um Diagrama Entidade-Relacionamento (DER) e um Esquema Relacional (ER), que estruturam a base de dados de maneira eficiente. Também foram desenvolvidas diversas Stored Procedures para a manipulação dos dados, assegurando a integridade e a consistência das operações realizadas no sistema. As UDFs foram implementadas para simplificar consultas complexas e promover a reutilização de código, aumentando a modularidade e a manutenção do sistema.

Este relatório apresenta de forma detalhada cada uma dessas etapas, incluindo exemplos de código e explicações sobre as decisões tomadas durante o desenvolvimento do Sistema de Gestão de Eventos.

Ánalise de requisitos

Requisitos Funcionais

Cliente:

Um cliente pode ser apenas uma empresa ou um particular.

Gestão de reservas:

O sistema deve permitir que, apenas os clientes, façam uma reserva para um evento;

Cada reserva deve ser gerida por um gestor de eventos da empresa.

Gestão de funcionários e serviços adicionais:

O gestor de eventos é responsável por gerir a reserva, assim como requisitar eventuais serviços adicionais necessários ao evento e ainda contratar funcionários para o evento.

Associação de Espaço, Evento e Pagamento à Reserva:

Cada reserva deve estar associada a um espaço onde o evento será realizado, a um evento específico e a um pagamento.

A existência de uma reserva exige o pagamento;

Requisitos Não Funcionais

O sistema deve ser fácil de usar e intuitivo para os clientes e administradores;

O sistema deve ser responsivo e compatível com diferentes dispositivos, como computadores, tablets e smartphones;

O sistema deve garantir a segurança e proteção dos dados dos clientes, gestores, funcionários e reservas.

Entidades

Cliente**:** Esta entidade representa os clientes que utilizam o sistema para fazer reservas para eventos.

Manager**:** Representa os gestores de eventos responsáveis por gerir as reservas feitas pelos clientes. Os gestores podem coordenar diferentes aspetos relacionados com os eventos como a contratação de serviços adicionais e de funcionários.

Reserva**:** Esta entidade representa as reservas feitas pelos clientes para eventos específicos. Cada reserva está associada a um cliente, um gestor, um espaço, um evento e um pagamento.

Espaço**:** Representa os espaços disponíveis para reserva, onde os eventos serão realizados.

Pagamento**:** Esta entidade regista os pagamentos feitos pelos clientes para confirmar uma reserva. Pode incluir informações como método de pagamento, data de pagamento e custo.

Evento**:** Representa os eventos que serão realizados nos espaços reservados. Cada evento está associado a uma reserva e pode incluir detalhes como data, hora, tipo de evento, descrição.

Funcionário**:** Esta entidade representa os funcionários que podem ser atribuídos a eventos para fornecer serviços adicionais.

Serviço adicional**:** Esta entidade representa os serviços adicionais que podem ser solicitados pelos clientes e contratados pelo gestor.

Diagrama Entidade-Relacionamento (DER)

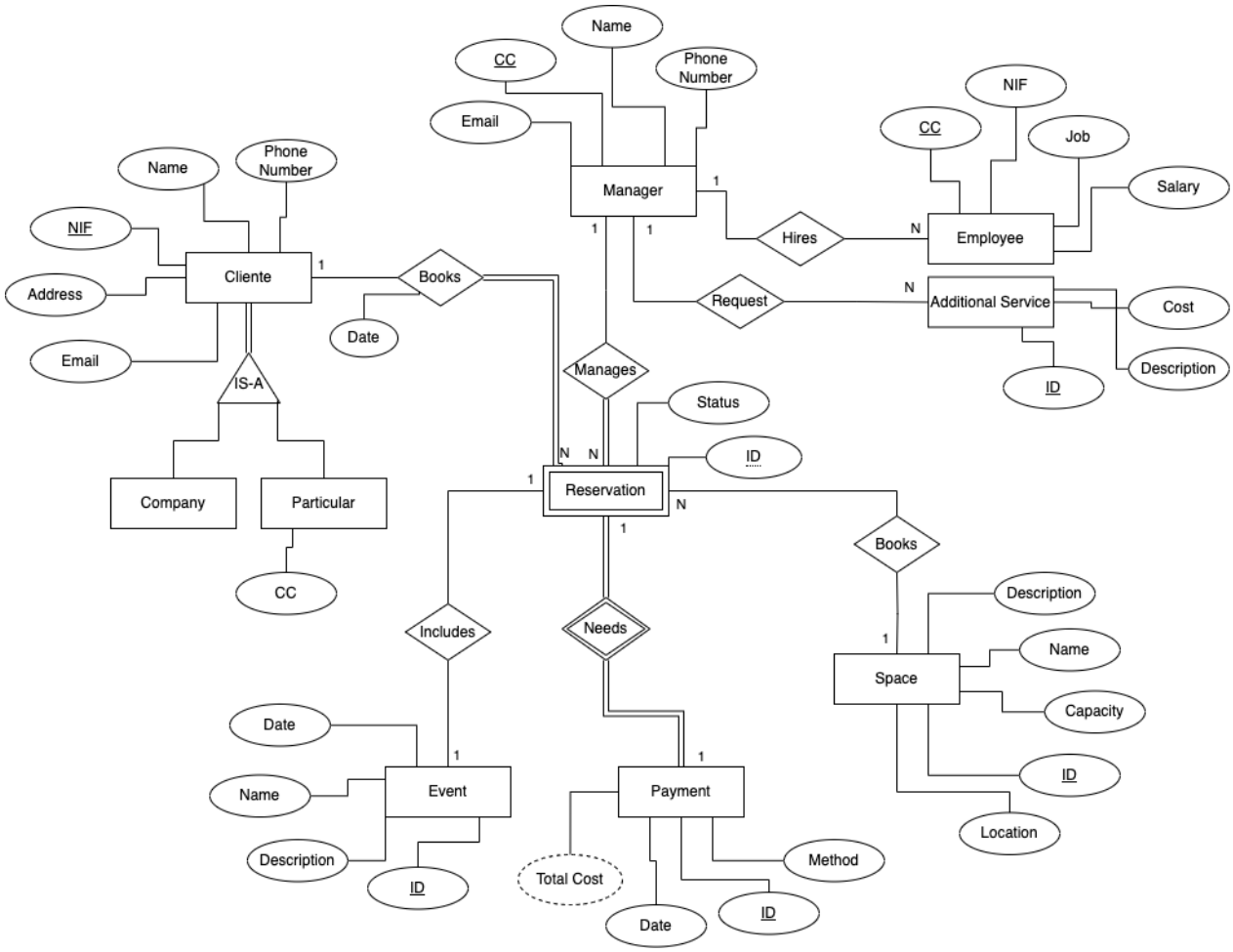


Figura 1 | Digrama Entidade-Relacionamento (DER)

Esquema Relacional (ER)

