

Camada (interface) de interoperabilidade
do
Sistema de Informação de Apoio ao Ensino (SIDE)

Por

Fernando Manuel Fernandes Rodrigues

Orientador: Professor Doutor Luís Filipe Leite Barbosa

Dissertação submetida à
UNIVERSIDADE DE TRÁS-OS-MONTES E ALTO DOURO
para obtenção do grau de
MESTRE
em Engenharia Informática

Camada (interface) de interoperabilidade
do
Sistema de Informação de Apoio ao Ensino (SIDE)

Por

Fernando Manuel Fernandes Rodrigues

Orientador: Professor Doutor Luís Filipe Leite Barbosa

Dissertação submetida à
UNIVERSIDADE DE TRÁS-OS-MONTES E ALTO DOURO
para obtenção do grau de
MESTRE
em Engenharia Informática

Orientação Científica :

Professor Doutor Luís Filipe Leite Barbosa

Professor Auxiliar do
Departamento de Engenharias
Universidade de Trás-os-Montes e Alto Douro

Acompanhamento do trabalho :

Mestre Jorge José dos Santos Borges

Coordenador do
Núcleo de Sistemas de Informação
Universidade de Trás-os-Montes e Alto Douro

"A mente que se abre a uma nova ideia jamais voltará ao seu tamanho original."

Albert Einstein (1879 – 1955) – Físico

"Eu não temo os computadores, temo a ausência deles."

Isaac Asimov (1920 – 1992) – Escritor

À minha mãe pela dedicação e apoio incondicional.

UNIVERSIDADE DE TRÁS-OS-MONTES E ALTO DOURO
Mestrado em Engenharia Informática

Os membros do Júri recomendam à Universidade de Trás-os-Montes e Alto Douro a aceitação da dissertação intitulada “**Camada (interface) de interoperabilidade do Sistema de Informação de Apoio ao Ensino (SIDE)**” realizada por **Fernando Manuel Fernandes Rodrigues** para satisfação parcial dos requisitos do grau de **Mestre**.

Dezembro 2018

Presidente: **Professor Doutor Hugo Alexandre Paredes Guedes da Silva,**

Professor Auxiliar com Agregação do Departamento de Engenharias da Universidade de Trás-os-Montes e Alto Douro

Vogais do Júri: **Professor Doutor Paulo Alexandre Vara Alves,**

Professor Adjunto da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Bragança

Professor Doutor Luís Filipe Leite Barbosa,

Professor Auxiliar do Departamento de Engenharias da Universidade de Trás-os-Montes e Alto Douro

Camada (interface) de interoperabilidade
do
Sistema de Informação de Apoio ao Ensino (SIDE)

Fernando Manuel Fernandes Rodrigues

Submetido na Universidade de Trás-os-Montes e Alto Douro
para o preenchimento dos requisitos parciais para obtenção do grau de
Mestre em Engenharia Informática

Resumo — O Sistema de Informação de Apoio ao Ensino (SIDE) é um sistema que assenta a maior parte do seu funcionamento na plataforma *web*. O objetivo principal é disponibilizar o acesso à informação e facilitar a gestão de informação por parte de todos os intervenientes na atividade académica dos vários departamentos pertencentes à Universidade de Trás-os-Montes e Alto Douro (UTAD).

Esta dissertação foca a evolução e desenvolvimento de vários temas e tecnologias e o seu contributo para a melhoria da interoperabilidade dos serviços da universidade.

O objetivo do trabalho desenvolvido no âmbito desta dissertação é a resolução do problema de interoperabilidade do Sistema de Informação de Apoio ao Ensino (SIDE) e agregação de informação académica dispersa em várias fontes de dados. Para alcançar o objetivo, desenvolveu-se e implementou-se uma API REST baseada numa arquitetura de microserviços e recorreu-se a tecnologias de código aberto.

Por último, a API REST desenvolvida é aplicada a um caso de utilização e é testada em termos de performance do seu funcionamento.

Palavras Chave: Interoperabilidade, Linguagens de programação *web*, Serviços *web*, REST, API, Desenho / Documentação automática de API, *Frameworks* de desenvolvimento (REST), Código aberto, Microserviços, Autenticação, Autorização.

Interoperability layer (interface)
of the
Teaching Support Information System (SIDE)

Fernando Manuel Fernandes Rodrigues

Submitted to the University of Trás-os-Montes and Alto Douro
in partial fulfillment of the requirements for the degree of
Master of Science in Informatics Engineering

Abstract — The Teaching Support Information System (SIDE) is a system that has most of its operation on *web* platform. The main objective is to provide information access and to facilitate information management by all those involved in the academic activity of the several departments of the University of Trás-os-Montes and Alto Douro (UTAD).

This dissertation focuses on the evolution and development of several subjects and technologies and their contribution to improving the interoperability of the university services.

The objective of the work developed in the scope of this dissertation is the resolution of the interoperability problem of the Teaching Support Information System (SIDE) and the aggregation of academic information dispersed in several data sources. To achieve the goal, we developed and implemented a REST API based on a microservice architecture and the use of open source technologies. Finally, the developed REST API is applied to a use case and is tested in terms of performance of its operation.

Keywords: Interoperability, *Web* programming languages, Webservices, REST, API, API Design / Automatic documentation, REST development *Frameworks*, Open-source, Microservices, Authentication, Authorization.

Agradecimentos

Quero aproveitar este espaço para deixar umas palavras de agradecimento a todos os que contribuíram direta e indiretamente para a elaboração desta dissertação, muito obrigado a todos.

Gostaria de deixar um reconhecimento pela colaboração e acompanhamento ao Eng. Jorge Borges que é meu chefe e coordenador do Núcleo de Sistemas de Informação da Universidade de Trás-os-Montes e Alto Douro e gostaria também de agradecer aos meus amigos e colegas dos Serviços de Informática e Comunicações da Universidade de Trás-os-Montes e Alto Douro.

Agradeço ao Professor Doutor Luís Filipe Leite Barbosa pela sua orientação, ajuda, amizade e em especial pela sua paciência.

Um agradecimento emotivo e especial à minha mãe por todos os sacrifícios, apoio e dedicação incondicional e quero igualmente deixar um carinho especial e manifestar um sentimento de saudade aos meus avós e bisavós, pois certamente que estariam felizes ao ver que consegui alcançar mais uma etapa da minha vida.

Fernando Manuel Fernandes Rodrigues

Índice geral

Resumo	xi
<i>Abstract</i>	xiii
Agradecimentos	xv
Índice de tabelas	xxiii
Índice de figuras	xxv
Índice de listagens de código	xxvii
Glossário, acrónimos e abreviaturas	xxix
1 Introdução	1
1.1 Introdução	1
1.2 Motivação e objetivos	3
1.3 Organização da dissertação	5
2 Estado da arte	7
2.1 Linguagens de programação <i>web</i> (<i>server-side</i>)	7
2.1.1 Introdução	7
2.1.2 C#	16
2.1.3 C	18
2.1.4 C++	19

2.1.5	D	20
2.1.6	Erlang	21
2.1.7	Go	22
2.1.8	Hack / HHVM	23
2.1.9	Haskell	24
2.1.10	Java	25
2.1.11	JavaScript	27
2.1.12	Node.js	31
2.1.13	Perl	34
2.1.14	PHP	35
2.1.15	Python	36
2.1.16	Ruby	37
2.1.17	Scala	38
2.1.18	XHP	39
2.1.19	Conclusão	39
2.2	Serviços <i>web</i>	41
2.2.1	Introdução	41
2.2.2	CORBA, RMI e COM/DCOM	41
2.2.3	XML-RPC	57
2.2.4	SOAP	63
2.2.5	REST	67
2.2.6	REST vs. SOAP	76
2.2.7	Conclusão	76
2.3	Desenho / documentação automática de API	78
2.3.1	Introdução	78
2.3.2	RAML	80
2.3.3	Slate	80
2.3.4	API blueprint	80
2.3.5	I/O Docs	81
2.3.6	Swagger / OpenAPI	81
2.3.7	SERIN	82
2.3.8	Conclusão	82
2.4	<i>Frameworks</i> de desenvolvimento (REST)	83
2.4.1	Introdução	83
2.4.2	Express	83
2.4.3	Flask-RESTful	84
2.4.4	Phoenix	85
2.4.5	Spring boot	86
2.4.6	Django REST Framework	87
2.4.7	Laravel	87

2.4.8	Zend Framework	88
2.4.9	CakePHP	89
2.4.10	Restlet	90
2.4.11	Spark	90
2.4.12	Sinatra	91
2.4.13	Restify	92
2.4.14	SailsJS	93
2.4.15	LoopBack	94
2.4.16	Gugamarket	94
2.4.17	Grails	95
2.4.18	Conclusão	96
2.5	Microserviços	97
2.5.1	Introdução	97
2.5.2	Arquitetura monolítica e microserviços	97
2.5.3	Conclusão	100
2.6	Sistemas de gestão de base de dados	101
2.6.1	Introdução	101
2.6.2	DBMS Navegacional	101
2.6.3	DBMS Relacional	102
2.6.4	Orientada a objetos	103
2.6.5	NoSQL e NewSQL	103
2.6.6	Conclusão	104
2.7	Métodos de autenticação	105
2.7.1	Introdução	105
2.7.2	Autenticação baseada em sessões	107
2.7.3	Autenticação baseada em <i>tokens</i>	107
2.7.4	Autenticação sem palavra-passe	107
2.7.5	<i>Single Sign On</i> (SSO)	108
2.7.6	<i>Sign-in</i> Social	108
2.7.7	Autenticação de dois fatores (2FA)	108
2.7.8	Conclusão	109
2.8	Controlo de acesso / Auditoria	109
2.8.1	Introdução	109
2.8.2	Controlo de acesso	110
2.8.3	Auditoria	112
2.8.4	Conclusão	113
2.9	Ferramentas de testes de API	114
2.9.1	Introdução	114
2.9.2	Apache JMeter	116
2.9.3	Gatling	116

2.9.4	OpenSTA	117
2.9.5	Siege	118
2.9.6	Conclusão	118
3	Trabalho desenvolvido	121
3.1	Introdução	121
3.2	Descrição dos métodos da API REST	128
3.2.1	Autenticação (_authentication)	129
3.2.2	Esquema (_schema)	130
3.2.3	Tabela (_table)	131
3.2.4	Inserção (_insert)	132
3.2.5	Atualização (_update)	132
3.2.6	Remoção (_delete)	133
3.2.7	Personalização (_custom)	133
3.2.8	Documentação (_documentation)	134
3.3	Documentação	134
3.4	Autenticação	136
3.5	Controlo de acesso	138
3.6	Tratamento de pedidos / filtragem de dados	143
3.7	Auditoria	145
3.8	Segurança	147
3.9	Conclusão	147
4	Casos de utilização	149
4.1	Introdução	149
4.2	Aplicação em Angular Material	149
4.2.1	Autenticação da aplicação	151
4.2.2	Listagem de elementos	151
4.2.3	Ordenação de elementos	154
4.2.4	Paginação de elementos	155
4.2.5	Atualização de elementos	156
4.3	Conclusão	158
5	Testes de carga	159
5.1	Introdução	159
5.2	Configuração Apache JMeter / Siege	160
5.3	Testes - Siege	161
5.3.1	Resultados	161
5.3.2	Análise de resultados	161
5.4	Testes - Apache JMeter	162
5.4.1	Resultados - Execução num processo	162

5.4.2	Resultados - Execução em dois processos	163
5.4.3	Resultados - Execução em quatro processos	164
5.4.4	Análise de resultados	165
5.5	Conclusão	166
6	Conclusão e Trabalho futuro	167
6.1	Conclusão	167
6.2	Trabalho futuro	169
	Referências bibliográficas	171
	Sobre o Autor	183

Índice de tabelas

2.1	Comparação de linguagens de programação <i>web</i> (<i>server-side</i>)	40
2.2	Tipos definidos em XML-RPC.	60
2.3	REST - Métodos HTTP	71
2.4	REST - Alguns códigos de estado HTTP	72
2.5	REST - Desenho de URI	72
2.6	REST - Classificação de métodos HTTP	73
2.7	REST - Cabeçalhos HTTP para controlo de <i>caching</i>	74
2.8	REST - Diretivas de <i>Cache-control</i>	75
2.9	<i>Frameworks</i> de desenvolvimento (REST)	96
2.10	Controlo de acesso em <i>frameworks</i> de desenvolvimento (REST)	112
2.11	Ferramentas de testes de API	115
3.1	Máquinas virtuais instaladas	124
5.1	Resultados de testes de carga com o Siege	161

Índice de figuras

1.1	SIDE - Sistema de Informação de Apoio ao Ensino	2
2.1	Arquitetura Cliente-servidor	9
2.2	Arquitetura Peer-to-peer	9
2.3	Modelo de eventos tradicional bloqueante	32
2.4	Modelo assíncrono de eventos não bloqueante	33
2.5	Arquitetura CORBA	43
2.6	Arquitetura de gestão de objetos (OMA)	45
2.7	Arquitetura RMI	50
2.8	Arquitetura DCOM	54
2.9	XML-RPC	58
2.10	SOAP	63
2.11	Estrutura da mensagem SOAP	66
2.12	Serviço <i>web</i> REST	68
2.13	Formato do pedido REST	70
2.14	Formato da resposta REST	71
2.15	Arquitetura monolítica e microserviços	98
2.16	Microserviços	99

2.17	Relação entre o controlo de acesso e as outras funções de segurança	111
3.1	Modelo implementado	122
3.2	Interface <i>web</i> para consulta de documentação em OpenApi.	136
3.3	Estrutura da tabela de <i>authentication</i>	137
3.4	Estrutura da tabela de <i>role</i>	140
3.5	Estrutura da tabela de <i>permission</i>	141
4.1	Formulário de autenticação da aplicação <i>sideapp</i>	151
4.2	Listagem de elementos na aplicação <i>sideapp</i>	152
4.3	Ordenação de elementos na aplicação <i>sideapp</i>	154
4.4	Paginação de elementos na aplicação <i>sideapp</i>	155
4.5	Seleção de elemento na <i>sideapp</i>	156
4.6	Atualização de elemento na <i>sideapp</i>	157
4.7	Elemento atualizado na <i>sideapp</i>	158
5.1	Configuração Apache JMeter	160
5.2	Tempo de resposta para sistema a ser executado num processo	162
5.3	Tempo de resposta para sistema a ser executado em dois processos	163
5.4	Tempo de resposta para sistema a ser executado em quatro processos	164
5.5	Gráfico - Tempo de resposta	165

Índice de listagens de código

2.1	Pedido XML-RPC	58
2.2	Estrutura <i><struct></i> do XML-RPC	60
2.3	Elemento <i><array></i> do XML-RPC	60
2.4	Resposta do XML-RPC	61
2.5	Resposta com falhas em XML-RPC	61
2.6	Exemplo de estrutura de pedido SOAP	63
2.7	Exemplo de código de pedido SOAP	64
2.8	Exemplo de estrutura de resposta SOAP	64
2.9	Formato JSON	69
2.10	Formato XML	69
3.1	Ficheiro de configuração global	123
3.2	Estrutura <i>return_data</i>	126
3.3	Método <i>clean_return_data</i>	127
3.4	Estrutura do módulo para método personalizado	127
3.5	Segmento da documentação implementada em OpenApi	134
3.6	Segmento da <i>hash</i> estática de controlo de acesso	138
3.7	Estrutura da tabela de sessão	142
3.8	Dados obtidos sem agregação	143

3.9	Dados obtidos com agregação	144
3.10	Auditoria a método da API REST em nível 10 (TRACE)	145
3.11	Exemplo de <i>return_data</i>	147
4.1	Instalação do Node.js e npm	150
4.2	Instalação do Angular CLI	150
4.3	Criação da aplicação <i>sideapp</i> e instalação do Angular Material	150
4.4	Criação dos serviços de dados através da Angular CLI	150
4.5	Serviço de dados para listagem de elementos	152
4.6	Serviço de dados para obtenção de dados de um elemento	156
4.7	Serviço de dados para atualização de dados de um elemento	157
5.1	Comando executado para teste de carga com o Siege	160
5.2	Configurações para sistema a ser executado num processo	162
5.3	Configurações para sistema a ser executado em dois processos	163
5.4	Configurações para sistema a ser executado em quatro processos	164

Glossário, acrónimos e abreviaturas

Glossário de termos

Aplicação — Utilização de técnicas informáticas para resolver um problema de aplicação, que conduz normalmente ao desenvolvimento ou à utilização de programas de aplicação como, por exemplo, folhas de cálculo ou processadores de texto. Nota: O termo “aplicação informática” é muitas vezes entendido no sentido de “programa de aplicação”.

