

# Relatório Final

Gestão de Lar de Idosos

Base de Dados

Ano lectivo 2016/2017



## Nursing Home Manager

Feel Like Home

Projecto realizado por:

- André Rodrigues 73152
- Pedro Santos 76532

Docentes:

- Carlos Costa
- Joaquim Sousa
- Sergio Matos

# Índice

<b>Introdução</b>	<b>2</b>
<b>Funcionalidades</b>	<b>4</b>
<b>Diagrama Entidade Relação</b>	<b>5</b>
<b>Modelo Entidade Relação</b>	<b>6</b>
<b>SQL DDL</b>	<b>8</b>
<b>Normalização</b>	<b>12</b>
<b>Índices</b>	<b>12</b>
<b>User-Defined Functions</b>	<b>12</b>
<b>Stored Procedures</b>	<b>18</b>
<b>Triggers</b>	<b>27</b>
<b>Interface Gráfica</b>	<b>29</b>
Patients	29
Adicionar Paciente	30
Editar Paciente	30
Human Resources	32
Appointments	33
Visits	33
Adicionar Visita	34
Manage Nursing Home	35

# Introdução

Este projeto surgiu no âmbito da cadeira Base de Dados e como tal, tem como objectivo a implementação de uma base de dados para gestão de um **Lar de Idoso** e uma interface para manipulação da mesma.

Este projecto será sustentado pelas seguintes premissas:

- O **lar** tem **pacientes** que são caracterizados pelo seu NIF, Nome, Idade, Género, Autorização para sair do edifício e Número de telefone.  
Também tem **funcionários** que são caracterizados pelo Nome, a data em que começaram a trabalhar no estabelecimento, Morada, NIF, Salário e Número de Telefone.
- Os **funcionário** estão associados:
  - A uma instância de turno e por sua vez esta instância tem associadas as faltas e o turno caracterizado por o inicio e fim do turno e em que dia existe o turno, assim com o seu ID.
  - A um tipo para designar o tipo de trabalho que têm e este tipo é caracterizado pela designação e pelo seu ID.
- Os **pacientes** estão associados:
  - A um quarto caracterizado pelo seu número identificador e pela sua capacidade. Este quarto por sua vez está associado a camas, que também têm um número identificador. No quarto têm de estar pessoas do mesmo sexo.
  - A uma cama.
  - Às saídas que efectuam do lar sendo estas caracterizadas pela data de saída e pela data de entrada.
  - Às doenças que lhes foram diagnosticadas, sendo caracterizadas pelo nome e tendo acesso à severidade da mesma.
  - Ao médico que o acompanha. O médico é caracterizado pelo Número de Telefone, Morada, Nome e NIF.
  - A uma consulta que será caracterizada por um identificador único, pela sua data e pela especialidade. A consulta estará associada a um Médico.

- A um horário para saber quando tem de tomar os medicamentos e esse horário está associado aos medicamentos que tem de tomar.
- A um ou mais familiares que são caracterizados pelo NIF, Nome, Morada e Número de telefone
- A visitas que são caracterizadas por um identificador único e pela data da respetiva visita. Por sua vez, as visitas vão estar associadas a visitantes que são caracterizadas pelo NIF, Nome, Morada e Número de telefone podendo ser da família ou não. Para além disso, no caso de ser da família é caracterizado pela grau parentesco e se não for da família qual a relação que mantinha com o paciente.

O **procedimento da realização** foi de acordo com o proposto pelos guiões práticos da cadeira. Começou-se por fazer uma análise de requisitos, reunindo informações sobre as principais entidades e as principais operações sobre estas. Posteriormente foi feito o desenho conceptual, modelo relacional e a sua implementação. Sobre este foram introduzidos dados e feitas algumas queries. Paralelamente foi desenvolvida a interface gráfica do sistema e foram implementados alguns stored procedures/triggers/user definition functions de acordo com as necessidades.

O presente relatório está organizado pela ordem de implementação.

## Funcionalidades

Após uma análise detalhada chegamos à conclusão que a aplicação de gestão do Lar de Idosos deveria permitir realizar as seguintes ações:

- Consultar pacientes;
- Consultar pacientes com autorização para sair do edifício.
- Consultar pacientes sem autorização para sair do edifício.
- Consultar se um certo paciente tem autorização para sair.
- Consultar as doenças que o paciente tem, assim como a severidade da doença.
- Consultar as consultas de cada paciente;
- Consultar as consultas existentes;
- Consultar as consultas que irão existir;
- Consultar as visitas;
- Consultar quem visitou o paciente.
- Consultar os trabalhadores do lar.
- Consultar os dias em que trabalha um certo funcionário.
- Consultar o horário completo de um certo funcionário.
- Consultar que quartos estão vazios.

## Diagrama Entidade Relação

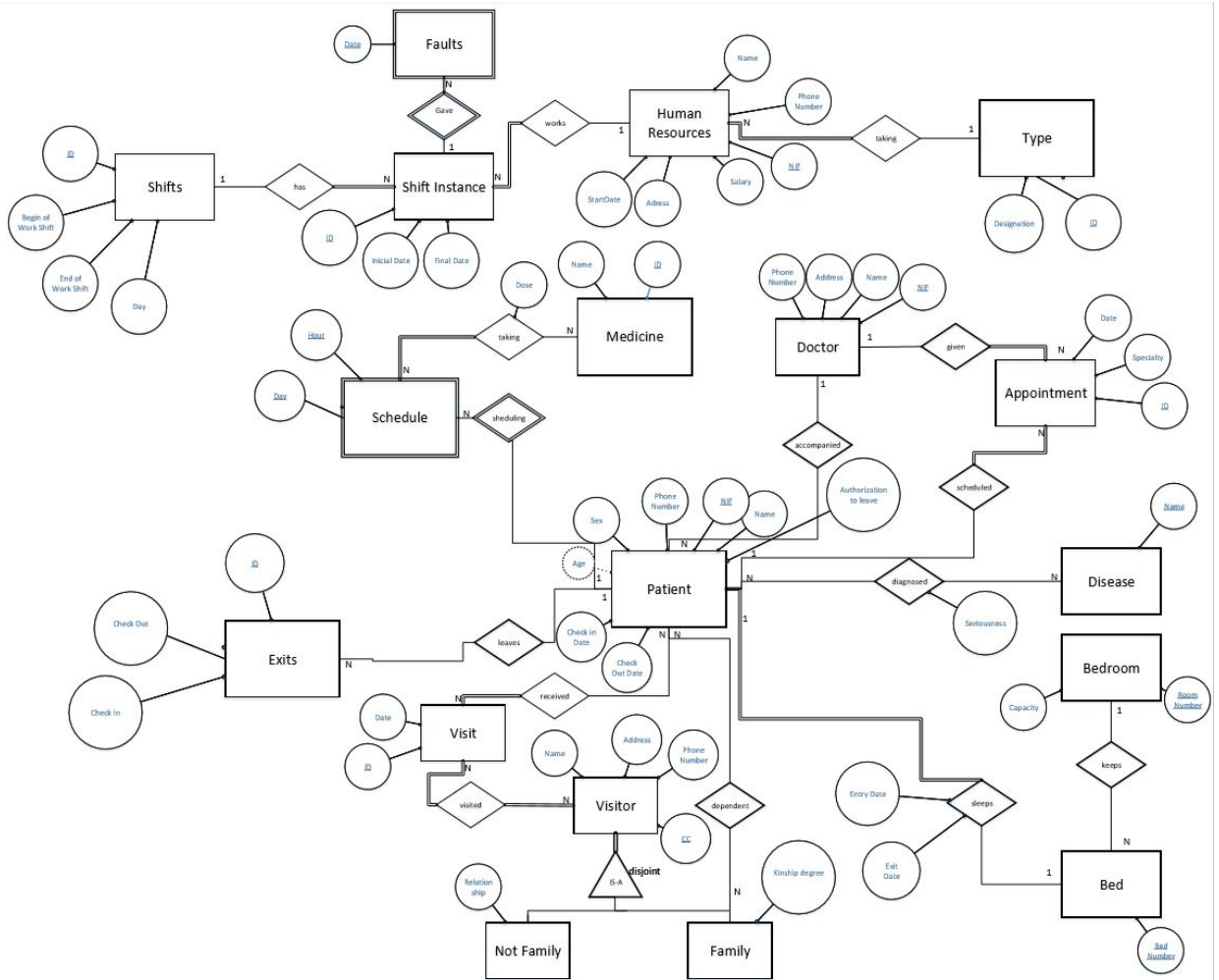
Numa primeira análise foi feita uma seleção das entidades que compõem a nossa base de dados principal (Lar de Idosos):

- **Patient**, paciente
- **Exits**, registo de saídas
- **Leaves**, sair do lar (passear por exemplo)
- **Disease**, tabela que indica as doenças
- **Diagnosed**, diagnóstico do paciente
- **Schedule**, scheduler das tomas dos medicamentos
- **Taking**, toma do medicamento
- **Medicine**, medicamento
- **Bed**, cama onde se situa o paciente
- **Bedroom**, quarto onde se situa a cama do paciente
- **Accompanied**, registo que indica qual o doutor que acompanha certo paciente
- **Doctor**, tabelas de doutores
- **Appointment**, registo de consultas

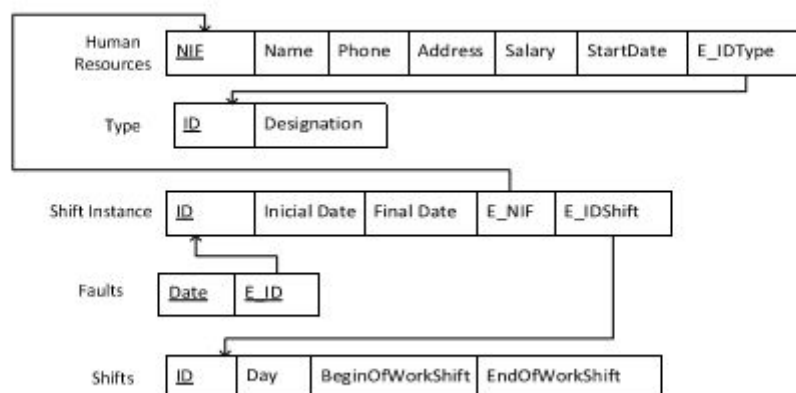
- **Dependent**, pessoa responsável pelo paciente no Lar
- **Family**, familiar do paciente
- **Not Family**, não familiar (amigo por exemplo)
- **Visitor**, pessoa familiar ou não, que pretende visitar um paciente
- **Visited**, registo de visitas efetuadas
- **Visit**, tabela de visitas
- **Received**, registo de visitas recebidas