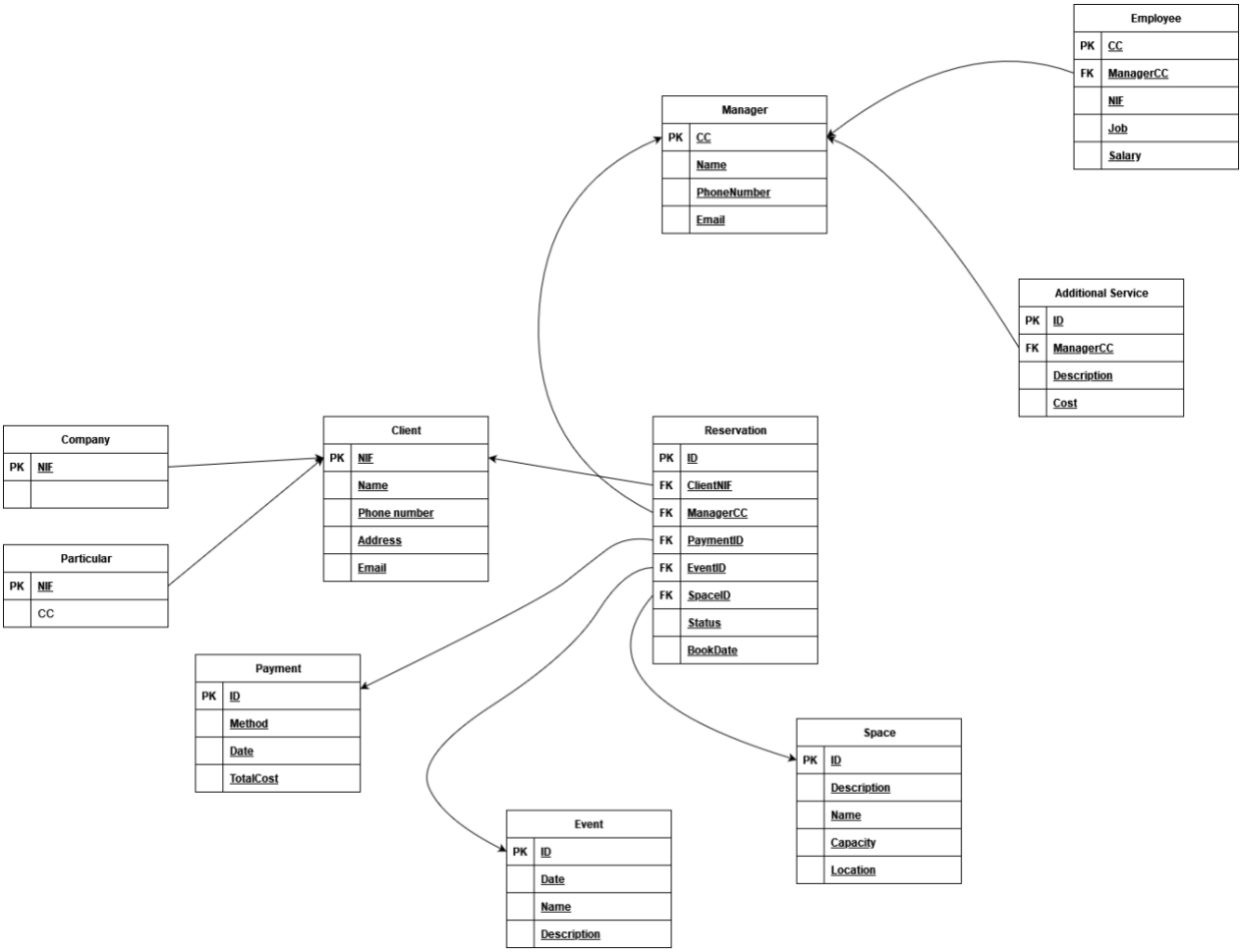


Figura 2 | Esquema Relacional (ER)

QUERIES SQL

Data Definition Language (DDL)

Ao definir a estrutura dos dados dos objetos que são guardados na base de dados, o DDL assume-se como uma parte fulcral da linguagem SQL.

O código que permite gerar as tabelas necessárias à implementação do nosso projeto segue apresentado abaixo.

```
--CREATE SCHEMA SGE;
--GO
CREATE TABLE SGE.Particular(
    CC INTEGER NOT NULL,
    ParticularNIF INTEGER,
    PRIMARY KEY (CC)
);

CREATE TABLE SGE.Empresa(
    EmpresaNIF INTEGER,
    PRIMARY KEY (EmpresaNIF)
);

CREATE TABLE SGE.Manager(
    CC INTEGER NOT NULL,
    Nome VARCHAR(255) NOT NULL,
    NumTelef INTEGER NOT NULL,
    Email VARCHAR(255) NOT NULL,
    PRIMARY KEY(CC)
);

CREATE TABLE SGE.Empregado(
    CC INTEGER NOT NULL,
    ManagerCC INTEGER NOT NULL,
    NIF INTEGER NOT NULL CHECK(LEN(NIF)=9),
    Trabalho VARCHAR(255) NOT NULL,
    Salario INTEGER NOT NULL CHECK(Salario >= 0),
    PRIMARY KEY(CC)
```

```
);

CREATE TABLE SGE.ServicoAdicional(
    ID INTEGER IDENTITY(1,1),
    ManagerCC INTEGER NOT NULL,
    Descricao VARCHAR(255),
    Custo INTEGER NOT NULL CHECK(Custo >= 0),
    PRIMARY KEY(ID)
);


CREATE TABLE SGE.Evento(
    ID INTEGER IDENTITY(1,1),
    DataEvento DATE NOT NULL,
    Nome VARCHAR(255) NOT NULL,
    Descricao VARCHAR(255),
    PRIMARY KEY(ID)
);

CREATE TABLE SGE.Espaco(
    ID INTEGER IDENTITY(1,1),
    Descricao VARCHAR(255),
    Nome VARCHAR(255) NOT NULL,
    Lotacao INTEGER NOT NULL CHECK(Lotacao > 0),
    Localizacao VARCHAR(255) NOT NULL,
    PRIMARY KEY(ID)
);

CREATE TABLE SGE.Pagamento(
    ID INTEGER IDENTITY(1,1),
    DataPagamento DATE NOT NULL,
    MetodoPagamento VARCHAR(255) NOT NULL,
    MontanteTotal INTEGER NOT NULL CHECK(MontanteTotal > 0)
    PRIMARY KEY(ID)
)

CREATE TABLE SGE.Reserva(
    ID INTEGER IDENTITY(1,1),
    ClienteNIF INTEGER NOT NULL,
    ManagerCC INTEGER NOT NULL,
```

```
    EventoID INTEGER NOT NULL UNIQUE,
    SpaceID INTEGER NOT NULL UNIQUE,
    PagamentoID INTEGER NOT NULL,
    ReservaStatus VARCHAR(255) NOT NULL,
    PRIMARY KEY(ID)
);


-- Adicionar constraint de chave estrangeira para a tabela GE.Particular
ALTER TABLE SGE.Particular
ADD CONSTRAINT ParticularNIF_FK FOREIGN KEY (ParticularNIF) REFERENCES SGE.Cliente(NIF)
    ON DELETE CASCADE;

-- Adicionar constraint de chave estrangeira para a tabela GE.Empresa
ALTER TABLE SGE.Empresa
ADD CONSTRAINT EmpresaNIF_FK FOREIGN KEY (EmpresaNIF) REFERENCES SGE.Cliente(NIF)
    ON DELETE CASCADE;

-- Adicionar constraint de chave estrangeira para a tabela GE.Empregado
ALTER TABLE SGE.Empregado
ADD CONSTRAINT ManagerCC_FK FOREIGN KEY (ManagerCC) REFERENCES SGE.Manager(CC)
    ON DELETE CASCADE;

-- Adicionar constraint de chave estrangeira para a tabela GE.ServicoAdicional
ALTER TABLE SGE.ServicoAdicional
ADD CONSTRAINT ServicoAdicional_ManagerCC_FK FOREIGN KEY (ManagerCC) REFERENCES
SGE.Manager(CC)
    ON DELETE CASCADE;

-- Adicionar constraints de chave estrangeira para a tabela GE.Reserva
ALTER TABLE SGE.Reserva
ADD CONSTRAINT Reserva_ClienteNIF_FK FOREIGN KEY (ClienteNIF) REFERENCES
SGE.Cliente(NIF)
    ON DELETE CASCADE;
ALTER TABLE SGE.Reserva
ADD CONSTRAINT Reserva_ManagerCC_FK FOREIGN KEY (ManagerCC) REFERENCES
SGE.Manager(CC)
    ON DELETE CASCADE;
ALTER TABLE SGE.Reserva
ADD CONSTRAINT Reserva_EventoID_FK FOREIGN KEY (EventoID) REFERENCES SGE.Evento(ID)
```

```
    ON DELETE CASCADE;
ALTER TABLE SGE.Reserva
ADD CONSTRAINT Reserva_SpaceID_FK FOREIGN KEY (SpaceID) REFERENCES SGE.Espaco(ID)
    ON DELETE CASCADE;
ALTER TABLE SGE.Reserva
ADD CONSTRAINT Reserva_PagamentoID_FK FOREIGN KEY (PagamentoID) REFERENCES
SGE.Pagamento(ID)
    ON DELETE CASCADE;
ALTER TABLE SGE.Reserva
```

