Relatório
DETI-Maker Lab

Desenvolvimento de uma base de dados e aplicação de gestão de um* hacker space *(MakerLab)

2016/2017
Bases de Dados
P1G3

Mestrado Integrado em Engenharia de Computadores e Telemática

Diogo Ferreira (76425)
Pedro Martins (76551)

Indíce

Introdução

Na disciplina de Base de Dados foi proposto o desenvolvimento de um projeto de uma base de dados de uma plataforma.

Neste relatório estão descritos os detalhes da implementação da base de dados da plataforma DETI MakerLab, um laboratório no DETI, direcionado ao desenvolvimento de projetos. De referir que este trabalho foi desenvolvido em conjunto com a disciplina de Interação Humano-Computador.

A aplicação necessita dos plug-ins Ookii e Extended WPF Toolkit para a sua utilização, encontrando-se links para os mesmos nas referências do presente documento. Para se poder testar a aplicação, deve-se alterar a password de acesso ao servidor SQL (“BenficaTetra”), e pode criar-se uma nova conta de utilizador e, para a secção de staff, utilizar as credenciais “manuel.arez@ua.pt” e “password”.

Análise de Requisitos

O DETI tem um novo laboratório que pretende ser um espaço aberto e livre para alunos e docentes realizarem projetos de Eletrónica, Telecomunicações e Informática. Este laboratório pretende ser um hospedeiro de projetos e que os seus utilizadores estejam organizados por projetos. Cada projeto pode, por sua vez, requisitar equipamentos, componentes, recursos e espaços para a sua realização. Seguindo o espírito Maker/Hacker, o laboratório deverá ser autogerido, apesar de contar com um técnico dedicado ao laboratório.

Para a criação desta plataforma seguiram-se as seguintes premissas:

- Existem dois tipos de utilizadores, identificados unicamente com o seu número mecanográfico e caracterizados por nome, email, password de acesso à plataforma e um caminho para o ficheiro da sua imagem de perfil:
    - Professores, identificados ainda pela área científica;
    - Alunos, identificados pelo curso que frequentam.
- A sala é também gerida por técnicos, identificados pelo número de funcionário, nome, email, password de acesso e também um caminho para a sua imagem de perfil.

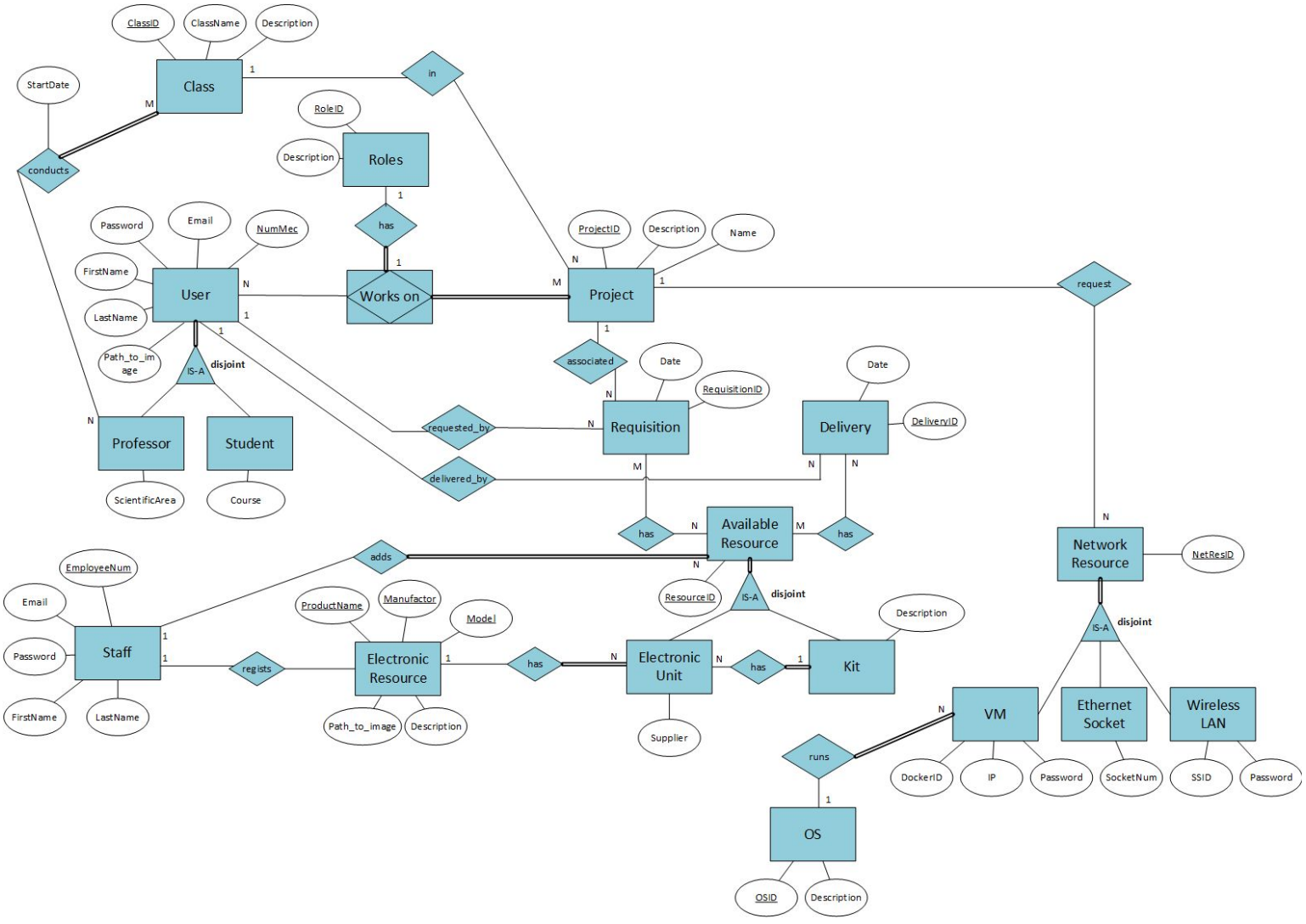
- Seguindo a lógica do conceito, os utilizadores participam em projetos, desempenhando um papel neles (Project Manager, Project Manager, Documentation Manager, Infrastructure Manager, Developer e Supervisor).
Os projetos são identificados por um número único, nome e a respetiva descrição e podem ser também feitos no âmbito de uma cadeira/disciplina (como PEI ou PEE).
- Uma disciplina é caracterizada pelo seu identificador único, pelo seu nome e descrição, e pelo professor responsável.
- No âmbito dos projetos, podem ser requisitados vários tipos de recursos, sejam eles materiais eletrónicos (RaspberryPi, Arduino, etc) ou recursos de rede (Máquinas virtuais, sockets Ethernet para se ligarem à rede das máquinas virtuais de um determinado projeto, redes WiFi). No entanto, essa requisição fica também associada a um utilizador membro do projeto, além de registada a data de requisição. Este tipo de requisição só é válido para recursos eletrónicos, pois a sua existência não depende de um projeto, enquanto que os recursos de rede só são válidos no

