Universidade de Aveiro
2014

Departamento de Eletrónica, Telecomunicações e Informática

Joana Catarina Santos Silva

Arquitetura de Apoio à Desmaterialização de Operações Bancárias

Universidade de Aveiro
2014

Departamento de Eletrónica, Telecomunicações e Informática

Joana Catarina Santos Silva

Arquitetura de Apoio à Desmaterialização de Operações Bancárias

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Sistemas de Informação, realizada sob a orientação científica do Doutor Cláudio Teixeira, Equiparado a Investigador Auxiliar e do Doutor Joaquim de Sousa Pinto, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

Dedico este trabalho à minha mãe e irmã pelo apoio incondicional.

o júri / the jury

presidente / president

Prof. Doutor José Manuel Matos Moreira
Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais / examiners committee

Doutora Mariana Curado Malta
Professora Adjunta no Instituto Superior de Contabilidade e Administração do Porto

Doutor Cláudio Jorge Vieira Teixeira
Equiparado a Investigador Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

agradecimentos /
acknowledgements

Os meus primeiros agradecimentos dirigem-se sem margem de dúvida para a minha família, mãe e irmã por todos os sacrifícios que fizeram para me permitir alcançar aquilo que hoje sou. Ao Carlos, Joana e Rita, pelo apoio incondicional que sempre me deram ao longo destes anos, nos momentos bons e menos bons.

Ao Professor Cláudio Teixeira e Joaquim Sousa Pinto por terem acreditado nas minhas capacidades aquando da proposta desta dissertação e pelo apoio prestado ao longo da mesma.

Aos meus colegas do IEETA, pela paciência, incentivo e apoio que demonstraram ao longo deste percurso.

A todos os colegas e amigos com que me cruzei ao longo do meu percurso académico e também no Núcleo de Estudantes de Informática, Associação de Eletrónica, Telecomunicações e Telemática da Universidade de Aveiro, Comissão de Curso e Comissão de Faina que acreditaram nas minhas capacidades e que me permitiram uma grande aprendizagem, crescimento profissional e pessoal.

A todos o meu muito obrigado.

palavras-chave

Sistemas bancários, desmaterialização, arquitetura, cloud, híbrida, mobile

resumo

Esta dissertação tem com principal objetivo a conceção de uma arquitetura de referência que permita desmaterializar as operações realizadas nos balcões das agências bancárias. É a relação entre a banca e os seus clientes, nomeadamente através dos gestores de clientes, que se pretende alcançar com o *Tablet Ecosystem Apps* (TEA). A solução pretende dotar as equipas de Gestão Comercial dos Bancos de uma solução móvel, ágil e intuitiva em que os processos de balcão estarão desmaterializados.

keywords

Bank system, dematerialization, architecture, cloud, hybrid, mobile

abstract

The main objective of this dissertation is to design a reference architecture which enables the dematerialization of the transactions that are executed at the counters of bank branches. The relationship between banks and their customers, through client managers, will improve with the Tablet Apps Ecosystem (TEA). The solution aims to provide the Management of Commercial Banks with a mobile, agile and intuitive solution through which the counter processes will be dematerialized.

Índice

Índice de Figuras

Índice de Tabelas

Lista de Acrónimos

ACS	Access Control Service
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
ATM	Automated Teller Machine
CDBI	Component Based Development and Integration
CDN	Content Delivery Network
CORBA	Common Object Request Broker Architecture
CORS	Cross-Origin Resource Sharing
DaaS	Database as a Service
EC2	Elastic Compute Cloud
GB	Gigabyte
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
JSON	JavaScript Object Notation
NIB	Número de Identificação Bancária
NIST	National Institute of Standards and Technology
OSF	Open Software Foundation
PaaS	Plataform as a Service
PALOP	Países Africanos de Língua Oficial Portuguesa
RMI	Remote Method Invocation
REST	REpresentational State Transfer
SaaS	Software as a Service
SMS	Short Message Service
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol

S3	Simple Storage Solution
SQL	Structured Query Language
SSL	Secure Sockets Layer
SHA	Secure Hash Algorithm
SWT	Simple Web Token
TaaS	Testing as a Service
TB	Terabyte
TDS	Tabular Data Stream
TEA	Tablet Ecosystem Apps
TLS	Transport Layer Security
UUID	Universal Unique Identifier
VM	Virtual Machine
XML	eXtensible Markup Language
W3C	World Wide Web Consortium
WSDL	Web Service Description Language
WS	Web Service
WRAP	Resource Web Authentication Protocol
WCF	Windows Communication Foundation

1

Introdução

1.1. Motivação

Hoje em dia, nas instituições financeiras, os sistemas de informação que formam as infraestruturas são designados de processos core. Estes processos são responsáveis pelas transações e cálculos financeiros que culminam em depósitos, atualizações de depósitos e aplicações financeiras. As instituições bancárias, como nós as conhecemos, encontram-se organizadas numa rede de balcões físicos espalhados por diferentes pontos. É nos balcões de atendimento que os clientes tratam de grande parte dos movimentos financeiros, nomeadamente os depósitos em numerário ou em cheque, abertura de contas, aquisição de créditos entre muitos outros movimentos.

As entidades bancárias, nacionais e internacionais, têm apostado ao longo dos últimos anos em soluções eletrónicas para os seus clientes, dando-lhes a liberdade de gerirem e realizarem os seus próprios movimentos, o designado *home banking*. Assim, atualmente os bancos disponibilizam mais do que um canal de interação com os clientes finais e internos: *home banking*, *call centres*, ATM, portais para parceiros e agentes de vendas são canais de comunicação comuns com os sistemas core do banco.

Depois do lançamento destes sistemas de *home banking*, muitas foram as instituições bancárias que continuaram a desenvolver serviços que lhes permitam estar cada vez mais perto dos seus clientes. O passo seguinte passou pela disponibilização destes serviços em dispositivos móveis, que permitem aos seus clientes realizar pequenas operações, como transferências bancárias, pagamentos e serviços entre outros tudo através dos seus dispositivos móveis, o denominado *mobile banking*. A Juniper Research estima que em 2019

sejam cerca de 1,75 mil milhões os utilizadores a recorrerem a este tipo de aplicações. Hoje em dia, os números apontam para cerca de 800 milhões de pessoas a recorrerem a estes serviços [1]. Estes números resultam da massificação dos *smartphones* e dos *tablets* bem como do uso da Internet como um canal privilegiado para possibilitar o acesso remoto aos sistemas de informação das instituições bancárias.

No entanto, o desenvolvimento de ferramentas de *home banking* tem tido uma maior expressão para o cliente final do que para os gestores de clientes. Este constrangimento prende-se com o facto dos processos de banca estarem assentes numa estrutura monolítica que dificulta a agilização dos mesmos, bem como o mapeamento para aplicações em dispositivos móveis. Ou seja, os gestores de conta necessitam de aceder aos sistemas core para a realização de processos e se a realização destes processos for feita fora da rede da entidade bancária, isto levanta questões de segurança no acesso à rede. Assim, o gestor de clientes fica limitado face aos locais e formas de acesso aos dados fora da instituição.

A mobilidade indicada pelos *tablets* não é revolucionária face ao que já nos é apresentada pelos computadores portáteis atuais, no entanto, estes proporcionam-nos uma mobilidade diferente, sendo facilmente transportáveis e de fácil e intuitiva utilização.

É a relação entre a banca e os seus clientes, nomeadamente através dos gestores de clientes, que se pretende alcançar com o *Tablet Ecosystem Apps* (TEA). A solução pretende dotar as equipas de Gestão Comercial dos Bancos de uma solução móvel, ágil e intuitiva em que os processos de balcão estarão desmaterializados.

1.2. Principais Objetivos

Tal como referido anteriormente, já existem muitas aplicações bancárias destinadas aos clientes finais, no entanto, as equipas de gestão comercial ainda se encontram limitadas às instituições bancárias para poderem realizar operações. Assim, o principal objetivo desta dissertação passa pela conceção de uma arquitetura de referência que permita desmaterializar as operações realizadas nos balcões das agências bancárias. Apesar de já ser possível a realização de um grande conjunto de operações através de dispositivos móveis,

estas são apenas consideradas pequenas operações. A grande maioria das operações ainda necessita de uma visita do cliente a um balcão físico, ou seja, apenas podem ser realizadas presencialmente [2] [3]. Por exemplo, ainda não é possível a abertura de uma conta através de uma aplicação móvel, nem o depósito de dinheiro. É este tipo de operações que o TEA pretende desmaterializar. Os canais de comunicação tradicionais entre o cliente e o banco são: contacto presencial, *call center*, ATM e Internet. Pretende-se que o TEA seja um novo canal de comunicação.

Para países desenvolvidos, pode não ser fácil entender este conceito dado que existem muitas e variadas instituições bancárias com vários balcões espalhados que abrangem a maioria da população, no entanto, para os Países Africanos de Língua Oficial Portuguesa (PALOP) este conceito terá certamente uma grande penetração no mercado.

Pretende-se que este novo canal de comunicação seja determinante para a sustentabilidade e crescimento da banca em Mercados Emergentes. Tem-se assistido à bancarização da população, no entanto, o crescimento dos balcões físicos não consegue acompanhar esta necessidade, bem como a formação de quadros para suportar a operação bancária [4] [5]. Os serviços móveis por outro lado têm uma elevada taxa de penetração nos países em desenvolvimento. Por exemplo o projeto M-Pesa da Safaricom no Quénia, no qual o serviço/infraestrutura de comunicações móvel é usado para fazer pagamentos e transferências [6]. A maioria dos países emergentes já possuem infraestruturas como o 3G ou o WI-FI que permitem a utilização de tecnologia móvel.

Assim, o TEA pretende responder aos temas identificados, potenciando o contacto pessoal, dado que será o banco a dirigir-se ao cliente e não o contrário, passando o conceito de centrado no banco para o conceito centrado no cliente, como já acontece em vários serviços, nomeadamente na saúde. Mas também por ser uma solução móvel nas mãos do gestor de conta, poderá contribuir ativamente para a bancarização de Países Emergentes pois apresenta uma taxa de penetração mais rápida do que a construção de balcões físicos e de toda a infraestrutura associada. O que se pretende é disponibilizar processos que atualmente apenas se encontram em balcões físicos.

1.3. Contribuição

Ao longo desta dissertação, pretende-se estudar, avaliar e desenvolver uma arquitetura que permita sustentar a desmaterialização de operações bancárias através de dispositivos móveis. Inicialmente serão estudadas algumas plataformas de cloud, a fim de perceber qual será a mais indicada para este fim. Seguidamente será apresentada a arquitetura proposta para a resolução do problema. Por fim será apresentada a solução protótipo que suporta a arquitetura desenvolvida e serão retiradas as conclusões.

1.4. Organização da Dissertação

A presente dissertação está dividida em seis capítulos, sendo este enquadramento inicial o primeiro capítulo. No segundo capítulo serão descritos alguns casos de estudo de serviços que já deram os primeiros passos no sentido da desmaterialização de operações e serão descritos ainda aspetos gerais relativos à Arquitetura Orientada a Serviços. Serão referidos os seus princípios e definições e a sua importância no contexto do problema.

No terceiro capítulo é abordada a questão do *cloud computing* e das suas vantagens na construção deste tipo de arquiteturas. Neste capítulo é também feita uma comparação entre alguns fornecedores de *cloud*, a fim de perceber a escolha feita.

No quarto capítulo é apresentada a arquitetura concebida para o problema bem como a sua definição. O quinto capítulo apresenta o protótipo desenvolvido para suportar a arquitetura e as suas funcionalidades. No sexto e último capítulo, são apresentados os problemas, trabalho futuro e as respetivas considerações finais.

2

Casos de Estudo e Conceitos Subjacentes

Para a compreensão dos conceitos expostos nos próximos capítulos, torna-se necessário conhecer os projetos desenvolvidos nesta área e compreender os conceitos subjacentes. Assim, neste capítulo serão apresentados alguns casos de estudo de arquiteturas já desenvolvidas nesta área e seguidamente serão apresentados alguns princípios e conceitos da definição de arquiteturas.

2.1. Casos de Estudo

O processo de definição e implementação de *software* é por vezes um processo difícil de definir. Esta dificuldade na definição e a constante evolução tecnológica que se assiste nas tecnologias de informação conduz a que várias instituições financeiras desenvolvam os seus próprios sistemas internos. Na maioria das vezes os sistemas desenvolvidos tornam-se isolados, o que conduz a silos de informação que evoluem em separado, mas que nem sempre respondem às necessidades distintas. Com necessidades entenda-se, por exemplo, a partilha de informação entre os vários sistemas das instituições financeiras. Em consequência, há necessidade de manter e suportar sistemas, que depois de um longo tempo de utilização dificilmente podem ser descontinuados. Ainda existe uma grande falha na cultura da interoperabilidade.

Já se vão encontrando algumas soluções que pretendem dar resposta a este problema, no entanto, não foi encontrada nenhuma informação referente à aglutinação, na mesma

arquitetura de aspetos de segurança, comunicação, armazenamento local e acesso a sistemas legados por uma camada integradora, como é o propósito desta dissertação.

A definição de uma arquitetura de referência pretende responder ao problema em questão. A utilização de SaaS (*Software as a Service*) na banca tem-se executado como uma forma de os próprios bancos resolverem alguns dos seus problemas como é o caso da redução de recursos humanos, agilização de processos e ainda a disponibilização dos já referidos serviços de *home banking*. Exemplo isso é a IDEALINVEST que oferece serviços de B-SaaS com alguns processos bancários pré-definidos e com interface de ligações aos sistemas legados dos bancos [7]. Esta tem-se tornado uma área de grande atividade e que está cada vez mais a despertar o interesse de empresas que se consideram apologistas do modelo tradicional, como é demonstrado pela aquisição por parte da Temenos da empresa TriNovus [8]. Também a BTR, conjuntamente com o VASCO Data Security, a KBC e a secção de serviços Financeiros da Oracle lançaram uma plataforma de SaaS específica para serviços de *backoffice* para retalho, contemplando depósitos correntes e a prazo, e pagamentos entre outros [9].

A CPay 360 propõem uma solução de SaaS para bancos comunitários [10]. Com esta solução argumenta que os seus clientes poderão competir (em termos de funcionalidades apresentadas aos clientes) com grandes multinacionais bancárias. Também a CSC [11] apresenta a EarlyResolution (SaaS) capaz de lidar com processos de empréstimos. A maioria das soluções apresentadas aborda a plataforma SaaS como gerador de interfaces *Web*/terminal para utilização dos funcionários do banco (balcão) [12] para gestão da presença do banco na Web com sistema de *home banking*. Em [13] é apresentado o NCR APTRA Mobile BankingGateway. Trata-se de um conjunto de *middleware* e solução *mobile* baseada no mFoundry [14]. Um dos aspetos interessantes do APTRA é a sua presença em três canais distintos: aplicação nativa, aplicação Web e comunicação por SMS.

Apesar de todas as soluções acima apresentadas, a falta de arquiteturas que sirvam de guia de referência para suportar as necessidades e requisitos regulatórios impostos pelo sector financeiro tem fomentado o crescimento e a evolução de soluções específicas e aplicáveis apenas a cada instituição financeira: o desenvolvimento destes sistemas aplicacionais é

caracterizado por requisitos próprios de cada País/Instituição de acordo com o modo como cada um interpreta as normas genéricas publicadas pelas Entidades Reguladoras. Assim, existe um baixo nível de reaproveitamento e de aceleradores que promovam estas tarefas de desenvolvimento.

Os desafios globais forçam as organizações financeiras a procurar uma nova base tecnológica para os negócios. A combinação de Arquitetura Orientada a Serviços (SOA) e SaaS oferece às organizações a maior expansibilidade possível por um baixo custo e permite às mesmas estar rapidamente prontas para uma mudança para SaaS. O SaaS combinado com um repositório de componentes de SOA fornece a capacidade de alinhar a tecnologia com os negócios. As aplicações monolíticas *on-premise* já não são viáveis para fornecer produtos e serviços em ambientes altamente dinâmicos [15].

Dados os exemplos mencionados anteriormente, facilmente podemos concluir que quase todos se baseiam em soluções SaaS para conseguirem aproximar-se das necessidades dos seus clientes. Por outro lado, soluções como o NCR APTRA apresenta três canais de comunicação com os seus clientes, para além do *middleware* e da solução *mobile* baseada no mFoundry. A Tabela 2.1 abaixo apresentada, mostra uma pequena comparação entre as soluções e o que estas oferecem.

	IDEALINVEST	BTR	CPAY360	CSC	NCR APTRA
Home Banking				X	
SaaS	X	X	X	X	
Sistemas Legados	X				
Backoffice		X			
Middleware					X

Tabela 2.1 - Tabela Comparativa entre soluções existentes

Através da análise da Tabela 2.1, podemos concluir que à exceção do NCR APTRA, todas as outras soluções se baseiam em SaaS. Podemos também ver que a BTR é a única que

disponibiliza um *backoffice* e que a NCR APTRA é a única que disponibiliza *middleware*. A IDEALINVEST é também a única que apresenta conexão com sistemas legados.

Apesar de todas as soluções mostrarem já bons progressos face à utilização de SaaS nas entidades financeiras, nenhuma delas consegue responder ao problema que com esta dissertação se pretende responder. A solução que mais se aproxima é a da IDEALINVEST que utiliza um cenário SaaS e ainda tem conexão com sistemas legados. No entanto, não apresenta soluções para a desmaterialização de operações em ambientes *mobile*, comunicação e armazenamento local tal como é pretendido.

A junção de SaaS e SOA permite às entidades bancárias redesenhar os processos de negócio o mais rápido possível [15]. Como tal, e dado o potencial de escala necessário na utilização do TEA será conveniente a adoção de um modelo de SaaS para a construção da arquitetura e é importante perceber como é que uma conjugação com o SOA pode ativar e melhorar os processos de negócio subjacentes.

Para podermos compreender como é que esta conjugação poderá ser benéfica para a construção de uma arquitetura de referência, é necessário compreender algumas definições, princípios e conceitos subjacentes. Estas definições passam por Serviços, Serviços *Web* e por fim a concreta definição de Arquitetura Orientada a Serviços (SOA). É também importante perceber como é que estes conceitos se relacionam com a arquitetura de referência que se pretende desenvolver ao longo desta dissertação.

Tal como referido anteriormente, pretende-se o desenho e conceção de uma arquitetura de referência que consiga suportar uma aplicação móvel que permita a desmaterialização de algumas operações bancárias. Já vimos alguns exemplos de casos onde se tentou utilizar arquiteturas que disponibilizavam serviços que permitem responder a diversos problemas, nomeadamente, serviços de *home banking*, serviços financeiros, empréstimos, entre outros.

À medida que o tamanho e a complexidade dos sistemas de *software* aumenta, o problema da conceção vai além dos algoritmos e estruturas de dados da computação: conceber, desenhar e especificar a estrutura geral do sistema emerge como um novo tipo de problema… Isto é a conceção ao nível da arquitetura de *software* [16].

A arquitetura é a estrutura ou estruturas do sistema que englobam componentes de *software*, propriedades visíveis externamente e relações entre elas [17].

Quando nos referimos à arquitetura, este conceito remete-nos quase automaticamente para o SOA, no entanto, este é um conceito com variadas definições e por isso, nem sempre de fácil compreensão.

Comecemos por perceber as definições e os conceitos de serviço. Segundo a World Wide Web Consortium (W3C), um serviço é um componente capaz de executar uma tarefa. Já a Component Based Development and Integration (CBDI) define como um tipo de capacidade descrita usando um *Web Service Description Language* (WDSL). Acrescenta que é um veículo pelo qual o que um consumidor precisa ou quer, é satisfeito de acordo com um contrato negociado (implícito ou explícito) que inclui o contrato, funções entre outros. No entanto é também necessário ter em consideração os serviços que operam sem WSDL, como é o caso do *Representational State Transfer* (ReST) que se trata de um protocolo de comunicação que opera sobre o protocolo HTTP [18]. Um serviço pode também ser definido como uma unidade autónoma não independente da plataforma e que pode ser publicado, descoberto e agregado a outros de novas formas. Os serviços podem invocar outros serviços.

Ainda no que respeita á definição de serviço *Web*, a W3C refere que é um sistema de *software* projetado para suportar a interação interoperável máquina-a-máquina através de uma rede. Tem uma interface descrita num formato em que as máquinas podem processar, como é o caso do WSDL, mas também do *Common Object Request Broker Architecture* (CORBA) e *Remote Method Invocation* (RMI). Por sua vez, a CBDI descreve-os como uma interface de programação de um recurso que está em conformidade com os protocolos WSnn, que se trata de um formato de *broadcast* de estações de rádio.

Tendo em conta as definições apresentadas, torna-se claro que a W3C adota uma abordagem um pouco mais estreita para a definição de serviço e outros artefactos do que a CBDI. A CBDI difere na medida em que nem todos os serviços são componentes pois nem todos eles executam uma tarefa, sugerindo até que é útil gerir o tipo, a definição e a realização como itens separados [19].

2.2. Arquitetura Orientada a Serviços

Segundo o consórcio World Wide Web, o SOA pode ser entendido como *"Um conjunto de componentes que podem ser invocados, e cujas descrições das definições pode ser publicadas e descobertas."* Muitas das definições encontradas são bastante semelhantes a esta, no entanto, a palavra arquitetura é muitas vezes definida como descrição de um estilo ou um conjunto de práticas [19].

O SOA pode ser descrito como sendo uma arquitetura distribuída. Esta permite a invocação de serviços nomeadamente CORBA, RMI e serviços *Web* (SOAP,REST,XML,JSON). Apresenta uma independência total e as suas principais vantagens passam pelo facto de ser escalável, reutilizável, reconfigurável e poder ter configurações dinâmicas. É um paradigma de utilização de sistema e de exposição de lógicas de negócio na *Web*. Fundamentalmente é um modelo arquitetural cujo objetivo é aumentar a eficiência, agilidade e produtividade, focando nos serviços os meios pelos quais é representada a lógica de negócio [20].

Os serviços *Web* não são um componente obrigatório do SOA, no entanto cada vez mais se tornaram num. A definição de SOA é muito mais ampla do que simplesmente a definição do serviço, pois esta também aborda a qualidade do serviço a partir da perspetiva do fornecedor e do consumidor. É importante a existência de uma estrutura para entender o que constitui um bom serviço, pois existem diferentes níveis de utilidade para os mesmos. Assim, torna-se imprescindível conhecer alguns princípios gerais e boas práticas [19].

2.2.1. Princípios Gerais e Boas Práticas

Eis alguns princípios gerais e boas práticas que devem constar de qualquer serviço: [20]

- Reusabilidade: os serviços devem ser construídos de forma a ser possível a sua reutilização noutros possíveis cenários de negócio;
- Granularidade: os serviços devem ser específicos o suficiente de forma a puderem dar resposta à resolução do problema;

- Modularidade: os serviços devem ser entendidos como sendo auto-suficientes. Devem ser entendidos como módulos que processam de forma independente e autónoma;
- Composabilidade e Componentização: deve suportar composição de serviços mais elaborados;
- Adesão a *standards*: sempre que existam *standards* estes devem ser utilizados e respeitados.

2.2.2. Princípios Específicos

Especificamente existem algumas características que não devem ser descuradas: [20]

- Encapsulamento: um serviço deve ser entendido como uma caixa negra. Apenas é importante saber quais são os dados esperados como parâmetros de entrada e qual é o resultado esperado;
- Separação da Camada de Negócio da Tecnologia de Base: devem ser abstraídas todas as referências ao serviço que está a suportar isso;
- Implementação única e vista global de componentes: este princípio acaba por ser um corolário dos princípios gerais acima mencionados: reutilização e implementação única;
- Reaproveitamento de lógicas existentes sempre que possível: tal como o título indica, sempre que possível deve-se pegar em lógicas externas e expô-las como um serviço, reaproveitando-as para um novo serviço;
- Serviços vagamente unidos: os serviços podem e devem funcionar com outros, no entanto necessitam da especificação de contratos. Um contrato, o que faz é indicar quais as necessidades do serviço para que possa funcionar e quais é que são os seus resultados;
- Gestão do ciclo de vida

- Uso eficiente dos recursos do sistema: é fundamental que os serviços acautelem questões de desempenho e escalabilidade;
- Maturidade e desempenho do serviço: os serviços devem ser robustos e ter um bom desempenho.

2.3. Tipos de arquiteturas do SOA

No SOA existem três perspetivas arquitetónicas interessantes:

- Arquitetura da Aplicação

Representa a solução de negócio voltada para quem consome serviços de um ou mais fornecedores e tenta integrá-los em processos de negócio;

- Arquitetura de Serviço

Proporciona uma ponte entre as implementações e os aplicativos de consumo, criando uma visão lógica de conjuntos de serviços que estão disponíveis para utilização, invocadas por uma interface comum e arquitetura de gestão.

