INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

ÁREA DEPARTAMENTAL DE ENGENHARIA DE ELECTRÓNICA E TELECOMUNICAÇÕES E DE COMPUTADORES

Gestão Unificada de Máquinas Virtuais Disponibilizadas por Diferentes Operadores de *Clouds* Públicas

Tiago Miguel Vila Martins, 37129
Licenciado em Engenharia Informática e Multimédia

Projeto para obtenção do Grau de Mestre
em Engenharia Informática e de Computadores

Orientador: Professor Doutor Carlos Jorge de Sousa Gonçalves

Júri:
Presidente: Professor Doutor Nuno Miguel Machado Cruz - ISEL/IPL
Vogais: Professor Doutor José Manuel de Campos Lages Garcia Simão - ISEL/IPL
Professor Doutor Carlos Jorge de Sousa Gonçalves - ISEL/IPL

Outubro, 2018

Dedico este projeto aos meus pais e à minha namorada, Maria João Vaz, que com todo o apoio e força, fizeram todos os esforços para que eu chegasse a esta etapa da minha vida. Aos meus primos, amigos e colegas de trabalho, pelo incentivo e pelo apoio constantes.

"A estrada para o sucesso não é fácil de navegar, mas com trabalho árduo, motivação e paixão, é possível alcançar o sonho americano."
Tommy Hilfiger

Agradecimentos

Quero agradecer, em primeiro lugar, ao meu orientador, Professor Carlos Gonçalves, por todo o apoio e dedicação para que este projeto chegasse ao fim com o seu objetivo cumprido.

Aos meus primos, amigos e colegas de trabalho, obrigado. Todos vocês, direta ou indiretamente, me ajudaram a chegar aqui. Tanto pelas conversas e discussões sobre ideias para este projeto, como pelas simples conversas sobre futebol. Ou até pelas corridas que partilhámos e me permitiram libertar alguma pressão que tinha acumulada em mim.

Agradecer à Maria João Vaz, pessoa com quem partilho a minha vida. Contigo tudo pareceu mais simples. Obrigado pela companhia, compreensão, carinho, paciência e pela tua capacidade de me trazer alegria durante todo o *stress* deste último ano.

Por último, mas não menos importante, quero agradecer aos meus pais. Sem eles nada disto seria possível. Obrigado por me darem a oportunidade de chegar até aqui, por me terem proporcionado esta experiência.

Resumo

Atualmente o uso de *cloud computing* está cada vez mais presente nas Tecnologias de Informação. Para simplificar, *cloud computing* é o fornecimento de serviços computacionais – servidores, armazenamento, bases de dados, rede, *software*, análise e muito mais – através da Internet.

A vasta oferta de operadores que fornecem este tipo de serviços cria um grande leque de opções para o utilizador. Este pode ter múltiplos serviços instanciados em diferentes operadores, mas para os gerir tem de utilizar as interfaces específicas de cada operador. Este trabalho foca-se apenas na vertente de gestão de máquinas virtuais.

Este projeto apresenta uma solução, sob a forma de uma aplicação Web, que permite a gestão, de uma forma transparente, de um conjunto de máquinas virtuais criadas em diferentes operadores de *cloud* pública. A gestão de uma máquina virtual implica operações como: criar, iniciar, reiniciar, parar e atualizar.

A aplicação que resulta deste projeto permite unificar a gestão de máquinas virtuais de diferentes operadores de *cloud* num ponto centralizado utilizando uma única API.

Palavras-chave:
Gestão de Máquinas Virtuais; Acesso Unificado a Máquinas Virtuais; Computação na Nuvem; Aplicação Web; REST API; Azure; AWS

Keywords:
Virtual Machine Management; Unified Access to Virtual Machines; Cloud Computing; Web Application; REST API; Azure; AWS

Abstract

Nowadays, cloud computing is more and more present in the Information Tecnologies. To put it simply, cloud computing supplies computer services - servers, storage, databases, network, software, analysis and more - through the Internet.

The diversity of providers that offer these types of service increases the options for the users. A user may have virtual machines in multiple providers. However, in order to manage those virtual machines, they need to use the specific user interfaces of each one. This project is focused only on the management of virtual machines.

This project brings a solution, in the form of a web application, that will allow the management, in a transparent way, of a set of virtual machines created on different public cloud providers. The management of a virtual machine includes actions such as create, start, restart, stop and refresh.

The application that results from this project allows to unify the management of virtual machines In different public cloud providers, through a centralized point using a single API.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

API	Application Programming Interface
CDN	Content Delivery Network
CRUD	Create-Read-Update-Delete
DNS	Domain Name System
EC2	Elastic Compute Cloud
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IAM	Identity & Access Management
JSON	JavaScript Object Notation
MVC	Model-View-Controller
ORM	Object-Relational Mapper
PaaS	Platform as a Service
REST	Representational State Transfer
SaaS	Software as a Service
SDK	Software Development Kit
SLA	Service Level Agreement
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TI	Tecnologias de Informação
UML	Unified Modeling Language
UVMM	Unified Virtual Machine Management
VPN	Virtual Private Network
XML	Extensible Markup Language
WSDL	Web Services Description Language

Capítulo 1

Introdução

A evolução da tecnologia é extraordinária, cada ano surgem novas tecnologias que vêm melhorar o dia-a-dia dos utilizadores e outras que mudam a forma como vemos o mundo, alterando as perspetivas que projetamos para o futuro.

O conceito de máquina virtual está a tornar-se comum nas Tecnologias de Informação (TI). Uma máquina virtual consiste num *software*, executado num computador, que executa programas como se fosse um computador real, este mecanismo designa-se por virtualização. Isto pode ser conseguido numa empresa sem recurso ao conceito de *cloud computing* [1].

As empresas estão a deixar os seus servidores físicos com custos elevados, para iniciar uma nova fase com as *clouds*. A grande variedade, de preços e condições que os operadores proporcionam, consegue satisfazer as distintas necessidades das empresas e utilizadores.

Cada operador possuí a sua aplicação própria para gerir os recursos criados. Entenda-se que a gestão de recursos consiste na criação, alteração ou remoção dos mesmos. As diferentes aplicações possuem diferentes interfaces de programação (*Application Programming Interface* — API), funcionalidades e formas de utilização. O modo de criação de um recurso, com as mesmas características e/ou funcionalidades, é distinto nos diferentes operadores. Estas são as razões que motivaram a realização deste projeto.

Há alguns anos atrás estávamos habituados a armazenar ficheiros e informação, bem como a utilizar aplicações, diretamente nos nossos próprios computadores ou dispositivos. No meio empresarial, este cenário é um pouco diferente, por norma as empresas utilizam aplicações que estão disponíveis em servidores. A principal vantagem desta abordagem consiste em ser possível, pelo menos na maioria das vezes, utilizar as aplicações mesmo sem acesso à Internet. Por outras palavras, é possível usar esses recursos em modo *offline*.

A evolução constante das TI está a fazer com que o acesso à Internet se torne cada vez mais vasto e rápido. Este cenário fornece as condições ideais

para a popularização da *cloud computing*, pois permite que este conceito se espalhe mundialmente. Na Figura 1.1 podemos observar uma visão geral deste conceito.

Figura 1.1: Ilustração do conceito *cloud computing*

Com o paradigma *cloud computing*, muitas aplicações, assim como ficheiros e outros dados relacionados, não necessitam de estar instalados ou armazenados no computador de cada utilizador ou num servidor local. Esses recursos passam a ficar disponíveis na *cloud*, ou seja, na Internet.

Cada vez mais é comum o uso de operadores de *cloud* públicas para alojar as aplicações Web. Com custos mais reduzidos e uma maior flexibilidade, continuam a ganhar cada vez mais quota de mercado nas TI, sendo cada vez menos usual a aquisição de servidores físicos para a instalação e manutenção das aplicações. Os operadores de *clouds* públicas disponibilizam interfaces e/ou aplicações Web que permitem a gestão de vários serviços entre os quais a gestão de máquinas virtuais, ou seja, trata-se de uma *Infrastructure as a Service* (IaaS). No entanto, cada um dos operadores fornece uma interface e/ou aplicação própria que na maioria dos casos são incompatíveis com as interfaces e/ou aplicações oferecidas pelos restantes fornecedores.

1.1 Problema

O objetivo deste projeto consistiu em disponibilizar um mecanismo que permita gerir, de modo unificado, as máquinas virtuais instanciadas em diferentes operadores de *cloud*, simplificando e reduzindo desta forma as ações do

utilizador. As vantagens que se destacam são:

- Utilização de uma única aplicação para gerir recursos em diferentes operadores;
- A ágil integração de novos operadores no sistema;
- A facilidade de distribuição que uma aplicação Web proporciona.

Um dos focos mais relevantes para este projeto, consiste na modularidade que se pretende para a integração de novos operadores. Sendo esta uma área cuja tendência é a entrada de novos operadores, é fundamental desenhar uma solução que permita incorporar facilmente essas novas opções.

Num cenário, em que um utilizador necessite de alocar, para um dado projeto, vários recursos em diferentes operadores, a gestão desses recursos pode tornar-se uma tarefa bastante complexa, devido à heterogeneidade das interfaces e/ou aplicações envolvidas.

Como exemplos de cenários de utilização destacam-se os seguintes: i) Utilização de aplicações com tolerância a falhas e balanceamento de carga; ii) Replicação de serviços em diferentes máquinas; iii) Comparar o desempenho de um *software* em diferentes ambientes.

A dificuldade de criar os recursos em diferentes operadores surge na necessidade de entender as diferentes aplicações oferecidas, os distintos modos de fazer a mesma ação ou até em compreender as diferentes APIs de cada um.

Este projeto propõe uma solução que permita unificar o acesso aos serviços IaaS de diferentes operadores de uma forma uniforme, escondendo do utilizador final os detalhes de configuração de cada operador. É neste sentido que surge a motivação para o desenvolvimento deste projeto, do qual resultou um protótipo - a aplicação *Unified Virtual Machine Management* (UVMM) - que pretende dar um contributo na simplificação da gestão de recursos instanciados em múltiplos operadores de *cloud* pública.

1.2 Objetivo

A aplicação UVMM destina-se principalmente a utilizadores que possuam contas em diferentes operadores de *cloud* públicas. O objetivo é que com esta aplicação os utilizadores possam fazer a gestão das várias máquinas virtuais num único ponto de controlo, simplificando este processo. Pretende-se que seja possível instanciar N máquinas virtuais distribuídas por diferentes

operadores. As funcionalidades que se destacam são: criar, iniciar, reiniciar, parar, atualizar e consultar detalhes.

Certamente que esta aplicação poderá ser usada também por utilizadores que apenas tenham conta num operador, no entanto a solução será mais conveniente para satisfazer as necessidades de utilizadores com diversas contas em diferentes *clouds*. Sendo um *software* que será oferecido na Web, um *Software as a Service* (SaaS), será tido em conta a responsividade do mesmo para que seja de fácil utilização mesmo em dispositivos móveis através do *browser*. Um dos propósitos mais significativos deste projeto passa por mostrar ao utilizador que é possível simplificar algo, que é bastante complexo de gerir em simultâneo.

Para um melhor enquadramento deste projeto é conveniente apresentar o seguinte conjunto de definições:

- ***Software as a Service* (SaaS)** [2]: é uma forma de distribuição e comercialização de *software*. Neste modelo o fornecedor do *software* responsabiliza-se por toda a estrutura necessária para a disponibilização do sistema e, o cliente, utiliza o *software* através da Internet, pagando um valor pelo serviço oferecido. Sendo assim, paga-se um valor pelos recursos utilizados e/ou tempo de utilização do *software*.

- ***Platform as a Service* (PaaS)** [2]: Plataforma como Serviço. Trata-se de um tipo de solução mais amplo para determinadas aplicações, incluindo todos (ou quase todos) os recursos necessários à sua execução, como armazenamento, bases de dados, escalabilidade (aumento automático da capacidade de armazenamento ou processamento), suporte a linguagens de programação, segurança, etc.

- ***Infrastructure as a Service* (IaaS)** [2]: Infraestrutura como Serviço. Parecido com o conceito de PaaS, mas aqui o foco é a estrutura de *hardware* ou de máquinas virtuais, sendo que o utilizador tem acesso a recursos do sistema operativo.

Na Figura 1.2 podemos observar alguns exemplos de tecnologias que são disponibilizadas como serviços. Dos operadores existentes podemos destacar alguns como Amazon Elastic Compute Cloud (EC2) [3], Microsoft Azure [4], Google Cloud Platform [5] e Luna Cloud [6]. Neste trabalho foram utilizados alguns destes operadores para testar a aplicação UVMM, nomeadamente, Amazon EC2 e Microsoft Azure.

Figura 1.2: Alguns exemplos de serviços

Em suma, a aplicação UVMM terá como objetivo simplificar a complexidade de gerir várias máquinas virtuais em diferentes operadores de *cloud* públicas.

Na Figura 1.3 podemos observar um fluxo simplificado que mostra a forma como a aplicação, de um modo geral, funciona, bem como as diferenças existentes entre as APIs dos diferentes operadores.

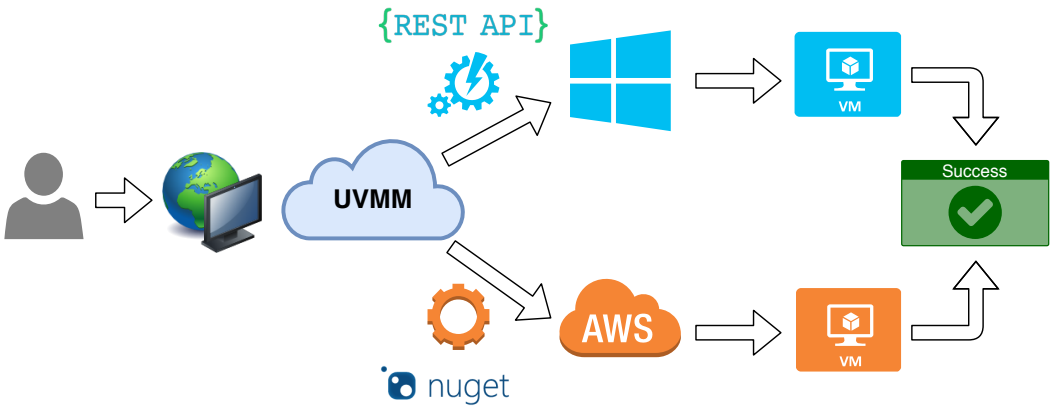

Figura 1.3: Fluxo *Unified Virtual Machine Management*

O utilizador acede à aplicação através de um *browser*. Após efetuar a sua autenticação pode fazer um pedido de criação de uma máquina virtual. Neste momento o pedido é processado e encaminhado para a implementação do operador escolhido, no caso exemplificado Azure ou Amazon, sendo que este comunica com a sua API enviando o pedido do utilizador. O acesso às APIs dos operadores é feito de diferentes modos. Por exemplo, a figura ilustra o acesso à API do Azure utilizando uma *Representational State Transfer* (REST) API e o acesso à API da Amazon utilizando um *Software Development Kit* (SDK), que é disponibilizado no NuGet [7], sendo que é o gestor de *packages* para .NET.

O resultado da criação da máquina virtual (sucesso ou insucesso) é apresentado ao utilizador. Outros operadores poderão disponibilizar outras formas de acesso aos seus serviços.

1.3 Desafios Iniciais

O maior desafio deste projeto foi a uniformização do acesso aos diferentes operadores num único *software*, isto porque cada operador tem diferentes especificidades/características em termos de: i) API; ii) Formas de autenticação; iii) Documentação fornecida; iv) Planos de experimentação.

Todos estes fatores foram obstáculos que se manifestaram em diferentes fases do desenvolvimento da solução. O custo inerente à criação de máquinas virtuais em ambiente *cloud* poderia ter sido um obstáculo intransponível. No entanto, os principais operadores de *cloud* oferecem planos gratuitos, ou de baixo custo, que permitem que um utilizador possa usufruir dos seus serviços. Neste projeto optou-se por utilizar como prova de conceito os operadores Amazon EC2 e Microsoft Azure. Dada a dimensão destes operadores, era expectável a existência de um bom suporte e documentação.

No decorrer do projeto verificou-se que as expectativas iniciais estavam corretas. Por exemplo, o operador Amazon EC2 disponibiliza bibliotecas, para várias linguagens de programação, que permitem a interação com a API permitindo que o programador se abstraia da complexidade dos pedidos HTTP que são trocados. No entanto outros operadores, como Azure, oferecem um SDK e têm a sua REST API publicada, indicando todos os *endpoints* e parâmetros necessários para realizar os pedidos, e o formato das respostas. É natural que neste caso o tempo de desenvolvimento possa ser superior comparativamente ao de usar uma biblioteca disponibilizada pelo próprio operador. Contudo optou-se por implementar o suporte ao Azure através da sua REST API, para demonstrar a flexibilidade que a solução tem em adaptar-se aos diferentes operadores.

É necessário realçar que o intuito deste projeto não é criar uma solução para gerir uma *cloud* no seu todo, apenas se pretende trabalhar com diversos operadores sem ter de lidar com as suas diferentes ferramentas de gestão e, por conseguinte criar e gerir máquinas virtuais de um modo mais simples.

Para implementar este projeto foi utilizada a linguagem de programação C#, com a *framework* .NET CORE 2.0 [8]. Esta *framework* é relativamente recente e veio mudar o paradigma defendido pela Microsoft durante anos, este restringia o uso do sistema operativo *Windows* para instalar uma aplicação desenvolvida em .NET. Com o .NET CORE passou a ser possível instalar a aplicação em qualquer máquina cujo o sistema operativo seja *Windows*, *Linux* ou *Mac OS*.

Um dos fatores que levantava alguma incerteza era a possibilidade de existirem algumas bibliotecas que ainda não estivessem totalmente disponíveis para esta nova versão. No entanto, ao longo deste projeto este problema não surgiu.

1.4 Organização do Documento

Além deste capítulo de introdução este documento está organizado da seguinte forma. No capítulo 2 (Estado da Arte e Trabalho Relacionado) é apresentado o estado da arte e o trabalho relacionado. A descrição da arquitetura proposta é apresentada no capítulo 3 (Arquitetura Proposta) e a sua implementação é apresentada no capítulo 4 (Implementação). O capítulo 5 (REST API) descreve a REST API que disponibiliza as funcionalidades desenvolvidas neste projeto, para que outros utilizadores possam inserir na sua aplicação a gestão de máquinas virtuais em diferentes operadores de *cloud* de forma unificada. As conclusões e o trabalho futuro são apresentados no capítulo 6 (Conclusões e Trabalho Futuro). Este documento termina com a bibliografia e os anexos, onde está incluída a explicação do mecanismo de *deploy* da solução, o modelo entidade-associação completo e os diagramas de classes, em *Unified Modeling Language* (UML), das bibliotecas desenvolvidas.

Capítulo 2

Estado da Arte e Trabalho Relacionado

Este capítulo apresenta o estado da arte e o trabalho relacionado. A secção 2.1, Principais Operadores de *Cloud*, apresenta os principais operadores de *cloud*, ou seja, aqueles que possuem maior quota de mercado atualmente. Na secção 2.2, *Software* de Suporte a *Clouds*, são apresentadas diversas aplicações/plataformas que permitem suportar serviços de *cloud* pública, privada e híbrida. Na secção 2.3, Trabalhos Relacionados, são apresentados projetos semelhantes. Este capítulo termina, na secção 2.4, Sumário, com um resumo de todo o capítulo.

2.1 Principais Operadores de *Cloud*

Os maiores operadores oferecem os seus serviços a uma escala mundial, com suporte técnico dedicado e com vasta documentação sobre o seu modo de funcionamento.

Na Figura 2.1 podemos observar a distribuição da quota de mercado entre os operadores, no primeiro trimestre de 2017 [9]. Enquanto a Amazon [3] é a principal referência em *cloud computing*, com 33% do mercado global, de acordo com o Synergy Research Group [10], os restantes operadores têm vindo a ganhar terreno. O Azure [4], que é a plataforma da Microsoft, detém o segundo lugar com uma quota de 10%. Já a IBM Cloud [11] aparece com a terceira maior referência com 8%, seguida da Google Cloud Platform [5] com 5%. O Oracle Cloud [12] e o Alibaba Cloud [13] têm, cada um, 3%. Microsoft, Google e Alibaba crescem cerca de 80%, ou mais, por ano, no entanto ainda permanecem com alguma desvantagem de quota de mercado comparativamente com a Amazon.

Figura 2.1: Quota de mercado dos operadores [9]

2.2 *Software* de Suporte a *Clouds*

O *software* de gestão de *cloud* encara o desafio de simplificar, e otimizar, as tarefas que estão envolvidas no processo de gestão de sistemas e infraestruturas baseadas numa *cloud* pública, privada ou híbrida. Estas aplicações possuem ferramentas flexíveis e escaláveis, estando orientadas para ajudar os utilizadores a encontrar as melhores estratégias de *cloud computing*. Essas estratégias, além das operações para criar e gerir recursos, podem incluir tarefas como auditorias de segurança, recuperação em caso de desastres e a realização de planos de contingência, entre outras.

Nas próximas secções serão apresentados 3 exemplos de *software* de suporte a *cloud*: i) OpenNebula na secção 2.2.1; ii) OpenStack na secção 2.2.2; iii) RightScale na secção 2.2.3.

2.2.1 OpenNebula

O OpenNebula [14] fornece recursos nas duas principais camadas de *Data Center Virtualization* e *Cloud Infrastructure*. Na Figura 2.2 pode observar-se a estrutura onde se enquadra esta solução.

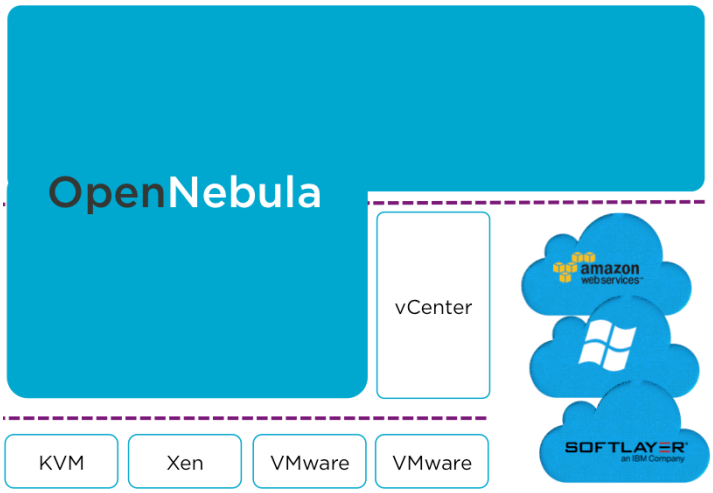

Figura 2.2: Estrutura do OpenNebula [15]

Grande parte dos utilizadores utilizam esta plataforma para gerir a virtualização de *data center*, para consolidar servidores e para integrar recursos de computação, armazenamento e rede. Esta plataforma tem controlo total sobre os recursos físicos e virtuais, fornecendo funcionalidades para a gestão, otimização e alta disponibilidade. O OpenNebula integra-se diretamente com hipervisores, como KVM [16], Xen [17] ou VMware ESX [18].