Data Manipulation Language (DML)

A Data Manipulation Language (DML) é uma parte essencial da linguagem SQL e desempenha um papel crucial na manipulação dos dados dentro de uma base de dados. Com DML, pudemos inserir, atualizar, remover e consultar dados nas tabelas criadas. No nosso contexto, Sistema de Gestão de Eventos, utilizamos DML para inserir diversos dados nas tabelas, de forma a garantir que a base de dados tenha as informações necessárias para suportar as funcionalidades do sistema.

Abaixo são apresentados exemplos da inserção dos dados.

```
-- Inserir dados na tabela SGE.Cliente
INSERT INTO SGE.Cliente (NIF, Nome, NumTelef, Endereco, Email, ReservaID)
VALUES
(123456789, 'João Silva', 912345678, 'Rua A, 123', 'joao.silva@example.com', NULL),
(987654381, 'Maria Oliveira', 987654391, 'Avenida B, 456', 'maria.oliveira@example.com', NULL),
(456123789, 'Carlos Sousa', 956123789, 'Travessa C, 789', 'carlos.sousa@example.com', NULL),
(153456789, 'João Silva', 912345648, 'Rua A, 140', 'joao.gomes@example.com', NULL);

INSERT INTO SGE.Particular (CC, ParticularNIF)
VALUES
(1111, 123456789),
(2222, 987654381),
(1234, 153456789);
GO

-- Inserir dados de teste na tabela SGE.Empresa
INSERT INTO SGE.Empresa (EmpresaNIF)
VALUES
(456123789),
(153456789);
GO

-- Inserir dados na tabela SGE.Manager
INSERT INTO SGE.Manager (CC, Nome, NumTelef, Email, ReservaID)
VALUES
(1001, 'Ana Costa', 911234567, 'ana.costa@example.com', NULL),
(1002, 'Pedro Martins', 922345678, 'pedro.martins@example.com', NULL);

INSERT INTO SGE.Pagamento (DataPagamento, MetodoPagamento, MontanteTotal)
VALUES
```

```
('2024-05-20', 'Cartão de Crédito', 150),
('2024-05-21', 'Transferência Bancária', 300),
('2024-05-05', 'Cartão de Crédito', 200),
('2024-05-01', 'Cartão de Crédito', 220),
('2004-12-03', 'Cartão de Crédito', 55);

-- Inserir dados na tabela SGE.Evento
INSERT INTO SGE.Evento (DataEvento, Nome, Descricao)
VALUES
('2024-05-10', 'Evento A', 'Descricao A'),
('2024-05-11', 'Evento B', 'Descricao B'),
('2024-05-12', 'Evento C', 'Descricao C'),
('2024-05-13', 'Evento D', 'Descricao D'),
('2025-05-15', 'Evento E', 'Descricao D');
GO

-- Inserir dados na tabela SGE.Espaco
INSERT INTO SGE.Espaco (Descricao, Nome, Lotacao, Localizacao)
VALUES
('Espaco A', 'Sala 1', 100, 'Local 1'),
('Espaco B', 'Sala 2', 200, 'Local 2'),
('Espaco C', 'Sala 3', 150, 'Local 3'),
('Espaco D', 'Sala 4', 250, 'Local 4'),
('Espaco E', 'Sala 1', 1000, 'Local 5');
GO

-- Inserir dados na tabela SGE.Reserva usando os IDs das tabelas anteriores
-- Presume-se que os IDs de Evento, Espaco e Pagamento são 1 e 2 conforme as inserções acima
INSERT INTO SGE.Reserva (ClienteNIF, ManagerCC, EventoID, SpaceID, PagamentoID, ReservaStatus,
ReservaData)
VALUES
(123456789, 1001, 1, 1, 1, 'Confirmada', '2024-05-10'),
(456123789, 1002, 2, 2, 2, 'Pendente', '2024-05-11'),
(987654381, 1001, 3, 3, 3, 'Pendente', '2024-05-04'),
(153456789, 1002, 4, 4, 4, 'Pendente', '2024-05-01'),
(456123789, 1000, 5, 5, 5, 'Confirmada', '2004-12-03');
```

User Defined Functions (UDFs)

As User Defined Functions (UDFs) são uma parte fundamental do SQL ao permitirem a extensão da funcionalidade da linguagem através da criação de funções personalizadas. As UDFs permitem assim simplificar o desenvolvimento de consultas complexas e na reutilização de código. Ajudam também a encapsular a lógica e permitem consultas repetitivas em funções reutilizáveis, tornando o código SQL mais modular e fácil de manter.

No nosso contexto, Sistema de Gestão de Eventos, as UDFs foram utilizadas principalmente para a filtragem e recuperação de dados. Essas funções foram projetadas para fornecer resultados específicos baseados em diferentes critérios de pesquisa, sem a necessidade de alterar, adicionar ou excluir dados da base de dados.