- Arquitetura de Componente

Descreve os vários ambientes que suportam as aplicações implementadas, os objetos de negócio e as suas implementações.

Estas arquiteturas podem ser vistas na perspetiva de qualquer consumidor ou fornecedor. A chave de qualquer arquitetura passa pelo facto de nenhum consumidor estar interessado nos detalhes de implementação do serviço, apenas no serviço prestado em si. A implementação da arquitetura varia de fornecedor para fornecedor. Da mesma forma que o consumidor, o fornecedor não está interessado nos detalhes da aplicação que vai consumir o serviço.

O consumidor foca-se na arquitetura de aplicação, os serviços utilizados mas não nos detalhes da arquitetura de componente. Existe interesse nalgum nível de detalhe nos

objetos de negócio em geral pois são de interesse mútuo, pois existem conceitos que ambos têm de conhecer. De forma semelhante, o fornecedor está focado na arquitetura de componentes, a arquitetura de serviço, mas não sobre a arquitetura da aplicação. Mais uma vez, devem partilhar regras, pré e pós condições [19].

Um dos grandes objetivos do SOA é uma grande rede de serviços de colaboração, que são publicados e disponibilizados para invocação do *Service Bus* [19]. Adotar o SOA é essencial para agilizar, integrar e flexibilizar negócios. Torna-se necessário não visualizar a arquitetura apenas pelo ponto de vista tecnológico mas também através da adoção de protocolos de serviços da *Web*. Fundamentalmente criar um ambiente orientado a serviços. É importante perceber que o SOA não é apenas uma arquitetura de serviços vista apenas pelo ponto de vista tecnológico, é também um conjunto de perspetivas, políticas, práticas e *frameworks* que permitem garantir que os serviços são adequadamente fornecidos e consumidos. Com SOA é fundamental implementar processos que forneçam duas perspetivas diferentes – para o consumidor e para o fornecedor [19].

É nesta perspetiva que este projeto pretende ser desenvolvido. Mais do que o desenvolvimento de serviços e a construção de uma arquitetura, é fundamental delinear a linha de negócio subjacente. É também importante que diferentes serviços provenientes de diferentes fornecedores cheguem ao cliente final sem que ele se aperceba disso. Assim, torna-se necessário que durante a conceção da arquitetura seja tido em conta a possibilidade de agregação de serviços de diferentes fornecedores e de sistemas legados. Nos próximos capítulos será descrita a arquitetura proposta bem como os serviços utilizados.

3

Cloud Computing

Segundo o Instituto Nacional de Standards e Tecnologia (NIST), o *Cloud Computing* é um modelo para permitir convenientemente o acesso à rede para um conjunto compartilhado de recursos computacionais configuráveis que podem ser rapidamente adicionados e libertados com o mínimo esforço de gestão ou interação do fornecedor de serviços [21].

O modelo *cloud* promove disponibilidade e é composto por cinco características essenciais: serviços *on-demand*, acesso à rede, conjunto de recursos, rápida elasticidade e serviços por medida. As tecnologias facilitadoras essenciais incluem: redes amplas, poderosos servidores de baixo custo e virtualização de alta performance [21].

Por norma, as aplicações que utilizamos encontram-se instaladas nos nossos próprios dispositivos. Já em ambientes empresariais, o mais presente passa por utilizar aplicações disponíveis em servidores, que são depois utilizadas por qualquer dispositivo dentro da rede.

No entanto, graças à evolução da tecnologia e das telecomunicações, o acesso à Internet está cada vez mais abrangente e mais rápido. Já existem muitos países onde é possível aceder à Internet a partir de qualquer local e a preços muito reduzidos [22]. É graças a esta evolução e abrangência que o conceito de *cloud computing* está cada vez mais presente e é cada vez mais popular.

O *cloud computing* permite que aplicações, dados e ficheiros não tenham que estar instalados nos nossos dispositivos e mesmo assim possamos usá-los como se estivessem. O conteúdo é disponibilizado através do acesso à Internet. O processo de desenvolvimento, armazenamento e manutenção cabe aos fornecedores de aplicações, ao cliente cabe apenas a tarefa de usar estas aplicações [22].

3.1. Características – Vantagens e Desvantagens

Como já foi mencionado anteriormente, uma das grandes vantagens das aplicações na *cloud* é o facto de estarem acessíveis em qualquer lugar através do acesso à Internet, no entanto existem muitas outras vantagens. O facto de poder ser executado em qualquer sistema operativo, sendo assim independente de sistema ou *hardware*. O cliente também não precisa de se preocupar com a estrutura e funcionamento interno das aplicações, pois estas ficam a cargo do fornecedor de serviços. Existe a flexibilidade de escalar os serviços de acordo com as necessidades que se vão sentindo. Também o preço de acesso depende do que se usa. [20].

Relativamente às desvantagens, a principal é o problema do *vendor lock-in*. Há mais de uma década que se tem trabalhado em prol da construção de soluções baseadas em normas e protocolos comuns que podem ser construídos, apoiados e substituídos independente do seu fornecedor [23]. No entanto, este conceito regrediu quando falamos de *cloud computing*. Ao usarmos uma solução baseada em *cloud*, significa estar a utilizar protocolos, padrões e ferramentas de um fornecedor específico, tornando a migração futura uma tarefa cara e difícil. Muitas vezes olha-se apenas para os custos de utilização de serviços, mas não são tido em consideração os custos de uma possível migração [23].

Quando se opta pela utilização de soluções *cloud*, é necessário ter em consideração os custos associados. É necessário fazer uma cuidada análise do que estará envolvido e quais os seus custos.

3.2. Camadas Lógicas

O *cloud computing* encontra-se dividido em três camadas lógicas: *Software as a Service* (SaaS), *Plataform as a Service* (PaaS) e *Infrastructure as a Service* (IaaS).

3.2.1. Software as a Service

O SaaS é uma forma de integrar aplicações como um serviço através da Internet. Em vez de efetuar uma instalação, é acedido através da Internet, excluindo assim gestão de *hardware* e *software*. Estas aplicações são muitas vezes chamadas de *software* baseado em *Web*. Estas são executadas através de fornecedores de SaaS e são estes mesmos fornecedores que gerem o acesso à aplicação, incluindo a segurança, disponibilidade e desempenho. Algumas das características do SaaS são o facto de ter uma arquitetura *multi-tentant*, isto é, todas as aplicações e utilizadores partilham uma única e comum infraestrutura que é mantida centralmente. Outra característica é o facto de ser de fácil personalização, isto é, os clientes podem facilmente personalizar as suas aplicações de acordo com os seus processos de negócio [24].

3.2.2. Plataform as a Service

O PaaS é uma forma de utilizar a plataforma como um serviço. É uma solução ampla para um conjunto determinado de aplicações, que inclui quase todos os recursos necessários à operação, como o armazenamento, base de dados, escalabilidade, suporte a linguagens de programação, segurança entre outros [22].

3.2.3. Infrastructure as a Service

O IaaS é uma forma de utilizar a infraestrutura como um serviço. O conceito é semelhante ao do PaaS, mas este é direcionado para o *hardware* e máquinas virtuais tendo o utilizador acesso a recursos do sistema operativo [22].

Estas são as três principais camadas do *cloud computing*, no entanto, ao longo do tempo têm surgido outra estruturas derivadas desta como é o caso do *Database as a Service* (DaaS) e do *Testing as a Service* (TaaS).

3.3. Tipos de Cloud

Dentro do conceito de *cloud computing*, é possível distinguir entre quatro tipos: *cloud* privada, *cloud* híbrida, *cloud* comunitária e *cloud* pública. Seguidamente vamos analisar a *cloud* privada e a *cloud* híbrida.

3.3.1. Cloud Privada

Neste tipo, os bastidores encontram-se dentro da infraestrutura da própria corporação que a utiliza. A corporação faz uso de uma *cloud* privada, construída e mantida dentro das suas instalações. Esta incorpora também a cultura corporativa, para que as políticas e objetivos da atividade da empresa sejam respeitados.

Numa primeira fase, os custos com o equipamento e sistemas de uma *cloud* privada podem ser elevados no entanto, os benefícios obtidos a médio e longo prazo poderão compensar os gastos [25].

3.3.2. Cloud Híbrida

De modo a flexibilizar as operações e até mesmo para um melhor controlo de custos, muitas organizações optam pela utilização de *clouds* híbridas, em que se conjuga os conceitos de *cloud* privada e de *cloud* pública. Algumas das aplicações e serviços encontram-se em *clouds* públicas enquanto outros serviços mais sensíveis se encontram sobre a responsabilidade de *clouds* privadas. Nesta solução também podem existir recursos que funcionam em sistemas locais, ou seja, *on-premise*, complementando o que se encontra na nuvem [22].

Cada vez mais as corporações optam por este conceito, pois consegue agregar as vantagens tanto das *clouds* privadas como das *clouds* públicas.

3.4. Aplicações em Cloud Computing

Apesar de este conceito ainda ser relativamente novo, se analisarmos os serviços que a usam, podemos facilmente perceber que afinal não se trata assim de um conceito tão novo. Todos os dias usamos serviços que utilizam este conceito, com o serviço de *email* da Gmail, o armazenamento da Dropbox ou até mesmo a partilhar de vídeos do Youtube.

Já existem vários serviços que incorporam este conceito de *cloud computing* como é o caso da Google Apps, Amazon como a Simple Storage Solution (S3) e a Elastic Compute Cloud (EC2) ou o Windows Azure da Microsoft [22].

3.5. Comparação entre Fornecedores de Serviços Cloud

Pegando nos exemplos acima mencionados, e dado que será necessário escolher uma solução *cloud* para o desenvolvimento da arquitetura e do respetivo protótipo, foi analisado o estudo da Nasuni – *The State of Cloud Storage - A Benchmark Compariosn of Performance*, *Availability and Scalability* [26]. A escolha deste estudo recai no facto de este ser um estudo com credibilidade e que apresenta de forma muito organizada um conjunto de testes e os seus respetivos resultados. Este estudo utilizou métricas de comparação e realizou testes de forma a entender o desempenho de alguns fornecedores de serviços *cloud*.

3.5.1. Métricas de Comparação

Neste estudo, a Nasuni definiu três métricas de comparação entre os diferentes fornecedores de serviços *cloud*:

Funcionalidade: Normalmente a grande maioria das interações que uma empresa tem com os fornecedores de serviços *cloud* consiste em comandos simples da API como o GET, PUT e DELETE. No entanto, as empresas devem considerar uma ampla gama de funcionalidades quando se trata de comparar fornecedores. Dado que a maioria das empresas são globais e se encontram um pouco por todo o mundo, torna-se fundamental que os fornecedores disponibilizem servidores em várias partes do mundo e apoiem a replicação

geograficamente cruzada. É também importante ter em consideração características como o processo de criação e gestão de contas, disponibilidade de bibliotecas, *software* para aceder aos dados e faturação.

Preço: O armazenamento em *cloud* é diferente do armazenamento tradicional, o que se reflete na forma como este é cobrado. Tradicionalmente, o armazenamento é cobrado em TB, no entanto, a maioria dos fornecedores cobra com base em GB armazenados por mês. Contudo, esta contagem não é assim tão linear pois muitas vezes são adicionados os custos de computação e os custos de rede.

Hoje em dia, os fornecedores oferecem formas de ajudar a calcular os custos conforme as necessidades dos utilizadores e permitem também prever custos no caso de ampliação de serviços.

O preço dos serviços é uma parte muito pequena desta comparação e deve ser utilizado em último caso na hora de tomar uma decisão pois as funcionalidades e a performance são indicadores bastante significativos.

Performance: É principalmente através da performance que a Nasuni compara os diferentes fornecedores, testando o funcionamento e a estabilidade durante longos períodos de tempo. Vamos conhecer a performance mínima de referência nestas três áreas:

- Leitura/Escrita/Eliminação: foi testada a forma como os fornecedores lidam com milhares de leituras, escritas e eliminações. Foram testados ficheiros com vários tamanhos e usando diferentes níveis de concorrência. Os testes correram durante 12 horas, usando múltiplas instâncias de máquinas e vários testes não seriais para reduzir a probabilidade de problemas de rede externos que poderiam distorcer os resultados.

- Disponibilidade: os testes de disponibilidade tiveram lugar num período de 30 dias e mediram o tempo de resposta de cada fornecedor a um único processo de leitura, escrita e eliminação em intervalos de 60 segundos. Ler e apagar arquivos aleatoriamente força a que os fornecedores provem a sua capacidade de ser sensível a todos os dados, o tempo todo, e não apenas para os dados que se encontram

armazenados em cache. Este teste calculou o tempo necessário para efetuar os três pedidos. Não se trata apenas da capacidade de resposta mas também da confiança no fornecedor bem como a latência.

- Escalabilidade: semelhante aos testes à disponibilidade, estes testes mediram a capacidade de executar de forma consistente o número de objetos e os aumentos de gestão. Este teste mediu a capacidade de cada fornecedor manter os níveis de desempenho como o número total de objetos armazenados num armazenamento único para milhões de pessoas.

3.5.2. Metodologia

O estudo da Nasuni inclui cinco fornecedores de serviços *cloud*, os quais: Amazon S3, Microsoft Azure Blob Storage, Google Cloud Storage, HP e Rackspace [26].

Neste estudo foram utilizadas duas máquinas virtuais com as características que se podem observar na Tabela 3.1.

	Máquina 1	Máquina 2
RAM	15-16 GB	4 GB
vCPUs	4	2
Sistema Operativo	Ubuntu 1204, 64 bit – Ubuntu 12.04 LTS	Ubuntu 1204, 64 bit – Ubuntu 12.04 LTS

Tabela 3.1 - Características da VMs

A distribuição do tamanho dos ficheiros usados pode ser observada na Tabela 3.2:

1KB	10KB	100KB	1MB	10MB	100MB	1GB
16.8%	24.6%	26.2%	9.7%	22.2%	0.4%	0.1%

Tabela 3.2 - Tamanho dos ficheiros

3.5.3. Resultados

3.5.3.1. Benchmark – Escrita

No teste de escrita, como podemos observar na Figura 3.1, a Microsoft lidera a performance de escrita. Além disso, a Microsoft superou todos os outros fornecedores em 14 das 23 combinações individuais testadas, tornando-se o alvo ideal para escrita em dados no formato de ficheiro. Já na escrita em ficheiros de tamanho superior a 1MB, como podemos observar na Figura 3.2, a Amazon destaca-se.

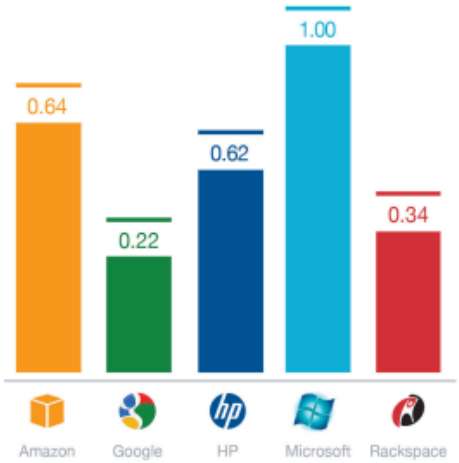

Figura 3.1 – Velocidade de armazenamento de escrita com ficheiros de todos os tamanhos [26]

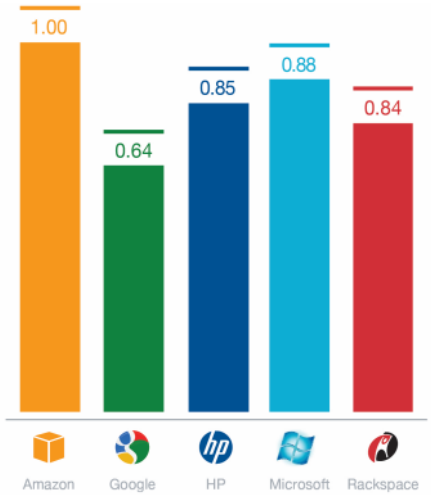

Figura 3.2 - Velocidade de armazenamento de escrita de ficheiros com tamanho > 1MB [26]

3.5.3.2. Benchmark – Leitura

No teste de leitura, mais uma vez a Microsoft lidera com bastante margem face aos restantes fornecedores, tanto na leitura de ficheiros de todos os tamanhos como na leitura de ficheiros superiores a 1MB, como podemos observar na Figura 3.3 e na Figura 3.4.

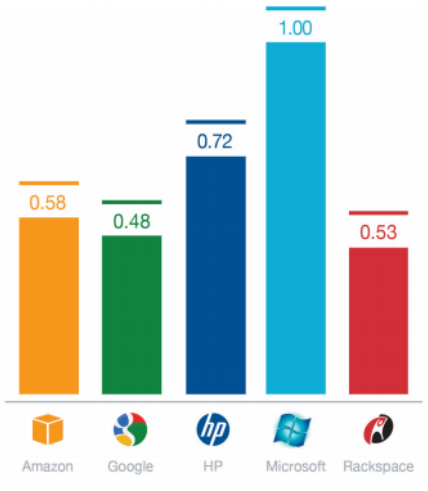

Figura 3.3 - Velocidade de armazenamento de leitura com ficheiros de todos os tamanhos [26]

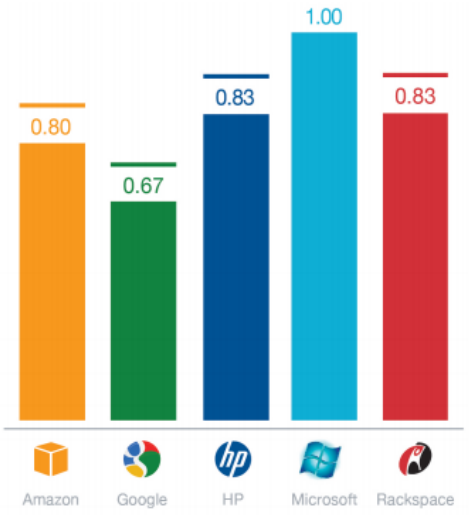

Figura 3.4 - Velocidade de armazenamento de leitura com ficheiros de tamanho > 1MB [26]

3.5.3.3. Benchmark – Eliminação

Para reforçar os resultados, nos restantes testes, a Microsoft é o dobro mais rápida na eliminação de ficheiros do que qualquer outro fornecedor. Em segundo lugar podemos encontrar a Amazon e com um desempenho bastante baixo encontramos a Rackspace, como podemos observar na Figura 3.5.

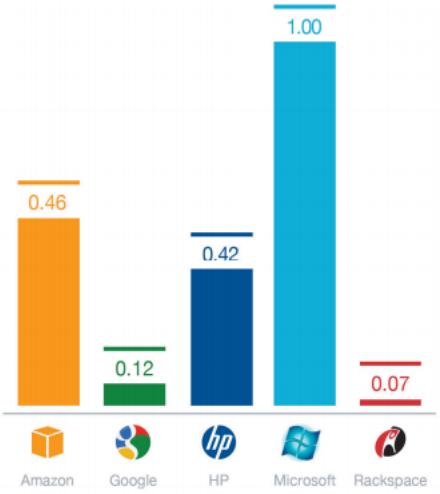

Figura 3.5 - Velocidade de eliminação [26]

3.5.3.4. Disponibilidade

A disponibilidade foi medida usando o tempo de resposta como métrica, que mede o tempo de reposta de cada fornecedor a um único processo de leitura, escrita ou eliminação em intervalos de 60 segundos. O tempo de resposta também inclui momentos associados com atrasos e tentativas.

Mais uma vez a Microsoft lidera o teste com um tempo médio de resposta de menos de 0.5 segundos durante um período de 30 dias. A Amazon encontra-se perto, com um resultado de 0.65 segundos. A Google apresenta o pior tempo de resposta com quase 2 segundos. Estes resultados podem ser observados na Figura 3.6.

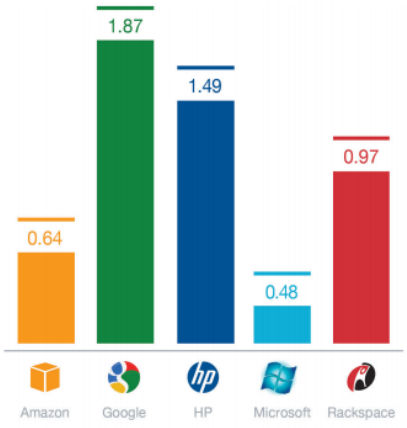

Figura 3.6 - Tempo médio de resposta (em segundos) [26]

Se analisarmos os resultados da Figura 3.7 ao longo do mês, podemos verificar também a variabilidade dos números. A Microsoft demonstra alguma estabilidade, contrariamente aos restantes que mostra uma maior variação nos resultados.

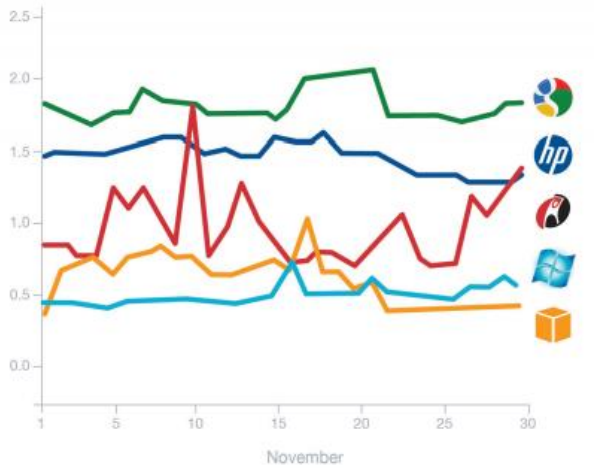

Figura 3.7 - Média diária do tempo de resposta (em segundos) [26]

Para além da disponibilidade dos dados e do sistema, o teste também mediu o tempo de atividade geral ou a percentagem de tempo que o fornecedor está acessível. Todos mostraram fortes percentagens, estando a Amazon e o Google na liderança, como podemos observar na Figura 3.8.

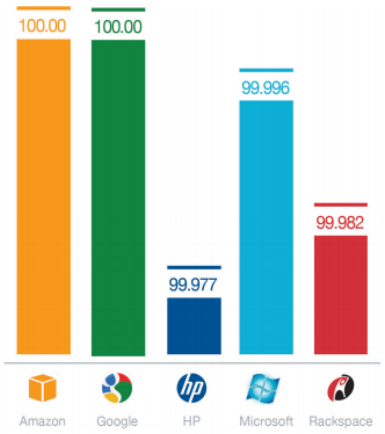

Figura 3.8 - Média de tempo a correr [26]

3.5.3.5. Escalabilidade

O número de objetos aumenta, a performance de alguns fornecedores degrada-se ou torna-se variável. Dependendo da arquitetura de cada fornecedor, alguns sistemas são desenhados para escalar através de contentores, não dentro deles. Este tipo de arquitetura traz limitações que se podem vir a tornar significativas após alguns meses ou mesmo anos de utilização.

Neste teste, todos os fornecedores foram carregados com novos objetos tão rapidamente quanto possível até 100 milhões de objetos ou 30 dias. A variação representa o quanto a velocidade de carregar os objetos mudou ao longo do tempo, causando inconsistência e variabilidade como os objetos foram carregados, que pode ser observado na Figura 3.9.

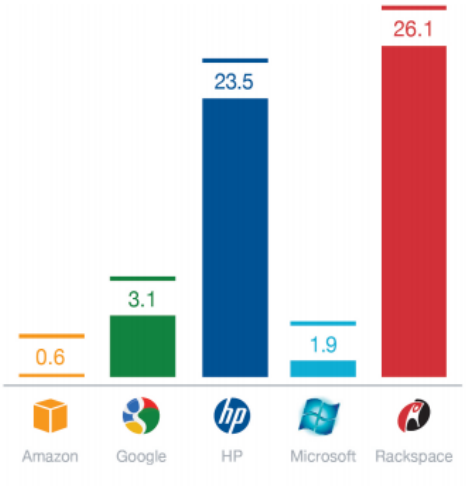

Figura 3.9 - Variância durante o escalonamento de novos objetos (% muda na velocidade de escrita) [26]

Comparando com o último resultado por anos, os erros de escrita e leitura durante a escalabilidade diminuem significativamente. Durante 100 milhões de escritas, apenas a HP e a Rackspace mostraram erros de escrita, como mostra a Figura 3.10.