contexto de um projeto (dado que são criados explicitamente para serem utilizados no âmbito do mesmo).

- Aquando da devolução de material eletrónico requisitado, é registado também a data de devolução para consulta posterior.
- Por outro lado, visto que os recursos de rede podem ser facilmente destruídos ou não-atribuídos (caso das portas Ethernet), não será guardado qualquer historial de utilização além daqueles que estão atribuídos de momento, como referenciado anteriormente.
- O material eletrónico é caracterizado pelo seu fabricante, nome e modelo, além de uma descrição do equipamento e de um caminho para uma imagem.
- Podem existir várias unidades associadas ao material eletrónico, sendo estas caracterizadas pelo seu identificador único.
- É o técnico do laboratório que cria todas as entradas de material eletrónico e as respetivas unidades recebidas por determinado fornecedor.
- O técnico também pode optar por agregar estas unidades num kit (por exemplo, Kit RaspberryPi 3, constituído pelo RaspberryPi 3, cartão MicroSD e fonte de alimentação), que terá também um identificador único e que poderá ser requisitado para projetos. No entanto, as unidades do material escolhidas para kits não poderão ser requisitadas individualmente.
- Os recursos de rede têm um número identificador próprio, mas podem ser de vários tipos:
    - Máquinas virtuais, caracterizadas unicamente pelo seu DockerID (ID do container Docker), pelo seu endereço IP e password de acesso (o nome de utilizador é sempre o nome do projeto), e ainda o ID sistema operativo que estarão a correr (SO esse que é apenas identificado pelo seu ID e descrição - p.e. Ubuntu 14:04);
    - Ethernet Sockets, caracterizadas pelo seu número na sala DETI MakerLab;
    - Rede Wireless, caracterizada unicamente pelo seu SSID e password de acesso.
- Todos os campos de password (dos utilizadores, WLAN's e máquinas virtuais) devem estar encriptados com uma função de hashing.