Applet — *Software* que executa uma atividade específica, dentro (do contexto) de outro programa maior (como por exemplo um navegador *web*), geralmente como um *Plugin*. O termo foi introduzido pelo *AppleScript* em 1993.

Assertion — Predicado que é inserido no programa para verificar uma condição que o programador supõe que seja verdadeira em determinado ponto.

Backend — Camada de acesso a dados de um *software*.

Backup — Qualificativo de um processo, técnica ou equipamento utilizado para ajudar a recuperar dados perdidos ou destruídos ou para manter um sistema em funcionamento. Nota: No contexto do *software* utilizado para realizar a

salvaguarda de ficheiros, obtemos as chamadas “cópias de segurança” (*backup copies*). No contexto de equipamento que permita redundância, temos por exemplo “fontes de alimentação de reserva” (*backup power supplies*) ou mesmo “discos de reserva” (*backup disks*).

Bottleneck — Designação do componente que limita o desempenho ou a capacidade de todo um sistema, que se diz ter um estrangulamento.

Cache — Componente de *hardware* ou *software* que armazena dados, para que pedidos futuros aos mesmos recursos sejam atendidos mais rapidamente.

Classe — Descrição que abstrai um conjunto de objetos com características similares. Mais formalmente, é um conceito que encapsula abstrações de dados e procedimentos que descrevem o conteúdo e o comportamento de entidades do mundo real, representadas por objetos.

Cliente — Termo empregado em computação e que representa uma entidade que consome os serviços de uma outra entidade servidora, em geral através da utilização de uma rede de computadores numa arquitetura cliente-servidor.

Clusters — Computadores fracamente ou fortemente ligados que trabalham em conjunto.

Código aberto, ou open-source em inglês — *Software* de computador com o seu código fonte disponibilizado e licenciado com uma licença de código aberto no qual o direito do autor fornece o direito de estudar, modificar e distribuir o *software* de forma gratuita para qualquer um e para qualquer finalidade.

Cookie — Arquivo de computador ou pacote de dados enviados por um sítio de Internet para o navegador do utilizador, quando o utilizador visita o sítio de Internet.

CSP-style — Linguagem formal para descrever padrões de interação em sistemas concorrenciais.

Data binding — Ligação de dados é uma técnica geral que une duas fontes de dados / informações e as mantém em sincronia em um processo que estabelece uma ligação entre interface de utilizador da aplicação e a lógica de negócio.

Framework — Conjunto de classes implementadas em uma linguagem de programação específica, utilizadas para auxiliar no desenvolvimento de *software*.

Frontend — Camada de apresentação de um *software*.

Garbage collector — Processo usado para a automação da gestão de memória.

Interface — Fronteira que facilita a comunicação entre o computador e o seu utilizador (interface gráfica ou textual), entre duas aplicações ou ainda entre dois dispositivos.

JSON — Formato compacto, de padrão aberto independente, de troca de dados simples e rápida (*parsing*) entre sistemas, especificado por Douglas Crockford em 2000, que utiliza texto legível por humanos, no formato atributo-valor (natureza auto-descritiva).

Linguagem de programação compilada — O código fonte, nessa linguagem, é executado diretamente pelo sistema operativo ou pelo processador, após ser traduzido por meio de um processo chamado compilação, usando um programa de computador chamado compilador, para uma linguagem de baixo nível.

Linguagem de programação interpretada — Linguagem de programação em que o código fonte nessa linguagem é executado por um programa de computador chamado interpretador, que em seguida é executado pelo sistema operativo ou processador.

HTTP — Protocolo utilizado para transferência de páginas *web* de hipertexto: é o protocolo de comunicação da *World Wide Web*(WWW).

HTML — Linguagem de marcação de hipertexto utilizada para escrever páginas de documentos para a *World Wide Web*(WWW), que possibilita a preparação

de documentos com gráficos e hiperligações, para visualização em sistemas compatíveis com a WWW.

Malware — Programas informáticos destinados a perturbar, alterar ou destruir todos ou parte dos módulos indispensáveis ao bom funcionamento de um sistema informático.

Marshalling — Processo de transformação da representação de memória de um objeto em um formato de dados compatível para armazenamento ou transmissão e é utilizado tipicamente quando os dados precisam ser movimentados entre diferentes partes de uma aplicação ou entre aplicações.

Máquina virtual — Consiste num *software* que executa programas como um computador real, também conhecido por processo de virtualização.

Metadados — Dados / informações que fornecem informações sobre outros dados.

Middleware — *Software* de interface que permite interação de diferentes aplicações informáticas, geralmente sendo executadas em diferentes plataformas de equipamento para troca de dados.

OAuth — Padrão aberto para autenticação, comumente utilizado para permitir que os utilizadores da Internet se possam autenticar em sítios de terceiros utilizando as suas contas da Google, Facebook, Microsoft, Twitter, etc.

Objeto — Referência a um endereço da memória que possui um valor. Um objeto pode ser uma variável, função ou estrutura de dados. Com a introdução da programação orientada a objetos, a palavra objeto refere-se a uma instância de uma classe.

P2P ou peer-to-peer — Rede de computadores que não tem clientes e servidores fixos, mas um conjunto de nós “equivalentes”, que funcionam ou como clientes ou como servidores dos outros nós da rede. Nota: O modelo *peer-to-peer* opõe-se ao modelo cliente-servidor.

Pipeline — Cadeia de elementos de processamento (processos, *threads*, corrotinas, funções, etc.), organizados de modo que a saída de cada elemento seja a entrada do próximo.

Plugin — Componente de *software* que adiciona um recurso específico a um programa de computador existente.

Protocolo — Em tecnologias da informação e da comunicação, conjunto de regras que formam uma linguagem utilizada pelos computadores para intercomunicação.

Proxy — Servidor situado entre uma aplicação cliente, como um programa de navegação, e o servidor real. O servidor intermediário interceta todos os pedidos para o servidor real, para ver se ele próprio os pode satisfazer e em caso negativo envia-os para o servidor real. Nota: O servidor intermediário tem dois objetivos principais: acelerar a satisfação dos pedidos do utilizador e filtrar conteúdos.

Servidor — *Software* ou computador, com sistema de computação centralizada que fornece serviços a uma rede de computadores, chamados de clientes.

Sistema operativo — *Software* de base de um computador destinado a controlar a execução de programas, a comunicação entre dispositivos e programas, assegurando as operações de entrada-saída, a atribuição de recursos aos diferentes processos, o acesso às bibliotecas de programas e aos ficheiros, assim como a compatibilidade dos trabalhos. Nota: O sistema operativo é o *software* mais importante a correr num computador.

Scripting — *Scripting* ou linguagem de *script* é uma linguagem de programação que suporta *scripts*, programas escritos para um ambiente especial *runtime* que automatiza a execução de tarefas, que poderiam alternativamente ser executadas uma a uma por um operador humano.

Spyware — *Software* que se aproveita da ligação Internet do utilizador para recolher e transmitir alguns dos respetivos dados pessoais, sem seu

conhecimento e autorização. Nota: O *software* espião, geralmente incorporado como um componente oculto de *software* gratuito (*freeware*) ou *software* de utilização condicionada (*shareware*) disponível na Internet, é indetectável pelo utilizador; é no entanto possível localiza-lo e suprimi-lo com programas específicos, concebidos expressamente para esse fim.

TCP/IP — Conjunto dos protocolos de comunicação utilizados na Internet para gerir a circulação de dados na rede, fragmentando a informação na origem sob a forma de pacotes de dados e reunindo novamente os pacotes no destino, assim como controlando eventuais erros de transmissão.

Vírus — Classe de *software* mal-intencionado que tem a capacidade de se auto-replicar e “infetar” partes do sistema operativo ou de outros programas, com o intuito de causar a perda ou alteração da informação. Nota: Os vírus de computador têm um comportamento semelhante em certos aspetos ao dos vírus biológicos, daí a razão do nome que lhes foi atribuído.

Windows — Família de sistemas operativos desenvolvidos, comercializados e vendidos pela Microsoft.

World wide web — Sistema baseado na utilização de hipertexto, que permite a pesquisa de informação na Internet, o acesso a essa informação e a sua visualização. Utiliza a linguagem HTML e o protocolo HTTP para apresentar e transmitir texto, gráficos, som e vídeo, e incorpora também outros protocolos de Internet tradicionais como Gopher, FTP, WAIS e Telnet. Nota: A *Web* foi inventada por Tim Berners-Lee e Robert Cailliau para o Centro Europeu de Pesquisa Nuclear na Suíça.

Lista de acrónimos

Sigla	Expansão
ABAC	<i>Attribute-based Access Control.</i>
ACID	<i>Atomicity, Consistency, Isolation, Durability.</i>
AJAX	<i>Asynchronous Javascript and XML.</i>
API	<i>Application Programming Interface.</i>
ATOM	<i>Atom Syndication Format / Atom Publishing Protocol.</i>
AWS	<i>Amazon Web Services.</i>
CLI	<i>Common Language Infrastructure.</i>
CLR	<i>Common Language Runtime.</i>
COM	<i>Microsoft Component Object Model.</i>
CORBA	<i>Common Object Request Broker Architecture.</i>
CRUD	<i>Create, Read, Update e Delete.</i>
DAC	<i>Discretionary Access Control.</i>
DBA	<i>Database administrator.</i>
DBMS	<i>Database management system.</i>
DCE/RPC	<i>Distributed Computing Environment / Remote Procedure Calls.</i>
DDCF	<i>Distributed Document Components.</i>
DLL	<i>Dynamic-link library.</i>
DOM	<i>Document Object Model.</i>
DSL	<i>Domain-Specific Language.</i>
EAI	<i>Enterprise Application Integration.</i>
EBAC	<i>Emotion-based Access Control.</i>
ECMA	<i>European Computer Manufacturers Association.</i>
FTP	<i>File Transfer Protocol.</i>
GUI	<i>Graphical user interface.</i>

Sigla	Expansão
HATEOAS	<i>Hypermedia as the Engine of Application State.</i>
HBAC	<i>History-Based Access Control.</i>
HTAP	<i>Hybrid transaction/analytical processing.</i>
HTTP	<i>Hypertext Transfer Protocol.</i>
HTTPS	<i>Hyper Text Transfer Protocol Secure.</i>
IAAS	<i>Infrastructure as a service.</i>
IBAC	<i>Identity-Based Access Control.</i>
IDE	<i>Integrated Development Environment.</i>
IDL	<i>Interface Description Language.</i>
IDP	<i>Identity provider.</i>
IIOB	<i>Internet Inter-ORB Protocol.</i>
IMS	<i>Information Management System.</i>
IOT	<i>Internet Of Things.</i>
ISO	<i>International Organization for Standardization.</i>
JDBC	<i>Java Database Connectivity.</i>
JMS	<i>Java Message Service.</i>
LBAC	<i>Lattice-based access control.</i>
LDAP	<i>Lightweight Directory Access Protocol.</i>
LLVM	<i>Low Level Virtual Machine.</i>
LRPC	<i>Lightweight Remote Procedure Calls.</i>
JIT	<i>Just-in-time.</i>
JSON	<i>JavaScript Object Notation.</i>
JRMP	<i>Java Remote Method Protocol.</i>
JVM	<i>Java Virtual Machine.</i>
JWT	<i>JSON Web Tokens.</i>
MAC	<i>Mandatory Access Control.</i>
MEP	<i>Message Exchange Pattern.</i>

Sigla	Expansão
MEAN	<i>MongoDB, Express.js, AngularJS e Node.js.</i>
MIDL	<i>Microsoft Interface Definition Language.</i>
MIME	<i>Multipurpose Internet Mail Extensions.</i>
MOM	<i>Message-Oriented Middleware.</i>
MTS	<i>Microsoft Transaction Server.</i>
MVC	<i>Model-view-controller.</i>
OAuth	<i>Open Authentication.</i>
ODBC	<i>Open Database Connectivity.</i>
ODBMS	<i>Object Database Management System.</i>
OLE	<i>Object Linking and Embedding.</i>
OLEDB	<i>Object Linking and Embedding, Database.</i>
OMA	<i>Object Management Architecture.</i>
ORB	<i>Object Request Broker.</i>
ORM	<i>Object-relational mapping.</i>
ORPC	<i>Object Remote Procedure Call.</i>
OTP	<i>One Time Passord.</i>
PAAS	<i>Platform as a service.</i>
PHP	<i>Hypertext Preprocessor.</i>
PIN	<i>Personal Identification Number.</i>
POA	<i>Portable Object Adapter.</i>
POSIX	<i>Portable Operating System Interface.</i>
RBAC	<i>Role-Based Access Control.</i>
RDBMS	<i>Relational Database Management System.</i>
RFC	<i>Request for Comments.</i>
REST	<i>Representational State Transfer.</i>
RIA	<i>Rich Internet application.</i>
RMI	<i>Remote Method Invocation.</i>

Sigla	Expansão
RPC	<i>Remote procedure call.</i>
RTL	<i>Right-to-left.</i>
RSS	<i>Rich Site Summary / Really Simple Syndication.</i>
SAML	<i>Security Assertion Markup Language.</i>
SIDE	<i>Sistema de Informação de Apoio ao Ensino.</i>
SIG	<i>Sistema de Informação Geográfica.</i>
SIGACAD	<i>Sistema Integrado de Gestão Académica da Universidade de Aveiro.</i>
SLA	<i>Service Level Agreement.</i>
SMS	<i>Short Message Service.</i>
SMTP	<i>Simple Mail Transfer Protocol.</i>
SOA	<i>Service-Oriented Architecture.</i>
SOAP	<i>Simple Object Access Protocol.</i>
SP	<i>Service Provider.</i>
SPA	<i>Single-page application.</i>
SQL	<i>Structured Query Language.</i>
SSL	<i>Secure Sockets Layer.</i>
SSO	<i>Single Sign On.</i>
UDDI	<i>Universal Description, Discovery and Integration.</i>
URI	<i>Uniform Resource Identifier.</i>
URL	<i>Uniform Resource Locator.</i>
VM	<i>Virtual Machine.</i>
VPN	<i>Virtual Private Network.</i>
WWW	<i>World wide web.</i>
WSDL	<i>Web Services Description Language.</i>
XML	<i>eXtensible Markup Language.</i>

Sigla	Expansão
XML-RPC	<i>eXtensible Markup Language - Remote Procedure Call protocol.</i>
XSS	<i>Cross-site scripting.</i>
YAML	<i>YAML Ain't Markup Language.</i>

Lista de abreviaturas

Abreviatura	Significado(s)
e.g.	por exemplo
et al.	e outros (autores)
etc.	etecetera, outros
i.e.	isto é, por conseguinte
vid.	veja-se, ver
vs.	versus, por comparação com



Introdução

1.1 Introdução

O Sistema de Informação de Apoio ao Ensino (SIDE)([Barbosa et al., 2011](#)) é um sistema que assenta a maior parte do seu funcionamento na plataforma *web* e tem como objetivo principal disponibilizar o acesso à informação decorrente da atividade académica dos vários departamentos pertencentes à Universidade de Trás-os-Montes e Alto Douro (UTAD)¹. Em perspetiva mais abrangente, a plataforma centra as suas operações no(a):

- Apoio e dinamização do processo de ensino e aprendizagem, indo de encontro às necessidades de integração dos fluxos de informação de diversos sistemas de informação universitários;
- Divulgação e comunicação entre os vários intervenientes no processo educativo e de avaliação, dando resposta aos desafios da instituição de ensino superior;
- Aumento simultaneamente da capacidade formativa, bem como da eficiência

¹Universidade de Trás-os-Montes e Alto Douro é uma universidade pública Portuguesa situada na região de Trás-os-Montes e Alto Douro (Vila Real). Foi fundada em 1986, tendo origem no Instituto Politécnico de Vila Real (fundado em 1973).