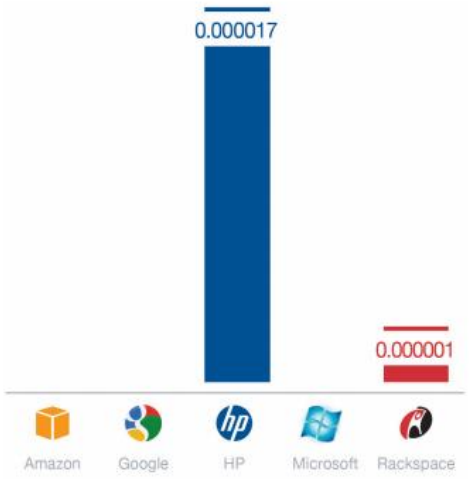

Figura 3.10 - Percentagem de erros de escrita [26]

Durante as leituras, apenas a Microsoft não apresentou nenhum erro. Segue-se a Amazon como podemos observar na Figura 3.11.

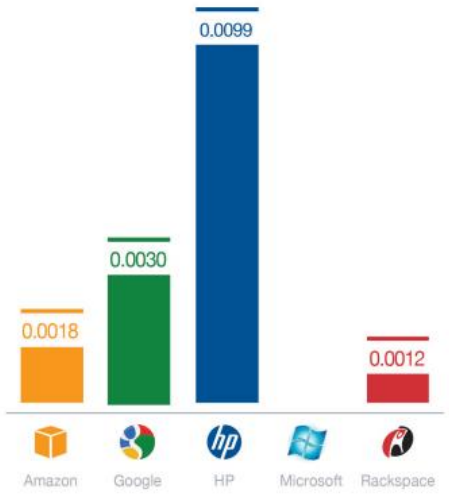

Figura 3.11 - Percentagem de erros de leitura [26]

3.5.3.6. Conclusão

A Microsoft apresentou de forma consistente uma melhor performance do que qualquer outro fornecedor com base nos testes apresentados, de acordo com as velocidades através da variação do tamanho dos ficheiros, com os tempos de resposta mais rápidos e com a taxa mais baixa de erros. Por estes motivos, a nível de *storage*, a Microsoft ultrapassa a Amazon e assume a posição de topo a nível de performance do ano de 2013.

Um outro estudo da *Cloud Spectator – A Comparative Analysis of 5 Large Cloud IaaS Providers* – destaca que os serviços de infraestrutura do Azure obtiveram o desempenho mais elevado e têm em média três vezes melhor preço pelo desempenho do que o Amazon EC2 [27] [28].

Com base nos estudos apresentados, a escolha do fornecedor de serviços de *cloud* recaí sobre a Microsoft.

3.6. Windows Azure

O Windows Azure é a plataforma *cloud* da Microsoft. Esta é flexível e permite criar, implementar e gerir aplicações numa rede global de *datacenters*, geridos pela própria

Microsoft. O Windows Azure permite usar um leque de linguagens de programação que passam pelo .NET, Node.js, Java, PHP, Ruby e Python para construir aplicações. Corre tanto em Linux como em Windows. As bibliotecas estão disponíveis para as várias linguagens citadas e são disponíveis de forma *open source* e hospedadas no GitHub [29].

Para além do que já foi referido, o Windows Azure é escalável e como tal permite facilmente escalar as aplicações. Permite ainda a monitorização dos recursos de forma flexível e de acordo com as necessidades [29].

As suas principais funcionalidades passam pela computação, desenvolvimento *Web* e *mobile*, dados e armazenamento, ferramentas de análise, rede, multimédia e integração de sistemas [29].

No próximo capítulo será apresentada e definida a arquitetura de referência desenhada com base nos serviços fornecidos pelo Windows Azure.

4

Tablet Ecosystem Application

Antes de iniciar a descrição da arquitetura e explicar o seu funcionamento, é importante primeiro contextualizar os objetivos para o seu funcionamento. Tal como tem vindo a ser descrito ao longo dos capítulos anteriores, pretende-se conceber uma arquitetura que suporte um sistema móvel que servirá para auxiliar os gestores bancários nas suas operações e com isto levar estas mesmas funções para fora das agências bancárias.

No entanto, os sistemas bancários apresentam sistemas e dados já existentes e que têm de certa forma de ser considerados na arquitetura. Em termos de acesso aos dados já existentes podemos ver em duas diferentes perspetivas: ou deverá permitir a migração integral dos dados para a nova estrutura ou deve proporcionar a criação de novos pontos de acesso.

É claro que é difícil um serviço bancário isolar-se dos seus componentes porque não é possível a migração de dados de alguns desses componentes. Foi nesse sentido que foi necessário pensar que forma seria possível continuar a permitir que o banco comunicasse com serviços *on-premise* e ao mesmo tempo que utilizasse serviços *cloud* que ofereçam todas as vantagens descritas anteriormente. Assim, a solução adotada é a de um sistema híbrido que permita trabalhar conjuntamente com os dois paradigmas.

Ao longo deste capítulo será descrito o funcionamento pretendido e qual a solução que se pensa responder às necessidades pretendidas.

4.1. Descrição do Projeto

O principal objetivo desta dissertação passa pela desmaterialização das operações que um gestor de conta executa num balcão físico. Com operações entenda-se abertura de contas bancárias, movimentos bancários (como levantamentos e depósitos), pedidos de crédito, entre outas operações que comumente são efetuadas apenas junto a um balcão. O termo desmaterializar pretende levar estas operações para junto do cliente, nomeadamente, levar o gestor bancário, munido de um dispositivo móvel como um *tablet*, até às empresas ou casas das pessoas que necessitam de efetuar estas operações. O conceito pode parecer estranho numa primeira fase, mas se pensarmos bem, facilmente percebemos que cada vez mais os serviços de um modo geral se aproximam das pessoas, reduzindo o tempo que os clientes têm de despender a tratar das chamadas burocracias. Este foi o mote que conduziu à tentativa de transcrever estes conceitos para uma aplicação móvel. Esta aplicação não se destina a uma única entidade bancária mas sim a um conjunto de entidades que a pretendam usar, suportando assim múltiplas instituições/organizações. Esta prática leva-nos ao conceito de *multi-tenant*. Uma aplicação *multi-tenant* é uma aplicação que partilha os recursos mas separa as instituições, ou "*tenants*". Assim, cada utilizador vê a aplicação da sua maneira. Alguns exemplos disso são o Office 365 e o Outlook.com. Da perspetiva dos fornecedores, os benefícios passam essencialmente pela eficiência operacional e de custos. A mesma versão da aplicação pode lidar como inúmeros clientes, todos por cima do mesmo sistema. Uma aplicação *multi-tenant* deve ter claros alguns objetivos e requisitos entre os quais providenciar manutenção e monitorização. Os benefícios para os utilizadores passam pelo isolamento, disponibilidade, escalabilidade e custos [30].

A utilização deste conceito estende-se também à forma como os dados da aplicação são mantidos. Existe a possibilidade de utilizar a aplicação com múltiplas bases de dados em que cada instituição tem a sua própria base de dados ou então ter apenas uma única base de dados para todas as instituições. A solução adotada foi ter uma base de dados única para todas as entidades bancárias, no entanto os dados encontram-se logicamente separados. Uma única instância irá suportar as diferentes entidades como mostra a Figura 4.1.

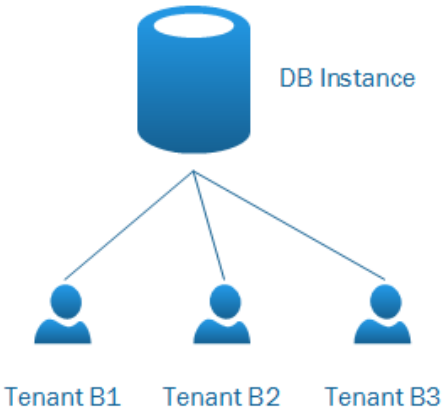

Figura 4.1 - Única instância multi-tenant

Um outro conceito muito utilizado ao longo deste projeto é o conceito de módulo. Tudo é entendido como sendo um módulo que se interliga com os restantes módulos, levando assim a que seja uma arquitetura modularizada, podendo estes módulos estar ou não estar num ambiente SaaS.

Assim, cada entidade bancária presente no sistema é constituída por um conjunto de módulos, em que esses mesmos módulos podem ser serviços na *cloud* ou serviços *on-premise* em que a migração não é possível ou é difícil. Esses módulos são definidos pela própria entidade bancária aquando da primeira utilização do sistema. Podemos entender estes módulos como sendo todo o serviço de gestão de clientes, ou gestão de depósitos entre outros. A entidade bancária tem plena liberdade de configurar estes módulos, ligando-os ou desligando-os e acrescentando novos sempre que considerar necessário.

Fundamentalmente, a arquitetura prende-se com a forma como estes módulos são geridos e acedidos pela aplicação. Quando um banco é registado no sistema, configura módulos onde gere os seus dados, a arquitetura prende-se com a interpretação dessa configuração e por aceder à informação onde quer que ela se encontre, esteja em ambiente SaaS ou *on-premise*. Os módulos de cada entidade bancária e a sua respetiva informação encontram-se armazenados numa base de dados relacional SQL em ambiente *cloud*, ou seja no Azure.

Seguidamente será apresentado o modelo de dados que suporta a informação referente aos módulos e o modelo de dados que suporta a restante informação.

4.2. Modelo de Dados

Foram construídos dois modelos de dados para suportar a arquitetura e consequentemente o protótipo. Um dos modelos de dados pretende armazenar as configurações referentes aos bancos e respetivos módulos e o outro modelo pretende suportar os dados sobre os diferentes módulos. Isto é, dados referentes a entidades bancárias, clientes, depósitos, créditos entre outros dados. Foi também referido que os modelos de dados são *multi-tenant* suportando assim várias entidades bancárias no mesmo modelo. Os modelos de dados encontram-se explicados mais especificamente no Anexo A.

4.2.1. Modelo de Dados de suporte à configuração dos módulos

O modelo de dados é referente à configuração dos módulos das diferentes entidades bancárias. Este modelo de dados encontra-se no SQL Azure.

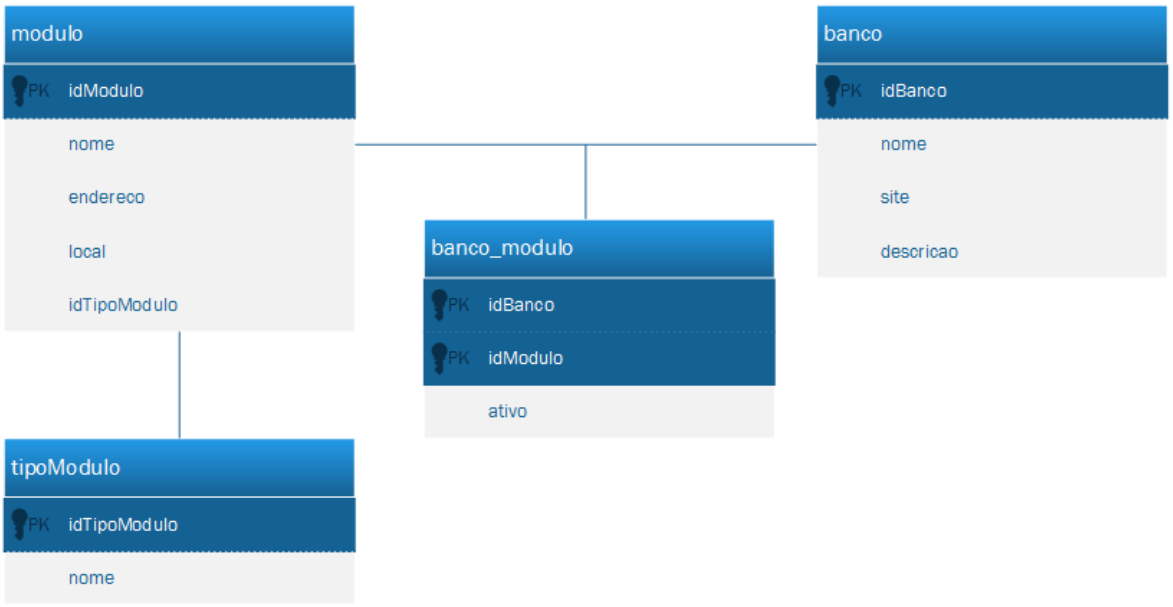

Figura 4.2 - Modelo de Dados – Módulos

Como podemos observar através da Figura 4.2, o modelo é composto por quatro entidades. Estas quatro entidades permitem guardar e relacionar os dados das entidades bancárias com os dados dos módulos que cada entidade detém.

4.2.2. Modelo de Dados de Suporte à aplicação

Este modelo de dados é o responsável pelo funcionamento do sistema. É aqui que ficam armazenados os dados referentes aos bancos, clientes, movimentos, contas, entre outros. Este modelo de dados encontra-se num servidor *on-premise*.

Como podemos observar através da Figura 4.3, o modelo é composto por oito entidades. Estas oito entidades permitem guardar e relacionar os dados das entidades bancárias com os diferentes módulos configurados.

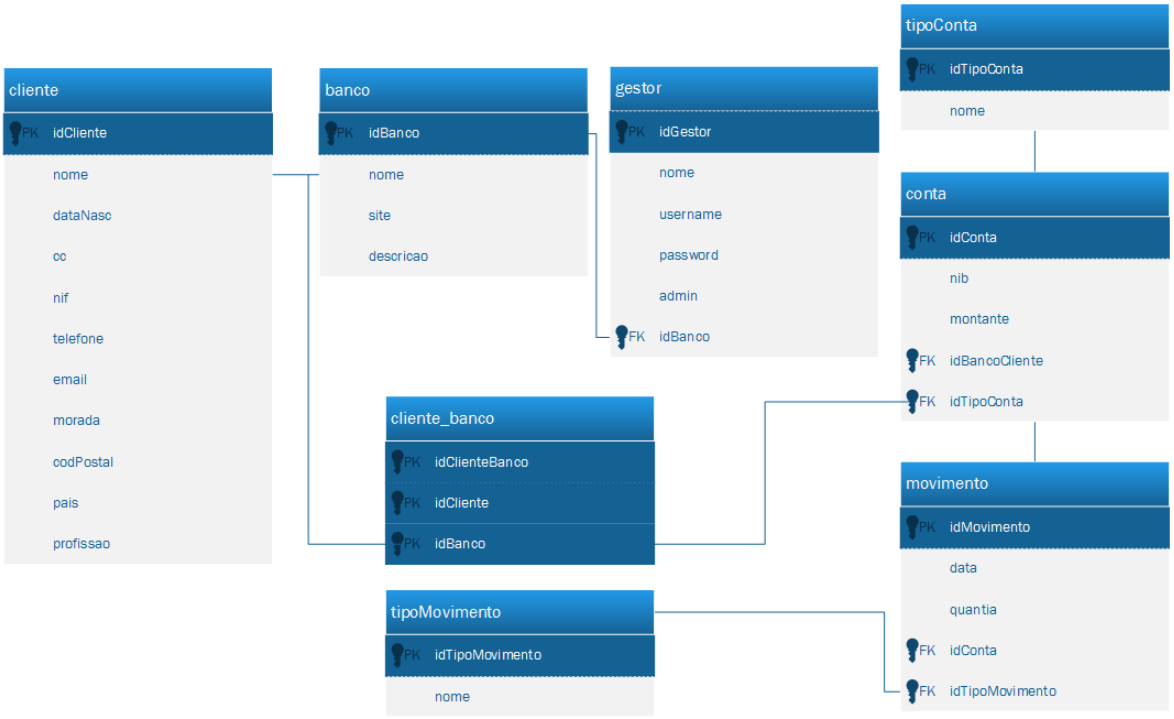

Figura 4.3 - Modelo de Dados - Aplicação

4.3. Módulos e Serviços Utilizados

Torna-se necessário descrever com mais detalhe os serviços do Azure que foram utilizados para a construção da arquitetura. Os serviços utilizados passam pelas funcionalidades de armazenamento, serviços móveis e serviços de barramento. Seguidamente serão apresentados os serviços com mais algum detalhe e o porquê da sua utilização.

4.3.1. Armazenamento – Azure SQL

Para este caso específico, foi utilizada uma base de dados Azure SQL. Esta torna possível configurar, monitorizar e escalar a base de dados caso necessário. A base de dados Azure SQL é útil para aplicações de negócios, serviços baseados em *cloud* ou soluções híbridas. É possível o compartilhamento de dados entre bases de dados SQL ou entre uma instância local de SQL Server e uma base de dados SQL Azure [31].

Dadas as características apresentadas da base de dados Azure SQL a escolha deveu-se ao facto de ser uma base de dados relacional, que permite a comunicação com outras instâncias SQL fora do Azure e ainda que tem a flexibilidade de crescer sempre que necessário pois, as entidades bancárias gerem grandes volumes de dados.

4.3.2. Serviços Móveis

Os serviços móveis são um serviço do Azure que tem como objetivo construir e hospedar *backends* para qualquer aplicação móvel, seja iOS, Android, Windows ou HTML5. Permite a incorporação de notificações *push*, integração social com as principais redes sociais e é altamente escalável. O código em *backend* pode ser C# ou Node.js e permite autenticação *Single Sing-On*.

O armazenamento pode ser feito através de SQL, Oracle, SAP, MongoDB entre outros. Para este caso específico foi utilizado o armazenamento em SQL. Quando o serviço móvel é criado, este pode ser associado à base de dados SQL dentro do Azure. O *backend* fornece suporte integrado que permite que aplicações remotas possam armazenar e recuperar

dados a partir dele – usa um formato OData baseado em JSON – de forma segura. Tal como todos os restantes serviços do Azure, este também tem a possibilidade de ser escalado conforme as necessidades [32].

Estes serviços são peças fundamentais para a arquitetura. Dado tratar-se de uma arquitetura para dispositivos móveis, nomeadamente *tablets*, estes permitem conceber aplicações que correm em qualquer dispositivo, sendo portanto *cross-plataform*, tal como se pretende, e por isso optou-se por construir utilizando HTML5 e Javascript. O *backend* é bastante intuitivo e de forma simples é possível configurar um conjunto de funções. Permite definir uma API personalizada onde se registam métodos HTTP como GET, POST, PUT, PATCH e DELETE. Cada função é definida para cada método e nele o que fazer com o resultado. Para além de muitas funcionalidades que os serviços móveis fornecem, para este projeto foi fundamentalmente usado para métodos HTTP.

4.3.3. Barramento de Serviço

O barramento de serviço é um sistema de mensagens baseado na *cloud* que permite ligar aplicações, serviços e dispositivos em qualquer lugar. Com o barramento de serviço é possível ligar aplicações executadas localmente com aplicações executadas no Azure. A integração de recursos como o SQL, armazenamento com mensagens de barramento assegura uma operação suave em operações de carga pesada e com variabilidade ao longo do tempo. Suporta ainda uma gama de protocolos padrão como o REST, AMQP, WS* e API.

A função de reencaminhamento do barramento permite resolver os problemas da comunicação entre aplicações locais e o mundo exterior, pois permite que os serviços *Web* locais projetem *endpoints* públicos. Assim, os sistemas podem aceder a estes serviços *Web* que continuam a ser executados localmente [33].

Tal como indicam as suas características, este serviço foi particularmente importante pois permite resolver os problemas de comunicação entre os serviços *Web* construídos mas permitir também a comunicação com os sistemas legados que as instituições bancárias têm

e dos quais não se podem desligar. Assim, os serviços são todos publicados em *endpoints* públicos onde podem ser facilmente acedidos.

4.3.4. Serviço de Controlo de Acesso

O serviço de controlo de acesso é um serviço Azure que oferece uma maneira fácil para que seja possível a autenticação dos clientes nas aplicações e serviços Web sem ter de adicionar lógica de autenticação às aplicações ou serviços. O Serviço de Controlo de Acesso (ACS) disponibiliza os seguintes serviços: [34]

- Integração com o Windows Identity Foundation;
- Suporte para provedores *Web* tais como Microsoft, Google, Yahoo e Facebook;
- Suporte para Serviços de Federação do Active Directory
- Portal de gestão que permite o acesso ACS.

Este serviço permite a autenticação dos serviços aquando da sua utilização. Apenas com autenticação é possível aceder aos serviços e receber o conteúdo pretendido.

O *Service Bus* usa um modelo de segurança baseado em declarações implementado através de um serviço de controlo de acesso. O serviço precisa de se autenticar através do serviço de controlo de acesso e adquire um *token* com uma declaração a fim de ser capaz de abrir um canal no *Service Bus*. Quando o serviço e o cliente estão configurados para usar o tipo de autenticação *RelayAccessToken*, o cliente precisa de adquirir um *token* de segurança do serviço de controlo de acesso que contenha uma declaração de Envio (*Send*). Ao enviar o pedido para o *Service Bus* o cliente precisa de incluir o *token* na secção de *header* do pedido SOAP. Por sua vez o *Service Bus* irá validar o *token* de segurança e em seguida, envia a mensagem pretendida [35].

4.4. Descrição da Arquitetura

De forma superficial, a **Figura 4.4** mostra o funcionamento do sistema através do *workflow* do processo de utilização da aplicação móvel. Para uma melhor compreensão da arquitetura, o diagrama foi dividido em três partes.

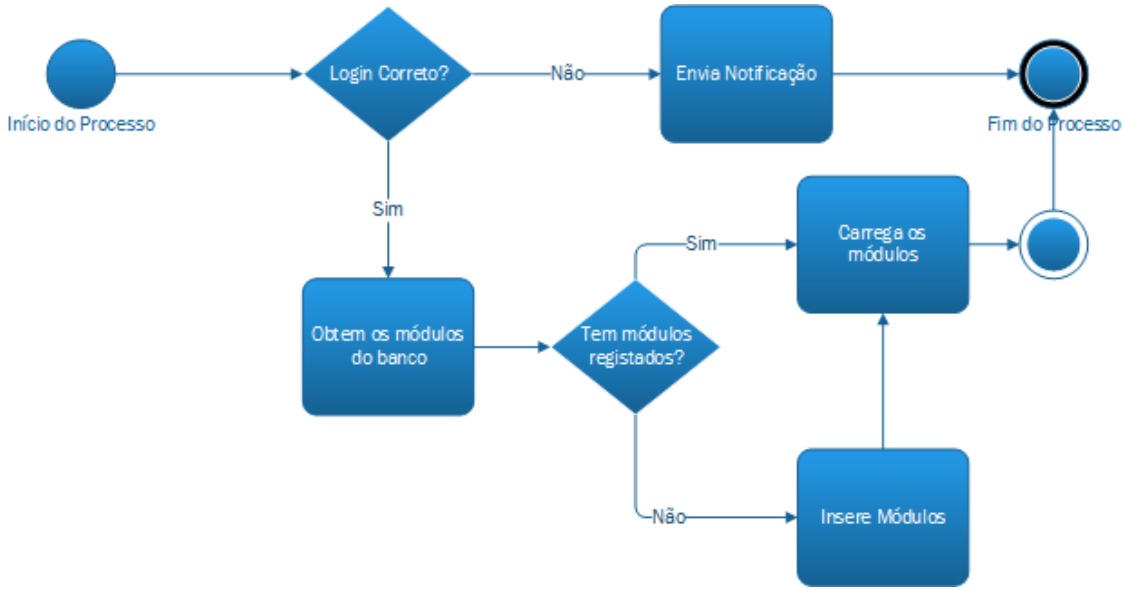

Figura 4.4 - Workflow de funcionamento do sistema

O processo inicia-se com a autenticação na aplicação. Na autenticação temos dois cenários possíveis. Ou o login se encontra errado e o gestor é notificado, terminando assim o processo, ou temos o caso do *login* estar correto. No caso de o login estar correto, a aplicação vai pedir todos os módulos da entidade bancária do gestor que tiver efetuado o login. No caso da entidade bancária em causa já ter módulos registados, os módulos irão ser carregados para o sistema de forma a puderem ser acedidos. Caso contrário, se a entidade bancária não tiver módulos registados ou se estes se encontrarem desativados, o sistema irá reencaminhar o gestor, caso este tenham permissões de administração, para o ecrã de registo de módulos.

Vamos agora perceber mais concretamente como é que o funciona o processo de obtenção de módulos após a autenticação.

