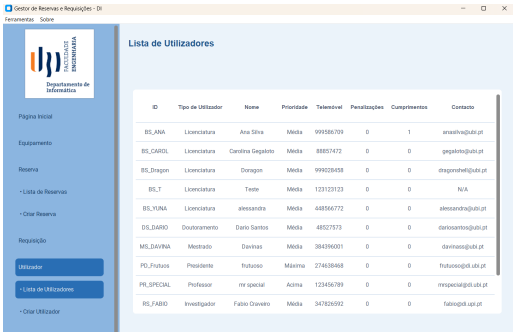
Universidade da Beira Interior

Licenciatura em Engenharia Informática

Bases de Dados – Trabalho Prático



Elaborado por:

Alessandra Delgado, N.º 51713
Carolina Gegaloto, N.º 51715
Ana Silva, N.º 52050

Docentes:

Professor João Muranho
Professor Rui Cardoso

6 de janeiro de 2025

Resumo

O presente trabalho foi desenvolvido para a unidade curricular Bases de Dados, e teve como intuito a aplicação dos conhecimentos adquiridos ao longo da mesma. Este trabalho consiste no desenvolvimento de uma aplicação de gestão de recursos de uma instituição, mais concretamente a reserva e requisição dos mesmos. O trabalho culminou na modelação de uma base de dados normalizada, com recurso às ferramentas Structured Query Language *(SQL) Server Management Studio* (SSMS) e *Microsoft Server SQL* (MSSQL), implementação de `triggers` e `procedures` para garantir as regras de negócio, e uma aplicação cliente desenvolvida na linguagem de programação `python`, com recurso à biblioteca gráfica `customtkinter`, para fácil utilização por parte do utilizador.

O relatório aborda, a motivação por detrás do desenvolvimento deste trabalho, o processo de modelação e a sua evolução, a implementação das diversas funcionalidades propostas e extras, e uma reflexão do trabalho desenvolvido e propostas para uma continuação futura.

Palavra-chave: Modelação, normalização, `procedures`, `python`, `triggers`.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Excertos de Código

Acrónimos

DG *DataGrip*

IP *Internet Protocol*

PC *PyCharm*

2FN Segunda Forma Normal

3FN Terceira Forma Normal

IDE *Integrated Development Environment*

PDF *Portable Document Format*

MVC *Model-View-Controller*

SQL *Structured Query Language*

SGBDR Sistema De Gestão de Base de Dados Relacional

SSMS *SQL Server Management Studio*

T-SQL *Transact Structured Query Language*

MSSQL *Microsoft Server SQL*

Capítulo

1

Introdução

1.1 Enquadramento

Este trabalho enquadra-se na unidade curricular Bases de Dados, que remete para o ensino de armazenamento, manipulação de dados contidos numa base de dados, tal como a estruturação da anteriormente referida.

1.2 Objetivos

Com a realização deste trabalho teve-se como objetivo implementar uma base de dados funcional, juntamente de uma aplicação capaz de estabelecer uma ponte entre si e a base de dados, de modo a inserir reservas por meio desta. No que toca à implementação da base de dados, também foi necessário implementar as suas regras de negócio e transações. Finalmente, também implementámos as *views* da base de dados, para facilitar a visualização de dados comummente utilizados cuja visualização somente poderia ser possível através de *queries* complexas.

1.3 Motivação

O trabalho foi desenvolvido como método de avaliação na unidade curricular, no intuito de desenvolver uma base de dados, juntamente com uma aplicação externa em `python` para uma mais fácil gestão das reservas.

1.4 Organização do Documento

Este relatório encontra-se dividido em vários capítulos. O primeiro capítulo, Introdução, dá uma breve descrição do trabalho, o motivo da sua realização e os seus objetivos.O segundo capítulo, Desenvolvimento da aplicação cliente/servidor sobre bases de dados, contém informações acerca da aplicação desenvolvida, relativamente ao `SQL Server`, à configuração do acesso ao servidor e acerca a aplicação desenvolvida em `python`. O terceiro capítulo, Modelação, visa mostrar o desenvolvimento do modelo Entidade Relação juntamente das decisões tomadas durante a sua conceção. O capítulo sucedente, Aplicação, remete a uma descrição mais detalhada do desenvolvimento do trabalho, inclusive tarefas desenvolvidas por cada elemento, tal como documentação relevante. O último capítulo serve de conclusão do relatório.

Capítulo

2

Tecnologias e Ferramentas Utilizadas

2.1 Introdução

Este capítulo irá descrever as ferramentas utilizadas no desenvolver deste trabalho.

2.2 Ferramentas

2.2.1 *Transact Structured Query Language (T-SQL)*

T-SQL é extensão proprietária da *Microsoft* e da *Sybase* de *Structured Query Language* (SQL) usada para interagir com bases de dados relacionais. Esta linguagem expande a SQL de forma a incluir programação *procedural*, variáveis locais e suporte de funções para o processamento de *strings*, processamento de datas e funções matemáticas.

2.2.2 *Linguagem Python*

Python é uma linguagem de programação de alto nível interpretada orientada a objetos, imperativa e funcional, de tipagem forte e dinâmica. Lançada por Guido van Rossum em 1991, esta linguagem dá prioridade à legibilidade do código sobre a velocidade ou expressividade, combinando uma sintaxe concisa e clara com os recursos da biblioteca *standard* e por módulos e *frameworks* desenvolvidos por terceiros.

2.2.3 *Microsoft Server SQL* (MSSQL)

Microsoft SQL Server é um Sistema De Gestão de Base de Dados Relacional (SGBDR) desenvolvido pela *Microsoft*. Como um servidor de bases de dados, a sua principal função é a de guardar e fornecer informação de acordo com os pedidos realizados por outras aplicações.

2.2.4 *SQL Server Management Studio* (SSMS)

O SSMS é um *software* que permite configurar, administrar e gerenciar todos os componentes do MSSQL. Esta ferramenta inclui editores para *scripts* e ferramentas gráficas que trabalham com objetos e recursos do servidor.

2.2.5 *Git*

Git é um sistema de controle de versões distribuído usado no desenvolvimento de *software*, desenvolvido inicialmente por Linus Torvalds para o *kernel Linux*.

2.2.6 *GitHub*

O GitHub é uma plataforma que permite a disponibilização *online* de repositórios de versões de controlo *Git*. É atualmente o santo graal na distribuição de código aberto na *internet*.

2.2.7 *GitKraken*

A plataforma *GitKraken* é uma ferramenta que disponibiliza uma interface gráfica para o utilizador, dessa forma facilitando a interação com repositórios *Git*, ao invés da interação por linha de comandos.

2.2.8 *Overleaf*

Overleaf é um editor de LaTeX que executa no navegador, e que permite a partilha e edição colaborativa em tempo real de documentos. Esta ferramenta automatiza o processo de compilação e produção de ficheiros do tipo *Portable Document Format* (PDF) a partir de ficheiros do tipo TeX.

2.2.9 *PyCharm* (PC)

PyCharm é um ambiente de desenvolvimento integrado (*Integrated Development Environment* (IDE)) para programar em *Python*, desenvolvido pela *JetBrains*, que permite a conexão com bases de dados.

2.2.10 Tkinter

Tkinter é a biblioteca padrão do *Python* para criar interfaces gráficas para o utilizador, leve, rápida e *cross-platform*.

2.2.11 CustomTkinter

CustomTkinter é uma biblioteca que expande o módulo padrão *Tkinter* para criar interfaces gráficas modernas e mais estilizadas, incluindo temas, suporte a widgets personalizados e uma aparência mais refinada, que o *Tkinter* não oferece.

2.2.12 *DataGrip* (DG)

Datagrip é um ambiente de desenvolvimento integrado (IDE) desenvolvido pela *JetBrains*, que, tal como o PC, permite a conexão com bases de dados. Com a opção para programar em várias extensões de SQL, incluindo T-SQL, o *DataGrip* é direcionado para a gestão bases de dados.

2.3 Conclusões

Este capítulo descreveu de forma detalhada as tecnologias e ferramentas utilizadas durante o desenvolvimento deste trabalho. No próximo capítulo será discutida detalhadamente a construção da base de dados.

Capítulo

3

Desenvolvimento da Aplicação Cliente/Servidor Sobre Bases de Dados

3.1 Introdução

Ao longo deste capítulo, serão expostas mais informações relativamente à aplicação cliente/servidor desenvolvida em *Python*, esta que permite a criação de reservas, requisições, utilizadores e equipamentos de forma intuitiva.

3.2 Aplicação Cliente/Servidor

A aplicação desenvolvida serve como uma ponte entre o utilizador e a base de dados, permitindo a inserção e visualização dos dados existentes na mesma. Esta aplicação está estruturada de acordo com a abstração *Model-View-Controller* (MVC) 3.1, que permite uma divisão em três partes lógicas, cada uma com um propósito único.

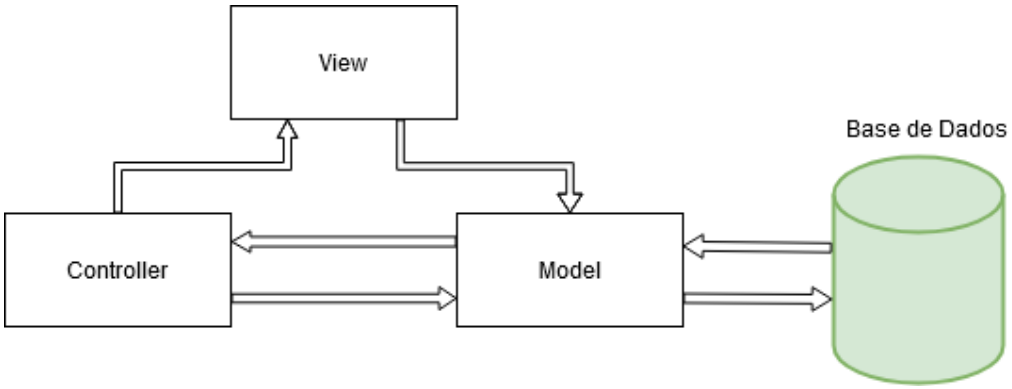


Figura 3.1: Estrutura MVC.

A aplicação está dividida em múltiplas diretorias de forma a cumprir com esta estrutura.

- Na diretoria *models* estão presentes os ficheiros que contêm as funções de acesso e inserção na base de dados, para as diferentes funcionalidades existentes;
- Na diretoria *views* estão presentes os ficheiros relacionados com a *front-end* da aplicação, divididos em *frames*, *nav*, *widgets* e *images*;
- Na diretoria *controllers* só existe um ficheiro com uma caixa de diálogo (`about.py`), uma vez que todas as regras de negócio da aplicação na realidade está implementada na base de dados, não tendo sido necessário implementar a *back-end* da aplicação.