Abaixo, apresentamos o código das diferentes UDFs implementadas, juntamente com comentários que explicam suas finalidades e os cenários de uso.

```
-- Função para filtrar Clientes por Reserva, Nome e NIF:

DROP FUNCTION IF EXISTS filtrarCLientesPorReservaENifENome;
GO

CREATE FUNCTION filtrarCLientesPorReservaENifENome(@reservaID INTEGER, @nif INTEGER,
@nome varchar(100)) RETURNS TABLE
AS
RETURN (
   SELECT C.NIF, C.Nome, C.NumTelef, C.Endereco, C.Email, C.ReservaID
   FROM SGE.Cliente as C JOIN SGE.Reserva as R ON C.ReservaID = R.ID
   WHERE
      (R.ID = @reservaID OR @reservaID IS NULL) AND
      (C.Nome = @nome OR @nome IS NULL) AND
      (C.NIF = @NIF OR @NIF IS NULL)
);
GO



-- Função para filtrar Manager por Reserva e CC:

DROP FUNCTION IF EXISTS filtrarManagerPorReservaENomeECartaoCidadao;
GO
```

```
CREATE FUNCTION filtrarManagerPorReservaENomeECartaoCidadao(@reservaID INTEGER, @Nome
VARCHAR(100), @CC INTEGER) RETURNS TABLE
AS
RETURN (
    SELECT M.CC, M.Nome, M.NumTelef, M.Email, M.ReservaID
    FROM SGE.Manager as M JOIN SGE.Reserva as R ON M.ReservaID = R.ID
    WHERE
        (R.ID = @reservaID OR @reservaID IS NULL) AND
        (M.Nome = @Nome OR @Nome IS NULL) AND
        (M.CC = @CC OR @CC IS NULL)
);
GO


-- Função para filtrar Reserva por ManagerCC, ReservaID, EspacoID, EventoID, PagamentoID:

DROP FUNCTION IF EXISTS filtrarReservaPorIdEManagerEEspacoEEventoEPagamento;
GO


CREATE FUNCTION filtrarReservaPorIdEManagerEEspacoEEventoEPagamento(@reservaID INTEGER,
@managerCC INTEGER, @espacoID INTEGER, @eventoID INTEGER, @pagamentoID INTEGER)
RETURNS TABLE
AS
RETURN (
    SELECT R.ID, R.ClienteNIF, R.ManagerCC, R.EventoID, R.SpaceID, R.PagamentoID,
R.ReservaStatus, R.ReservaData
    FROM SGE.Reserva as R
    JOIN SGE.Manager as M ON R.ManagerCC = M.CC
    JOIN SGE.Evento as E ON R.EventoID = E.ID
    JOIN SGE.Espaco as S ON R.SpaceID = S.ID
    JOIN SGE.Pagamento as P ON R.PagamentoID = P.ID
    WHERE
        (M.CC = @managerCC OR @managerCC IS NULL) AND
        (R.ID = @reservaID OR @reservaID IS NULL) AND
        (E.ID = @espacoID OR @espacoID IS NULL) AND
        (S.ID = @eventoID OR @eventoID IS NULL) AND
        (P.ID = @pagamentoID OR @pagamentoID IS NULL)
```

```
);
GO
```

Views

As Views em SQL permitem criar consultas personalizadas e armazená-las como objetos virtuais numa base de dados. Estas permitem a simplificação do acesso aos dados e a abstração de complexidade nas consultas SQL. Facilitam também a apresentação dos dados, permitindo que os utilizadores obtenham informações específicas sem a necessidade de escrever consultas complexas repetidamente.

No nosso contexto, do Sistema de Gestão de Eventos, as Views foram amplamente utilizadas para simplificar a visualização de dados na interface do utilizador, bem como para apoiar as User Defined Functions (UDFs) e as Stored Procedures. As Views ajudaram a organizar os dados de maneira eficiente ao criar vistas específicas e filtradas das tabelas relacionadas com os clientes, reservas, managers, espaços, eventos e pagamentos.

A seguir, apresentamos o código das Views implementadas, detalhando as suas finalidades e os critérios utilizados para a seleção e apresentação dos dados.

```
-- View para Cliente e Particular
DROP VIEW IF EXISTS SGE.ClienteParticular
GO

CREATE VIEW SGE.ClienteParticular AS
   SELECT C.NIF, C.Nome, C.NumTelef, C.Endereco, C.Email, C.ReservaID, P.CC
   FROM (SGE.Particular AS P JOIN SGE.Cliente AS C ON C.NIF = P.ParticularNIF);
GO
```

```
-- View para Cliente e Empresa
DROP VIEW IF EXISTS SGE.ClienteEmpresa;
GO

CREATE VIEW SGE.ClienteEmpresa AS
   SELECT C.NIF, C.Nome, C.NumTelef, C.Endereco, C.Email, C.ReservaID, E.EmpresaNIF
   FROM SGE.Empresa AS E
   JOIN SGE.Cliente AS C ON C.NIF = E.EmpresaNIF;
GO
```

```sql
-- View para Espaço da Reserva
DROP VIEW IF EXISTS SGE.ReservaEspaco;
GO

CREATE VIEW SGE.ReservaEspaco AS
    SELECT
        R.ID AS ReservaID, R.ReservaData, R.SpaceID, E.Nome AS EspacoNome,
        E.Descricao AS EspacoDescricao,
        E.Lotacao AS EspacoLotacao,
        E.Localizacao AS EspacoLocalizacao
    FROM SGE.Reserva AS R
    JOIN SGE.Espaco AS E ON R.SpaceID = E.ID;
GO


-- View para Evento da Reserva
DROP VIEW IF EXISTS SGE.ReservaEvento;
GO

CREATE VIEW SGE.ReservaEvento AS
    SELECT
        R.ID AS ReservaID, R.ReservaData, R.EventoID,
        E.Nome AS EventoNome,
        E.DataEvento AS EventoData,
        E.Descricao AS EventoDescricao
    FROM SGE.Reserva AS R
    JOIN SGE.Evento AS E ON R.EventoID = E.ID;
GO


-- View para Pagamento da Reserva
DROP VIEW IF EXISTS SGE.ReservaPagamento;
GO

CREATE VIEW SGE.ReservaPagamento AS
    SELECT
```

```
    R.ID AS ReservaID, R.ReservaData, R.PagamentoID, P.DataPagamento, P.MetodoPagamento,
P.MontanteTotal
  FROM SGE.Reserva AS R
  JOIN SGE.Pagamento AS P ON R.PagamentoID = P.ID;
GO



-- View para histórico de Reservas (cujos Eventos já foram realizados)
DROP VIEW IF EXISTS SGE.ReservasEventosRealizados;
GO

CREATE VIEW SGE.ReservasEventosRealizados AS
  SELECT
    R.ID AS ReservaID, E.DataEvento, E.Nome AS EventoNome, R.ClienteNIF, R.ManagerCC,
R.SpaceID, R.PagamentoID
  FROM SGE.Reserva AS R
  JOIN SGE.Evento AS E ON R.EventoID = E.ID
  WHERE E.DataEvento < GETDATE();  -- Filtrar eventos que já aconteceram
GO



-- View para Eventos futuros
DROP VIEW IF EXISTS SGE.ReservasEventosFuturos;
GO

CREATE VIEW SGE.ReservasEventosFuturos AS
  SELECT
    R.ID AS ReservaID, E.DataEvento, E.Nome AS EventoNome, R.EventoID, R.ClienteNIF,
R.ManagerCC, R.SpaceID, R.PagamentoID
  FROM SGE.Reserva AS R
  JOIN SGE.Evento AS E ON R.EventoID = E.ID
  WHERE E.DataEvento > GETDATE();
GO
```

Triggers

Os Triggers são objetos de base de dados que são ativos automaticamente em resposta a determinados eventos, como inserção, atualização ou exclusão de dados numa tabela. Eles são essenciais para manter a integridade dos dados, automatizar processos e garantir que certas ações sejam executadas automaticamente sempre que um evento específico ocorre.

Desta forma, para o Sistema de Gestão de Eventos, os Triggers foram implementados para atender necessidades específicas, garantindo a consistência e a correta manutenção dos dados. Embora muitas verificações e regras de integridade tenham sido definidas diretamente nas instruções DDL (Data Definition Language) e nas Stored Procedures, os Triggers foram utilizados para automatizar ações que seriam repetitivas ou complexas de gerir manualmente.