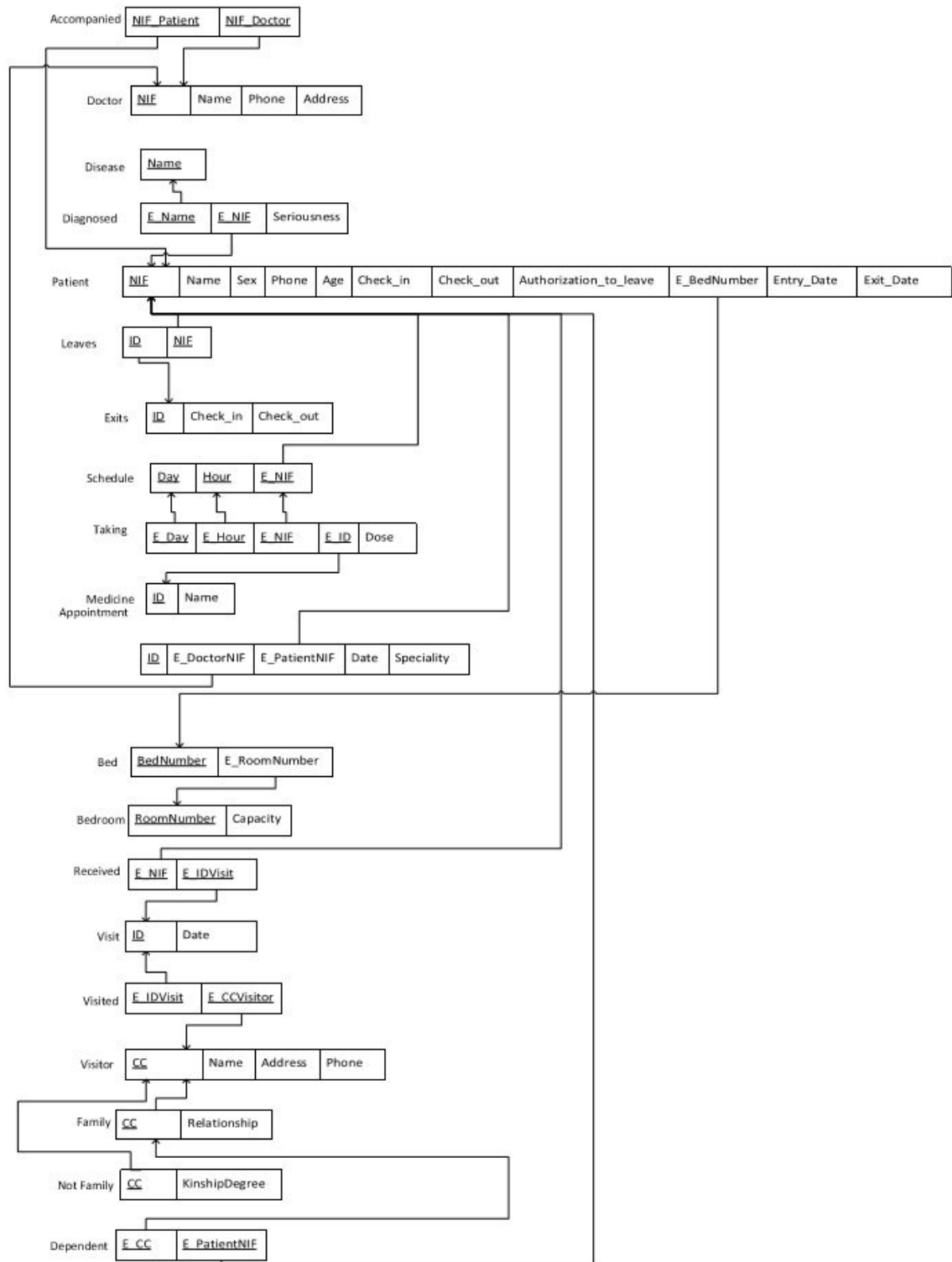
E na base de dados do trabalhadores (Human Resources):

- **Human Resources**, trabalhadores do Lar
- **Type**, tipo de trabalhadores
- **ShiftInstance**, instâncias de turnos
- **Shift**, turnos (horários)
- **Faults**, faltas no horário de trabalho



## Modelo Entidade Relação







# SQL DDL

```
CREATE TABLE LarIdosos.BEDROOM(  
    RoomNumber    INT          NOT NULL    IDENTITY(1, 1),  
    Capacity       INT,  
    CONSTRAINT PKBEDROOM PRIMARY KEY (RoomNumber)  
);  
  
CREATE TABLE LarIdosos.BED(  
    BedNumber      INT          NOT NULL    IDENTITY(1, 1),  
    E_RoomNumber   INT,  
    CONSTRAINT PKBED PRIMARY KEY (BedNumber),  
    CONSTRAINT FKBED FOREIGN KEY (E_RoomNumber) REFERENCES LarIdosos.BEDROOM(RoomNumber)  
);  
  
CREATE TABLE LarIdosos.DOCTOR(  
    NIF            varchar(9)    NOT NULL,  
    Name           varchar(30),  
    Phone          INT,  
    Address        varchar(30),  
    CONSTRAINT PKDOCTOR PRIMARY KEY (NIF)  
);  
  
CREATE TABLE LarIdosos.PATIENT(  
    NIF            varchar(9),  
    Name           varchar(30),  
    Sex            varchar(1),  
    Phone          INT,  
    Age            INT,  
    Check_in       DATE,  
    Check_out      DATE,  
    Authorization_to_leave BIT,  
    E_BedNumber    INT,  
    Entry_Date     DATE,  
    Exit_Date      DATE,  
    CHECK(Check_in < Check_out),  
    CHECK(Entry_Date < Exit_Date),  
    CONSTRAINT PKPATIENT PRIMARY KEY (NIF),  
    CONSTRAINT FKPATIENT2 FOREIGN KEY (E_BedNumber) REFERENCES LarIdosos.BED(BedNumber)  
);  
  
CREATE TABLE LarIdosos.DISEASE(  
    Name           varchar(15)    NOT NULL,  
    CONSTRAINT PKDISEASE PRIMARY KEY (Name)  
);  
  
CREATE TABLE LarIdosos.DIAGNOSED(  
    E_Name         varchar(15)    NOT NULL,  
    E_NIF          varchar(9)     NOT NULL,  
    Seriousness    INT,  
    Disable        BIT,  
    CONSTRAINT FKDIAGNOSED FOREIGN KEY (E_Name) REFERENCES LarIdosos.DISEASE(Name),  
    CONSTRAINT FKDIAGNOSED2 FOREIGN KEY (E_NIF) REFERENCES LarIdosos.PATIENT(NIF),  
    CONSTRAINT PKDIAGNOSED PRIMARY KEY (E_Name, E_NIF)  
);  
  
CREATE TABLE LarIdosos.EXITS(  
    ID             INT          NOT NULL    IDENTITY(1, 1),  
    Check_in       DATE,  
    Check_out      DATE,  
    CHECK(Check_in < Check_out),  
    CONSTRAINT PKEXITS PRIMARY KEY (ID)  
);
```

```

CREATE TABLE LarIdosos.SCHEDULE(
    Day      varchar(15)      NOT NULL,
    Hour     TIME             NOT NULL,
    E_NIF    varchar(9),
    CONSTRAINT FKSCCHEDULE FOREIGN KEY (E_NIF) REFERENCES LarIdosos.PATIENT(NIF),
    CONSTRAINT PKSCCHEDULE PRIMARY KEY (Day, Hour, E_NIF)
);

CREATE TABLE LarIdosos.MEDICINE(
    ID       INT              NOT NULL      IDENTITY(1, 1),
    Name     varchar(30) NOT NULL,
    CONSTRAINT PKMEDICINE PRIMARY KEY (ID)
);

CREATE TABLE LarIdosos.TAKING(
    E_Day    varchar(15) NOT NULL,
    E_Hour   TIME        NOT NULL,
    E_NIF    varchar(9)  NOT NULL,
    E_ID     INT          NOT NULL,
    Dose     INT,
    Disable  Bit Default (0),
    CONSTRAINT FKTKING FOREIGN KEY (E_Day, E_Hour, E_NIF) REFERENCES LarIdosos.SCHEDULE(Day, Hour, E_NIF),
    CONSTRAINT FKTKING2 FOREIGN KEY (E_ID) REFERENCES LarIdosos.MEDICINE(ID),
    CONSTRAINT PKTKING PRIMARY KEY (E_Day, E_Hour, E_NIF, E_ID)
);

CREATE TABLE LarIdosos.APPOINTMENT(
    ID              INT              NOT NULL      IDENTITY(1, 1),
    E_DoctorNIF    varchar(9),
    E_PatientNIF   varchar(9),
    Date           DATETIME,
    Speciality     varchar(30),
    Disable        BIT DEFAULT 0,
    CONSTRAINT PKAPPOINTMENT PRIMARY KEY (ID),
    CONSTRAINT FKAPPOINTMENT FOREIGN KEY (E_DoctorNIF) REFERENCES LarIdosos.DOCTOR(NIF),
    CONSTRAINT FKAPPOINTMENT2 FOREIGN KEY (E_PatientNIF) REFERENCES LarIdosos.PATIENT(NIF)
);

CREATE TABLE LarIdosos.VISIT(
    ID              INT              NOT NULL      IDENTITY(1, 1),
    Date           DATETIME,
    CONSTRAINT PKVISIT PRIMARY KEY (ID)
);

CREATE TABLE LarIdosos.RECEIVED(
    E_NIF          varchar(9)      NOT NULL,
    E_IDVisit      INT            NOT NULL,
    CONSTRAINT FKRECEIVED FOREIGN KEY (E_NIF) REFERENCES LarIdosos.PATIENT(NIF),
    CONSTRAINT FKRECEIVED2 FOREIGN KEY (E_IDVisit) REFERENCES LarIdosos.VISIT(ID),
    CONSTRAINT PKRECEIVED PRIMARY KEY (E_NIF, E_IDVisit)
);

CREATE TABLE LarIdosos.ACCOMPANIED(
    NIF_Patient    varchar(9)      NOT NULL,
    NIF_Doctor     varchar(9)      NOT NULL,
    CONSTRAINT FKACCOMPAINED FOREIGN KEY (NIF_Doctor) REFERENCES LarIdosos.DOCTOR(NIF),
    CONSTRAINT FKACCOMPAINED2 FOREIGN KEY (NIF_Patient) REFERENCES LarIdosos.PATIENT(NIF),
    CONSTRAINT PKACCOMPAINED PRIMARY KEY (NIF_Patient, NIF_Doctor)
);