3.3 Configuração do Acesso ao Servidor

O acesso ao servidor de bases de dados pela aplicação cliente é realizado pelo ficheiro de código *Database.py*, com os métodos *connect* e *close*, sendo estes responsáveis pela conexão e pelo fecho da mesma, respetivamente. Este ficheiro de código tira proveito da biblioteca *pyodbc* para tratar das ligações à base de dados.

A aplicação cliente, pelo menu superior de ferramentas, pode conectar-se à base de dados pela inserção dos seguintes dados:

1. Endereço – O endereço *Internet Protocol* (IP) do servidor de bases de dados;
2. Base de Dados – O nome da base de dados;

3. Utilizador – O nome do utilizador da base de dados;

4. Palavra-passe – A palavra-passe do utilizador da base de dados.

Sendo a conexão terminada via opção disponibilizada no mesmo menu.

O excerto de código 3.1 ilustra a utilização da biblioteca *pyodbc* para se realizar a conexão a uma base de dados. Neste exemplo a base de dados `teste_di` está alojada na própria máquina.

```
def connect():
    return pyodbc.connect(
        "DRIVER={ODBC Driver 17 for SQL Server};"
        "SERVER= localhost;"
        "DATABASE=teste_di;"
        "UID=sa;"
        "PWD=sa;"
    )
```

Excerto de Código 3.1: Exemplificação de uma conexão à base de dados via pyodbc.

3.4 Frontend Python

O *frontend* do `python` foi desenvolvido com recurso à biblioteca gráfica `CustomTkinter` 2.2.11. Esta biblioteca oferece um mecanismo de agrupamento de elementos gráficos(`widget`), as `frames`. Com a utilização de `frames` podemos desenvolver as diversas páginas que compõem a aplicação `python`.

Deste modo, definiu-se o *layout* 3.2 base composto por duas `frames`, uma de navegação e uma que possuiria o conteúdo da página selecionada.

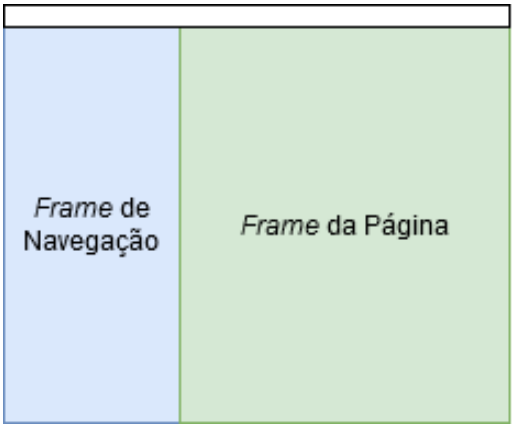


Figura 3.2: *Layout* base da aplicação `python`.

3.4.1 Barra de Navegação

A barra de navegação inclui as secções Página Inicial, Equipamento, Reserva, Requisição e Utilizador. Para cada secção existem duas páginas principais: Listagem e Criar/Adicionar, representadas por botões. Ao se pressionar um destes botões, a aplicação `python` irá renderizar a página correspondente e esconder a atual.

3.4.2 Conteúdo

A página selecionada pelo utilizador é renderizada nesta parte do *layout*. Na aplicação `python`, uma página corresponde a uma `frame`, situada na diretoria *views*. A plataforma é composta pelas seguintes páginas:

- `equipment_index.py` – Listagem de todos os equipamentos registados na base de dados;
- `equipment_new.py` – Responsável pela adição de novos equipamentos à base de dados;
- `reserve_index.py` – Responsável pela listagem das reservas registadas na base de dados;
- `reserve_new.py` – Página responsável pela criação de novas reservas na base de dados;
- `reserve_edit.py` – Formulário responsável pela alteração do estado de uma reserva;
- `requisition_index.py` – Listagem de todas as requisições presentes na base de dados;
- `requisition_new.py` – Formulário responsável pela criação de uma nova requisição na plataforma;
- `requisition_edit.py` – Página responsável pela devolução total/parcial dos equipamentos de uma requisição;
- `user_index.py` – Listagem de todos os utilizadores registados na aplicação;
- `user_new.py` – Responsável pela criação de novos utilizadores.

Cada uma destas páginas possui a implementação do método `reload`, responsável pela limpeza e carregamento da mesma. Este método, é crucial na navegação da plataforma, uma vez que é invocado pela barra de navegação.

O excerto de código 3.2 demonstra o código parcial de um método `reload`, mais concretamente a secção responsável pela limpeza da `frame` para que esta seja reconstruída com os dados atualizados da base de dados.

```
1 def reload(self) -> None:
2     for widget in self.winfo_children():
3         widget.destroy()
4         . . .
```

Excerto de Código 3.2: Implementação parcial do método `reload` de uma `frame`.

Para além deste método, as páginas compostas por formulários possuem também a implementação do método `is_valid` responsável por garantir que os dados inseridos pelo utilizador estejam corretos, por exemplo, a verificação de que uma data esteja no formato correto.

O resultado, é uma aplicação elegante de fácil utilização 3.3.

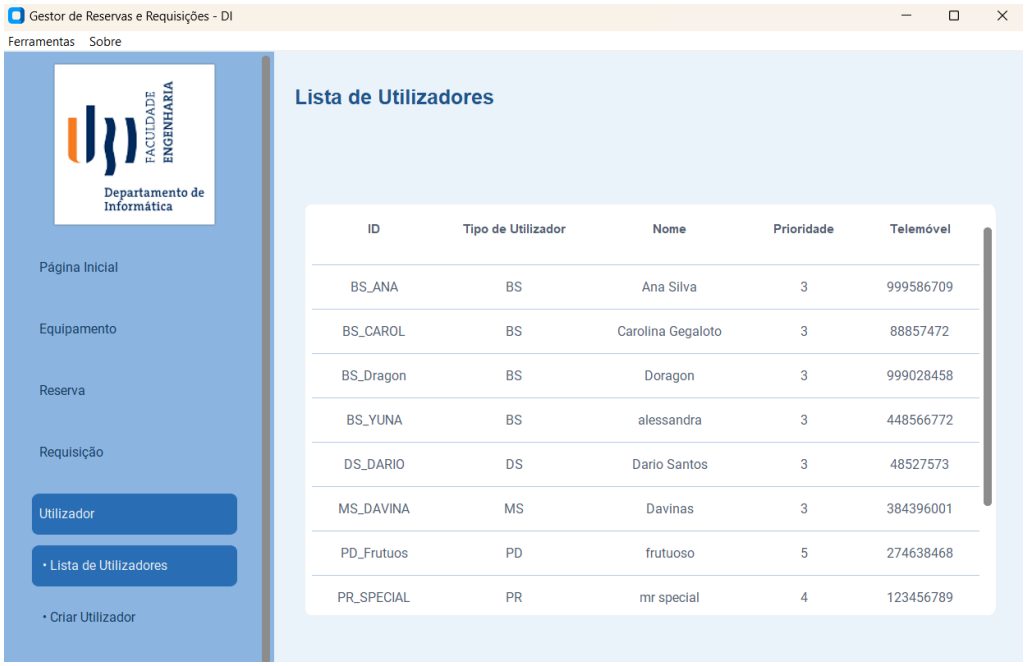


Figura 3.3: Interface gráfica da aplicação desenvolvida.

3.5 Conclusões

Este capítulo descreveu de forma a implementação da aplicação cliente/servidor, desde a sua estrutura à conexão com o servidor de bases de dados. O próximo capítulo irá abordar a modelação da base de dados implementada.

Capítulo

4

Modelação

4.1 Introdução

No presente capítulo será detalhado o processo da criação do modelo entidade-relação do trabalho, juntamente das decisões tomadas durante a sua conceção, assim como a identificação de problemas de ambiguidade ao longo da sua evolução.

4.2 Descrição da Organização

A base de dados desenvolvida segue o esquema relacional da figura 4.5. Este dispõe de uma coleção de tabelas de modo a representar o problema proposto no enunciado, normalizadas segundo a norma Terceira Forma Normal (3FN). As tabelas que constituem a base de dados podem ser agrupadas em quatro grupos, as tabelas referentes ao utilizador, à reserva, à requisição e ao equipamento.

Na sua tua totalidade, as tabelas que constituem a base de dados são as seguintes:

- `TblContact`;
- `TblUser_DI`;
- `TblUser_Priority`;
- `TblPriority_Map`.
- `TblEquipment`;

- `TblReservation`;
- `TblRes_Equip`;
- `TblRequisition`;
- `TblReq_Equip`;
- `TblDevolution`;

4.2.1 Tabelas Representativas do Utilizador

O utilizador é representado diretamente pela tabela `TblUser_DI`. Porém, devido ao facto do correio eletrónico ser opcional, este é armazenado na tabela `TblContact`. A tabela utilizador possui também a informação relacionada à prioridade do utilizador, as suas faltas e os respetivos cumprimentos. Devido ao utilizador possuir uma prioridade volátil, dependendo do cumprimento da devolução dos equipamentos, relacionada inicialmente ao seu estatuto, foram utilizadas duas tabelas auxiliares na criação de um utilizador.

A tabela `User_Priority` mapeia o tipo de utilizador a uma respetiva prioridade base, de forma a quando for inserido um utilizador, o mesmo seja atribuído uma prioridade concordante.

A tabela `Priority_Map` atribui a cada nível de prioridade um número inteiro de forma a facilitar as comparações entre utilizadores, como, por exemplo, quem ficaria com um determinado equipamento caso o quisessem na mesma hora, em vez de ter de comparar `string`, utilizamos antes números inteiros podendo só verificar qual dos números é maior e obtemos uma resposta, o que permite evitar linhas de código desnecessárias melhorando a legibilidade do código. A tabela 4.1 ilustra os tipos de utilizador que existem na plataforma e a sua codificação. É de notar que foi adicionado o tipo de utilizador presidente, este utilizador possui prioridade máxima, não sofre penalizações e não é limitado pela preempção.

Codificação	Cargo	Prioridade Inicial
BS	Licenciatura	3
DS	Doutoramento	3
MS	Mestrado	3
PD	Presidente	5
PR	Professor	4
RS	Investigador	3
SF	Apoio	3
XT	Externo	3

Tabela 4.1: Cargos existentes na plataforma, e as suas respetivas prioridades iniciais.

É de notar que, para além dos tipos indicados no enunciado, foi adicionado o tipo 'PD' para representar um utilizador do tipo presidente, de forma a facilitar o cumprimento das regras de negócio.

4.2.2 Tabela do Equipamento

O equipamento é representado pela tabela `TblEquipment`, que possui uma categoria, uma descrição para cada equipamento e um estado. O estado é utilizado pelos `triggers` e `procedures` para se filtrar os equipamentos nas diferentes listas da plataforma, e uma associação correta dos equipamentos pelas reservas.

Estado
Available
Reserved
InUse

Tabela 4.2: Estados existentes para um equipamento na base de dados.