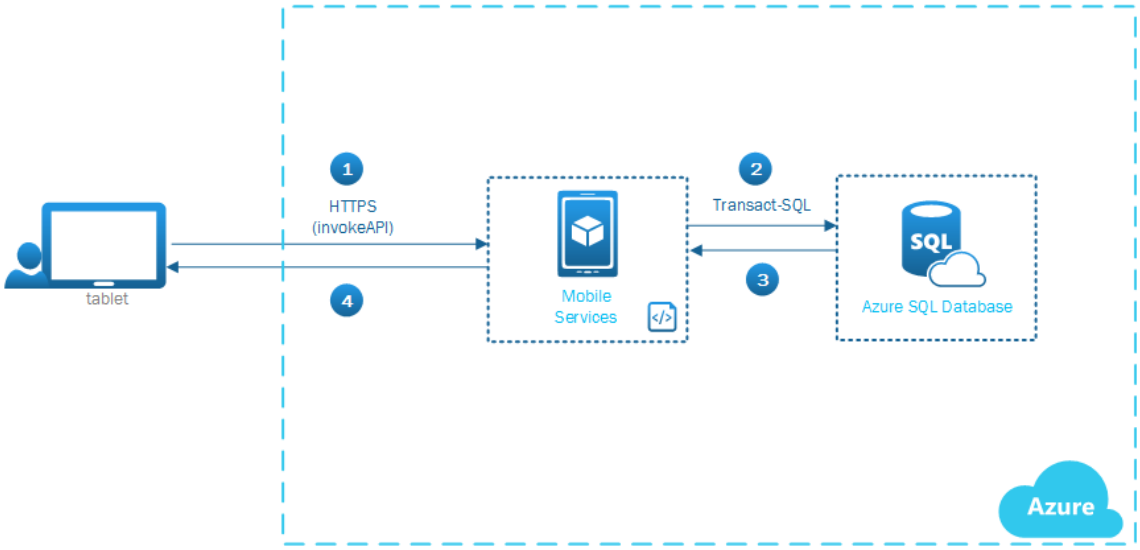

Figura 4.5 - Obter Módulos de um Banco

A **Figura 4.5** ilustra o processo de obtenção dos módulos de um determinado banco. Após o gestor ter efetuado a autenticação no sistema, é feito um pedido para obter os módulos que essa entidade registou no sistema. Tal como referido anteriormente, a base de dados que armazena as configurações dos módulos de uma entidade bancária encontra-se no Azure. Vamos perceber os passos:

1. A aplicação envia um pedido para a API do Windows Azure Mobile Service através de HTTPS. Através do método *invokeAPI* exposto pelo *Mobile Service* chama o serviço. O serviço contém um conjunto de métodos;
2. Esses métodos chamam *Stored Procedures* SQL armazenados na base de dados Azure SQL. Estas chamadas são efetuadas através do objeto mssql, que permite executar declarações Transact-SQL que chamam os *stored procedures*;
3. Os *stored procedures* enviam os resultados do pedido efetuado;
4. O resultado é devolvido para a aplicação a fim de ser processado.

Todos os processos que se refiram aos módulos de uma entidade bancária e às suas respetivas configurações encontram-se inteiramente na *cloud*, não havendo referência a sistemas locais. Esta decisão foi tomada tendo em consideração que dado que é um dos primeiros passos e o mais fundamental para o funcionamento do sistema, é importante que esteja sempre disponível e que seja escalável dado que a quantidade de módulos pode crescer rapidamente. Bem como o processo de gestão de módulos, também o sistema de autenticação se encontra inteiramente na *cloud*.

Tal como também já foi mencionado anteriormente, quando uma entidade bancária regista um módulo, este pode encontrar-se ou na *cloud*, através de serviços móveis do Azure, ou então pode dar-se o caso de ser um serviço *on-premise*. Para ambos os casos é necessário o acesso aos dados de modo a realizar as operações desejadas pelo utilizador.

Vamos agora analisar o cenário em que a aplicação pretende aceder a um módulo que se encontra na *cloud*, através de um serviço móvel do Azure.

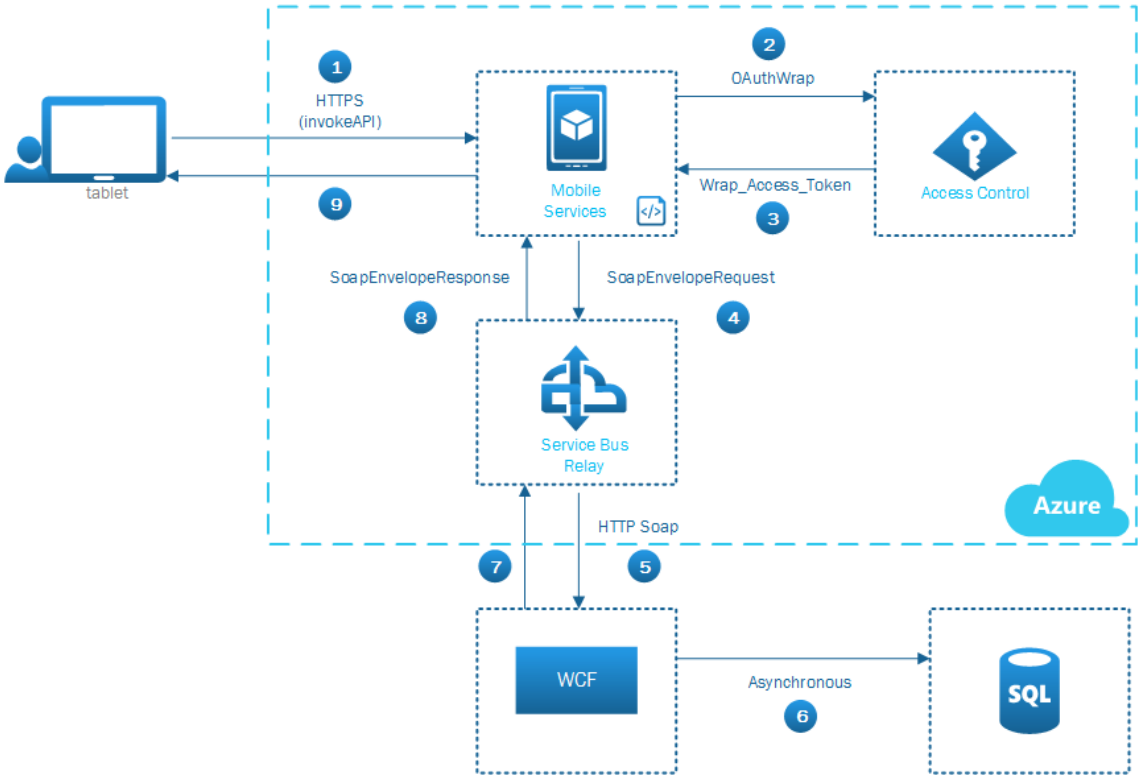

Figura 4.6 - Cenário de Acesso a um módulo na cloud

A Figura 4.6 retrata o cenário de acesso a um módulo de uma entidade bancária que se encontre na *cloud*, mais especificamente num serviço móvel. Sucintamente, o gestor de conta de uma entidade bancária envia para a aplicação pedidos de operações, mas a aplicação em vez de aceder diretamente à base de dados e devolver os resultados pretendidos, invoca uma linha de negócio que está a executar num servidor. A linha de negócios usa uma camada WCF para expor as suas funcionalidades via SOAP para aplicações externas. Mais especificamente, o WFC usa um *endpoint BasicHttpRelayBinding* para expor as suas funcionalidades através do *Service Bus*.

De forma a facilitar a compreensão da arquitetura, vamos analisar todos os passos processados desde o pedido até a obtenção de resultados [35].

1. A aplicação invoca a API presente no serviço móvel. Essa mesma API contém métodos que permitem interagir com os dados;
2. A API, por sua vez, envia um pedido ao serviço de controlo de acesso para adquirir o *token* necessário para se autenticar no *Service Bus*. É usado o protocolo de autenticação OAuth;
3. O serviço de controlo de acesso retorna um *token* de segurança;
4. O serviço móvel usa a API definida para extrair o *wrap_access_token* enviado do serviço de controlo de acesso. A API usa diferentes funções de acordo com o pedido do cliente. Cada uma das funções cria um envelope SOAP que invoca um serviço WCF. No *header* contém o *token* de acesso num formato base 64. O body contêm o resultado de cada chamada. É usado o módulo node-uuid para gerar um id único do *token* de segurança de cada chamada. Usa o módulo HTTPS do Node.js para enviar o envelope soap para o *Service Bus*;
5. O *Service Bus* valida e retira o *token* de segurança. Depois encaminha o pedido para um dos *listeners* do serviço WCF;
6. O serviço WCF usa programação assíncrona para aceder aos dados que se encontram numa base de dados local;

7. Por sua vez, o serviço WCF retorna a resposta para o *Service Bus*;
8. O *Service Bus* reencaminha a mensagem para o serviço móvel. A API usa o módulo xml2js para alterar o formato da resposta SOAP de XML para JSON. Depois, nivela o objeto JSON eliminando *arrays* desnecessários. Extraí os dados de representação SOAP e cria uma resposta em JSON;
9. O serviço móvel retorna os dados em JSON para a aplicação cliente;

Esta é a linha de negócio do acesso à aplicação quando se tenta aceder a um módulo que se encontra na *cloud*, através de um serviço móvel.

No entanto, tal como já foi referido, também existe o caso de os módulos registados pela entidade bancária não se encontrarem num serviço móvel mas sim num serviço *on-premise*. Nesse caso, a arquitetura é ligeiramente diferente para conseguir suportar esta alternativa. Vamos agora analisar o cenário de acesso a um módulo que se encontre num serviço *on-premise* num outro servidor.

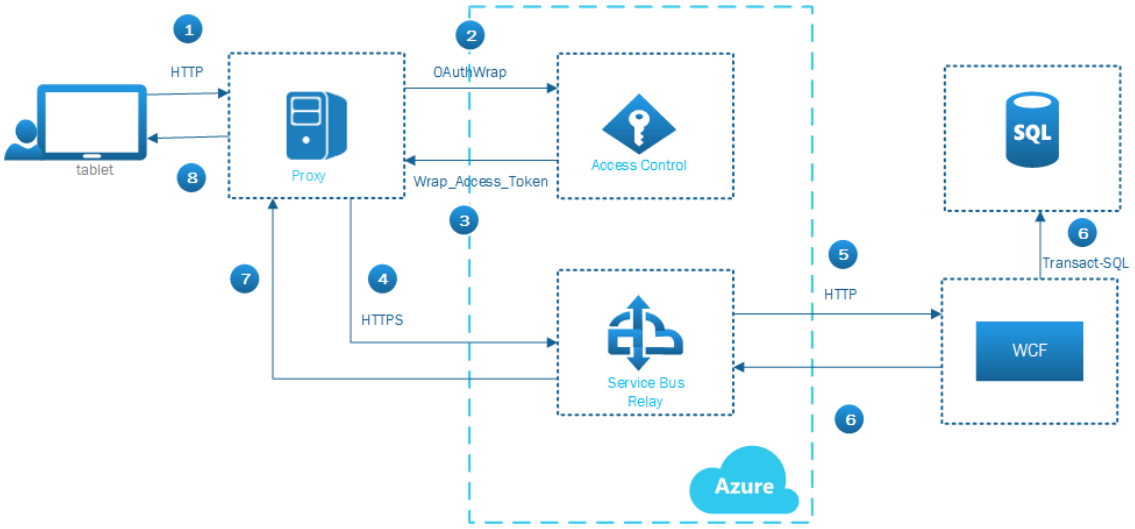

Figura 4.7 – Cenário de acesso a um módulo local

Face à solução apresentada anteriormente, a diferença para esta, retratada na Figura 4.7, é o facto de esta não usar os serviços móveis mas apenas serviços WCF. Contudo esta solução implica que sejam implementados serviços *Web* em *datacenters*. Estes serão posteriormente acedidos através do *Service Bus Relay* tal como a arquitetura tenta representar. Sucintamente, o serviço ao qual se pretende aceder é exposto no *Service Bus Relay*, tal que o gestor quando faz o pedido, este é primeiramente encaminhado para um *proxy* e é aqui que é efetuado o pedido ao *Service Bus Relay*. A fim de obter o *token* de segurança para puder aceder aos dados, o *proxy* envia um pedido ao *Access Control*. Já com o *token* de segurança é possível aceder ao *Service Bus Relay* onde o serviço se encontra exposto e que por sua vez, é este que acede à base de dados a fim de recolher os dados pretendidos enviando tudo de volta para o cliente.

Vamos analisar mais detalhadamente os vários passos realizados:

1. A fim de efetuar o seu pedido, o cliente invoca o *proxy* enviando-lhe o endereço ao qual pretende aceder, o método que pretende invocar e os parâmetros de entrada;
2. No *proxy*, é enviado um pedido ao serviço de controlo de acesso para adquirir o *token* necessário para se autenticar no *Service Bus Relay*. É usado o protocolo de autenticação OAuth WRAP;
3. O serviço de controlo de acesso retorna um *token* de segurança;
4. Já com o *token* de segurança, é feito o acesso ao *Service Bus Relay* que posteriormente acede ao WCF;
5. Já com acesso ao WCF, este por sua vez acede à base de dados SQL que se encontra num servidor local;
6. Os dados são retornados de novo para o *Service Bus Relay*;
7. O resultado do pedido é enviado de novo para o *proxy*;
8. O *proxy* retorna os resultados para o gestor a fim de estes poderem ser processados e mostrados devidamente.

Através das arquiteturas mostradas anteriormente, podemos ver que seja qual for o tipo de módulo que a entidade bancária registar, é possível através de uma solução híbrida aceder a esses mesmos módulos. O *Service Bus Relay* assume um papel importante ao permitir expor serviços externos através da *cloud* e assim facilitar o acesso ao mesmo. O componente de serviços móveis mostrou-se também uma boa revelação, dada a sua capacidade de *backend*.

Na Figura 4.8 é possível ver os componentes que constituem a arquitetura global.

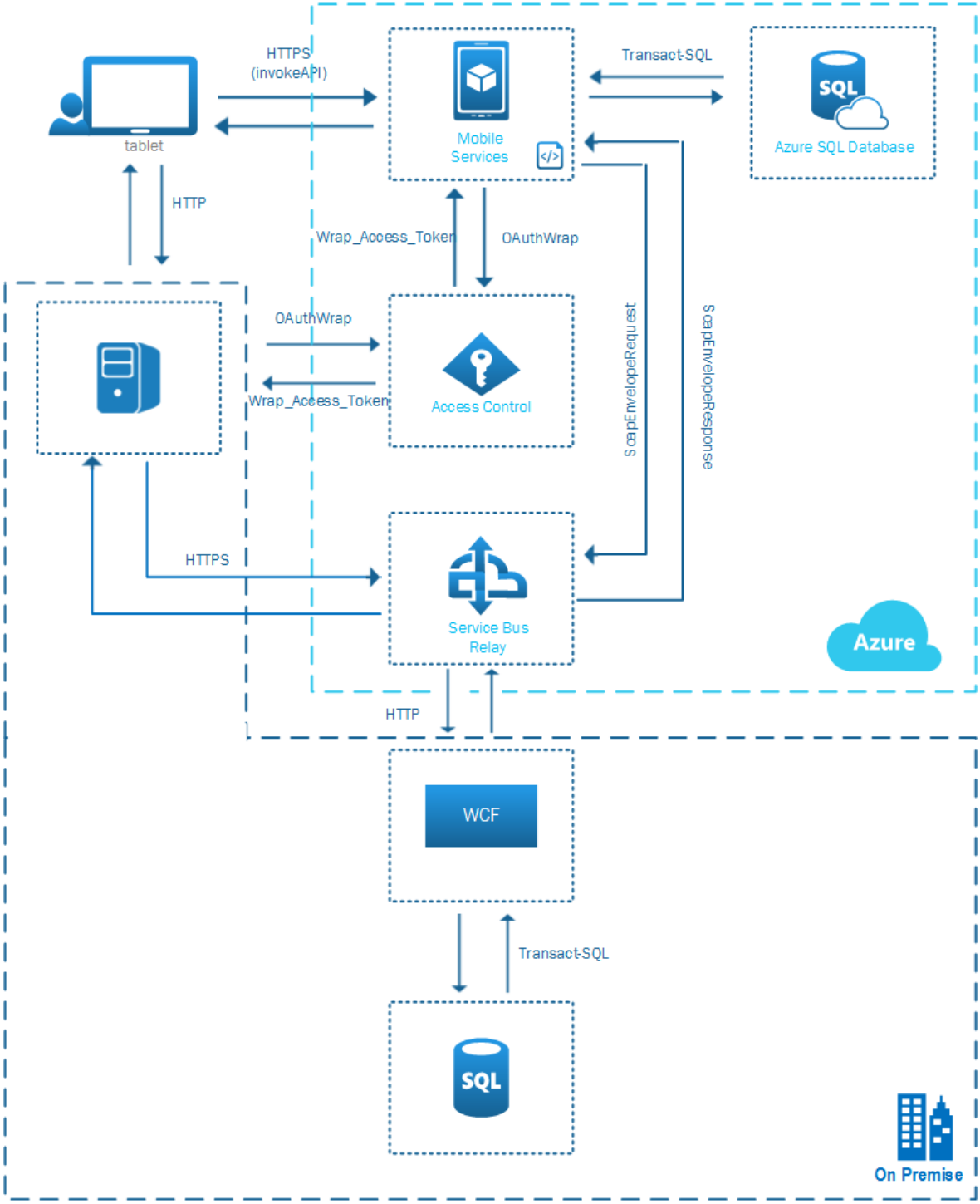

Figura 4.8 - Arquitetura completa

No próximo capítulo é demonstrado o protótipo desenvolvido tendo em conta as arquiteturas acima representadas.

5

Protótipo

Um dos objetivos deste projeto passava pela construção de um protótipo que apoiasse a arquitetura desenvolvida. Assim, o protótipo pretende dar uma visão do que poderia ser a aplicação móvel suportada por esta arquitetura. Tanto os modelos de dados como o protótipo em si são simplistas pois são apenas a título exemplificativo. Para tal, foram desenvolvidos dois módulos: Clientes e Movimentos. Estes módulos foram construídos tanto na *cloud* como num servidor local. Tanto os modelos de dados como as funcionalidades foram construídos a título exemplificativo para demonstrar um possível funcionamento.

Numa primeira fase vão ser apresentadas as tecnologias usadas no desenvolvimento, seguidamente serão apresentados os módulos utilizados também na implementação. De seguida serão mostrados os serviços desenvolvidos tanto na *cloud* como localmente para apoiar este protótipo e por fim será mostrado o mesmo e apresentadas as suas funcionalidades.

5.1. Tecnologias

Para o desenvolvimento deste protótipo, foram usadas várias tecnologias e ferramentas, algumas já referidas anteriormente aquando da descrição dos módulos utilizados do Azure. Para além das tecnologias utilizadas do Azure e já descritas, existem outras ainda não referidas como é o caso:

5.1.1. HTML5

O HTML5 é uma linguagem estruturada e permite apresentar conteúdo para a *World Wide Web*. É o elemento chave da utilização da Internet e o número cinco indica a quinta versão. Este permitiu a construção das páginas onde o cliente pode navegar e utilizar a aplicação. Foi também utilizado o local *storage* do HTML5 para armazenar várias informações para o funcionamento da aplicação [36] [37].

5.1.2. Javascript

O javascript é uma linguagem de programação multi-paradigma. É uma das principais linguagens *client-side* dos *browsers*. Permitiu definir as várias funções da interação do cliente com a aplicação [38].

5.1.3. Jquery / Jquery Mobile

O jQuery é uma biblioteca do javascript *cross-browser* e que permite auxiliar nos *scripts* do lado do cliente que interagem com o HMTL. Foi utilizado para manipulação e controlo dos dados inseridos pelos utilizadores. Já o jQuery Mobile é uma *framework* baseada em HTML5 e jQuery que permite desenvolver aplicações Web *responsive* que funcionam nas plataformas mais conhecidas como *desktop*, portáteis, *tablets* e *smartphones* [39].

5.2. Módulos

Ao longo do desenvolvimento do protótipo, foram utilizados alguns módulos de forma a permitir agilizar o sistema e simplificar alguns passos. Seguidamente, serão apresentados quais os principais objetivos dos módulos. À exceção do OAuth WRAP e do CryptoJS, todos os restantes módulos pertencem ao Node.js.

5.2.1. OAuth WRAP

O OAuth WRAP (*Resource Web Authentication Protocol*) é uma tentativa de redefinir a maneira como os serviços *Web* acedem a outros serviços para ver ou editar conteúdo em nome de um utilizador. Quando as aplicações e serviços lidam com a autenticação através do Serviço de Controlo de Acesso, o cliente deve obter um *token* de segurança emitido pelo ACS para fazer *login* na sua aplicação ou serviço. A fim de obter o *token*, o cliente deve autenticar-se diretamente ao ACS ou enviar um *token* de segurança emitido pelo provedor de identidade. O ACS valida o *token* em questão, processa as declarações de identidade neste símbolo através do mecanismo de regras ACS, calcula as declarações de identidade de saída e emite um *token* de segurança de saída.

Todas as solicitações de *tokens* através deste protocolo são transmitidas através de SSL. O ACS emite sempre um *token* Web simples (SWT) através do protocolo como resposta a um pedido de *token* corretamente formado. Todas as solicitações de *token* através deste protocolo são enviadas para o ACS num pedido HTTP POST.

Existem vários métodos de pedir um *token* ao ACS através do protocolo: Pedido *Password Token*, Pedido SWT *token* e ainda o pedido SAML token. Neste projeto, o método utilizado foi o primeiro, pois, o cliente detém a *password* e o *username* que permite fazer pedido diretamente ao ACS via protocolo. Esta é a forma mais fácil de usar este protocolo, pois para além de estabelecer uma conexão SSL, este método não requer capacidade criptográfica. Este tipo de método tem a forma HTTPS POST. Os parâmetros de entrada são: *wrap_scope*, *wrap_name* e *wrap_password* [40].

No seguinte exemplo podemos ver um exemplo de um pedido feito para o *namespace* “teamservicebus”:

```
POST /WRAPv0.9/ HTTP/1.1
Host: teamservicebus.accesscontrol.windows.net
Content-Type: application/x-www-form-urlencoded

wrap_scope=http%3A%2F%2Fteamservicebus.com%2Fservices%2F&
```

```
wrap_name=owner&
wrap_password=5znwNTZDYC39dqhFOTDtnaikd1hiuRa4XaAj3Y9kJhQ%3D
```

Este módulo tanto é usado nos serviços móveis como no lado do cliente para que este possa aceder aos serviços publicados no *Service Bus Relay*.

5.2.2. Node UUID

O Node UUID é um módulo de Node.js que permite gerar identificadores únicos. Neste caso específico, permite gerar identificadores únicos para o *token* de segurança a cada chamada do serviço. UUID significa *Universal Unique Identifier* e é um padrão de identificadores gerados e padronizados pela Open Software Foundation (OSF) [41].

O objetivo do UUID é ser capaz de gerar identificadores únicos em sistemas distribuídos sem ponto de coordenação central. Um UUID tem a seguinte forma xxxxxxxx-xxxx-Mxxx-Nxxx-xxxxxxxxxxxx em que o N indica a variante e o M indica a versão do UUID. O módulo é usado da seguinte forma:

```
var uuid = require('node-uuid');
var uuid1 = uuid.v1();
```

O resultado terá o seguinte formato:

```
uuid1 -> ec654ec1-7f8f-11e3-ae96-b385f4bc450c
```

5.2.3. HTTPS

O módulo HTTPS é também um módulo pertencente ao Node.js. O HTTPS é o protocolo HTTP mas sobre TLS/SSL. No contexto do projeto, este é usado para enviar envelopes SOAP para o serviço *Service Bus Relay*. O módulo é usado da seguinte forma: [42]

```
var https = require('https');
var options = {
    host: namespace + '-sb.accesscontrol.windows.net',
    path: '/WRAPv0.9/',
    method: 'POST'
  };
var req = https.request(options, function (res) {
        …
};
```

5.2.4. Xml2js

O módulo xml2js é também um módulo pertencente ao Node.js e que tem como principal objetivo formatar mensagens SOAP de XML para JSON. Isto é, é um *parser* que converte dados em formato XML para o formato JSON. O módulo é usado da seguinte forma: [43]

```
var xml2js = require('xml2js');
var parser = new xml2js.Parser();
parser.parseString(soap, function (error, json) {
        …
};
```

5.2.5. Mssql

O módulo mssql é também um módulo pertencente ao Node.js e tem como principal objetivo conectar com bases de dados MS-SQL. Este módulo usa alguns outros módulos TDS

e oferece uma interface unificada. Neste caso, este módulo serve para aceder à base de dados SQL Azure a partir de um serviço móvel. O módulo é usado da seguinte forma: [44]

```
var mssql = request.service.mssql;
mssql.query(sql, params {       …
};
```

5.2.6. CryptoJS

Este módulo é uma coleção de *standards* e de algoritmos criptográficos de segurança implementados em Javascript que usam um conjunto de boas práticas e padrões. Neste caso específico, o algoritmo usado foi o SHA-2, mais concretamente o SHA256 que é uma das variantes do SHA-2. Este módulo foi utilizado no lado do cliente a fim de codificar as *passwords* de acesso ao sistema tanto no *login* como no registo de novas entidades bancárias. O módulo é usado da seguinte forma: [45]

```
<script src=”http://crypto-
js.googlecode.com/svn/tags/3.1.2/build/rollups/sha256.js”></script>
var hash = Crypto.SHA256(password);
```

Os módulos acima descritos permitem adicionar novas funcionalidades ao projeto. Para além da segurança, outros módulos são usados para simplificar o acesso aos dados. Os módulos do Node.js foram parte importante no desenvolvimento, nomeadamente na construção das APIs dos Serviços Móveis. Dado o *backend* em Javascript, a utilização de módulos do Node.js foi fácil e intuitiva. Alguns destes módulos foram instalados e descarregados através do GitHub.