2.2.2 OpenStack

O OpenStack [19] é um sistema operativo de *cloud* que controla grandes conjuntos de recursos de computação, armazenamento e rede em todo o *data center*. Estes são geridos através da sua aplicação Web ou com o auxílio da sua API. O serviço que realiza a gestão das máquinas virtuais e interage com o hipervisor, já que suporta vários como o KVM, Xen, VMware, Hyper-V [20], QEMU [21], entre outros, é o Nova Compute. Na Figura 2.3 é possível verificar os diferentes componentes que constituem este sistema.

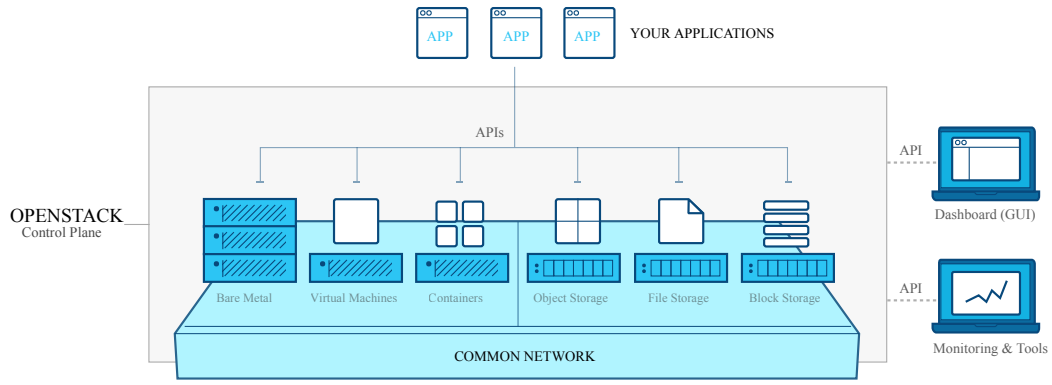

Figura 2.3: Estrutura do OpenStack [22]

O processo de *Auto Scaling*, que consiste na criação de uma nova instância quando a atual está com problemas ou não consegue responder a todos os pedidos, é um exemplo de um serviço que é disponibilizado e pode ser ajustado à necessidade e configuração do utilizador.

Este sistema oferece vários serviços, que podem ser configurados facilmente através da aplicação Web ou da sua API, como por exemplo: *firewall*, *Virtual Private Network* (VPN), *Object Storage*, *Block Storage*, *Domain Name System* (DNS) e base de dados.

2.2.3 RightScale

A RightScale [23] é uma empresa que vende *software* como um serviço para gestão de *cloud computing* para vários operadores. Na Figura 2.4 são apresentadas as duas soluções disponibilizadas.

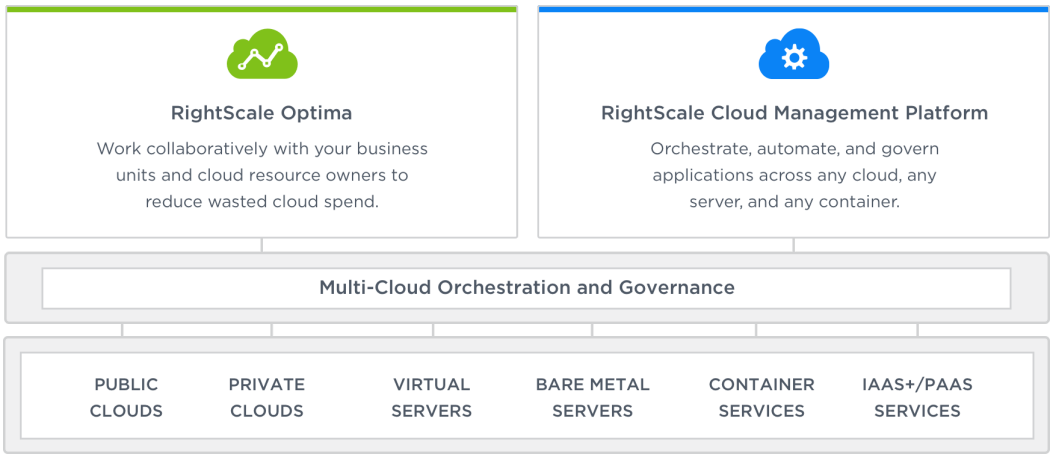

Figura 2.4: Estrutura do RightScale [24]

O RightScale Cloud Management Platform é uma solução abrangente que permite às organizações de TI fornecer um acesso instantâneo a um conjunto de serviços de *cloud* pública, privada e híbrida.

Esta plataforma suporta diversos operadores de *cloud* com o objetivo de fornecer interoperabilidade. A sua capacidade de gerir toda a infraestrutura de *cloud* a partir de uma única aplicação ajuda a evitar as restrições a qualquer operador.

2.3 Trabalhos Relacionados

Na tese de doutoramento de Carlos Gonçalves [25] houve a necessidade de criar várias máquinas virtuais em diferentes operadores de *cloud*. No entanto devido à falta de ambientes de desenvolvimento unificados que permitam o acesso a essa variedade de sistemas heterogéneos, foi necessário criar uma ferramenta para permitir criar e gerir máquinas virtuais em diferentes operadores. Esta ferramenta tem as implementações de cada operador separadas, ou seja, para a listagem das máquinas virtuais existentes em dois operadores, tem de se aceder a cada um individualmente, não existindo uma listagem geral do que existe em todos os operadores.

O projeto de Boris Savić [26] comparou duas plataformas de *cloud*, o OpenStack e o VmWare vCloud [27]. Devido às diferenças entre as suas APIs e para reduzir esta divergência, foram analisadas duas soluções: Libcloud [28] e jclouds [29]. Embora estas soluções resolvam as diferenças entre as APIs, não oferecem o suporte necessário para efetuar as configurações de

uma máquina virtual com base nos *Service Level Agreement* (SLA) do operador. Sendo assim foi desenvolvida a biblioteca XCloud para poder aceder a diferentes plataformas de *cloud* através de uma interface uniforme, sendo que oferece a capacidade de realizar pesquisas de SLA e, por conseguinte facilita a escolha do operador. O XCloud possuí uma interface gráfica acessível através de uma aplicação Web.

O Libcloud [28] é uma biblioteca desenvolvida em Python para interagir com os serviços de vários operadores de *cloud* usando uma API unificada. Esta foi criada com o objetivo de facilitar a criação de aplicações que funcionem com vários serviços oferecidos por diferentes operadores. Os serviços suportados são: i) *Cloud Servers* e *Block Storage*; ii) *Cloud Object Storage* e *Content Delivery Network*(CDN); iii) *Load Balancers as a Service*; iv) *DNS as a Service*.

O Apache jclouds [29] é uma biblioteca semelhante à Libcloud, com a diferença de esta ser escrita em Java. Esta oferece duas APIs: `ComputeService` e `BlobStore`. A primeira gere máquinas virtuais na *cloud*, sendo possível criar novas instâncias e instalar *software* nas mesmas. A segunda permite gerir os serviços de armazenamento nos operadores.

2.4 Sumário

Todas as plataformas e soluções apresentadas tratam de aspetos semelhantes entre si, sendo que umas são mais restritas que outras. Enquanto o OpenNebula é um esforço de código aberto focado nas necessidades do utilizador, o OpenStack é um esforço orientado pelos fornecedores.

Devido ao foco nos utilizadores, o OpenNebula oferece funcionalidades, como a gestão de redes virtuais e/ou a instalação e configuração automática de ambientes, que têm em conta necessidades que encaixam melhor no *Enterprise Cloud Computing*, e não tendo em conta as necessidades que resultam de um acordo entre os diferentes fornecedores que participam nos requisitos do projeto. Como as três ferramentas permitem a *cloud computing* de infraestrutura, há alguma sobreposição nos recursos que fornecem. No entanto, cada modelo de *cloud* apresenta diferentes restrições a nível arquitetural e requer interfaces especializadas, recursos de gestão e suporte à integração. Todos servem diferentes necessidades e implementam filosofias completamente distintas.

Por sua vez, o UVMM permite simplificar o processo de gestão de má-

quinas virtuais em diferentes operadores de *cloud*, ou seja, disponibiliza um ponto único de acesso, suportado numa interface comum independentemente do operador a que a máquina virtual pertence. Na Figura 2.5 é apresentada a forma como esta solução se enquadra com os diversos operadores de *cloud*.

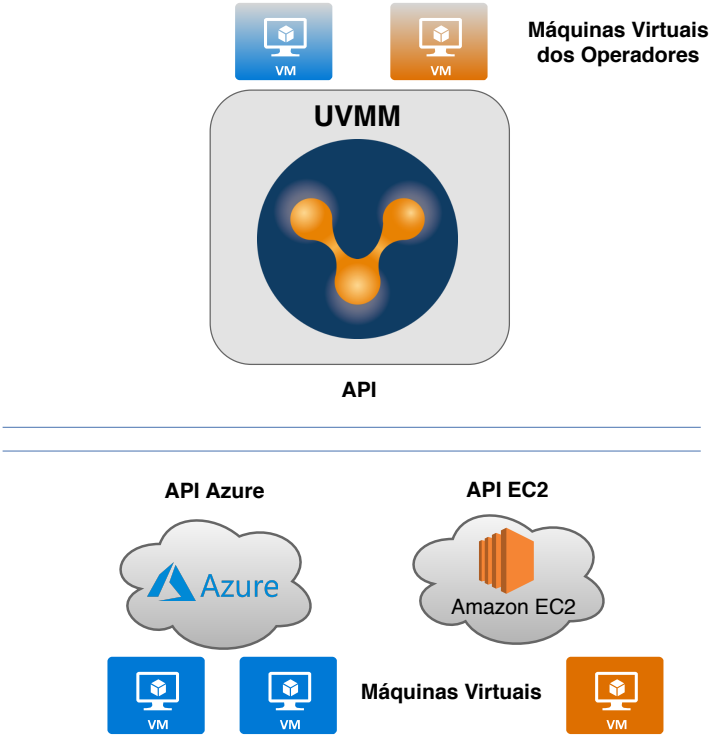

Figura 2.5: Estrutura do UVMM

A API disponibilizada pelo UVMM comunica com as APIs dos operadores, permitindo aceder às máquinas virtuais. A partir deste momento é possível realizar a gestão dessas máquinas e/ou criar novas. As máquinas virtuais criadas ou importadas através da aplicação Web, são mantidas numa base de dados específica da aplicação, ou seja, as suas informações são guardadas para ser possível manter um histórico das ações realizadas.

As plataformas acima referidas, OpenNebula, OpenStack e RightScale, pretendem gerir toda a infraestrutura de *cloud*, permitindo mesmo criar uma *cloud* privada composta por diversos recursos. O foco da solução UVMM é conseguir aceder a várias contas de *cloud* e conseguir gerir os recursos de máquinas virtuais de um modo centralizado.

Relativamente aos trabalhos relacionados, as bibliotecas Libcloud e jclouds são semelhantes à solução proposta pelo UVMM. No entanto o UVMM distingue-se destas bibliotecas nos seguintes aspetos: i) É oferecida como

uma aplicação Web; ii) Fácil integração de novos operadores; iii) É oferecida uma REST API que permite integrar a solução em qualquer aplicação.

A API do UVMM tem como objetivo permitir que outros desenvolvedores possam usufruir desta funcionalidade para criarem as suas próprias aplicações, ou evoluírem a solução dando suporte a novos operadores. A simplicidade de incorporar um novo operador na API é uma das maiores vantagens. No entanto para suportar esse operador na aplicação Web são necessárias outras tarefas como a criação de novas páginas ou a definição de novas tabelas na base de dados.

Capítulo 3

Arquitetura Proposta

Para garantir uma boa modularidade em toda a solução, foi feito um desenvolvimento separado por várias camadas. Esta abordagem permite reduzir o acoplamento funcional, aumentar a segurança, visto que cada camada pode aplicar diferentes mecanismos de proteção e estar em execução noutro servidor, e dividir as responsabilidades. A Figura 3.1 representa uma arquitetura típica num projeto segundo a metodologia de três camadas: camada de acesso a dados apresentada na secção 3.1, camada de negócio apresentada na secção 3.2 e camada de apresentação apresentada na secção 3.3.

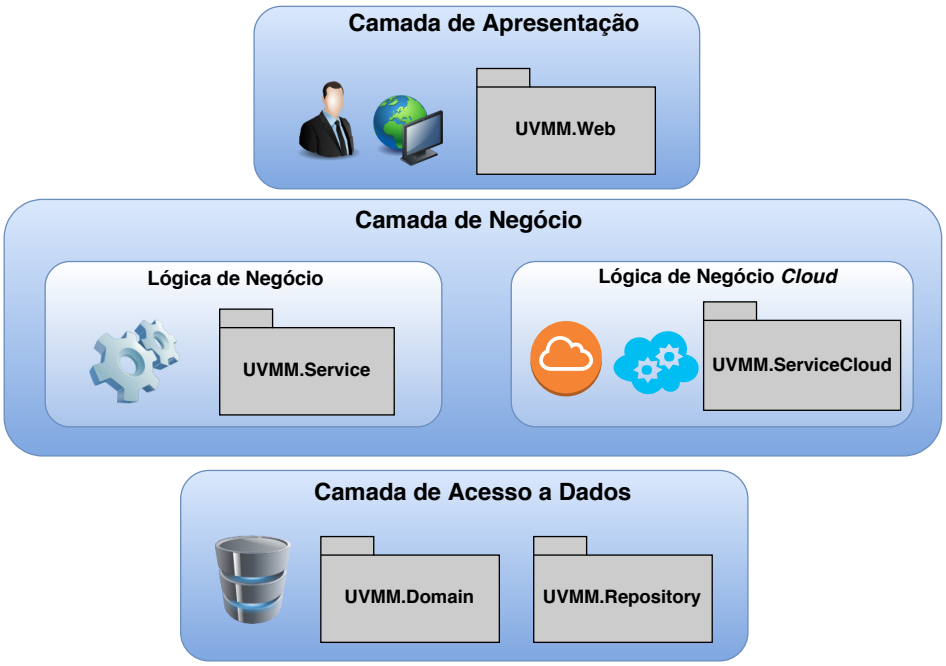

Figura 3.1: Arquitetura em camadas

Cada camada comunica diretamente com a camada que lhe é adjacente ou subjacente, caso exista. Assim sendo, a camada de apresentação comunica com a camada de negócio, enquanto que esta comunica com a camada de acesso a dados e com a camada de apresentação.

O capítulo termina com o Sumário, na secção 3.4, que resume os aspetos principais abordados neste capítulo.

3.1 Camada de Acesso a Dados

A camada de acesso a dados tem como função a escrita e leitura de informação na base de dados e, contém as entidades de domínio mapeadas em objetos, para permitir a interação sem ser necessário o uso de interrogações em *Structured Query Language* (SQL). Foi necessário criar entidades que permitam guardar e gerir informações dos utilizadores, das suas contas de *cloud*, das máquinas virtuais criadas e das permissões de cada um. Estes dados permitem que seja possível verificar o histórico de uma máquina virtual desde a sua criação até à sua eliminação.

Na Figura 3.2 podemos ver em detalhe o exemplo do que é o modo de funcionamento desta camada.

A base de dados, situada à esquerda, é a componente fundamental desta camada, pois é neste recurso que será mantida toda a informação. Para converter a estrutura da base de dados em objetos de domínio, é necessário mapear as tabelas através de um mecanismo de mapeamento de objetos relacional. Com esta abordagem é possível interagir com a base de dados sem ser imprescindível escrever instruções em SQL. A vantagem é a redução da complexidade, uma vez que é criado um nível de abstração sobre as operações SQL efetuadas sobre os dados.

3.2 Camada de Negócio

A camada de negócio possui dois elementos fundamentais. Na Figura 3.3 podemos ver com maior detalhe a função de cada um.

Um desses elementos, lógica de negócio, gere os dados da aplicação permitindo guardar os dados dos utilizadores, das máquinas virtuais criadas, entre outros. O outro, lógica de negócio de *cloud*, realiza toda a comunicação e interação com os diferentes operadores. Essa comunicação pode ser efetuada através de pedidos a uma REST API ou com o auxílio de um *Software Development Kit* (SDK) fornecido pelo operador.

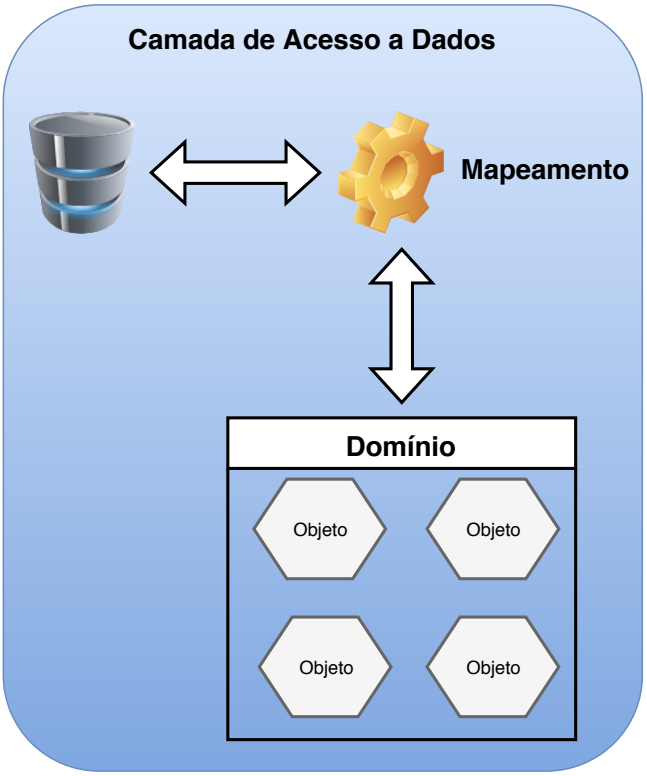

Figura 3.2: Detalhe da camada de acesso a dados

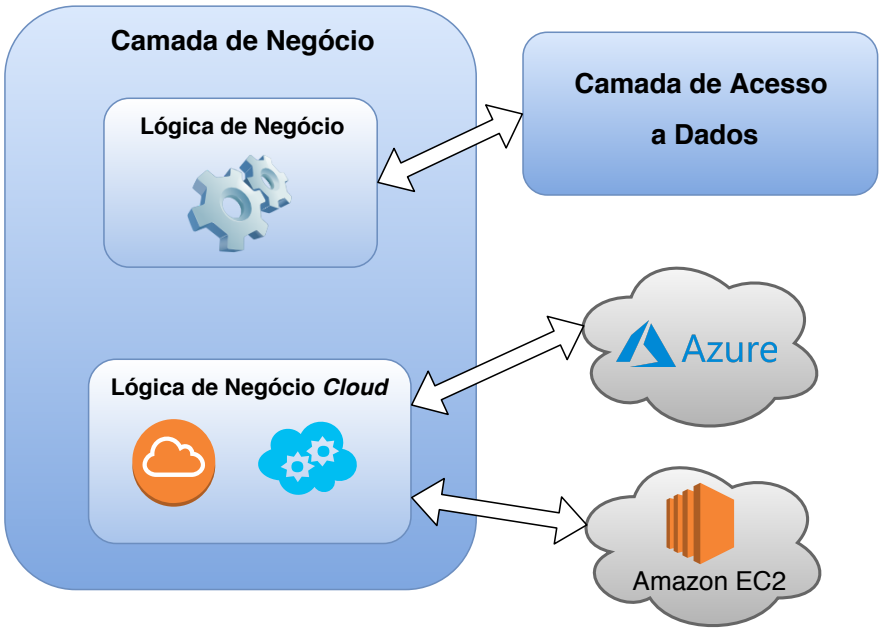

Figura 3.3: Detalhe da camada de negócio

3.3 Camada de Apresentação

A camada de apresentação coordenada a renderização das diferentes páginas, mostrando ao utilizador a informação necessária. Necessita de comunicar com a camada de negócio para obter todas as informações necessárias para as apresentar ao utilizador. Tendo em conta o seu conjunto de autorizações alguns conteúdos não serão apresentados, impedindo-o de visualizar informações que não lhe pertencem ou cujas permissões que possuí não suficientes para as ver. Na Figura 3.4 podemos observar com mais detalhe a composição desta camada.

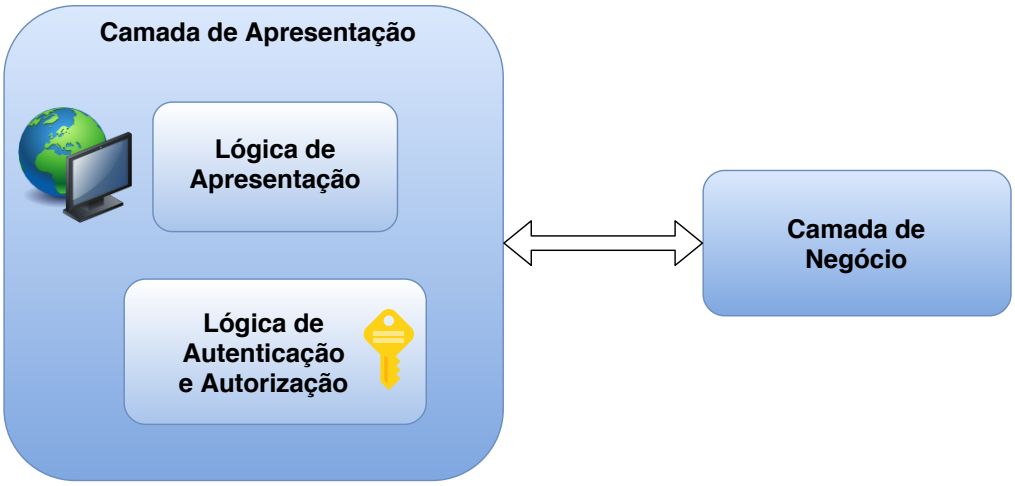

Figura 3.4: Detalhe da camada de apresentação

Nesta camada estão definidos todos os componentes que complementam a informação de uma máquina virtual. Alguns deles, tais como o histórico e as notas, são partilhados por qualquer operador, no entanto é possível criar componentes específicos e configura-los apenas para o operador em questão. Isto é possível devido à flexibilidade que existe em parametrizar estes componentes por operador.

3.4 Sumário

A arquitetura apresentada neste capítulo não está desenhada tendo em conta nenhuma linguagem de programação em concreto. Esta visão genérica pretende mostrar, de forma clara, o modo como a solução foi estruturada e quais as motivações para este tipo de abordagem, a divisão em camadas.