A seguir, apresentamos os Triggers implementados e suas finalidades.

```
--Apagar o Cliente das tabelas particular ou empresa quando Cliente  apagado

CREATE TRIGGER DeleteClienteTrig
ON SGE.Cliente
AFTER DELETE
AS
BEGIN
  IF EXISTS (SELECT 1 FROM SGE.Particular WHERE ParticularNIF IN (SELECT NIF FROM deleted))
  BEGIN
    DELETE FROM SGE.Particular
    WHERE ParticularNIF IN (SELECT NIF FROM deleted);
  END

  IF EXISTS (SELECT 1 FROM SGE.Empresa WHERE EmpresaNIF IN (SELECT NIF FROM deleted))
  BEGIN
    DELETE FROM SGE.Empresa
    WHERE EmpresaNIF IN (SELECT NIF FROM deleted);
  END
END;
```

```
--Incrementar o custo total com o custo do serviço adicional

GO
CREATE TRIGGER ServicoTotal
ON SGE.ServicoAdicional
AFTER INSERT
AS
BEGIN
   UPDATE p
   SET MontanteTotal = MontanteTotal + i.Custo
   FROM SGE.Pagamento p
   INNER JOIN SGE.Reserva r ON p.ID = r.PagamentoID
   INNER JOIN inserted i ON r.ManagerCC = i.ManagerCC;
END;



--Adicionar o salario do empregado ao custo total

GO
CREATE TRIGGER EmpregadoTotal
ON SGE.Empregado
AFTER INSERT
AS
BEGIN
   UPDATE p
   SET MontanteTotal = MontanteTotal + i.Salario
   FROM SGE.Pagamento p
   INNER JOIN SGE.Reserva r ON p.ID = r.PagamentoID
   INNER JOIN inserted i ON r.ManagerCC = i.ManagerCC;
END;
```

Stored Procedures

As Stored Procedures assumem relevância em bases de dados ao permitirem que código SQL seja armazenado e executado de forma estruturada e reutilizável. Garantem ainda atomicidade em transações complexas ao assegurar que todas as operações sejam bem-sucedidas ou nenhuma delas é executada. Além disso, as Stored Procedures contribuem para a manutenção da integridade e segurança dos dados, encapsulando a lógica do negócio diretamente no banco de dados.

No nosso contexto do Sistema de Gestão de Eventos, as Stored Procedures foram implementadas principalmente para a manipulação dos dados, facilitando a adição, alteração, exclusão e a consulta de informações nas tabelas. Cada Stored Procedure foi criada com uma finalidade específica, garantindo que operações críticas sejam realizadas de maneira eficiente e segura.

Abaixo, são apresentados os códigos das Stored Procedures implementadas acompanhados de um comentário explicativo sobre a sua finalidade.

```
----------REMOVER CLIENTE----------
GO
CREATE PROCEDURE RemoveCliente
   @NIF INTEGER
AS
BEGIN

   BEGIN TRANSACTION
   DELETE FROM SGE.Cliente
   WHERE NIF = @NIF

   IF @@ROWCOUNT = 0
   BEGIN
      ROLLBACK TRANSACTION
      PRINT 'ERRO! Cliente n◆o encontrado';
   END
   ELSE
   BEGIN
      COMMIT TRANSACTION
      PRINT 'Cliente removido com sucesso';
   END
END;
```

Esta Stored Procedure remove um cliente da tabela Cliente com base no seu NIF. Caso o cliente não seja encontrado, a transação é revertida e uma mensagem de erro é exibida.

```
----------ADICIONAR CLIENTE----------
GO
CREATE PROCEDURE AdicionarCliente
   @NIF INTEGER,
   @Nome VARCHAR(255),
   @NumTelef INTEGER,
   @Endereco VARCHAR(255),
   @Email VARCHAR(255),
   @ReservaID INTEGER,
   @CC INTEGER = NULL
AS
BEGIN
   BEGIN TRANSACTION

   IF @CC IS NULL
   BEGIN
      INSERT INTO SGE.Cliente(NIF, Nome, NumTelef, Endereco, ReservaID, Email)
      VALUES (@NIF, @Nome, @NumTelef, @Endereco, @ReservaID, @Email)

      INSERT INTO SGE.Empresa (EmpresaNIF)
      VALUES (@NIF)
   END
   ELSE
   BEGIN
      INSERT INTO SGE.Cliente(NIF, Nome, NumTelef, Endereco, ReservaID, Email)
      VALUES (@NIF, @Nome, @NumTelef, @Endereco, @ReservaID, @Email)

      INSERT INTO SGE.Particular(CC, ParticularNIF)
      VALUES(@CC, @NIF)
   END

   BEGIN
   COMMIT TRANSACTION
   END
END
```

Esta Stored Procedure adiciona um novo cliente à tabela Cliente. Dependendo se o cliente é uma empresa ou um particular, o registo correspondente é inserido nas tabelas Empresa ou Particular.

```
----------ADICIONAR RESERVA----------
GO
CREATE PROCEDURE AdicionarReserva
   @ID INTEGER,
   @ClientNIF INTEGER,
   @ManagerCC INTEGER,
   --@PaymentID INTEGER,
   @EventID INTEGER,
   @SpaceID INTEGER,
   @Status VARCHAR(255),
   @BookDate DATE

AS
BEGIN
   BEGIN TRANSACTION

   BEGIN
      INSERT INTO SGE.Reserva(ID, ClienteNIF, ManagerCC, EventoID, SpaceID, ReservaStatus,
ReservaData)
      VALUES (@ID, @ClientNIF, @ManagerCC, @EventID, @SpaceID, @Status, @BookDate)
   END

   BEGIN
   COMMIT TRANSACTION
   END
END
```

Esta Stored Procedure adiciona uma nova reserva à tabela Reserva. Todos os detalhes da reserva, como o NIF do cliente, o CC do manager, o ID do evento, o ID do espaço, o status da reserva e a data da reserva são especificados como parâmetros.

```
----------REMOVER RESERVA----------
GO
CREATE PROCEDURE RemoveReserva
   @ID INTEGER
AS
BEGIN

   BEGIN TRANSACTION
   DELETE FROM SGE.Reserva
   WHERE ID = @ID

   IF @@ROWCOUNT = 0
   BEGIN
      ROLLBACK TRANSACTION
      PRINT 'ERRO! Reserva n◆o encontrada';
   END
   ELSE
   BEGIN
      COMMIT TRANSACTION
      PRINT 'Reserva Removida com sucesso';
   END
END;
```

Esta Stored Procedure remove uma reserva da tabela Reserva com base no seu ID. Caso a reserva não seja encontrada, a transação é revertida e uma mensagem de erro é exibida.

```
----------EDITAR RESERVA----------
GO
CREATE PROCEDURE EditarReserva
   @ID INTEGER,
   @ClientNIF INTEGER,
   @ManagerCC INTEGER,
   --@PaymentID INTEGER,
   @EventID INTEGER,
   @SpaceID INTEGER,
   @Status VARCHAR(255),
   @BookDate DATE
AS
BEGIN
```

```
    BEGIN TRANSACTION;

    UPDATE SGE.Reserva
    SET
        ClienteNIF = COALESCE(@ClientNIF, ClienteNIF),
        ManagerCC = COALESCE(@ManagerCC, ManagerCC),
        EventoID = COALESCE(@EventID, EventoID),
        SpaceID = COALESCE(@SpaceID, SpaceID),
        ReservaStatus = COALESCE(@Status, ReservaStatus),
        ReservaData = COALESCE(@BookDate, ReservaData)

    WHERE ID = @ID;

    IF @@ROWCOUNT = 0
    BEGIN
        ROLLBACK TRANSACTION;
        PRINT 'ERRO! Reserva n�o encontrada';
    END
    ELSE
    BEGIN
        COMMIT TRANSACTION;
        PRINT 'Dados editados com sucesso';
    END
END;
```