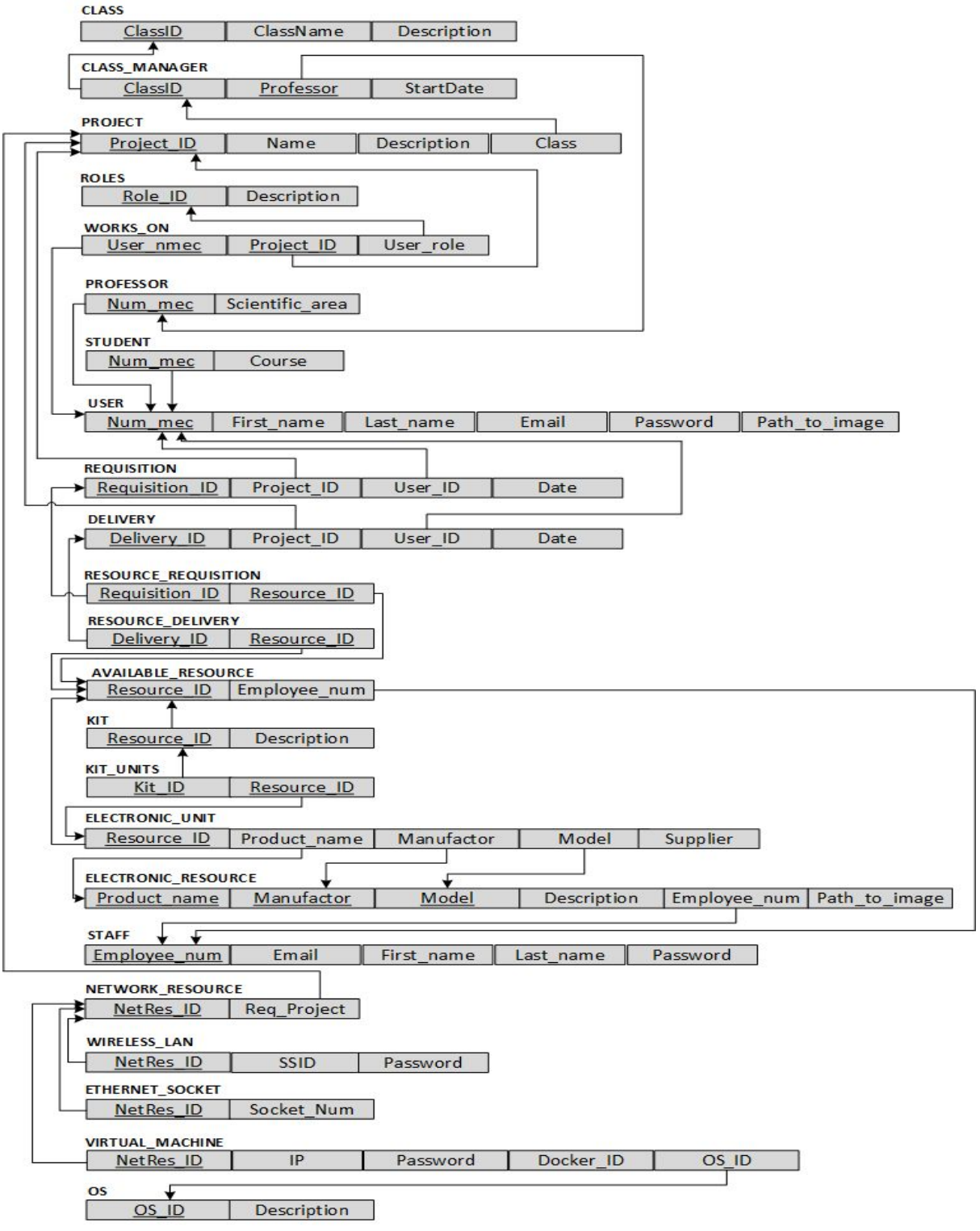
Diagrama Entidade-Relação

No seguimento da análise de requisitos anteriormente apresentada, foi elaborado o seguinte diagrama entidade-relação, que também segue em anexo para melhor visualização:



Modelo - Esquema Relacional

No seguimento do diagrama anterior, foi elaborado o seguinte modelo relacional da base de dados:



SQL DDL (Data Definition Language)

Tendo como base os diagramas anteriores, seguiu-se a construção das tabelas da base de dados.

O primeiro passo foi a definição de um SCHEMA para acolher todas as tabelas. O esquema de tabelas foi o definido com o Modelo Relacional anteriormente apresentado, destacando-se apenas os factos de terem sido criados dois tipos `DML.UsersList` e `DML.ResourcesList`, que são tabelas para auxiliar a manipulação de utilizadores e de recursos, respetivamente; mas também o facto de se utilizarem campos `DECIMAL(5, 0)` para guardar os números mecanográficos e de funcionário (considerando que 100000 é limite real deles), `VARBINARY(128)` para salvar o resultado da encriptação das passwords utilizando uma função de hashing e, por fim, a utilização de um campo `VARCHAR(200)`, em vez de `BLOB`, para guardar a localização das imagens necessários ao funcionamento da plataforma.