```

```

CREATE TABLE LarIdosos.VISITOR(
  CC      varchar(9)      NOT NULL,
  Name    varchar(30),
  Address varchar(30),
  Phone   INT,
  CONSTRAINT PKVISITOR PRIMARY KEY (CC)
);

CREATE TABLE LarIdosos.VISITED(
  E_IDVisit    INT      NOT NULL,
  E_CCVisitor  varchar(9) NOT NULL,
  CONSTRAINT FKVISITED FOREIGN KEY (E_IDVisit) REFERENCES LarIdosos.VISIT(ID),
  CONSTRAINT FKVISITED2 FOREIGN KEY (E_CCVisitor) REFERENCES LarIdosos.VISITOR(CC),
  CONSTRAINT PKVISITED PRIMARY KEY (E_IDVisit, E_CCVisitor)
);

CREATE TABLE LarIdosos.FAMILY(
  E_CC      varchar(9)      NOT NULL,
  Relationship varchar(30),
  CONSTRAINT FKFAMILY FOREIGN KEY (E_CC) REFERENCES LarIdosos.VISITOR(CC),
  CONSTRAINT PKFAMILY PRIMARY KEY (E_CC)
);

CREATE TABLE LarIdosos.NOT_FAMILY(
  E_CC      varchar(9)      NOT NULL,
  KinshipDegree varchar(15),
  CONSTRAINT FKNOT_FAMILY FOREIGN KEY (E_CC) REFERENCES LarIdosos.VISITOR(CC),
  CONSTRAINT PKNOT_FAMILY PRIMARY KEY (E_CC)
);

CREATE TABLE LarIdosos.DEPENDENT(
  E_CC      varchar(9)      NOT NULL,
  E_PatientNIF varchar(9) NOT NULL,
  CONSTRAINT FKDEPENDENT FOREIGN KEY (E_CC) REFERENCES LarIdosos.FAMILY(E_CC),
  CONSTRAINT FKDEPENDENT2 FOREIGN KEY (E_PatientNIF) REFERENCES LarIdosos.PATIENT(NIF),
  CONSTRAINT PKDEPENDENT PRIMARY KEY (E_CC, E_PatientNIF)
);

CREATE TABLE LarIdosos.LEAVES(
  ID      INT      NOT NULL,
  E_NIF   varchar(9) ,
  CONSTRAINT FKLEAVES FOREIGN KEY (ID) REFERENCES LarIdosos.EXITS(ID),
  CONSTRAINT FKLEAVES2 FOREIGN KEY (E_NIF) REFERENCES LarIdosos.PATIENT(NIF),
  CONSTRAINT PKLEAVES PRIMARY KEY (ID, E_NIF)
);

```



```

CREATE TABLE LarIdosos.Type(
    ID          INT          NOT NULL      IDENTITY(1, 1),
    Designation VARCHAR(30)    UNIQUE,
    Disable     BIT DEFAULT 0,
    PRIMARY KEY (ID)
);

CREATE TABLE LarIdosos.HumanResources(
    NIF          varchar(9)    NOT NULL,
    Name         VARCHAR(30),
    Phone        INT          UNIQUE,
    Address       VARCHAR(30),
    Salary        INT,
    StartDate     DATE,
    E_IDType      INT          NOT NULL,
    PRIMARY KEY (NIF),
    FOREIGN KEY (E_IDType) REFERENCES LarIdosos.Type(ID)
);

CREATE TABLE LarIdosos.Shift(
    ID          INT          NOT NULL      IDENTITY(1, 1),
    Day         VARCHAR(30),
    BeginOfWorkShift TIME,
    EndOfWorkShift TIME,
    PRIMARY KEY (ID)
);

CREATE TABLE LarIdosos.ShiftInstance(
    ID          INT          NOT NULL IDENTITY(1, 1),
    InicialDate DATE,
    FinalDate   DATE,
    E_NIF        varchar(9)    NOT NULL,
    E_IDShift     INT          NOT NULL,
    PRIMARY KEY (ID),
    FOREIGN KEY (E_NIF) REFERENCES LarIdosos.HumanResources(NIF),
    FOREIGN KEY (E_IDShift) REFERENCES LarIdosos.Shift(ID)
);

CREATE TABLE LarIdosos.Faults(
    Date        DATE,
    E_ID         INT          NOT NULL,
    FOREIGN KEY (E_ID) REFERENCES LarIdosos.ShiftInstance(ID),
    PRIMARY KEY (Date, E_ID)
);

```

## Normalização

Após análise do modelo relacional chegamos à conclusão que não existe nenhuma violação das formas normais, estando a base de dados na 3ª Forma Normal.

## Índices

Após análise do modelo relacional e considerando as queries utilizadas no projeto concluímos que não era necessário a criação de novos índices para além dos que são criados por default para chaves primárias, unique clustered index.

## User-Defined Functions

- A UDF **getFreeBeds** retorna todas as Beds de um determinado RoomNumber.

```
CREATE FUNCTION dbo.getFreeBeds(@RoomNumber INT)
RETURNS TABLE
AS
RETURN
    SELECT BedNumber FROM (
        SELECT E_RoomNumber, BedNumber FROM LarIdosos.BED
    EXCEPT
        SELECT E_RoomNumber, BedNumber FROM LarIdosos.PATIENT join LarIdosos.BED
            ON E_BedNumber = BedNumber) AS tmp join LarIdosos.BEDROOM on tmp.E_RoomNumber = RoomNumber
    WHERE tmp.E_RoomNumber = @RoomNumber
GO
```

- A UDF **getFreeRoomsAndBeds** retorna todos os Rooms livres e as suas camas também livres.

```
CREATE FUNCTION dbo.getFreeRoomsAndBeds()
RETURNS TABLE
AS
RETURN(
    select distinct RoomNumber from (select RoomNumber, BedNumber from LarIdosos.BED
    join LarIdosos.BEDROOM on E_RoomNumber = RoomNumber

    except
    select RoomNumber, BedNumber from LarIdosos.PATIENT
    join LarIdosos.BED on E_BedNumber = BedNumber
    join LarIdosos.BEDROOM on E_RoomNumber = RoomNumber) as tmp
)
GO
```

- A UDF **getHumanResourcesFaults** retorna todas as faltas dadas de um determinado trabalhador.

```
CREATE FUNCTION dbo.getHumanResourceFaults(@NIF Varchar(9))
RETURNS TABLE
AS
RETURN(
    SELECT LarIdosos.Shift.Day, BeginOfWorkShift, EndOfWorkShift, E_IDShift, FinalDate, count(LarIdosos.Faults.E_ID) as NumberOfFaults
    FROM (LarIdosos.ShiftInstance JOIN LarIdosos.Shift on E_IDShift= LarIdosos.Shift.ID)
    LEFT OUTER JOIN LarIdosos.Faults on LarIdosos.Faults.E_ID = LarIdosos.ShiftInstance.ID
    Where E_nif = @NIF
    GROUP BY Day, BeginOfWorkShift, EndOfWorkShift, E_IDShift, FinalDate
)
GO
```

- A UDF **getHumanResources** retorna todos os trabalhadores do Lar de Idosos

```
CREATE FUNCTION dbo.getHumanResources(@WorkerName varchar(30)=NULL, @WorkerNif varchar(9)=NULL,
@WorkerPhone INT=NULL, @WorkerAddress varchar(30)=NULL, @Designation varchar(30)=NULL, @PageNumber INT, @RowsPage INT)
RETURNS TABLE
AS
RETURN(
    select NIF, Name, Phone, Address, Salary, StartDate, Designation from LarIdosos.HumanResources
    join LarIdosos.Type on E_IDType = ID WHERE (@WorkerName IS NULL OR Name = @WorkerName)
    AND
    (@WorkerNif IS NULL OR NIF = @WorkerNif)
    AND
    (@WorkerPhone IS NULL OR Phone = @WorkerPhone)
    AND
    (@WorkerAddress IS NULL OR SOUNDEX([Address]) = SOUNDEX(@WorkerAddress))
    AND
    (@Designation IS NULL OR Designation = @Designation) ORDER BY NIF
    OFFSET ((@PageNumber - 1) * @RowsPage) ROWS
    FETCH NEXT @RowsPage ROWS ONLY
)
```

- A UDF **getHumanResourcesSchedule** retorna o horário de um determinado trabalhador do Lar de Idosos.

```
CREATE FUNCTION dbo.getHumanResourceSchedule(@NIF Varchar(9))
RETURNS TABLE
AS
RETURN(
    SELECT Day, BeginOfWorkShift, EndOfWorkShift,E_IDShift
    FROM (LarIdosos.ShiftInstance
        JOIN LarIdosos.Shift on E_IDShift= LarIdosos.Shift.ID)
    Where E_nif = @NIF and FinalDate IS NULL
)
GO
```

- A UDF **getHumanTypes** retorna todos os tipos de trabalhadores existentes

```
CREATE FUNCTION dbo.getHumanTypes()
RETURNS TABLE
AS
RETURN(
    select Designation,Id from LarIdosos.Type
)
GO
```

- A UDF **getPatientAppointments** retorna todas as consultas de um determinado Doutor.

```
CREATE FUNCTION dbo.getPatientAppointments(@NIF varchar(9))
RETURNS TABLE
AS
RETURN(
    select LarIdosos.DOCTOR.Name, Date, Speciality, LarIdosos.APPOINTMENT.ID FROM (
        LarIdosos.APPOINTMENT JOIN LarIdosos.DOCTOR on E_DoctorNIF = LarIdosos.DOCTOR.NIF)
    WHERE LarIdosos.APPOINTMENT.Disable = 0 AND LarIdosos.APPOINTMENT.E_PatientNif = @NIF
)
GO
```

- A UDF **getPatientDiseases** retorna um diagnóstico contendo as doenças de um determinado paciente e a sua seriedade.

```
CREATE FUNCTION dbo.getPatientDiseases(@NIF INT)
RETURNS TABLE
AS
RETURN
    SELECT E_name,Seriousness FROM LarIdosos.DIAGNOSED
    WHERE LarIdosos.DIAGNOSED.E_NIF = @NIF and LarIdosos.DIAGNOSED.Disable = 0
GO
```



- A UDF **getPatientMedicines** retorna quais os medicamentos e o horario da toma de cada medicamento de um determinado paciente.

```
CREATE FUNCTION dbo.getPatientMedicines(@NIF Varchar(9))
RETURNS TABLE
AS
RETURN(
    select Name, Dose, E_Day,E_Hour,E_ID FROM (
        LarIdosos.TAKING JOIN LarIdosos.MEDICINE on E_ID=ID
    )
    Where E_nif = @NIF AND LarIdosos.TAKING.Disable = 0
)
GO
```