Esta Stored Procedure atualiza os detalhes de uma reserva existente na tabela Reserva. Apenas os campos especificados são atualizados, utilizando a função Coalesce para manter os valores atuais quando os novos valores são nulos.

```
----------EDITAR CLIENTE----------
GO
CREATE PROCEDURE EditarCliente
    @NIF INTEGER,
    @Nome VARCHAR(255) = NULL,
    @NumTelef INTEGER = NULL,
    @Endereco VARCHAR(255) = NULL,
    @ReservaID INTEGER,
    @Email VARCHAR(255) = NULL
```

```
AS
BEGIN
   BEGIN TRANSACTION;

   UPDATE SGE.Cliente
   SET
      Nome = COALESCE(@Nome, Nome),
      NumTelef = COALESCE(@NumTelef, NumTelef),
      Endereco = COALESCE(@Endereco, Endereco),
      ReservaID = COALESCE(@ReservaID, ReservaID),
      Email = COALESCE(@Email, Email)
   WHERE NIF = @NIF;

   IF @@ROWCOUNT = 0
   BEGIN
      ROLLBACK TRANSACTION;
      PRINT 'ERRO! Cliente n�o encontrado';
   END
   ELSE
   BEGIN
      COMMIT TRANSACTION;
      PRINT 'Dados editados com sucesso';
   END
END;
```

Esta Stored Procedure atualiza os detalhes de um cliente existente na tabela Cliente. Apenas os campos especificados são atualizados, utilizando a função Coalesce para manter os valores atuais quando os novos valores são nulos.

```
----------PESQUISAR CLIENTE----------
GO
CREATE PROCEDURE PesquisarCliente
   @NIF INTEGER
AS
BEGIN
   SELECT * FROM SGE.Cliente
   WHERE  NIF = @NIF
END;
```

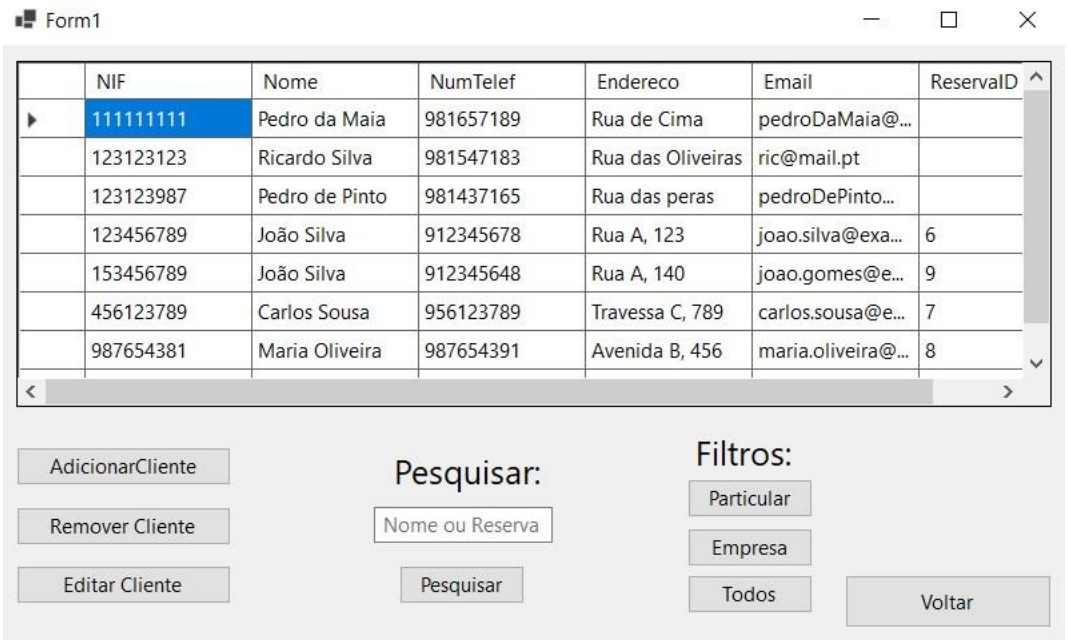
Esta Stored Procedure procura um cliente específico na tabela Cliente com base no seu NIF, retornando todos os dados associados a esse cliente.

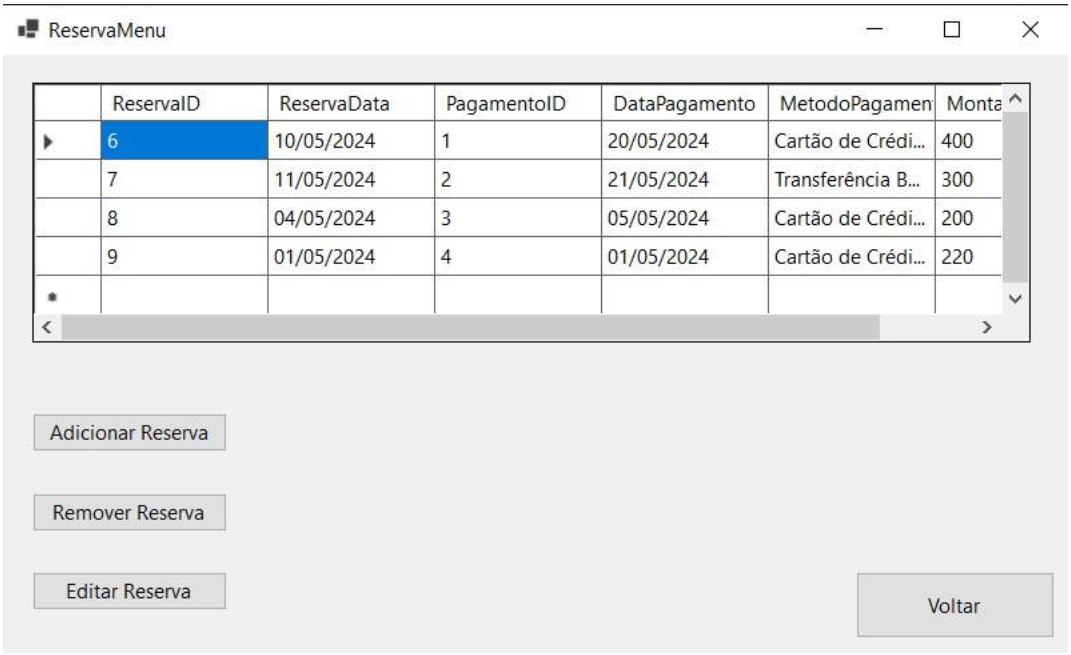
```
----------PESQUISAR RESERVA----------
GO
CREATE PROCEDURE PesquisarReserva
   @ID INTEGER
AS
BEGIN
   SELECT * FROM SGE.Reserva
   WHERE  ID = @ID
END;
```