Segue-se a definição da base de dados, que também seguirá em anexo no ficheiro `TABLE_CREATION.SQL`:

```
GO
CREATE SCHEMA [DML]

GO
CREATE TYPE DML.UsersList AS TABLE (
	UserID DECIMAL(5,0),
	RoleID INT
);

GO
CREATE TYPE DML.ResourcesList AS TABLE (
	ResourceID INT
);

GO
CREATE TABLE DML.DMLUser (
	NumMec		DECIMAL(5,0)	NOT NULL,
	FirstName	VARCHAR(15),
	LastName	VARCHAR(15),
	Email		VARCHAR(50)	NOT NULL,
	PasswordHash	VARBINARY(128)	NOT NULL,
	PathToImage	VARCHAR(200),
	PRIMARY KEY (NumMec),
	UNIQUE (Email)
);

CREATE TABLE DML.Professor (
```

```
	NumMec			DECIMAL(5,0)	NOT NULL,
	ScientificArea		VARCHAR(50),
	PRIMARY KEY (NumMec),
	FOREIGN KEY (NumMec) REFERENCES DML.DMLUser(NumMec)
	 ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE DML.Student (
	NumMec			DECIMAL(5,0)	NOT NULL,
	Course			VARCHAR(50)	NOT NULL,
	PRIMARY KEY (NumMec),
	FOREIGN KEY (NumMec) REFERENCES DML.DMLUser(NumMec)
	 ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE DML.Staff (
	EmployeeNum		DECIMAL(5,0)	NOT NULL,
	FirstName		VARCHAR(15),
	LastName		VARCHAR(15),
	Email			VARCHAR(50)	NOT NULL,
	PasswordHash		VARBINARY(128)	NOT NULL,
	PathToImage		VARCHAR(200),
	PRIMARY KEY (EmployeeNum),
	UNIQUE (Email)
);

CREATE TABLE DML.Class (
	ClassID			INT			IDENTITY(1,1),
	ClassName		VARCHAR(50)		NOT NULL,
	ClDescriptionVARCHAR(max),
	PRIMARY KEY (ClassID),
	UNIQUE(ClassName)
);

CREATE TABLE DML.ClassManager (
	ClassID			INT			NOT NULL,
	Professor		DECIMAL(5,0)	NOT NULL,
	StartDate		DATE			NOT NULL,
	PRIMARY KEY (ClassID),
	FOREIGN KEY (ClassID) REFERENCES DML.Class (ClassID)
	 ON UPDATE CASCADE ON DELETE CASCADE,
	FOREIGN KEY (Professor) REFERENCES DML.Professor(NumMec)
);

CREATE TABLE DML.Roles (
	RoleID			INT			IDENTITY(1,1),
	RoleDescription		VARCHAR(50)		NOT NULL,
	PRIMARY KEY (RoleID),
	UNIQUE (RoleDescription)
);

CREATE TABLE DML.Project (
	ProjectID		INT			IDENTITY(1,1),
	PrjName			VARCHAR(50)		NOT NULL,
```

```
	PrjDescription		VARCHAR(max),
	Class			INT,
	PRIMARY KEY (ProjectID),
	FOREIGN KEY (Class) REFERENCES DML.Class(ClassID)
);

CREATE TABLE DML.WorksOn (
	UserNMec		DECIMAL(5,0)	NOT NULL,
	ProjectID		INT			NOT NULL,
	UserRole		INT			NOT NULL,
	PRIMARY KEY (UserNMec, ProjectID),
	FOREIGN KEY (UserNMec) REFERENCES DML.DMLUser(NumMec)
	 ON UPDATE CASCADE ON DELETE CASCADE,
	FOREIGN KEY (ProjectID) REFERENCES DML.Project(ProjectID)
	 ON UPDATE CASCADE ON DELETE CASCADE,
	FOREIGN KEY (UserRole) REFERENCES DML.Roles(RoleID)
);

CREATE TABLE DML.Requisition (
	RequisitionID		INT		IDENTITY(1,1),
	ProjectID		INT		NOT NULL,
	UserID			DECIMAL(5,0)	NOT NULL,
	ReqDate			DATE		NOT NULL,
	PRIMARY KEY (RequisitionID),
	FOREIGN KEY (ProjectID) REFERENCES DML.Project(ProjectID),
	FOREIGN KEY (UserID) REFERENCES DML.DMLUser(NumMec)
);

CREATE TABLE DML.Delivery (
	DeliveryID		INT		IDENTITY(1,1),
	ProjectID		INT		NOT NULL,
	UserID			DECIMAL(5,0)	NOT NULL,
	DelDate			DATE		NOT NULL,
	PRIMARY KEY (DeliveryID),
	FOREIGN KEY (ProjectID) REFERENCES DML.Project(ProjectID),
	FOREIGN KEY (UserID) REFERENCES DML.DMLUser(NumMec)
);

CREATE TABLE DML.AvailableResource (
	ResourceID		INT		IDENTITY(1,1),
	EmployeeNum		DECIMAL(5,0)	NOT NULL,
	PRIMARY KEY (ResourceID),
	FOREIGN KEY (EmployeeNum) REFERENCES DML.Staff(EmployeeNum)
);

CREATE TABLE DML.ResourceRequisition (
	RequisitionID		INT		NOT NULL,
	ResourceID		INT		NOT NULL,
	PRIMARY KEY (RequisitionID, ResourceID),
	FOREIGN KEY (RequisitionID) REFERENCES DML.Requisition(RequisitionID)
	 ON UPDATE CASCADE ON DELETE CASCADE,
	FOREIGN KEY (ResourceID) REFERENCES DML.AvailableResource(ResourceID)