- A UDF **getPatients** retorna todos os pacientes e toda a informação associada, com a possibilidade de pesquisa dinâmica, assim como alterar a ordem de retorno.

```
CREATE FUNCTION dbo.getPatients(@PatientNif VARCHAR(9)=NULL,@PatientName VARCHAR(30)=NULL,@Sex varchar(1)=NULL,
    @authorization BIT=NULL,@RoomNumber INT=NULL,@PhoneNNumber INT=NULL,@Checkout BIT=NULL,@PageNumber INT,
    @RowsPage INT, @sortOrder VARCHAR(5),@sortColumn VARCHAR(30))
RETURNS TABLE
AS
RETURN(
    select LarIdosos.PATIENT.NIF, LarIdosos.PATIENT.Name, LarIdosos.PATIENT.Sex, LarIdosos.PATIENT.Phone,
    LarIdosos.PATIENT.Age, LarIdosos.PATIENT.Check_in, LarIdosos.PATIENT.Check_out, Authorization_to_leave,
    E_BedNumber, Entry_Date, Exit_Date, RoomNumber, LarIdosos.VISITOR.Name as DependentName, LarIdosos.VISITOR.CC as DependentCC,
    LarIdosos.VISITOR.Address as DependentAddress, LarIdosos.VISITOR.Phone as DependentPhone, LarIdosos.FAMILY.Relationship
    from( LarIdosos.PATIENT JOIN LarIdosos.BED on E_BedNumber = BedNumber
    JOIN LarIdosos.BEDROOM on E_RoomNumber = RoomNumber
    FULL OUTER JOIN LarIdosos.DEPENDENT on E_PatientNIF = NIF
    LEFT OUTER JOIN LarIdosos.FAMILY on LarIdosos.DEPENDENT.E_CC = LarIdosos.FAMILY.E_CC
    LEFT OUTER JOIN LarIdosos.VISITOR on LarIdosos.FAMILY.E_CC = LarIdosos.VISITOR.CC )
    WHERE (@PatientNif IS NULL OR LarIdosos.PATIENT.NIF = @PatientNif)
    AND (@PatientName IS NULL OR LarIdosos.PATIENT.Name = @PatientName)
    AND (@Sex IS NULL OR LarIdosos.PATIENT.Sex = @Sex)
    AND (@authorization IS NULL OR Authorization_to_leave = @authorization)
    AND (@RoomNumber IS NULL OR RoomNumber = @RoomNumber)
    AND (@PhoneNNumber IS NULL OR LarIdosos.PATIENT.Phone = @PhoneNNumber)
    AND (@Checkout IS NOT NULL OR LarIdosos.Patient.Check_out IS NULL ) ORDER BY
        case
            when @sortOrder <> 'ASC' then ''
            when @sortColumn = 'Name' then LarIdosos.PATIENT.Name
            end ASC
        ,case
            when @sortOrder <> 'ASC' then ''
            when @sortColumn = 'Nif' then LarIdosos.PATIENT.NIF
            end ASC
        ,case
            when @sortOrder <> 'ASC' then ''
            when @sortColumn = 'Check in' then Check_in
            end ASC
        ,case
            when @sortOrder <> 'ASC' then ''
            when @sortColumn = 'Check out' then Check_out
            end ASC
        ,case
            when @sortOrder <> 'ASC' then ''
            when @sortColumn = 'Room Number' then RoomNumber
            end ASC
        ,case
            when @sortOrder <> 'ASC' then ''
            when @sortColumn = 'Bed Number' then BedNumber
            end DESC
```



```

,case
when @sortOrder <> 'DESC' then ''
when @sortColumn = 'Name' then LarIdosos.PATIENT.Name
end DESC
,case
when @sortOrder <> 'DESC' then ''
when @sortColumn = 'Nif' then LarIdosos.PATIENT.NIF
end DESC
,case
when @sortOrder <> 'DESC' then ''
when @sortColumn = 'Check in' then Check_in
end DESC
,case
when @sortOrder <> 'DESC' then ''
when @sortColumn = 'Check out' then Check_out
end DESC
,case
when @sortOrder <> 'DESC' then ''
when @sortColumn = 'Room Number' then RoomNumber
end DESC
,case
when @sortOrder <> 'DESC' then ''
when @sortColumn = 'Bed Number' then BedNumber
end DESC
OFFSET ((@PageNumber - 1) * @RowsPage) ROWS
FETCH NEXT @RowsPage ROWS ONLY
)

```

- A UDF **searchAppointments** retorna os appointments com a possibilidade de pesquisa dinâmica, assim como alterar a ordem de retorno.

```

CREATE FUNCTION dbo.searchAppointments(@DoctorNif VARCHAR(9)=NULL,@PatientNif VARCHAR(9)=NULL,@DoctorName varchar(30)=NULL,
@PatientName varchar(30)=NULL,@Date DateTime=NULL,@Speciality varchar(30)=NULL, @PageNumber INT, @RowsPage INT)
RETURNS TABLE
AS
RETURN
SELECT E_DoctorNIF as DoctorNIF, LarIdosos.DOCTOR.Name as DoctorName, E_PatientNIF as PatientNIF,
LarIdosos.PATIENT.Name as PatientName, Date, Speciality from (LarIdosos.APPOINTMENT JOIN LarIdosos.Doctor on E_DoctorNIF = LarIdosos.DOCTOR.NIF)
JOIN LarIdosos.PATIENT ON E_PatientNIF = LarIdosos.PATIENT.NIF
WHERE (@DoctorNif IS NULL OR LarIdosos.APPOINTMENT.E_DoctorNIF = @DoctorNif)
AND
(@PatientNif IS NULL OR LarIdosos.APPOINTMENT.E_PatientNIF = @PatientNif)
AND
(@DoctorName IS NULL OR LarIdosos.DOCTOR.Name = @DoctorName)
AND
(@PatientName IS NULL OR LarIdosos.PATIENT.Name = @PatientName)
AND
(@Date IS NULL OR CAST(Date as DATE) = @Date)
AND
(@Speciality IS NULL OR LarIdosos.APPOINTMENT.Speciality = @Speciality) ORDER BY LarIdosos.PATIENT.NIF
OFFSET ((@PageNumber - 1) * @RowsPage) ROWS
FETCH NEXT @RowsPage ROWS ONLY

```

- A UDF **getVisitors** retorna os visitantes, com a possibilidade de pesquisa dinâmica.

```
CREATE FUNCTION dbo.getVisitors(@VisitorName varchar(30)=NULL,@VisitorCC varchar(9)=NULL,
    @VisitorPhone INT=NULL,@PageNumber INT, @RowsPage INT)
RETURNS TABLE
AS
RETURN(
    select Name, CC, Phone from LarIdosos.Visitor WHERE
        (@VisitorName IS NULL OR LarIdosos.VISITOR.Name = @VisitorName)
        AND
        (@VisitorCC IS NULL OR CC = @VisitorCC)
        AND
        (@VisitorPhone IS NULL OR Phone = @VisitorPhone) ORDER BY LarIdosos.VISITOR.CC
    OFFSET ((@PageNumber - 1) * @RowsPage) ROWS FETCH NEXT @RowsPage ROWS ONLY
)
```

- A UDF **getVisits** retorna um registo de todas as visitas efetuadas indicando o visitante, tipo de visitante e o paciente visitado e tem a possibilidade de pesquisa dinâmica, assim como alterar a ordem de retorno.

```
CREATE FUNCTION dbo.getVisits(@PatientNif varchar(9)=NULL,@PatientName varchar(30)=NULL,@VisitorName varchar(30)=NULL,
    @VisitorCC varchar(9)=NULL,@VisitorPhone INT=NULL,@Date Date=NULL,@PageNumber INT, @RowsPage INT,@sortOrder VARCHAR(30),@sortColumn VARCHAR(30))
RETURNS TABLE
AS
RETURN(
    select LarIdosos.PATIENT.Name as PatientName, NIF, LarIdosos.VISITOR.Name as VisitorName, CC, LarIdosos.VISITOR.Phone,
    LarIdosos.VISITOR.Address,LarIdosos.VISIT.Date, LarIdosos.NOT_Family.KinshipDegree, LarIdosos.FAMILY.Relationship from LarIdosos.VISITOR
    join LarIdosos.VISITED on CC = E_CCVisitor
    join LarIdosos.VISIT on E_IDVisit = ID
    join LarIdosos.RECEIVED on LarIdosos.RECEIVED.E_IDVisit = ID
    join LarIdosos.PATIENT on E_NIF = NIF
    Left Outer join LarIdosos.NOT_FAMILY on CC = LarIdosos.NOT_FAMILY.E_CC
    Left Outer join LarIdosos.FAMILY on CC = LarIdosos.FAMILY.E_CC WHERE
        (@VisitorName IS NULL OR LarIdosos.VISITOR.Name = @VisitorName)
        AND
        (@VisitorCC IS NULL OR CC = @VisitorCC)
        AND
        (@VisitorPhone IS NULL OR LarIdosos.VISITOR.Phone = @VisitorPhone)
        AND
        (@PatientNif IS NULL OR NIF = @PatientNif)
        AND
        (@PatientName IS NULL OR LarIdosos.PATIENT.Name = @PatientName)
        AND
        (@Date IS NULL OR CAST(Date as DATE) = @Date)
    ORDER BY case
        when @sortOrder <> 'ASC' then ''
        when @sortColumn = 'Patient Name' then LarIdosos.PATIENT.Name
        end ASC
    ,case
        when @sortOrder <> 'ASC' then ''
        when @sortColumn = 'Visitor Name' then LarIdosos.VISITOR.Name
        end ASC
    ,case
        when @sortOrder <> 'ASC' then ''
        when @sortColumn = 'Patient Nif' then NIF
        end ASC
    ,case
        when @sortOrder <> 'ASC' then ''
        when @sortColumn = 'Visitor CC' then CC
        end ASC
    ,case
        when @sortOrder <> 'ASC' then ''
        when @sortColumn = 'Date' then Date
        end ASC
    ,case
        when @sortOrder <> 'DESC' then ''
        when @sortColumn = 'Patient Name' then LarIdosos.PATIENT.Name
        end DESC
)
```

```

,case
when @sortOrder <> 'DESC' then ''
when @sortColumn = 'Visitor Name' then LarIdosos.VISITOR.Name
end DESC
,case
when @sortOrder <> 'DESC' then ''
when @sortColumn = 'Patient Nif' then NIF
end DESC
,case
when @sortOrder <> 'DESC' then ''
when @sortColumn = 'Visitor CC' then CC
end DESC
,case
when @sortOrder <> 'DESC' then ''
when @sortColumn = 'Date' then Date
end DESC

OFFSET ((@PageNumber - 1) * @RowsPage) ROWS FETCH NEXT @RowsPage ROWS ONLY

GO
)