Na secção 3.1 foi apresentada a camada de acesso a dados. Esta contém um mecanismo de mapeamento, que transforma as tabelas da base de dados em objetos de domínio. Com estes objetos é possível criar e alterar os dados no sistema de armazenamento.

A camada de negócio, apresentada na secção 3.2, garante a validação da informação proveniente da camada de apresentação e, aplica todas as regras de negócio necessárias. Estas regras dividem-se em duas partes: i) lógica das entidades de domínio da própria solução; ii) lógica de negócio de cada operador.

Na secção 3.3, foi apresentada a camada de apresentação. Aqui é onde se aplica a lógica de apresentação, ou seja, tendo em conta o utilizador e as suas autorizações é ajustada a informação que lhe é apresentada. É nesta camada que é feita toda a lógica associada à autenticação e autorização de utilizadores.

O suporte de um novo operador necessita de alguns desenvolvimentos nas três camadas apresentadas. Na camada de apresentação é necessário criar novas vistas. Na camada de negócio são precisos novos serviços para a implementação da comunicação com o operador. Na camada de acesso a dados pode ser necessário criar novas tabelas. No entanto, a solução está preparada para estas mudanças sem necessidade de configurações complexas, ou seja, não existe nenhuma lógica de negócio que implique acrescentar uma nova condição, num determinado serviço ou controlador, sempre que é adicionado o suporte a um novo operador. Para cada novo operador existe um conjunto de tarefas que devem ser desenvolvidas, contudo não é necessário alterar outras regras de negócio para criar compatibilidade com o novo operador.

No capítulo 4, Implementação, cada camada será explicada em detalhe, sendo apresentadas todas as características e objetivos de cada uma.

Capítulo 4

Implementação

Este capítulo destina-se à apresentação e explicação de toda a solução implementada para atingir o objetivo da aplicação, uniformizar, num ponto central, a gestão de máquinas virtuais criadas em diferentes operadores de *cloud*.

Para o funcionamento da aplicação é necessário uma estrutura de dados para suportar todas as funcionalidades oferecidas. Estas funcionalidades incluem por exemplo o suporte à autenticação ou guardar o histórico sobre as ações efetuadas numa máquina virtual. Por esse motivo é necessário manter persistência dessa informação.

Este capítulo está dividido em 5 secções, cuja organização está feita tendo em conta a arquitetura de camadas implementada pela solução. Na secção 4.1 (Modelo de Dados) é apresentado o modelo de dados da solução. Em seguida, na secção 4.2 (Camada de Acesso a Dados) é apresentada a camada de acesso a dados. Seguidamente, na secção 4.3 (Camada de Negócio), é mostrada a estrutura da camada de negócio, e por fim a secção 4.4 (Camada de Apresentação) centra-se na explicação da camada de apresentação. O capítulo é finalizado com o Sumário, na secção 4.5, em que é feita uma síntese de todo o conteúdo descrito no presente capítulo.

4.1 Modelo de Dados

O modelo de dados criado teve em conta não só a componente de informações que se pretende guardar de cada utilizador, bem como o histórico das ações que foram realizadas, como por exemplo todas aquelas que envolvem a gestão de uma máquina virtual desde a sua criação até à sua remoção. Desta forma será possível, por exemplo, verificar o histórico de ações que foram aplicadas a uma máquina virtual. Foi ainda tido em conta a necessidade de existir uma

forma de parametrizar os componentes de cada operador através da base de dados.

O PostgreSQL [30] foi o sistema de gestão de base de dados utilizado. Esta escolha foi feita tendo em conta que, atualmente, este é um dos sistemas *open source* mais avançados e completos [31]. Na Figura 4.1 podemos observar o modelo entidade-associação simplificado que foi desenhado para a solução.

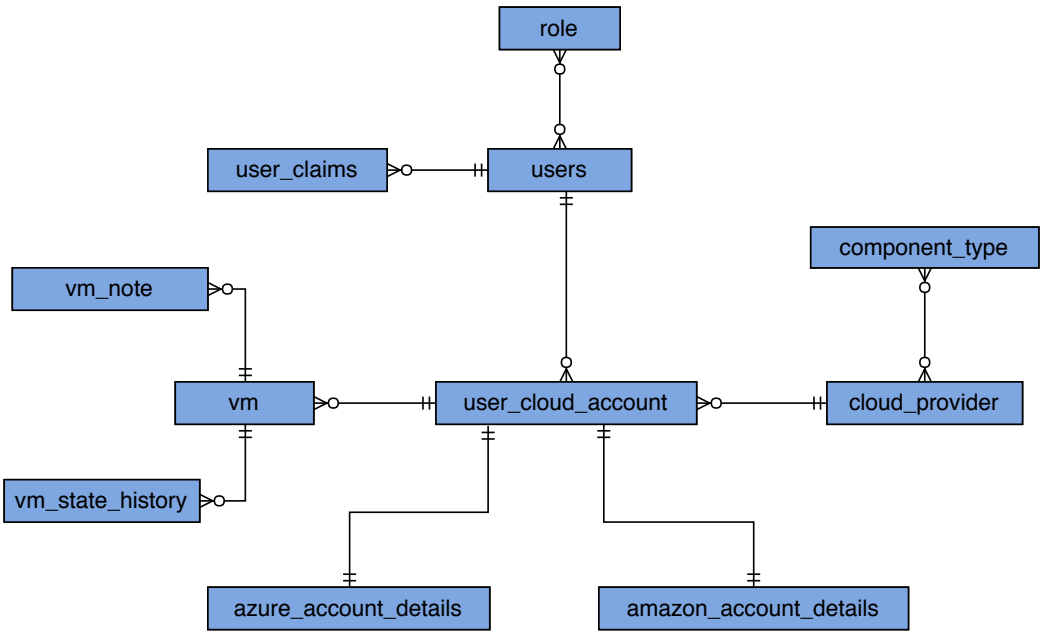

Figura 4.1: Modelo Entidade-Associação simplificado

Na Tabela 4.1 podemos verificar em detalhe o intuito de cada entidade da base de dados.

Tabela 4.1: Descrição das entidades da base de dados

Nome Tabela	Descrição
`users`	Identifica cada utilizador na aplicação. O campo de *email* é fundamental para o processo de autenticação.
`role`	Todos os perfis existentes na aplicação. Um exemplo é o perfil de administrador.
`user_claims`	Permite guardar os atributos que identificam um utilizador, por exemplo o seu nome completo ou *e-mail*.
`user_cloud_account`	Cada utilizador terá uma ou muitas contas de *cloud* registadas na aplicação. Uma conta pode ser ativada ou desativada sempre que o utilizador desejar.
`cloud_provider`	Contém todos os operadores de *cloud* suportados pela aplicação.
`azure_account_details`	Contém os detalhes de uma conta do Azure.
`amazon_account_details`	Contém os detalhes de uma conta da Amazon.
`vm`	Regista todas as máquinas virtuais criadas através da aplicação.
`vm_state_history`	Regista todas as ações efetuadas numa máquina virtual através da aplicação.
`vm_note`	Regista todas as notas criadas sobre máquinas virtuais.
`component_type`	Guarda todos os tipos de componente existentes na aplicação.
`cloud_provider_component_type`	Associa os componentes aos operadores que tenham suporte para os mesmos.

As entidades `role` e `user_role` são fundamentais para ser mais simples limitar os acessos de cada utilizador dentro da aplicação. Por exemplo, pode existir um papel de administração que consegue ter acesso a todas as funcionalidades e configurações da aplicação, enquanto que um *role* de utilizador

normal apenas lhe dá acesso às funcionalidades base.

Cada utilizador poderá ter várias contas de *cloud* registadas (entidade `user_cloud_account`), do mesmo operador. Por exemplo, é possível ter duas contas na plataforma Azure. Este detalhe foi tido em conta visto que não se pretende limitar o uso da aplicação a uma única conta por operador. Num cenário real de utilização é mais provável que os utilizadores a que se destina esta aplicação, possuam diversas contas no mesmo ou em operadores diferentes.

As entidades que contêm os detalhes de cada tipo de conta, tais como `azure_account_details` e `amazon_account_details`, serão fundamentais na medida em que são extensões da tabela base, a `user_cloud_account`. Por cada operador (entidade `cloud_provider`), que existir, deverá ser criada uma nova tabela de *account details*.

A entidade `component_type` armazena todos os tipos de componentes existentes na aplicação. Um componente é uma parte da solução que permite realizar uma determinada tarefa, no contexto de uma máquina virtual. Os componentes são configurados por operador, sendo que o principal objetivo é que o seu funcionamento seja igual, independentemente do operador. No entanto, pode existir a necessidade de criar componentes específicos. Os identificadores de cada um serão usados para efetuar a ligação à entidade `cloud_provider`, sendo que esta associação origina a tabela `cloud_provider_component_type`, esta tem como objetivo indicar os componentes que cada operador pode visualizar e utilizar.

Na próxima secção, Camada de Acesso a Dados, será apresentado o modo como este modelo foi aplicado, bem como as técnicas utilizadas para realizar a gestão e interação com os seus dados.

4.2 Camada de Acesso a Dados

Esta camada é responsável pela integração entre o sistema de gestão de base de dados e a aplicação. Aqui serão realizadas todas as ações de escrita e leitura de informação para o sistema de armazenamento. A camada divide-se em duas bibliotecas, `UVMM.Domain`, apresentada na secção 4.2.1, que tem todas as entidades do modelo de dados mapeadas em objetos e a `UVMM.Repository`, apresentada na secção 4.2.2, que realiza todo trabalho de leitura e escrita de dados.

4.2.1 Biblioteca de Domínio — `UVMM.Domain`

Esta biblioteca contém o mapeamento das entidades da base de dados para objetos. Foi utilizada a Entity Framework Core em conjunto com a biblioteca `Npgsql` [32], que permite a ligação entre uma base de dados PostgreSQL e uma aplicação .NET Core.

A Entity Framework é já um dos *Object-Relational Mapper* (ORM) mais populares para o .NET, a versão Core desta *framework* surgiu no seguimento da nova versão do .NET, e trouxe consigo uma melhoria no seu desempenho [33].

Um ORM é uma técnica que permite consultar e alterar informações de uma base de dados, usando um paradigma orientado a objetos. É criada uma biblioteca, escrita na linguagem de programação que se pretender, com todas as entidades da base de dados mapeadas em objetos, incluindo as dependências entre si, sendo estas obtidas através das relações entre as entidades. Na Figura 4.2 pode observar-se o modo de funcionamento de um ORM.

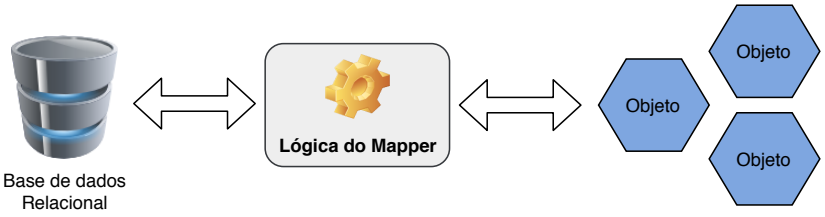

Figura 4.2: Funcionamento de um *Object-Relational Mapper*

No caso da presente solução, as classes C# que representam as entidades da base de dados utilizam as anotações da biblioteca `DataAnnotations` para garantir o correto mapeamento das tabelas e colunas. Isto porque na base de dados as tabelas e colunas têm o seu nome em letras minúsculas e cada palavra é separada por um *underscore*, e no código pretende-se usar *Camel Case*. Esta é a denominação em inglês para a forma de escrever palavras compostas ou frases, onde cada palavra é iniciada com maiúsculas e as várias palavras são unidas sem espaços. Por exemplo, a frase "Primeiro Nome" será convertida para "PrimeiroNome".

A título de exemplo a Figura 4.3 apresenta o mapeamento da entidade `user_cloud_account` para a classe `UserCloudAccount`, que disponibiliza um conjunto de atributos, bem como os respetivos *getters* e *setters*, e que permite esconder do utilizador a complexidade do código SQL.

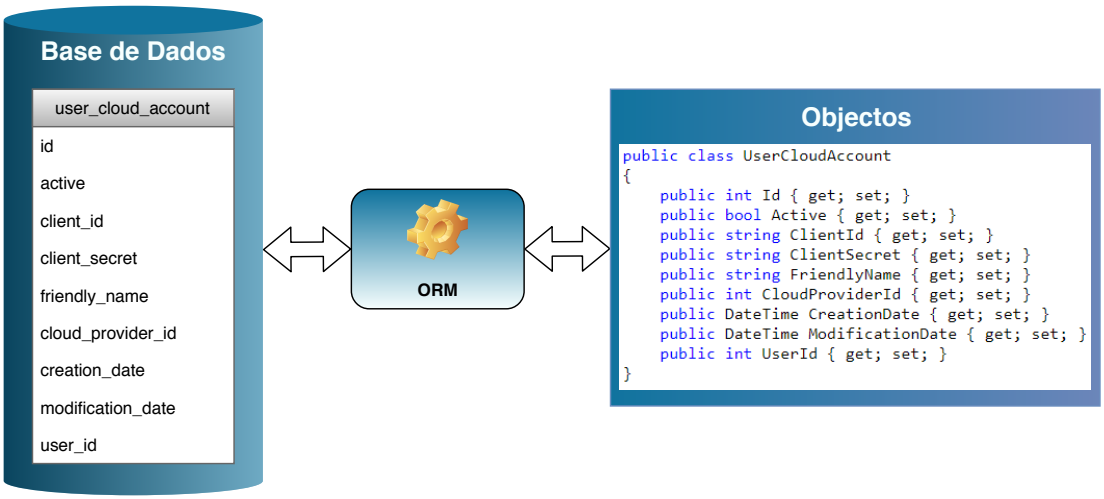

Figura 4.3: Mapeamento da entidade `UserCloudAccount`

O atributo `Column` é aplicado às propriedades das classes de entidade para configurar o nome da coluna correspondente. Sendo assim a tabela que guarda os dados das contas do utilizador, `user_cloud_account`, na base de dados corresponde à classe `UserCloudAccount` na solução. Da mesma forma que a coluna `client_id` corresponde a `ClientId` na classe.

Manutenção da Base de Dados

Em soluções onde a arquitetura é centrada nas classes de domínio, a base de dados é criada após a definição das entidades, através de processos realizados de modo automático com o auxílio de ferramentas específicas ou *frameworks*. Neste sentido, foi aplicado o padrão *Code First* [34] com a ajuda da Fluent API da Entity Framework Core [35]. Esta *framework* permite realizar o mapeamento das classes e gerar a base de dados a partir delas. Todas as operações necessárias são feitas através da linha de comandos, uma vez que uma das grandes novidades que a versão .NET Core oferece, é a possibilidade de desenvolver aplicações utilizando a linha de comandos, que se denomina .NET CLI (*Command-Line Interface*). O comandos utilizados estão presentes na Tabela 4.2.

Tabela 4.2: Comandos para gestão da base de dados

Comando	Descrição
add-migration	Inicialmente é definido um domínio tendo em conta as classes existentes. Cada migração seguinte irá conter as alterações necessárias a efetuar comparativamente ao que existe na base de dados.
update-database	Cria ou atualiza a base de dados tendo em conta o contexto, as classes de domínio e a última migração realizada.

O processo de manutenção da base de dados baseia-se neste conjunto de comandos. Devido a estas migrações que a Entity Framework Core proporciona, sendo que permitem criar ou atualizar a estrutura da base de dados baseando-se nas classes definidas no domínio, garante-se que todas as alterações feitas nestas classes são espelhadas na base de dados, bastando para isso adicionar uma nova migração. Com estas ações consegue-se manter a base de dados sempre atualizada.

4.2.2 Biblioteca de Repositório — `UVMM.Repository`

A função desta biblioteca é aceder à base de dados, efetuando todo o tipo de ações sobre esta. Cada entidade da base de dados terá um repositório dedicado.

Nesta camada foi usado o *Repository Pattern* [36], sendo este padrão muito utilizado devido às suas vantagens. Uma das qualidades que mais se destaca é a capacidade de uniformizar o código com a utilização de genéricos. Outra das vantagens é ajudar a ter uma solução devidamente estruturada em camadas, aumentando o desacoplamento entre elas. Com isto a lógica de negócio não sabe como ou onde os dados são armazenados. Apenas tem conhecimento que pode escrever e ler usando a camada de repositório.

Além deste foi também usado o padrão *Unit Of Work* que, segundo Martin Fowler [37], mantém uma lista de objetos afetados por uma transação, coordena a escrita das alterações e trata possíveis problemas de concorrência. Pode ser visto como um contexto, sessão ou objeto que acompanha as alterações das entidades durante uma transação.

A Figura 4.4 apresenta as classes `Repository` e `UnitOfWork`.

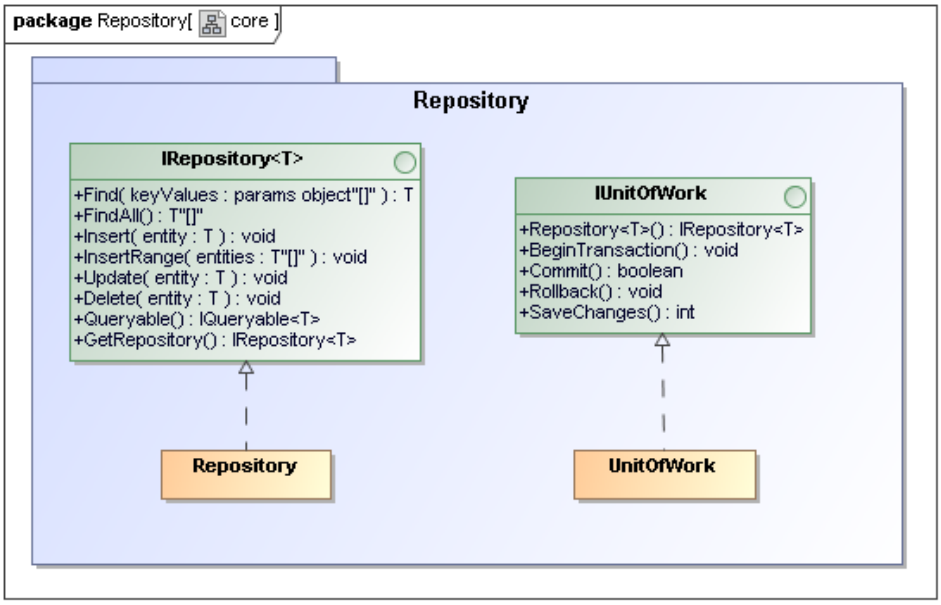

Figura 4.4: Diagrama de classes da biblioteca `UVMM.Repository`

A classe `Repository`, tem como objetivo definir as operações *Create*, *Read*, *Update* e *Delete* (CRUD), para qualquer entidade da base de dados. Caso seja necessário, é possível criar um repositório específico para cada entidade com vista a aumentar as funcionalidades deste. Esta realidade aplica-se principalmente para a definição de operações de leitura que são exclusivas em cada entidade.

A classe `UnitOfWork`, tem como objetivo definir os métodos essenciais ao controlo de escrita de informação na base de dados. Em primeiro lugar mantém as alterações das entidades em memória. Posteriormente envia essas atualizações para a base de dados com uma transação única. Após os objetos sofrerem todas as mudanças necessárias, como inserir, atualizar ou remover, durante uma transação, quando esta for concluída todas estas atualizações serão remetidas, como um agregado para ser persistido fisicamente na base de dados de uma só vez. No método `BeginTransaction` é definido o nível de isolamento `ReadCommited` para as transações. Este nível permite que apenas sejam lidos os dados que já estão *commited* na base de dados, garantindo a integridade dos dados na medida em que nenhuma outra transação irá obter dados que foram criados ou alterados durante outra qualquer transação. O problema em não usar um nível com estas características surge quando é necessário terminar a transação, descartando todas as alterações efetuadas (`rollback`), e outras transações já fizeram alguma lógica com dados que vão deixar de existir.

4.3 Camada de Negócio

Esta camada trata das regras necessárias para realizar tarefas como: i) Garantir a integridade dos dados; ii) Realizar os algoritmos mais exigentes que o negócio assim o exigir; iii) Efetuar as validações da informação enviada da camada de apresentação.

Existem duas bibliotecas contidas nesta camada, uma para gerir o negócio da própria aplicação, `UVMM.Service` apresentada na secção 4.3.1, e outra para gerir a lógica dos operadores de *cloud*, `UVMM.ServiceCloud` apresentada na secção 4.3.2.

4.3.1 Biblioteca de Serviço UVMM — `UVMM.Service`

Esta biblioteca tem como principal objetivo receber os pedidos dos controladores, efetuar a lógica de negócio necessária e comunicar com a camada de acesso a dados.

Cada entidade da base de dados terá um serviço dedicado para o controlo das operações de CRUD e outras que sejam necessárias. Todos os serviços terão uma interface dedicada para possibilitar a injeção de dependências. Esta técnica tem as seguintes vantagens tais como: i) Código mais reutilizável; ii) Código mais simples de testar; iii) Código mais legível.

Injeção de dependências é um padrão que implementa o princípio de Inversão de Controlo para resolver dependências de objetos. Por outras palavras, quando se está a desenvolver código, iremos criar e usar diferentes classes. Uma classe (Classe A) pode usar outras classes (Classe B e/ou D). Assim, as classes B e D são dependências da classe A. Uma analogia simples será uma classe `Carro`. Um carro pode depender de outras classes como `Motor`, `Pneus`, entre outras. A injeção de dependências sugere que, em vez das classes dependentes, neste caso a Classe A, criarem as suas dependências (Classe B e Classe D), a classe deve ser injetada com a instância concreta da dependência.

4.3.1.1 `ServiceCore`

Este serviço tem como objetivo disponibilizar o controlo de transações à camada de apresentação. Para isso, irá fazer a chamada dos métodos da classe `UnitOfWork` da camada de repositório, cujo objetivo é a gestão de transações. O método `BeginTransaction` deve ser chamado aquando do início da lógica de criação de um objeto. No final de toda a lógica de criação ser aplicada, caso não exista nenhum erro deve ser chamado o método `Commit`, caso contrário deverá ser chamado o método `Rollback`. Durante um determinado processo

a base de dados pode sofrer várias alterações nos seus dados, o que pode levar a muitos pedidos pequenos para escrita de informação, aumentando assim o tempo necessário para efetuar a lógica de negócio. O padrão *Unit Of Work* garante uma melhoria no desempenho e permite uma maior integridade dos dados, uma vez que, uma transação só é finalizada com sucesso caso todas as pequenas alterações sejam válidas.

4.3.2 Biblioteca de Serviço *Cloud* — `UVMM.ServiceCloud`

Esta biblioteca destina-se às implementações para suportar os operadores de *cloud*. Aqui são definidas as interfaces que devem ser cumpridas por cada novo operador, para que exista um ponto central uniforme entre os distintos operadores.