	 ON UPDATE CASCADE ON DELETE CASCADE
);
```

```
CREATE TABLE DML.ResourceDelivery (
       DeliveryID          INT             NOT NULL,
       ResourceID          INT             NOT NULL,
       PRIMARY KEY (DeliveryID, ResourceID),
       FOREIGN KEY (DeliveryID) REFERENCES DML.Delivery(DeliveryID)
        ON UPDATE CASCADE ON DELETE CASCADE,
       FOREIGN KEY (ResourceID) REFERENCES DML.AvailableResource(ResourceID)
        ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE DML.Kit (
       ResourceID          INT             NOT NULL,
       KitDescription      VARCHAR(100)  NOT NULL,
       PRIMARY KEY (ResourceID),
       FOREIGN KEY (ResourceID) REFERENCES DML.AvailableResource(ResourceID)
        ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE DML.ElectronicResource (
       ProductName         VARCHAR(50)     NOT NULL,
       Manufacturer        VARCHAR(50)     NOT NULL,
       Model               VARCHAR(50)     NOT NULL,
       ResDescription      VARCHAR(max),
       EmployeeNum         DECIMAL(5,0)    NOT NULL,
       PathToImage         VARCHAR(200),
       PRIMARY KEY (ProductName, Manufacturer, Model),
       FOREIGN KEY (EmployeeNum) REFERENCES DML.Staff(EmployeeNum)
);

CREATE TABLE DML.ElectronicUnit (
       ResourceID          INT             NOT NULL,
       ProductName         VARCHAR(50)     NOT NULL,
       Manufacturer        VARCHAR(50)     NOT NULL,
       Model               VARCHAR(50)     NOT NULL,
       Supplier            VARCHAR(50)     NOT NULL,
       PRIMARY KEY (ResourceID),
       FOREIGN KEY (ResourceID) REFERENCES DML.AvailableResource(ResourceID)
         ON UPDATE CASCADE ON DELETE CASCADE,
       FOREIGN KEY (ProductName, Manufacturer, Model) REFERENCES
         DML.ElectronicResource(ProductName, Manufacturer, Model)
         ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE DML.KitUnits (
       KitID               INT             NOT NULL,
       ResourceID          INT             NOT NULL,
       PRIMARY KEY (KitID, ResourceID),
       FOREIGN KEY (KitID) REFERENCES DML.Kit (ResourceID)
        ON UPDATE CASCADE ON DELETE CASCADE,
       FOREIGN KEY (ResourceID) REFERENCES DML.ElectronicUnit (ResourceID)
        ON UPDATE NO ACTION ON DELETE NO ACTION
);
```

```
CREATE TABLE DML.OS (
	OSID		INT		IDENTITY(1,1),
	OSName		VARCHAR(50)	NOT NULL,
	PRIMARY KEY (OSID),
	UNIQUE (OSName)
);

CREATE TABLE DML.NetworkResource (
	NetResID	INT		IDENTITY(1,1),
	ReqProject	INT		NOT NULL,
	PRIMARY KEY (NetResID),
	FOREIGN KEY (ReqProject) REFERENCES DML.Project(ProjectID)
	 ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE DML.VirtualMachine (
	NetResID	INT		NOT NULL,
	IP		VARCHAR(15)	NOT NULL,
	PasswordHash	VARBINARY(128)	NOT NULL,
	DockerID	VARCHAR(50)	NOT NULL,
	OSID		INT		NOT NULL,
	PRIMARY KEY (NetResID),
	UNIQUE (DockerID),
	FOREIGN KEY (NetResID) REFERENCES DML.NetworkResource(NetResID)
	 ON UPDATE CASCADE ON DELETE CASCADE,
	FOREIGN KEY (OSID) REFERENCES DML.OS(OSID)
);

CREATE TABLE DML.EthernetSocket (
	NetResID	INT		NOT NULL,
	SocketNum	DECIMAL(5,0)	NOT NULL,
	PRIMARY KEY (NetResID),
	UNIQUE (SocketNum),
	FOREIGN KEY (NetResID) REFERENCES DML.NetworkResource(NetResID)
	 ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE DML.WirelessLAN (
	NetResID	INT		NOT NULL,
	SSID		VARCHAR(50)	NOT NULL,
	PasswordHash	VARBINARY(128)	NOT NULL,
	PRIMARY KEY (NetResID),
	UNIQUE (SSID),
	FOREIGN KEY (NetResID) REFERENCES DML.NetworkResource(NetResID)
	 ON UPDATE CASCADE ON DELETE CASCADE
);
```