Categoria
Peripherals
Computer
Projector
Stationery
Other

Tabela 4.3: Categorias existentes para um equipamento na base de dados.

4.2.3 Tabelas Respetivas da Reserva

Na aplicação desenvolvida, um utilizador pode realizar várias reservas cada uma com N equipamentos associados. Sendo esta uma relação muitos para muitos (vários equipamentos podem estar em várias reservas), esta relação

foi representada com auxílio de uma tabela de modo a se conservar a normalização 3FN. Assim sendo, uma reserva é representada com auxílio a duas tabelas:

- `TblReservation` – responsável pelos atributos gerais da reserva;
- `TblRes_Equip` – responsável por relacionar os equipamentos à reserva.

Tomemos a seguinte situação, descrita na tabela 4.4 (onde as datas e a coluna de atribuição foram omitidas), como exemplo para perceber a normalização da tabela `TblReservation`.

id_reserv	id_equip	id_user	reg_date	time_start	time_end	satus_res
2025001	1	PR_SPECIAL	...	...	...	Active
2025001	2	PR_SPECIAL	...	...	...	Active
2025001	3	PR_SPECIAL	...	...	...	Active

Tabela 4.4: Estados existentes para um equipamento na base de dados.

Como se consta na tabela, a mesma reserva trata de 3 equipamentos diferentes. Certamente, poderíamos tratar de `id_reserv` e `id_equip` como uma chave primária. Nesse caso, haveria o conjunto repetitivo {`id_user`, `reg_date`, `time_start`, `time_end`, `satus_res`}. Isto acontece, pois a dependência funcional `id_reserv` → `id_equip` é parcial. Esta tabela deve, então, ser normalizada para Segunda Forma Normal (2FN), colocando numa tabela à parte (`TblRes_Equip`) os atributos {`id_reserv`, `id_equip`, `assigned_to`}.

Sendo o *id* da reserva crescente e reiniciado todos os anos, foi desenvolvida a *procedure* `MakeId`, auxiliando-se com a tabela `Res_SeqId` que possui os atributos `current_year`, ano correspondente de uma sequência, e `current_seq`, que contém o número da última reserva criada para cada ano. A `MakeId` recebe esses dois atributos como parâmetros para determinar o *id* de novas reservas, para que a cada ano, a sequência reinicie.

A tabela `TblReservation` também possui uma relação estrangeira com a tabela `TblUser_DI`, uma data de início, uma data de fim, uma data de registo e um estado. As datas são utilizadas em conjunto para auxílio na atribuição dos equipamentos pelas reservas, aquando a existência de várias reservas que partilhem total ou parcialmente dos mesmos equipamentos. A coluna estado possuí informação em relação ao estado atual da reserva, por exemplo, se esta encontra-se satisfeita ou em espera. A tabela 4.5 ilustra os múltiplos estados que esta pode apresentar. Para além dos estados pedidos no enunciado do trabalho prático, ao longo de diversas implementações sentiu-se a necessidade de permitir mais um estado para a reserva – *NotSatisfied*.

Estado	Final?
Active	Não
Waiting	Não
Cancelled	Sim
Forgotten	Sim
Satisfied	Sim
NotSatisfied	Sim

Tabela 4.5: Estados existentes para uma reserva na base de dados.

A tabela `TblRes_Equip`, para além da associação dos equipamentos a uma reserva, possui também o atributo `assigned_to` que indica se o equipamento especificado está atribuído, ou não, à reserva. Isto, pois como algumas reservas podem sobrepor-se umas as outras enquanto não forem satisfeitas, e como a tabela `TblReservation` também contém reservas antigas, é necessário ter um campo que indique quem é a próxima pessoa que vai ter o equipamento para requisitar. O segundo atributo informativo, o `essential` indica se um equipamento é essencial ou não para esta reserva.

4.2.4 Tabelas Respetivas da Requisição

A criação da tabela `TblReq_Equip` visa a normalização da tabela `TblRequisition`, tendo como base o mesmo exemplo apresentado na tabela 4.4, uma vez que, no caso da requisição, também a dependência funcional `id_req` → `id_equip` é parcial (vários equipamentos em várias requisições).

Para além do que é explicito no enunciado, percebeu-se que as devoluções dos equipamentos não seriam tratadas na tabela `TblReq_Equip`, tendo sido criada a `TblDevolution`. Esta tabela só guarda o registo dos equipamentos devolvidos e de qual requisição se trata, sendo os seus atributos {`id_req`, `id_equip`, `reg_date`}, com chave primária {`id_req`, `id_equip`}, sendo cada uma uma chave estrangeira.

Desta forma, a tabela `TblDevolution` só vai conter os equipamentos que já foram devolvidos, utilizando o atributo `return_date` para guardar o dia e a hora de quando foi devolvido. Optamos por fazer uma nova tabela em vez de inserir o atributo na tabela `TblReq_Equip`, pois existem alguns equipamentos que enquanto não forem devolvidos estariam com esse campo a `null` por tempo indefinido, que não está conforme as regras de normalização de bases de dados.

Esta abordagem simplifica a implementação de `triggers` relativos a faltas no que toca ao retorno tardio de equipamentos. Também são importantes

para a determinação do estado de uma reserva, sendo importantes para a *procedure* `DetermineStatus`.

4.3 Modelo de dados

Inicialmente, a partir de uma primeira leitura do enunciado, deduziu-se o seguinte diagrama entidade-associação para o projeto (fig. 4.1). Num estado inicial as devoluções não eram consideradas ainda, nem o utilizador era relacionado diretamente com a requisição, nem era considerado o tempo da requisição, gerando ambiguidades como "Qual utilizador realizou requisição X?", "Quando é que o utilizador Y deve devolver equipamento Z"? – Para além disso, era apenas considerado que o utilizador pudesse apenas levantar um equipamento, o que não está consoante a lógica da reserva de então.

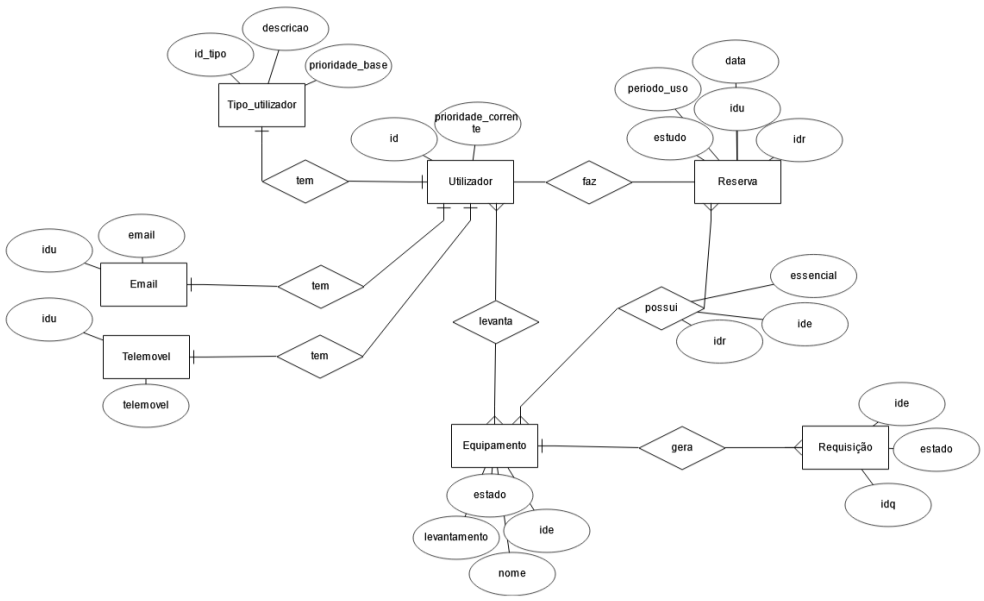


Figura 4.1: Primeira tentativa de modelação da base de dados.

Após as primeiras implementações, percebeu-se o erro cometido com a relação entre o utilizador e a requisição, e os equipamentos da requisição, dando origem a um novo diagrama (fig. 4.2).

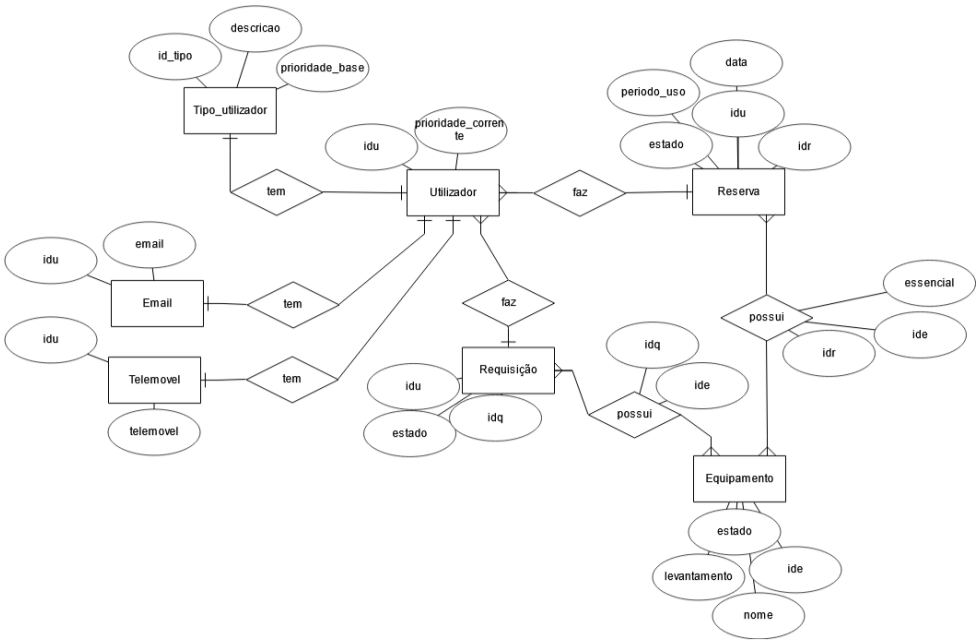


Figura 4.2: Segunda tentativa de modelação da base de dados.

Na versão a seguir, o período de duração da requisição é considerado; e um utilizador, tal como é explícito no enunciado, tem obrigatoriamente um número de telemóvel, dando origem ao diagrama 4.3. O atributo 'levantamento' de um equipamento desaparece, pois já o estado do equipamento `InUse` diz-nos que um equipamento foi levantado. Em contrapartida, adicionada-se o atributo de categoria ao equipamento, crucial para futuras filtragens na aplicação. Percebe-se também que o período de uma reserva deve ser tratado como duas datas (início e fim) e não como um simples período.