5.3. Funcionalidades

A aplicação protótipo encontra-se dividida em duas partes: a parte administrativa e a parte de utilização por parte das entidades bancárias, mais concretamente pelos seus gestores de conta. A nível de utilizadores, existem três categorias dentro dos gestores de conta:

- Super-Administrador: responsável pelo registo das entidades bancárias e registo de gestores de conta;
- Administrador: responsável pelo registo dos gestores de conta e respetiva gestão bancária;
- Simples: sem permissões de registo de gestores de conta, apenas de gestão bancária.

5.3.1. Administração

Tal como referido anteriormente, existe um super administrador da aplicação e é este o responsável pelo registo das entidades bancárias para utilização do sistema bem como o registo de gestores de conta. Assim, o administrador ao aceder à aplicação tem a possibilidade de adicionar novos bancos na aplicação (Figura 5.1) e ver todos os bancos registados (Figura 5.2).

Figura 5.1 - Adicionar nova entidade bancária

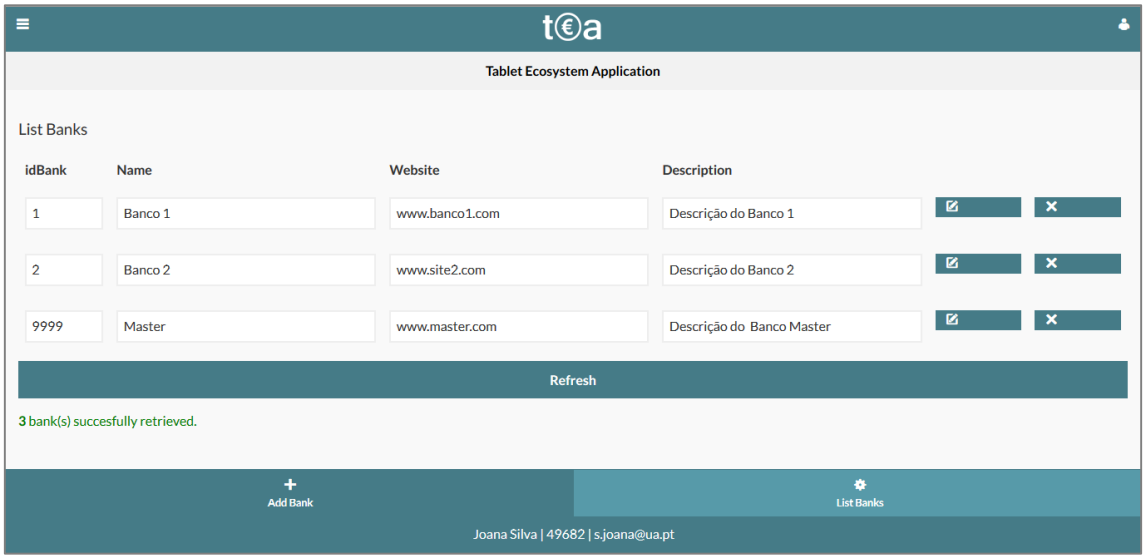

Figura 5.2 - Listagem das entidades bancárias registadas

Como já referido, o super administrador pode ainda registar gestores de conta (**Figura 5.3**) e ver todos os gestores registados (**Figura 5.4**).

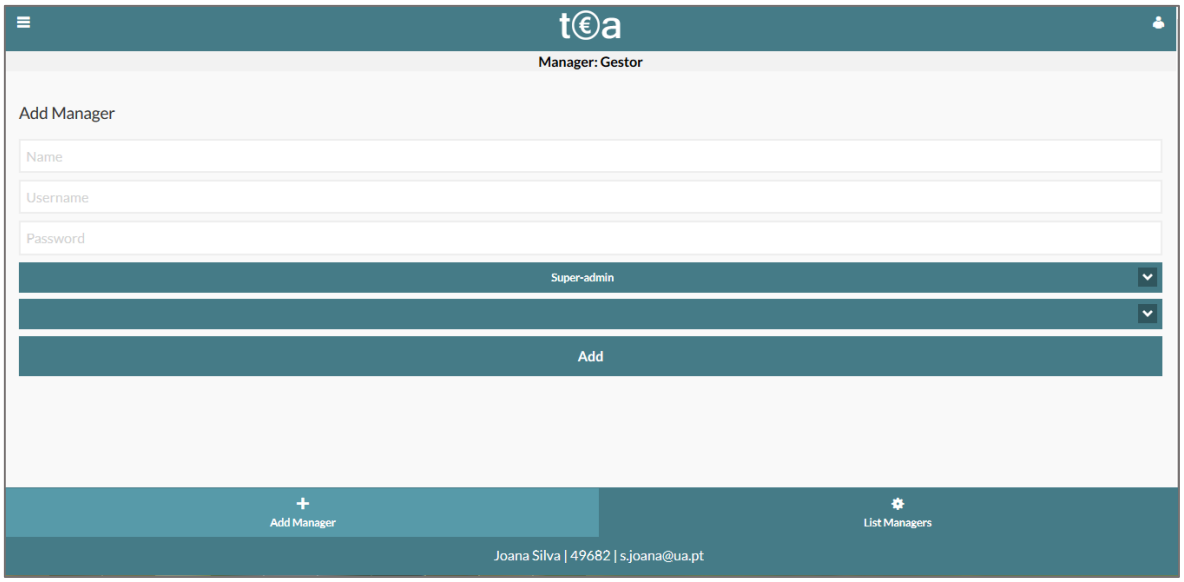

Figura 5.3 - Adicionar novo gestor

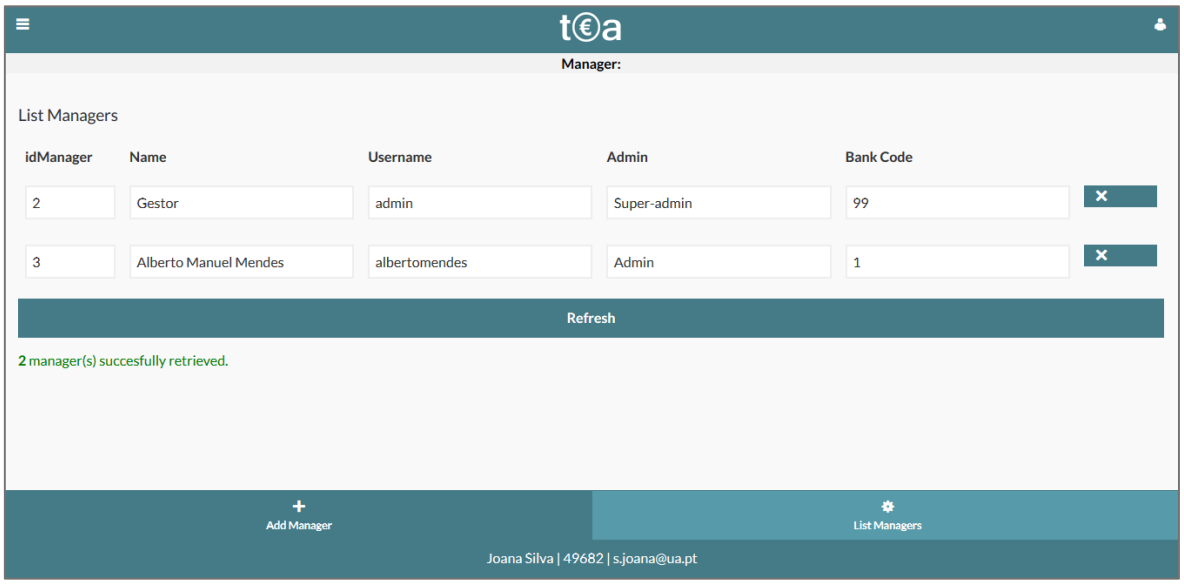

Figura 5.4 - Listagem dos gestores registados

5.3.2. Entidades Bancárias

Tendo em conta os módulos que cada entidade bancária tem registados e ativos, as funcionalidades vão também variando. As funcionalidades descritas fazem parte dos módulos construídos a fim de testar a arquitetura descrita. Todas as entidades bancárias têm um módulo por definição que é o módulo responsável pelos módulos.

5.3.2.1. Módulo – Módulos

Tal como referido, este módulo encontra-se presente ao gestor com permissões de administrador de cada entidade bancária. Quando o gestor de uma entidade bancária faz login pela primeira vez, a aplicação vai verificar quais os módulos dessa entidade. Se verificar que não existe nenhum, é apresentado ao gestor um formulário de inserção de módulos onde este pode inserir os que pretender. Seguidamente, a aplicação irá carregar os módulos e apresentá-los-á no menu que permite o acesso aos mesmos.

Também no menu, está sempre disponível o acesso ao módulo de gestão de módulos. O módulo de gestão de módulos permite:

- Adicionar novos módulos: a qualquer altura, a entidade bancária poderá adicionar novos módulos; (Figura 5.5)

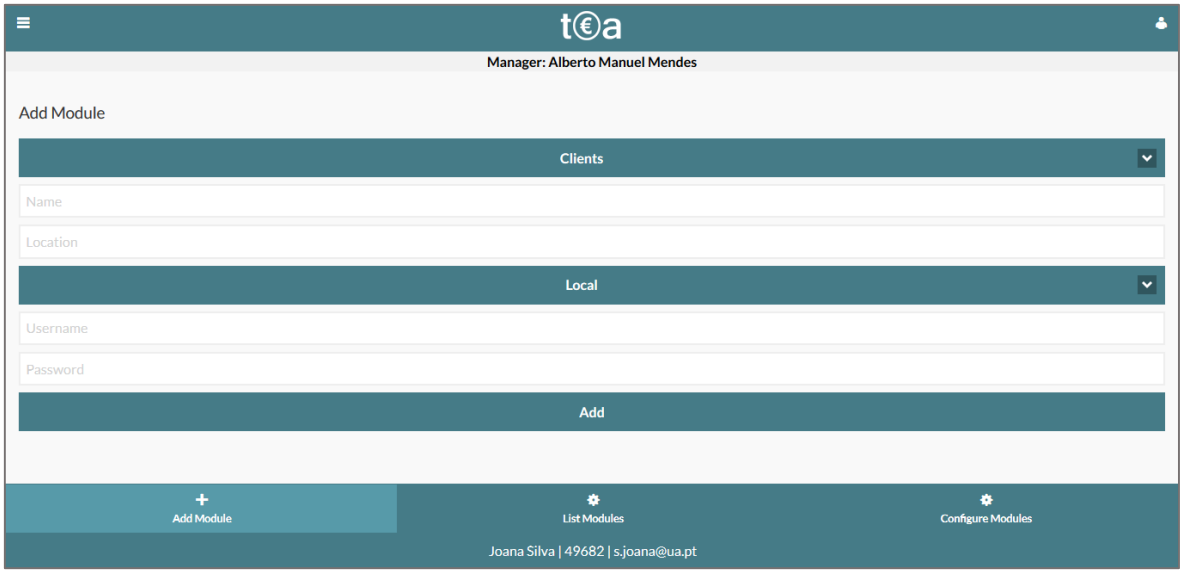

Figura 5.5 - Adicionar novos módulos

- Gestão de módulos: o gestor com permissões de administração tem acesso a todos os seus módulos registados, onde pode alterar dados ou eliminar módulos se necessário; (Figura 5.6)

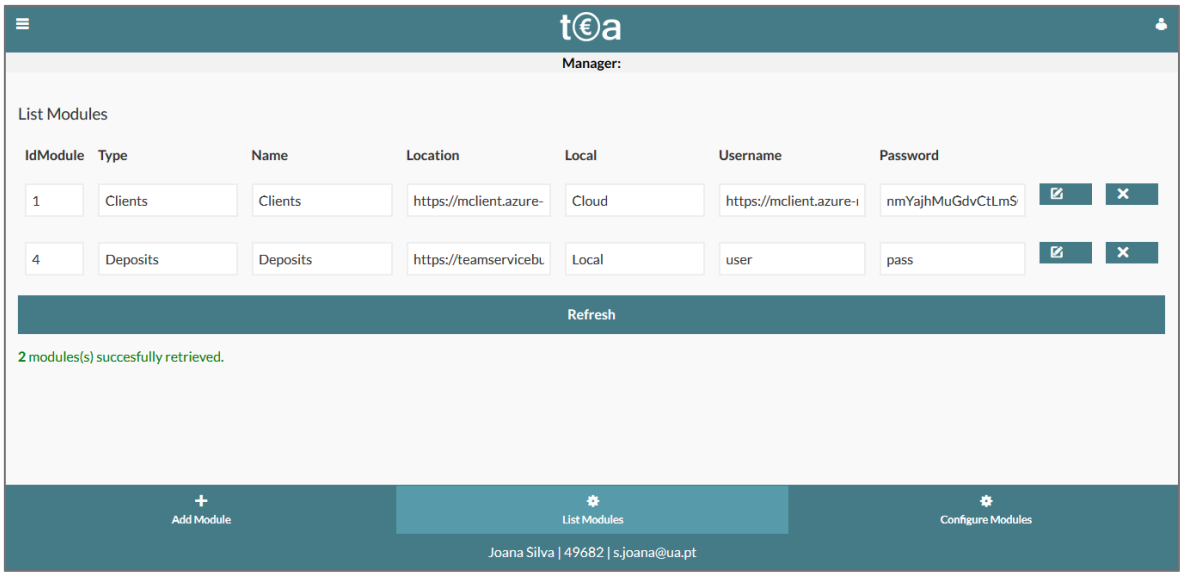

Figura 5.6 - Gestão de módulos

- Ativação/Desativação de módulos: existe ainda a possibilidade de ativar ou desativar módulos. (Figura 5.7)

Figura 5.7 - Ativação/desativação de módulos

5.3.2.2. Módulo – Clientes

O módulo de clientes que foi desenvolvido a fim de testar a arquitetura e detém algumas funcionalidades que permitem a gestão dos clientes das entidades bancárias. Pode ser acedido por qualquer gestor associado a uma entidade bancária.

- Adicionar Clientes: é possível adicionar novos clientes. Ao adicionar clientes, estes são automaticamente associados à entidade bancária. É também criada automaticamente uma conta bancária e é gerado um NIB identificativo; (Figura 5.8)

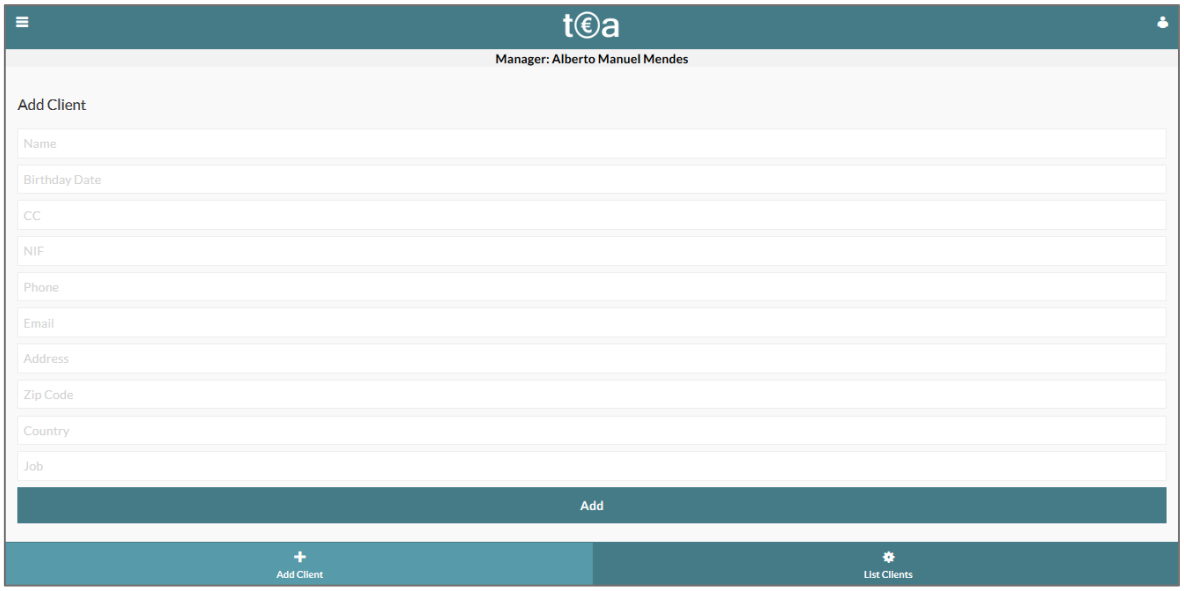

Figura 5.8 - Adicionar clientes

- Gestão de clientes: o gestor de uma entidade bancária pode consultar todos os seus clientes, editar alguns dados ou mesmo removê-los da sua entidade caso necessário. (Figura 5.9)

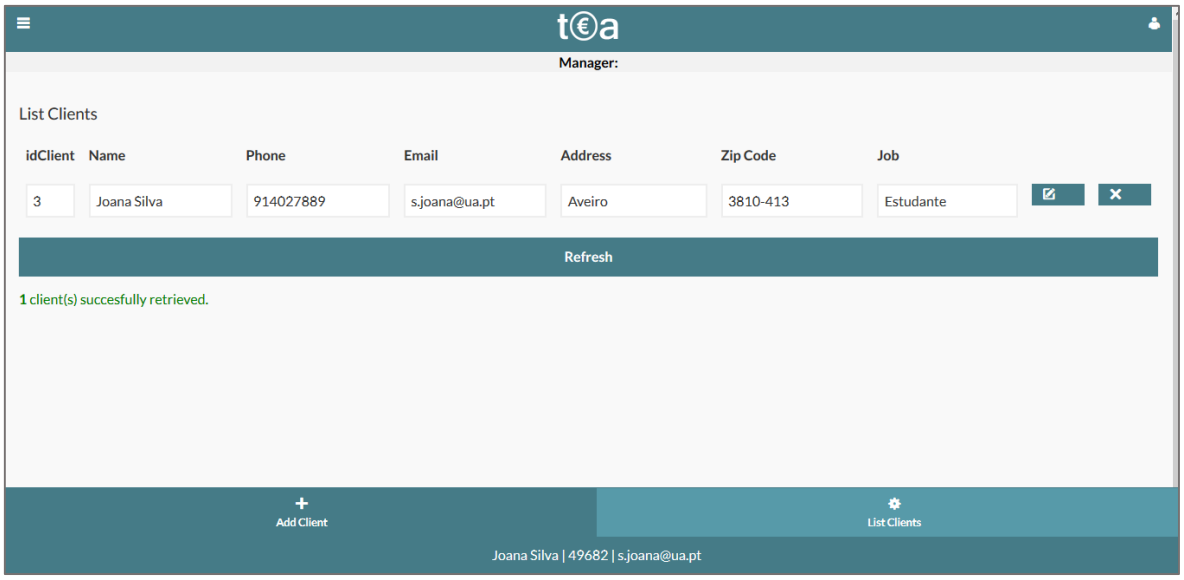

Figura 5.9 - Gestão de clientes

5.3.2.3. Módulo – Movimentos

O módulo de movimentos desenvolvido a fim de testar a arquitetura e detém algumas funcionalidades que permitem aos gestores de conta, a gestão dos movimentos bancários dos clientes das entidades bancárias.

- Ver Saldo: é possível consultar o saldo de um determinado cliente introduzindo apenas o NIB; (Figura 5.10)

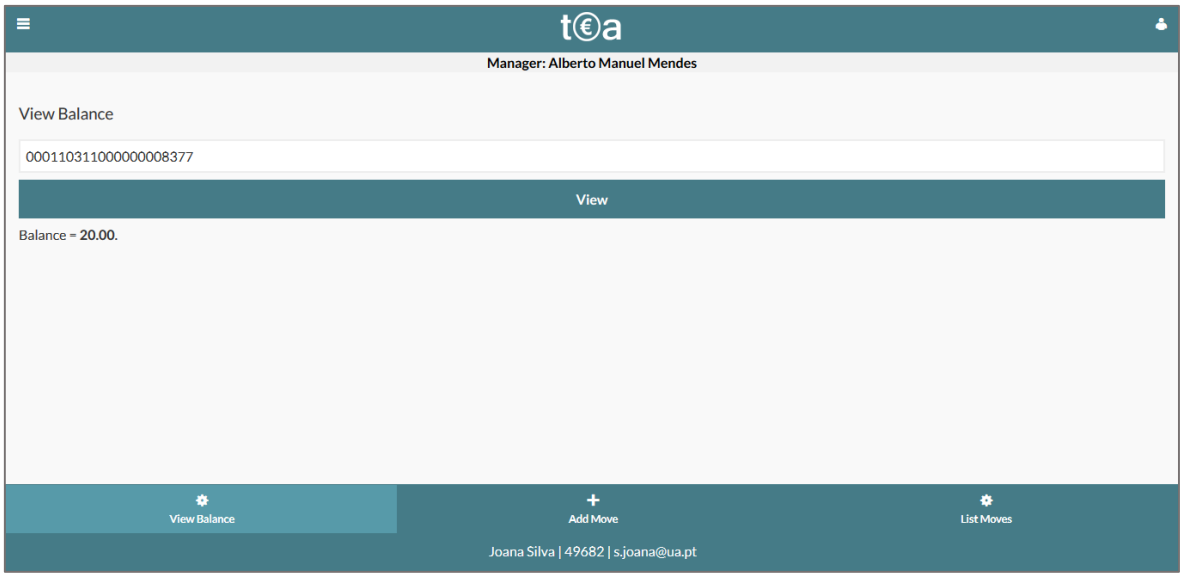

Figura 5.10 - Ver saldo

- Movimentos: o gestor pode registar um depósito ou um levantamento de um determinado montante de dinheiro, introduzindo o NIB e a quantia desejada; (**Figura 5.11**)

Figura 5.11 – Movimentos

- Ver Movimentos: é possível consultar os movimentos bancários de um determinado cliente introduzindo o NIB e um intervalo de datas para a consulta. (**Figura 5.12**)

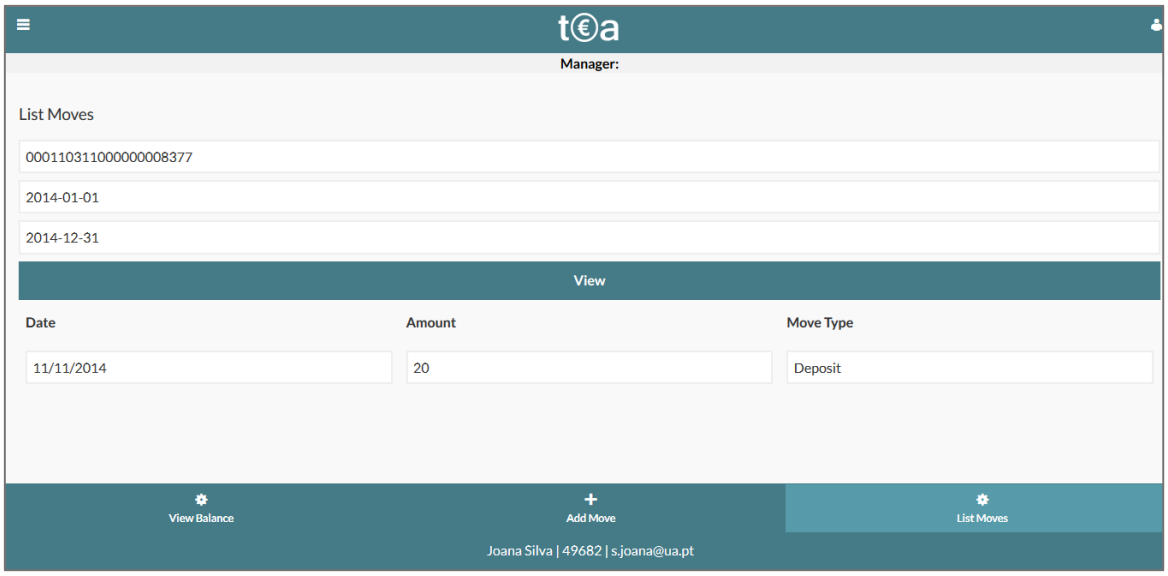

Figura 5.12 - Ver movimentos

Todas estas funcionalidades podem ser vistas através de imagens demonstrativas presentes no Anexo C.

6

Conclusões

Terminada a apresentação e respetiva descrição da arquitetura e protótipo desenvolvido, é altura de retirar algumas conclusões sobre os problemas encontrados, trabalho a desenvolver futuramente e por fim as conclusões gerais.