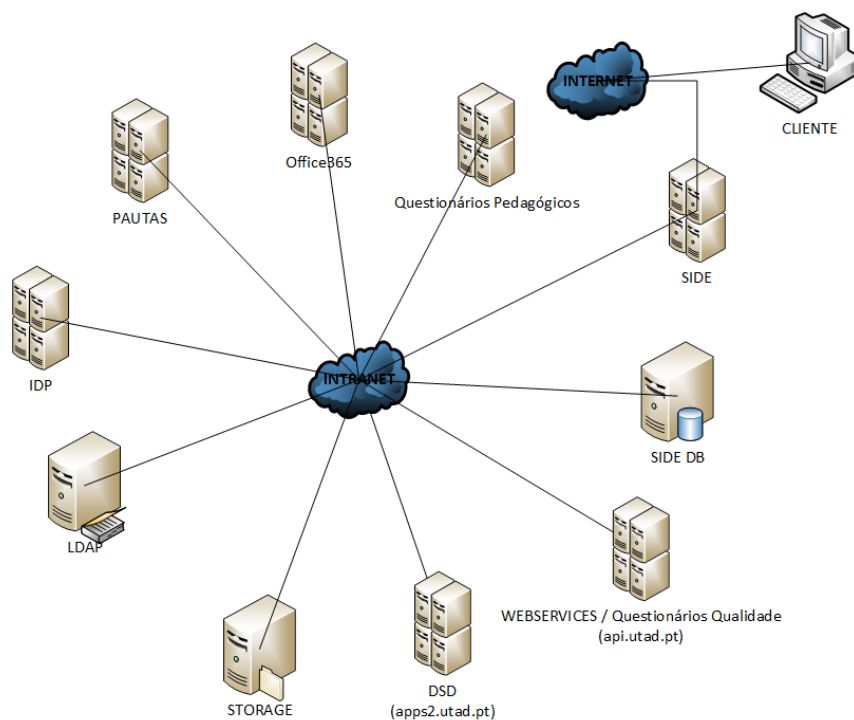


Figura 1.1 – SIDE - Sistema de Informação de Apoio ao Ensino

do processo de gestão académica;

Numa perspetiva mais específica são colocados vários níveis de acesso ao utilizador, permitindo a:

- Gestão e armazenamento de dados recorrentes do funcionamento dos cursos (inscrições, avaliações, horários, exames, ocupação de salas, calendário, sumários, faltas, etc.);
- Divulgação e manutenção de informação dos cursos / departamentos / escolas da UTAD;
- Utilização de novas tecnologias associadas ao ensino, através de fornecimento de informação aos alunos dos respetivos cursos (disciplinas, recursos bibliográficos, avaliações, inscrições, *downloads*, etc.);
- Processamento de informação que possibilite e sirva de apoio a decisões dos órgãos de gestão;

Numa perspetiva de desenvolvimento, o sistema visa ser flexível e modular, permitindo incorporar facilmente novos módulos / serviços ajustados às necessidades dos vários intervenientes da área educativa da UTAD, tais como: serviços de alunos (inscrições, consultas, submissões, etc.), serviços de docentes (gestão de conteúdos pedagógicos / administração da unidade curricular), serviços de direção de curso (gestão de informação útil ao apoio decisório), gestão da informação estrutural / administrativa do curso, etc. Também existe uma atenção muito relevante durante a implementação no que diz respeito ao acesso aos dados, salvaguardando e protegendo os dados de acordo com a legislação em vigor. Em estilo de conclusão podemos afirmar que o SIDE desempenha um papel muito importante, senão fundamental na interligação dos vários componentes do processo educativo / académico da Universidade de Trás-os-Montes e Alto Douro (UTAD). O SIDE é um sistema que se encontra ligado direta e indiretamente a outros sistemas de informação (vid. figura 1.1), sendo que grande parte de informação académica é importada indiretamente do SIGACAD², que atualmente é a aplicação utilizada pela UTAD para a gestão de informação académica.

1.2 Motivação e objetivos

A interoperabilidade é definida como a capacidade de um sistema (informatizado ou não) para interagir e comunicar com outro de forma transparente (ou o mais próximo disso)(Silva, 2004), ou habilidade de transferir e utilizar informações de maneira uniforme e eficiente entre várias organizações e sistemas de informação (semelhantes ou não). A motivação e contribuição desta dissertação estão identificadas no trabalho de desenvolvimento de uma camada intermédia visando a transparência no acesso e a necessidade de interligar e reestruturar o Sistema de Informação de Apoio ao Ensino (SIDE). Desenhar / implementar uma API³ utilizando

²Sistema Integrado de Gestão Académica da Universidade de Aveiro.

³*Application Programming Interface.*

uma *framework*⁴ para desenvolver serviços *web* que possam ser executados em diferentes ambientes (IAAS⁵(Loeffler, 2011), PAAS⁶(Loeffler, 2011), *Application containers*⁷(Hogg, 2014), etc.) e agregando informação (*views*, *database links*, etc.) proveniente de várias fontes, de forma a construir objetos que facilitem a interligação entre os vários sistemas de informação e o SIDE, é identificada como a principal contribuição deste trabalho.

Este trabalho tem como objetivo a especificação, implementação e utilização de uma interface de interoperabilidade do SIDE. A interface de interoperabilidade será composta por várias camadas de diferente nível de complexidade / abstração, facilitando futuras atividades de migração e reestruturação do sistema atualmente existente. Na etapa final pretende-se desenvolver vários casos de utilização da interface de interoperabilidade (aplicação *web*, móvel, etc.). A interface de interoperabilidade do Sistema de Informação de Apoio ao Ensino (SIDE) deverá permitir:

- Alterar a camada de mais baixo nível (acesso aos dados), sem que para isso seja necessário alterar as camadas de mais alto nível, facilitando o trabalho envolvido em futuras migrações;
- Estender e desenvolver novas funcionalidades;
- Testar novas soluções no que toca à camada aplicacional e de acesso aos dados;
- Gerir vários tipos de perfis de acesso aos dados;

⁴Conjunto de classes implementadas em uma linguagem de programação específica, utilizadas para auxiliar o desenvolvimento de *software*.

⁵*Infrastructure as a service*.

⁶*Platform as a service*.

⁷Recursos do sistema operativo no qual o núcleo (*kernel*) permite a existência várias instâncias isoladas de espaço de utilizador (*userspace*), tais instâncias são chamadas *containers*.

1.3 Organização da dissertação

Esta dissertação está organizada em seis capítulos. Este capítulo inicial possui uma nota introdutória e estabelece os motivos e objetivos desta dissertação. No segundo capítulo, será revisto o estado da arte relativo às tecnologias e temas abordados nesta dissertação, a saber: linguagens de programação *web* (*server-side*), serviços *web*, desenho e documentação automática de API, *frameworks* de desenvolvimento (REST), microserviços, sistemas de gestão de base de dados, métodos de autenticação, controlo de acesso / auditoria e ferramentas de testes de API. O terceiro capítulo aborda o trabalho desenvolvido para a elaboração da dissertação, em que é implementada uma API REST. No quarto capítulo é discutida a implementação de um caso de utilização do trabalho desenvolvido, em que se concretiza uma aplicação que utiliza a API REST. No penúltimo capítulo são apresentados testes de carga realizados à API implementada e são estabelecidas algumas conclusões ao trabalho desenvolvido. O último capítulo centra-se na abordagem de notas conclusivas e são estabelecidas bases para trabalho futuro.



Estado da arte

2.1 Linguagens de programação *web* (*server-side*)

2.1.1 Introdução

Nesta secção será abordada uma breve descrição da origem da Internet, por conseguinte, para entendermos a sua origem e evolução será necessário recuar no tempo e focar alguns marcos importantes da história da comunicação. Um dos primeiros marcos foi a ARPANET¹ (Abbate, 1994) que surgiu durante a década de 1960 e juntamente com o protocolo² de comunicação TCP/IP³ tornaram-se as fundações da Internet atual. Em 1990 Tim Berners-Lee (cientista do CERN⁴)

¹Advanced Research Projects Agency Network.

²Em tecnologias da informação e da comunicação, conjunto de regras que formam uma linguagem utilizada pelos computadores para intercomunicação.

³Conjunto dos protocolos de comunicação utilizados na Internet para gerir a circulação de dados na rede, fragmentando a informação na origem sob a forma de pacotes de dados e reunindo novamente os pacotes no destino, assim como controlando eventuais erros de transmissão.

⁴A Organização Europeia para a pesquisa Nuclear, conhecida como CERN é o maior laboratório de física de partículas do mundo, localizado em Meyrin, na região em Genebra, na fronteira Franco-Suíça.

inventa o HyperText Markup Language (HTML⁵), Berners-Lee também foi o responsável pelo desenvolvimento do protocolo HTTP⁶ e em 1991 a World Wide Web é apresentada ao público pelo CERN (Zimmerman, 2012), que juntamente com o protocolo HTTP apareceram como uma necessidade para ligar várias universidades de forma a aumentar o trabalho acadêmico colaborativo.

Os primeiros computadores eram programados em código máquina⁷, mais tarde surgiram as primeiras linguagens de programação⁸, tendo Ada Lovelace (Aiello, 2016) desenvolvido o primeiro trabalho em linguagem de programação. As linguagens de programação foram evoluindo e com o surgimento de novas tecnologias e novos conceitos as linguagens de programação também ganharam orientações mais específicas, conforme o contexto da sua utilização.

Com a Internet surgiram um conjunto de arquiteturas, a saber, a arquitetura cliente-servidor que é uma arquitetura de rede em que cada computador ou processo na rede⁹ desempenha o papel de cliente ou de servidor (vid. figura 2.1). Servidores normalmente são máquinas ou processos com papéis dedicados: servidor de ficheiros (*file-server*), servidor de impressão (*print-server*), servidor de rede (*network-server*), etc. Clientes são computadores ou estações de trabalho onde os utilizadores executam aplicações¹⁰. Clientes necessitam de recursos geridos por servidores tais como ficheiros (armazenamento), dispositivos, poder de processamento, etc.

⁵Linguagem de marcação de hipertexto utilizada para escrever páginas de documentos para a *World Wide Web* (WWW), que possibilita a preparação de documentos com gráficos e hiperligações, para visualização em sistemas compatíveis com a WWW.

⁶Protocolo utilizado para transferência de páginas *Web* de hipertexto: é o protocolo de comunicação da *World Wide Web* (WWW).

⁷Código máquina é uma linguagem de computador que é diretamente interpretada pelo CPU de um computador (unidade central de processamento) e é a linguagem na qual todos os programas devem ser convertidos antes de poderem ser executados.

⁸Uma linguagem de programação é um método padronizado para comunicar instruções para um computador. (Dershem and Jipping, 1990)

⁹Rede formada por um conjunto de computadores e seus periféricos interconectados uns aos outros.

¹⁰Utilização de técnicas informáticas para resolver um problema de aplicação, que conduz normalmente ao desenvolvimento ou à utilização de programas de aplicação como, por exemplo, folhas de cálculo ou processadores de texto. Nota: O termo “aplicação informática” é muitas vezes tomado no sentido de “programa de aplicação”.

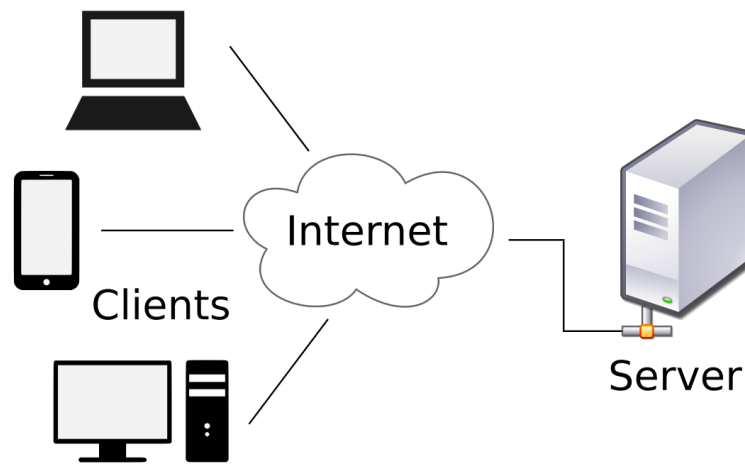


Figura 2.1 – Arquitetura Cliente-servidor (Vignoni, 2011)

Outro tipo de arquitetura muito utilizado em redes de computadores é a arquitetura *peer-to-peer*¹¹ e distingue-se por cada nó (*peer*) da rede desempenhar idêntico papel.

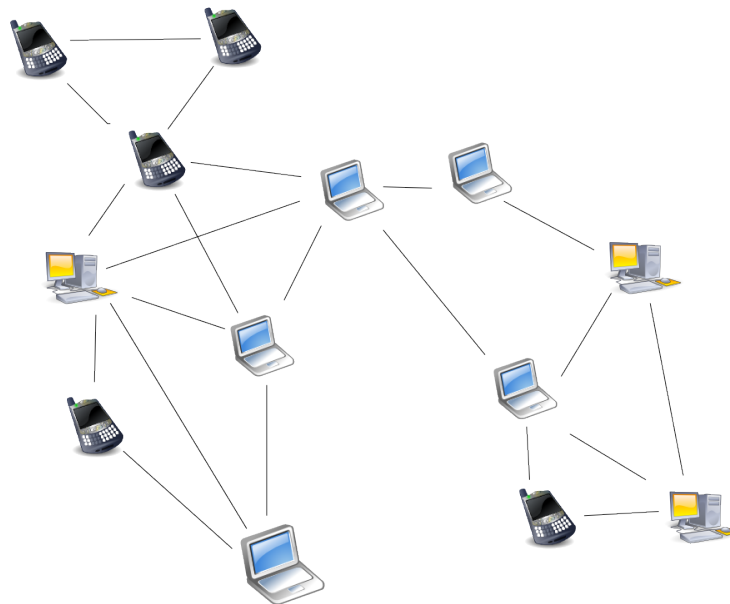


Figura 2.2 – Arquitetura Peer-to-peer (Mesoderm, 2013)

¹¹Rede de computadores que não tem clientes e servidores fixos, mas um conjunto de nós “equivalentes”, que funcionam ou como clientes ou como servidores dos outros nós da rede.

Ambas as arquiteturas de rede são muito utilizadas, existindo vantagens e desvantagens na sua utilização.

Vantagens da arquitetura Cliente-servidor:

- Facilidade de administração do servidor devido à sua natureza centralizada;
- Resolução mais fácil de problemas de segurança como vírus¹², *malware*¹³, *spyware*¹⁴;
- Sistema de *backups*¹⁵ mais simplificado, pois os *backups* críticos são feitos a nível de servidor;
- etc.

Vantagens da arquitetura *Peer-to-peer*:

- Facilidade de instalação e configuração de nós (*peers*);
- Todos os conteúdos / recursos são partilhados por todos os nós, ao contrário da arquitetura cliente-servidor em que conteúdos / recursos são partilhados pelos servidores;

¹²Classe de *software* mal-intencionado que tem a capacidade de se auto-replicar e “infetar” partes do sistema operativo ou de outros programas, com o intuito de causar a perda ou alteração da informação. Nota: Os vírus de computador têm um comportamento semelhante em certos aspetos ao dos vírus biológicos, daí a razão do nome que lhes foi atribuído.

¹³Programas informáticos destinados a perturbar, alterar ou destruir todos ou parte dos módulos indispensáveis ao bom funcionamento de um sistema informático.

¹⁴*Software* que se aproveita da ligação Internet do utilizador para recolher e transmitir alguns dos respetivos dados pessoais, sem seu conhecimento e autorização. Nota: O *software* espião, geralmente incorporado como um componente oculto de *software* gratuito (*freeware*) ou *software* de utilização condicionada (*shareware*) disponível na Internet, é indetectável pelo utilizador; é no entanto possível localizá-lo e suprimi-lo com programas específicos, concebidos expressamente para esse fim.