4.3.2.1 `ServiceCloudManager`

Esta classe é o ponto fulcral desta camada, uma vez que é aqui que chegarão todos os pedidos relacionados com máquinas virtuais. O seu papel será de intermediário para reencaminhar os pedidos para a implementação do operador correta.

A pasta de `interfaces` contém todas as que existem e que podem ser usadas nesta camada.

- `IVirtualMachineService` - define os métodos que cada operador deve implementar para suportar a gestão de máquinas virtuais.
- `IRestServiceCloud` - define os métodos necessários para servir a REST API.
- `IAuthenticationService` - define os métodos necessários para a autenticação nos operadores.

Para um melhor entendimento podemos observar a Figura 4.5, que mostra a relação entre as interfaces. Os métodos de cada interface podem ser vistos em detalhe na Figura C.7, nos Anexos.

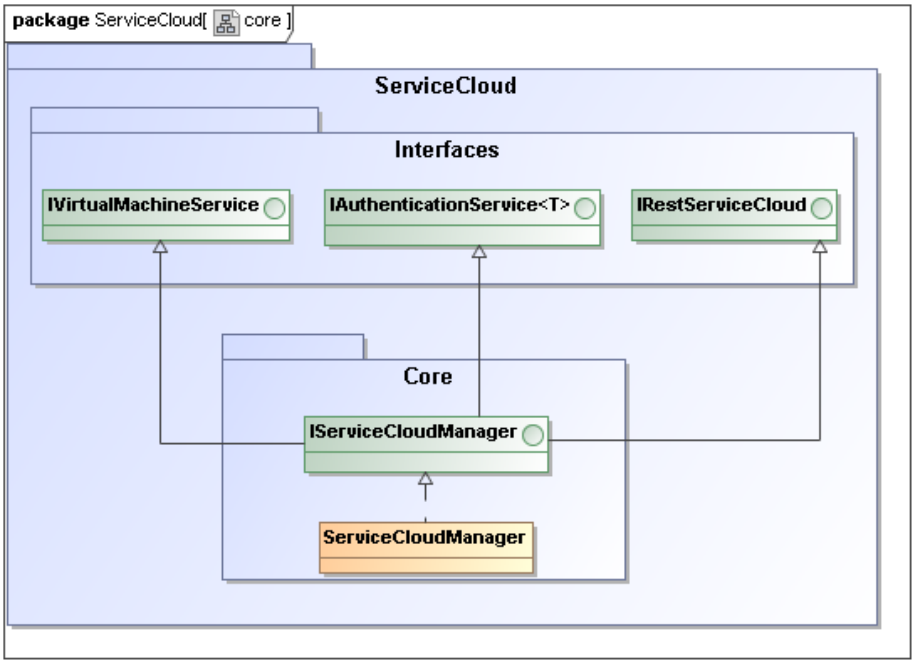

Figura 4.5: Diagrama UML simplificado das interfaces da biblioteca `UVMM.ServiceCloud`

Sendo assim a classe de `ServiceCloudManager` terá de implementar a interface de `IVirtualMachineService`, para que em cada método seja capaz de chamar o método do serviço correto. Dependendo da conta do utilizador que estiver a ser utilizada para efetuar uma ação sobre uma máquina virtual, o algoritmo terá de ser capaz de interpretar a que operador pertence e para que implementação de `IVirtualMachineService` redirecionar o pedido.

A interface `IRestServiceCloud` tem como intuito definir os métodos que são suportados pela REST API da solução. Esta difere da anteriormente apresentada, na medida em que os parâmetros de entrada dos métodos são os atributos estritamente necessários, não sendo usados objetos do UVMM. Deste modo o objeto `UserCloudAccount` é repartido em 3 parâmetros: `clientId`, `clientSecret` e `cloudProviderName`.

Por último, a interface `IAuthenticationService` possui apenas o método `Authenticate` para ser implementado. Este método define um tipo de retorno genérico que é definido pela classe que a implementar. Esta generalização garante que qualquer tipo de retorno é permitido, possibilitando a cada operador efetuar a sua lógica de autenticação para efetuar os pedidos à sua API.

4.3.2.2 `CloudControl`

Para simplificar o processo anteriormente descrito foi criado um atributo cujo objetivo é identificar cada classe de operador. Este atributo apenas exige que seja definido o nome do operador de *cloud* associado à implementação. No *manager* será criada uma lista com todas as classes que implementam a interface de máquina virtual e, por cada pedido que chegue será comparado o nome do operador da conta com o atributo de cada classe da lista, retornando uma instância da classe cujo atributo coincide com o nome do operador na base de dados.

4.3.2.3 Azure REST API

A implementação para suportar o operador Azure foi feita tendo como base a sua REST API [38] disponibilizada e documentada. Esta API fornece várias operações sobre uma grande variedade de recursos. No caso da presente solução o componente mais usado foi o `Compute`, sendo que é o que dá acesso às máquinas virtuais.

A maioria dos serviços do Azure exigem que os pedidos sejam autenticados com credenciais válidas antes de poder enviar esses mesmos pedidos à API. A autenticação é coordenada pela *Active Directory* do Azure e fornece ao cliente um *token* de acesso para comprovar que está autenticado. Este *token* é enviado para o serviço do Azure no cabeçalho HTTP *Authorization* nos pedidos à REST API. Além de que este *token* de segurança fornece também informações adicionais ao serviço, permitindo que este valide se o cliente tem as autorizações necessárias para executar a operação requisitada. Para configurar o acesso à REST API têm de ser feitos os seguintes passos:

1. Registar a aplicação cliente na *Active Directory* do Azure.

2. Definir as permissões para que o cliente tenha autorização para aceder à API do Azure Resource Manager.

3. Extrair o ID da aplicação cliente (*clientId*) criada, gerar uma chave (*client secret*) e obter o identificador do diretório (*tenantId*) nas propriedades da *Active Directory* do Azure.

Além dos 3 identificadores (*clientId*, *client secret*, *tenantId*), apresentados anteriormente, é ainda necessário extrair o identificador da subscrição. Em conjunto estas são as variáveis necessárias para configurar uma conta deste operador.

Modelos JSON

As respostas aos pedidos que são feitos à API do Azure são obtidas no formato *JavaScript Object Notation* (JSON). Devido à complexidade de algumas das respostas, nomeadamente quando se obtém uma máquina virtual, foi necessária a definição de modelos para onde são mapeadas as respostas. Uma máquina virtual tem um conjunto de propriedades, que por sua vez tem um perfil de hardware, de armazenamento, de sistema operativo, entre outros.

4.3.2.4 Amazon EC2 SDK

A Amazon tem também a sua REST API [39], no entanto disponibiliza *kits* de desenvolvimento para várias linguagens de programação, incluindo .NET Core [40], o que proporciona uma abstração para o programador, facilitando assim o desenvolvimento. A biblioteca que é utilizada tem as suas próprias classes, sendo que existe uma para definir uma máquina virtual. No entanto, e como o objetivo é unificar o conceito de máquina virtual, é necessário converter este objeto para a classe de máquina virtual que foi criada para ser comum a qualquer operador, contendo na sua descrição atributos que são transversais.

Para ter acesso à conta da Amazon são necessárias algumas configurações por parte do utilizador numa fase inicial:

1. Criar um *Security Group* no serviço EC2.
2. Criar um *Key Pair* no serviço EC2.
3. Criar um utilizador para a aplicação no serviço Identity & Access Management.

Um *Security Group* atua como uma *firewall* virtual que controla o tráfego de uma ou mais instâncias. Ao criar uma nova instância de máquina virtual é fundamental associar a um ou vários grupos deste tipo. Para cada grupo podem ser definidas várias regras que permitem o tráfego com destino e/ou origem nas instâncias associadas.

Relativamente ao *Key Pair*, a Amazon EC2 usa criptografia de chave pública/privada para encriptar e desencriptar os dados de autenticação. A criptografia deste tipo usa uma chave pública para encriptar a informação, como por exemplo uma palavra-passe, e o destinatário usa a chave privada para desencriptar os dados. Em suma os pares de chaves são constituídos por uma chave pública e uma chave privada. A chave privada é usada para

criar uma assinatura digital e, em seguida, a Amazon usa a chave pública correspondente para validar a assinatura.

Para obter um *clientId* e um *client secret* para autenticar a aplicação, é necessário criar um utilizador através do serviço de Identity & Access Management (IAM). Este utilizador deve ser associado à permissão *AmazonEC2FullAccess*.

4.3.2.5 Tradutores de Respostas

Tal como foi apresentado anteriormente, cada operador tem as suas respostas num determinado formato, tanto pode ser uma classe sua como pode ser em JSON. Cabe à aplicação conseguir converter essa informação recebida, para os objetos que são conhecidos e utilizados na lógica de negócio. Para cada operador deverá ser implementada uma classe que realize esse mapeamento. Na Figura 4.6 pode-se verificar o modo de funcionamento da lógica dos tradutores.

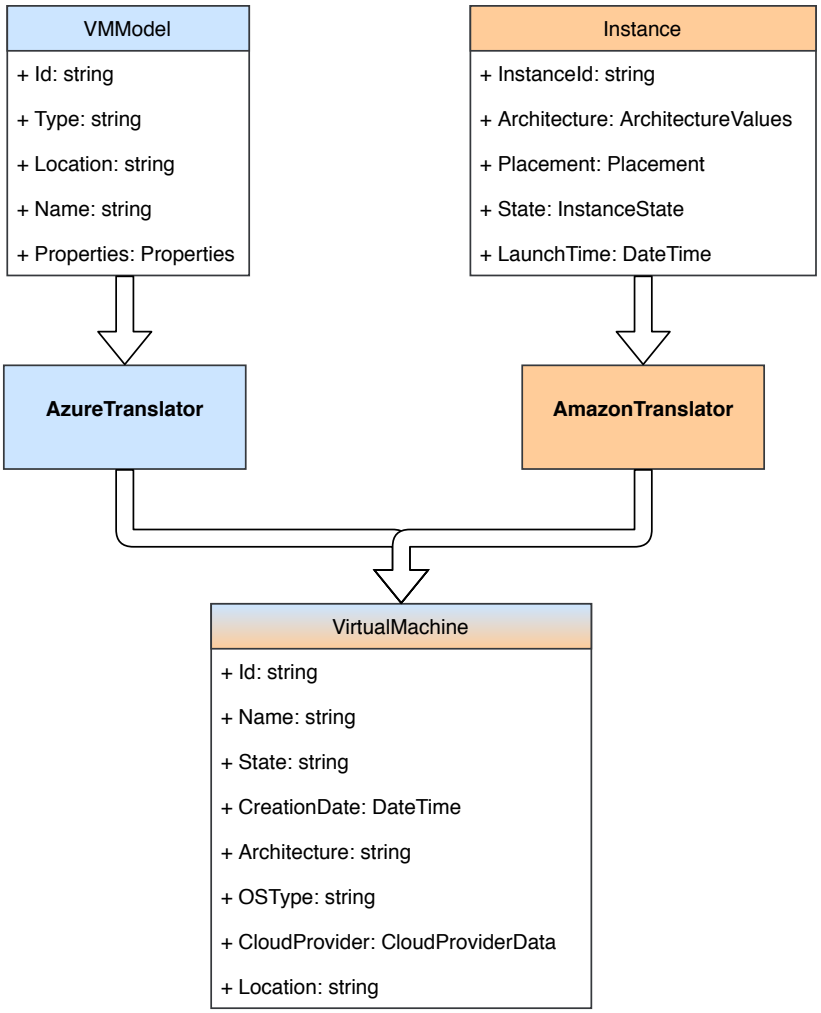

Figura 4.6: Modo de funcionamento dos tradutores

Entenda-se que as classes a azul são do Azure e a laranja são da Amazon. O `VMModel` foi o objeto que foi criado para mapear o modelo JSON que é recebido da API do Azure, relativamente a uma máquina virtual. Já uma `Instance` corresponde ao objeto da biblioteca da Amazon que identifica e caracteriza uma máquina virtual. Estes dois objetos passam cada um pelo tradutor respetivo e ambos resultam num objeto unificado, `VirtualMachine`, que tem como objetivo ser o ponto comum no que corresponde a uma máquina virtual. Este objeto encontra-se na biblioteca `UVMM.DataModel` que tem como objetivo conter todos os objetos que são utilizados para mapear os dados provenientes dos diferentes operadores de *cloud*. Cada objeto definido teve em conta os atributos fundamentais para identificar um recurso. Um exemplo disso mesmo, é o objeto `VirtualMachine` aqui apresentado.

4.3.2.6 Concretização no Operador Amazon EC2

Esta secção tem como objetivo apresentar os desenvolvimentos que foram necessários realizar para dar suporte ao operador Amazon EC2. Com esta explicação pretende-se exemplificar o conjunto de tarefas que implica o suporte a um operador.

Para a implementação do suporte a este operador foi necessário criar uma classe que implementa as interfaces para permitir a gestão de máquinas virtuais, `IVirtualMachineManagement`, e para realizar a autenticação do utilizador de modo a efetuar pedidos à sua API, `IAuthenticationService`. A implementação da primeira interface referida, permite fornecer suporte às operações de gestão de uma máquina virtual. A segunda serve para efetuar a autenticação no operador seguindo as suas próprias regras. Com a implementação destes dois contratos é garantido o suporte, ao nível da camada de negócio de *cloud*, deste operador. Para qualquer outro operador que se pretenda incorporar na solução é imprescindível a implementação destas mesmas interfaces. A classe que implementa estas interfaces necessita ter a anotação `CloudControl` com o valor "Amazon", sendo que é este o nome do operador na tabela `CloudProvider`. No caso do operador em questão, os métodos da interface que realiza a gestão de máquinas virtuais, foram desenvolvidos com o SDK disponibilizado pelo próprio, para realizar a comunicação com a sua API. Foi necessário criar um tradutor para este operador, `AmazonTranslator`, para converter os objetos recebidos da biblioteca utilizada nos objetos uniformizados da solução, presentes na biblioteca `UVMM.DataModel`.

4.3.3 Biblioteca de Serviço de E-Mail — `UVMM.ServiceMail`

As notificações por *e-mail* são muito convenientes, pois permitem garantir que o utilizador está sempre informado acerca de questões relacionadas com a sua conta ou mesmo com a aplicação. Este foi o motivo que levou à implementação desta biblioteca, sendo que é uma peça que enriquece a solução. Na Figura 4.7 é apresentada a arquitetura de classes da presente biblioteca.

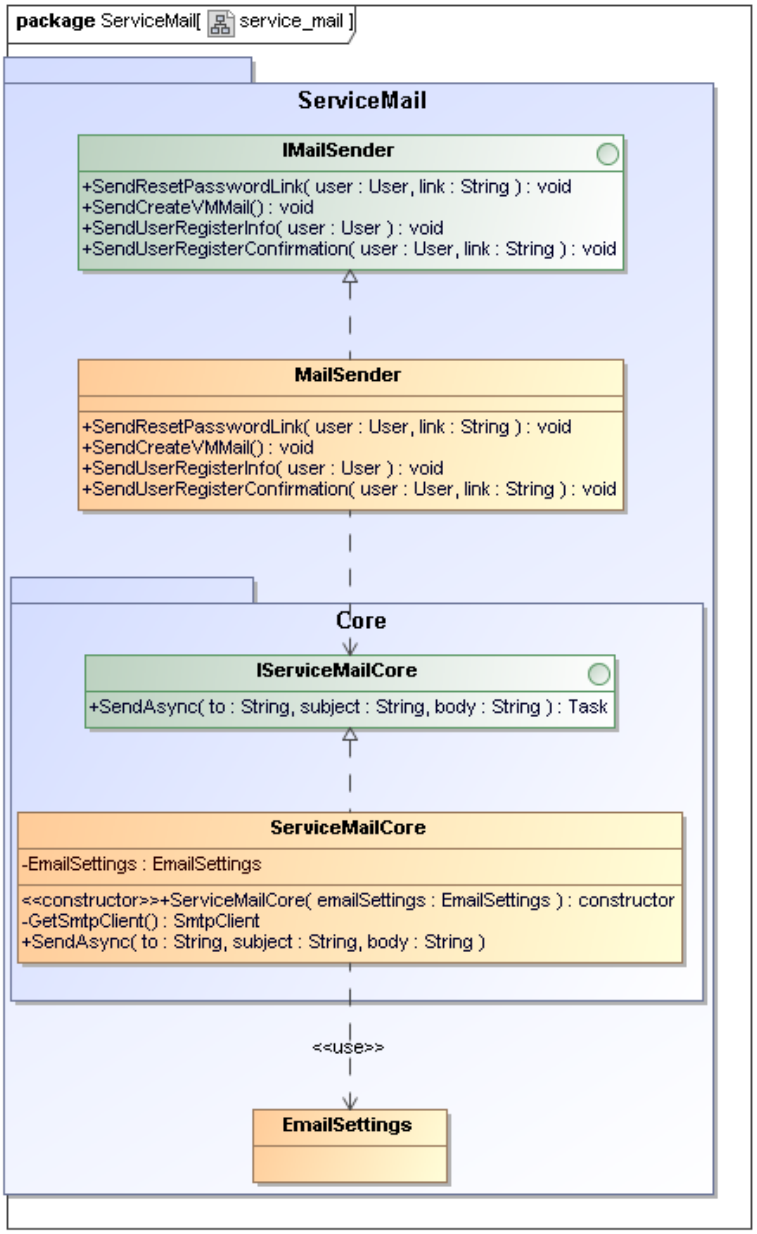

Figura 4.7: Diagrama UML da biblioteca `UVMM.ServiceMail`

4.3.3.1 `ServiceMailCore`

Esta é classe principal desta biblioteca, visto que é aqui que são efetivamente enviados os *e-mails*. No envio de uma notificação deste tipo, há sempre propriedades que são comuns independentemente do cariz da mesma. Por

esse motivo o método `SendAsync` apenas recebe como parâmetro os atributos que podem variar: *e-mail* de destino, assunto e corpo da mensagem. O método é, tal como o nome indica, assíncrono para não criar atrasos durante a execução e garantir que o utilizador pode continuar a realizar outras tarefas, enquanto o método termina.

`EmailSettings`

A classe anteriormente apresentada depende totalmente da presente classe. É aqui que são mapeados os valores dos atributos existentes no ficheiro *app-settings*, dos quais estão presentes na Tabela 4.3.

Tabela 4.3: Atributos para configurar *e-mail*

Atributo	Descrição
SendMails	Permite configurar se o envio de *e-mails* está ativo ou não.
SmtpServer	Servidor SMTP usado para enviar os *e-mails*.
Port	Número da porta no servidor.
UsernameEmail	Endereço de *e-mail* usado para ser o remetente.
UsernamePassword	Palavra-passe do *e-mail* usado para ser o remetente.
FromEmail	Endereço de *e-mail* usado para ser o remetente.
EnableSSL	Indicar se é necessário ou não uma comunicação SSL ao servidor de *e-mail*.

O servidor escolhido foi o Gmail uma vez que este permite enviar no máximo 500 *e-mails* diários [41], sem ter qualquer custo associado. Caso este número seja excedido, a Google poderá bloquear temporariamente a conta de Gmail sem qualquer aviso e, serão necessárias 24 horas para que seja possível recuperar o acesso ao *e-mail*. Foi tido em conta que o Gmail não foi desenhado para enviar *e-mails* em massa. Foi criada uma nova conta

no Gmail (*uvmm.app.notification@gmail.com*) para ser o remetente de todas as notificações da aplicação.

4.3.3.2 `MailSender`

A presente classe tem como objetivo definir os vários métodos para enviar as diferentes notificações da aplicação. O ponto fundamental é a injeção da dependência `IServiceMailCore`.

Os métodos desta classe apenas devem definir o *e-mail* do destinatário, o assunto e o corpo da mensagem, sendo que são os parâmetros de entrada do método `SendAsync` da dependência injetada.

4.4 Camada de Apresentação

Esta camada tem como objetivo apresentar a informação ao utilizador de forma a que o mesmo consiga interpretar e interagir com a aplicação, efetuando ações que serão reencaminhadas para a camada de negócio. Define a lógica de apresentação da aplicação, esta sabe o que mostrar, quando mostrar e a quem mostrar.

Esta camada teve como base de implementação o padrão *Model-View-Controller* (MVC), este terá uma secção dedicada para ser apresentada a motivação que levou à escolha deste padrão. Serão ainda apresentadas as funcionalidades que a aplicação proporciona, sendo também explicada a lógica de cada uma. Na apresentação dessas funcionalidades serão apresentadas algumas páginas, para permitir mostrar como os controladores funcionam. Por exemplo, na explicação do registo de utilizador, será apresentada a lógica aplicada no controlador e a sua respetiva página, permitindo visualizar a informação que é apresentada ao utilizador.

4.4.1 Padrão *Model-View-Controller*

O padrão MVC gere a forma como os componentes comunicam: o utilizador realiza uma ação e, em resposta, a aplicação retorna um modelo que é mostrado numa vista. O ASP.NET Core MVC implementa uma variante do padrão MVC, sendo que é adequada principalmente para aplicações Web.

Na Figura 4.8 pode-se observar uma demonstração da comunicação que é estabelecida no MVC.

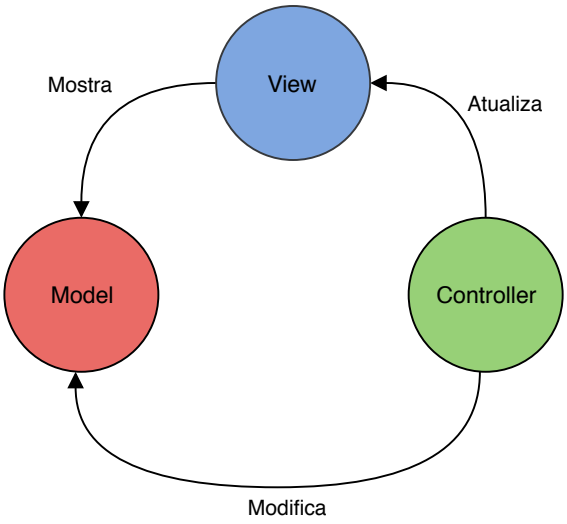

Figura 4.8: Padrão *Model-View-Controller*

A utilização deste padrão tem como benefício o isolamento da lógica de negócio da lógica de apresentação. Isto permite que se possa alterar o modo como a informação é mostrada ao utilizador, sem modificar a forma como se obtém essa informação, ou seja, consegue-se alterar a lógica de apresentação sem alterar as regras de negócio.

A comunicação entre a interface de utilizador e as regras de negócio é definida através de um controlador. Quando algum evento é executado na interface gráfica, como clicar num botão, a interface irá comunicar com o controlador que por sua vez comunica com a camada de serviço.

4.4.2 Autenticação e Autorização

Por um lado a autenticação é o processo de saber quem é o utilizador, enquanto que a autorização é o processo de garantir que o utilizador apenas tem acesso ao que tem permissão.