```

- A UDF **getDoctors** retorna os doctors registados.

```

CREATE FUNCTION dbo.getDoctors()
RETURNS TABLE
AS
RETURN(
    select NIF,Name,Phone,Address from LarIdosos.DOCTOR
)

```

## Stored Procedures

- A Stored Procedure **sp\_addFaulttoHumanResource** insere uma falta no turno em que o trabalhador faltou.

```

CREATE PROCEDURE dbo.sp_addFaulttoHumanResource (@NIF INT, @beginofWorkShift Time = null, @endofWorkShift Time = null, @day varchar(30))
as
begin
    if (not Exists(Select * from LarIdosos.HumanResources where NIF=@NIF ))
    BEGIN
        RAISERROR('O recurso humano não existe!',16,1)
        return;
    END

    DECLARE @idShiftInstace INT;

    SELECT @idShiftInstace = LarIdosos.ShiftInstance.ID FROM (LarIdosos.ShiftInstance
        JOIN LarIdosos.Shift ON LarIdosos.Shift.ID = E_IDShift)
    WHERE BeginOfWorkShift = @beginofWorkShift AND endofWorkShift = @endofWorkShift AND Day = @day;
    IF (@@ROWCOUNT = 0)
    BEGIN
        RAISERROR('Não existe esse turno',16,1);
        RETURN;
    END

    if(NOT EXISTS(SELECT * FROM LarIdosos.Faults WHERE E_ID=@idShiftInstace AND Date = CONVERT (date, GETDATE()))
    BEGIN
        INSERT INTO LarIdosos.Faults values(GETDATE(),@idShiftInstace);
    END
    ELSE
    BEGIN
        RAISERROR('Não pode marcar mais que uma falta no mesmo dia na mesma data.',16,1)
        RETURN;
    END

    RETURN;
END
GO

```

- A Stored Procedure **sp\_addPatientAppointment** adiciona uma nova consulta ao Paciente.

```
CREATE PROCEDURE dbo.sp_addPatientAppointment
    @PatientNif varchar(9),
    @DoctorNif varchar(9),
    @Date DateTime,
    @Speciality varchar(30)
AS
BEGIN
    if (Exists(Select * from LarIdosos.APPOINTMENT
        Where E_PatientNIF=@PatientNif AND E_DoctorNIF = @DoctorNif AND Date = @Date And Speciality = @Speciality ))
    begin
        raiserror('O paciente já tem associada esta consulta',16,1);
        return;
    end
    if(Exists(Select * from LarIdosos.APPOINTMENT Where E_PatientNIF=@PatientNif AND Date = @Date ))
    begin
        raiserror('O paciente já tem consulta para essa hora nesse dia',16,1);
        return;
    end

    Declare @disable BIT;
    SET @Disable = 0 ;

    INSERT INTO LarIdosos.APPOINTMENT values (
        @DoctorNif,
        @PatientNif,
        @Date,
        @Speciality,
        @Disable
    )

    return;
end
```

- A Stored Procedure **sp\_deleteAppointment** remove uma consulta com determinada identificação.

```
CREATE PROCEDURE dbo.sp_deleteAppointment(@ID INT)
AS
BEGIN
    Delete FROM LarIdosos.APPOINTMENT WHERE ID = @ID;
    RETURN
end
```

- A Stored Procedure **sp\_deleteMedicine** remove o registo da toma de um medicamento de determinado Paciente.

```
CREATE PROCEDURE dbo.sp_deleteMedicine(@E_Day varchar(15), @E_Hour time, @E_NIF varchar(9), @medicineID int)
AS
BEGIN
    Delete FROM EXEMPLO1.TAKING WHERE E_day = @E_Day AND E_Hour = @E_Hour AND E_NIF = @E_NIF AND E_ID = @medicineID;
    RETURN
end
```

- A Stored Procedure **sp\_newVisitor** insere um novo visitante podendo este ser um familiar ou não, caso seja é inserido na tabela Family, senão é inserido em

Not\_Family.

```
CREATE PROCEDURE dbo.sp_newVisitor ( @CC varchar(9), @Name varchar(30), @Address varchar(15),  
@Phone int,@Relationship VARCHAR(30),@Family BIT)  
as  
begin  
    if (Exists(Select CC from LarIdosos.VISITOR where CC=@CC ))  
        RETURN  
    if (Exists(Select NIF from LarIdosos.PATIENT where NIF=@CC ))  
        RETURN  
  
    INSERT INTO LarIdosos.VISITOR values(  
        @CC,  
        @Name,  
        @Address,  
        @Phone  
    )  
    if (@Family = 1)  
    BEGIN  
        Insert Into LarIdosos.FAMILY values(@CC,@Relationship);  
    END  
    ELSE  
    BEGIN  
        INSERT INTO LarIdosos.NOT_FAMILY values(@CC,@Relationship)  
    END  
    RETURN  
END
```

- A Stored Procedure **sp\_finishShiftOfHumanResources** elimina o turno de um trabalhador.

```
CREATE PROCEDURE dbo.sp_finishShiftOfHumanResources (@ID INT)  
AS  
BEGIN  
    Delete FROM LarIdosos.shift WHERE (ID = @ID )  
    RETURN  
end
```

- A Stored Procedure **sp\_insertHumanResources** insere um novo trabalhador no lar, caso exista, faz update dos seus dados.



```
CREATE PROCEDURE [dbo].[sp_insertHumanResources]
    @NIF    varchar(9),
    @Name   varchar(30),
    @Phone  INT,
    @Address varchar(30),
    @Salary INT,
    @Start_Date DATE = null,
    @E_IDType INT
AS
BEGIN
    if (Exists(Select NIF from LarIdosos.HUMANRESOURCES where NIF=@NIF ))
    begin
        UPDATE LarIdosos.HUMANRESOURCES
        SET NIF = @NIF,
            Name = @Name,
            Phone = @Phone,
            Address = @Address,
            Salary = @Salary,
            StartDate = @Start_Date,
            E_IDType = @E_IDType
        where NIF = @NIF
    end
    else
    begin
        INSERT INTO LarIdosos.HUMANRESOURCES values (
            @NIF,
            @Name,
            @Phone,
            @Address,
            @Salary,
            @Start_Date,
            @E_IDType
        )
    end
    return
end
```

- A Stored Procedure **sp\_insertPATIENT** insere um novo paciente no lar caso este não exista.

```

CREATE PROCEDURE [dbo].[sp_insertPATIENT]
    @NIF    varchar(9),
    @Name   varchar(30),
    @Sex    varchar(1),
    @Phone  INT,
    @Age    INT,
    @Check_in  DATE,
    @Check_out DATE,
    @Authorization_to_leave BIT,
    @E_BedNumber INT,
    @Entry_Date DATE,
    @Exit_Date DATE

AS
BEGIN
    if (Exists(Select NIF from LarIdosos.PATIENT where NIF=@NIF ))
        RAISERROR('O Patient ja existe!',16,1);
    else
        INSERT INTO LarIdosos.PATIENT values (
            @NIF,
            @Name,
            @Sex,
            @Phone,
            @Age,
            @Check_in,
            @Check_out,
            @Authorization_to_leave,
            @E_BedNumber,
            @Entry_Date,
            @Exit_Date
        )
    return
end

```

- A Stored Procedure **sp\_newDependent** associa um encarregado/responsável pelo paciente (por exemplo um filho que coloca o pai no lar).

```

CREATE PROCEDURE dbo.sp_newDependent (@E_NIF varchar(9), @Name varchar(30), @CC varchar(9),
    @Phone int, @Address varchar(30), @Relationship varchar(30))
as
begin
    if (Exists(Select NIF from LarIdosos.PATIENT where NIF=@E_NIF ))
        if (not Exists(Select CC from LarIdosos.VISITOR where CC=@CC ))
            INSERT INTO LarIdosos.VISITOR values (
                @CC,
                @Name,
                @Address,
                @Phone
            )
        if(not Exists(SELECT * FROM LarIdosos.FAMILY where E_CC=@CC))
            BEGIN
                INSERT INTO LarIdosos.FAMILY values (
                    @CC,
                    @Relationship
                )
            END

            INSERT INTO LarIdosos.DEPENDENT values (
                @CC,
                @E_NIF
            )

    RETURN
END
GO

```

- A Stored Procedure **sp\_newDiagnosed** cria um novo diagnóstico que indicar as doenças associadas e o seu grau de seriedade.

```
CREATE PROCEDURE dbo.sp_newDiagnosed (@E_Name varchar(30), @E_NIF varchar(9), @Seriousness int, @Disable BIT)
as
begin
    if (Exists(Select E_NIF, E_Name from LarIdosos.DIAGNOSED where E_NIF=@E_NIF AND E_Name = @E_Name ))
        RAISERROR('Diagnosed já existente!',16,1);
    else
    begin
        if (not Exists(Select Name from LarIdosos.DISEASE where Name=@E_Name ))
            INSERT INTO LarIdosos.DISEASE values(
                @E_Name
            )
        if (Exists(Select NIF from LarIdosos.PATIENT where NIF=@E_NIF ))
            INSERT INTO LarIdosos.DIAGNOSED values(
                @E_Name,
                @E_NIF,
                @Seriousness,
                @Disable
            )
        else
            RAISERROR('O NIF do Patient nao existe!',16,1);
        end
    end
    RETURN
END
```

- A Stored Procedure **sp\_newDoctor** insere um novo Doutor.



```

CREATE PROCEDURE dbo.sp_newDoctor
    @NIF    varchar(9),
    @Name   varchar(30),
    @Phone  INT,
    @Address varchar(30)

AS
BEGIN
    if (Exists(Select NIF from LarIdosos.DOCTOR where NIF=@NIF ))
    begin
        UPDATE LarIdosos.HUMANRESOURCES
        SET NIF = @NIF,
            Name = @Name,
            Phone = @Phone,
            Address = @Address
        where NIF = @NIF
    end
    else
    begin
        INSERT INTO LarIdosos.DOCTOR values (
            @NIF,
            @Name,
            @Phone,
            @Address
        )
    end
    return
end