¹⁵Qualificativo de um processo, técnica ou equipamento utilizado para ajudar a recuperar dados perdidos ou destruídos ou para manter um sistema em funcionamento. Nota: No contexto do *software* utilizado para realizar a salvaguarda de ficheiros, obtemos as chamadas “cópias de segurança” (*backup copies*). No contexto de equipamento que permita redundância, temos por exemplo “fontes de alimentação de reserva” (*backup power supplies*) ou mesmo “discos de reserva” (*backup disks*).

- Arquitetura mais fiável, devido à sua natureza descentralizada. Se um dos nós falha, a rede não é afetada ao contrário de uma arquitetura cliente-servidor em que se um servidor falha então toda a rede é afetada;
- Não necessita um administrador a tempo inteiro, cada nó (*peer*) possui o seu próprio administrador que gere todos os conteúdos / recursos partilhados.
- Custo de manutenção deste tipo de arquitetura muito mais baixo que o custo de manutenção de uma arquitetura cliente-servidor;
- etc.

Na arquitetura cliente-servidor, o cliente e o servidor são conhecidos também como anfitriões (*hosts*). Na *web*, o cliente (navegador *web*) inicia o contacto com o servidor requisitando um serviço (HTTP *request*), i.e., o servidor providencia a página *web* (HTML) requisitada através do protocolo HTTP (HTTP *response*). As páginas *web* fornecidas podem ser estáticas ou dinâmicas. Assim, páginas dinâmicas do lado do servidor são páginas que são geradas por um servidor aplicacional, que podem utilizar linguagens de *scripting*¹⁶ ou não do lado do servidor, e conjuntamente com determinadas parametrizações gerar a página *web* e também a preparar para processamento opcional posterior do lado do cliente.

Posteriormente, serão abordadas algumas definições e conceitos necessários para enquadrar os assuntos tratados nos parágrafos seguintes. Assim, em termos informáticos, erros de exceção (e.g., *trap*, *exception*, *fault*) são tipicamente um tipo de interrupção síncrona causada por uma condição excecional (e.g., divisão por zero, acesso inválido de memória, vetor fora de limite, etc.). Em contrapartida, existem os erros *untrapped* (e.g., *buffer overflow*¹⁷, etc.). Os erros de exceção (*trapped errors*) param imediatamente a execução de um programa, enquanto que

¹⁶*Scripting* ou linguagem de *script* é uma linguagem de programação que suporta *scripts*, programas escritos para um ambiente especial *runtime* que automatiza a execução de tarefas, que poderiam alternativamente ser executadas uma a uma por um operador humano.

¹⁷Transbordamento de dados ou estouro de *buffer* (região de memória física utilizada para armazenar temporariamente dados enquanto estão a ser movidos de um lugar para outro) é uma anomalia onde um programa, ao escrever dados num *buffer*, ultrapassa os limites do *buffer* e sobrescreve a memória adjacente.

nos erros *untrapped* o programa pode continuar a execução sem problema aparente. O sistema de tipos de dados (estilo de tipagem) é um conjunto de regras que atribui uma propriedade chamada tipo aos vários blocos de um programa de computador, como variáveis, expressões, funções ou módulos. Segundo alguns autores, o propósito fundamental de um sistema de tipos de dados é evitar a ocorrência de erros de execução durante a execução de um programa.(Cardelli, 1996, p 1)

Mais à frente nesta secção várias linguagens de programação serão classificadas quanto ao sistema de tipos de dados, pelo que, serão abordados alguns sistemas:

Linguagem de programação estática – Linguagem de programação onde o bom comportamento é determinado antes da execução.(Cardelli, 1996)

Linguagem de programação dinâmica – Linguagem de programação onde o bom comportamento é verificado durante a execução.(Cardelli, 1996)

Linguagem de programação forte – Linguagem de programação onde erros proibidos¹⁸ não podem ocorrer durante a execução (depende da definição de erros proibidos).(?)

Linguagem de programação fraca – Linguagem de programação que é estática, mas não fornece garantias de falta de erros de execução.(Cardelli, 1996)

Linguagem de programação segura – Linguagem de programação onde não podem ocorrer erros *untrapped*.(Cardelli, 1996)

Linguagem de programação nominativa / nominal – Linguagem de programação em que compatibilidade e equivalência de tipos de dados é determinada por declarações explícitas e / ou nome dos tipos de dados.

Linguagem de programação inferida – Linguagem de programação em que o tipo de dados é deduzido do contexto em tempo de compilação ou permite a declaração dinâmica na qual a variável é declarada e pode receber um valor de qualquer tipo de dados em tempo de execução.

Linguagem de programação manifesta – Linguagem de programação onde é feita a identificação explícita pelo programador do tipo de dados de cada variável declarada.

Linguagem de programação estrutural – Linguagem de programação em

¹⁸Especificados pelo arquiteto da linguagem.

que a compatibilidade de tipo e equivalência são determinadas pela estrutura ou definição atual do tipo, e não por outras características, como seu nome ou local de declaração.

Linguagem de programação “*duck typing*” – Linguagem de programação em que os métodos e propriedades de um objeto determinam a semântica válida, em vez de sua herança de uma classe particular ou implementação de uma interface explícita.

Linguagem de programação gradual – Linguagem de programação em que algumas variáveis e expressões podem receber tipos e a sua validade é verificada em tempo de compilação (linguagem estática) e algumas expressões podem ser deixadas sem tipo de dados e eventuais erros de tipo são reportados em tempo de execução (linguagem dinâmica). Este sistema de tipos de dados permite que os programadores escolham o sistema de tipo de dados mais apropriado, a partir de uma única linguagem de programação.

A relação entre um conjunto de algoritmos¹⁹ e o conjunto de dados sobre os quais atuam os algoritmos é designada por paradigma de programação. As linguagens de programação também serão classificadas quanto ao paradigma de linguagem de programação. Desta forma, serão discutidos alguns paradigmas de linguagens de programação:

Linguagem de programação imperativa – Programação imperativa é um paradigma de programação que descreve a computação como ações, enunciados ou comandos que mudam o estado (variáveis) de um programa.

Linguagem de programação orientada a objetos – Paradigma de programação baseado no conceito de “objetos”, que podem conter dados, na forma de campos, frequentemente conhecidos como atributos e código na forma de procedimentos, geralmente conhecidos como métodos.

Linguagem de programação funcional – Paradigma de programação que trata a computação como uma avaliação de funções matemáticas e que evita estados ou dados mutáveis.

¹⁹Sequência de instruções para resolver um problema.

Linguagem de programação genérica – Estilo de programação de computadores no qual os algoritmos são escritos em termos de tipos que serão especificados posteriormente e que serão então instanciados quando necessários para tipos específicos fornecidos como parâmetros.

Linguagem de programação procedural – Paradigma de programação derivado da programação estruturada, com base no conceito de utilização de procedimentos. Os procedimentos, também conhecidos como rotinas, sub-rotinas ou funções, contêm simplesmente uma série de instruções a serem executadas.

Linguagem de programação reflexiva – Paradigma de programação que é utilizado como uma extensão para o paradigma orientado a objetos, para adicionar auto-otimização e aumentar a flexibilidade de uma aplicação. Neste paradigma a computação não é trabalhada somente durante a compilação do programa, mas também durante sua execução.

Linguagem de programação orientada a eventos – Paradigma de programação em que o controlo do fluxo do programa orientado a eventos é conduzido por indicações externas, chamadas eventos.

Linguagem de programação estruturada – Paradigma de programação orientado a melhorar a clareza, qualidade e tempo de desenvolvimento de um programa recorrendo a utilização de sub-rotinas e três estruturas básicas: sequência, seleção e iteração.

Linguagem de programação concorrente – Paradigma de programação utilizada na implementação de programas de computador que utilizam execução concorrente de várias tarefas computacionais interativas, que podem ser implementadas como programas separados ou como um conjunto de *threads* criadas por um único programa.

Linguagem de programação distribuída – Paradigma de programação em que o sistema possui componentes que estão localizados em diferentes computadores na rede, que então comunicam e coordenam as ações através da troca de mensagens.

Linguagem de programação generativa – Abordagem de programação que assenta na seleção automática e síntese de componentes a partir de especificações de alto nível e geradores de código.

Linguagem de programação de avaliação preguiçosa (*lazy evaluation*) – Técnica utilizada em programação para atrasar a computação até ao ponto em que o resultado da computação é necessário e também evita avaliações repetidas

Linguagem de programação baseada em protótipos – Estilo de programação orientada a objetos na qual a reutilização de comportamento (conhecida como herança) é executada por meio de um processo de reutilização de objetos existentes via delegação que servem como protótipos.

Linguagem de programação orientada a aspeto – Paradigma de programação de computadores que permite aos programadores separar e organizar o código de acordo com a sua importância para a aplicação. Todo o programa escrito no paradigma orientado a objetos possui código que é alheio da implementação do comportamento do objeto. Este código é todo aquele utilizado para implementar funcionalidades secundárias e que se encontra espalhado por toda a aplicação. A programação orientada a aspeto permite que esse código seja encapsulado e modularizado.

Linguagem de programação multi-paradigma – Linguagem de programação com suporte para múltiplos paradigmas de programação.

Nas subsecções seguintes é feita uma retrospectiva sobre as linguagens de programação *web* utilizadas no desenvolvimento do *backend*²⁰ dos sítios *web* mais populares(EBizMBA, 2018) (i.e., os mais visitados).

²⁰Camada de acesso a dados de um *software*.

2.1.2 C#

Desenvolvida por:	Microsoft
Desenhada por:	Microsoft
Criada em:	Julho de 2000
Última versão estável:	7.3 / 7 de maio, 2018
Sistema operativo / Plataforma:	CLI
Licença:	MIT / CLR MIT
Extensão do ficheiro:	.cs
Padrão / Especificação:	ECMA-335 e ISO/IEC 23271
Estilo de tipagem:	Estática, dinâmica, forte, segura, nominativa, parcialmente inferida
Paradigma:	Imperativa, orientada a objetos, funcional, procedural, genérica, reflexiva, orientada a eventos, estruturada, concorrente

Notas:

C# (pronuncia-se *c sharp*) (Microsoft, 2000) e o seu desenvolvimento é liderado por Anders Hejlsberg. O C# é uma das linguagens de programação desenhada pela CLI²¹. Tem grandes influências do Java e Object Pascal. O código fonte é compilado pela CLI e interpretado pela máquina virtual CLR²².

Outras características:

Simplificada - Não existem ponteiros²³, manipulação direta de memória não permitida, gestão automática de memória e coletor de lixo (*garbage collection*), etc.

Moderna - Simples para construir aplicações robustas / escaláveis / interoperáveis, suporte para converter qualquer componente num serviço *web* que pode ser consumido por uma aplicação a correr em qualquer plataforma.

Orientada a objetos - Encapsulamento de dados, herança, polimorfismo, interfaces,

²¹ *Common Language Infrastructure* - Especificação aberta (ECMA-335 e ISO/IEC 23271) desenvolvida pela Microsoft que descreve o código executável e ambiente *runtime* que forma o core da Microsoft .NET Framework e das implementações Mono e Portable.NET.

²² *Common Language Runtime*.

²³ Tipo de dado de uma linguagem de programação cujo valor se refere diretamente a um outro valor alocado noutra área da memória, através de seu endereço.

tipos primitivos (*int*, *float*, *double*) também são objetos.

Type-safe - Impossível fazer *type casting*²⁴ não segura (e.g., *double* para *boolean*), tipos de dados automaticamente inicializados a zero (0) e objetos a nulo (*null*), vetores com índice base zero (0) e *bound checked*²⁵, verificação de *overflow*²⁶ de tipos de dados.

Interoperabilidade - Diversos mecanismos para interoperabilidade como suporte nativo para objetos COM e aplicações Windows, utilização restrita de ponteiros nativos, etc.

Escalável e atualizável - Não necessita registro de DLL²⁷, suporte nativo de interfaces e *method overriding*²⁸ que facilitam atualização, etc.

²⁴Diferentes formas de, implícita ou explicitamente, alterar uma entidade de um tipo de dados para outro.

²⁵Método para verificar se a variável está dentro de alguns limites antes de ser utilizada.

²⁶Armazenamento de mais dados do que a memória suporta.

²⁷*Dynamic-link library*.

²⁸Permite que uma subclasse ou classe derivada forneça uma implementação específica de um método que já é fornecido por uma de suas superclasses ou classe pai.

2.1.3 C

Desenvolvida por:	Dennis Ritchie & Bell Labs (Secção de pesquisa e de desenvolvimento da AT&T, empresa de telecomunicações dos Estados Unidos, atualmente sendo uma subsidiária da Nokia)
Desenhada por:	Dennis Ritchie
Criada em:	1972
Última versão estável:	C18 / junho, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	Dependente da implementação
Extensão do ficheiro:	.c, .h
Padrão / Especificação:	ANSI X3J11 (ANSI C); ISO/IEC JTC1/SC22/WG14 (ISO C)
Estilo de tipagem:	Estática, fraca, manifesta, nominal
Paradigma:	Estruturada, imperativa, procedural

Notas:

C ([Ritchie, 1993](#)) é uma linguagem genérica de programação imperativa.

Outras características: fiabilidade, portabilidade, flexibilidade, interatividade, modularidade, eficiência, etc.

2.1.4 C++

Desenvolvida por:	-
Desenhada por:	Bjarne Stroustrup
Criada em:	1979
Última versão estável:	ISO/IEC 14882:2017 / 15 de dezembro, 2017
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	-
Extensão do ficheiro:	.C, .cc, .cpp, .cxx, .c++, .h, .hh, .hpp, .hxx, .h++
Padrão / Especificação:	ISO/IEC 14882:1998; ISO/IEC 14882:2003;ISO/IEC 14882:2011;ISO/IEC 14882:2014;ISO/IEC 14882:2017
Estilo de tipagem:	Estática, nominativa, parcialmente inferida
Paradigma:	Imperativa, orientada a objetos, funcional, procedural, genérica

Notas:

C++ (Stroustrup, 1994) é uma linguagem de programação que possui como características: imperativa, genérica e orientada a objetos, também possui características que lhe permitem manipulação de memória em baixo nível.

Outras características: simplicidade, portabilidade, dependente da plataforma, sensível a maiúsculas e minúsculas, baseada em compiladores, baseada em sintaxe, utilização de ponteiros, etc.

2.1.5 D

Desenvolvida por:	D Language Foundation
Desenhada por:	Walter Bright, Andrei Alexandrescu
Criada em:	Dezembro de 2001
Última versão estável:	2.082.1 / 10 de outubro, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	Boost
Extensão do ficheiro:	.d
Padrão / Especificação:	-
Estilo de tipagem:	Estática, forte, inferida
Paradigma:	Imperativa, orientada a objetos, funcional, procedural, genérica, reflexiva, generativa, concorrente

Notas:

A linguagem de programação D ([Bright and Digital Mars, 2001](#)) surgiu em 2001 por Walter Bright da Digital Mars²⁹ e é uma linguagem imperativa, orientada a objetos e de multi-paradigma.

Outras características: coletor de lixo (*garbage collection*), funções especiais (*Delegates*, etc.), vetores especiais e operações especiais sobre vetores (redimensionamento, *slicing*, etc.), eficiência, fiabilidade, compatibilidade, compilação condicional, texto *unicode*, comentários para documentação, etc.

²⁹Empresa norte-americana que se dedica ao desenvolvimento de compiladores de c / c++ e ambientes de desenvolvimento integrado.

2.1.6 Erlang

Desenvolvida por:	Ericsson
Desenhada por:	Joe Armstrong, Robert Virding e Mike Williams
Criada em:	1986
Última versão estável:	21.0 / 20 de 2018, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	MPL modificada
Extensão do ficheiro:	.erl, .hrl
Padrão / Especificação:	-
Estilo de tipagem:	Dinâmica, forte
Paradigma:	Funcional, orientada a eventos, concorrente, distribuída

Notas:

Erlang ([Armstrong et al., 1986](#)) é um linguagem de programação de aplicação geral (genérica), concorrente e funcional. Erlang também é uma linguagem de programação com mecanismos de *garbage collector*³⁰. Foi desenvolvida por Joe Armstrong, Robert Virding e Mike Williams da Ericsson³¹.