Esta é uma das funcionalidades mais importantes em qualquer aplicação que seja baseada em utilizadores, sendo necessário fornecer a estes um mecanismo de autenticação seguro e fidedigno.

Para garantir uma maior robustez foi incorporada a biblioteca `Identity` do .NET Core. Esta facilita o processo de criação de contas de utilizador, autenticação e a gestão dos privilégios que este vai ter na aplicação, ou seja, a autorização. Para compreender o modo de funcionamento desta biblioteca é fundamental entender o conceito de `Claim`. Um `Claim` nada mais é que um atributo que está relacionado com o utilizador que está autenticado. Criando uma analogia simples, uma pessoa tem várias características como o

peso, altura, data de nascimento, nome, entre outras. Todos estes atributos formam os `Claims` desta pessoa. Ao utilizar o `Identity` para autenticar um utilizador, é criado um objeto do tipo `ClaimsPrincipal` que contém todos os atributos, com o respetivo valor, que estão definidos para um utilizador da aplicação. Existem já vários tipos de `Claims` no enumerado `ClaimTypes`, no entanto estes podem não ser suficientes ou os seus nomes podem não se adequar ao que se pretende na lógica da aplicação. Para expandir este leque de opções e poder ser específico foi criado o enumerado `ApplicationClaimTypes` para o efeito.

A biblioteca cria tabelas específicas na base de dados para armazenar toda a informação de um utilizador, que é necessária para o registar e posteriormente autenticar. Na Tabela 4.4 podemos verificar as tabelas que intervêm neste processo, e as respetivas alterações realizadas no que toca à sua denominação.

Tabela 4.4: Alterações dos nomes das entidades da biblioteca `Identity`

Classe `UVMM`	Classe `Identity`
`users`	`AspNetUsers`
`role`	`AspNetRoles`
`user_role`	`AspNetUserRoles`
`user_claims`	`AspNetUserClaims`

Na classe `UvmmContext` pode-se alterar várias informações das tabelas, como o nome da própria tabela, o nome das colunas, entre outras características que são configuráveis, no método `OnModelCreating`. Neste sentido foram feitas as alterações dos nomes das tabelas, como foi apresentado anteriormente na Tabela 4.4, para manter a coerência sintática. O mesmo foi aplicado para as colunas, ou seja, as palavras passaram para letra minúscula e cada uma ficou separada por um *underscore*.

4.4.2.1 Registo de Utilizador

Para que um utilizador possa usar a aplicação necessita de se registar. Para isso este deve fornecer um endereço de *e-mail* válido, bem como o seu primeiro e último nome e uma *password*. Na Figura 4.9 podemos ver em detalhe a página de registo.

Figura 4.9: Página de registo

Após efetuar o registo com sucesso, é gerado um *token* que é enviado juntamente com um *link*, através de uma notificação por *e-mail*, que reencaminha o utilizador para a página de confirmação do *e-mail*. Na Figura 4.10 pode observar-se a página que se obtém ao clicar no *link* que é enviado no *e-mail*. Nesta página foi colocado um *link* que redireciona o utilizador para a página de *login*.

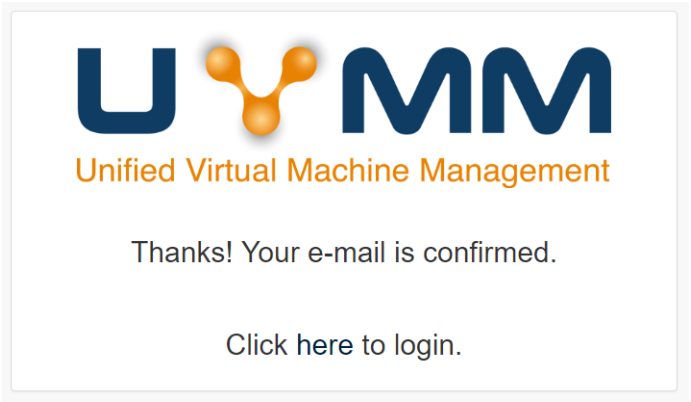

Figura 4.10: Página de confirmação de *e-mail*

4.4.2.2 *Login*

Após o utilizador confirmar o seu endereço de *e-mail* poderá efetuar o *login* na aplicação, Figura 4.11. Neste formulário é necessário colocar o *e-mail* e a respetiva *password*.

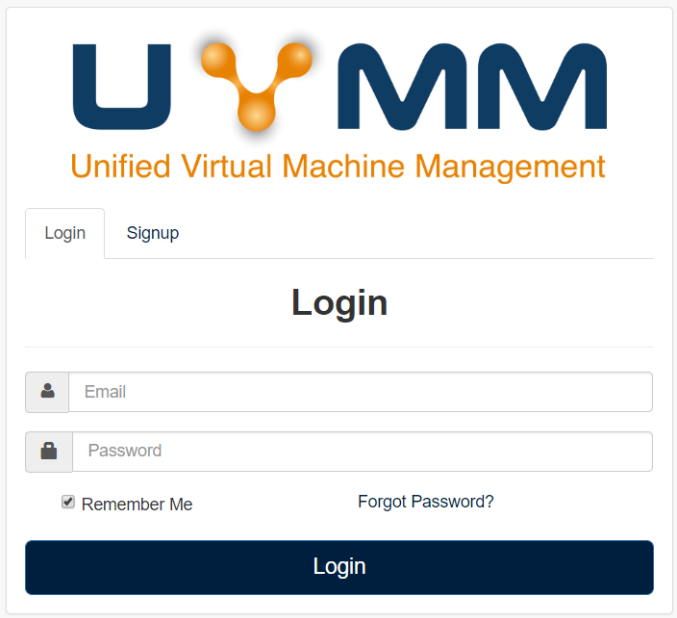

Figura 4.11: Página de *login*

`AccountController`

Para realizar todas as ações relacionadas com o registo e autenticação de utilizadores foi criado o presente *controller*. Este é o responsável por comunicar com os serviços do `Identity`.

No construtor são injetadas duas dependências que são fundamentais, o `UserManager` e o `SignInManager`. Na Tabela 4.5 podemos verificar que métodos destas dependências são utilizados.

A classe `UserManager` realiza a gestão dos utilizadores da aplicação, enquanto que a `SignInManager` realiza a gestão das sessões dos utilizadores.

`ClaimsPrincipalFactory`

Esta classe tem como objetivo acrescentar algum detalhe ao que já é feito quando se faz o *login*. Quando o utilizador é autenticado com sucesso é chamado o método `CreateAsync` da classe `UserClaimsPrincipalFactory` que efetivamente cria o conjunto de `Claims` do utilizador. Após esta associação

Tabela 4.5: Métodos das dependências `UserManager` e `SignInManager`

Classe	Método	Objetivo
`UserManager`	`CreateAsync`	Cria um novo utilizador para a aplicação.
`SignInManager`	`SignInAsync`	Cria uma sessão para um utilizador.
`SignInManager`	`PasswordSignInAsync`	Autentica um utilizador com o seu *email* e *password*. Este método é utilizado ao receber a informação do formulário de *login*.
`SignInManager`	`SignOutAsync`	Remove a sessão ativa de um utilizador, ou seja, efetua o *logout*.

são definidas as `Claims` específicas da aplicação, personalizando desta forma os atributos que estão ligados ao utilizador. O que esta *factory* faz é juntar estes dois processos, redefinindo o método `CreateAsync`.

`ApplicationPrincipal`

O objetivo desta classe é representar o utilizador e as suas `Claims`. Deste modo todos os atributos presentes estão expostos na Tabela 4.6.

A *factory* apresentada anteriormente é a responsável por atribuir os valores a cada um destes atributos.

AuthenticatedBaseController

Esta classe herda de `Controller` e apenas define a variável `ApplicationUser` que é uma instância do tipo `ApplicationPrincipal`. Todos os *controllers* devem estender esta classe para que tenham acesso ao utilizador que está autenticado.

É de grande importância ter acesso a esta informação nos controladores para que seja possível enviar o identificador do utilizador para os serviços, sempre que este seja um parâmetro requerido. Deste modo é possível aplicar as regras de negócio e validações tendo em conta o utilizador que fez o pedido.

Tabela 4.6: Atributos presentes nas `Claims`

Atributo	Descrição
Id	Identificador do utilizador na tabela `users`
UserName	*Username* do utilizador na aplicação
FirstName	Primeiro nome do utilizador na aplicação
LastName	Apelido do utilizador na aplicação
FullName	Junção do primeiro nome e apelido
Email	*E-mail* do utilizador na aplicação

4.4.2.3 Recuperação de *Password*

Esta funcionalidade é essencial ao bom funcionamento da aplicação, permitindo ao utilizador a possibilidade de recuperar a sua *password* no caso de não se lembrar. O mecanismo é muito simples, o utilizador fornece o endereço de *e-mail* que está associado à conta. Este primeiro formulário está apresentado na Figura 4.12.

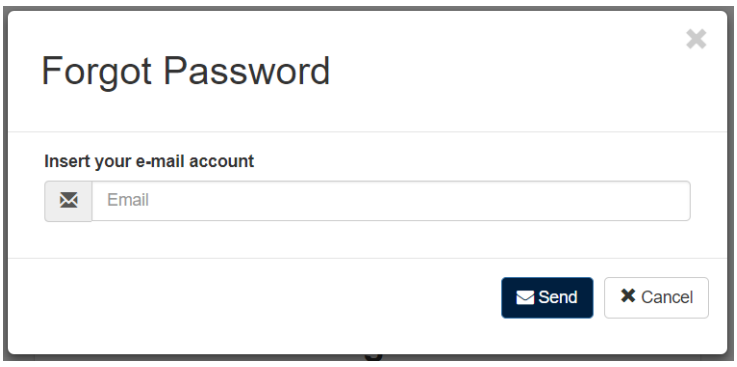

Figura 4.12: Página para recuperação de *password*

É verificado se esse utilizador existe, caso exista é gerado um *token* através do método `GeneratePasswordResetTokenAsync` da classe `UserManager`, e é enviado juntamente com uma notificação. Nessa notificação é incorporado um *link* que reencaminha o utilizador para a página de alteração de *password*, Figura 4.13.

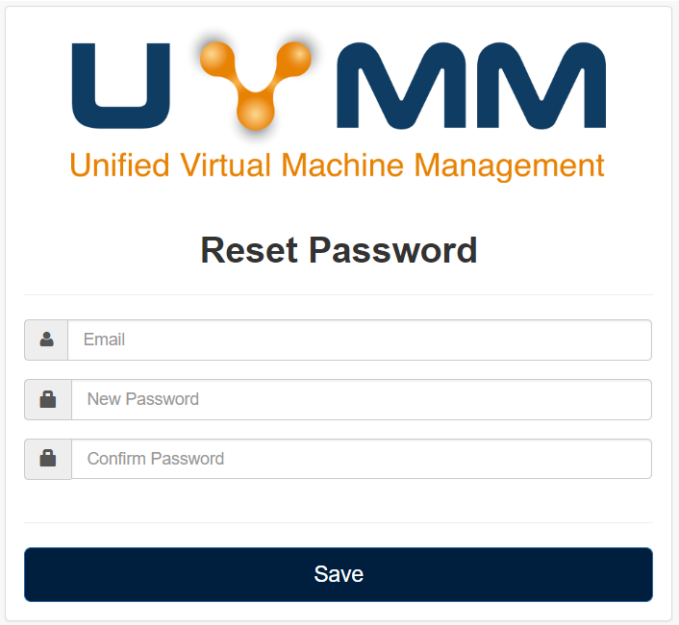

Figura 4.13: Página para definição de nova *password*

4.4.3 Página Inicial após *Login*

Na página inicial da aplicação o utilizador tem uma visão geral do que pode fazer e do que está a acontecer no momento. São mostradas informações úteis como o número de máquinas virtuais que estão em execução, sendo possível efetuar uma pesquisa automática por essas. Na Figura 4.14 podemos observar a informação que é exibida na página inicial.

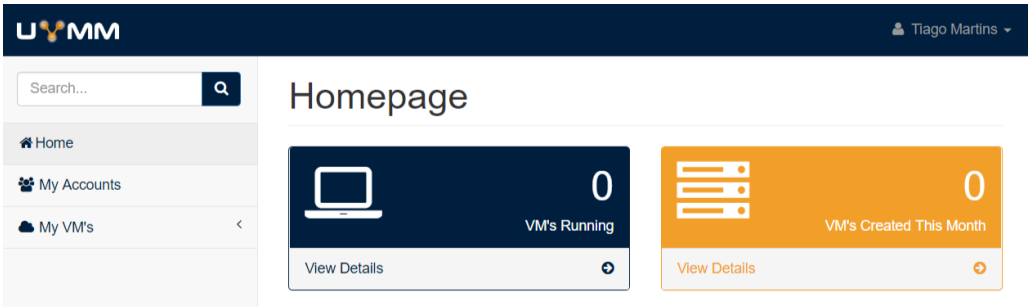

Figura 4.14: Página inicial da aplicação após *login*

No menu lateral, do lado esquerdo, estão os *links* para os ecrãs de gestão de contas, "My Accounts", e de criação e visualização de máquinas virtuais, "My VM's". No canto superior direito é onde é possível terminar a sessão,

clicando no nome de utilizador e selecionando a opção "Logout". Ao centro, existem duas caixas que têm como objetivo servir de acesso rápido a determinadas informações. Por um lado mostra o número de máquinas virtuais em execução naquele momento e efetua uma pesquisa por elas (caixa do lado esquerdo), clicando em "View Details". Por outro lado mostra o número de máquinas que foram criadas no mês atual, sendo igualmente possível efetuar uma pesquisa por elas (caixa do lado direito).

As informações apresentadas são geridas pelo `HomeController`. No entanto, é o `AccountController` que gere a sessão, logo é nele que é feita a ação de *logout*.

4.4.4 Gestão de Contas *Cloud*

Cada utilizador pode ter uma ou várias contas de *cloud* configuradas na aplicação, não existindo qualquer limite por operador. Esta gestão é feita pelo utilizador no ecrã de contas de *cloud*, Figura 4.15.

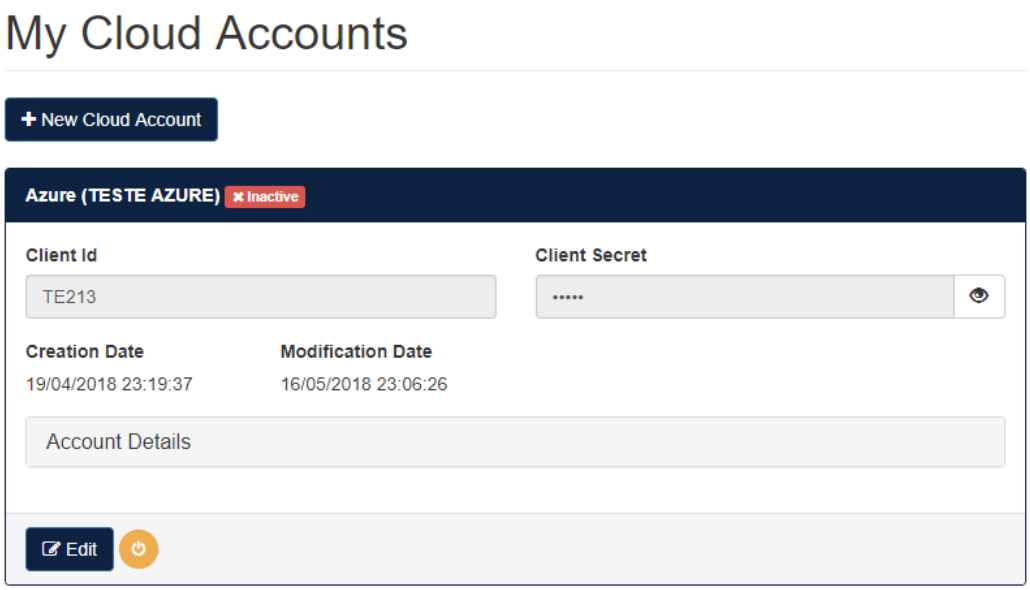

Figura 4.15: Página de gestão de contas *cloud*

Aqui podem ser modificados os vários parâmetros que identificam uma conta, sendo que, dependendo das especificidades do operador de *cloud*, existem detalhes que são exclusivos. Para garantir essa distinção e não comprometer a modularidade, cada operador tem uma tabela na base de dados dedicada, para guardar esses detalhes.

No entanto existem dois atributos que são transversais a qualquer conta de um operador, apresentados na Tabela 4.7.

Tabela 4.7: Atributos comuns entre operadores

Atributo	Descrição
Client Id	É um atributo que serve para o operador identificar a aplicação
Client Secret	É um atributo que é usado pelo operador para autenticar a aplicação sempre que esta solicitar o acesso à conta do utilizador

Ambos são fundamentais para identificar o utilizador no operador e garantir o acesso à gestão da conta.

Passando para a lógica de apresentação, o maior desafio foi conseguir apresentar um formulário que se adapta ao operador escolhido, isto é, que apresenta os campos específicos de cada um. A solução passou por criar *controllers* com diferentes responsabilidades:

- `MyAccountsController`

  Permite criar, editar e listar as contas de *cloud*. Contém ainda um mecanismo de redirecionamento para o controlador específico dos detalhes de cada operador.

- `AccountDetails(providerName)Controller`

  Controlador dedicado à gestão dos detalhes de um operador. Permite criar e editar os mesmos.

- `AccountDetails(providerName)`

  Permite mostrar os detalhes do operador.

Este ecrã apresenta ao utilizador a listagem de todas as contas que este tem criadas na aplicação. Pode criar uma conta nova sempre que desejar, ou se for necessário pode editar algum atributo da conta. Além destas funcionalidades essenciais, o utilizador pode ativar ou desativar uma conta, sendo que isto será refletido apenas dentro da aplicação. Esta ação implica por exemplo que deixe de ser possível criar novas máquinas virtuais associadas a essa conta.

Sendo que dependendo da conta existem diferentes detalhes, foi necessário implementar uma lógica que permitisse apresentar os campos corretos para

cada conta. Para isso, e querendo uma solução simples, foi implementado um `ViewComponent` dedicado para cada operador, em que o seu objetivo é única e exclusivamente exibir os detalhes do mesmo.

Um `ViewComponent` apresenta um comportamento semelhante a uma `PartialView`, isto é, renderiza um troço de *HyperText Markup Language* (HTML) e não necessariamente uma página inteira. A vantagem deste tipo de componente é que permite um maior controlo sobre aquilo que se pretende mostrar, podendo ser aplicada alguma lógica de negócio no próprio componente. Este é tipicamente chamado diretamente de uma vista, através da *Tag Helper* [42] `Component`, seguida do método `InvokeAsync`. Esta chamada é definida dentro da vista de detalhes base da conta, sendo que a distinção dos componentes é feita tendo em conta o nome do operador da conta.

Criação

Ao criar uma nova conta é mostrado um formulário com os campos que são comuns a qualquer uma: i) Client Id; ii) Client Secret; iii) Friendly Name; iv) Cloud Provider.

Além destes são mostrados campos exclusivos de acordo com o *Cloud Provider* escolhido no formulário base. Estes variam de acordo com as necessidades e funcionalidades que o operador de *cloud* pode proporcionar. Na Figura 4.16 podemos ver um exemplo dos campos que são exibidos caso seja escolhido o operador Amazon.

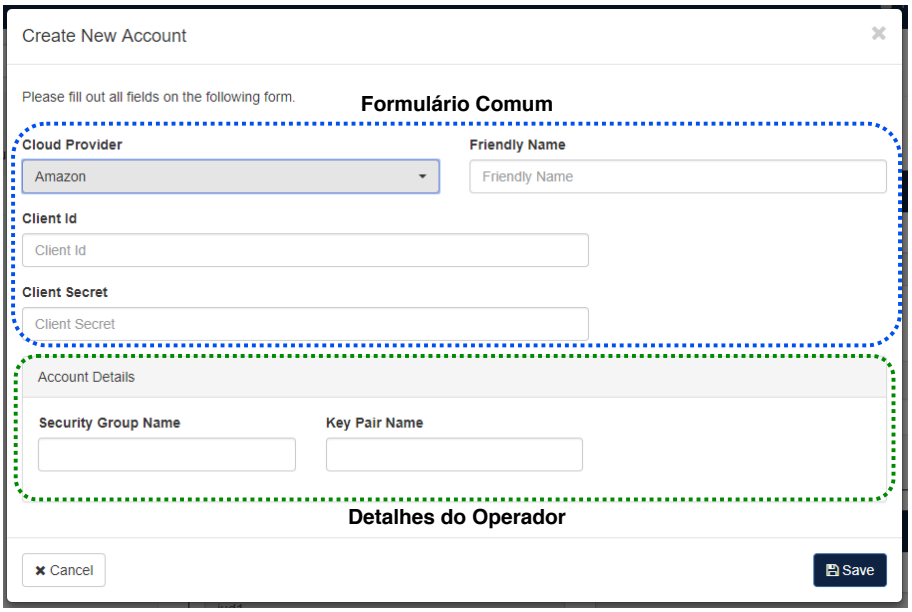

Figura 4.16: Formulário de criação de conta para o operador Amazon

Já na Figura 4.17 podemos observar os detalhes que são mostrados no formulário caso o utilizador escolha o Azure como operador.

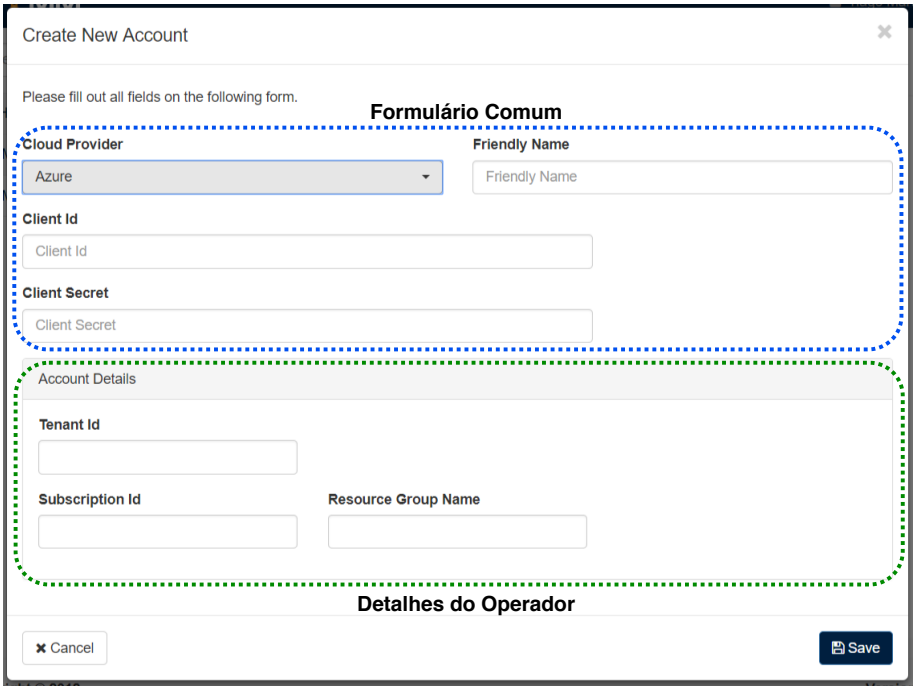

Figura 4.17: Formulário de criação de conta para o operador Azure

Como é possível verificar este formulário está divido em duas partes, uma que é igual para qualquer operador que seja escolhido (formulário comum), e outra em que, de acordo com o que é escolhido, mostra campos específicos (detalhes do operador). Assim sendo é necessário criar um componente que mostra e gere os detalhes de cada operador, ou seja, sempre que se acrescente o suporte a um novo operador deve ser adicionado este componente.