6.1. Problemas Encontrados

Ao longo da conceção da arquitetura e do seu protótipo, foram vários os problemas encontrados. O primeiro problema prende-se com a migração dos dados dos sistemas bancários para este tipo de sistemas. A migração completa é impossível ou difícil e portanto foi necessário encontrar forma de conseguir conciliar as duas soluções. Assim, a solução passou por conceber uma arquitetura em que, por um lado fosse possível utilizar serviços SaaS mas que também permitisse o acesso a silos de informação contidos em servidores locais das entidades bancárias. A solução modular pareceu a mais indicada dado que permite o acesso a ambos, utilizando o conceito de módulo para separar o acesso à informação.

Também o modelo de dados trouxe alguns problemas. Para além do armazenamento dos dados referentes às entidades bancárias e seus respetivos clientes, entre outros, era necessário o armazenamento das configurações dos módulos pertencentes a cada entidade bancária. Numa primeira fase foi construído um único modelo de dados que suportava toda a informação de forma conjunta. No entanto, depois de alguma análise, achou-se pertinente dividir este modelo. As configurações dos módulos foram transferidas para uma base de dados Azure SQL, dada a sua disponibilidade e escalonamento. Esta separação do modelo partilha um problema dado que ambos os modelos de dados apresentam uma entidade em

comum: o banco. Dado que é imprescindível que os dados desta entidade sejam os mesmos em ambos os modelos, a solução adotada passou pela replicação da informação em ambas as entidades. Esta solução está longe de ser a indicada, pois, o que faria sentido seria uma solução de base de dados distribuída em que a instância pudesse ser partilhada. No entanto, o SQL Server não suporta a distribuição de instâncias e como tal, não foi adotada tal solução.

Já relativamente à conceção do protótipo, existem atualmente duas abordagens para o desenvolvimento de aplicações para *tablets*: através de tecnologia nativa (Objective C para iOS, Java para Android e C# para Windows 8), ou através de tecnologias *Web* compatíveis com *CrossDevice* (HTML5 e Javascript) [46].

Embora a primeira abordagem requeira um maior esforço de desenvolvimento para disponibilizar a mesma solução nos diversos tipos de *tablet*, pois implica a conceção e implementação em todos os tipos de *software* com especificidades tecnológicas únicas; estudos recentes [47] indicam que a tecnologia nativa tem melhorias de performance muito significativas quando comparada com tecnologias *CrossDevice*. Por essa razão, as camadas a implementar serviços nativos do *tablet* e operações com grau de complexidade e processamento elevado deveriam ser efetuadas nestas tecnologias específicas. No entanto, dado que o protótipo foi construído a título exemplificativo, o mesmo foi construído utilizando tecnologias *CrossDevice*, como o HTML5 e Javascript e ainda com ajuda do jQuery Mobile.

Outro dos problemas que surgiu na conceção do protótipo passou pelo acesso aos módulos locais que se encontram expostos através do *service bus relay*. O *service bus* expõe os *endpoints* através de um endereço no entanto a comunicação entre ambos encontrava-se bloqueada por causa de estas se encontrarem em domínios diferentes, surgindo o problema do Cross-Origin Resource Sharing (CORS) [48]. Para ultrapassar este problema foi necessário desenhar um *proxy* que recebesse os pedidos e posteriormente reencaminhasse para o *service bus relay*. Este *proxy* pode ser encarado como um intermediário no acesso aos serviços locais expostos.

Aquando do registo das entidades bancárias no sistema, também houve alguma preocupação relativa a este processo. Havia duas hipóteses: ou cada banco fazia o seu

registo independente através da aplicação, antes de iniciar sessão, ou então haveria um responsável por esse mesmo registo. Optou-se pela segunda opção, pois, se cada banco efetuasse o seu próprio registo, isto poderia ser pouco seguro e conduzir a fragilidades no sistema. Assim, foi criado um super utilizador, com privilégios de inserção de entidades bancárias e gestão das mesmas.

Claramente que ao longo deste trabalho, muitos outros problemas foram surgindo e foram ultrapassados. Foi necessário a tomada de algumas decisões, no entanto tendo sempre em conta, que essas mesmas decisões contribuíram para um melhor desenvolvimento deste mesmo projeto.

6.2. Trabalho Futuro

O trabalho desenvolvido e descrito ao longo desta dissertação tem ainda muito por desenvolver. Tratando-se de uma aplicação com implicações bancárias, é claro que é necessário um processo rigoroso de aplicação dos conceitos aqui apresentados. Torna-se assim portanto evidente que é necessário toda uma camada de segurança na comunicação dentro e fora da arquitetura. Será necessário garantir que os dados não são acessíveis fora da aplicação, garantindo assim total segurança no armazenamento e manuseamento da informação. Também o controlo de acesso por parte das entidades bancárias deverá ser monitorizado, bem como deverão ser implementados mecanismos alternativos de autenticação forte.

Para além das questões de segurança, claramente fundamentais, existem muitos outros fatores que poderiam tornar este projeto um forte canal de comunicação entre as entidades bancárias e os clientes. Alguns dos desafios de melhoria incluem a possibilidade de acesso a dados *offline*. Muitas vezes, principalmente nos países em desenvolvimento, o acesso à rede pode não responder às necessidades, fazendo com que seja perdida a comunicação. Assim, era desejável que fosse possível um cenário de execução de operações *offline*. No entanto, este cenário também apresenta problemas concorrenciais ao nível de alterações feitas pelas entidades bancárias. Seria necessário determinar a precedência das operações e como se poderia desenvolver o processo de reversão à situação anterior.

Outro desafio interessante a acrescentar é a possibilidade de iniciar um processo num *tablet* e depois continuar noutro *device*. Este desafio implica a distribuição de chaves de codificação da informação armazenada.

Muitos são os desafios que podem assentar por cima desta proposta de arquitetura. Trata-se de uma área que apesar de todos os avanços que tem sofrido, é sempre possível levar mais além e com isso reduzir em custos como é o caso da infraestrutura de balcões físicos. Muitas propostas de melhoria podem ser acrescentadas a fim de tornar um sistema sustentável e possível.

6.3. Considerações Finais

O principal objetivo desta dissertação é a conceção de uma arquitetura de apoio à desmaterialização de operações em agências bancárias. Com esta arquitetura pretende-se assegurar atributos tais como integridade conceptual, interoperabilidade, segurança, robustez, fiabilidade, escalabilidade, disponibilidade e usabilidade de plataforma. Para isso, foi necessário considerar vários requisitos tais como o facto de ser concebido para dispositivos móveis e ter em conta os sistemas bancários já existentes e permitir interligar os vários sistemas já existentes com os novos. Para tal, foi analisada a adoção de um modelo *Software as a Service* para a construção de um cenário híbrido no qual existem módulos em modelo SaaS e outro num modelo de serviço tradicional, com ligação ao sistema bancário.

A arquitetura concebida pretende responder às necessidades acima descritas. A adoção de soluções *cloud* pretendeu responder a requisitos como fiabilidade, escalabilidade e disponibilidade. Por outro lado, o componente de comunicação com sistemas legados pretendeu a conexão com os sistemas atuais e com os quais um rotura total não faria qualquer sentido.

O protótipo tentou mostrar uma possível utilização da arquitetura e possíveis funcionalidades que a mesma pode operar. É assim possível que cada entidade bancária adicione e faça a gestão dos seus módulos. Foi possível mostrar a utilização dos dois cenários que a arquitetura suporta e como estes funcionam em simultâneo.

Claramente que a solução apresentada tem ainda muita margem de desenvolvimento e melhorias, no entanto, é uma tentativa de aproximação dos clientes e dos bancos e um grande passo para os mercados emergentes, onde esta poderá ser uma excelente solução para fomentar a bancarização da população.

Bibliografia

[1] Notícias ao Minuto, “Aplicações Bancárias vão Superar Banca Online em cinco anos,” 9 Julho 2014. [Online]. Available: http://www.noticiasaominuto.com/tech/246870/aplicacoes-bancarias-vao-superar-banca-online-em-cinco-anos. [Acedido em Agosto 2014].

[2] B. King, Branch Today, Gone Tomorrow, Marshall Cavendish Business, 2013.

[3] B. King, Bank 3.0: Why Banking Is No Longer Somewhere You Go But Something You Do, 1ª ed., Wiley, 2012.

[4] V. Ribeirinho e J. Silva, Financial Services - Angola Banking Survey, KPMG, 2012.

[5] N. Kshetri e S. Acharya, “Mobile Payments in Emerging Markets,” *IEEE,* pp. 9-13, 25 Julho 2012.

[6] M. W. Buku e M. W. Meredith, “Safaricom and M-Pesa in Kenya: Financial Inclusion and Financial Integrity,” *Washington Journal of Law, Tecnhology & Arts,* vol. 8, 2013.

[7] “B-SaaS,” [Online]. Available: http://www.b-saas.com/. [Acedido em Dezembo 2013].

[8] J. Camhi, “Bank Tech,” 22 Março 2013. [Online]. Available: http://www.banktech.com/core-systems/temenos-acquires-us-saas-core-banking-provider-trinovus/d/d-id/1296255?. [Acedido em Dezembo 2013].

[9] B. Services, “BTR Services,” [Online]. Available: http://www.btr-services.be/our-solutions-services/saas-banking-platform. [Acedido em Dezembro 2013].

[10] Cpay360, “Cpay360,” Reliable Software Solutions for Community Banks, [Online]. Available: http://cpay360.com/markets/community-banks. [Acedido em Dezembro 2013].

[11] CSC, “Banking Software,” [Online]. Available: http://www.csc.com/banking/offerings/11090-banking_software. [Acedido em Dezembro 2013].

[12] “Cloud Computing for Financial Markets,” CISCO, 2011.

[13] NCR APTRA, “NCR APTRA Mobile Banking Gateway,” [Online]. Available: http://www.ncr.com/products/banking/mobile-online/aptra-mobile-banking-gateway.

[14] mFoundry, “Mobile Banking & Payments,” [Online]. Available: http://www.mfoundry.com/products-services/#mobile-banking. [Acedido em Dezembro 2013].

[15] ETNA Software, “SaaS & SOA in Retail Banking - Combination for Success”.

[16] D. Garlan e M. Shaw, “An Introduction to Software Architecture,” *Carnegie Mellon University Pittsburgh,* 1993.

[17] L. Bass, P. Clements e R. Kazman, Software Architecture in Practice, Addison-Wesley Professional, 2012.

[18] M. Elkstein, “What is REST?,” Fevereiro 2008. [Online]. Available: http://rest.elkstein.org/2008/02/what-is-rest.html. [Acedido em Agosto 2014].

[19] S. Brown e A. Sehmi, “Understanding Service-Oriented Architecture,” *The Architecture Journal,* vol. I, Janeiro 2004.

[20] C. Teixeira, *Arquiteturas Distribuídas.* [Aula]. Universidade de Aveiro, 2011.

[21] P. Mell e T. Grance, "The NIST Definition of Cloud Computing," *National Institute of Standards and Tecnhology,* nº Recommendations of the National Institute of Standards and Tecnhology, pp. 1-3, 2011.

[22] E. Alecrim, "Infowester," 10 Janeiro 2013. [Online]. Available: http://www.infowester.com/cloudcomputing.php. [Acedido em Agosto 2014].

[23] J. McKendrick, "Cloud Computing's Vendor Lock-In Problem: Why the Industry is Taking a Step Backward," *Forbes,* 2011.

[24] Salesforce, "https://www.salesforce.com/," 01 Setembro 2014. [Online]. Available: https://www.salesforce.com/saas/. [Acedido em Agosto 2014].

[25] Interoute, "Interoute," 02 09 2014. [Online]. Available: http://www.interoute.com/cloud-article/what-private-cloud. [Acedido em Agosto 2014].

[26] Nasuni, "White Paper: The State of Cloud Storage," 2013.

[27] Cloud Spectator, "Cloud Server Performance: A Comparative Analysis of 5 Large Cloud IaaS Providers," 2013.

[28] Microsoft Azure, "Microsoft Azure," [Online]. Available: http://azure.microsoft.com/pt-pt/campaigns/azure-vs-aws/. [Acedido em Agosto 2014].

[29] Microsoft Azure, "Microsoft Azure," [Online]. Available: http://azure.microsoft.com/pt-pt/overview/what-is-azure/.

[30] Microsoft Azure, "Microsoft Azure," [Online]. Available: http://azure.microsoft.com/en-us/documentation/articles/dotnet-develop-multitenant-applications/. [Acedido em Setembro 2014].

[31] Microsoft Azure, “Microsoft Azure,” Base de Dados SQL, [Online]. Available: http://azure.microsoft.com/pt-pt/services/storage/.

[32] Microsoft Azure, “Microsoft Azure,” Serviços Móveis, [Online]. Available: http://azure.microsoft.com/pt-pt/documentation/services/mobile-services/. [Acedido em Setembro 2014].

[33] Microsoft Azure, “Microsoft Azure,” Barramento de Serviços, [Online]. Available: http://azure.microsoft.com/pt-pt/documentation/services/service-bus/. [Acedido em Setembro 2014].

[34] Microsoft Azure, “Microsoft Azure,” [Online]. Available: http://azure.microsoft.com/pt-pt/services/active-directory/. [Acedido em Setembro 2014].

[35] P. Salvatori, “Microsoft Azure,” 10 Setembro 2013. [Online]. Available: http://code.msdn.microsoft.com/windowsazure/How-to-integrate-a-Mobile-8780500c. [Acedido em Setembro 2014].

[36] W3Schools, “W3Schools,” [Online]. Available: http://www.w3schools.com/html/html5_intro.asp. [Acedido em Setembro 2014].

[37] C. Heilmann, “Smashing Magazine,” 11 Outubro 2010. [Online]. Available: http://www.smashingmagazine.com/2010/10/11/local-storage-and-how-to-use-it/. [Acedido em Setembro 2014].

[38] W3Schools, “W3Schools,” [Online]. Available: http://www.w3schools.com/js/. [Acedido em Agosto 2014].

[39] jQuery, “jQuery Mobile,” [Online]. Available: http://jquerymobile.com/. [Acedido em Setembro 2014].

[40] Microsoft Azure, “Microsoft Azure,” 21 Fevereiro 2014. [Online]. Available: http://msdn.microsoft.com/en-us/library/azure/hh674475.aspx. [Acedido em Setembro 2014].

[41] T. Pawlak, “Blog Tom Pawlak,” Fevereiro 2014. [Online]. Available: http://blog.tompawlak.org/generate-unique-identifier-nodejs-javascript. [Acedido em Setembro 2014].

[42] Node.js, “Node JS,” [Online]. Available: http://nodejs.org/api/https.html. [Acedido em Setembro 2014].

[43] NPMJS, “NPMJS,” [Online]. Available: https://www.npmjs.org/package/xml2js. [Acedido em Setembro 2014].

[44] NPMJS, “NPMJS,” [Online]. Available: https://www.npmjs.org/package/mssql. [Acedido em Setembro 2014].

[45] M. Jeff. [Online]. Available: https://code.google.com/p/crypto-js/#MD5. [Acedido em Setembro 2014].

[46] A. Charland e B. Leroux, *Communications of the ACM CACM,* vol. 54, pp. 49-53, 2011.

[47] D. Crawford, “Sealed Abstract,” 6 Maio 2013. [Online]. Available: http://sealedabstract.com/rants/mobile-Web-apps-are-slow. [Acedido em Setembro 2014].

[48] I. Flatow's, “Israel's Blogging Community,” 2 Julho 2011. [Online]. Available: http://blogs.microsoft.co.il/idof/2011/07/02/cross-origin-resource-sharing-cors-and-wcf/. [Acedido em Setembro 2014].

[49] Microsoft, “Microsoft Developer Network,” [Online]. Available: http://msdn.microsoft.com/en-us/library/ms733127(v=vs.110).aspx. [Acedido em Outubro 2014].

[50] Microsoft, “Microsoft Developer Network,” [Online]. Available: http://msdn.microsoft.com/en-us/library/system.servicemodel.Web.Webinvokeattribute(v=vs.110).aspx. [Acedido em Outubro 2014].

[51] Microsoft, “Microsoft Developer Network,” [Online]. Available: http://msdn.microsoft.com/en-us/library/system.servicemodel.Web.Webgetattribute(v=vs.110).aspx. [Acedido em Outubro 2014].

[52] Westerb Oregon University, “Connecting Applications through the Windows Azure Service Bus,” [Online]. Available: http://www.wou.edu/~rvitolo06/WATK/Labs/ServiceBusConnectingApps/Lab.html/html/DocSet_6569841c-703e-4795-ba6f-cb5edfb2fe4c.html. [Acedido em Julho 2014].

Anexo A

Modelo de Dados

Neste anexo serão descrito os modelos de dados elaborados para suporte à arquitetura.

Modelo de Dados de suporte à configuração dos módulos

Banco

A entidade banco tem como objetivo armazenar as informações das diferentes agências bancárias.

Atributo	Descrição
idBanco	Identificador do código da entidade bancária
nome	Nome da entidade bancária
site	Página Web da entidade bancária
descrição	Descrição da entidade bancária

Tabela 1 – Entidade Banco

O banco tem um relacionamento de muitos-para-muitos com a entidade módulo.

Módulo

A entidade módulo tem como objetivo armazenar as informações referentes aos módulos que os bancos registam.

Atributo	Descrição
idModulo	Identificador do módulo
nome	Nome do módulo
endereço	Endereço onde o módulo se encontra exposto
local	Identificador se o módulo se encontra na *Cloud* ou Local
username	*Username* de acesso ao serviço
password	*Password* de acesso ao serviço
idTipoModulo	Identificador do tipo de módulo

Tabela 2 - Entidade Módulo

O módulo tem um relacionamento de muitos-para-muitos com a entidade banco. Um módulo tem também um tipo de módulo.

TipoModulo

A entidade tipoModulo tem como objetivo armazenar os diferentes tipos de módulos que o sistema suporta.

Atributo	Descrição
idTipoModulo	Identificador do tipo de módulo
nome	Nome do tipo de módulo

Tabela 3 - Entidade TipoModulo

O tipoModulo tem um relacionamento de um-para-muitos com a entidade módulo.

Banco_Modulo

A entidade banco_modulo tem como objetivo armazenar os módulos que pertencem a cada entidade bancária.

Atributo	Descrição
idBanco	Identificador da entidade bancária

idModulo	Identificador do módulo
ativo	*Flag* que permite determinar se o módulo para uma determinada entidade bancária se encontra ativo ou não

Tabela 4 - Entidade Banco Módulo

O banco_modulo é uma entidade associativa que surge do relacionamento de muitos-para-muitos entre as entidades banco e módulo.

Modelo de dados de Suporte à aplicação

Cliente

A entidade cliente tem como objetivo armazenar a informação referente aos clientes das diferentes entidades bancárias.

Atributo	Descrição
idCliente	Identificador do cliente
nome	Nome do cliente
dataNasc	Data de Nascimento
cc	Número do cartão do cidadão
nif	Número de contribuinte
telefone	Telefone
email	Email
morada	Morada completa
codPostal	Código de Postal
pais	País de residência
profissao	Profissão

Tabela 5 - Entidade Cliente

A entidade cliente tem um relacionamento de muitos-para-muitos com a entidade banco.

Banco

A entidade banco tem como objetivo armazenar as informações das diferentes agências bancárias.

Atributo	Descrição
idBanco	Identificador do código da entidade bancária
nome	Nome da entidade bancária
site	Página Web da entidade bancária
descrição	Descrição da entidade bancária

Tabela 6 - Entidade Banco

O banco tem um relacionamento de muitos-para-muitos com a entidade cliente e um-para-muitos com a entidade gestor.

Gestor

A entidade gestor tem como objetivo armazenar informações referentes os gestores de conta de cada entidade bancária.

Atributo	Descrição
idGestor	Identificador do gestor de conta de um determinado banco
nome	Nome do gestor de conta
username	Username de acesso à aplicação
password	Password de acesso à aplicação
admin	Flag que sinaliza se o gestor de conta em questão é administrador
idBanco	Identificador da entidade bancária ao qual pertence

Tabela 7 - Entidade Gestor

O gestor tem um relacionamento de muitos-para-um com a entidade banco.

Cliente_Banco

A entidade cliente_banco tem como objetivo armazenar os clientes de cada entidade bancária.

Atributo	Descrição
idClienteBanco	Identificador do cliente de um determinado banco
idCliente	Identificador do cliente
idBanco	Identificador do código da entidade bancária

Tabela 8 - Entidade ClienteBanco

O cliente_banco é uma entidade associativa que surge do relacionamento de muitos-para-muitos entre as entidades banco e cliente. O cliente de um determinado banco (cliente_banco) tem um relacionamento de um-para-muitos com a entidade conta.

Conta

A entidade conta tem como objetivo armazenar os dados referentes às diferentes contas dos clientes de uma determinada entidade bancária.

Atributo	Descrição
idConta	Identificador do número de conta
nib	Número de Identificação Bancária
montante	Montante
idClienteBanco	Identificador do cliente de um determinado banco
idTipoConta	Identificador do tipo de conta

Tabela 9 - Entidade Conta

A entidade conta tem um relacionamento de muitos-para-um com a entidade cliente_banco. Tem ainda um relacionamento de muitos-para-um com a entidade tipoConta e de um-para-muitos com a entidade movimento.

TipoConta

A entidade tipoConta tem como objetivo armazenar os diferentes tipos de contas que um cliente pode ter.

Atributo	Descrição
idTipoConta	Identificador do tipo de conta
nome	Nome do tipo de conta

Tabela 10 - Entidade TipoConta

A entidade TipoConta tem um relacionamento de um-para-muitos com a entidade conta.

Movimento

A entidade movimento tem como objetivo armazenar os movimentos bancários que são feitos pelos clientes das diferentes agências bancárias.

Atributo	Descrição
idMovimento	Identificador do movimento
data	Data da transação
quantia	Quantia transacionada
idConta	Identificador da conta
idTipoMovimento	Identificador do tipo de movimento

Tabela 11 - Entidade Movimento

A entidade movimento tem um relacionamento de muitos-para-um com e entidade conta e tipoMovimento.

TipoMovimento

A entidade tipoMovimento tem como objetivo armazenar os diferentes tipos de movimentos que podem existir numa conta bancária.

Atributo	Descrição
idTipoMovimento	Identificador do tipo de movimento
nome	Nome do tipo de movimento

Tabela 12 - Entidade TipoMovimento

A entidade tipoMovimento tem um relacionamento de um-para-muitos com a entidade movimento.

Anexo B

Este anexo pretende deixar uma memória futura para o desenvolvimento de arquiteturas assentes nestas tecnologias.

Processo de Desenvolvimento

O desenvolvimento do protótipo está dividido em quatro fases. A solução é composta por:

Contratos: contém os serviços e os contratos usados pelo serviço WCF;

Serviço: contém o serviço WCF que representa a linha de negócios da aplicação invocada pelos serviços móveis. O serviço corre numa aplicação consola;

Aplicação: contém a aplicação em HTML5/Javascript;

Serviços Locais: contém a definição dos serviços locais, expostos no *service bus relay*

Contratos

Este projeto contém os vários contratos dos serviços utilizados na definição das funções utilizadas para o desenvolvimento.