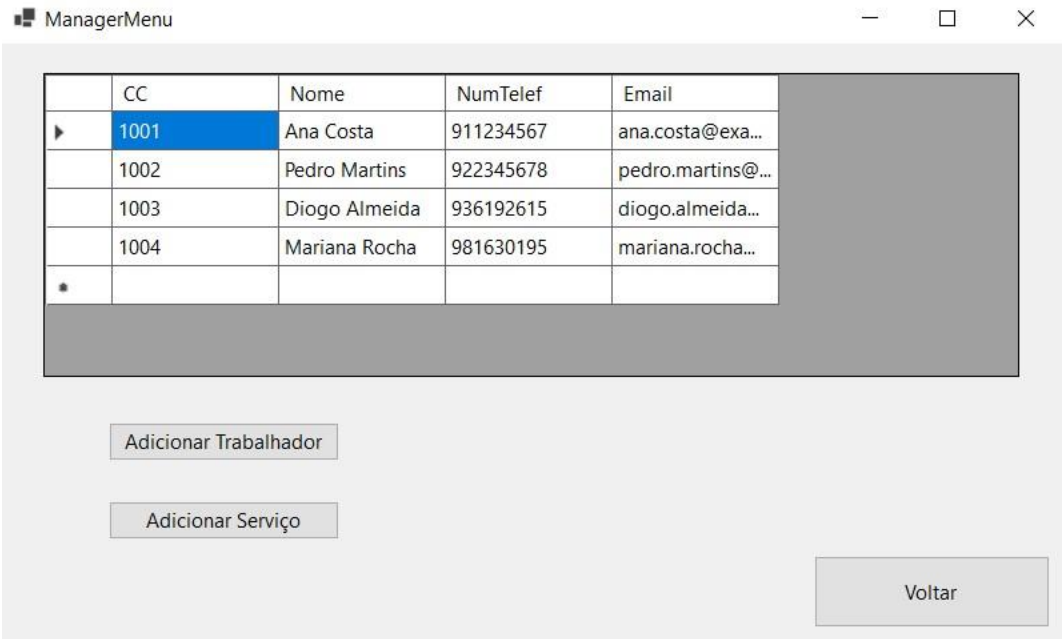
Esta Stored Procedure procura uma reserva específica na tabela Reserva com base no seu ID, retornando todos os dados associados a essa reserva.