```

- A Stored Procedure **sp\_newSchedule** trata da determinação e criação de horários e turnos dos trabalhadores.

```

CREATE PROCEDURE dbo.sp_newSchedule (@NIF INT,@initialDate Date, @finalDate Date=null, @beginofWorkShift Time, @endofWorkShift Time, @day varchar(30))
as
begin
    if (not Exists(Select * from LarIdosos.HumanResources where NIF=@NIF ))
    BEGIN
        RAISERROR('O recurso humano não existe!',16,1)
        return;
    END

    DECLARE @idShift INT;
    Select @idShift=ID FROM LarIdosos.Shift WHERE BeginOfWorkShift = @beginofWorkShift AND EndOfWorkShift = @endofWorkShift AND Day = @day;

    if(@@ROWCOUNT = 0)
    BEGIN
        INSERT INTO LarIdosos.Shift VALUES (@day, @beginofWorkShift, @endofWorkShift);
        Select @idShift=ID FROM LarIdosos.Shift WHERE BeginOfWorkShift = @beginofWorkShift AND EndOfWorkShift = @endofWorkShift AND Day = @day;
    END

    if(not Exists(SELECT * FROM LarIdosos.ShiftInstance WHERE E_NIF=@NIF AND E_IDShift = @idShift))
    BEGIN
        INSERT INTO LarIdosos.ShiftInstance VALUES (@initialDate,@finalDate,@NIF,@idShift);
        RETURN;
    END

    if(Exists(SELECT * FROM LarIdosos.ShiftInstance WHERE FinalDate is NULL AND E_NIF=@NIF AND E_IDShift = @idShift))
    BEGIN
        RAISERROR('Já tem associado esse horário!',16,1)
        return;
    END

    if(Exists(SELECT * FROM LarIdosos.ShiftInstance WHERE FinalDate IS NOT null AND E_NIF=@NIF AND E_IDShift = @idShift))
    BEGIN
        INSERT INTO LarIdosos.ShiftInstance VALUES (@initialDate,@finalDate,@NIF,@idShift);
    END

    RETURN;
END

```

- A Stored Procedure **sp\_newTaking** cria um horário para a toma de um medicamento.

```
CREATE PROCEDURE dbo.sp_newTaking (@E_Day varchar(15), @E_Hour time, @E_NIF varchar(9), @medicineName varchar(30), @Dose int)
as
begin
    if (not Exists(Select * from LarIdosos.PATIENT where NIF=@E_NIF ))
        RAISERROR('O Patient não existe!',16,1)

    DECLARE @medicineID INT;
    SELECT @medicineID = id from LarIdosos.medicine where Name = @medicineName;
    if @@ROWCOUNT = 0
    BEGIN
        INSERT INTO LarIdosos.medicine values (@medicineName)
        SELECT @medicineID = id from LarIdosos.medicine where Name = @medicineName;
    END

    SELECT * from LarIdosos.Schedule Where day=@E_Day and hour=@E_Hour and E_NIF = @E_NIF;
    if @@ROWCOUNT = 0
    BEGIN
        INSERT INTO LarIdosos.Schedule values (@E_Day,@E_Hour, @E_NIF);
    END

    DECLARE @DISABLE BIT;
    SET @DISABLE = 0;
    IF(Exists(Select * from LarIdosos.TAKING Where E_day = @E_Day
        AND E_Hour = @E_Hour AND E_NIF = @E_NIF AND E_ID = @medicineID ))
    Begin
        UPDATE LarIdosos.TAKING SET Dose = @Dose Where E_day = @E_Day
        AND E_Hour = @E_Hour AND E_NIF = @E_NIF AND E_ID = @medicineID;
    END
    ELSE
    BEGIN
        INSERT INTO LarIdosos.TAKING values(
            @E_Day,
            @E_Hour,
            @E_NIF,
            @medicineID,
            @Dose,
            @Disable
        )
    END
    RETURN
END
```

- A Stored Procedure **sp\_updatePATIENT** faz o update dos dados do paciente.

```

CREATE PROCEDURE [dbo].[sp_updatePATIENT]
    @NIF      varchar(9),
    @Name     varchar(30),
    @Sex      varchar(1),
    @Phone    INT,
    @Age      INT,
    @Check_in DATE,
    @Check_out DATE,
    @Authorization_to_leave BIT,
    @E_BedNumber INT,
    @Entry_Date DATE,
    @Exit_Date DATE
AS
BEGIN
    BEGIN TRANSACTION;
    BEGIN TRY
        UPDATE LarIdosos.PATIENT
        SET
            NIF = @NIF,
            Name = @Name,
            Sex = @Sex,
            Phone = @Phone,
            Age = @Age,
            Check_in = @Check_in,
            Check_out = @Check_out,
            Authorization_to_leave = @Authorization_to_leave,
            E_BedNumber = @E_BedNumber,
            Entry_Date = @Entry_Date,
            Exit_Date = @Exit_Date
        WHERE NIF=@NIF
    END TRY

    BEGIN CATCH
        SELECT
            ERROR_NUMBER() AS ErrorNumber
            ,ERROR_SEVERITY() AS ErrorSeverity
            ,ERROR_STATE() AS ErrorState
            ,ERROR_PROCEDURE() AS ErrorProcedure
            ,ERROR_LINE() AS ErrorLine
            ,ERROR_MESSAGE() AS ErrorMessage;

        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
    END CATCH;
    IF @@TRANCOUNT = 0
        COMMIT TRANSACTION;
END

```

- A Stored Procedure **sp\_Visit** insere uma nova visita no registo com visitante e paciente (visitado) associados.

```
CREATE PROCEDURE dbo.sp_Visit (@Date DATETIME, @NIF varchar(9), @CC varchar(9))
AS
BEGIN
    declare @tmp int;
    if (Exists(Select NIF from LarIdosos.PATIENT where NIF=@NIF ))
        if (Exists(Select CC from LarIdosos.VISITOR where CC=@CC ))
            INSERT INTO LarIdosos.VISIT values (
                @Date
            )
            Select top 1 @tmp = ID from LarIdosos.VISIT order by ID desc

            INSERT INTO LarIdosos.RECEIVED(E_NIF, E_IDVisit) values(
                @NIF, @tmp
            )
            INSERT INTO LarIdosos.VISITED(E_IDVisit, E_CCVisitor) values(
                @tmp, @CC
            )
    RETURN
end
```

- A Stored Procedure **sp\_newRoom** cria um novo quarto com o número desejado de camas.

```
CREATE PROCEDURE dbo.sp_newRoom(@capacity INT=2, @NewRoomId int OUTPUT)
as
begin
    IF(@capacity < 0 )
    BEGIN
        RAISERROR('Capacity must be a number positive',16,1)
    END
    if(NOT EXISTS(SELECT * FROM LarIdosos.BEDROOM))
    BEGIN
        SET @NewRoomId = 1;
    END
    ELSE
    BEGIN
        Select @NewRoomId = RoomNumber from LarIdosos.BEDROOM order by RoomNumber asc
    END

    insert into LarIdosos.BEDROOM (Capacity) values (@capacity);

    While(@capacity > 0)
    BEGIN
        INSERT INTO LarIdosos.BED (E_RoomNumber) values (@NewRoomId);
        SET @capacity = @capacity - 1;
    END

    RETURN
END
```

- A Stored Procedure **sp\_newType**, cria um novo tipo de trabalhador no lar.

```
CREATE PROCEDURE dbo.sp_newType(@type VARCHAR(30))
as
begin
    DECLARE @ID int;
    SELECT @ID = id from LarIdosos.Type WHERE Designation = @type
    if(@@ROWCOUNT = 0)
    BEGIN
        insert into LarIdosos.Type (Designation,Disable) values (@type,0);
        RETURN;
    END

    UPDATE LarIdosos.Type SET Disable = 0 WHERE id = @ID;
    RETURN;
END
```

- A Stored Procedure **sp\_deleteType**, remove o tipo de trabalhador selecionado.

```
CREATE PROCEDURE dbo.sp_deleteType (@id INT)
AS
BEGIN
    Delete FROM LarIdosos.Type WHERE (id=@id)
    RETURN
end
```

- A Stored Procedure **sp\_deleteDisease**, remove o a Disease da sua tabela de registos associada, neste caso a tabela diagnosed.

```
CREATE PROCEDURE dbo.sp_deleteDisease (@E_Name varchar(30), @E_NIF varchar(9))
AS
BEGIN
    Delete FROM LarIdosos.DIAGNOSED WHERE (E_Name = @E_Name and E_NIF = @E_NIF)
    RETURN
end
go
```



- A Stored Procedure **sp\_updateDependent**, faz o update os dados do dependent.

```
CREATE PROCEDURE dbo.sp_updateDependent
    @E_NIF      varchar(9),
    @CC         varchar(9),
    @Name       varchar(30),
    @Address    VARCHAR(30),
    @Phone      INT,
    @Relationship VARCHAR(30)
AS
BEGIN
    IF (NOT EXISTS (SELECT * FROM LarIdosos.VISITOR WHERE CC = @CC))
    BEGIN
        EXEC sp_newDependent @E_NIF, @Name, @CC, @Phone, @Address, @Relationship;
        RETURN;
    END
    ELSE
    BEGIN
        IF (NOT EXISTS (SELECT * FROM LarIdosos.dependent WHERE E_CC = @CC))
        BEGIN
            UPDATE LarIdosos.VISITOR SET      Name = @Name, Phone = @Phone, Address = @Address WHERE CC=@CC;
            INSERT INTO LarIdosos.DEPENDENT values (@CC,@E_NIF)
            return;
        END
        ELSE
        BEGIN
            UPDATE LarIdosos.VISITOR SET      Name = @Name, Phone = @Phone, Address = @Address WHERE CC=@CC;
            return;
        END
    END
END
```

## Triggers

- O Trigger instead of **deleteDisease** aquando uma tentativa de remoção de um diagnóstico, atualiza a variável **Disable** de modo a que informação não seja apagada, mantendo um histórico, apenas quando **Disable** está a 0 é que a informação é revelada.

```
CREATE TRIGGER deleteDisease ON LarIdosos.DIAGNOSED
INSTEAD OF DELETE
AS
BEGIN
    UPDATE LarIdosos.DIAGNOSED SET Disable = 1
    WHERE (
        E_Name = (SELECT E_Name FROM deleted) and E_NIF = (SELECT E_NIF FROM deleted)
    );
END
GO
```

- O Trigger instead of **deleteAppointment**, do mesmo modo que o **deleteDisease**, atualiza a variável **Disable** na tentativa de remoção'.

```
CREATE TRIGGER deleteAppointment ON LarIdosos.Appointment
INSTEAD OF DELETE
AS
BEGIN
    UPDATE LarIdosos.Appointment SET Disable = 1
    WHERE (
        ID = (SELECT ID FROM deleted)
    );
END
GO
```

- O Trigger instead of **finishShifttoHumanResource** faz set da **FinalDate** na tentativa de remoção do turno de um trabalhador, mantendo a informação dos turnos que este trabalhador já obteve anteriormente.

```
CREATE TRIGGER finishShiftofHumanResource ON LarIdosos.Shift
INSTEAD OF DELETE
AS
BEGIN
    UPDATE LarIdosos.ShiftInstance SET FinalDate = GETDATE()
    WHERE (
        E_IDShift = (SELECT ID FROM deleted)
    );
END
GO
```

- O Trigger instead of **deleteMedicine** têm o mesmo efeito que os anteriores, atualização da variável.

```
CREATE TRIGGER deleteMedicine ON LarIdosos.TAKING
INSTEAD OF DELETE
AS
BEGIN
    UPDATE LarIdosos.TAKING SET Disable = 1
    WHERE (
        E_Hour = (SELECT E_Hour FROM deleted) and
        E_NIF = (SELECT E_NIF FROM deleted) and
        E_Day = (SELECT E_Day FROM deleted) and
        E_ID = (SELECT E_ID FROM deleted)
    );
END
GO
```

- O Trigger instead of **deleteType** coloca o tipo de trabalhador 'eliminado' como disabled.

```
CREATE TRIGGER deleteType ON LarIdosos.Type
INSTEAD OF DELETE
AS
BEGIN
    UPDATE LarIdosos.Type SET Disable = 1
    WHERE (
        ID = (SELECT ID FROM deleted)
    );
END
GO
```





# Interface Gráfica

A interface gráfica está dividida em diferentes seções, Patients, Human Resources, Appointments, Visits e Manage Nursing Home.