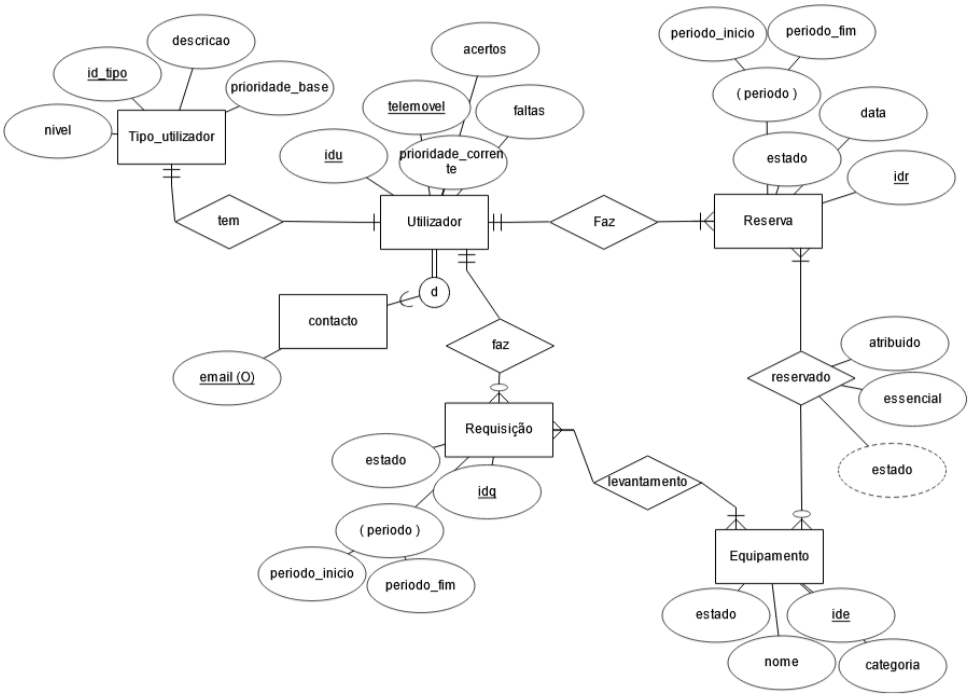


Figura 4.3: Terceira tentativa de modelação da base de dados.

Considerou-se ainda em colocar o estado da reserva na relação proposta entre os equipamentos e as reservas, por falta de noção de como os equipamentos deveriam ser atribuídos, até então. Percebeu-se também, que as penalizações de um utilizador estão diretamente ligadas ao mesmo, tendo sido adicionados os atributos 'faltas' e 'acertos' ao diagrama. Com as últimas implementações, percebeu-se que seria conveniente guardar a prioridade de um utilizador como um número inteiro, para fáceis incrementações e decrementações, tendo-se adicionado o atributo 'nível' para substituir a 'prioridade_base' (uma linha de caracteres, até então), e os atributos do tipo prioridade serem guardados como inteiros.

No entanto, esta alteração não foi suficiente. Como seria possível aceder às prioridades 'Mínima' e 'Abaixo da Média' se, na verdade, nenhum utilizador por defeito contém nenhuma dessas prioridades? Além disso, como seria possível saber que equipamento foi devolvido e de que requisição pertencia? Como seria possível saber que penalização aplicar a um utilizador após a devolução de um equipamento? A colocação destas questões foi um passo crucial para o crescimento para o diagrama 4.4, e para as implementações finais na base de dados.

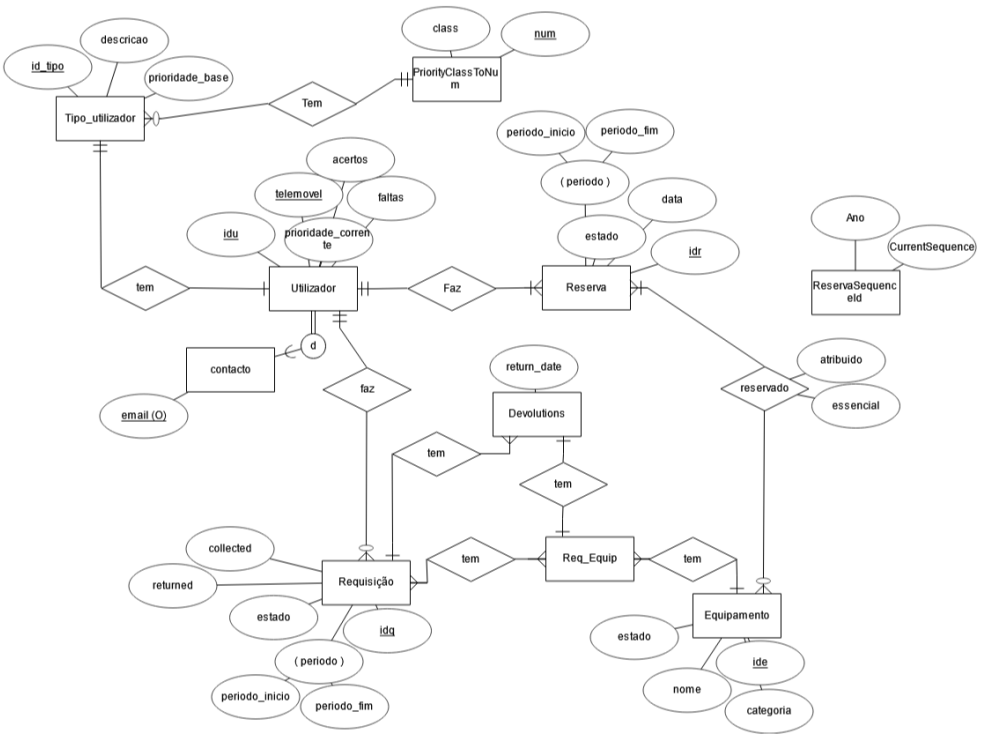


Figura 4.4: Quarta tentativa de modelação da base de dados.

Finalmente, é feito o mapeamento entre as prioridades existentes e um número inteiro correspondente (entidade `PriorityClassToNum`), e é considerada uma entidade própria para devoluções. Foi também adicionado ao diagrama entidade-associação a entidade `ReservaSequenceId`, que já existia na base de dados, mas por lapso, não fra adicionado nas versões anteriores.

O esquema relacional final do projeto é ilustrado na figura 4.5, onde a implementação final está em inglês.

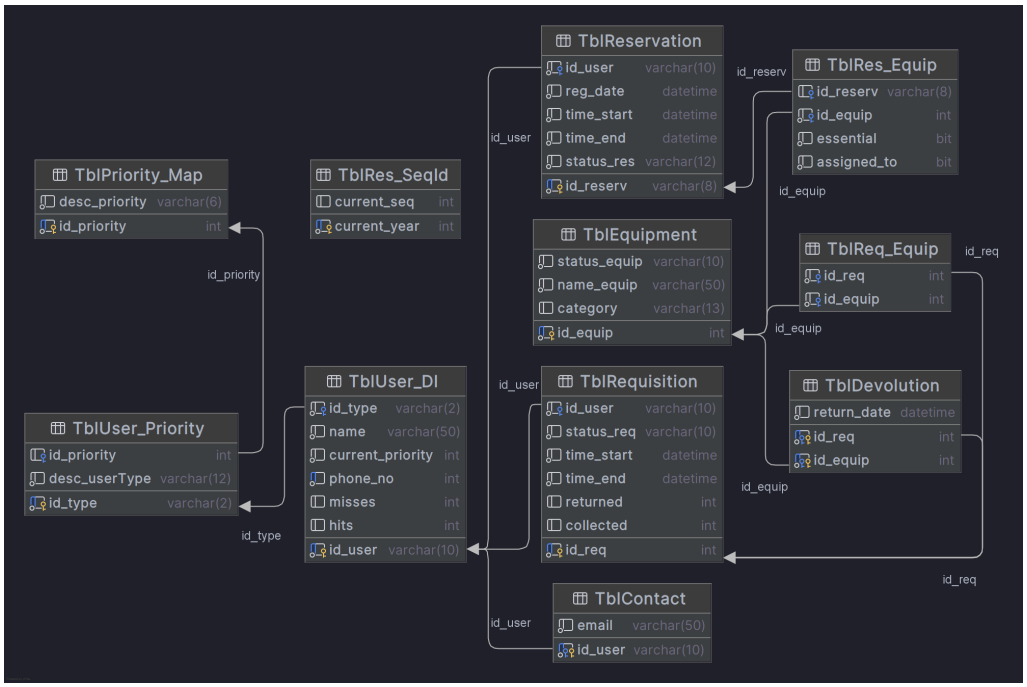


Figura 4.5: Esquema relacional da base de dados implementada.

4.4 Considerações

O modelo final cumpriu todos os objetivos definidos no inicio do projeto, segue todas as regras de modelação e normalização de bases de dados, e tem em conta tudo o que foi imposto pela própria natureza do projeto. De forma a seguir algumas regras como a ausência de valores a null ou valores duplicados fossem cumpridas foram criadas tabelas que tornassem tal objetivo possivel, como Res_Equip, Req_Equip, pois em vez de mantermos os dados das reservas/requisições junto com quais equipamento estão-lhes respetivamente atribuidos. Desta forma podemos afirmar que a base de dados permanece consistente durante a sua utilização.

Capítulo

5

Aplicação

5.1 Distribuição de Tarefas

5.1.1 Descrição precisa das tarefas

5.1.1.1 Tarefas de Alessandra Delgado

Listagem dos `procedures` feitos:

- `ReservationToRequisition`

Listagem dos `triggers` feitos:

- `UpdateEquipmentStatusToInUse`
- `ReservationToSatisfied`
- `UpdateUserPriorityOnInsert`

Listagem dos `scripts` feitos:

- `RUN_LOCAL`
- `RUN_UNI`
- `DropConstraints`
- `SetTriggerOrder`
- `DropTables`

Listagem dos ficheiros da aplicação *python* feitas:

- `enums`
  - `equipmentCategory`
  - `equipmentStatus`
  - `priorityType`
  - `requisitionStatus`
  - `reservationEquipmentType`
  - `reservationStatus`
  - `userType`
- `task`
- `views`
  - `equipment_index`
  - `equipment_new`
  - `user_index`
  - `user_new`
  - `reservation_index`
  - `reservation_new`
  - `reservation_edit`
  - `requisition_index`
  - `requisition_new`
  - `requisition_edit`
  - `nav`
- `app`

Listagem de secções escritas do relatório:

- Resumo
- Ferramentas e Tecnologias Utilizadas 2
- Configuração do Acesso ao Servidor 3.3
- Aplicação Cliente/Servidor 3.2
- *Front-end Python* 3.4
- Modelação de Dados 4.3
- Acesso à Base de Dados 5.2

5.1.1.2 Tarefas de Ana Silva

Listagem dos `triggers` feitos:

- `SetEquipmentStatusReserved`
- `ChangePriorityOnHitOrMissLimit`
- `EquipToAvailFinalState`
- `newRes_assignEquip`
- `PrioChange_AssignEquip`

Listagem dos `scripts` feitos:

- `CreateViews`

Listagem dos ficheiros da aplicação *python* feitas:

- `views`
  - `home`
- `models`
  - `Database`
  - `Equipment`
  - `Requisition`
  - `Reservation`
  - `UserDI`

Listagem de secções escritas do relatório:

- `Acesso à Base de Dados`, 5.2
- `Modelação de Dado`, 4.3
- `Descrição da organização`, 4.2
- `Modelo de dados`, 4.3

5.1.1.3 Tarefas de Carolina Gegaloto

Listagem dos `triggers` feitos:

- `ChangePriorityOnHitOrMissLimit`
- `IncrementHitOrMissOnReturn`
- `PenaltyOnCancelledReservation`
- `PenaltyOnUncollectedEquipment`

Listagem de secções escritas do relatório:

- `Enquadramento`, 1.1
- `Motivação`, 1.3
- `Objectivos`, 1.2
- `Introdução do desenvolvimento`, 3.1
- `Aplicação cliente/servidor`
- `Frontend Python`
- `Introdução (Modelação)`
- `Atributos returned e collected`
- `Conclusões`
- `Epílogo`

5.1.1.4 Tarefas realizados em conjunto

Listagem dos `procedures` feitos:

- `MakeID`
- `AssignEquipmentToUser`
- `DetermineStatus`

Listagem dos `triggers` feitos:

- `ReturnedEquip`

Listagem dos `scripts` feitos:

- `CreateTables`

5.2 Acesso à Base de Dados

A aplicação `python`, acede à base de dados com recurso à biblioteca `pyodbc`. Por defeito, esta não vêm conectada a nenhuma base de dados, carecendo da indicação do utilizador do IP, nome da base de dados, nome do utilizador e respetiva palavra-passe. Estes dados podem ser inseridos via menu superior de ferramentas 5.1, após a sua inserção a aplicação irá estabelecer uma conexão com a base de dados indicada utilizando para isso o ficheiro `DataBase` encarregue da abertura da conexão com a base de dados e o seu fecho. O excerto de código 5.1 contém a função encarregue da abertura da conexão, nesta antes da tentativa de abertura da conexão é terminada qualquer conexão ativa, garantindo o seu bom funcionamento.

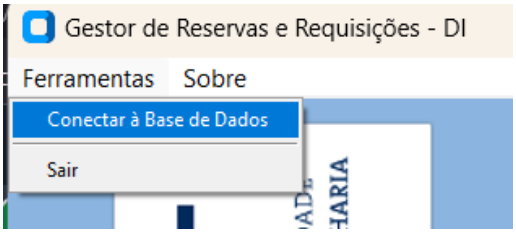


Figura 5.1: Opção de conexão da base de dados da aplicação `python`.

```
def connect():
    global conn

    # Terminate connection if there's one open
    if conn is not None:
        conn.close()

    try:
        conn = pyodbc.connect(
            f"DRIVER={{SQL Server}};SERVER={db_ip};DATABASE={db_name};
                UID={db_user};PWD={db_pass}")
        messagebox.showinfo("Sucesso", "Ligacao efetuada com sucesso!")
    except Exception as e:
        messagebox.showerror("Erro", f"Erro ao adicionar dados: {e}")
```

Excerto de Código 5.1: Método `connect` do modelo `DataBase`.

Como enunciado na secção 3.2, a aplicação desenvolvida segue o modelo, MVC, assim sendo, o acesso à base de dados foi abstraído do código dos formulários isolado nos seus ficheiros correspondentes, na diretoria *models*. Para cada tabela da base de dados, que o `python` necessita obter informação, foi implementado um ficheiro adequado com os métodos necessários. Estes

ficheiros contêm os métodos com as *queries* de inserção e pesquisa do SQL. Foram implementadas os seguintes modelos, cada um com os seus respetivos métodos de pesquisa e inserção:

- `Devolution`;
- `Equipment`;
- `Req_Equip`;
- `Requisition`;
- `Res_Equip`;
- `Reservation`;
- `UserDI`;
- `UserPriority`;
- `Contact`.

Estes modelos são então utilizados pelas inúmeras páginas que compõe a aplicação, removendo a duplicação de código. O excerto de código 5.2 ilustra a utilização do modelo `UserPriority` com o intuito de obter todos os tipos de utilizador, este método é utilizado no formulário de criação de utilizador. É também possível observar-se o detalhe da utilização do operador ternário na linha 4, para o caso em que o utilizador esteja a tentar aceder a página antes de se conectar a uma base de dados.

```
1  # Query database to get all user priorities
2  user_priorities = UserPriority.get_user_priorities()
3
4  self.user_type = ctk.StringVar(self, user_priorities[0] if len(
       user_priorities) > 0 else '')
5
6  ...
7
8  ctk.CTkLabel(self, text="Tipo de Utilizador").grid(row=5, column=0, padx
       =20, pady=(20, 0), sticky="w")
9  combo = ctk.CTkComboBox(self, values=user_priorities, variable=self.
       user_type, width=200)
10 combo.grid(row=6, column=0, pady=(3, 0), padx=20, sticky="w")
```

Excerto de Código 5.2: Exemplificação da utilização dos modelos implementados.

5.3 Funcionalidade

A plataforma dispõe de múltiplas funcionalidades, tais como

1. Gestão dos recursos da base de dados;
2. Sistema de faltas e cumprimentos;
3. Distribuição automática de equipamentos;
4. Validação dos dados de entrada;
5. Filtração da lista de utilizadores;
6. Filtração da lista de equipamentos;
7. Devolução parcial de equipamentos;
8. Determinação automática do estado das reservas;
9. Evolução de reserva para requisição;
10. Página inicial;
11. Sistema de preempção;
12. Utilizador Presidente.

5.3.1 Gestão dos Recursos da Base de Dados

5.3.1.1 Descrição Geral

Sendo a aplicação proposta uma aplicação de gestão de reservas e requisições que trabalha sobre utilizadores e equipamentos, surgiu a necessidade de uma gestão intuitiva e fácil uso por parte do utilizador.

5.3.1.2 Aplicação

A aplicação desenvolvida divide-se em quatro grupos, cada um responsável pela gestão de uma entidade, esta divisão é representada na aplicação pela barre lateral de navegação. Enumeradas na lista inferior, estas possuem páginas de consulta dos seus elementos, páginas de adição e, no caso da reserva e da requisição, páginas de consulta.

- Equipamento;

- Reserva;
- Requisição;
- Utilizador.

De modo a agilizar o código desenvolvido, cada ação realizada sobre uma entidade corresponde a um ficheiro de código, ou seja, a criação de uma reserva e a edição da mesma representam dois ficheiros de código.

5.3.2 Sistema de Faltas e Cumprimentos

5.3.2.1 Descrição Geral

O cumprimento dos prazos estabelecidos numa reserva ou requisição, são responsabilidades que devem ser cumpridas para o bom funcionamento da instituição. Sendo que o seu incumprimento deverá resultar numa penalização do utilizador. Estes prazos consistem no levantamento da reserva no seu prazo estabelecido, e a consequente devolução dos equipamentos.

5.3.2.2 Aplicação

Foi implementado um sistema de faltas e cumprimentos, representados pelas colunas `misses` e *hits* na tabela `TblUser_DI`. Estas duas colunas são utilizadas para se aplicar, ou recompensar, o utilizador conforme o seu cumprimento das normas estabelecidas. O parâmetro mais importante para o utilizador é a sua prioridade, por ser o campo principal na decisão de alocação de recursos pelas reservas.

Ao se realizar uma reserva esta é preenchida por um conjunto de dados tais como a sua data de início e de fim. O incumprimento do levantamento duma reserva, com equipamentos atribuídos, no seu prazo estabelecido, ou o cancelamento 2 horas antes do seu início, resultam numa penalização. Os `triggers` `PenaltyOnUncollectedEquipment` e `PenaltyOnCancelledReservation` asseguram estas regras na plataforma.

No momento da devolução de equipamentos de uma requisição, o `trigger` `IncrementHitOrMissOnReturn` verifica se estes estão a ser devolvidos no prazo estabelecido e, no caso em que não o sejam, atribui uma penalização consoante do tempo em atraso. A tabela 5.1 relaciona o tempo em atraso com a quantidade de penalizações que o utilizador irá receber.

Tempo de Atraso	N.º de Penalização
15 minutos até 1h59	1
2h00 até 2h59	2
3h00 até 3h59	3
4h00 até 4h59	4
5h00 ou superior	5

Tabela 5.1: Número de penalizações consoante o atraso da devolução dos equipamentos.

Caso o utilizador atinja uma penalização máxima de cinco, a sua prioridade atual é decrementada numa unidade (até um mínimo de um) e a coluna `misses` volta ao estado inicial. No caso do utilizador realizar uma devolução no prazo estipulado a coluna `hits` será incrementada numa unidade que ao atingir o valor de dois resultará na incrementação da prioridade do utilizador, numa unidade, (até um máximo da prioridade inicial do tipo de utilizador). Este sistema assegurado pelo`trigger` `ReducePriorityOnMissLimit`, responsável pelo decremento ou incremento da prioridade do utilizador

Esta funcionalidade é ignorada para o utilizador do tipo presidente, sendo este o único que possui uma prioridade fixa na plataforma.

Foi também adicionado um estado adicional da reserva, para além das estipuladas no enunciado, para distinguir entre uma reserva esquecida e uma não satisfeita. No caso de uma reserva não possuir equipamentos atribuídos não é prudente punir o utilizador visto que esta reserva não possui equipamentos que possam ser levantados. Neste caso, a reserva é marcada como *NotSatisfied* e o sistema de punições é ignorado.

5.3.3 Distribuição Automática de Equipamentos

5.3.3.1 Descrição Geral

Como a plataforma poderá possuir múltiplas reservas ativas em simultâneo, podendo estas se debruçar nos mesmos equipamentos, existiu a necessidade de que estes possam ser distribuídos de forma correta pelas diferentes reservas.

5.3.3.2 Aplicação

A abordagem realizada foi a do desenvolvimento do `procedure` `AssignEquipmentToUser` que, em conjunto com o `trigger` `SetEquipmentStatusReserved`, ao ser detetado uma introdução de uma nova reserva na base de dados, responsabiliza-

se pela distribuição dos equipamentos seguindo um conjunto de critérios. Estes critérios consistem em:

1. Prioridade – A prioridade do utilizador é o critério mais importante;

2. Essencial – A marcação de um equipamento como essencial numa reserva irá dar a esta uma prioridade superior a uma reserva em que o equipamento em questão não seja essencial;

3. Data de Início – Reservas que se iniciem numa data mais próxima irão possuir prioridade superior;

4. Data de Registo – Uma reserva realizada há mais tempo terá uma prioridade superior a uma realizada mais recentemente.

5.3.4 Validação dos Dados de Entrada

5.3.4.1 Descrição Geral

Toda a interação humana pode incorrer em erros inesperados, logo existiu a necessidade de garantir que os dados introduzidos pelos utilizadores da plataforma estejam concordantes de um conjunto prévio de regras.