Outras características: robusta, concorrencia, programação distribuída, capacidade de recuperação de erros, *hot code loading*³², etc.

³⁰Processo usado para a automação da gestão de memória.

³¹Empresa de tecnologia sueca, fabricante de equipamentos de telefone fixo e móvel. Líder mundial no setor de telecomunicações, foi fundada em 1876 como uma loja de reparação de telégrafos por Lars Magnus Ericsson.

³²Atualização de código sem necessidade de parar o serviço.

2.1.7 Go

Desenvolvida por:	Google
Desenhada por:	Robert Griesemer, Rob Pike e Ken Thompson
Criada em:	Novembro de 2009
Última versão estável:	1.11.1 / 1 de outubro, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	BSD e patenteada
Extensão do ficheiro:	.go
Padrão / Especificação:	Go Language Specification
Estilo de tipagem:	Estática, forte, estrutural, inferida
Paradigma:	Imperativa, estruturada, concorrente, orientada a objetos, procedural, reflexiva, orientada a eventos

Notas:

Go (também conhecida por golang) ([Griesemer et al., 2009](#)) é uma linguagem de programação desenvolvida por Robert Griesemer, Rob Pike e Ken Thompson da Google³³ em 2009. É uma linguagem compilada³⁴, tipagem estática, com *garbage collector*, sistema baseado em propriedades, *memory safety*³⁵ e característica concorrential *CSP-style*³⁶. O compilador e as outras ferramentas de suporte da Google são gratuitos.

Outras características: binários (Go gera binários com todas as dependências embebidas), desenho da linguagem (simples e fácil de entender), gestor de pacotes, suporte para programação concorrential, suporte para testes, etc.

³³Empresa tecnológica norte-americana especializada em serviços e produtos da Internet. Foi fundada em 1998 por Larry Page e Sergey Brin.

³⁴O código fonte, nessa linguagem, é executado diretamente pelo sistema operativo ou pelo processador, após ser traduzido por meio de um processo chamado compilação, usando um programa de computador chamado compilador, para uma linguagem de baixo nível.

³⁵Proteção contra problemas de *software* e vulnerabilidades de segurança em operações de acesso da memória.

³⁶Linguagem formal para descrever padrões de interação em sistemas concorrentiais.

2.1.8 Hack / HHVM

Hack:

Desenvolvida por:	Facebook
Desenhada por:	Julien Verlauguet, Alok Menghrajani, Drew Paroski e outros
Criada em:	2014
Última versão estável:	-
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	BSD
Extensão do ficheiro:	-
Padrão / Especificação:	-
Estilo de tipagem:	Estática, dinâmica, fraca, gradual
Paradigma:	Imperativa, orientada a objetos, funcional, procedural, reflexiva

HHVM:

Desenvolvida por:	Facebook
Desenhada por:	-
Criada em:	Dezembro de 2011
Última versão estável:	3.27.2 / 15 de agosto, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	BSD
Extensão do ficheiro:	Não aplicável
Padrão / Especificação:	Não aplicável
Estilo de tipagem:	Não aplicável
Paradigma:	Não aplicável

Notas:

Hack (O'Sullivan, 2014) é uma linguagem de programação para a HipHop Virtual Machine (HHVM) (Adams et al., 2014), criada pelo Facebook³⁷ como um dialeto

³⁷Empresa tecnológica norte-americana com sede em Menlo Park (Califórnia). Desenvolve media online social e serviços de redes sociais. Foi fundada em 2004 por Mark Zuckerberg,

do PHP³⁸. A linguagem Hack foi desenvolvida inicialmente por Julien Verлагuet, Alok Menghrajani, Drew Paroski e outros. O HipHop Virtual Machine (HHVM) é uma máquina virtual³⁹ de código aberto baseada em compilação JIT (*just-in-time*) e que serve o motor de execução das linguagens de programação PHP e Hack. O HHVM foi inicialmente desenvolvido pelo Facebook em 2011.

Outras características:-

2.1.9 Haskell

Desenvolvida por:	-
Desenhada por:	Lennart Augustsson, Dave Barton, Brian Boutel, Warren Burton, Joseph Fasel, Kevin Hammond, Ralf Hinze, Paul Hudak, John Hughes, Thomas Johnsson, Mark Jones, Simon Peyton Jones, John Launchbury, Erik Meijer, John Peterson, Alastair Reid, Colin Runciman, Philip Wadler
Criada em:	1990
Última versão estável:	Haskell 2010 / julho, 2010
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	-
Extensão do ficheiro:	.hs, .lhs
Padrão / Especificação:	2010, Haskell 2010
Estilo de tipagem:	Estática, forte, inferida
Paradigma:	Funcional, genérica, avaliação preguiçosa

Notas:

Haskell ([Augustsson et al., 2010](#)) é uma linguagem de programação genérica,

Eduardo Saverin, etc.

³⁸PHP: *Hypertext Preprocessor*.

³⁹Consiste num software que executa programas como um computador real, também conhecido por processo de virtualização.

puramente funcional, com semântica não estrita, tipagem forte e estática. Apareceu em 1990 e o seu nome surge em homenagem a Haskell Curry⁴⁰.

Outras características:

Avaliação preguiçosa (*lazy evaluation*) – Técnica utilizada em programação para atrasar a computação até um ponto em que o resultado da computação é considerado necessário.

Polimorfismo – Um valor pode assumir vários tipos de dados dependendo do contexto funcional.

Tipagem estática – Todos os erros de um programa podem ser verificados antes da execução ou *runtime*.

2.1.10 Java

Desenvolvida por:	Sun Microsystems (pertence à Oracle Corporation)
Desenhada por:	James Gosling
Criada em:	Maio de 1995
Última versão estável:	11.0.1 / 6 de outubro, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	GNU General Public License, Java Community Process
Extensão do ficheiro:	.java, .class, .jar
Padrão / Especificação:	Java Language Specification
Estilo de tipagem:	Estática, forte, segura, nominativa, manifesta, inferida
Paradigma:	Estruturada, imperativa, concorrente, funcional, genérica, orientada a objetos, reflexiva, procedural, orientada a eventos

⁴⁰Haskell Brooks Curry (Millis, 12 de setembro de 1900 - State College, 1 de setembro de 1982) foi um matemático dos Estados Unidos. Conhecido pelo seu trabalho na lógica combinatória.

Notas:

A linguagem de programação Java ([Gosling, 1995](#)) surgiu em 1995 e foi inicialmente implementada por James Gosling. É uma linguagem genérica, concorrential baseada em classes⁴¹ e orientada a objetos. Java é compilada em *bytecode* que depois é interpretado pela *Java Virtual Machine*. Atualmente é desenvolvida pela Sun Microsystems⁴² que faz parte da Oracle⁴³.

Outras características:

Simplicidade – Algumas características do c/c++ são eliminadas como a declaração *goto*, ficheiros *header*, *overloading* de operadores⁴⁴, múltipla herança e ponteiros.

Suporta programação distribuída – Suporta TCP/IP, pode abrir e aceder a objetos remotos na Internet.

Robusta – Tipagem forte, não suporta ponteiros, possui *garbage collection*.

Portabilidade – Especifica tamanhos de tipo de dados primitivos e o seu comportamento aritmético.

Alta performance – Embora seja uma linguagem interpretada, suporta compiladores *Just-in-time*, o que compila dinamicamente os *bytecodes* para código máquina.

Multithreading – Suporta *multithreading*, incluindo primitivas de sincronização, o que torna a programação de *threads* muito mais simples.

Linguagem dinâmica – Carregamento dinâmico de classes, compilação dinâmica e gestão automática de memória (*garbage collection*).

⁴¹Classe é uma descrição que abstrai um conjunto de objetos com características similares. Mais formalmente, é um conceito que encapsula abstrações de dados e procedimentos que descrevem o conteúdo e o comportamento de entidades do mundo real, representadas por objetos. ([Pressman and Maxim, 2016](#))

⁴²Empresa que foi adquirida pela Oracle Corporation em 2009. A SUN era, originalmente, fabricante de computadores, semicondutores e *software* com sede em Santa Clara, Califórnia, no Silicon Valley (Vale do Silício). As fábricas da Sun localizam-se em Hillsboro, no estado do Oregon, nos Estados Unidos, e em Linlithgow, na Escócia.

⁴³Empresa multinacional de tecnologia e informática dos Estados Unidos, especializada no desenvolvimento e comercialização de *hardware* e *software* de base de dados.

⁴⁴Caso específico de polimorfismo em que operadores tem diferentes implementações dependendo do tipo de argumentos.

2.1.11 JavaScript

Desenvolvida por:	Netscape Communications Corporation, Mozilla Foundation, Ecma International
Desenhada por:	Brendan Eich
Criada em:	Dezembro de 1995
Última versão estável:	ECMAScript 2018 / junho, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	BSD e patenteada
Extensão do ficheiro:	.js .mjs
Padrão / Especificação:	ISO/IEC C++ 1998; ISO/IEC C++ 2003
Estilo de tipagem:	Dinâmica, fraca, “ <i>duck typing</i> ”
Paradigma:	Imperativa, funcional, orientada a eventos, procedural, orientada a objetos, baseada em protótipos

Notas:

JavaScript (JS) ([Severance, 2012b](#)) é uma linguagem de programação dinâmica, de alto nível, tipagem fraca, baseada em protótipos e de multi-paradigma. Apareceu em 1995 e foi inicialmente implementada em 10 dias por Brendan Eich e desenvolvida pela Netscape Communications Corporation⁴⁵.

É a principal linguagem de programação do lado do cliente nos navegadores *web*, tendo sido a linguagem com mais *pull-requests*⁴⁶ do Github([BVBA, 2017](#)), no entanto começa a ser utilizada do lado do servidor através de ambientes como o Node.js. Inicialmente a linguagem chamava-se *Mocha*, nome atribuído por Marc Andreessen (fundador da Netscape) e foi renomeada para *LiveScript* quatro meses mais tarde([Ilegbodu, 2015](#)). Entre 1996 e 1997 a Netscape levou o JavaScript ao *Ecma standards organization* para desenvolver e manter uma especificação da linguagem que se tornasse um padrão entre os vários fabricantes de navegadores *web*. O *Ecma Technical Committee 39* (melhor conhecido como TC39) foi criado para desenvolver

⁴⁵Empresa de serviços de computadores dos Estados Unidos, mais conhecida pelo seu navegador *web* Netscape Navigator.

⁴⁶Permite notificar alterações submetidas a uma versão de *software* num repositório do Github.

a linguagem e em junho de 1997 publicou a *ECMA-262 Ed.1*. ECMAScript é o nome oficial do padrão (*standard*), sendo o JavaScript a implementação mais popular do padrão (TIOBE, 2018). ActionScript (Macromedia⁴⁷) e JScript (Microsoft) são outros exemplos da implementação. As versões de JavaScript vão acompanhado a evolução da especificação, por exemplo, a primeira versão de JavaScript era o ECMAScript 1. Os nomes das especificações costumam abreviar-se como ESx. As características do ECMAScript passam por um processo de propostas. As propostas são submetidas normalmente pela comunidade de programadores e são classificadas como propostas oficiais se o TC39 concordar com a sua validade. A proposta é implementada em vários motores JavaScript (“em implementação”) e depois de um processo de maturação é incluída na próxima versão de ECMAScript e torna-se padrão (*standard*).

Em 1997 surgiu o ECMAScript 1. Um ano depois surgiu o ECMAScript 2, apenas com pequenas alterações, de maneira a manter a compatibilidade com o padrão ISO do JavaScript. Em dezembro de 1999 surge o ECMAScript 3 com algumas características como: expressões regulares⁴⁸ (Regex), tratamento de exceções (bloco *try/catch*) e formatação de saída numérica.

Existiu um renascimento do JavaScript quando surgiram bibliotecas JavaScript como o JQuery⁴⁹, Prototype⁵⁰, Dojo⁵¹ e Mootools⁵². Também surgiu o Ajax⁵³ com Jesse James Garrett em fevereiro de 2005 (Garrett, 2005), o que poderá ter precipitado a necessidade da comissão TC39 retomar o trabalho no ES4 durante o outono de 2005.

⁴⁷Empresa de desenvolvimento de *software* gráfico e web. Formada em 1992 sendo uma fusão da Authorware, Inc. (criadores do Authorware) e MacroMind-Paracomp (criadores do Macromind Director). A sua sede encontra-se localizada em São Francisco, Califórnia.

⁴⁸Forma concisa e flexível de identificar cadeias de caracteres de interesse, como caracteres particulares, palavras ou padrões de caracteres.

⁴⁹Biblioteca de funções Javascript que interage com o HTML, desenvolvida para simplificar os *scripts* interpretados no navegador do cliente (*client-side*).

⁵⁰*Framework* de código aberto em JavaScript, utilizado para o auxílio no desenvolvimento de aplicações *web*.

⁵¹Biblioteca em JavaScript, de código fonte aberto, projetado para facilitar o rápido desenvolvimento de interfaces ricas.

⁵²*Framework* de código aberto em JavaScript, utilizado para criação de aplicações *web* baseadas no paradigma Ajax.

⁵³*Asynchronous JavaScript and XML* - Utilização metodológica de tecnologias como JavaScript e XML, existentes nos navegadores, para tornar páginas *web* mais interativas com o utilizador, utilizando solicitações assíncronas de informação.

Assim retomaria-se o ES3 (com 7 anos), juntamente com as propostas do ES4 e as aprendizagens do ActionScript e JScript. No entanto quando a comissão se reuniu formaram-se dois grupos: “Grupo ECMAScript 4” (Adobe⁵⁴, Mozilla⁵⁵, Opera⁵⁶ e Google) que queriam trabalhar numa grande atualização e o “Grupo ECMAScript 3.1” (Microsoft e a Yahoo⁵⁷) que queriam apenas trabalhar numa pequena atualização (*small subset*). Em 2008 a contenda foi resolvida e foi obtido um compromisso:

- ECMAScript 3.1 passou a ECMAScript 5 (atualização incremental do ES3).
- A comissão TC39 iria desenvolver uma grande versão, utilizando as propostas da ES4 e tornando-se maior que a atualização incremental do ES5, com o nome de código “Harmony”, surgindo assim o ECMAScript 6.
- Algumas propostas do ES4 seriam abandonadas e retomadas em versões posteriores.

O compromisso levou ao abandono oficial do ECMAScript 4, mas as funções da comissão TC39 saíram reforçadas de forma a evitar futuras contendas.

Em 2009 surgiu o ECMAScript 5 (atualização incremental do ES3). Suportada em todos os navegadores, com a exceção do *Internet Explorer 8*. Também foi introduzido o modo estrito (*strict mode*), tornando a linguagem mais limpa, proibindo determinadas características, executando verificações adicionais em *runtime* e levantando mais exceções.

Com o nome de código “Harmony”, o ECMAScript 6 surgiu em 2015, o que trouxe

⁵⁴Companhia americana que desenvolve programas de computador com sede em San Jose, Califórnia. Foi fundada em dezembro de 1982 por John Warnock e Charles Geschke.

⁵⁵Comunidade de *software* livre criada em 1998 por membros da Netscape.

⁵⁶Empresa de *software* da Noruega fundada em 1995 pelos engenheiros de *software* Jon S. von Tetzchner e Geir Ivarse.

⁵⁷Portal *web* sediado em Sunnyvale, Califórnia que é uma subsidiária integral da Verizon Communications através da Oath Inc.. O Yahoo foi fundado por Jerry Yang e David Filo em janeiro de 1994.

importantes características como: *classes*⁵⁸, *arrows*⁵⁹, módulos⁶⁰, *generators*⁶¹, *promises*⁶², etc.