Na criação de uma conta são usados dois controladores que são essenciais: o `MyAccountsController` e o específico do operador. Após o utilizador escolher o operador é chamado o método `GetCreateAccountDetails`, do `MyAccountsController`, cujo objetivo é ser redirecionado para o método `GetCreate` do controlador desse operador. Este método renderiza o restante formulário com os campos que constituem os detalhes daquele operador de *cloud*.

Edição

Os detalhes da conta e os detalhes do operador são editados separadamente. Optou-se por esta abordagem por ser mais simples e por dar ao utilizador

uma forma mais organizada de gerir as informações das suas contas.

Por um lado os detalhes base da conta são geridos inteiramente pelo `MyAccountsController`. Por outro lado os detalhes do operador são geridos pelo seu controlador dedicado. Existem dois botões para editar respetivamente os detalhes, ao clicar nesses botões surge uma *popup* com os campos editáveis.

4.4.5 Gestão de Máquinas Virtuais

Após o utilizador configurar as suas contas de *cloud* pode então começar a usufruir da aplicação na sua íntegra. Desde consultar, criar, iniciar ou eliminar uma máquina virtual, o utilizador terá ao seu dispor um grande conjunto de funcionalidades.

Há que ter em atenção que apenas é possível a gestão de máquinas virtuais criadas ou importadas a partir da aplicação. Uma das hipóteses seria carregar todas as máquinas virtuais existentes em cada conta. No entanto, o que se pretende é que o utilizador tenha aqui um detalhe mais pormenorizado sobre aquilo que pretende gerir nesta aplicação.

Pesquisa

Após criar máquinas virtuais é de todo o interesse poder consultar as suas características e realizar algumas das ações disponíveis.

Na Figura 4.18 podemos observar o ecrã que foi desenvolvido para permitir pesquisar máquinas virtuais tendo em conta diversos filtros. Esta filtragem concede uma forma de obter uma determinada máquina virtual com determinadas características, na Tabela 4.8 podemos verificar todos esses parâmetros.

Tabela 4.8: Atributos usados para pesquisa de máquinas virtuais

Atributo	Descrição
Operador	Um dos operadores suportados pela aplicação.
Conta de *cloud*	Uma conta *cloud* que esteja configurada.
Identificador da máquina virtual	Identificador atribuído pelo operador a uma máquina virtual aquando da sua criação.
Estado	Estados possíveis de uma máquina virtual: *RUNNING*, *STOPPED* ou *DELETED*.
Sistema Operativo	Windows ou Linux.
Arquitetura	Pode optar por x64 ou x86.
Intervalo de data de criação	Escolher um intervalo de datas para obter todas as máquinas virtuais criadas nesse intervalo de tempo.

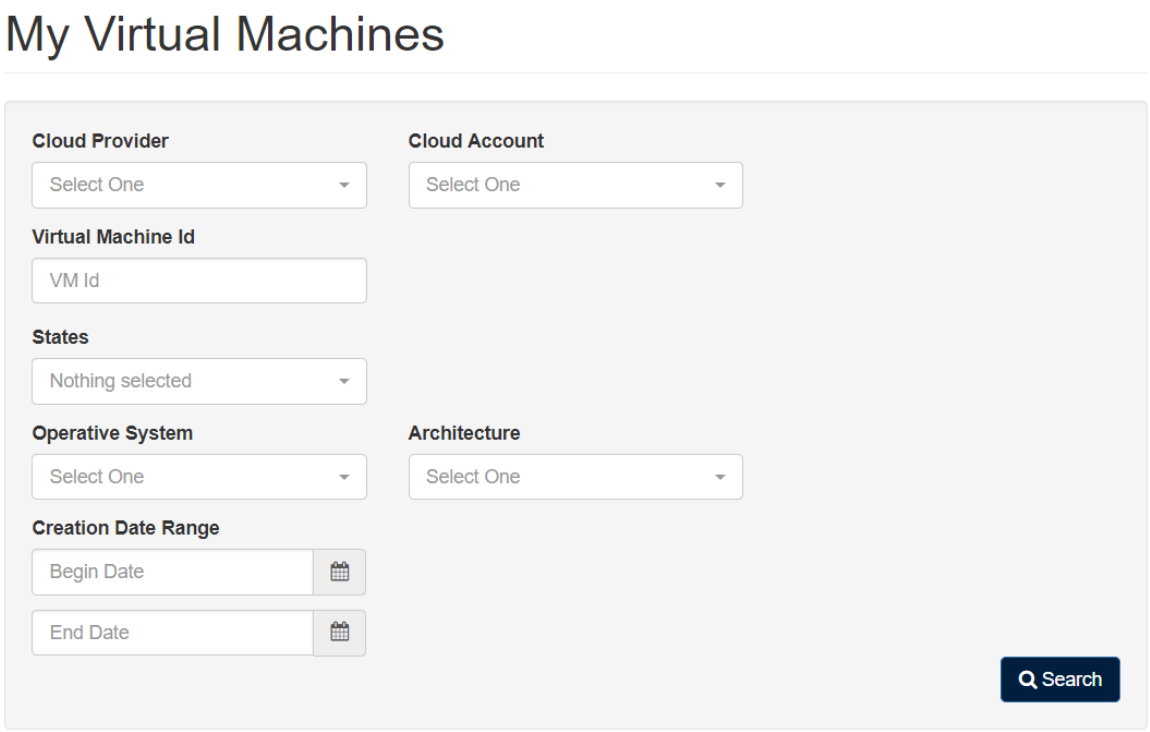

Figura 4.18: Filtros de pesquisa

O resultado obtido da pesquisa efetuada apresenta as máquinas virtuais que fazem correspondência com os filtros aplicados. Neste caso a informação que é apresentada sobre cada máquina é uniforme, independentemente do operador. Na Figura 4.19 é apresentado um exemplo da informação apresentada ao utilizador.

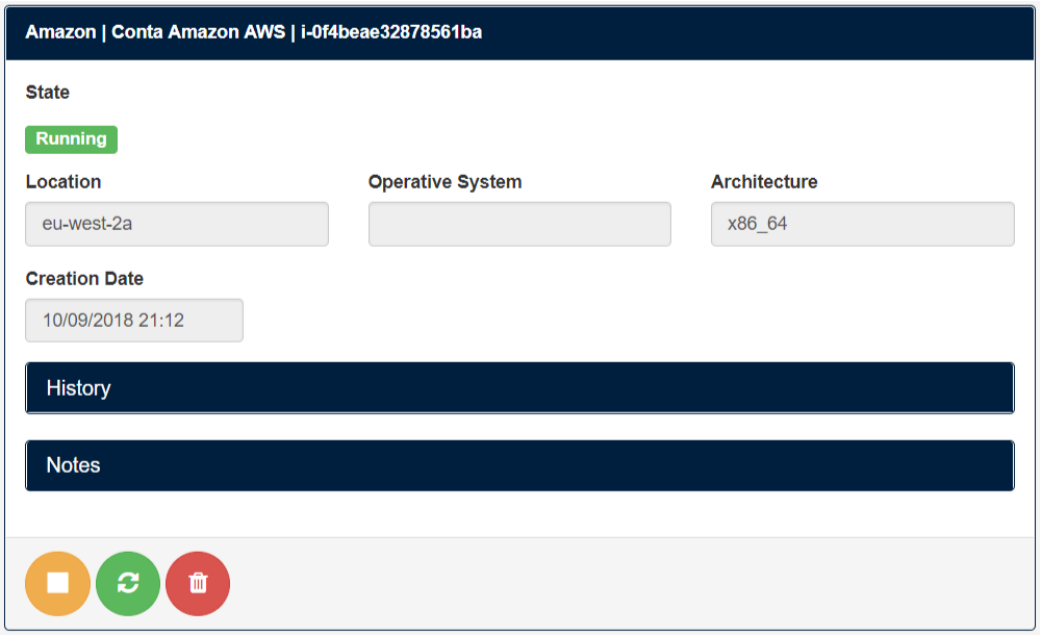

Figura 4.19: Detalhes de uma máquina virtual

De notar que o cabeçalho dos detalhes contém a seguinte informação: i) Nome do operador; ii) Nome da conta de *cloud*; iii) Nome da máquina virtual.

Tendo em conta o seu estado, algumas ações poderão não aparecer na lista de ações disponíveis, situada em formato de botões no rodapé. Por exemplo, caso a máquina esteja em execução é dispensável exibir o botão para iniciar.

Já na Figura 4.20 podemos ver essa mesma máquina virtual mas na aplicação do seu operador.

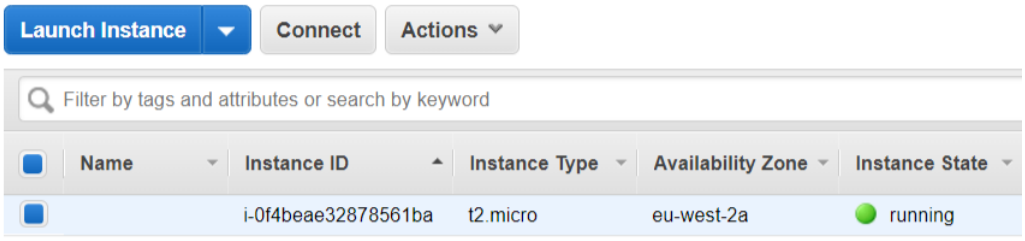

Figura 4.20: Máquina virtual na aplicação do operador Amazon

O *Instance ID* corresponde ao identificador da máquina virtual, presente no cabeçalho do componente. O *Instance State* corresponde ao estado e é mostrado com bastante destaque no início do corpo do componente. É ainda mapeada a *Availability Zone* que equivale ao campo *Location*.

Criação

Tendo em conta que apenas é possível gerir máquinas virtuais criadas ou importadas a partir da aplicação, o ecrã que possibilita a criação das mesmas é fundamental.

Nesta primeira versão apenas é requerido que o utilizador escolha a conta de *cloud* para a qual pretende criar uma nova instância de máquina virtual, Figura 4.21. Todo o processo de gerar essa nova máquina é feito internamente, com um conjunto de parametrizações definidas por omissão na aplicação. Como trabalho futuro deverá ser criado um componente, que aparece aquando da criação de uma máquina virtual, em que o utilizador pode personalizar os atributos da máquina a criar. De notar que apenas são apresentadas as contas de *cloud* que estão ativas.

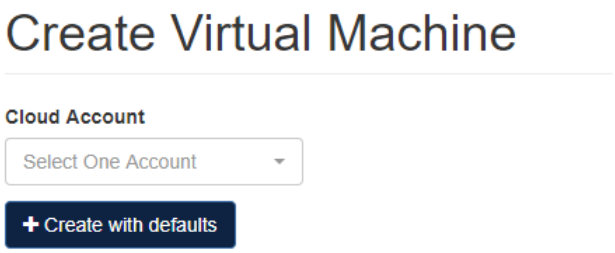

Figura 4.21: Página de criação de uma máquina virtual

Depois de ser criada com sucesso, o utilizador é reencaminhado para a página de pesquisa, em que é feita, de modo automático, a pesquisa pela nova máquina virtual, colocando o campo do identificador de máquina virtual preenchido com o valor que foi atribuído a esta no operador.

Em primeiro lugar é executado o método `Create`, que pertence à classe `ServiceCloudManager`. Em segundo lugar, caso não exista nenhum erro na execução anterior, é feita uma nova inserção na tabela `VM`. Para o efeito é executado o método `Create` da classe `VMService`, que criará essa entrada tendo como base a informação obtida na resposta do `ServiceCloudManager`.

Importação

A criação de máquinas virtuais a partir da aplicação é bastante importante, como vimos anteriormente. No entanto é natural que num momento de transição para a aplicação, já se tenham várias máquinas virtuais criadas no operador. Assim sendo é fundamental criar uma forma de importar esses recursos para o UVMM. Neste sentido foi desenvolvido o ecrã para importação

de máquinas virtuais. Na Figura 4.22 podemos observar a página correspondente a esta funcionalidade.

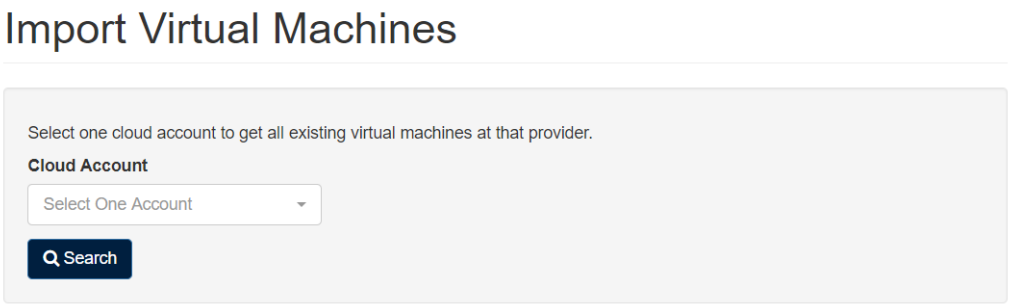

Figura 4.22: Pesquisa para importação

O modo de funcionamento é muito simples, basta escolher a conta de *cloud* que se pretende e serão mostradas todas as máquinas virtuais que existem no operador mas que ainda não estão mapeadas na solução.

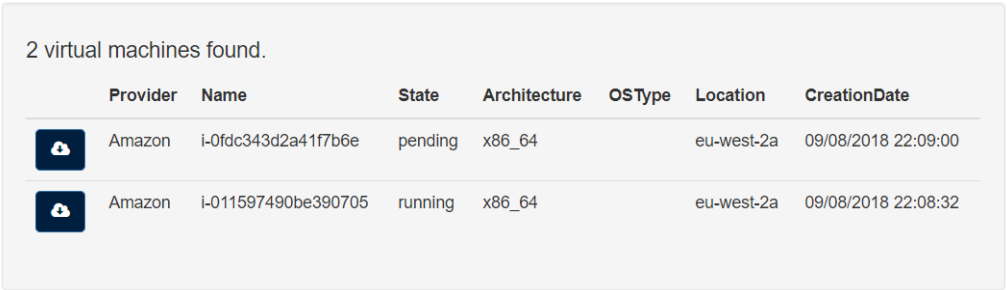

Figura 4.23: Resultados da pesquisa

Nos resultados obtidos, Figura 4.23, são exibidas as informações que são conhecidas pela solução, tais como:

- Operador
- Nome/Identificador
- Estado
- Arquitetura
- Sistema Operativo
- Localização

- Data de Criação

Após analisar e decidir que máquinas quer importar, basta o utilizador clicar no botão existente na tabela e confirmar a sua intenção. Após a execução desta ação, e caso tudo tenha corrido sem qualquer tipo de erro, é mostrada uma mensagem de sucesso. A Figura 4.24 mostra um exemplo da informação que é mostrada após importar uma máquina virtual.

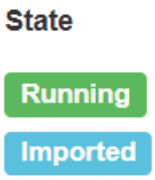

Figura 4.24: Etiqueta de importação

Para dar suporte a esta funcionalidade foi definido um método que permite obter todas as máquinas virtuais de uma determinada conta de *cloud*, `GetAllByAccount` na classe `ServiceCloudManager`. Este método retorna uma lista de todas as máquinas virtuais existentes no operador, para a conta escolhida, sendo feita posteriormente a verificação de quais já existem na aplicação, usando para isso uma consulta à base de dados, validando a sua existência na tabela `VM`.

4.4.6 Páginas de Erro

É importante entender o porquê das páginas de erro personalizadas serem importantes. A página de erro 404, que é renderizada quando se tenta aceder a um recurso que já não está disponível no endereço do pedido, deve ser capaz de manter o utilizador com interesse em continuar a navegar na aplicação. Muitas vezes os *sites* deixam de lado a personalização deste tipo de páginas de erro e, em alguns casos, nem sequer existem. Há que ter em atenção que o utilizador, ao navegar na aplicação, vai julgar a qualidade do conteúdo com base naquilo que vê e como vê.

Na Figura 4.25 podemos verificar a forma como foram envolvidas as páginas de erro no contexto da aplicação.

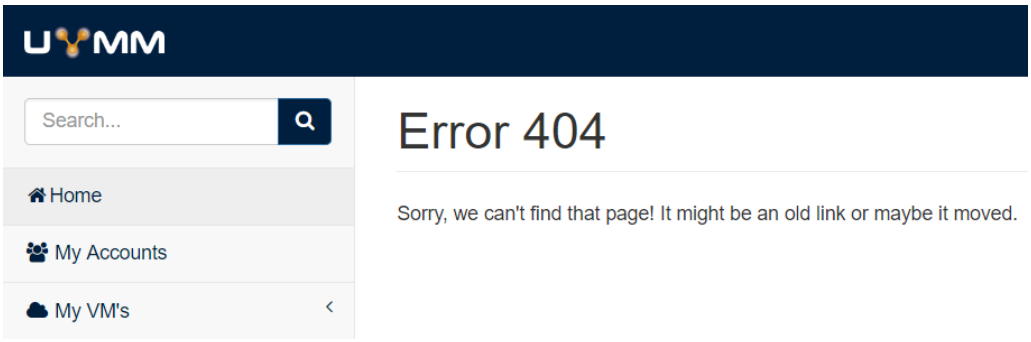

Figura 4.25: Erro 404

Além de se manter o menu lateral para permitir ao utilizador continuar a navegar, é mostrada uma mensagem personalizada para alguns tipos de erro. Na próxima Tabela, 4.9, são apresentados os códigos de erro e as respetivas mensagens que aparecem ao utilizador.

Tabela 4.9: Erros e mensagens apresentadas

Erro	Mensagem
403	Sorry, but you can't access this page.
404	Sorry, we can't find that page! It might be an old link or maybe it moved
500	Sorry! An internal error occurred
Access Denied	You don't have permission to access this page.

O erro *Access Denied* ocorre quando o utilizador tenta aceder a um *controller* que necessita de um *role*, ou conjunto de *roles*, para interagir com ele. O .NET Core possui um mecanismo de autorização baseada em perfis (*Role-based Authorization*) [43]. Com a anotação `Authorize` consegue-se especificar que perfis são necessários para um utilizador ter permissão para aceder a uma determinada classe ou método. Por omissão, no caso do acesso ser recusado, o utilizador é reencaminhado para uma página com o nome *AccessDenied*.

4.4.7 Componentes Configuráveis

Em toda e qualquer aplicação é relevante ter funcionalidades que acrescentem valor ao seu objetivo. No entanto por vezes é complexo criar sistemas que se possam adaptar às necessidades de cada cliente, quando se pretende que vários clientes coabitem na mesma solução. Pensando neste aspeto, foi criado o conceito de componente. Um componente é uma funcionalidade que pode ser configurada através da base de dados para ser apresentada para todos os operadores, ou no limite para apenas um. Isto leva-nos ao ponto chave deste conceito, que consiste na capacidade de criar componentes que podem ser específicos e dedicados a um determinado operador. Um operador tem um conjunto de componentes, e um componente pode pertencer a vários operadores.

4.4.7.1 Histórico

Este componente é essencial para ter uma noção cronológica das ações que foram aplicadas numa máquina virtual. Na Figura 4.26 podemos observar a forma como a informação é apresentada no componente.

Figura 4.26: Componente de Histórico

4.4.7.2 Notas

Este componente tem como objetivo mostrar as notas que são criadas automaticamente pela aplicação. Confere também a possibilidade de criar notas de forma espontânea, podendo ser guardado qualquer tipo de informação adicional que se pretenda. A Figura 4.27 mostra o aspeto do componente.

Figura 4.27: Componente de Notas

4.5 Sumário

O modelo de dados, apresentado na secção 4.1, serviu para apresentar a estrutura de dados necessária para o funcionamento de todas as funcionalidades da solução. Para gerir e obter esses dados foi implementada uma camada de acesso a dados, apresentada na secção 4.2, que permite interagir com a base de dados sem recurso à linguagem SQL. Isto é possível devido à utilização de um ORM, que mapeia as entidades da base de dados em objetos. Esta camada é utilizada pela camada de negócio, apresentada na secção 4.3, que aplica a lógica de negócio, realiza a comunicação com as APIs dos operadores de *cloud* e realiza a validação dos dados recebidos. Estes dados são enviados através da camada de apresentação, apresentada na secção 4.4, cuja função é mostrar as informações ao utilizador, permitindo que este interaja com a aplicação. Nesta camada é realizada a lógica de autenticação e autorização. Um utilizador tem um conjunto de permissões que lhe permitem aceder a determinados conteúdos na aplicação. Caso tente forçar o acesso a alguma página a que não está autorizado, será reencaminhado para uma página de erro. Nesta camada foram também apresentados os ecrãs para realizar a gestão de contas de *cloud* e de máquinas virtuais.

Capítulo 5

REST API

Este capítulo tem como objetivo a apresentação da REST API que foi desenvolvida para disponibilizar a solução a outros desenvolvedores que queiram implementar as suas próprias aplicações, tendo por base as funcionalidades descridas na API.

5.1 Motivação

A quantidade de dispositivos que surgem ano após ano e que nos permitem interagir com todo o tipo de aplicações, tem aumentado significativamente. A criação de sistemas que possuem apenas uma interface gráfica deixa de fazer sentido em alguns casos.

Deste modo as REST API permitem desenvolver funcionalidades de forma desacoplada das interfaces de utilizador, o que permite criar a lógica de negócio para vários dispositivos e sem estar dependente de uma determinada linguagem de programação.

Além disso, a quantidade de pessoas que possuem acesso à Internet é cada vez maior, e por isso é necessário criar sistemas que sejam cada vez mais escaláveis e acessíveis.

5.2 Porquê REST?

O *Representational State Transfer* (REST) usa o protocolo *Hypertext Transfer Protocol* (HTTP) como meio de comunicação na troca de mensagens entre o cliente e o servidor. Um cliente envia uma mensagem sob a forma de um pedido HTTP e o servidor retribui com uma resposta HTTP.