## Patients

Nesta seção são listados todos os pacientes presentes ou que já frequentaram o lar, é possível inserir e fazer update de um paciente, a sua remoção não o eliminará, apenas o tornará invisível perante as pesquisas.

A pesquisa permite diversas associações entre parâmetros distintos, por exemplo, procurar um paciente do sexo feminino que possua autorização para sair e cujo quarto seja X e também permite ordenação por atributos (Order by).

**Nursing Home Manager**

Patients

Human Resources

Appointments

Visits

Manage Nursing Home

### Patients List

Search Options:

Patient Name:

Patient Nif:

Patient Phone:

Patient Room:

Authorization to leave: Yes ☐ No ☐

sex: ☐ f ☐ m ☐

Order By: 

Date

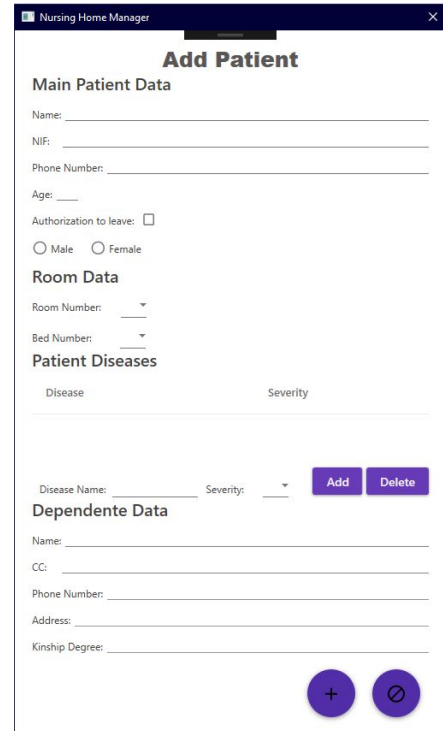
DESC

Show Only Patients with no Checkout: ☒

Name	Nif	Phone	Age	Sex	Authorization	BedNumber	RoomNumber	Checkin	Checkout	EntryDate	ExitDate
Joana Silva	244162719	965566771	78	f	True	9	4	09/06/2017		09/06/2017	
Judite Nascimento	244545778	964555887	76	f	True	12	4	09/06/2017		09/06/2017	

## Adicionar Paciente

Ao adicionar um Paciente é necessário inserir toda a informação associada como por exemplo, o seu dependent/responsável, NIF, Nome, entre outros.



**Nursing Home Manager**

**Add Patient**

**Main Patient Data**

Name: \_\_\_\_\_

NIF: \_\_\_\_\_

Phone Number: \_\_\_\_\_

Age: \_\_\_\_\_

Authorization to leave: ☐

☐ Male ☐ Female

**Room Data**

Room Number: \_\_\_\_\_

Bed Number: \_\_\_\_\_

**Patient Diseases**

Disease	Severity

Disease Name: \_\_\_\_\_ Severity: \_\_\_\_\_ **Add** **Delete**

**Dependente Data**

Name: \_\_\_\_\_

CC: \_\_\_\_\_

Phone Number: \_\_\_\_\_

Address: \_\_\_\_\_

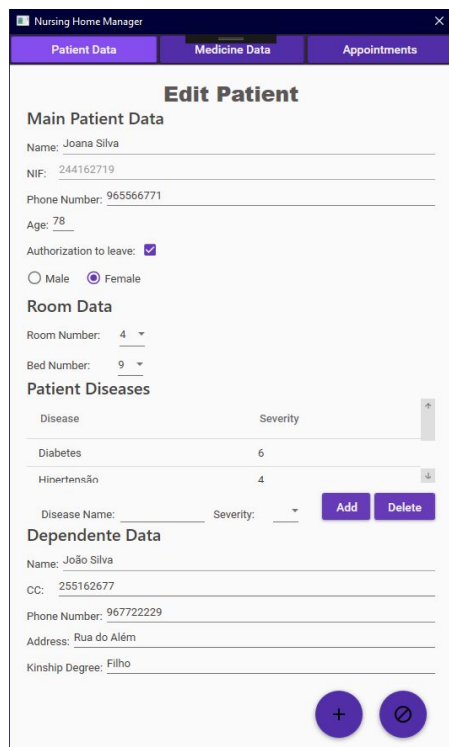
Kinship Degree: \_\_\_\_\_

**+** **⊗**

## Editar Paciente

A página do paciente está dividida em 3 seções, Patient Data, Medicine Data e Appointments.

Quando é feito duplo clique num paciente temos acesso á primeira seção da página do paciente onde podemos ver as suas informações e editá-las. É possível a inserção de Diseases, o seu grau de severidade e também a sua remoção.



**Nursing Home Manager**

**Edit Patient**

**Main Patient Data**

Name: Joana Silva

NIF: 244162719

Phone Number: 965566771

Age: 78

Authorization to leave: ☒

☐ Male ☒ Female

**Room Data**

Room Number: 4

Bed Number: 9

**Patient Diseases**

Disease	Severity
Diabetes	6
Hinertensão	4

Disease Name: \_\_\_\_\_ Severity: \_\_\_\_\_ **Add** **Delete**

**Dependente Data**

Name: João Silva

CC: 255162677

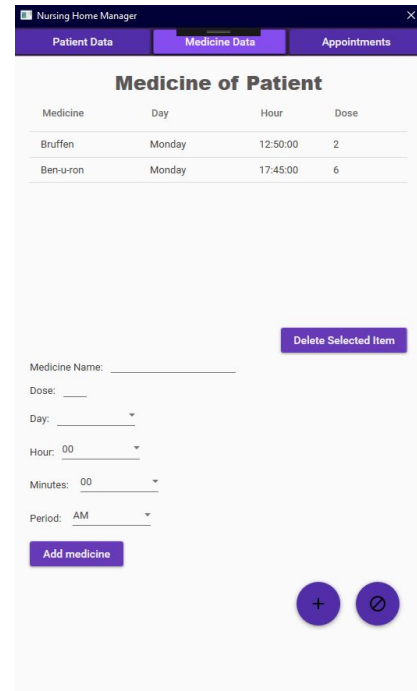
Phone Number: 967722229

Address: Rua do Além

Kinship Degree: Filho

**+** **⊗**

Na segunda seção, Medicine Data, destina-se à informação relativa à toma de medicamentos, qual o medicamento, a sua dose e o horário da toma. Podemos ainda adicionar um novo medicamento e o seu horário de toma e por fim remover uma toma previamente definida.



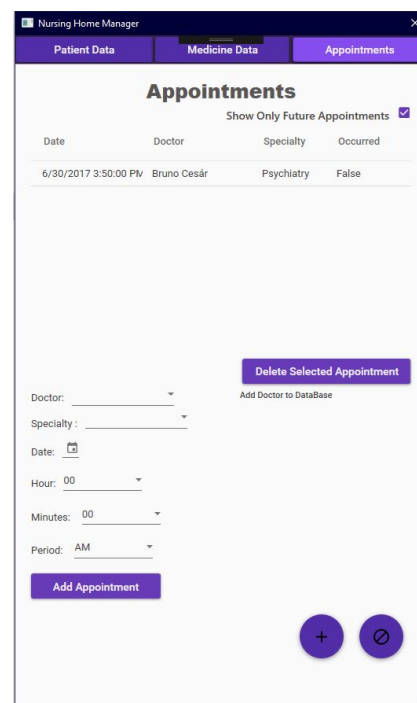
The screenshot shows the 'Medicine of Patient' section of the Nursing Home Manager application. It features a table with columns for Medicine, Day, Hour, and Dose. Below the table are input fields for adding a new medicine, including Medicine Name, Dose, Day, Hour, Minutes, and Period. There are also buttons for 'Delete Selected Item' and 'Add medicine', and a plus icon for adding new items.

Medicine	Day	Hour	Dose
Bruffen	Monday	12:50:00	2
Ben-u-ron	Monday	17:45:00	6

Medicine Name: \_\_\_\_\_  
Dose: \_\_\_\_\_  
Day: \_\_\_\_\_  
Hour: 00  
Minutes: 00  
Period: AM  
Add medicine  
Delete Selected Item  
+

Na terceira seção podemos obter informação relativo às consultas marcadas, como a data da consulta, o doutor, especialidade e saber se esta já ocorreu, isto é, se o paciente já foi à consulta em questão.

Para além de consultar toda a informação das consultas é possível a remoção e inserção de novas consultas.