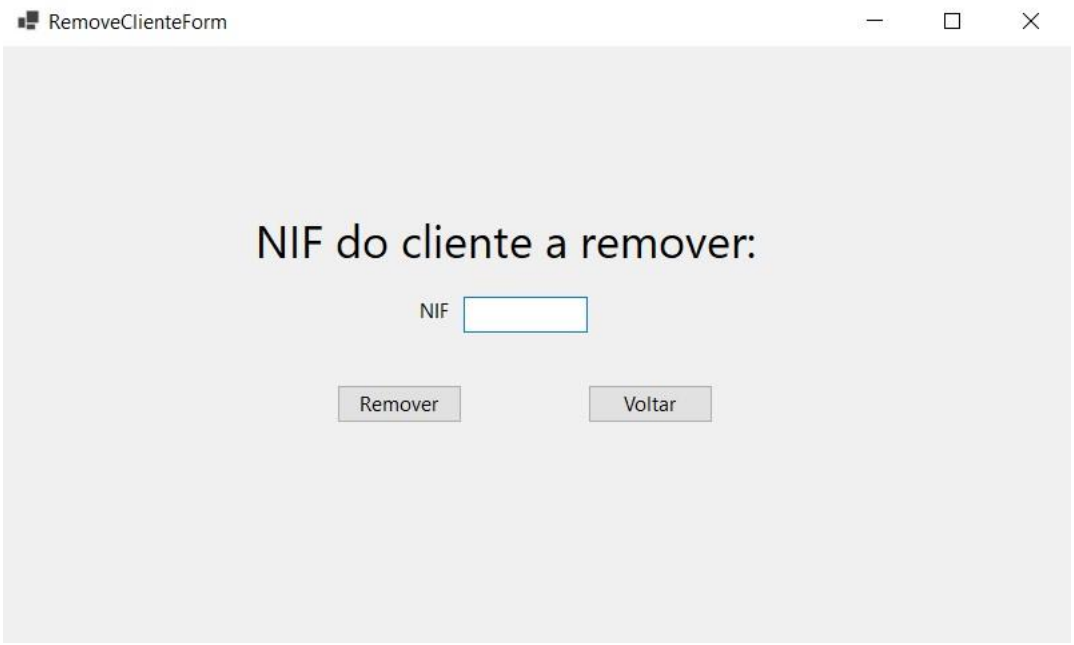
Interface

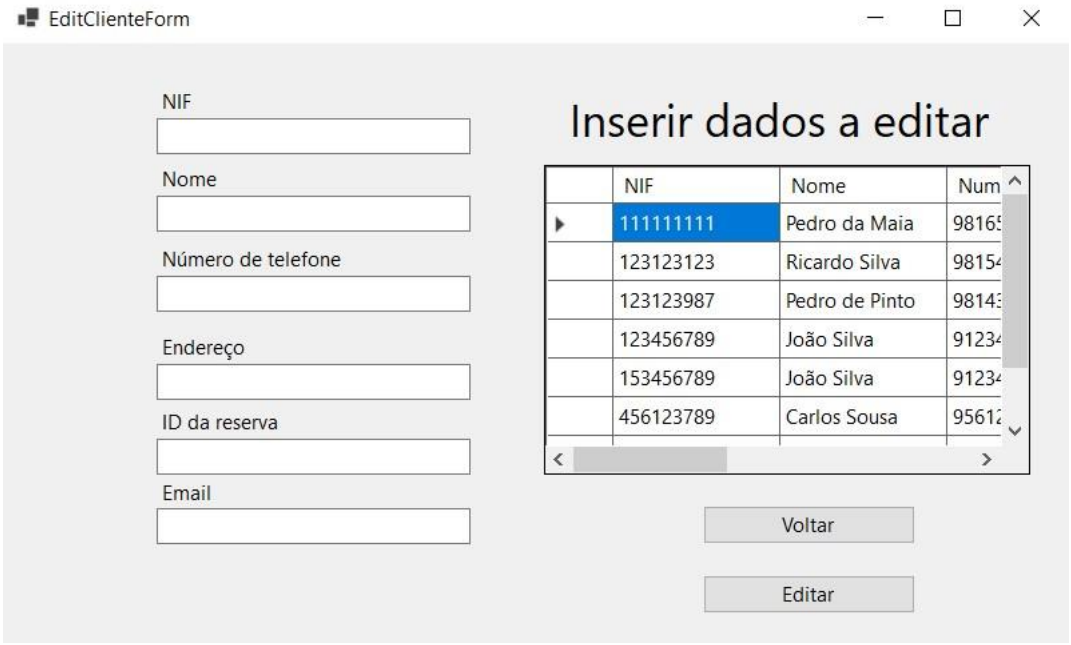

OpeningMenu
Bem Vindo!
Área Cliente
Área Reserva
Área Manager

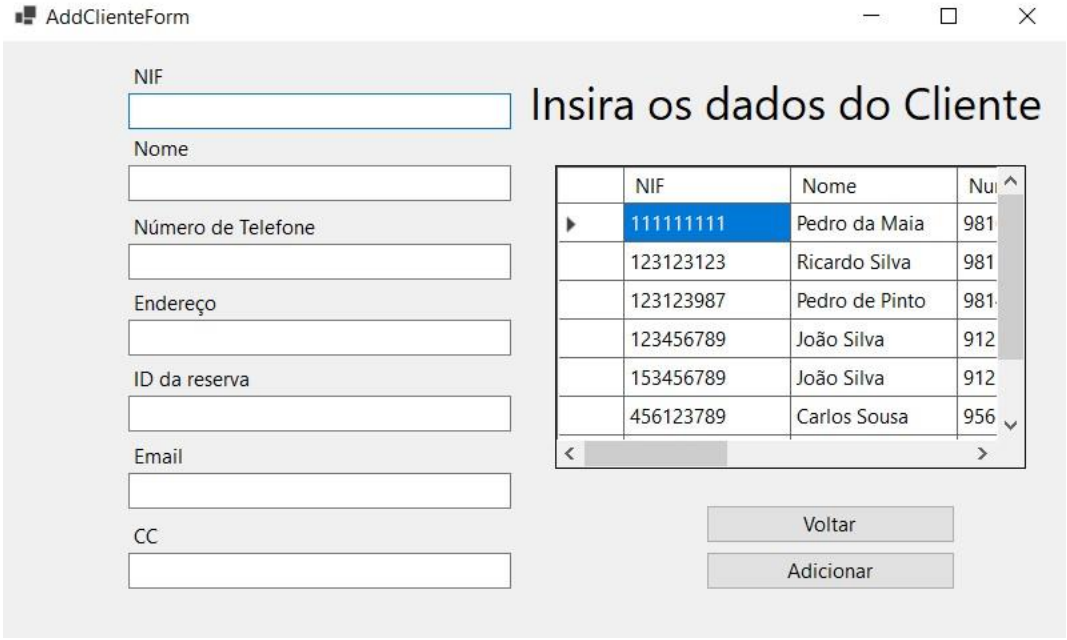

Form1
NIF
Nome
NumTelef
Endereco
Email
ReservaID
111111111
Pedro da Maia
981657189
Rua de Cima
pedroDaMaia@...
123123123
Ricardo Silva
981547183
Rua das Oliveiras
ric@mail.pt
123123987
Pedro de Pinto
981437165
Rua das peras
pedroDePinto...
123456789
João Silva
912345678
Rua A, 123
joao.silva@exa...
6
153456789
João Silva
912345648
Rua A, 140
joao.gomes@e...
9
456123789
Carlos Sousa
956123789
Travessa C, 789
carlos.sousa@e...
7
987654381
Maria Oliveira
987654391
Avenida B, 456
maria.oliveira@...
8
AdicionarCliente
Remover Cliente
Editar Cliente
Pesquisar:
Nome ou Reserva
Pesquisar
Filtros:
Particular
Empresa
Todos
Voltar

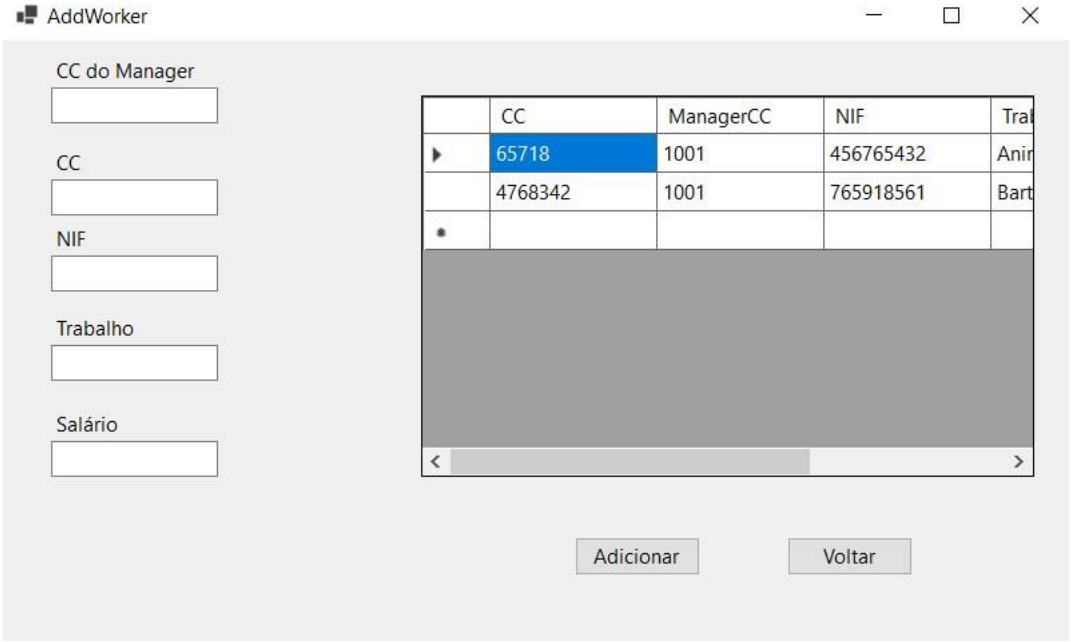

ReservaMenu
ReservaID
ReservaData
PagamentoID
DataPagamento
MetodoPagamen
Monta
6
10/05/2024
1
20/05/2024
Cartão de Crédi...
400
7
11/05/2024
2
21/05/2024
Transferência B...
300
8
04/05/2024
3
05/05/2024
Cartão de Crédi...
200
9
01/05/2024
4
01/05/2024
Cartão de Crédi...
220
Adicionar Reserva
Remover Reserva
Editar Reserva
Voltar


ManagerMenu
CC
Nome
NumTelef
Email
1001
Ana Costa
911234567
ana.costa@exa...
1002
Pedro Martins
922345678
pedro.martins@...
1003
Diogo Almeida
936192615
diogo.almeida...
1004
Mariana Rocha
981630195
mariana.rocha...
Adicionar Trabalhador
Adicionar Serviço
Voltar


RemoveClienteForm
NIF do cliente a remover:
NIF
Remover
Voltar


EditClienteForm
NIF
Nome
Número de telefone
Endereço
ID da reserva
Email
Inserir dados a editar
NIF
Nome
Num
111111111
Pedro da Maia
123123123
Ricardo Silva
123123987
Pedro de Pinto
123456789
João Silva
153456789
João Silva
456123789
Carlos Sousa
Voltar
Editar


AddClienteForm
NIF
Nome
Número de Telefone
Endereço
ID da reserva
Email
CC
Insira os dados do Cliente
NIF
Nome
111111111
Pedro da Maia
123123123
Ricardo Silva
123123987
Pedro de Pinto
123456789
João Silva
153456789
João Silva
456123789
Carlos Sousa
Voltar
Adicionar


AddWorker
CC do Manager
CC
NIF
Trabalho
Salário
CC
ManagerCC
NIF
65718
1001
456765432
4768342
1001
765918561
Adicionar
Voltar

Conclusão

O desenvolvimento do Sistema de Gestão de Eventos demonstrou ser um projeto desafiador e enriquecedor, que envolveu a aplicação de diversos conceitos e técnicas de gestão de bases de dados. Ao longo deste relatório, documentámos todas as fases do projeto, desde a análise de requisitos até à implementação das diferentes funcionalidades, passando pela criação do Diagrama Entidade-Relacionamento (DER) e do Esquema Relacional (ER).

Através da implementação das queries, nomeadamente Stored Procedures, Views e Triggers, conseguimos assegurar a integridade, consistência e eficiência das operações no sistema. As Stored Procedures permitiram a execução organizada e reutilizável de blocos de código SQL, enquanto as Views facilitaram a apresentação dos dados e a criação de consultas personalizadas. Os Triggers foram utilizados para automatizar determinadas ações e garantir a manutenção da integridade dos dados.

O sistema desenvolvido oferece uma plataforma integrada e segura para a gestão de reservas de eventos, onde clientes e gestores podem interagir de maneira eficiente. A análise detalhada dos requisitos e a estrutura eficiente da base de dados foram cruciais para o sucesso do projeto. A utilização de User Defined Functions (UDFs) contribuiu para a simplificação das consultas complexas, promovendo a reutilização de código e melhorando a modularidade do sistema.

Em resumo, o projeto alcançou os objetivos propostos pelo grupo, proporcionando uma solução robusta e eficaz para a gestão de eventos. Este sistema não só facilita o trabalho dos gestores, mas também melhora a experiência dos clientes, garantindo uma organização mais eficiente e integrada dos eventos.