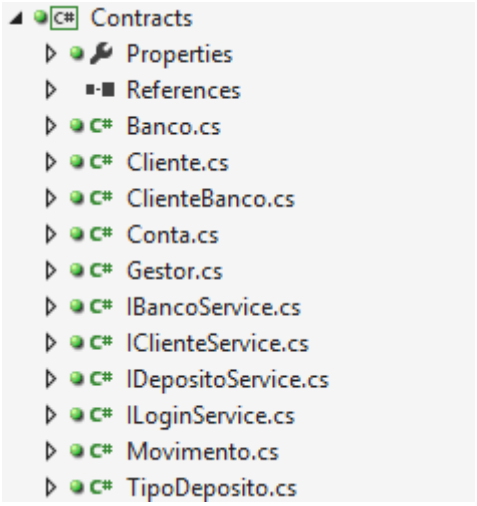

Figura 1 - Ficheiros - Contracts

Os serviços estão organizados por Bancos, Clientes, Movimentos e Autenticação. Para cada um dos serviços foi criado um contrato de dados, que é um acordo formal entre o serviço e o cliente que tem por objetivo descrever abstratamente os dados a serem trocados [49]. O seguinte exemplo ilustra o contrato de dados de um banco.

```
namespace Contracts
{
    [DataContract(Name = "bank", Namespace = "…")]
    public class Banco
    {
        [DataMember(Name = "idBanco", Order = 1)]
        public int idBanco { get; set; }
        [DataMember(Name = "nome", Order = 2)]
        public string nome { get; set; }
        [DataMember(Name = "site", Order = 3)]
        public string site { get; set; }
        [DataMember(Name = "descricao", Order = 4)]
        public string descricao { get; set; }
    }
}
```

O IBancoService armazena o contrato usado pelo serviço. Aqui é especificado o método de acesso, o *template*, os formatos de pedido e resposta e ainda os argumentos de cada método. Um método POST é definido da seguinte forma [50]:

```
//Do WebInvoke deve constar o método, o template, o formato de pedido/resposta
[WebInvoke(Method = "POST", UriTemplate = "methodName", RequestFormat = WebMessageFor-
mat.Json, ResponseFormat = WebMessageFormat.Json)]

//Do OperationContract deve constar o nome do método, e o nome do método de res-
posta
[OperationContract(Action = "methodName", ReplyAction = "methodNameResponse")]

//Do return deve constar o nome do parâmetro retornado
[return: MessageParameter(Name = "returnParameter")]

//O método deve conter o tipo de dados retornado, o nome do método e o nome e pa-
râmetro de envio
DataType methodName([MessageParameter(Name = "ParameterName")]DataType parameter);
```

O método addBank exemplifica o contrato de um método POST:

```
namespace Contracts
{
    [ServiceContract(Namespace = …")]
    public interface IBancoService
    {

        [WebInvoke(Method = "POST", UriTemplate = "addBank", RequestFormat = WebMessa-
geFormat.Json, ResponseFormat = WebMessageFormat.Json)]
        [OperationContract(Action = "addBank", ReplyAction = "addBankResponse")]
        [return: MessageParameter(Name = "bank")]
        Banco addBank([MessageParameter(Name = "bank")]Banco bank);

    }
}
```

Já se se tratar de um GET, o pedido é feito da seguinte forma [51]:

```
//Do WebGet deve constar template com a indicação dos parâmetros, o body style e
o formato da resposta
[WebGet(UriTemplate = "methodName?parameter={parameterValue}", BodyStyle = WebMessageBo-
dyStyle.Bare, ResponseFormat = WebMessageFormat.Json)]

//Do OperationContract deve constar o nome do método, e o nome do método de res-
posta
[OperationContract(Action = "methodName", ReplyAction = "methodNameResponse")]

//Do return deve constar o nome do parâmetro retornado
[return: MessageParameter(Name = "returnParameter")]

//O método deve conter o tipo de dados retornado, o nome do método e o nome e pa-
râmetro de envio
DataType methodName([MessageParameter(Name = "ParameterName")]DataType paramater);
```

O método getBank exemplifica o contrato de um método POST:

```
namespace Contracts
{
    [ServiceContract(Namespace = "…
    public interface IBancoService
    {

[WebGet(UriTemplate = "getBank?id={idBanco}", BodyStyle = WebMessageBodyStyle.Bare, Res-
ponseFormat = WebMessageFormat.Json)]
        [OperationContract(Action = "getBank", ReplyAction = "getBankResponse")]
        [return: MessageParameter(Name = "banks")]
        List<Banco> getBank([MessageParameter(Name = "idBanco")]int idBanco);
       }
}
```

Os restantes ficheiros assemelham-se aos exemplos acima mencionados, seguindo a mesma estrutura e organização.

Serviço

O projeto *Service* contém o código dos diferentes serviços invocados pelos serviços móveis, bem como os ficheiros que efetuam o acesso ao modelo de dados.

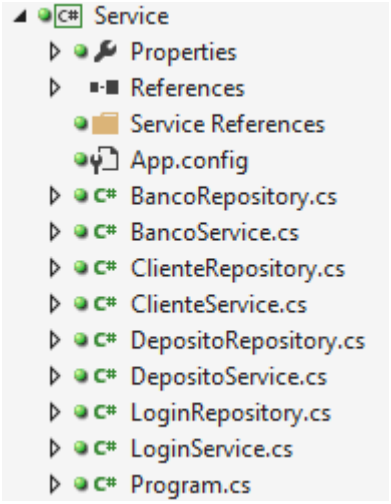

Figura 2 - Ficheiros – Service

O seguinte código corresponde ao serviço BankService cuja interface foi mostrada anteriormente.

```
namespace Service
{
    [ServiceBehavior(Namespace = "…")]
    public class BancoService : IBancoService
    {

       public List<Banco> getBank(int idBanco)
        {
                return BancoRepository.getBank(idBanco).Result;
        }

        public Banco addBank(Banco bank)
        {
            return BancoRepository.addBank(bank).Result;
        }


       […]

    }
}
```

Dentro dos serviços, são chamados os métodos que acedem ao modelo de dados, como se pode ver através do seguinte exemplo:

```
namespace Service
{
    public static class BancoRepository
    {
        […]

        public static async Task<List<Banco>> getBank(int idBanco)
        {


          // Verifica se o(s) parâmetro(s) têm valores válidos
          // Conecta a base de dados através do SQL Connection
          // Chama o Stored Procedure e envia-lhe os respetivos parâmetros
          // Retorna os resultados (se aplicável)


        }

    }
}
```

É também neste projeto que os serviços são expostos no *service bus relay*. Os dados são configurados no ficheiro App.Config nos campos *Bindings*, *Services* e *Behaviors* [52]:

```
<!-- Binding Configuration -->
       <…>
<!-- Behavior Configuration -->
       <…>
<!-- Service Conbfiguration -->
<services>
<service name="Service.BancoService" behaviorConfiguration="serviceBehavior">
<endpoint address="https://teamservicebus.servicebus.windows.net/mbank/basichttp"
          behaviorConfiguration="serviceBusEndpointBehavior"
          binding="basicHttpRelayBinding"
          bindingConfiguration="basicHttpRelayBinding"
          contract="Contracts.IBancoService"
          name="MBank"/>
      </service>
    […]
    </services>
```

No ficheiro Program.cs estas mesmas configurações são instanciadas:

```
// Instancia os hosts usando os dados definidos no ficheiro de configuração

var banco = new ServiceHost(typeof(BancoService));
banco.Open();
[…]
```

Para a execução da aplicação é obrigatório que o projeto *Services* seja iniciado, pois só assim é que os serviços serão expostos no *service bus* para posterior comunicação com os serviços móveis.

Nos serviços móveis localizados no Windows Azure, como mostra a **Figura 3**, cada uma das funções presentes nos diversos contratos utilizados encontra-se também no respetivo serviço móvel.

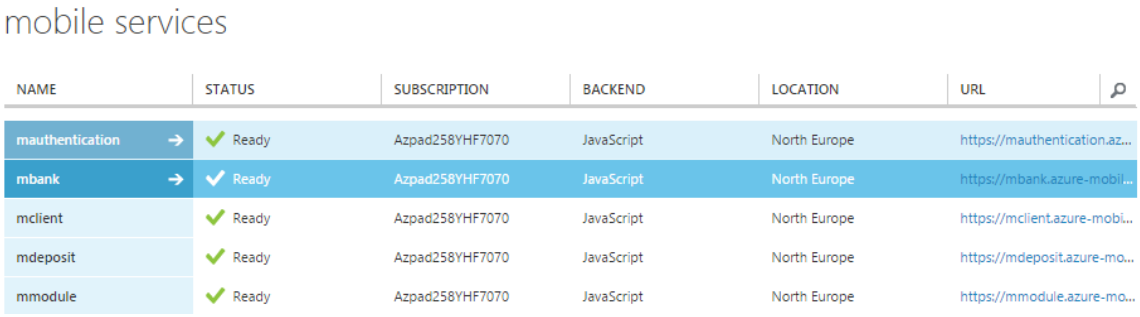

Figura 3 - Serviços Móveis

Na API de cada um dos serviços móveis, está o *backend* que permite definir os métodos de acordo com o pedido. São também carregados os módulos Node.js necessários e é definido também o endereço do *service bus*, o utilizador e a *password* de acesso que permite expor os serviços. O método getAcsToken é chamado antes de cada uma das funções e tem por objetivo obter o *token* de segurança. A Figura 4 permite exemplificar isso.

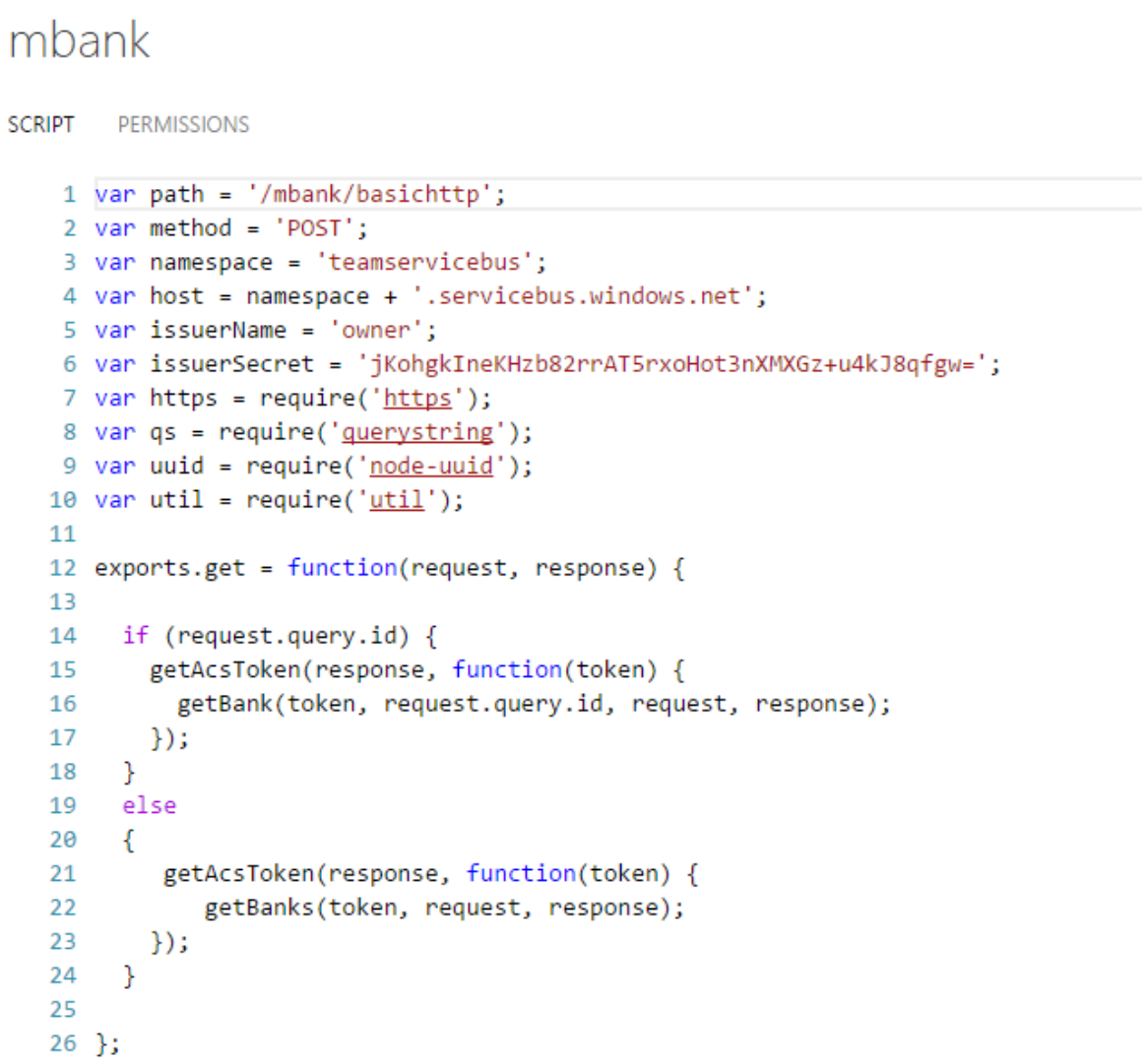

Figura 4 - Backend de um serviço móvel

Site

Este projeto contém os ficheiros HTML, Javascript e CSS utilizados para a construção da aplicação móvel. É aqui que se encontra também o *proxy* que irá permitir o acesso aos serviços locais expostos através do *service bus relay*.

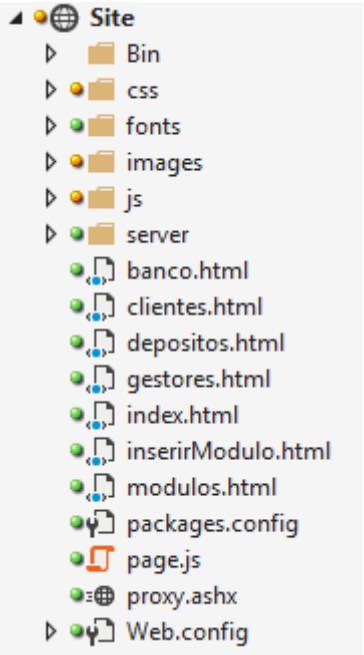

Figura 5 - Ficheiros – Site

Serviços Locais

Tal como referido anteriormente, para a utilização da arquitetura através do acesso local, é necessário a construção de serviços que são depois expostos através do *service bus relay* para assim serem consumidos pela aplicação. Assim, foram construídos dois serviços para que fosse possível testar o acesso aos serviços *on-premise*.

Tal como os serviços acima referidos, estes também são constituídos pelos contratos de dados, serviços e os respetivas interfaces e seguem a mesma lógica.

Após a construção dos serviços, é necessário expô-los no *service bus* para que possam ser posteriormente acedidos. Assim, foi criado um projeto que referência os serviços que se

pretende que sejam expostos. O ficheiro de configuração App.Config contém os seguintes dados:

```
<!-- Binding Configuration -->
       <…>
<!-- Behavior Configuration -->
       <…>
<!-- Service Conbfiguration -->
<services>
 <service name="MC.MClients" behaviorConfiguration="serviceBehavior">
       <endpoint name="MClients" contract="MC.IMClients" binding="WebHttpRelayBinding"
       bindingConfiguration="WebHttpRelayBinding" behaviorConfiguration="sharedSecre-
       tClientCredentials" address=""/>
</service>

<service name="MD.MDeposit" behaviorConfiguration="default">
       <endpoint name="MDeposits" contract="MD.IMDeposit" binding="WebHttpRelayBinding"
       bindingConfiguration="WebHttpRelayBinding" behaviorConfiguration="sharedSecre-
       tClientCredentials" address=""/>
</service>
```

Para concluir a exposição dos serviços, à semelhança do que foi referido anteriormente é necessário instanciar os mesmos.

```
// É necessário segurança a nível de transporte, através do https
var serviceNamespace = "teamservicebus";

Uri addressClients = ServiceBusEnvironment.CreateServiceUri("https", serviceNamespace,
"mclientslocal");
Uri addressDeposits = ServiceBusEnvironment.CreateServiceUri("https", serviceNamespace,
"mdepositslocal");

WebServiceHost mclient = new WebServiceHost(typeof(MClients),addressClients);
WebServiceHost mdeposit = new WebServiceHost(typeof(MDeposit), addressDeposits);

mclient.Open();
mdeposit.Open();
```

De forma geral, estes foram os passos necessários para a construção do protótipo e respetiva exemplificação da arquitetura.

Anexo C

Protótipo

Tal como referido anteriormente, será apresentado neste anexo o protótipo na íntegra para que seja possível perceber a sua organização e principais funcionalidades. Antes de iniciar a aplicação é fundamental garantir que os serviços móveis estão a ser expostos no *service bus*, permitindo assim, que seja possível a sua invocação através deste, como mostra a Figura 6Figura. Também os serviços locais a utilizar, devem encontrar-se igualmente expostos para que o *proxy* da aplicação os possa aceder, como mostra a Figura 7.

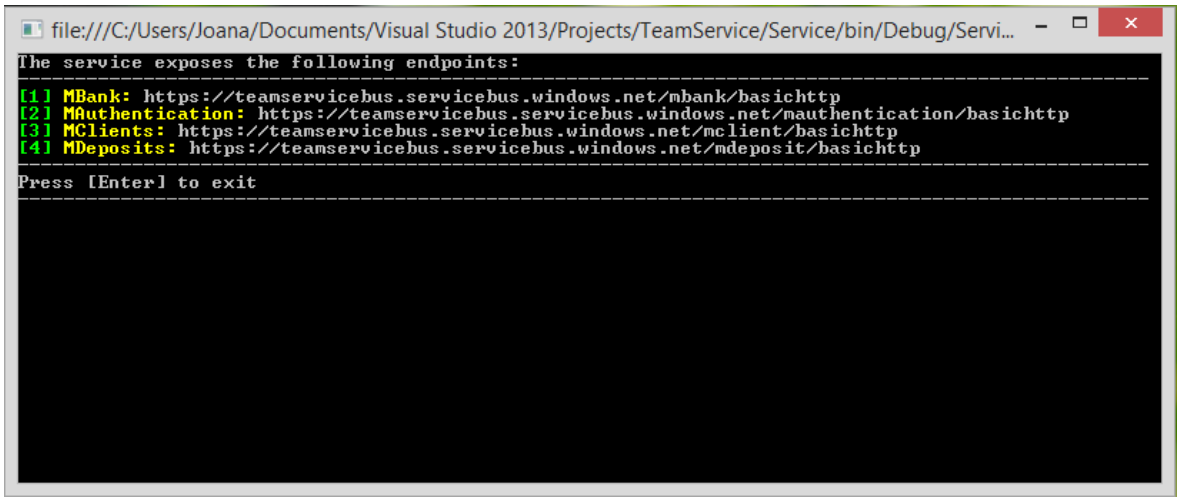

Figura 6 - Serviços Móveis expostos através do service bus

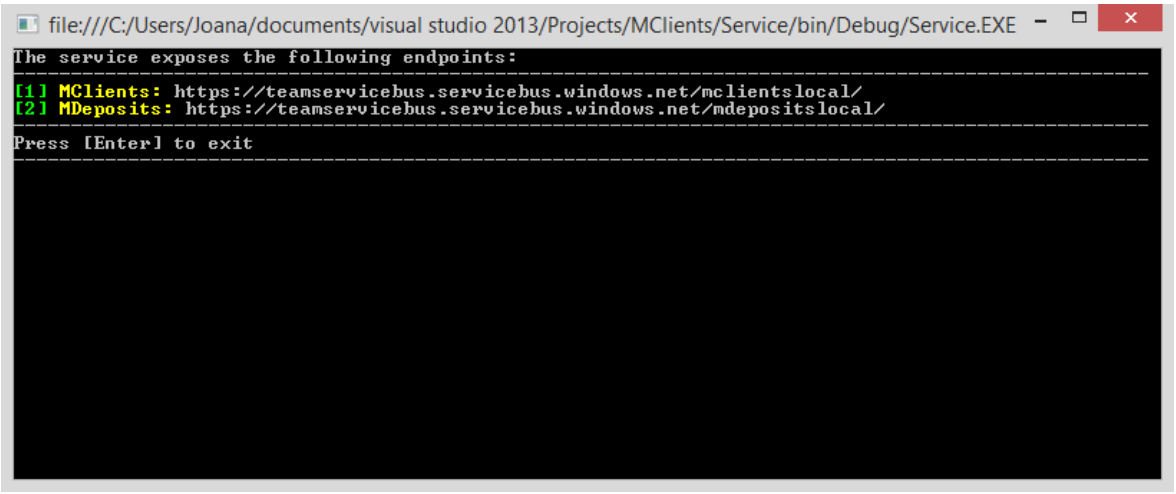

Figura 7 - Serviços locais expostos através do service bus

As imagens seguintes têm por objetivo demonstrar as funcionalidades do sistema tendo em conta a proposta de arquitetura apresentada. Existem dois módulos configurados em que o módulo *Clients* se encontra na *cloud* e o módulo *Deposits* é um módulo local exposto através do *service bus*. No entanto, como se irá perceber, a utilização é transparente e o gestor não sente qualquer diferença na utilização dos módulos.

Quando a aplicação é iniciada, é mostrado ao gestor o ecrã de *login* da mesma (Figura 8). Aqui, o gestor insere as credenciais de acesso. Após verificação das mesmas, o gestor terá acesso então às funcionalidades da aplicação (Figura 9).

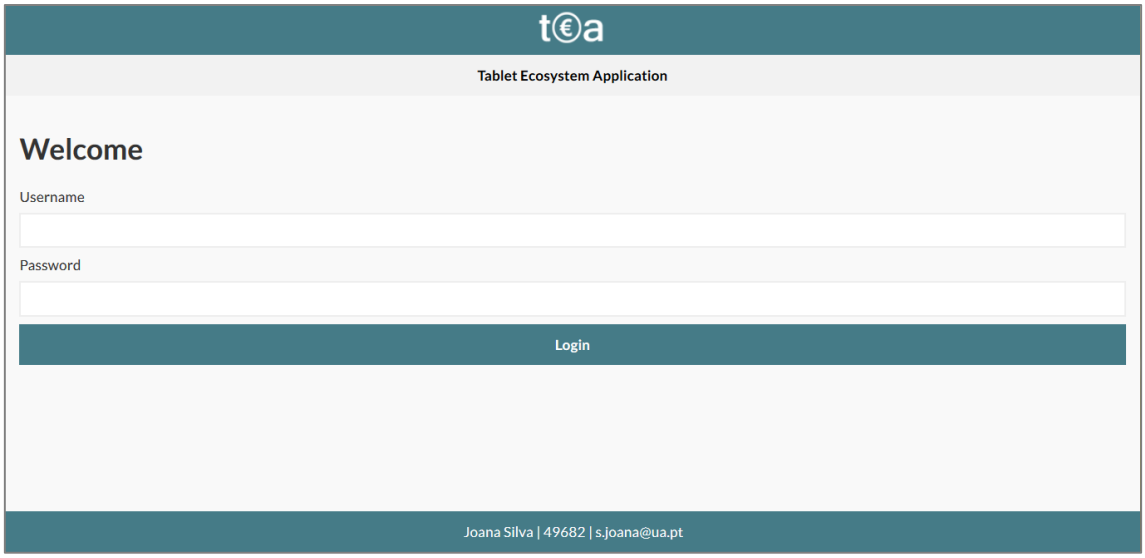

Figura 8 – Ecrã de Login

Figura 9 - Ecrã de Boas-Vindas

A partir do ecrã de boas-vindas, é possível aceder aos menus laterais que permitem aceder aos módulos que cada a entidade bancária registou. No menu lateral esquerdo (**Figura 10**), é possível aceder ao módulo de configuração de módulos e aos módulos configurados em si.

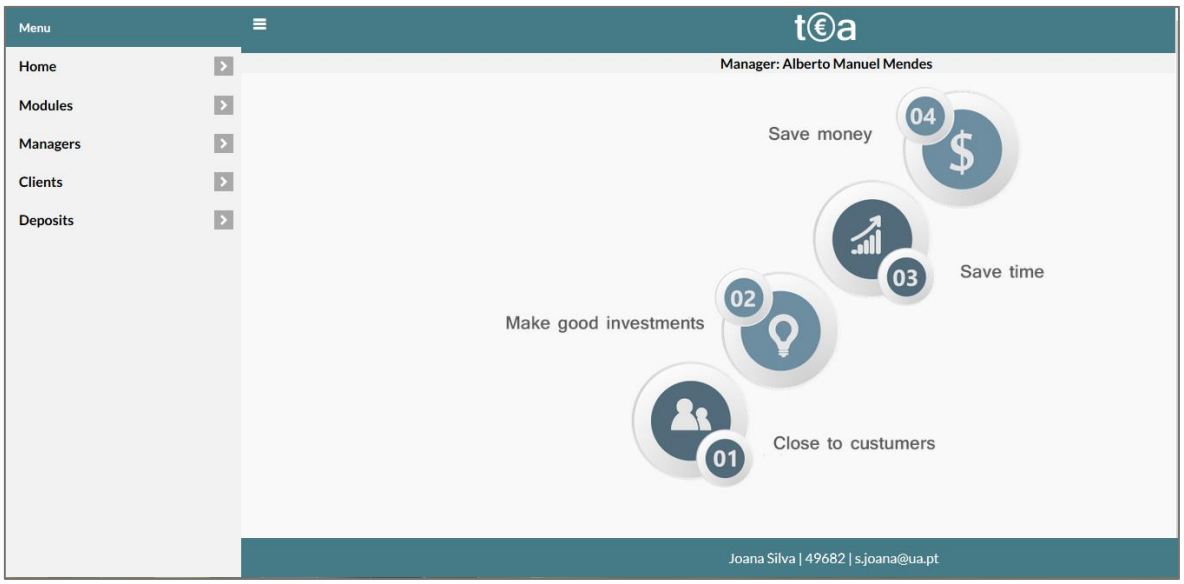

Figura 10 - Menu lateral esquerdo

Já o menu lateral situado à direita (**Figura 11**), permite editar os dados do gestor com sessão iniciada assim como efetuar o *logout*.