Outra opção seria a utilização do *Simple Object Access Protocol* (SOAP). Este é também um protocolo de troca de mensagens mas em formato *Exten-*

sible Markup Language (XML), para ser usado em ambientes distribuídos. O mais comum é descrever a interface do serviço SOAP com *Web Services Description Language* (WSDL). Basicamente, cada serviço terá um ficheiro WSDL que terá a definição de todas as funções bem como o tipo de dados que são usados, nos pedidos e respostas, e *endpoints*.

Foi escolhido o REST pois uma das suas maiores vantagens é sua flexibilidade e simplicidade. Não são impostas restrições relativamente ao formato das mensagens, o programador pode optar pelo formato que considerar mais adequado para as mensagens que são trocadas, de acordo com as suas necessidades. Os formatos mais comuns são JSON, XML e texto puro, mas em teoria qualquer formato pode ser usado.

É importante explicar sucintamente os métodos que são utilizados com mais frequência, para isso podemos observar a Tabela 5.1.

Tabela 5.1: Métodos mais comuns em pedidos REST

Método	Descrição
`POST`	Utilizado maioritariamente para criar novos recursos.
`GET`	Utilizado para ler/obter a representação de um recurso.
`PUT`	Utilizado maioritariamente para alterar a representação de um recurso.
`DELETE`	Utilizado para remover um recurso.

Os métodos apresentados na Tabela 5.1 correspondem às operações CRUD.

5.3 Descrição de Funcionalidades

A descrição da API está disponível no seguinte link:
`https://uvmmwebapi20180724110907.azurewebsites.net/swagger/index.html`

As funcionalidades disponíveis são todas aquelas que estão descritas na interface `IRestServiceCloud`, cuja implementação é feita na classe, apresentada na secção 4.3.2.1, `ServiceCloudManager`. A única diferença para a interface `IVirtualMachineService` são os parâmetros de entrada dos métodos, sendo que nesta última são usados objetos compostos.

Na Tabela 5.2 estão descritas todas as funcionalidades disponibilizadas na API. O endereço base de todos os o pedidos é: `/api/VirtualMachine/` que se adiciona ao endereço do *host* onde estiver instalada a REST API. No presente caso o host é: `uvmmwebapi20180724110907.azurewebsites.net`. Assim sendo um exemplo de um pedido é:

```
POST https://uvmmwebapi20180724110907.azurewebsites.net/
api/virtualmachine/all
```

Tabela 5.2: Métodos disponíveis na API

Método	Pedido	Descrição
`POST`	`/all`	Obter todas as máquinas virtuais.
`POST`	`/{virtualMachineId}`	Obter uma máquina virtual.
`PUT`	`/create/{virtualMachineId}`	Criar uma máquina virtual.
`POST`	`/start/{virtualMachineId}`	Iniciar uma máquina virtual.
`POST`	`/stop/{virtualMachineId}`	Parar uma máquina virtual.
`POST`	`/restart/{virtualMachineId}`	Reiniciar uma máquina virtual.
`POST`	`/refresh`	Atualizar uma máquina virtual.
`DELETE`	`/delete/{virtualMachineId}`	Remover uma máquina virtual.

De notar que todos os pedidos anteriormente apresentados são relativos a uma conta de *cloud* que está definida no corpo da mensagem.

5.4 Validação de Pedidos

Para garantir que todos os parâmetros essenciais são enviados nos pedidos à API, foi usada a biblioteca `DataAnnotations` que permite acrescentar anotações para validar os modelos. A anotação mais comum é a `Required` que obriga ao preenchimento do atributo. No entanto, no caso da solução, o modelo `UserCloudAccount` é bastante complexo, pois incorpora os modelos relativos aos detalhes de cada operador. O problema surge devido ao facto de que o modelo dos detalhes de uma conta, por exemplo da Amazon, apenas deve ser obrigatório quando se tratar de um pedido a uma máquina virtual desse operador. Por esse motivo foi necessário implementar uma anotação que satisfaça esta especificidade. Foi desenvolvida a anotação `RequiredIf`, que recebe como parâmetro o nome do operador. A implementação do método `IsValid` é o que permite efetivamente personalizar a validação do mo-

delo. O nome do operador que é recebido no construtor é comparado com o valor do atributo `CloudProvider` do modelo `UserCloudAccount`. Neste sentido apenas é validado o modelo de detalhes que corresponde ao operador referente ao pedido efetuado.

A autenticação dos pedidos nos operadores é feita de modo específico tendo em conta as regras de cada um. Com a implementação da interface `IAuthenticationService`, é garantido que cada um realiza a sua lógica de autenticação necessária para realizar pedidos à sua API, tal como foi apresentado na seccção 4.3.2.1 (`ServiceCloudManager`).

Capítulo 6

Conclusões e Trabalho Futuro

O UVMM apresenta uma contribuição para resolver o problema da diferenciação que existe entre os operadores de *cloud*, sendo que cada um tem as suas funcionalidades e forma de gerir as máquinas virtuais.

A solução implementada sob a forma de aplicação Web permite uma maior disponibilidade e usabilidade, pois é um tipo de interface muito comum, bastando uma ligação à Internet para poder interagir com a mesma. Devido ao facto de ter uma interface de utilizador com um *layout* responsivo, garante-se uma maior compatibilidade com qualquer tipo de dispositivo, estando deste modo preparada para ser utilizada num *smartphone* ou *tablet*. Este tipo de suporte é cada vez mais valorizado, uma vez que nos permite realizar tarefas complexas com um dispositivo de pequenas dimensões, que trazemos diariamente no nosso bolso. Um exemplo prático, aplicado à presente solução, é o facto de ser possível iniciar ou parar uma máquina virtual usando para isso um dispositivo móvel, como o nosso *smartphone* pessoal.

Entende-se que nem sempre uniformizar e centralizar sistemas é uma tarefa fácil. Em virtude do que foi mencionado relativamente aos desafios conhecidos e à incerteza de alguns obstáculos, esta solução teve uma fase inicial mais lenta. Devido à necessidade de estudar e ler as APIs dos operadores e perceber as configurações necessárias para ser possível aceder às mesmas, o momento inicial de desenvolvimento foi mais demoroso. Apesar disso os operadores escolhidos para serem suportados possuem boa documentação e muitos exemplos de como usar as suas APIs.

A escolha da linguagem de programação C#, teve em conta a vasta e extensa *framework* ASP.NET Core MVC, que possui uma ótima implementação do padrão MVC para aplicações Web. Tendo em conta os aspetos observados relativamente à evolução do .NET Core, estes foram um estímulo para reforçar a vontade de optar por esta linguagem. O ponto referido nos desafios iniciais, na secção 1.3, relativamente à possibilidade da não existên-

cia de algumas bibliotecas, não se revelou um obstáculo, não tendo ocorrido nenhum tipo de impedimento neste aspeto.

Não obstante, a arquitetura implementada não obriga ao uso desta linguagem, qualquer outra poderia ter sido usada para desenvolver toda a lógica existente. Do mesmo modo que outro sistema de gestão de base de dados poderia ter sido usado, a escolha do PostgreSQL teve em conta detalhes como o custo associado para a disponibilidade da solução num contexto real, em que é necessária a existência de uma instância de base de dados.

Foi imprescindível que, levando-se em consideração que se pretende a fácil integração de novos operadores, seja garantido o desenho de uma aplicação que seja capaz de garantir esse suporte, tendo como base a modularidade. Para cada operador é necessário criar alguns componentes específicos em cada camada. Numa solução que prima pela centralização, esta necessidade de partes específicas pode surgir com alguma estranheza, no entanto há sempre um momento em que tem de se criar uma exclusividade para garantir que se cumpre todos os requisitos de um operador.

Face à crescente adoção da *cloud computing*, acredita-se que esta aplicação faça ainda mais sentido, visto que a tendência é o aumento de utilizadores de operadores de *cloud*. Isto enfatiza a vantagem que o UVMM possui perante este panorama. Melhorar a eficiência na gestão de máquinas virtuais criadas em diversos operadores, tornar-se-á uma realidade com a utilização desta solução.

O trabalho futuro é vasto pois há várias formas de aumentar as funcionalidades da solução. Uma delas é a adição de novas operações disponibilizadas pela REST API, aumentando a diversidade de ações sobre as máquinas virtuais. Um exemplo é a possibilidade de instanciar máquinas escolhendo uma imagem predefinida. A melhoria da personalização no momento da criação de uma máquina virtual é uma das tarefas mais relevantes, na medida em que permitirá ao utilizador uma maior liberdade relativamente às características da máquina a criar. Outra possibilidade de fazer crescer a solução, e uma das mais relevantes, é o suporte a outros operadores.

Com uma maior diversidade de operadores disponíveis, a REST API e a aplicação Web serão certamente mais atrativas a uma maior quantidade de utilizadores. Relativamente à aplicação Web será necessário implementar novos ecrãs para as novas funcionalidades sobre as máquinas virtuais. O suporte a novos operadores é garantido desenvolvendo pequenos componentes dedicados nas três camadas: acesso a dados, negócio e apresentação.

A REST API surgiu num momento em que o projeto já estava com bastante desenvolvimento realizado, por esse motivo a abordagem inicial não incluiu esta componente. No entanto, chegou-se à conclusão que seria uma forma de tornar esta solução aberta para outros programadores poderem

utiliza-la ou expandir o suporte a outros operadores. Apesar de não ter feito parte do planeamento de tarefas para este projeto, assim que surgiu esta possibilidade foi-lhe atribuída alta prioridade por todos os motivos que foram enunciados anteriormente.

Bibliografia

[1] Emerson Alecrim. O que é cloud computing? https://www.infowester.com/cloudcomputing.php.

[2] Javier Barabas. Iaas paas saas - modelos de serviço de cloud. https://www.ibm.com/cloud/learn/iaas-paas-saas.

[3] Amazon EC2. https://aws.amazon.com/pt/ec2/.

[4] Microsoft Azure. https://azure.microsoft.com/pt-pt/.

[5] Google Cloud Platform. https://cloud.google.com/.

[6] Luna Cloud. https://www.lunacloud.com/pt/.

[7] NuGet. https://www.nuget.org/.

[8] Renato Groffe. Asp.net core: visão geral, exemplos práticos e novidades. https://goo.gl/xxUvZG.

[9] TheStreet. How microsoft and google are gunning for amazon's cloud dominance. https://www.thestreet.com/story/14109584/1/amazon-remains-the-biggest-cloud-in-the-sky.html.

[10] Synergy. https://www.srgresearch.com/.

[11] IBM Cloud. https://www.ibm.com/cloud/.

[12] Oracle Cloud. https://cloud.oracle.com/home.

[13] Alibaba Cloud. https://www.alibabacloud.com/.

[14] OpenNebula. https://opennebula.org/about/technology/.

[15] OpenNebula. About the opennebula technology. https://opennebula.org/about/technology/.

[16] KVM. https://www.linux-kvm.org/page/Main_Page.

[17] XEN Project. https://www.xenproject.org/about-us.html.

[18] VMware ESX. https://www.vmware.com/products/esxi-and-esx.html.

[19] OpenStack. https://www.openstack.org/software/.

[20] Hyper-V. https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/mt169373(v=ws.11).

[21] QEMU. https://www.qemu.org/.

[22] OpenStack. What is what is openstack? https://www.openstack.org/software/.

[23] RightScale. https://www.rightscale.com/.

[24] RightScale. Two solutions from rightscale. https://www.rightscale.com/products-and-services/products.

[25] Carlos Gonçalves. *Parallel and Distributed Statistical-based Extraction of Relevant Multiwords from Large Corpora.* PhD thesis, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2017.

[26] Boris Savić. Access to different cloud platforms through uniform interface. Master's thesis, Universidade de Liubliana, 2013.

[27] VMware vCloud Suite. https://www.vmware.com/content/dam/digitalmarketing/vmware/pt/pdf/VMware-vCloud-Suite-Datasheet.pdf.

[28] Libcloud. https://libcloud.apache.org/.

[29] Apache jclouds. https://jclouds.apache.org/.

[30] PostgreSQL. https://www.postgresql.org/.

[31] Aaron Walker. Best free and open-source database software. https://blog.g2crowd.com/blog/database/best-free-and-open-source-database-software/.

[32] Npgsql. http://www.npgsql.org/.

[33] .NET ORM Benchmark. https://github.com/DevExpress/XpoNetCoreDemos/tree/master/ORMBenchmark.

[34] EntityFramework Tutorial. What is code-first? http://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx.

[35] Learn Entity Framework Core. Entity framework core fluent api. https://www.learnentityframeworkcore.com/configuration/fluent-api.

[36] Scott Addie e Luke Latham Steve Smith. Repository pattern with asp.net core. https://docs.microsoft.com/en-us/aspnet/core/fundamentals/repository-pattern?view=aspnetcore-2.1.

[37] Martin Fowler. Unit of work. https://martinfowler.com/eaaCatalog/unitOfWork.html.

[38] Azure REST API. https://docs.microsoft.com/en-us/rest/api/azure/.

[39] Amazon EC2 API. https://docs.aws.amazon.com/AWSEC2/latest/APIReference/Welcome.html.

[40] Amazon EC2 SDK. https://docs.aws.amazon.com/sdkfornet/v3/apidocs/index.html?page=EC2/NEC2.html.

[41] Limits for sending and getting mail. https://support.google.com/mail/answer/22839?hl=en.

[42] Rick Anderson. Tag helpers in asp.net core. https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro?view=aspnetcore-2.1.

[43] Role based authorization in ASP.NET Core. https://docs.microsoft.com/en-us/aspnet/core/security/authorization/roles?view=aspnetcore-2.1.

Anexos

A Mecanismo de *Deployment*

A.1 Base de Dados

O ElephantSQL oferece o PostgreSQL como um serviço. Esta plataforma assume a responsabilidade das tarefas de administração do PostgreSQL, como a instalação, a atualização da última versão estável e a gestão de *backup's*. Cabe ao programador gerir apenas as tabelas e o seu conteúdo, focando-se assim no desenvolvimento. Esta plataforma possuí vários planos para a base de dados, sendo que o seu custo vai aumentando de acordo com a capacidade de hardware que se pretenda. Para o caso da presente solução, foi suficiente o plano denominado por *Tiny Turtle* que coloca a base de dados num servidor de alto desempenho partilhado e com um limite de 20 MB de dados.

A.2 Aplicação e REST API

O projeto foi desenvolvido inteiramente no Visual Studio 2017. Este possuí uma ferramenta de publicação de projetos, que permite instalar uma aplicação de modo rápido e simples no Serviço de Aplicações do Azure ou numa máquina virtual. Para isso basta ter a conta Microsoft configurada no Visual Studio e ter essa mesma conta associada a uma conta no Azure. Com este conjunto de configurações é possível publicar a aplicação com apenas alguns cliques.