5.3.4.2 Aplicação

Esta funcionalidade foi implementada em todos os formulários da aplicação, nesta, cada campo do formulário possui um conjunto de condições que devem estar corretas para que o formulário seja submetido com sucesso e os seus dados guardados na base de dados. De modo a facilitar o uso da aplicação, quando um formulário incorre em erros os campos correspondentes ficam a vermelho e é mostrada uma mensagem de erro própria ao utilizador, assim este poderá tomar resolver facilmente o problema e submeter o formulário. A figura 5.2 ilustra um exemplo de um formulário com um conjunto de mensagens de erro de auxílio ao utilizador.

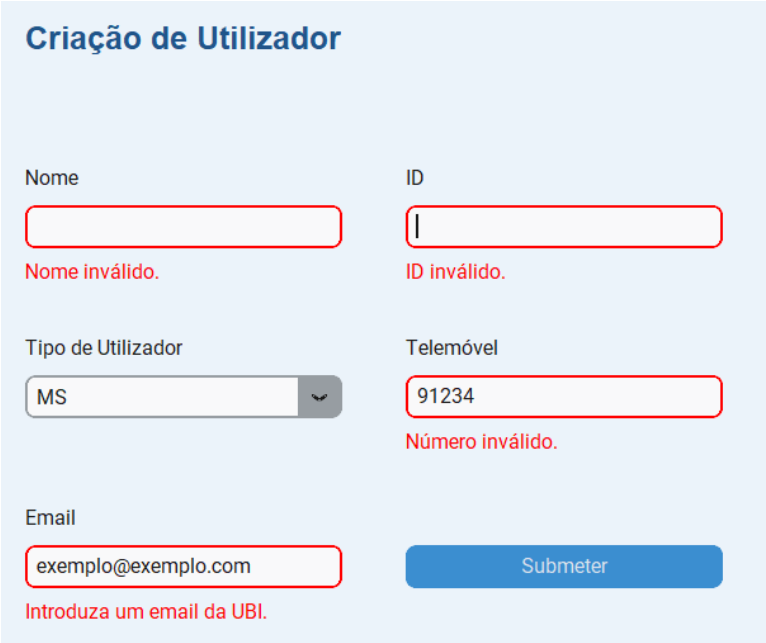


Figura 5.2: Exemplo de um formulário com mensagens de erro de validação.

De modo a garantir uma boa qualidade do código, estas validações foram colocadas num método próprio `is_valid`, presente em todos os formulários, que irá percorrer todos os campos do formulário e garantir o cumprimento das suas regras.

O excerto de código 5.3 demonstra as restrições existentes no preenchimento do *e-mail* do utilizador, que constituem na garantia que o email seja do tipo '`@ubi.pt`' ou '`@di.ubi.pt`'.

```
1 if self.email.get() != "" and (not self.email.get().endswith("@ubi.pt")
      and not self.email.get().endswith("@di.ubi.pt")):
2     valid = False
3     self.email_error.configure(text="Introduza um email da UBI.")
4     self.email.configure(border_color="red")
5 else:
6     self.email_error.configure(text="")
7     self.email.configure(border_color="#979DA2")
```

Excerto de Código 5.3: Validações sobre o campo *e-mail* no formulário de adição de utilizador.

5.3.5 Filtração da Lista de Utilizadores

5.3.5.1 Descrição Geral

Com o preenchimento da aplicação com múltiplos utilizadores, esta torna-se de difícil utilização devido ao esforço adicional na seleção do utilizador no momento da criação de uma reserva ou de uma requisição. Quer pela imensidade dos dados, ou pelo esquecimento de algum dado específico do utilizador em procura.

5.3.5.2 Aplicação

Os formulários em que é necessária a escolha de um utilizador para a sua submissão possuem a capacidade da filtração da `combobox` dos utilizadores por múltiplos campos. A filtração da `combobox` permite o filtro por nome, tipo de utilizador e número de telemóvel. A figura 5.3 ilustra a utilização desta funcionalidade de modo a facilitar a seleção do utilizador, nesta, a `combobox` foi filtrada pelo tipo de utilizador.

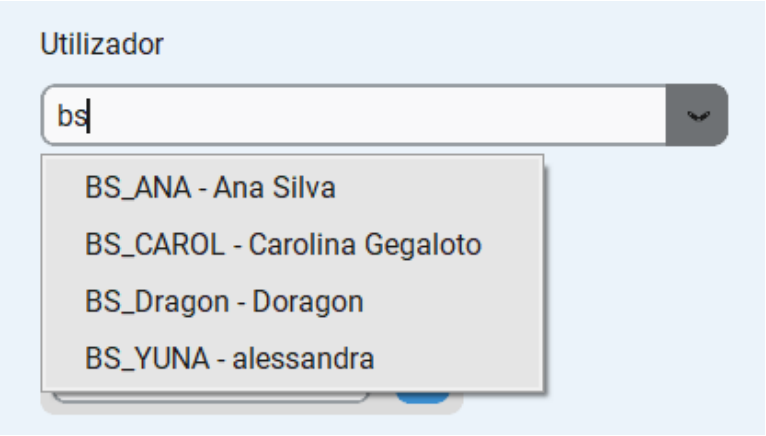


Figura 5.3: Exemplificação de uma filtração da `combobox` dos utilizadores por tipo de utilizador.

5.3.6 Filtração da Lista de Equipamentos

5.3.6.1 Descrição Geral

Com o preenchimento da aplicação com múltiplos equipamentos, cada um com as suas categorias, torna-se essencial a filtração da lista de equipamentos por múltiplos critérios, sendo estes a categoria do equipamento e, quando aplicável, a prioridade da reserva a que o equipamento está associado.

5.3.6.2 Aplicação

As listas de equipamentos embebidas no formulário de adição de reserva e requisição estão munidos de uma `combobox` onde é possível a seleção de uma categoria por parte do utilizador, esta é utilizada para se aplicar um filtro na mesma. No caso de existirem equipamentos que estejam atribuídos a reservas com prioridades superiores à do utilizador selecionado estes são omitidos da lista de equipamentos, visto que não é necessário poluir a lista de equipamentos com equipamentos que não serão atribuídos à reserva ou requisição do utilizador.

Esta funcionalidade tira proveito da utilização de *views* para facilitar a construção da *query* necessária para se obter esta informação. A figura 5.4 demonstra um excerto desta *view*. Nesta é possível observar-se os campos *priority* e *category* essenciais no mecanismo de filtragem. De notar que os campos vazios desta *view* significam que os equipamentos não se encontram atribuídos a uma reserva.

	Equipment_ID	Name	Status	Category	Time_start	Time_end	Priority
1	1	Toshiba TDP-S8U DLP	Reserved	<null>	2025-01-06 22:00:00.000	2025-01-06 23:00:00.000	3
2	25	TUF	Reserved	computer	2025-01-09 13:00:00.000	2025-01-09 14:00:00.000	3
3	2	Toshiba TDP-S8U DLP	Reserved	<null>	<null>	<null>	<null>
4	3	Toshiba TDP-S8U DLP	Reserved	<null>	<null>	<null>	<null>
5	4	Toshiba TDP-S8U DLP	Reserved	<null>	<null>	<null>	<null>
6	5	Toshiba TDP-S8U DLP	Reserved	<null>	<null>	<null>	<null>
7	6	Toshiba TDP-S8U DLP	Reserved	<null>	<null>	<null>	<null>

Figura 5.4: Dados parciais da *view* `ViewEquipmentPriority`

5.3.7 Devolução Parcial de Equipamentos

5.3.7.1 Descrição Geral

Quando um utilizador realiza uma requisição pode selecionar múltiplos equipamentos, podendo necessitar destes durante intervalos de tempo maiores ou menores. Assim, necessita de existir a capacidade da plataforma aceitar a devolução parcial dos equipamentos, evoluindo uma requisição para o estado final de entregue apenas quando forem devolvidos todos os equipamentos.

5.3.7.2 Aplicação

A plataforma suporta a devolução parcial ou total dos equipamentos de uma requisição, para isto foi implementada a tabela `TblDevolution` que relaciona um equipamento a uma requisição, assim é possível determinar-se se um equipamento de uma requisição encontra-se devolvido. O formulário de en-

trega de equipamentos adequa também o seu *design* para informar o utilizador quais os equipamentos já foram devolvidos 5.5.

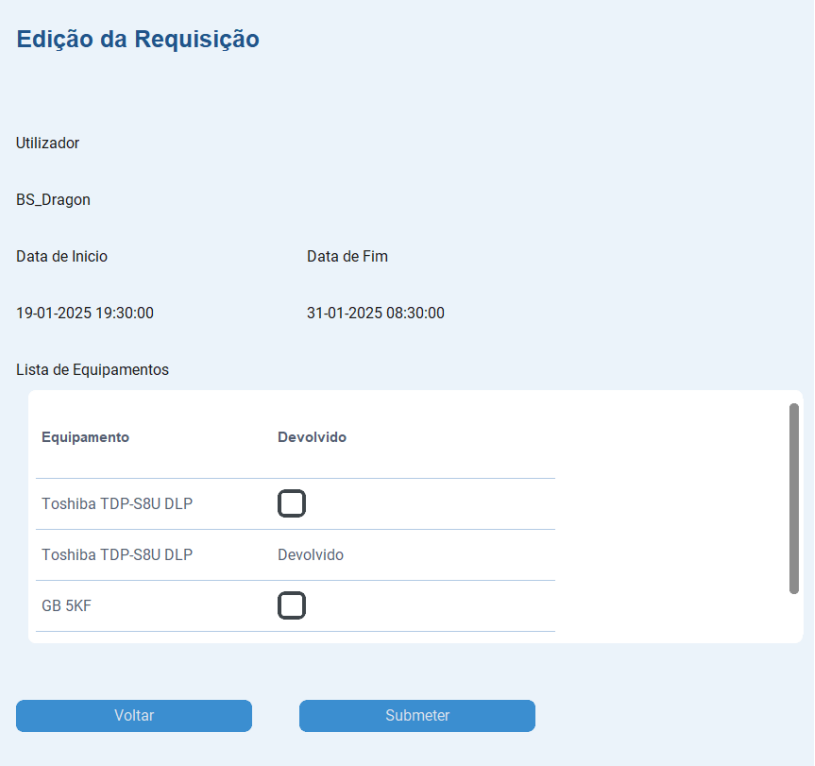


Figura 5.5: Formulário de devolução de equipamentos de uma requisição.

5.3.8 Determinação Automática do Estado das Reservas

5.3.8.1 Descrição Geral

As reservas podem encontrar-se em múltiplos estados dependendo da quantidade dos equipamentos atribuídos à mesma. Assim, é necessário a verificação do estado de uma reserva periodicamente.