SQL DML (Data Manipulation Language)

Depois de criada a estrutura de base dados, a mesma deve ser preparada para ser manipulada.

Inserção de Dados

O ficheiro de inserção de dados na base de dados segue em anexo com o nome `INSERT_DATA.sql` e apenas insere o mínimo de dados possíve,l para se poder começar a trabalhar na plataforma.

Por isso, são inseridos:

- Utilizadores, tanto professores como estudantes (utilizando o Stored procedure definido para tal, pois torna-se menos complexa a elaboração da query);
- Um elemento de Staff, apenas para dar acesso à secção de staff, já que a interface gráfica não permite o registo de novos elementos de staff;
- Várias disciplinas e os seus regentes;
- Os diversos papéis que os membros de projetos podem desempenhar;
- Alguns recursos eletrónicos;
- Os diversos Sistemas Operativos disponíveis para correrem em máquinas virtuais.

Consultas / Manipulação

Todas as consultas e manipulação feitas à base de dados utilizando a interface gráfica desenvolvida são feitas usando apenas Views, User Defined Functions e Stored Procedures.

As vistas utilizadas podem ser consultadas no ficheiro `VIEWS.SQL` e são as seguintes:

- **`DML.LAST_PROJECTS:`** Devolve os últimos 5 projetos criados;
- **`DML.LAST_REQUISITIONS:`** Devolve as últimas 5 requisições efetuadas;
- **`DML.ALL_ELECTRONIC_RESOURCES:`** Devolve todos os recursos / equipamentos eletrónicos;

- **`DML.ALL_ELECTRONIC_UNITS:`** Devolve todas as unidades de todos os equipamentos eletrónicos;
- **`DML.REQUESTED_RESOURCES:`** Função auxiliar que, para cada unidade/kit, conta o número de vezes que cada um foi requisitado;
- **`DML.DELIVERED_RESOURCES:`** Função auxiliar que, para cada unidade/kit, conta o número de vezes que cada um foi entregue;
- **`DML.ACTIVE_RESOURCES_REQS:`** Usando as duas funções anteriormente referidas, devolve todas as unidades/kits atualmente requisitados;
- **`DML.PROJECT_INFO:`** Devolve todos os projetos atualmente registados na plataforma;
- **`DML.ELECTRONIC_RESOURCES_INFO:`** Devolve todos os equipamentos eletrónicos atualmente registados na plataforma.

Normalização

Para evitar a existência de campos a `NULL`, foram criadas tabelas extra para certas relações que, apesar de adicionar os campos a `NULL`, poderiam ser integradas como atributos em certas relações. Um desses exemplos é a tabela **`KitUnits`** que associa um ID de equipamento ao de um Kit, permitindo assim saber que unidades contém. No entanto, existe um caso em que se optou por deixar o atributo em vez de criar uma nova relação: na relação **`Projects`**. Isto prende-se com o facto de que, depois de analisados os requisitos, percebemos que a maioria dos projetos que serão registados na plataforma serão realizados no contexto de uma disciplina e, portanto, não faria sentido estar a criar uma tabela apenas para associar projetos a disciplinas que seria muito populada e estaria a ocupar mais espaço em memória.

Com o objetivo de reduzir a redundância, foram também aplicados processos de normalização a todas as relações, de modo a que não houvesse atributos multi-valor, *nested-relations*, que todos os atributos não pertencentes a qualquer chave candidata dependessem totalmente dessa chave, que não existissem dependências funcionais entre atributos não-chave e que, todos os atributos sejam funcionalmente dependentes da chave, de toda a chave e nada mais, o que nos leva a concluir que as relações estão na forma **BCNF**.

Índices

No intuito de otimizar as consultas feitas à base de dados, tentou -se integrar o conceito de índices. No entanto, a maioria das consultas realizadas já utilizavam as chaves primárias das tabelas, não sendo portanto necessário a criação de índices para melhoramentos de performance. Apesar disso, foram aplicados alguns *non-clustered indexes* em atributos que apareciam muitas vezes nas cláusulas *WHERE* e/ou que eram utilizados em relações JOIN com maior frequência. Com o objetivo de diminuir os page splits (porque não são inserções ordenadas, principalmente na inserção de kits que podem ter descrições iguais) e unidades (aplicado na chave estrangeira de ElectronicResource, uma operação que, segundo a análise de requisitos, será extremamente utilizada), foi utilizado um fill factor de 70%.

O ficheiro que contém a definição dos índices encontra-se em anexo, segundo o nome `INDEXES.SQL`.

- **`CREATE INDEX IxKitDescription ON DML.Kit(KitDescription)`:** Non-clustered index para a descrição de um Kit de equipamentos;
- **`CREATE INDEX IxElectronicUnit ON DML.ElectronicUnit(ProductName, Manufacturer, Model)`:** Non-clustered index para a chave estrangeira de ElectronicResource na tabela Electronic Unit.

SQL Programming

SQL Programming é uma parte nuclear da utilização desta plataforma, pois toda a manipulação da mesma é feita a partir dela.

Triggers

Foram apenas utilizados triggers para a verificação da inserção de estudantes/professores e unidades electrónicas/kits, visto que, sendo relações com herança, devem ser verificados aquando a inserção numa das relações, se já não existe esse valor na outra relação.