B Modelo Entidade-Associação

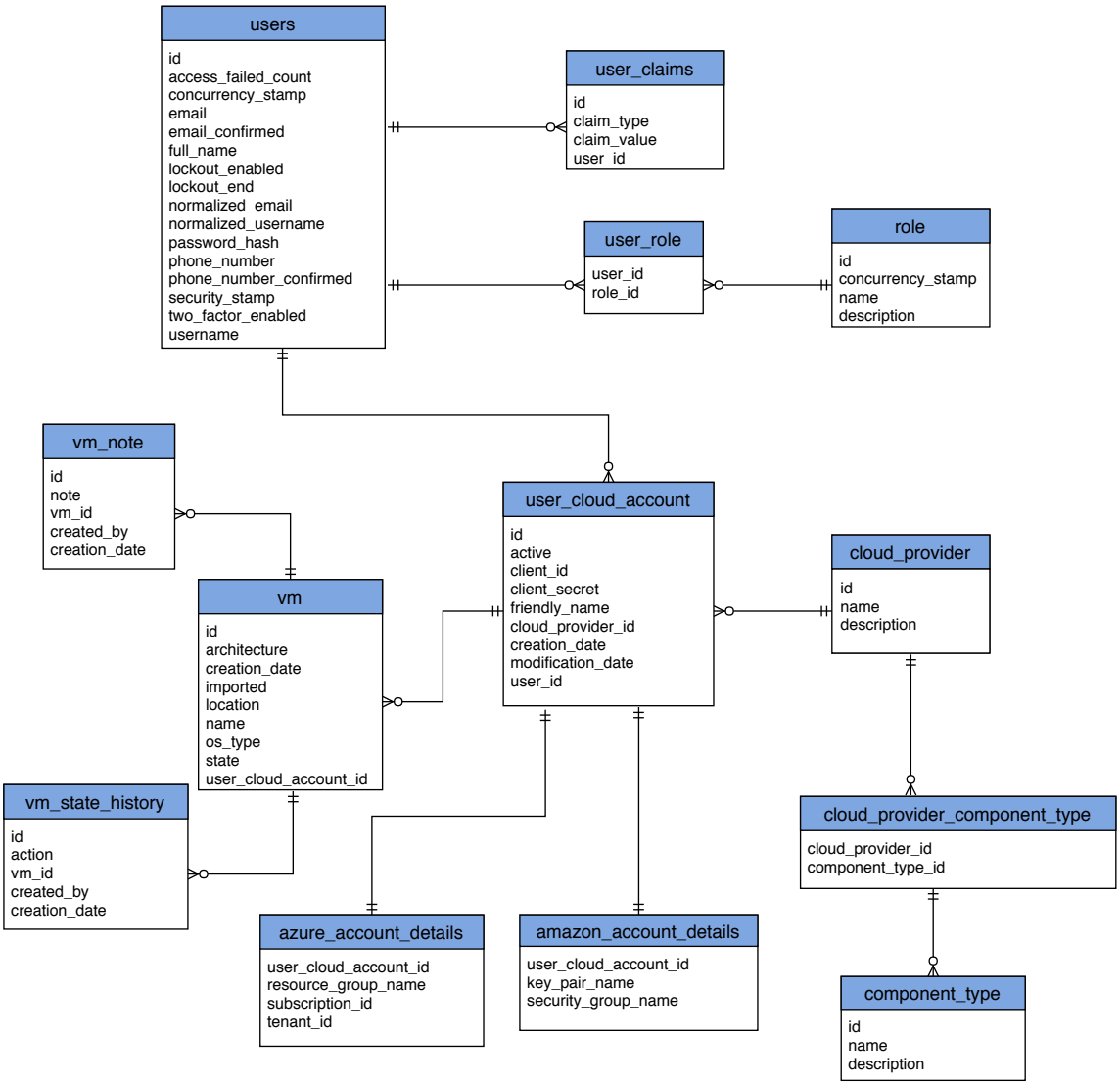

Figura B.1: Modelo Entidade-Associação

C Diagramas *Unified Modeling Language* de classes

C.1 `UVMM.Repository`

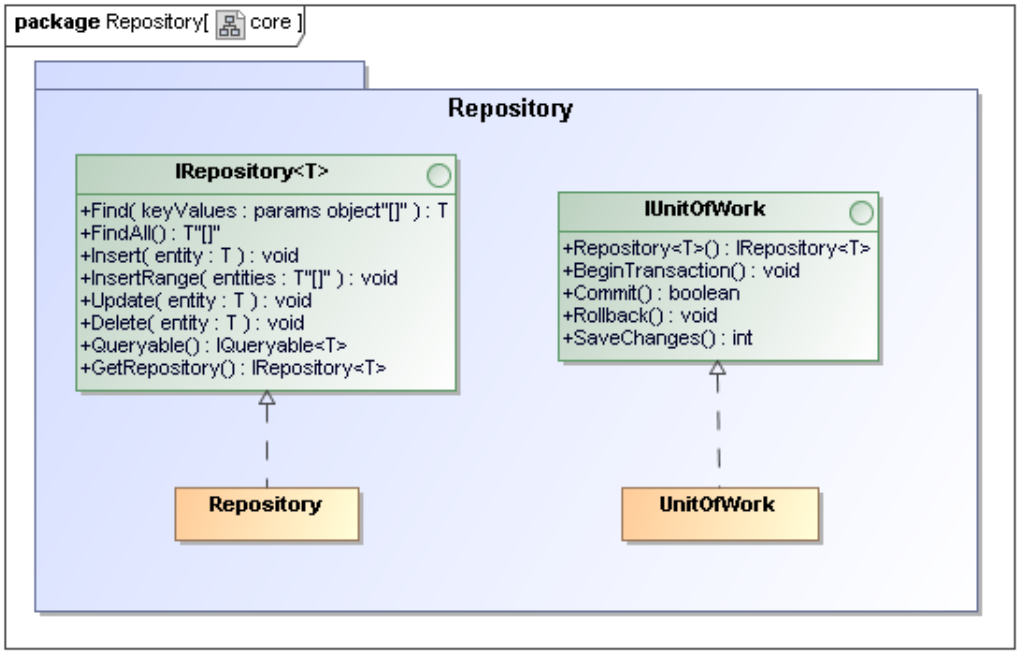

Figura C.2: Diagrama UML do *core* da biblioteca `UVMM.Repository`

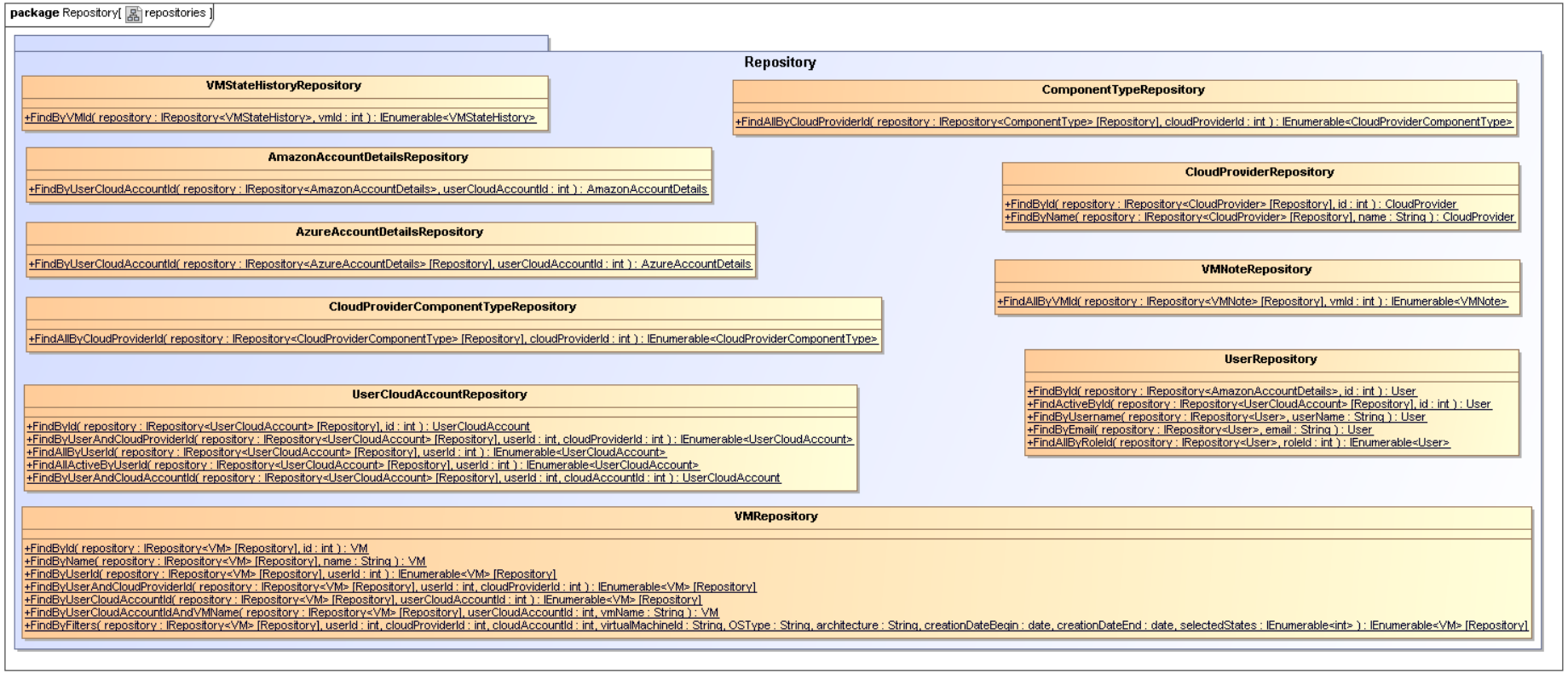

Figura C.3: Diagrama UML da biblioteca `UVMM.Repository`

C.2 UVMM.Service

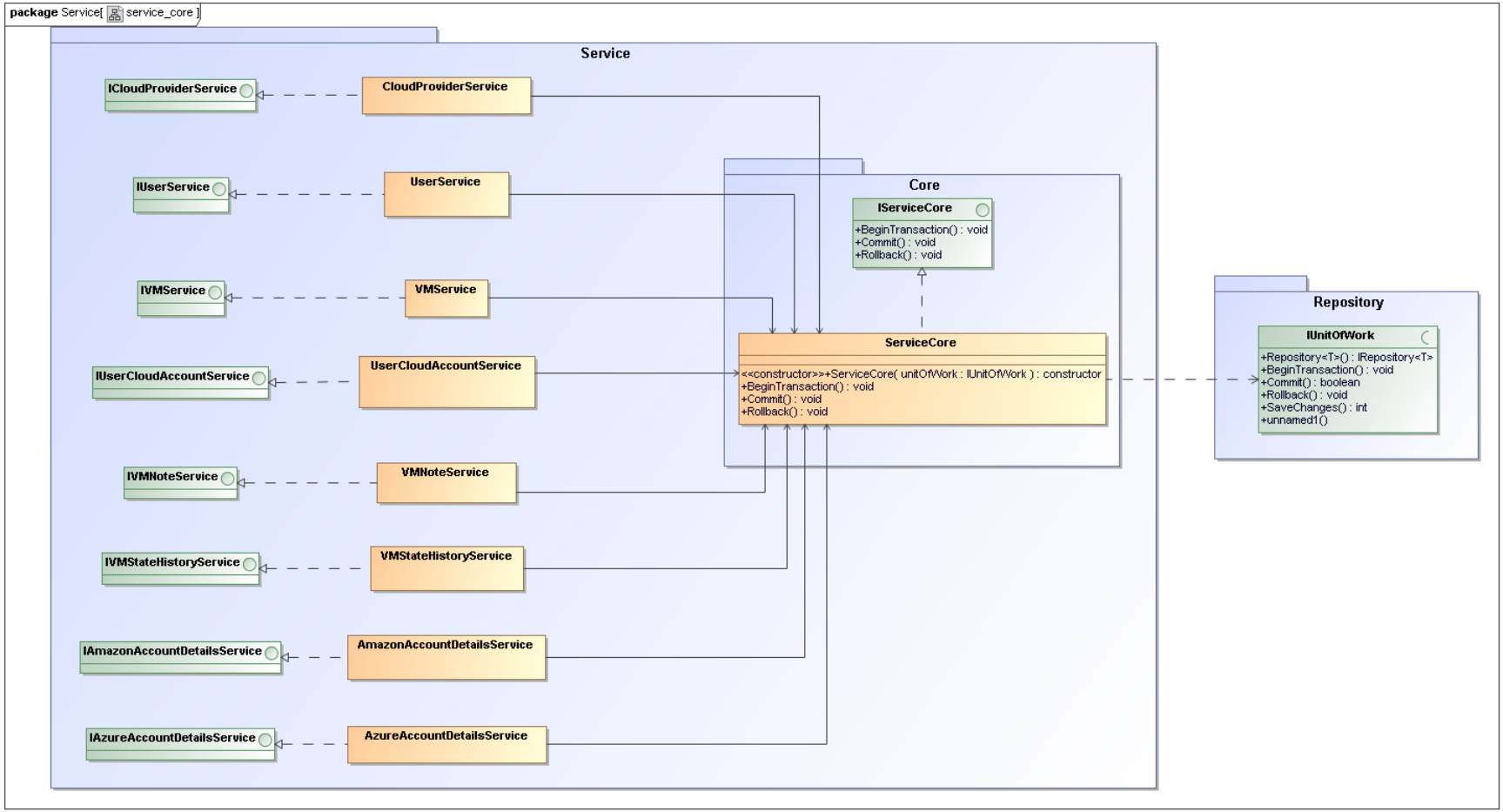

Figura C.4: Diagrama UML simplificado da biblioteca `UVMM.Service`

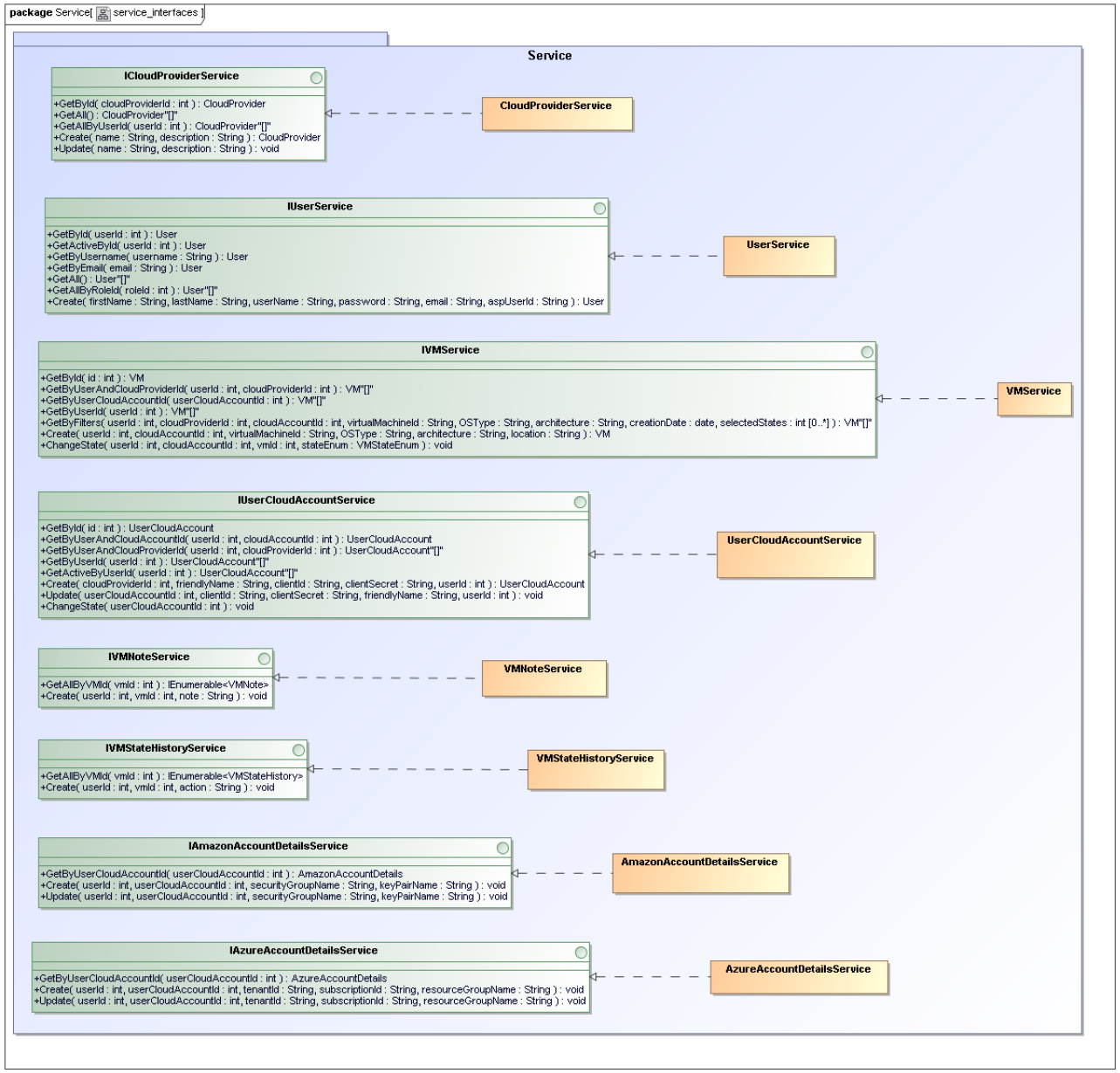

Figura C.5: Diagrama UML da biblioteca `UVMM.Service`

C.3 UVMM.ServiceCloud

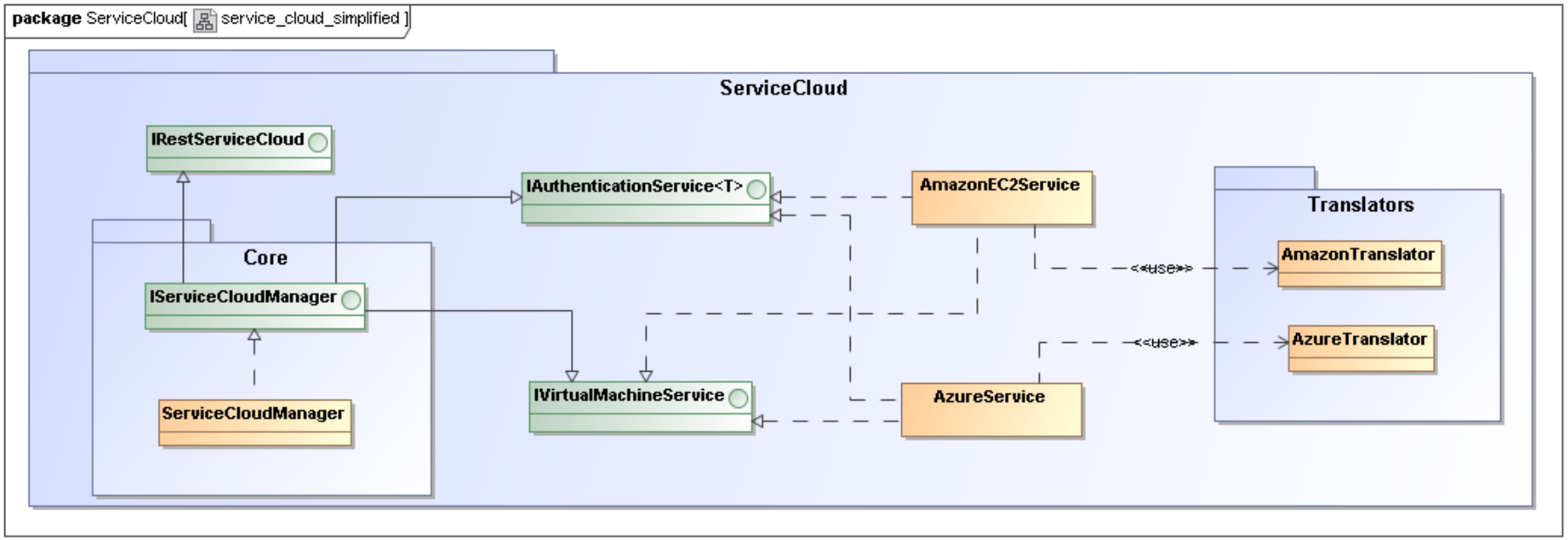

Figura C.6: Diagrama UML simplificado da biblioteca `UVMM.ServiceCloud`

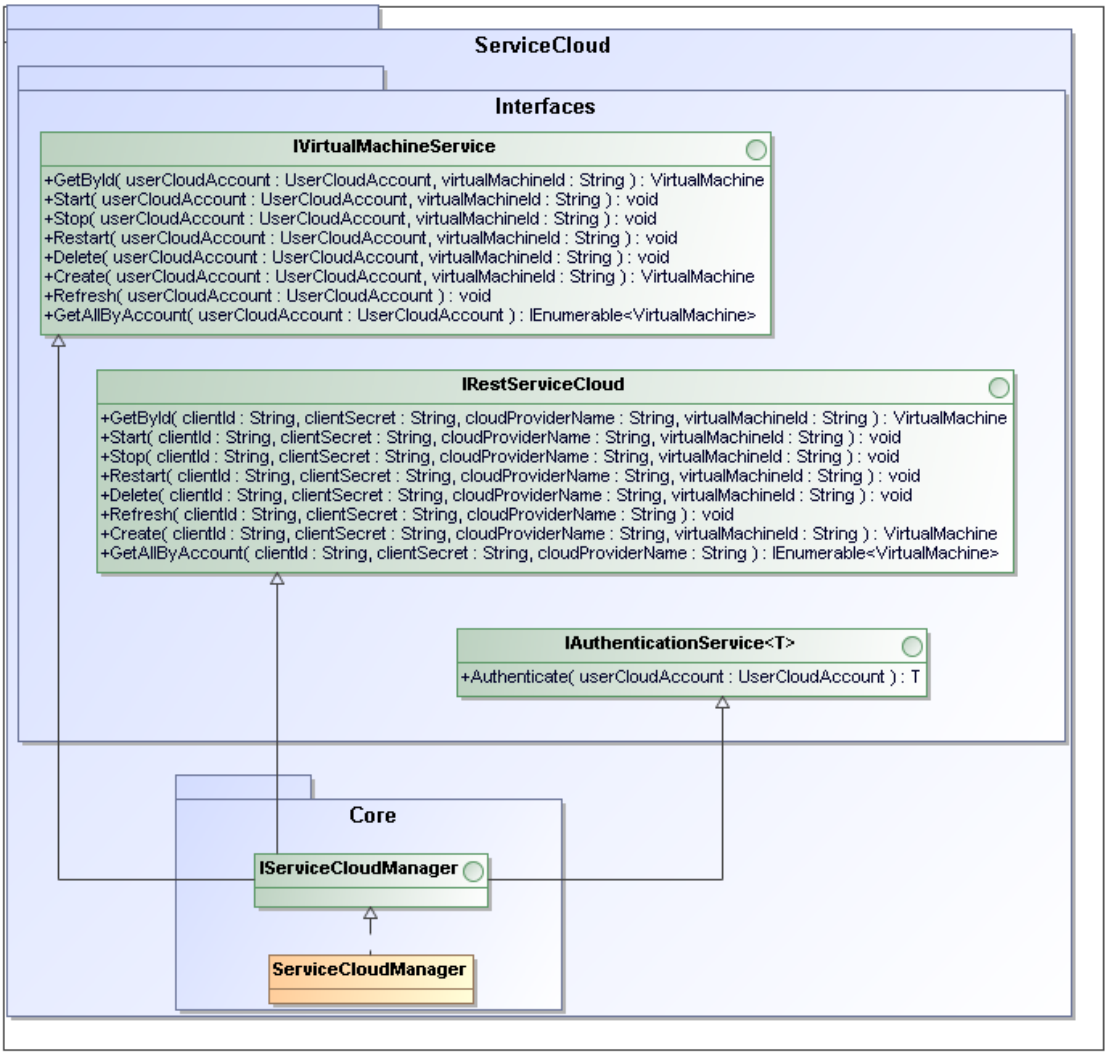

Figura C.7: Diagrama UML das interfaces da biblioteca `UVMM.ServiceCloud`

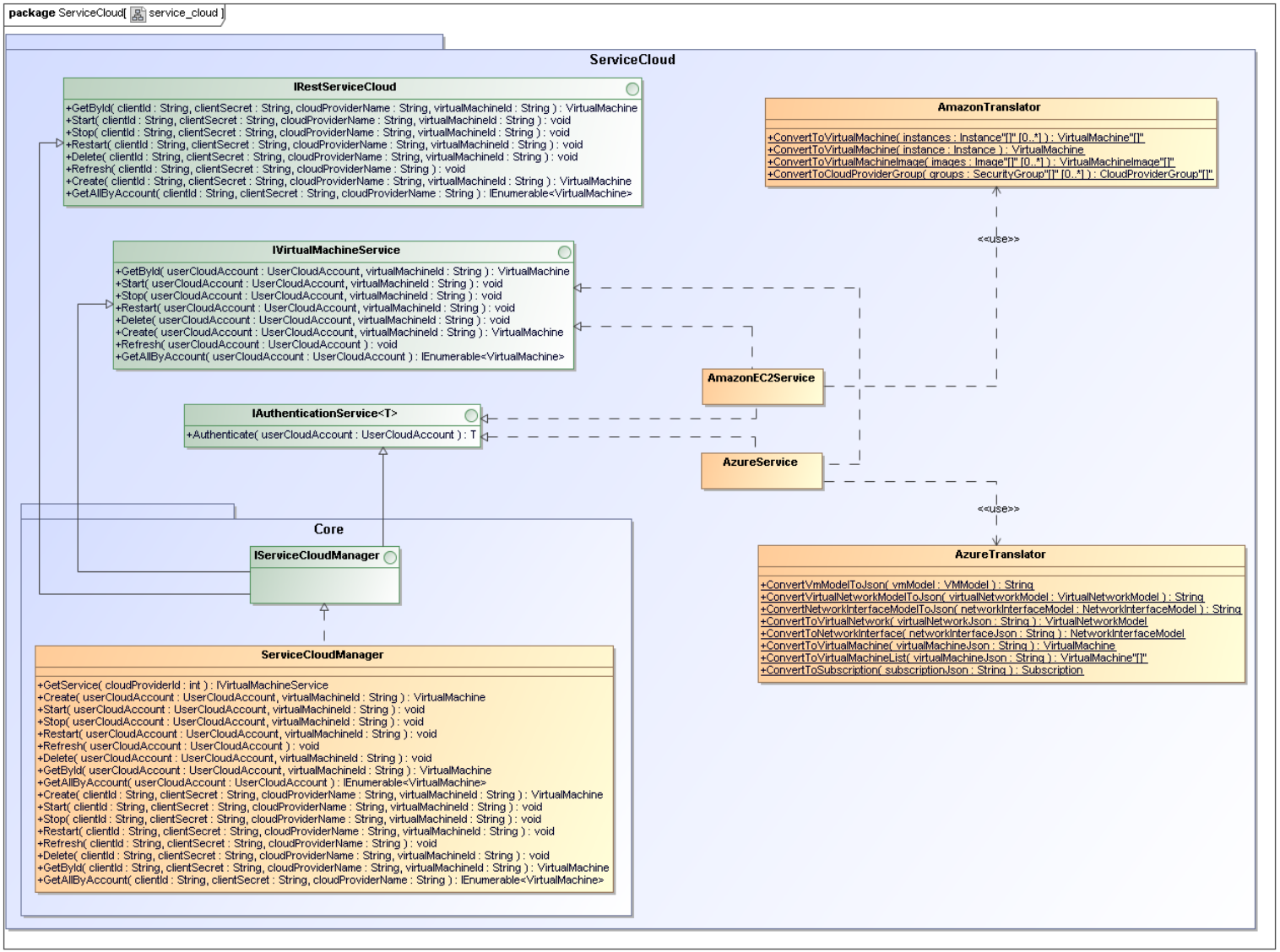

Figura C.8: Diagrama UML da biblioteca `UVMM.ServiceCloud`

C.4 UVMM.ServiceMail

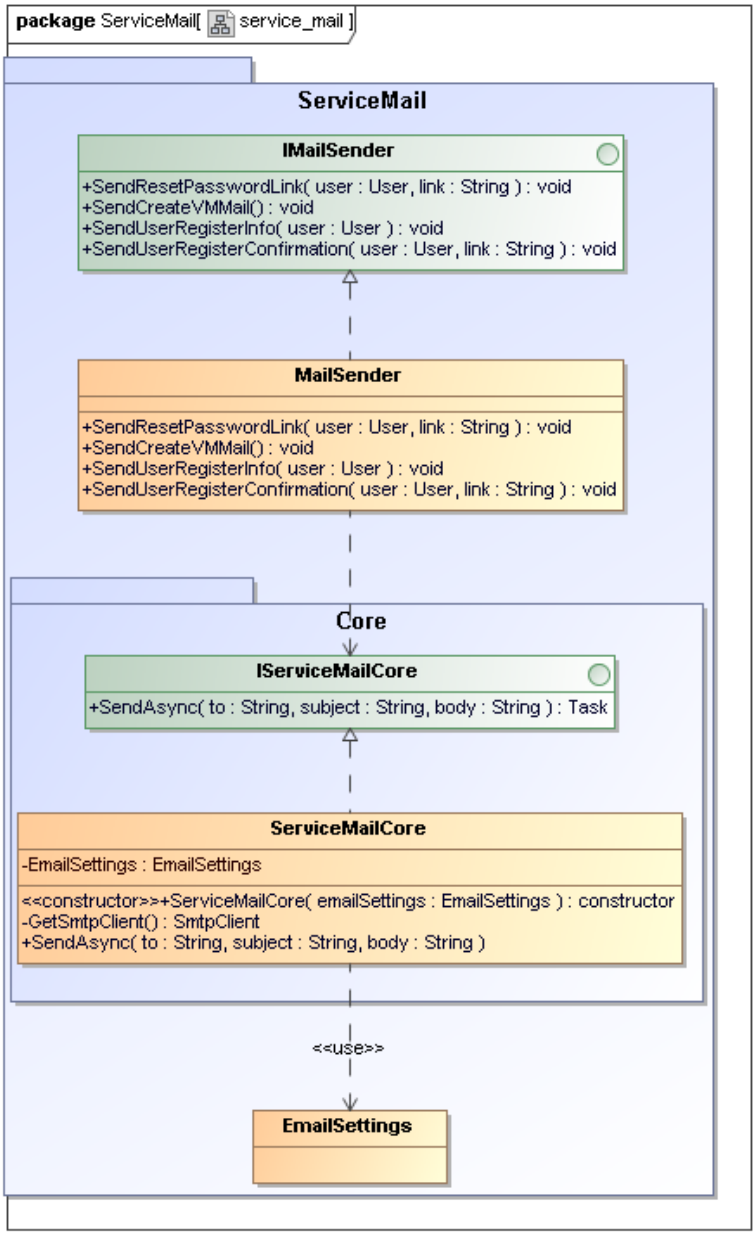

Figura C.9: Diagrama UML da biblioteca UVMM.ServiceMail