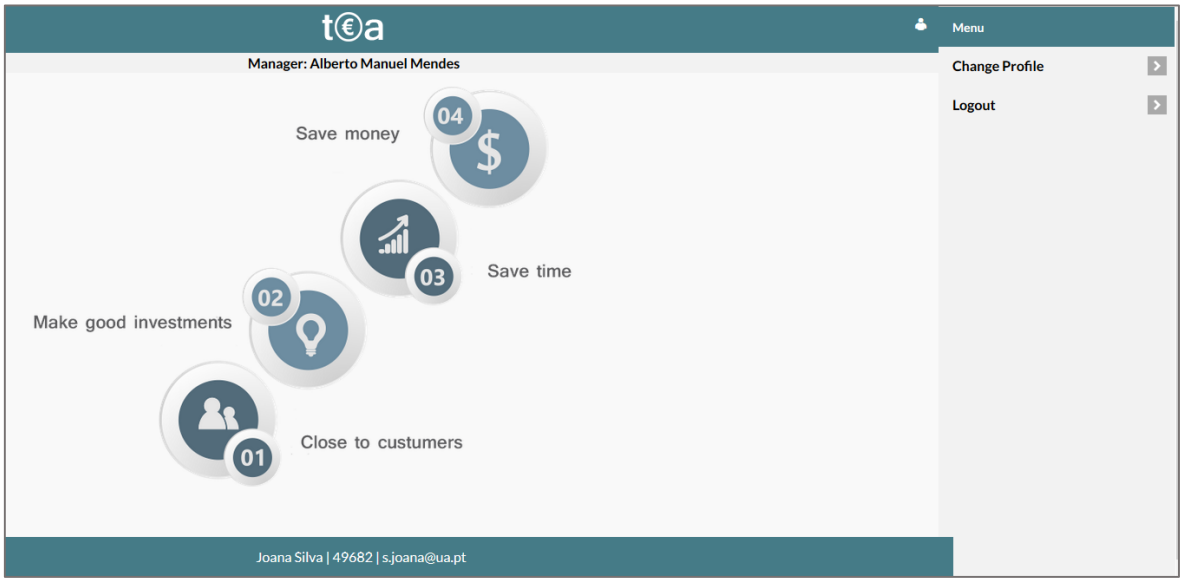

Figura 11 - Menu lateral direito

Relativamente aos módulos inicialmente configurados pela entidade bancária em questão, é possível através da aplicação criar novos módulos e fazer a gestão dos mesmos. Para criar novos módulos, basta aceder ao menu lateral esquerdo e escolher o menu *Modules*. (**Figura 12**)

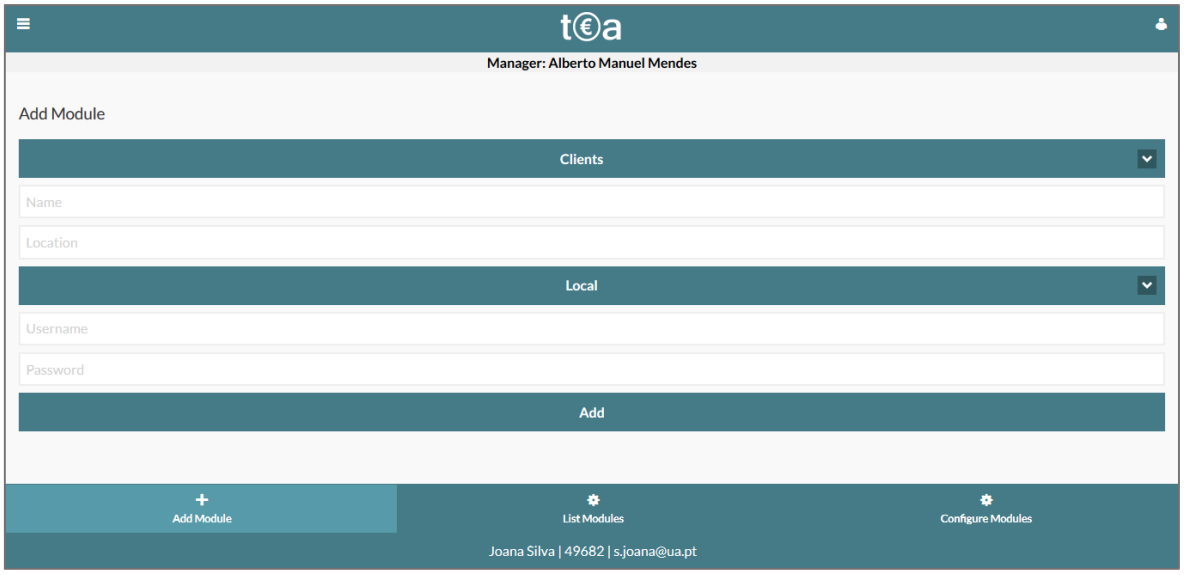

Figura 12 - Adicionar novo módulo

Para criar um novo módulo, basta escolher o tipo de módulo, seguidamente dar-lhe um nome, indicar o endereço Web e se se encontra num servidor local ou na *cloud*. Por fim, indicar o *username* e a *password* de acesso ao módulo.

A gestão de módulos, para além de permitir obter uma listagem dos mesmos, permite ainda editar os dados ou eliminar caso seja necessário (**Figura 13**).

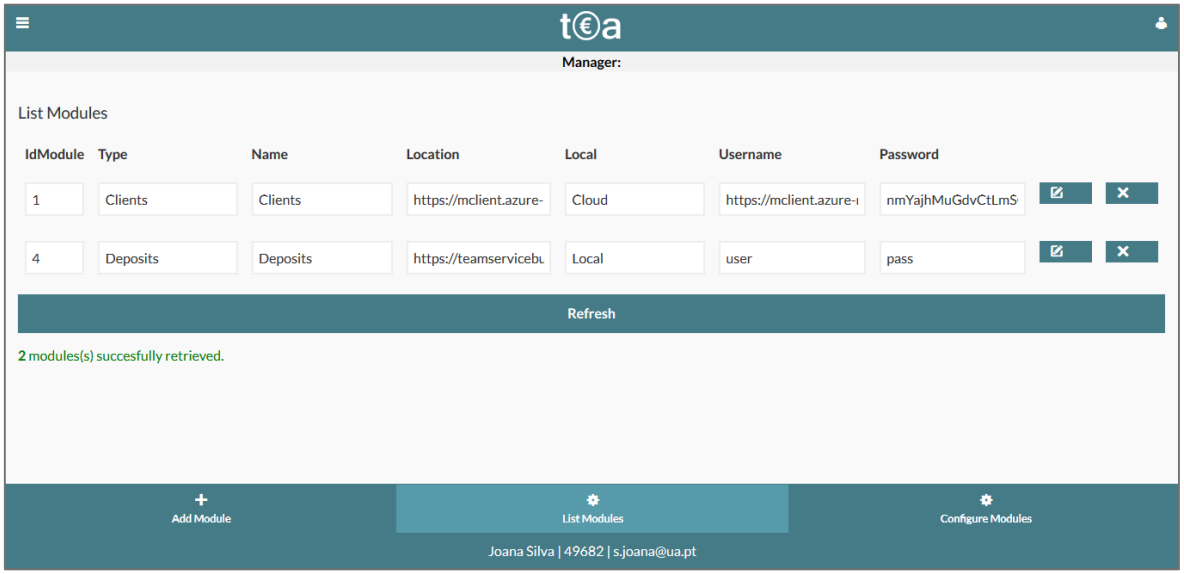

Figura 13 - Gestão de módulos

Existe ainda a possibilidade de configurar os módulos, desativando-os ou ativando-os caso seja necessário (**Figura 14**).

Figura 14 - Configurar módulos

Já no módulo de clientes, existe a possibilidade de inserir novos clientes. Como consequência da adição de um novo cliente, é criada também uma conta associada ao mesmo (**Figura 15**).

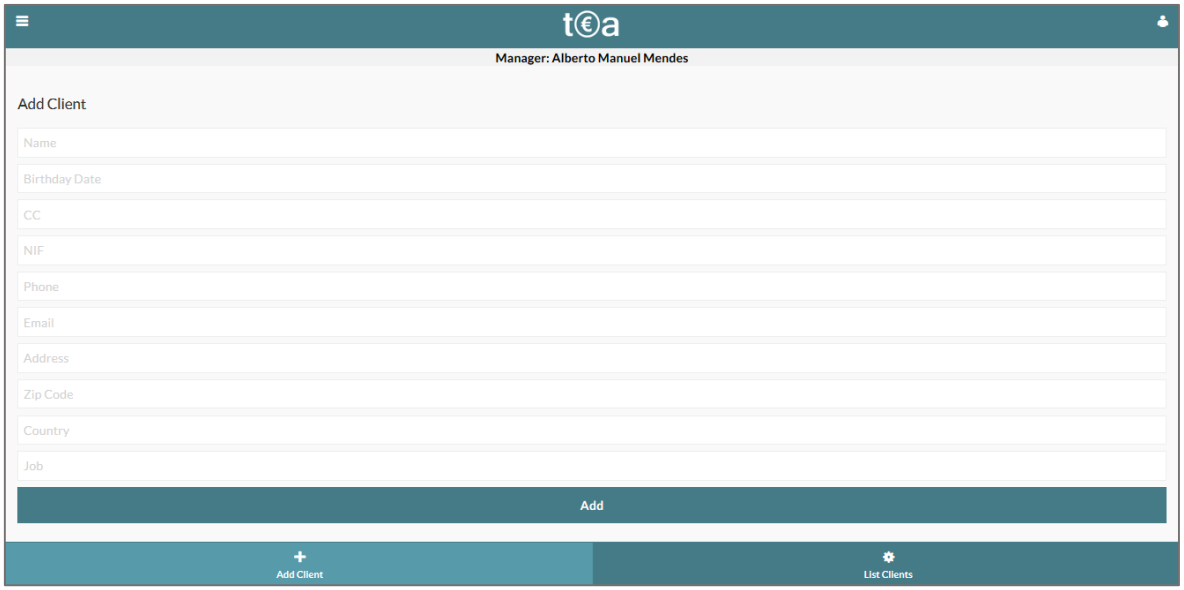

Figura 15 - Adicionar clientes

Para adicionar novos clientes, é necessário inserir dados tais como o nome, data de nascimento, número do cartão de cidadão, número de contribuinte, número de telefone, email, morada, código-postal, país e profissão.

Também existe a possibilidade de gerir os clientes, podendo ver todos os clientes da entidade bancária, assim como editar os seus dados, e eliminá-los do sistema caso seja necessário (**Figura 16**).

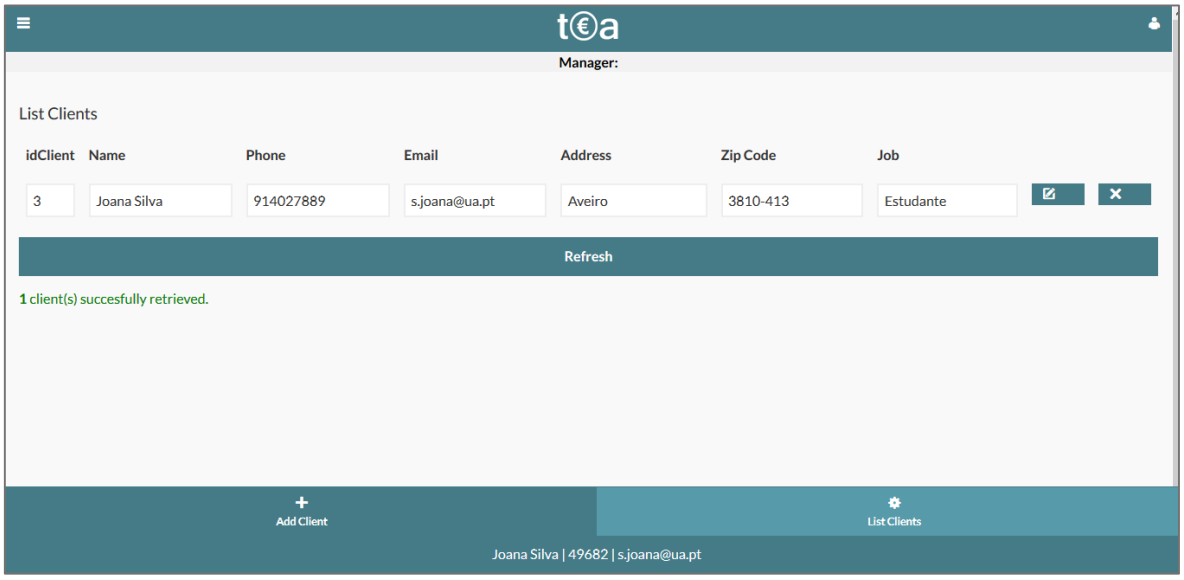

Figura 16 - Gerir clientes

Já no módulo de depósitos, existem também várias operações que são possíveis de realizar. Podemos começar por visualizar o saldo de uma determinada conta (**Figura 17**).

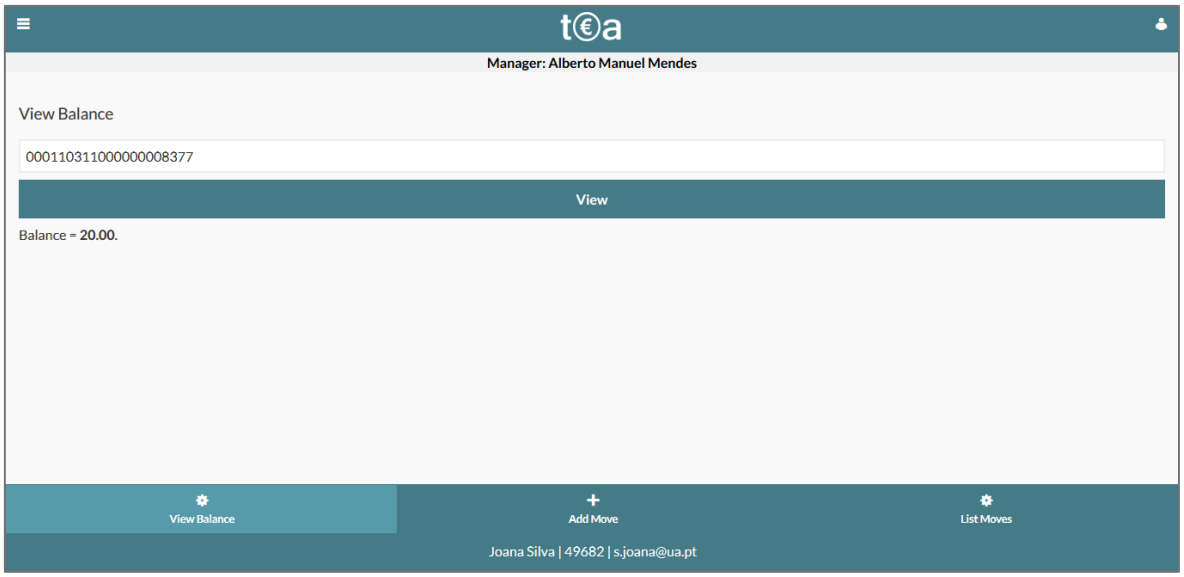

Figura 17 - Visualizar saldo

É também possível adicionar movimentos tais como depósitos ou levantamentos de/para uma determinada conta (**Figura 18**).

Figura 18 - Adicionar movimentos

Para além de adicionar movimentos, é também possível ver os movimentos entre um determinado intervalo de tempo (**Figura 19**).

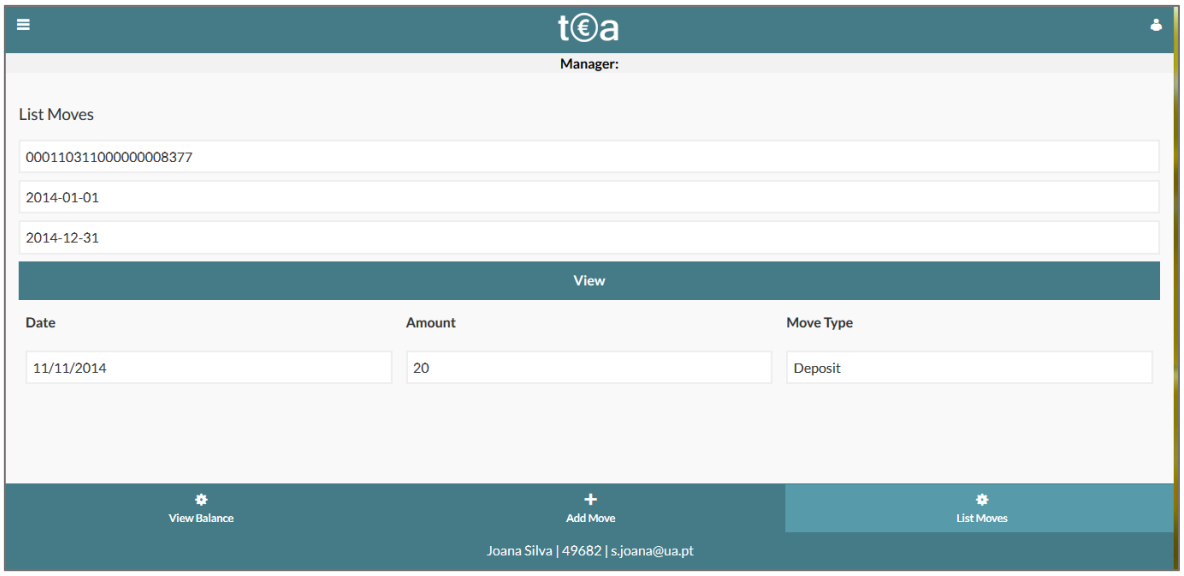

Figura 19 - Lista de movimentos

As funcionalidades demonstradas anteriormente dizem respeito às operações que as entidades bancárias podem realizar. No entanto, também existe um painel administrativo e que permite a um super administrador fazer a gestão de entidades bancárias, adicionando, editando e eliminando entidades bancárias (**Figura 20**) e (**Figura 21**).

Figura 20 - Adicionar entidades bancárias

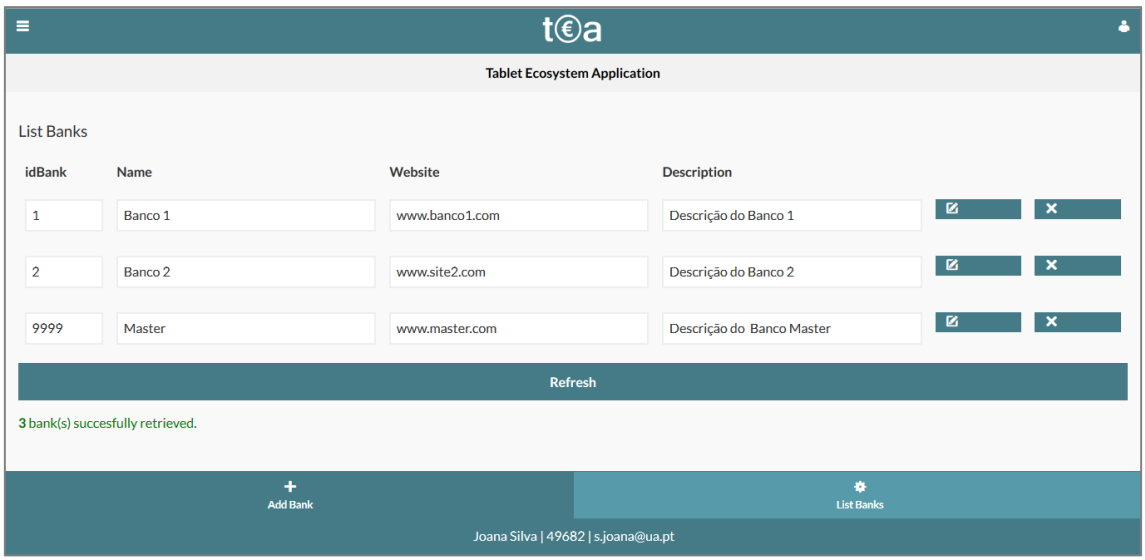

Figura 21 - Gerir entidades bancárias

Também o super-administrador pode registar gestores bancários e associá-los às respetivas entidades bancárias (**Figura 22**) e ainda visualizar todos os gestores registados (**Figura 23**).

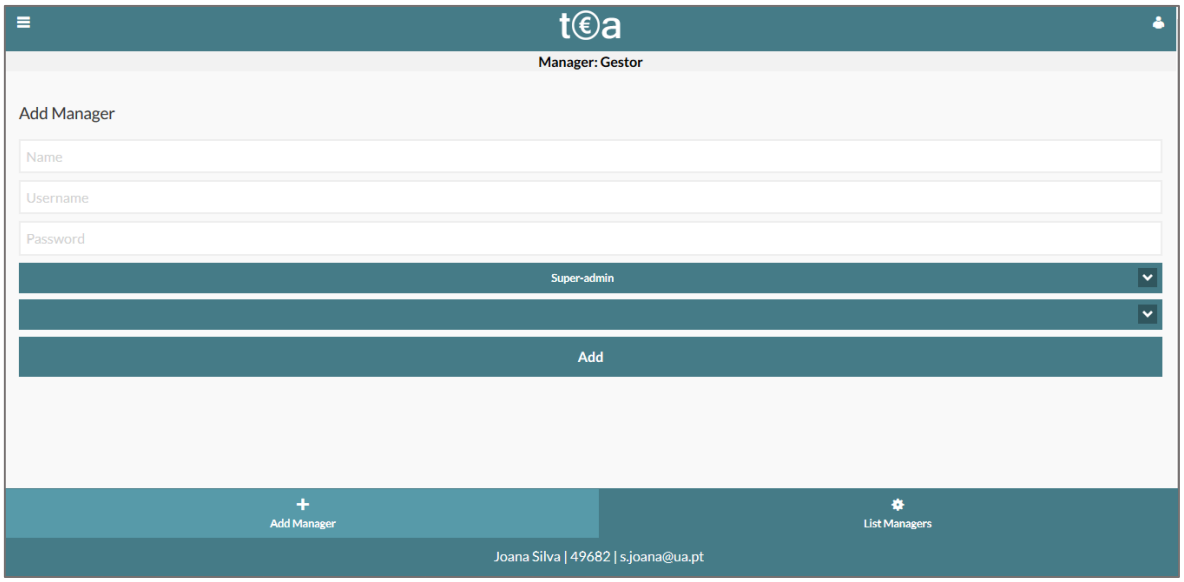

Figura 22 - Registo de Gestores

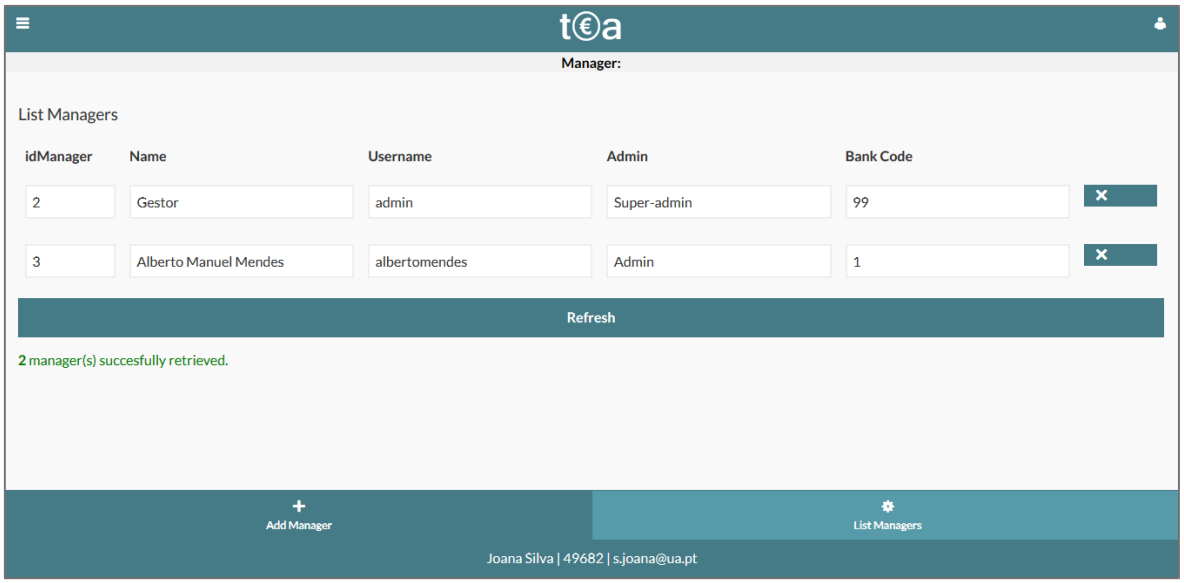

Figura 23 - Gerir gestores de conta