Todos os triggers encontram-se em anexo no ficheiro `TRIGGERS.SQL`.

```
-- Verificação de existência de duplicados em professor / estudante
GO
CREATE TRIGGER DML.CHECK_PROFESSOR ON DML.Professor
AFTER INSERT, UPDATE
AS
	IF (EXISTS(SELECT NumMec FROM DML.Student WHERE NumMec in (SELECT NumMec
FROM inserted)))
		BEGIN
			RAISERROR ('Professor not updated/inserted - a student with
same Mec. Num. exists.', 16,1);
			ROLLBACK TRAN;
		END

GO
CREATE TRIGGER DML.CHECK_STUDENT ON DML.Student
AFTER INSERT, UPDATE
AS
	IF (EXISTS(SELECT NumMec FROM DML.Professor WHERE NumMec in (SELECT NumMec
FROM inserted)))
		BEGIN
			RAISERROR ('Student not updated/inserted - a professor with
same Mec. Num. exists.', 16,1);
			ROLLBACK TRAN;
		END

-- Verificação de existência de duplicados em unidades / kits
GO
CREATE TRIGGER DML.CHECK_ELECTRONIC_UNIT ON DML.ElectronicUnit
AFTER INSERT, UPDATE
AS
	IF (EXISTS(SELECT ResourceID FROM DML.Kit WHERE ResourceID in (SELECT
ResourceID FROM inserted)))
		BEGIN
```

```
                    RAISERROR ('Electronic Unit not updated/inserted - a Kit with
same ID exists.', 16,1);
                    ROLLBACK TRAN;
             END

GO
CREATE TRIGGER DML.CHECK_KIT ON DML.Kit
AFTER INSERT, UPDATE
AS
       IF (EXISTS(SELECT ResourceID FROM DML.ElectronicUnit WHERE ResourceID in
(SELECT ResourceID FROM inserted)))
             BEGIN
                    RAISERROR ('Kit not updated/inserted - an Electronic Unit with
same ID exists.', 16,1);
                    ROLLBACK TRAN;
             END
```

Stored Procedures

Foram criados diversos Stored Procedures para manipulação de dados da base dados, no entanto, devido à extensão do ficheiro, vamos apenas apresentar o propósito de cada um, sendo que a definição dos mesmos segue em anexo, no ficheiro `PROCEDURES.SQL`. Tentou-se que todos os Stored Procedures que envolvem mais do que uma operação que provocasse alterações na base de dados executassem as suas instruções de forma mais atómica possível, de tal forma que foram usadas transações para englobar tais operações. De salientar ainda que, todos os SPs que envolvam o registo de utilizadores foram encriptados para impedir o acesso à função de encriptação aos utilizadores.

- **`DML.REGISTER_STUDENT`:** regista um estudante na plataforma, segundo as informações que lhe são passadas como argumento, sendo aplicada na password uma função de hashing para encriptar a mesma, não a deixando à vista do utilizador;
- **`DML.REGISTER_PROFESSOR`:** regista um professor na plataforma, segundo as informações que lhe são passadas como argumento, sendo aplicada na password uma função de hashing para encriptar a mesma, não a deixando à vista do utilizador;
- **`DML.CREATE_EQUIPMENT`:** adiciona um equipamento à plataforma, aceitando todos os atributos que constituem a tabela ElectronicResource;
- **`DML.USERS_INFO`:** devolve todas as informações dos utilizadores, sejam eles estudantes ou professores;

- **DML.REQUISITION_UNITS:** devolve todas as unidades / kits que fazem parte da requisição com o ID que lhe é passado como argumento;
- **DML.PROJECT_USERS:** devolve todos os membros e papéis desempenhados de um determinado projeto;
- **DML.ADD_PROJECT_USERS:** aceitando um ID de um projeto e uma lista de utilizadores com o seu ID e o papel que desempenham no projeto, regista essas informações na base de dados;
- **DML.UPDATE_USER_ROLE:** aceitando um ID de um projeto, o número mecanográfico de um utilizador e o ID de um papel, atualiza o papel desempenhado pelo utilizador no projeto.;
- **DML.DELETE_PROJECT_USER:** dado um projeto e um utilizador, remove esse utilizador do projeto;
- **DML.PROJECT_ACTIVE_REQS:** devolve todos os recursos atualmente requisitados por um determinado projeto;
- **DML.RESOURCES_TO_REQUEST:** devolve todos os recursos disponíveis para requisição, isto é, aqueles que não estão requisitados (e no caso das unidades de equipamentos eletrónicos, aqueles a ser usados em kits);
- **DML.ADD_UNITS:** adiciona um certo número de unidades à plataforma de um determinado equipamento eletrónico;
- **DML.CREATE_PROJECT:** adiciona um novo projeto à plataforma, aceitando um nome, descrição, ID da disciplina no qual será realizado e devolve o ID do projeto criado;
- **DML.CREATE_REQUISITION:** cria uma nova requisição de recursos na base dados, aceitando um projeto, o número mecanográfico de um utilizador, registando ainda a data da mesma, devolvendo esse mesmo ID da requisição criada;
- **DML.REQUEST_RESOURCES:** cria novas entradas na tabela de recursos requisitados segundo a lista dos recursos a requisitar e o ID da requisição passados como argumento;
- **DML.CREATE_DELIVERY:** cria uma nova entrega de recursos na base dados, aceitando um projeto, o número mecanográfico de um utilizador, registando ainda a data da mesma, devolvendo esse mesmo ID da entrega criada;
- **DML.DELIVER_RESOURCES:** cria novas entradas na tabela de recursos devolvidos segundo a lista dos recursos a devolver e o ID da entrega passados como argumento;