5.3.8.2 Aplicação

Esta funcionalidade foi implementada com recurso da `procedure` `DetermineStatus`, responsável pela averiguação do estado de uma reserva, e do uso de um `cronjob`, de modo que a `procedure` fosse executada periodicamente. Porém, a utilização deste `cronjob` introduziu inicialmente o problema do bloqueio da *thread* principal do programa. Através da utilização de múltiplas *threads* este

problema foi anulado, movendo-se o *cronjob* para a sua própria *thread*. Esta *thread* adicional é terminada automaticamente com o fim da execução da interface gráfica do programa.

O excerto de código 5.4 ilustra a inicialização da nova `thread` e o seu termino, de modo a finalizar o programa corretamente. A variável `stop` é utilizada para forçar o termino do `cronjob`, e dá-lhe a indicação para terminar de correr dando fim à thread.

```
1 stop = threading.Event()
2
3 thread = threading.Thread(target = cron.init, args=(stop,))
4 thread.start()
5
6 ...
7
8 stop.set()
9 thread.join()
```

Excerto de Código 5.4: Inicialização de uma *thread*.

5.3.9 Evolução de Reserva para Requisição

5.3.9.1 Descrição Geral

Sendo a requisição a conclusão do processo de reserva na plataforma, e possuindo esta a mesma informação da reserva, é possível gerar uma requisição automática mediante uma reserva.

5.3.9.2 Aplicação

Esta funcionalidade foi implementada pela `procedure` `ReservationToRequisition`, que quando invocada gera automaticamente uma requisição extrapolando a informação necessária dos campos da reserva. Esta `procedure` é invocada pelo `trigger` `ReservationToSatisfied` que deteta alterações na tabela `TblReservation` e invoca a `procedure` quando deteta o levantamento de reservas.

5.3.10 Página Inicial

5.3.10.1 Descrição Geral

Com a adição de novos dados na plataforma, é crucial que o utilizador da mesma tenha acesso rápido e fácil a uma determinada quantidade de dados. Estas podem incluir: a existência de reservas ou requisições ativas, quantos utilizadores existem na plataforma, entre outros.

5.3.10.2 Aplicação

Dada a natureza da funcionalidade, foi desenvolvida uma página própria na aplicação que, por defeito, é a página inicial da mesma. Esta página apresenta uma determinada quantidade de estatísticas que podem ser utilizadas pelo utilizador para agilizar o planeamento das suas tarefas.

- Reservas ativas;
- Reservas em espera;
- Requisições ativas;
- Equipamentos disponíveis;
- Equipamentos totais;
- Utilizadores registados.

Como exemplificado na secção 5.2, a divisão das responsabilidades, segundo o modelo MVC, facilita a implementação desta página, sendo apenas necessária a consulta da base de dados segundo os seus múltiplos modelos.

5.3.11 Sistema de Preempção

5.3.11.1 Descrição Geral

Numa plataforma de reservas, é natural que estas se sobreponham eventualmente, podendo levar a situações constrangedores, visto que esta pode resultar na anulação de uma reserva no dia do seu início. Assim, foi implementado um sistema para bloquear novas reservas num espaço de 48 horas antes do fim das reservas.

5.3.11.2 Aplicação

Esta funcionalidade é uma verificação adicional no formulário de criação de reserva e requisição. Nesta, na seleção de equipamentos, a plataforma irá verificar a reserva a que estes estão atribuídos, se o estiverem, e irá verificar se a data de início da mesma encontra-se num espaço de 48 horas da nova reserva.

Esta verificação é realizada com recurso à *view* 5.4, ilustrada previamente. Esta reúne os dados necessários do equipamento para se realizar uma comparação com as datas da reserva a que o equipamento se encontra atribuído.

Esta restrição é obviamente ignorada para o caso do utilizador do tipo presidente, que pode realizar reservas e/ou requisições sem qualquer restrição adicional.

5.3.12 Tipo Presidente

5.3.12.1 Descrição Geral

Sendo esta plataforma planeada para ser utilizada em instituições, é normal que existam exceções à norma. Utilizadores com cargos especiais que permitam a ultrapassagem das restrições impostas devido a urgências.

5.3.12.2 Aplicação

De modo a prevenir a existência destas situações, foi implementado o cargo *presidente*, este cargo possui a prioridade máxima, é isento do sistema de penalizações e não é restringido pelo sistema de preempção.

Capítulo

6

Conclusões e Trabalho Futuro

6.1 Conclusões Principais

Dado o presente estado do trabalho, consideramos que os objetivos principais propostos foram conseguidos, como a automatização das regras de negócio na base de dados, através de `triggers` e `procedures`, de acordo com uma modelação que respeita as regras de normalização lecionadas nas aulas da unidade curricular, e a criação de uma aplicação que permite ao utilizador uma fácil navegação. Foi conseguida a adição e visualização da informação da base de dados a partir da mesma, principalmente a criação de reservas e requisições, sendo possível, a interação com as mesmas (edição) e sua visualização (listagem). Para além deste objetivo, para uma melhor automatização, foi implementado *threading* para a tarefa `cron`, que permite o agendamento periódico da avaliação de reservas pendentes/ativas.

6.2 Trabalho futuro

Mesmo estando conforme o pedido, considera-se que outras funcionalidades e sofisticações adicionais poderiam ser implementadas para elevar o estatuto do trabalho. São estas:

- Sistema de *login* – A criação um sistema de *login* na plataforma, permite a separação de um administrador de um utilizador comum;
- Uso de transações – O uso de transações permite o tratamento de acessos concorrentes à base dados, e um melhor tratamento dos dados. Não obstante, o tratamento de acessos concorrentes é tratado na aplicação, mas apenas para a inquirição dos dados na base de dados.

- Melhorar a validação de criação de uma reserva – Principalmente quando há sobreposição de reservas com equipamentos exatamente iguais.

Capítulo

7

Epílogo

Tendo em conta todos os tópicos abordados ao longo da unidade curricular, julgamos que incluir mais sessões práticas dedicadas à construção de uma base de dados, à criação de `triggers`, `procedures`, cursores e uso de transações seria benéfico para os alunos. Isto com base no facto que são elementos importantes para a elaboração do trabalho prático, este que, por natureza, é crucial à avaliação dos alunos.

Bibliografia

[1] Thomas Connolly and Carolyn Begg. *Database Systems, A Practical Approach to Design, Implementation and Management*. Pearson, 6th edition, 2015.

Apêndice

A

Criação de Tabelas e Restrições

```
CREATE TABLE TblRes_SeqId
(
    current_year INT PRIMARY KEY,
    current_seq  INT
);

CREATE TABLE TblPriority_Map
(
    id_priority    INT         NOT NULL,
    desc_priority VARCHAR(6) NOT NULL,
    PRIMARY KEY (id_priority)
);

CREATE TABLE TblUser_Priority
(
    id_type        VARCHAR(2)  NOT NULL,
    id_priority    INT,
    desc_userType VARCHAR(12) NOT NULL,
    PRIMARY KEY (id_type),
    FOREIGN KEY (id_priority) REFERENCES TblPriority_Map (id_priority)
        ON DELETE NO ACTION ON UPDATE NO ACTION,
);

CREATE TABLE TblUser_DI
(
    id_user           VARCHAR(10) NOT NULL UNIQUE,
    id_type           VARCHAR(2)  NOT NULL,
    name              VARCHAR(50) NOT NULL,
    current_priority INT          NOT NULL DEFAULT 3,
    phone_no          INT          NOT NULL UNIQUE,
    misses            INT                   DEFAULT 0,
    hits              INT                   DEFAULT 0,
```

```
33
34      CONSTRAINT CHK_TYPE CHECK (id_type IN ('PD', 'PR', 'RS', 'BS', 'MS',
            'DS', 'SF', 'XT')),
35      CONSTRAINT CHK_PRIORITY CHECK (current_priority BETWEEN 1 AND 5),
36      CONSTRAINT CHK_MISSES CHECK (misses BETWEEN 0 AND 5),
37      CONSTRAINT CHK_HITS CHECK (hits BETWEEN 0 AND 2),
38
39      PRIMARY KEY (id_user),
40      FOREIGN KEY (id_type) REFERENCES TblUser_Priority (id_type)
41          ON DELETE NO ACTION ON UPDATE NO ACTION,
42  );
43
44  CREATE TABLE TblContact
45  (
46      id_user VARCHAR(10) NOT NULL UNIQUE,
47      email   VARCHAR(50) NOT NULL,
48      PRIMARY KEY (id_user),
49      FOREIGN KEY (id_user) REFERENCES TblUser_DI (id_user)
50          ON DELETE NO ACTION ON UPDATE NO ACTION,
51  );
52
53  CREATE TABLE TblEquipment
54  (
55      id_equip      INT IDENTITY (1,1),
56      status_equip VARCHAR(10) NOT NULL DEFAULT 'Available',
57      name_equip    VARCHAR(50) NOT NULL,
58      category      VARCHAR(13),
59      CONSTRAINT CHK_STATUS_EQUIPMENT CHECK (status_equip IN ('Available',
            'Reserved', 'InUse')),
60      PRIMARY KEY (id_equip)
61  );
62
63  CREATE TABLE TblReservation
64  (
65      id_reserv  VARCHAR(8)  NOT NULL UNIQUE DEFAULT 'N/A',
66      id_user    VARCHAR(10) NOT NULL,
67      reg_date   DATETIME    NOT NULL,
68      time_start DATETIME    NOT NULL,
69      time_end   DATETIME    NOT NULL,
70      status_res VARCHAR(12) NOT NULL,
71
72      CONSTRAINT CHECK_STATUS_RESERVATION CHECK (status_res IN
73                                                 ('Active', 'Satisfied', '
                                                       Cancelled', '
                                                       Forgotten', 'Waiting'
                                                       ,
74                                                  'NotSatisfied' , '
                                                       Suspended')),
75      PRIMARY KEY (id_reserv),
```

```
76      FOREIGN KEY (id_user) REFERENCES TblUser_DI (id_user)
77          ON DELETE NO ACTION ON UPDATE NO ACTION,
78  );
79
80  CREATE TABLE TblRequisition
81  (
82      id_req      INT          NOT NULL IDENTITY (1,1),
83      id_user     VARCHAR(10) NOT NULL,
84      status_req VARCHAR(10) NOT NULL DEFAULT 'Active',
85      time_start DATETIME     NOT NULL,
86      time_end    DATETIME     NOT NULL,
87      returned    INT                  DEFAULT 0,
88      collected   INT                  DEFAULT -1,
89
90      CONSTRAINT CHK_COLLECTED CHECK (collected >= -1),
91      CONSTRAINT CHK_RETURN CHECK (returned >= 0),
92      CONSTRAINT CHK_STATUS_REQUISITION CHECK (status_req IN ('Active', '
            Closed')),
93      PRIMARY KEY (id_req),
94      FOREIGN KEY (id_user) REFERENCES TblUser_DI (id_user)
95  );
96
97  CREATE TABLE TblRes_Equip
98  (
99      id_reserv    VARCHAR(8),
100     id_equip     INT NOT NULL,
101     essential    BIT NOT NULL,
102     assigned_to BIT NOT NULL DEFAULT 0,
103     FOREIGN KEY (id_reserv) REFERENCES TblReservation (id_reserv)
104         ON DELETE NO ACTION ON UPDATE CASCADE,
105     FOREIGN KEY (id_equip) REFERENCES TblEquipment (id_equip)
106         ON DELETE NO ACTION ON UPDATE CASCADE,
107 );
108
109 CREATE TABLE TblReq_Equip
110 (
111     id_req    INT NOT NULL,
112     id_equip INT NOT NULL,
113     FOREIGN KEY (id_req) REFERENCES TblRequisition (id_req)
114         ON DELETE NO ACTION ON UPDATE CASCADE,
115     FOREIGN KEY (id_equip) REFERENCES TblEquipment (id_equip)
116         ON DELETE NO ACTION ON UPDATE CASCADE,
117 );
118
119 CREATE TABLE TblDevolution
120 (
121     id_req       INT       NOT NULL,
122     id_equip     INT       NOT NULL,
123     return_date DATETIME NOT NULL,
```

```
124     PRIMARY KEY (id_req, id_equip),
125     FOREIGN KEY (id_req) REFERENCES TblRequisition (id_req),
126     FOREIGN KEY (id_equip) REFERENCES TblEquipment (id_equip)
127 );
```