Em 2016 e 2017 surgiram respetivamente o ECMAScript 7 e 8. Em junho de 2018 foi finalizada a versão 9 do ECMAScript (ECMAScript 2018)(Group, 2018) e as suas novas características são:

- Propriedades *Rest/Spread* para objetos – Operador *Rest* (...) permite recuperar propriedades que não tinham sido extraídas. O operador *Spread* que também são três pontos (...) é utilizado para criar novos objetos (reestruturar).
- Iteradores assíncronos – Atuam como iteradores síncronos, exceto o método *next()* que retorna um *promise* para um par *{value: Mixed, done: Boolean}*.
- Levantamento de restrições de utilização de literais⁶³ em modelos (*templates*) – Revisto o comportamento de literais, permitindo que programadores criem as suas próprias mini linguagens através da utilização de DSL (*Domain-specific language*).
- *Promise.prototype.finally()* – Finaliza a implementação da *promise*, permitindo registar um *callback* que é chamado quando a *promise* termina como esperado ou é rejeitada.
- *Lookbehind assertions* para expressões regulares (RegEx) – Permite que padrões sejam precedidos por outros padrões em expressões regulares.
- *Flag s(dotAll)* para expressões regulares – Permite ao operador ponto (.) ser igualado a qualquer carácter. A *flag* terá uma ativação declarada, e não por omissão, para manter compatibilidade com expressões regulares existentes.

⁵⁸Embora permitido anteriormente de forma mais complexa, torna-se agora mais simples a sua declaração, permitindo instanciação de objetos, super classes, construtores, métodos estáticos.

⁵⁹Sintaxe de *arrows*, semelhante a *c#*.

⁶⁰Suporte nativo para utilização de módulos.

⁶¹Em modo assíncrono, *generators* são funções cuja execução pode ser suspensa e colocadas novamente em execução mantendo o seu contexto de execução.

⁶²Promessa, ou *promise* é um objeto *proxy* que representa um resultado desconhecido(Kambona et al., 2013). Assim uma operação assíncrona devolve uma *promise* que é responsável por devolver um valor resultante de uma operação futura(Network, 2014).

⁶³Notação para representar um valor fixo no código fonte.

- Caracteres de escape⁶⁴ para aceder a propriedades em expressões regulares unicode⁶⁵ – Caracteres de escape para aceder a propriedades sob a forma de $|p\{\dots\}$ e $|P\{\dots\}$.
- Grupos nomeados ou *named groups* em expressões regulares – Permite aos programadores escrever expressões regulares e fornecer nomes (identificadores) no formato $?<name>\dots$ para diferentes partes de um grupo na expressão regular.

Outras características:-

2.1.12 Node.js

Desenvolvida por:	Node.js Developers, Joyent, contribuintes do Github
Desenhada por:	Ryan Dahl
Criada em:	Maio de 2009
Última versão estável:	8.11.3 (LTS) / 12 de junho, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	MIT license
Extensão do ficheiro:	-
Padrão / Especificação:	ECMA-335 e ISO/IEC 23271
Estilo de tipagem:	Dinâmica, fraca, “ <i>duck typing</i> ”
Paradigma:	Imperativa, funcional, orientada a eventos, procedural, orientada a objetos, baseada em protótipos

Notas:

Node.js (Dahl, 2017) surgiu em 2009 e foi desenvolvido originalmente por Ryan Dahl.

⁶⁴Um único carácter numa cadeia de caracteres que altera o significado de seu sucessor. Uma sequência de escape é o subconjunto de caracteres da cadeia de caracteres formado pelo carácter de escape e o carácter com o significado alterado. Geralmente, o carácter de escape é a barra “\”.

⁶⁵Padrão que permite aos computadores representar e manipular, de forma consistente, texto de qualquer sistema de escrita existente.

Trata-se de uma plataforma de desenvolvimento *web* baseada no motor JavaScript da Google. Com o aparecimento do Node.js tornou-se possível a utilização de JavaScript para desenvolvimento de componentes de um servidor de uma aplicação *web*. Como já existe grande popularidade de utilização de JavaScript no desenvolvimento de aplicações cliente, agora com o Node.js torna-se mais simples a integração de aplicações cliente e servidor(Chaniotis et al., 2015). O Node.js foi na sua maioria desenvolvido em c e c++, tendo baixo consumo de memória, grande eficiência e suportando processos de servidor de longa duração(Tilkov and Vinoski, 2010). O Node.js é executado em modo *single-threaded*, baseando o seu funcionamento em eventos de entrada e saída não bloqueante, na utilização do Node.js é registado um *callback* que é chamado quando determinada operação termina, sendo assim uma linguagem assíncrona, orientada a eventos, não bloqueante.

Outras características:

Seguidamente temos o modelo de eventos tradicional bloqueante:

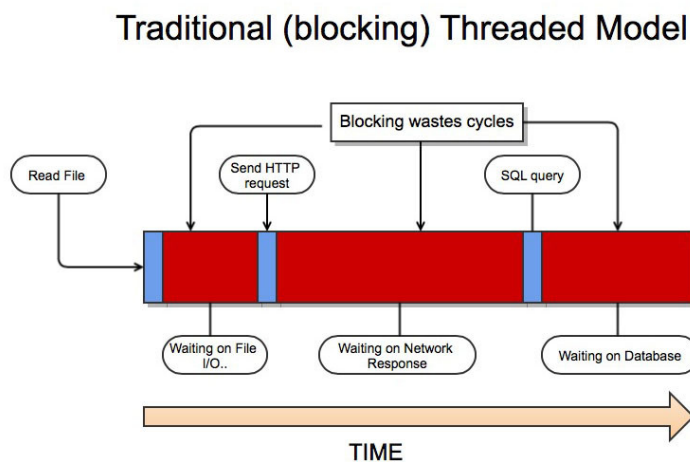


Figura 2.3 – Modelo de eventos tradicional bloqueante(StackOverflow, 2014a)

Modelo assíncrono de eventos não bloqueante, em que se baseia o Node.js:

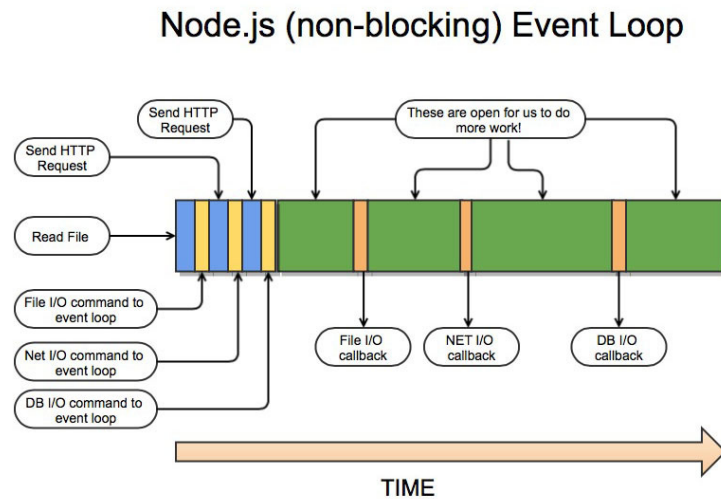


Figura 2.4 – Modelo assíncrono de eventos não bloqueante([StackOverflow](#), 2014b)

Consequentemente entende-se o Node.js como plataforma de desenvolvimento de servidores *web*, para sistemas escaláveis, sem os problemas implícitos da gestão de um sistema *multithreading*([Severance](#), 2012b). O Node.js é executado apenas num processo, pelo que se podem utilizar mecanismos que permitem correr múltiplas instâncias do mesmo processo (*clustering*), através de um processo de partilha de *sockets* entre processos, podendo rentabilizar os recursos disponibilizados pelo sistema operativo.

2.1.13 Perl

Desenvolvida por:	Larry Wall
Desenhada por:	Larry Wall
Criada em:	Dezembro de 1987
Última versão estável:	5.28.0 / 23 de junho, 2018; 5.26.2 / 14 de abril, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	Artistic License 1.0; GNU General Public License
Extensão do ficheiro:	.pl, .pm, .t, .pod
Padrão / Especificação:	-
Estilo de tipagem:	Dinâmica
Paradigma:	Estruturada, imperativa, funcional, genérica, orientada a eventos, orientada a objetos, reflexiva, procedural

Notas:

Perl (*Practical Extraction and Reporting Language*) (Wall, 1987) é uma linguagem de programação genérica inicialmente desenhada e desenvolvida por Larry Wall em 1987. É uma linguagem dinâmica interpretada⁶⁶ de alto nível.

Outras características: sintaxe simples, versatilidade, motor de expressões regulares, enorme biblioteca de módulos, etc.

⁶⁶Linguagem de programação em que o código fonte nessa linguagem é executado por um programa de computador chamado interpretador, que em seguida é executado pelo sistema operativo ou processador.

2.1.14 PHP

Desenvolvida por:	The PHP Development Team; Zend Technologies
Desenhada por:	Rasmus Lerdorf
Criada em:	1995
Última versão estável:	7.2.11 / 11 de outubro, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	PHP License
Extensão do ficheiro:	.php, .phtml, .php3, .php4, .php5, .php7, .phps, .php-s, .pht
Padrão / Especificação:	PHP language specification e RFC
Estilo de tipagem:	Dinâmica, fraca, gradual
Paradigma:	Imperativa, funcional, orientada a objetos, reflexiva, procedural

Notas:

O PHP ([Severance, 2012a](#)) é uma linguagem de *scripting* do lado de servidor. É uma linguagem de programação genérica, no entanto é muito popular no desenvolvimento *web*. Apareceu em 1995, foi inicialmente desenhada por Rasmus Lerdorf e é desenvolvida pela “The PHP Development Team”⁶⁷ e pela “Zend Technologies”⁶⁸.

Outras características: simplicidade, rápida, interpretada, *case sensitive*, eficiente, segura, flexível, reportagem de erros, tipagem fraca, monitorização de acesso em tempo real, etc.

⁶⁷Grupo internacional de programadores que lideram o desenvolvimento do PHP e projetos relacionados.

⁶⁸Empresa norte-americana fabricante de *software*, localizada em Cupertino, California, Estados Unidos.

2.1.15 Python

Desenvolvida por:	Python Software Foundation
Desenhada por:	Guido van Rossum
Criada em:	1991
Última versão estável:	3.7.1 / 20 de outubro, 2018; 2.7.15 / 1 de maio, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	Python Software Foundation License
Extensão do ficheiro:	.py, .pyc, .pyd, .pyo, .pyw, .pyz
Padrão / Especificação:	Python Enhancement Proposals
Estilo de tipagem:	Dinâmica, forte, gradual, “ <i>duck typing</i> ”
Paradigma:	Imperativa, orientada a objetos, funcional, procedural, reflexiva, orientada a eventos, orientada a aspeto

Notas:

Python ([van Rossum, 1991](#)) é uma linguagem de programação genérica que apareceu em 1991 e foi desenhada por Guido van Rossum. Atualmente é desenvolvida pela “Python Software Foundation”⁶⁹.

Outras características: linguagem interpretada, extensível, etc.

⁶⁹Organização sem fins lucrativos que se dedica à linguagem de programação Python.

2.1.16 Ruby

Desenvolvida por:	Yukihiro Matsumoto e outros
Desenhada por:	Yukihiro Matsumoto
Criada em:	1995
Última versão estável:	2.5.3 / 18 de outubro, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	Dupla: Ruby License / FreeBSD License
Extensão do ficheiro:	.rb
Padrão / Especificação:	2011(JIS X 3017), 2012(ISO/IEC 30170)
Estilo de tipagem:	Dinâmica, forte, “ <i>duck typing</i> ”
Paradigma:	Imperativa, orientada a objetos, funcional, reflexiva, orientada a aspeto

Notas:

A linguagem de programação Ruby ([Matsumoto, 1995](#)) é uma linguagem de programação funcional, orientada a objetos, imperativa e reflexiva. Apareceu em 1995 e foi desenhada por Yukihiro Matsumoto.

Outras características: *garbage collection*, *operator overloading*, tratamento de exceções, etc.

2.1.17 Scala

Desenvolvida por:	École Polytechnique Fédérale de Lausanne
Desenhada por:	Martin Odersky
Criada em:	Janeiro de 2004
Última versão estável:	2.12.7 / 27 de setembro, 2018
Sistema operativo / Plataforma:	JVM, JavaScript, LLVM
Licença:	BSD 3-clausulas
Extensão do ficheiro:	.scala, .sc
Padrão / Especificação:	Scala Language Specification (SLS)
Estilo de tipagem:	Estática, forte, inferida, estrutural
Paradigma:	Imperativa, orientada a objetos, funcional, genérica, reflexiva, orientada a eventos, concorrente

Notas:

Scala (Odersky, 2004) é uma linguagem de programação genérica, com suporte para desenvolvimento funcional e com forte sistema de escrita estático (*static typing*). Apareceu em 2004 e foi desenhada por Martin Odersky. Atualmente é desenvolvida pelo laboratório “Programming Methods Laboratory of École Polytechnique Fédérale de Lausanne”⁷⁰.

Outras características: flexibilidade sintática, sistema de tipos unificado, etc.

⁷⁰Escola de engenheiros de Lausanne, na Suíça fundada em 1853 com o nome de *Escola Especial de Lausanne*.

2.1.18 XHP

Desenvolvida por:	Facebook
Desenhada por:	Marcel Laverdet
Criada em:	Fevereiro de 2010
Última versão estável:	2.6.0 / 10 de maio, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	BSD License; MIT License
Extensão do ficheiro:	-
Padrão / Especificação:	-
Estilo de tipagem:	Dinâmica, fraca
Paradigma:	Imperativa, orientada a objetos, funcional, procedural, reflexiva

Notas:

XHP ([Laverdet, 2010](#)) é uma linguagem de programação que aparece como uma extensão de PHP e Hack desenhada por Marcel Laverdet em 2010 e desenvolvida pelo Facebook. Aparece fortemente integrada com o XML. O Facebook desenvolveu uma extensão semelhante para JavaScript (JSX).

Outras características: verificação de sintaxe de HTML, proteção automática de XSS⁷¹, mutação de objetos⁷², HTML personalizado, etc.

2.1.19 Conclusão

Nesta secção concluímos que existe um grande número de linguagens de programação *web* para o desenvolvimento do *backend* dos sítios *web* mais populares. Sendo a sua maioria linguagens de programação de código aberto, denota uma preocupação destas empresas / entidades com os custos em termos de licenciamento de *software* e é também um fator importante na escolha destas linguagens o fato de poderem

⁷¹ *Cross-site scripting*.

⁷² Objetos XHP podem ser manipulados durante a renderização com métodos (*setAttribute()*, *getAttribute()*, *appendChild()*) da API DOM.

facilmente desenvolver e melhorar as linguagens, bem como aceder a suporte / documentação de grandes comunidades de programadores.

Linguagem de Programação:	Modelo de execução:	Algumas influências:	Utilizações principais:
C#	Interpretada e compilada	Java, C++	Aplicação, negócio, <i>client-side</i> , genérica, <i>server-side</i> , <i>web</i> , etc.
C	Compilada	Algol, BCPL	Aplicação, sistema, genérica, operações de baixo nível, etc.
C++	Compilada	C, Simula, Algol 68	Aplicação, sistema, etc.
D	Compilada	C, C++, C#, Java	Aplicação, sistema, etc.
Erlang	Compilada	Prolog, ML, Smalltalk, PLEX, LISP	Aplicação, programação distribuída, etc.
Go	Compilada	Alef, APL, BCPL, C, CSP, Limbo, Modula, Newsqueak, Oberon, occam, Pascal, Smalltalk	Aplicação, <i>server-side</i> , <i>web</i> , etc.
Hack	Compilada	C e Perl	<i>Server-side</i> , <i>web</i> , etc.
Haskell	Interpretada e compilada	Clean, FP, Gofer, Hope and Hope+, Id, ISWIM, KRC, Lisp, Miranda, ML e ML padrão, Orwell, SASL, Scheme, SISAL	Aplicação, etc.
Java	Interpretada e compilada	Ada 83, C++, C#, Eiffel, Mesa, Modula-3, Oberon, Objective-C, UCSD Pascal, Object Pascal	Aplicação, <i>client-side</i> , negócio, genérica, desenvolvimento de dispositivos móveis, <i>server-side</i> , <i>web</i> , etc.
Javascript	Interpretada	Lua, Scheme, Perl, Self, Java, C, Python, AWK, HyperTalk	<i>Client-side</i> , <i>server-side</i> , etc.
Node.js	Interpretada e compilada	Lua, Scheme, Perl, Self, Java, C, Python, AWK, HyperTalk	<i>Client-side</i> , <i>server-side</i> , etc.
Perl	Interpretada	C, Shell, awk, sed, Lisp	Aplicação, <i>scripting</i> , processamento de texto, <i>web</i> , etc.
PHP	Interpretada	C e Perl	<i>Server-side</i> , <i>web</i> , etc.
Python	Interpretada	ABC, ALGOL 68, APL, C, C++, CLU, Dylan, Haskell, Icon, Java, Lisp, Modula-3, Perl, ML padrão	Aplicação, genérica, <i>web</i> , <i>scripting</i> , inteligência artificial, computação científica, etc.
Ruby	Interpretada	Ada, C++, CLU, Dylan, Eiffel, Lisp, Lua, Perl, Python, Smalltalk	Aplicação, <i>scripting</i> , <i>web</i> , etc.
Scala	Interpretada e compilada	Eiffel, Erlang, Haskell, Java, Lisp, Pizza, ML padrão, OCaml, Scheme, Smalltalk, Oz	Aplicação, programação distribuída, <i>web</i> , etc.
XHP	Compilada	C e Perl	<i>Server-side</i> , <i>web</i> , etc.