- **DML.DELIVER_NET_RESOURCES:** apaga (devolve) todos os resources de networking associados à lista passada como argumento;
- **DML.REQUEST_VM:** aceitando o ID de um projeto, o IP, password, DockerID e ID do Sistema Operativo pretendido, devolve o ID de uma nova máquina virtual que foi adicionada à plataforma;
- **DML.REQUEST_SOCKETS:** aceitando um projeto e uma lista de sockets, associa essas sockets ethernet ao projeto;
- **DML.REQUEST_WLAN:** cria uma nova rede wireless, segundo o ID de um projeto e uma password, devolvendo o ID da rede recentemente criada;
- **DML.UPDATE_WLAN:** atualiza a password de uma determinada rede wireless;
- **DML.CREATE_KIT:** aceitando o ID do funcionário que o cria, uma descrição e uma lista de IDs de recursos que farão parte do mesmo, adiciona um kit à base de dados;
- **DML.KIT_UNITS:** devolve todas as unidades que constituem o kit com ID passado como argumento.

User Defined Functions

Foram também desenvolvidas diversas User Defined Functions para a consulta de dados na base dados, no entanto, devido à extensão do ficheiro, vamos apenas apresentar o propósito de cada uma, sendo que a definição das mesmas segue em anexo no ficheiro `UDF.SQL`. De notar que todos as UDFs que envolvam verificações de login foram encriptadas, para impedir o acesso à função de encriptação aos utilizadores.

- **DML.CHECK_LOGIN:** Dado um email e a password, verifica se existe um utilizador com essas credenciais e devolve-o;
- **DML.CHECK_STAFF_LOGIN:** Dado um email e a password, verifica se existe um elemento de staff com essas credenciais e devolve-o;
- **DML.USER_REQS:** Dado um número mecanográfico, devolve todas as requisições do referido utilizador;
- **DML.LAST_EQUIP_REQUISITIONS:** Dada a chave primária de um equipamento (nome do produto, modelo e fabricante), devolve informação relativa às últimas 5 requisições desse equipamento;

- **DML.LAST_KIT_REQUISITIONS:** Dado o identificador de um kit, devolve informação relativa às últimas 5 requisições desse kit;
- **DML.USER_PROJECTS:** Dada um número mecanográfico, devolve todas as requisições do dado utilizador;
- **DML.PROJECT_REQS:** Dada um identificador de um projeto, devolve todas as requisições associadas a esse projeto;
- **DML.PROJECT_COUNT_REQS:** Função auxiliar que, dado um projeto, conta o número de vezes que foram requisitados todos os recursos associados a esse projeto;
- **DML.PROJECT_COUNT_DELS:** Função auxiliar que, dado um projeto, conta o número de vezes que foram entregues todos os recursos associados a esse projeto;
- **DML.CLASS_REQS:** Dado o identificador de uma disciplina, devolve todas as requisições associadas à mesma.
- **DML.LAST_MANAGER:** Dado o identificador de uma disciplina, devolve o último regente da cadeira.
- **DML.VM_INFO:** Dado um identificador, devolve todas as informações relativas a uma máquina virtual;
- **DML.SOCKETS_INFO:** Dado um identificador, devolve todas as informações relativas a uma socket ethernet;
- **DML.WLAN_INFO:** Dado um identificador, devolve todas as informações relativas a uma rede wireless;
- **DML.AVAILABLE_SOCKETS:** De todas as 20 sockets disponíveis, evolve todas as sockets que não estão de momento associadas de momento a quaisquer projetos;

Conclusões

A base de dados desenvolvida foi integrada numa aplicação para gestão da mesma, desenvolvida na disciplina de Interação Humano-Computador. No entanto, existem algumas relações que se consideram não vir a ser alteradas, como a inserção de staff apenas poder ser feita pelo administrador da base de dados, assim como a inserção de disciplinas ou mesmo "roles" em projetos, reservando as mesmas para uma expansão futura de funcionalidades que a mesma aplicação possa vir a ter.

Um dos problemas encontrados foi a integração do uso de imagens na plataforma que, devido a restrições do WPF, foi necessário guardar caminhos absolutos em vez de relativos. Isto pode levar a que algumas imagens não sejam realmente carregadas (principalmente as dos utilizadores e produtos já introduzidas na plataforma), aparecendo uma imagem "dummy" no seu lugar. No entanto, quaisquer novas entradas na base de dados inseridas pelo utilizador, sejam elas de novos utilizadores ou novos componentes eletrónicos, serão corretamente guardadas e carregadas.

A aplicação desenvolvida pode ter aplicações reais, devido à sua ótima gestão da um *hacker space* tal como é o MakerLab. A base de dados desenvolvida é, portanto, o reflexo da melhor gestão possível dos recursos desta aplicação.

Referências

Plug-in Ookii: http://www.ookii.org/software/dialogs/
Extended WPF Toolkit: http://wpftoolkit.codeplex.com/