Apêndice

B

Dados Iniciais

```
INSERT INTO TblPriority_Map (id_priority, desc_priority)
VALUES (5, 'Maxima'),
       (4, 'Acima'),
       (3, 'Media'),
       (2, 'Abaixo'),
       (1, 'Minima');

INSERT INTO TblUser_Priority (id_type, desc_userType, id_priority)
VALUES ('PD', 'Presidente', 5),
       ('PR', 'Professor', 4),
       ('RS', 'Investigador', 3),
       ('BS', 'Licenciatura', 3),
       ('MS', 'Mestrado', 3),
       ('DS', 'Doutoramento', 3),
       ('SF', 'Apoio', 3),
       ('XT', 'Externo', 3);

INSERT INTO TblUser_DI(id_user, id_type, name, phone_no)
VALUES ('PD_Frutuos', 'PD', 'frutuoso', '274638468');

INSERT INTO TblUser_DI(id_user, id_type, name, phone_no)
VALUES ('PR_SPECIAL', 'PR', 'mr special', '123456789');

INSERT INTO TblUser_DI(id_user, id_type, name, phone_no)
VALUES ('RS_FABIO', 'RS', 'Fabio Craveiro', '347826592');

INSERT INTO TblUser_DI(id_user, id_type, name, phone_no)
VALUES ('DS_DARIO', 'DS', 'Dario Santos', '48527573');

INSERT INTO TblUser_DI(id_user, id_type, name, phone_no)
VALUES ('MS_DAVINA', 'MS', 'Davinas', '384396001');

```

```
33 INSERT INTO TblUser_DI(id_user, id_type, name, phone_no)
34 VALUES ('BS_Dragon', 'BS', 'Doragon', '999028458');
35
36 INSERT INTO TblUser_DI(id_user, id_type, name, phone_no)
37 VALUES ('BS_YUNA', 'BS', 'alessandra', '448566772');
38
39 INSERT INTO TblUser_DI(id_user, id_type, name, phone_no)
40 VALUES ('BS_ANA', 'BS', 'Ana Silva', '999586709');
41
42 INSERT INTO TblUser_DI(id_user, id_type, name, phone_no)
43 VALUES ('BS_CAROL', 'BS', 'Carolina Gegaloto', '88857472');
44
45 INSERT INTO TblUser_DI(id_user, id_type, name, phone_no)
46 VALUES ('SF_BETTEN', 'SF', 'Guilherme Paulo', '848393582');
47
48 INSERT INTO TblUser_DI(id_user, id_type, name, phone_no)
49 VALUES ('XT_MONIZ', 'XT', 'Moniz', '885748209');
50
51
52 INSERT INTO TblContact(id_user, email)
53 VALUES ('PD_Frutuos', 'frutuoso@di.ubi.pt'),
54        ('PR_SPECIAL', 'mrspecial@di.ubi.pt'),
55        ('RS_FABIO', 'fabio@di.upi.pt'),
56        ('DS_DARIO', 'dariosantos@ubi.pt'),
57        ('MS_DAVINA', 'davinass@ubi.pt'),
58        ('BS_Dragon', 'dragonshell@ubi.pt'),
59        ('BS_YUNA', 'alessandra@ubi.pt'),
60        ('BS_ANA', 'anasilva@ubi.pt'),
61        ('BS_CAROL', 'gegaloto@ubi.pt'),
62        ('SF_BETTEN', 'guilherme@gmail.com'),
63        ('XT_MONIZ', 'moniz4@gmail.com');
64
65 INSERT INTO TblEquipment (name_equip, category)
66 VALUES ('Toshiba TDP-S8U DLP', 'projector'),
67        ('Toshiba TDP-S8U DLP', 'projector'),
68        ('Toshiba TDP-S8U DLP', 'projector'),
69        ('Toshiba TDP-S8U DLP', 'projector'),
70        ('Toshiba TDP-S8U DLP', 'projector'),
71        ('Toshiba TDP-S8U DLP', 'projector'),
72        ('Toshiba TDP-S8U DLP', 'projector'),
73        ('HP 230', 'peripherals'),
74        ('HP 230', 'peripherals'),
75        ('HP 230', 'peripherals'),
76        ('Logitech MX Keys S', 'peripherals'),
77        ('Logitech MX Keys S', 'peripherals'),
78        ('Logitech MX Keys S', 'peripherals'),
79        ('Logitech MX Keys S', 'peripherals'),
80        ('Sony DCR405', 'other'),
81        ('Sony DCR405', 'other'),
```

```
82          ('Sony DCR405', 'other'),
83          ('Sony DCR405', 'other'),
84          ('Sony DCR405', 'other'),
85          ('Sony DCR405', 'other'),
86          ('Sony DCR405', 'other'),
87          ('Sony DCR405', 'other'),
88          ('Asus TUF', 'computer'),
89          ('Gigabyte GB 5KF', 'computer'),
90          ('Whiteboard Marker BIC', 'stationery');
```

Apêndice

C

Criação de Views

```
1  USE teste_di
2  GO
3
4  DROP VIEW IF EXISTS UserInfo
5  GO
6  CREATE VIEW UserInfo
7  AS SELECT u.id_user AS Identification, up.desc_userType AS 'Position',
8      u.name AS Name, u.phone_no AS 'Phone number', COALESCE(c.email, '')
           AS 'Email'
9    FROM TblUser_DI u
10   JOIN TblUser_Priority up ON u.id_type = up.id_type
11   LEFT JOIN TblContact c ON u.id_user = c.id_user;
12 GO
13
14 DROP VIEW IF EXISTS UserPriority
15 GO
16 CREATE VIEW UserPriority
17 AS SELECT u.id_user AS Identification, u.name AS Name, p.desc_priority
       AS Priority
18   FROM TblUser_DI u, TblUser_Priority up, TblPriority_Map p
19   WHERE u.id_type = up.id_type
20   AND up.id_priority = p.id_priority
21 GO
22
23 DROP VIEW IF EXISTS ActiveReservations
24 GO
25 CREATE VIEW ActiveReservations
26 AS SELECT r.id_user AS 'User', r.id_reserv AS 'Reservation id', r.
       time_start AS 'Start time',
27     CASE
28           WHEN DATEDIFF(DAY, GETDATE(), r.time_start) = 0
```

```
29          THEN CAST(DATEDIFF(HOUR, GETDATE(), r.time_start) AS NVARCHAR) +
                ' hours'
30          ELSE CAST(DATEDIFF(DAY, GETDATE(), r.time_start) AS NVARCHAR) +
                ' days'
31      END AS [Time left to start]
32     ,r.status_res AS 'Status'
33     FROM TblReservation r
34     WHERE r.status_res IN ('Active','Waiting')
35 GO
36
37 DROP VIEW IF EXISTS PendingRequisitions
38 GO
39 CREATE VIEW PendingRequisitions
40 AS SELECT r.id_user AS 'User', r.id_req AS 'Requisition id', r.time_end
       AS 'End time',
41     CASE
42          WHEN DATEDIFF(DAY, GETDATE(), r.time_end) = 0
43          THEN CAST(DATEDIFF(HOUR, GETDATE(), r.time_end) AS NVARCHAR) + '
                hours'
44          ELSE CAST(DATEDIFF(DAY, GETDATE(), r.time_end) AS NVARCHAR) + '
               days'
45      END AS [Time left to end]
46     ,r.status_req AS 'Status'
47     FROM TblRequisition r
48     WHERE r.status_req LIKE 'Active'
49 GO
50
51 DROP VIEW IF EXISTS ResourceState
52 GO
53 CREATE VIEW ResourceState
54 AS SELECT e.id_equip AS 'Equipment ID', e.name_equip AS 'Name', e.
       status_equip AS 'Status',
55       CASE
56          WHEN r.id_reserv IS NOT NULL THEN r.id_reserv
57          WHEN req.id_req IS NOT NULL THEN CAST(req.id_req AS VARCHAR)
58          ELSE ''
59      END AS 'Assignment ID',
60    CASE
61          WHEN r.id_reserv IS NOT NULL THEN r.id_user
62          WHEN req.id_req IS NOT NULL THEN req.id_user
63          ELSE ''
64      END AS 'User ID'
65     FROM TblEquipment e
66     LEFT JOIN TblRes_Equip re ON e.id_equip = re.id_equip
67     LEFT JOIN TblReservation r ON re.id_reserv = r.id_reserv
68     LEFT JOIN TblReq_Equip reqe ON e.id_equip = reqe.id_equip
69     LEFT JOIN TblRequisition req ON reqe.id_req = req.id_req;
70 GO
71
```

```
72 DROP VIEW IF EXISTS ViewEquipmentPriority
73 GO
74 CREATE VIEW ViewEquipmentPriority AS
75 SELECT
76         e.id_equip AS 'Equipment_ID',
77         e.name_equip AS 'Name',
78         e.status_equip AS 'Status',
79     e.category AS 'Category',
80     r.time_start AS 'Time_start',
81     r.time_end AS 'Time_end',
82         u.current_priority AS 'Priority'
83 FROM TblEquipment e
84 LEFT JOIN TblRes_Equip re ON e.id_equip = re.id_equip AND re.assigned_to
       = 1
85 LEFT JOIN TblReservation r ON re.id_reserv = r.id_reserv AND r.
     status_res IN ('Active', 'Waiting')
86 LEFT JOIN TblUser_DI u ON r.id_user = u.id_user
87 WHERE e.status_equip NOT LIKE 'InUse'
88 GO
```