Tabela 2.1 – Comparação de linguagens de programação *web* (*server-side*)

Em muitos casos as empresas / entidades são as grandes impulsionadoras da criação e padronização das linguagens de programação utilizadas para o desenvolvimento do *backend*.

2.2 Serviços *web*

2.2.1 Introdução

Com o aumento do poder computacional, as empresas de tecnologia começaram a desenvolver sistemas operativos⁷³ mais rápidos e flexíveis, redes de computador de diferentes tamanhos e complexidade e as aplicações necessitaram de comunicar entre elas de forma dinâmica através de serviços *web*.

Segundo a W3C(*World Wide Web Consortium*)⁷⁴, define-se serviço *web* como um sistema de *software* desenhado para suportar interoperabilidade / interação máquina-máquina sobre uma rede. Possui uma interface⁷⁵ descrita em formato que pode ser processado por uma máquina (especificamente WSDL⁷⁶). Outros sistemas interagem com o serviço *web* de forma perceptível pela descrição através de mensagens SOAP⁷⁷, utilizando o protocolo HTTP serializando XML⁷⁸ em conjunto com outras tecnologias da *web*(Booth et al., 2004).

2.2.2 CORBA, RMI e COM/DCOM

A computação por distribuição de objetos ganhou cada vez mais importância no mundo da computação (Patil et al., 2011). Uma característica importante nas grandes redes de computadores como a Internet ou grandes intranets corporativas

⁷³*Software* de base de um computador destinado a controlar a execução de programas, a comunicação entre dispositivos e programas, assegurando as operações de entrada-saída, a atribuição de recursos aos diferentes processos, o acesso às bibliotecas de programas e aos ficheiros, assim como a compatibilidade dos trabalhos. Nota: O sistema operativo é o *software* mais importante a correr num computador.

⁷⁴Principal organização internacional de normalização técnica para a World Wide Web (abreviado WWW or W3).

⁷⁵Fronteira que facilita a comunicação entre o computador e o seu utilizador (interface gráfica ou textual), ou entre duas aplicações ou ainda entre dois dispositivos.

⁷⁶*Web Services Description Language* - Linguagem baseada em XML utilizada para descrever serviços *web* funcionando como um contrato do serviço.

⁷⁷*Simple Object Access Protocol* - Protocolo Simples de Acesso a Objetos é um protocolo para troca de informações estruturadas em uma plataforma descentralizada e distribuída.

⁷⁸*Extensible Markup Language* - Linguagem normalizada genérica de marcação, i.e, linguagem capaz de descrever uma organização lógica, estruturada de dados através de *tags* definidas.

era a heterogeneidade, i.e., redes em que se interligavam computadores e dispositivos com diferentes sistemas operativos e / ou protocolos. O objetivo da metodologia de utilização de distribuição e objetos era permitir a interoperabilidade de aplicações e independência de plataforma, sistema operativo, linguagem de programação, rede e protocolo.

Existiu uma grande utilização de tecnologias baseadas em objetos, os paradigmas orientados a objetos foram cada vez mais utilizados em sistemas distribuídos. Com a implementação das tecnologias baseadas em objetos surgiram algumas questões relativas à partilha de endereços de memória, partilha de objetos, etc. No entanto nestes sistemas cada uma das componentes do objeto distribuído operavam entre si como um todo. Em síntese, os objetos poderiam estar distribuídos em diferentes computadores, sendo executados no seu próprio espaço de memória fora da aplicação, no entanto, pareceriam objetos locais à própria aplicação.

As três tecnologias mais populares baseadas no paradigma de objetos distribuídos são: CORBA⁷⁹, JAVA RMI⁸⁰ da Sun Microsystems e objetos COM/DCOM⁸¹ da Microsoft.

CORBA

CORBA é uma tecnologia baseada numa arquitetura pedido-resposta e especificada pela OMG⁸² (Patil et al., 2011). Existe uma implementação do objeto no servidor que é invocado pelo cliente. Não existe nenhuma limitação em termos de espaço de endereçamento, cliente e servidor podem coexistir no mesmo espaço de endereçamento ou podem ter o seu próprio espaço independente, podem também estar localizados no mesmo computador ou em diferentes computadores. Objetos

⁷⁹*Common Object Request Broker Architecture.*

⁸⁰*Remote Method Invocation.*

⁸¹*Microsoft Component Object Model.*

⁸²Organização internacional que aprova padrões abertos para aplicações orientadas a objetos. Esse grupo define também a OMA (*Object Management Architecture*), um modelo padrão de objeto para ambientes distribuídos.

CORBA são objetos que suportam uma interface IDL⁸³ e as referências remotas são identificadas como *Object References*. Existe uma especificação da linguagem IDL e como ela é mapeada para outras linguagens. O *proxy*⁸⁴ ou o representante local do cliente é identificado como *IDL stub*, o *proxy* do lado do servidor é o *IDL skeleton*. Para *marshalling*⁸⁵ o pedido e a resposta, a informação é entregue num formato canónico definido pelo protocolo IIOP⁸⁶ utilizado pela interoperabilidade do CORBA na Internet. O esboço (*stub*) IDL utiliza o interface de invocação dinâmica para o *marshalling* do lado do cliente. De forma semelhante os esqueletos IDL utilizam a interface dinâmica de esqueleto para inverter o processo (*unmarshalling*). O pedido e resposta podem ter como parâmetros *Object References* e objetos remotos podem ser passados por referência.

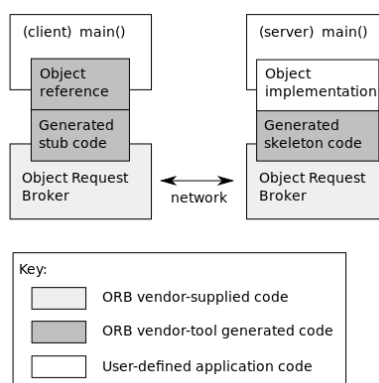


Figura 2.5 – Interface definida utilizando CORBA IDL (Alksentrs, 2008)

Para registar um serviço (objeto) no ORB⁸⁷, CORBA utiliza um adaptador

⁸³Interface Description Language.

⁸⁴Servidor situado entre uma aplicação cliente, como um programa de navegação, e o servidor real. O servidor intermediário interceta todos os pedidos para o servidor real para ver se ele próprio os pode satisfazer e, em caso negativo, envia-os para o servidor real. Nota: O servidor intermediário tem dois objetivos principais: acelerar a satisfação dos pedidos do utilizador e filtrar conteúdos.

⁸⁵Processo de transformação da representação de memória de um objeto em um formato de dados compatível para armazenamento ou transmissão, e é usado tipicamente quando os dados precisam ser movimentados entre diferentes partes de uma aplicação ou entre aplicações.

⁸⁶Internet Inter-ORB Protocol.

⁸⁷Object Request Broker.

(POA⁸⁸) que fornece métodos como *activate_object()* e *activate()*. É utilizado um serviço de atribuição de nomes (CORBA *naming service*), que fornece uma gama de nomes para o mapeamento entre os *Object References*. Os *Object References* podem ser obtidos de duas maneiras: o cliente pode pedir ao serviço de atribuição de nomes ou pode pedir ao serviço de *trading* uma lista de serviços (*object references*) que possuem propriedades semelhantes a uma determinada chave de pesquisa. O cliente estabelece a ligação ao servidor quando obtém o *object reference* para a implementação. O cliente pode utilizar dois tipos de invocação de serviços:

- **Invocação estática e entrega** – O cliente é compilado com a informação do serviço de especificação IDL. O *proxy* implementa os métodos mapeados da interface de mapeamento IDL, com o código do *proxy* correspondente a ser encapsulado no esboço (*stub*) IDL. Da mesma forma o esqueleto (*skeleton*) IDL é o *proxy* do lado do servidor.
- **Invocação dinâmica e entrega** – O objeto pedido é retornado através do método do *proxy* *create_request()*. Também existe forma de submeter argumentos com o pedido. Para obter informação sobre a interface alvo, o repositório de interfaces pode ser consultado. O processo é iniciado pelo método *invoke()*, que é um método do objeto pedido. O *proxy* do lado do servidor possui o código do esqueleto dinâmico que responde ao pedido através do método *invoke()* suportado pela interface *PortableServer::DynamicImplementation*.

CORBA permite a combinação de invocação estática por parte do cliente e entrega dinâmica por parte do servidor e vice-versa. Do lado do servidor a escolha entre entrega estática ou dinâmica é feita pelo próprio objeto no servidor (pode ser determinada pela próprio registo no POA), independentemente de como o pedido foi efetuado, através de invocação estática ou dinâmica.

Meio termo entre a invocação estática e dinâmica é o descarregamento dinâmico

⁸⁸*Portable Object Adapter*.

de esboços (*stubs*), que pode ser utilizado em clientes CORBA desenvolvidos em JAVA. Neste caso o código de cliente pode ser compilado sem a utilização de um compilador IDL (apenas será necessária a informação da forma da interface). Posteriormente serão descritos os elementos que constituem a arquitetura CORBA.

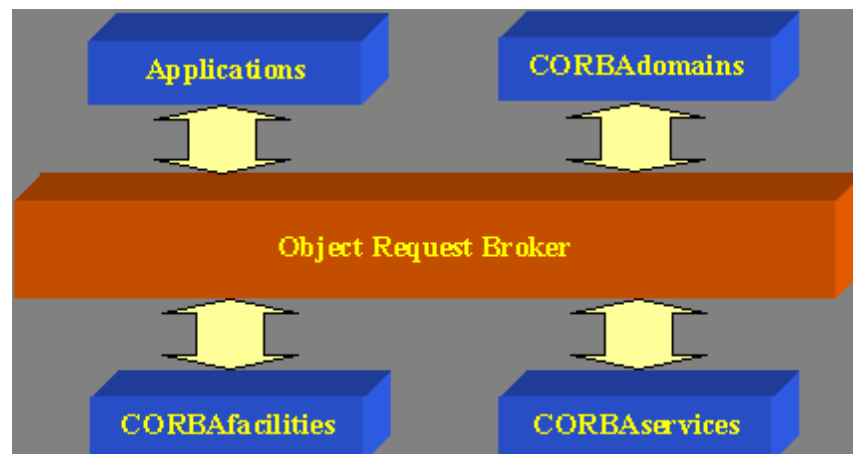


Figura 2.6 – Arquitetura de gestão de objetos (OMA)(Hasegawa, 2000)

Object Request Broker (ORB) - De forma a que os objetos façam pedidos e recebam respostas localmente de objetos remotos é utilizado o *Object Request Broker (ORB)*, que é um barramento de objetos. Utilizando este barramento, o cliente não se apercebe dos mecanismos utilizados na comunicação, ativação e armazenamento de objetos de servidor. Existe uma linguagem pré-definida que permite atingir este objetivo, que é o IDL (*Interface Definition Language*). CORBA 1.1 especificou a IDL (*Interface Definition Language*) em 1991. O CORBA 2.0 especificou interoperabilidade entre *Object Request Brokers* de vários proprietários. Existe uma grande variedade de serviços de *middleware*⁸⁹ fornecidos pelo CORBA ORB. Objetos conseguem descobrir em *runtime* e invocar novos serviços utilizando a ORB. A ORB é mais sofisticada que formas alternativas de *middleware* cliente / servidor, incluído a tradicional RPC⁹⁰(Raj, 1998), MOM⁹¹, procedimentos armazenados em base de

⁸⁹ *Software* de interface que permite interação de diferentes aplicações informáticas, geralmente sendo executadas em diferentes plataformas de equipamento, para troca de dados.

⁹⁰ *Remote Procedure Calls*.

⁹¹ *Message-Oriented Middleware*.

dados ou serviços peer-to-peer.

O CORBA ORB possui os seguintes benefícios: invocação de métodos dinamicamente e estaticamente, ligação a linguagens de alto nível, sistema auto descritivo, transparência local e remota, segurança e transações embebidas, mensagens polimórficas, coexistência e adaptabilidade com sistemas existentes.

CORBAServices - Coleções de serviços com interfaces especificadas através de IDL a nível de sistema. São utilizados para aumentar e complementar as funcionalidades da ORB. A OMG especificou alguns serviços que em seguida são enumerados.

Estabelecidos padrões de dezasseis serviços pela OMG:

- Serviço de ciclo de vida (*Life Cycle Service*) – Define as operações de criação, cópia, transposição e remoção de componentes no barramento.
- Serviço persistente (*Persistence Service*) – Fornece uma interface para guardar componentes de forma persistente numa variedade de servidores, incluindo objetos de base de dados (ODBMS⁹²), bases de dados relacionais (RDBMS⁹³) e ficheiros simples.
- Serviço de atribuição de nomes (*Naming Service*) – Permite localizar outros componentes através do nome, utilização de contexto de federação de nomes. Objetos podem ser localizados em diretórios de rede ou contextos de atribuição de nomes como ISO's X. 500, OSF's DCE, Sun's NIS+, etc.
- Serviço de eventos (*Event Service*) – Permite aos componentes no barramento o registo e anulação de registo em determinados eventos. O serviço define um objeto bem determinado e conhecido como canal de eventos que agrega e distribui eventos por componentes.

⁹²*Object Database Management System.*

⁹³*Relational Database Management System.*

- Serviço de controlo concurrencial (*Concurrency Control Service*) – Serviço que gere a concorrência de transações e *threads*.
- Serviço de transações (*Transaction Service*) – Fornece um sistema de submissões de duas fases entre componentes utilizando um sistema de transações simples ou encadeado.
- Serviço de relações (*Relationship Service*) – Facilita a criação de relações dinâmicas entre componentes desconhecidos entre si. Serviço utilizado para reforçar integridade de relações entre componentes.
- Serviço de exteriorização (*Externalization Service*) – Fornece padronização na forma de obter dados de e para os componentes.
- Serviço de pedidos (*Query Service*) – Permite operações de inquirição aos objetos, sendo um *superset* do SQL⁹⁴.
- Serviço de licenciamento (*Licensing Service*) – Serviços de licenciamento de objetos em utilização, permitindo mecanismos de compensação e controlo da utilização em qualquer ponto do ciclo de vida do componente. Suporta compensação por sessão, por nó, por instância e por *site*.
- Serviço de propriedades (*Properties Service*) – Serviços de associação de valores e propriedades com determinados componentes. Utilizando este serviço as propriedades podem ser associadas com o estado de um componente.
- Serviço de controlo temporal (*Time Service*) – Fornece uma interface de sincronização temporal num ambiente de objeto distribuído. Também permite operações para definir e gerir eventos despoletados por *triggers* temporais.
- Serviço de segurança (*Security Service*) – Fornece uma *framework* completa de segurança de objetos distribuídos. Suporta autenticação, lista de controlo de acessos, confidencialidade. Também gere a delegação de credenciais entre objetos.
- Serviço de controlo de trocas (*Trader Service*) – Funciona como a serviço de publicação de serviços / pedidos dos objetos (páginas amarelas).