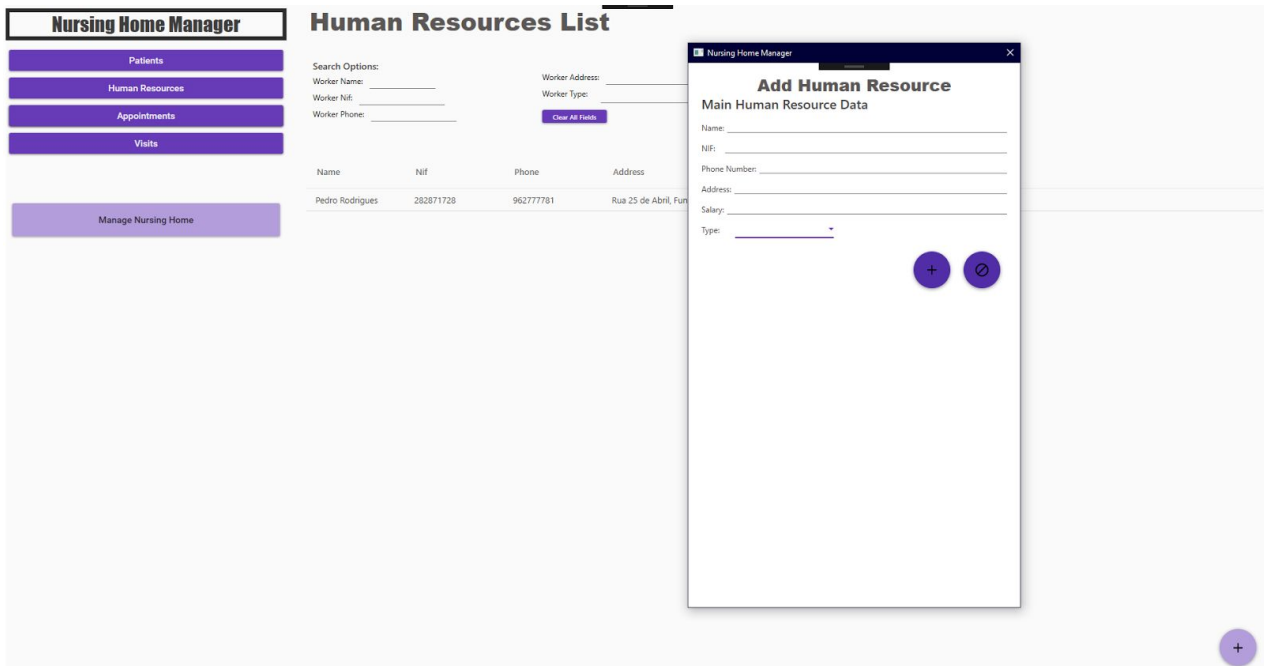
The screenshot shows the 'Appointments' section of the Nursing Home Manager application. It features a table with columns for Date, Doctor, Specialty, and Occurred. Below the table are input fields for adding a new appointment, including Doctor, Specialty, Date, Hour, Minutes, and Period. There are also buttons for 'Delete Selected Appointment' and 'Add Appointment', and a plus icon for adding new items.

Date	Doctor	Specialty	Occurred
6/30/2017 3:50:00 PM	Bruno Cesar	Psychiatry	False

Doctor: \_\_\_\_\_  
Specialty: \_\_\_\_\_  
Date: \_\_\_\_\_  
Hour: 00  
Minutes: 00  
Period: AM  
Add Appointment  
Delete Selected Appointment  
Add Doctor to DataBase  
+

## Human Resources

A segunda seção é a Human Resources destinada a informação relativa aos trabalhadores do lar, é possível pesquisar e inserir novos trabalhadores no lar, cada trabalhador terá um type associado, por exemplo Cozinheiro, Auxiliar, entre outros.



The screenshot displays the 'Nursing Home Manager' application interface. On the left, a sidebar menu includes 'Patients', 'Human Resources', 'Appointments', 'Visits', and 'Manage Nursing Home'. The main area is titled 'Human Resources List' and features search filters for Worker Name, NIF, Address, and Type, along with a 'Clear All Fields' button. Below the filters is a table with one entry: Pedro Rodrigues, NIF 282871728, Phone 962777781, and Address Rua 25 de Abril, Fun. An 'Add Human Resource' modal is open, showing fields for Name, NIF, Phone Number, Address, Salary, and Type, with '+' and 'x' buttons at the bottom. A floating '+' button is also visible in the bottom right corner.

Name	Nif	Phone	Address
Pedro Rodrigues	282871728	962777781	Rua 25 de Abril, Fun

## Appointments

Esta seção serve para pesquisar consultas, os filtros de pesquisa permitem procurar consultas de um determinado Doutor, Paciente, especialidade, entre outros.

**Nursing Home Manager**

- Patients
- Human Resources
- Appointments
- Visits

Manage Nursing Home

### Appointments List

Search Options:  
Doctor Name:   
Doctor NIF:   
Patient Name:

Patient NIF:   
Specialty:   
Date:

Order By: Date  DESC   
Show Only Future Appointments ☒

Patient Name	Patient NIF	Doctor Name	Doctor NIF	Specialty	Date	Occurred
Joana Silva	244162719	Bruno Cesar	291182818	Psychiatry	6/30/2017 3:50:00 PM	False

## Visits

Esta seção destina-se para pesquisar registo de visitas, os filtros de pesquisa permitem procurar visitas de um determinado visitante (quem visitou), paciente (quem o visitou), data da visita, entre outros.

**Nursing Home Manager**

- Patients
- Human Resources
- Appointments
- Visits

Manage Nursing Home

### Visits List

Search Options:  
Patient Name:   
Patient NIF:   
Visitor Name:   
Visitor CC:

Visitor Phone:   
Date:

Patient Name	Patient NIF	Visitor Name	Visitor CC	Date	Visitor
Joana Silva	244162719	255162677	6/9/2017 5:53:25 PM	96772	

**Nursing Home Manager**

### Add Visit

Find Patient ☒

Find Visitor ☐

Patient Name:   
Patient NIF:   
Patient Phone:   
Room Number:

Name	NIF	Phone
Joana Silva	244162719	965566771

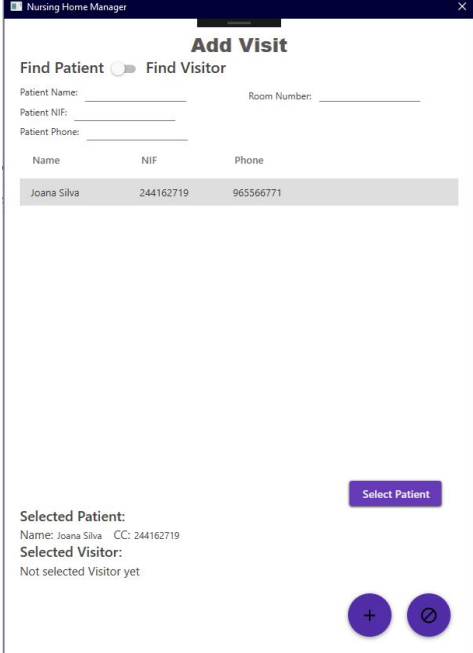
Selected Patient:  
Name: Joana Silva CC: 244162719

Selected Visitor:  
Not selected Visitor yet

## Adicionar Visita

Ao adicionar uma nova visita, existe uma opção de procura para facilitar a inserção dos dados, procura pelo paciente e pelo visitor (caso exista registado).

Após a procura do paciente é necessário fazer o select do mesmo, uma vez selected este irá aparecer no campo “Selected Patient” no canto inferior esquerdo.



**Nursing Home Manager**

**Add Visit**

Find Patient ☒ Find Visitor

Patient Name: \_\_\_\_\_ Room Number: \_\_\_\_\_

Patient NIF: \_\_\_\_\_

Patient Phone: \_\_\_\_\_

Name	NIF	Phone
Joana Silva	244162719	965566771

**Selected Patient:**  
Name: Joana Silva CC: 244162719

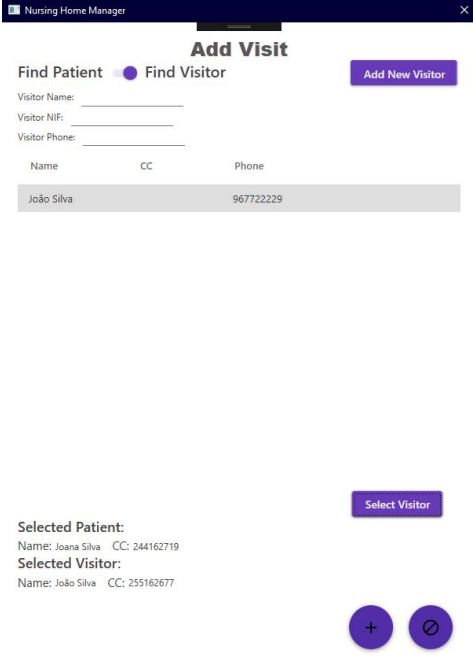
**Selected Visitor:**  
Not selected Visitor yet

Select Patient

+

-

Da mesma forma, após a procura do Visitor é necessário fazer o select do mesmo, caso este exista. Uma vez selected este irá aparecer no campo “Selected Visitor” no canto inferior esquerdo.



**Nursing Home Manager**

**Add Visit**

Find Patient ☐ Find Visitor ☒ **Add New Visitor**

Visitor Name: \_\_\_\_\_

Visitor NIF: \_\_\_\_\_

Visitor Phone: \_\_\_\_\_

CC: \_\_\_\_\_

Name	CC	Phone
João Silva		967722229

**Selected Patient:**  
Name: Joana Silva CC: 244162719

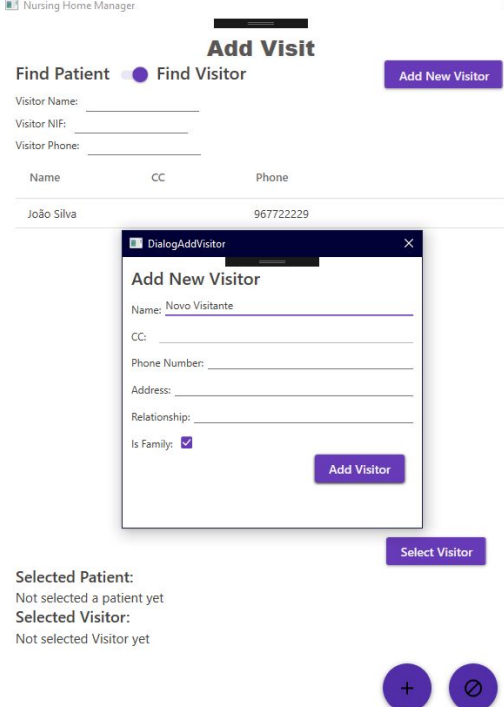
**Selected Visitor:**  
Name: João Silva CC: 255162677

Select Visitor

+

-

Caso o visitor não exista, ou seja, não esteja registado, é dada a opção de inserir um novo Visitor na qual é necessário preencher toda a informação associada e por fim concluir o registo da visita.



## Manage Nursing Home

Por fim, esta última seção destina-se à administração e gestão do lar, possui as seguintes funcionalidades, inserir novos quartos com a capacidade desejada (número de camas) e inserir novos tipos de trabalhadores do lar.