⁹⁴ *Structured Query Language*.

- Serviço de coleções (*Collection Service*) – Interfaces CORBA que genericamente criam e manipulam as coleções mais comuns.
- Serviço de arranque (*Startup Service*) – Permite o início de serviços automaticamente quando a ORB é invocada.

CORBAfacilities - São coleções de *frameworks* definidas em IDL que fornecem serviços de utilização direta de objetos aplicativos. Existem duas categorias, vertical e horizontal, que definem as regras de compromisso que os objetos de negócio necessitam para colaborar eficientemente. Em 1994 a OMG estabelece a Common Facilities Request for Proposal 1 (RFP1) e em 1996 é adotada a OpenDoc⁹⁵ da Apple⁹⁶ como a tecnologia para distribuição de documentos (DDCF⁹⁷). As *common facilities* incluem agentes móveis, *frameworks* de objetos de negócio e internacionalização.

CORBA Business Objects - De forma a permitir naturalmente a descrição de conceitos independentes de aplicação como empregado, cliente, conta, pagamento, são utilizados objetos de negócio. Encorajam a visão de *software* que transcende ferramentas, aplicações, bases de dados e outros conceitos de sistema. Objeto de negócio, por definição, independente de uma única aplicação, sendo um componente a nível aplicativo. É reconhecido como um objeto de utilização diária. Por outro lado, objetos a nível de sistema representam entidades que apenas fazem sentido para programadores de sistemas.

No modelo CORBA o objeto de negócio consiste em três tipos de objetos:

- *Business objects* - Encapsulam armazenamento, meta-dados, regras de negócio

⁹⁵ *Framework* criada pela Apple para documentos compostos como alternativa ao *Object Linking and Embedding* (OLE) da Microsoft.

⁹⁶ Empresa multinacional norte-americana que tem o objetivo de projetar e comercializar produtos eletrônicos de consumo, *software* de computador e computadores pessoais.

⁹⁷ *Distributed Document Components*.

e concorrência numa entidade ativa. Também definem como os objetos reagem a alterações das vistas.

- *Business process objects* - Encapsulam lógica de negócio a nível corporativo. No sistema tradicional MVC, o *controller* é que controla o processo. No modelo CORBA, as funções de processo de curta duração são controladas pelo *business objet*. Os processos de longa duração entre *business objects* são controlados pelo *business process object*. Define como os objetos reagem a alterações de ambiente.
- *Presentation Objects* - Representa visualmente o objeto ao utilizador. Cada objeto de negócio pode ter múltiplas representações, conforme o objetivo. As representações comunicam diretamente com o objeto de negócio para representar o objeto. A OMG também reconhece que existem interfaces para objetos não visuais.

Tipicamente o objeto de negócio e o objeto de processo encontram-se no servidor onde o objeto de negócio tem diferentes representações distribuídas para múltiplos clientes. A arquitetura CORBA fornece uma forma de descrever todos os objetos constituintes por interfaces IDL e que podem ser executados na ORB. Não importa se os objetos são executados no mesmo servidor ou não, no que respeita ao cliente estão a lidar com um único componente de objeto de negócio, nem que este esteja fatorizado em objetos a serem executados em diferentes máquinas.

A utilização da arquitetura CORBA possui as seguintes vantagens: interface independente da linguagem, compatibilidade com objetos definidos anteriormente (*backward compatibility*), infraestrutura de objetos distribuídos rica, transparência local / rede, comunicação direta de objetos e interface de invocação dinâmica.

Java RMI

Java RMI é um mecanismo que permite a invocação de um método num objeto que existe noutra espaço de endereços (Patil et al., 2011). O outro espaço de

endereços pode estar na mesma máquina ou não. O mecanismo RMI é basicamente um mecanismo RPC⁹⁸ orientado a objetos.

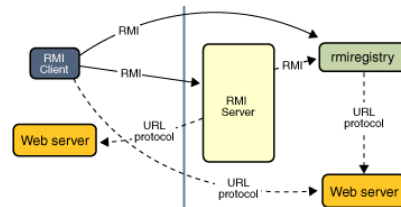


Figura 2.7 – Arquitetura RMI(Oracle, 2017)

A arquitetura Java RMI assenta num protocolo chamado JRMP⁹⁹. A linguagem de programação Java baseia-se em *Java Object Serialization* que permite que os objetos sejam *marshaled* (ou transmitidos) como um fluxo de dados. Como a serialização de objetos é específica de Java, tanto o servidor Java RMI, como o objeto cliente tem de ser desenvolvidos em Java. Cada objeto Java RMI define uma interface, que pode ser utilizada para aceder ao objeto de servidor fora da máquina virtual JAVA local (JVM¹⁰⁰) e na JVM de outra máquina. A interface expõe uma série de métodos, indicativos dos serviços oferecidos pelo objeto servidor. Para um cliente localizar um objeto servidor pela primeira vez, o RMI depende de um mecanismo de nomenclatura chamado *RMIRegistry* que é executado no servidor que detém informação acerca dos objetos servidor disponíveis. O cliente Java RMI obtém informação de um objeto de referência para um objeto Java RMI servidor, executando uma pesquisa pela referência do objeto servidor e invocando métodos do objeto servidor tal como se o objeto estivesse localizado no espaço de memória local do cliente. Nos nomes dos objetos servidor Java RMI são utilizados URL¹⁰¹ e para um cliente obter a referência de um objeto servidor, deve especificar o URL do objeto servidor, tal como o URL de uma página *web*.

Como Java RMI é implementado em Java pode ser utilizado em vários sistemas diferentes (multi-plataforma) desde que exista uma *Java Virtual Machine* nessa

⁹⁸Remote procedure call.

⁹⁹Java Remote Method Protocol.

¹⁰⁰Java Virtual Machine.

¹⁰¹Uniform Resource Locator.

plataforma(Raj, 1998).

Definição de funções da classe remota como uma interface - Na linguagem de programação Java, um objeto remoto é uma instância de uma classe que implementa uma interface remota. A interface remota vai declarar cada um dos métodos que vão ser invocados de outras JVM. As interfaces remotas possuem as seguintes características:

- A interface remota tem de ser declarada pública.
- A interface remota estende a interface *java.rmi.Remote*.
- Cada método tem de declarar a *java.rmi.RemoteException* (ou a super classe *RemoteException*) na sua cláusula *throws*, conjuntamente com as exceções específicas da aplicação.
- O tipo de dados de um objeto remoto que é passado como argumento ou é retornado tem de ser declarado como *remote interface type*.

Implementação das classes no servidor - No mínimo, a implementação de uma classe de um objeto remoto tem de possuir: implementação da interface remota, definição do construtor do objeto remoto e a implementação dos métodos que vão ser invocados remotamente. Neste contexto, a classe servidor possui o método *main* que cria uma instância do objeto remoto e a sua ligação (nome) registado no *rmiregistry*.

Implementação de uma interface remota - Na linguagem de programação Java, quando uma classe declara uma implementação de interface é estabelecido um contrato entre a classe e o compilador. O contrato estabelece que a classe promete que fornecerá o corpo dos métodos ou a definição, de todos os métodos declarados na interface. Os métodos na interface são implicitamente públicos e abstratos. Assim, se a implementação da classe não cumpre o contrato, torna-se por definição classe abstrata e o compilador detetará que a classe não foi declarada como abstrata. Por conveniência, a implementação da classe pode estender a classe

remota *java.rmi.server.UnicastRemoteObject*. Ao ser estendida, a classe pode ser utilizada para criar um objeto remoto que:

- Utiliza o sistema RMI de comunicação por defeito baseado em *sockets*¹⁰².
- Execução em *full time*.

Definição de um construtor de um objeto remoto - A definição do construtor da classe remota fornece a mesma funcionalidade de uma classe não remota: inicializa as variáveis de cada instância da classe e retorna a instância da classe ao programa que chamou o construtor. Cada instância do objeto remoto tem de ser exportada. Exportando um objeto remoto torna-o disponível para receber pedidos a métodos remotos.

Implementação para cada método remoto - A classe de implementação de um objeto remoto tem de possuir o código de cada método remoto especificados na interface remota. Argumentos, valores de retorno podem ser de qualquer tipo de dados da plataforma Java, incluindo tipo de dados *object*, desde que implementem a interface *java.io.Serializable*. Programas cliente e servidor têm de ter acesso à definição da classe *Serializable* de cada tipo de dados utilizado. Se cliente e servidor estiverem em diferentes máquinas, as definições das classes *Serializable* tem de ser transmitidas o que pode constituir uma falha de segurança. Por omissão, em RMI, todos os objetos são copiados, a não ser que sejam marcados como *static* ou *transient*. Objetos remotos são passados por referência. A referência a um objeto remoto é a referência a um esboço (*stub*).

Criação e instalação de um gestor de segurança - O método *main* de um servidor deve criar e instalar um gestor de segurança, o *RMISecurityManager*(Greenhalgh, 2008). Gestor de segurança necessita de ser

¹⁰²Ponto final de um fluxo de comunicação entre processos através de uma rede de computadores.

executado para que exista garantia que as classes utilizadas não executam operações não permitidas. Um programa Java pode especificar um gestor de segurança para especificar a sua política de segurança. Um programa não terá gestor de segurança a não ser que seja especificado. Determinadas operações exigem a existência de um gestor de segurança.

Criação de uma ou mais instâncias de um objeto remoto - O método *main* do servidor necessita criar uma ou mais instâncias da implementação do objeto remoto, que fornece o serviço. O construtor exporta o objeto remoto, uma vez criado, fica disponível para receber pedidos.

Registo de um objeto remoto - Para um cliente conseguir invocar um método de um objeto remoto, tem de encontrar a referência ao objeto remoto. O sistema RMI fornece um serviço de registos que permite a ligação a um objeto remoto em forma de *URL* (“//host/objectname”), onde o *objectname* é um nome do tipo *string* normal. Por razões de segurança, uma aplicação pode ligar ou desligar apenas a um registo em execução no mesmo servidor, evitando que um cliente altere / remova as entradas do registo remoto do servidor. As pesquisas no entanto podem ser efetuadas por qualquer cliente.

Vantagens da utilização da tecnologia JAVA RMI:

- Suporta invocação integrada em objetos em diferentes máquinas virtuais (*JVM*).
- Permite *callbacks* de servidores *applets*¹⁰³.
- Integrar de forma mais simples e natural o modelo de objeto distribuído mantendo a semântica da linguagem Java.

¹⁰³ *Software* que executa uma atividade específica, dentro (do contexto) de outro programa maior (como por exemplo um navegador *web*), geralmente como um *Plugin*. O termo foi introduzido pelo *AppleScript* em 1993.

- Diferenças entre o modelo de objeto distribuído e o modelo local Java.
- Tornar a escrita de aplicações distribuídas o mais simples possível.
- Segurança fornecida pelo *Java runtime environment*.
- Única linguagem de programação.
- Gratuito.

COM/DCOM

DCOM é a extensão distribuída do COM que constrói uma camada de objeto de pedido de procedimento remoto ORPC¹⁰⁴ em cima de DCE/RPC¹⁰⁵ para suportar objetos remotos (Patil et al., 2011). COM é um modelo de desenvolvimento de componentes em ambiente *Windows*¹⁰⁶. Um componente é reutilizável em forma binária e pode ser acoplado a outros componentes de outros vendedores de *software* com muito pouco esforço. O COM tem origem no OLE¹⁰⁷. O COM permite a interação de componentes no mesmo ou em espaços de endereços separados. O DCOM fornece a mesma funcionalidade, mas em máquinas separadas através de uma rede. A figura seguinte retrata a arquitetura DCOM (Markus Horstmann, 1997):

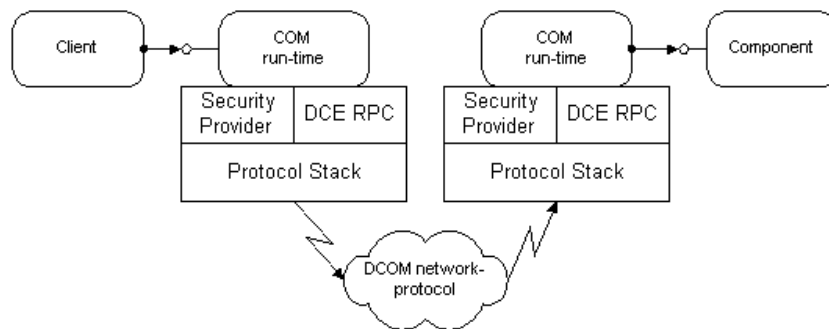


Figura 2.8 – Arquitetura DCOM (Xiao, 2005)

O servidor COM é um programa que implementa classes e interfaces COM.

¹⁰⁴ *Object Remote Procedure Call*.

¹⁰⁵ *Distributed Computing Environment / Remote Procedure Calls*.

¹⁰⁶ Família de sistemas operacionais desenvolvidos, comercializados e vendidos pela Microsoft.

¹⁰⁷ *Object Linking and Embedding*.

Servidores COM existem em três configurações básicas(Eddon and Eddon, 1998): *in-process* ou servidor DLL, servidor *standalone* ou EXE e a terceira configuração é o servidor baseado em Windows NT. Objetos COM são os mesmos independentemente do tipo de servidor. Para o programa cliente o tipo de servidor é transparente. No entanto, a construção do servidor varia muito com a configuração. Pode, no entanto, existir um quarto tipo de servidor, o servidor *surrogate*, que é essencialmente um servidor que permite que um servidor *in-process* seja executado remotamente. *Surrogates* são úteis para tornar servidores COM baseados em *DLLs* disponíveis na rede. No COM, pedidos e respostas são geridos pelo LRPC¹⁰⁸. DCOM suporta clientes e servidores a residir em nós separados. Em DCOM, o mecanismo de pedidos e respostas é idêntico ao COM, excepto que o LRPC é substituído pelo *Object-Oriented RPC* que utiliza um protocolo de rede desenvolvido com base nos pedidos de procedimentos remotos *DCE* da OSF¹⁰⁹. Nem o cliente, nem a componente se apercebem que a ligação se tornou mais longa. Tanto no COM, como no DCOM os pedidos aos métodos remotos são síncronos.

Lado do servidor - Regra cardinal do COM é que um objeto COM só pode ser acedido por interfaces. O programa cliente está completamente isolado da implementação do servidor através de interfaces. O cliente não sabe nada do objeto COM ou da classe C++ que implementa o objeto COM, apenas consegue ver a interface. Do lado do servidor, a interface e os objetos partilhados são identificados pelo GUI¹¹⁰ no ficheiro IDL. A linguagem IDL utilizada pelo COM/DCOM é conhecida por MIDL¹¹¹. A especificação da interface é compilada pelo compilador padrão IDL da Microsoft (MIDL)(Markus Horstmann, 1997), que cria o código de esboço (*stub*) de servidores e o *proxy* de clientes, no entanto o código também pode ser gerado por outros compiladores da Microsoft (Visual C++¹¹²).

¹⁰⁸*Lightweight Remote Procedure Calls.*

¹⁰⁹Organização fundada em 1988 para criar um padrão aberto para a implementação de um sistema operativo Unix.

¹¹⁰*Graphical user interface.*

¹¹¹*Microsoft Interface Definition Language.*

¹¹²Ferramenta de desenvolvimento de *software* da Microsoft, utilizando a linguagem C++.

Lado do cliente - Do lado do cliente a implementação varia com tipo de servidor (InProc ou EXE). O cliente acede sempre aos métodos expostos pela interface. Quando termina de utilizar o objeto, o cliente liberta o método da interface, o que resulta num decréscimo do contador de referência do servidor. Depois de libertados os recursos, as ligações RPC são fechadas.

MTS - Todas as arquiteturas apresentadas até ao momento são arquiteturas cliente-servidor de duas camadas, o que torna a gestão de recursos do servidor uma responsabilidade do programador. A Microsoft introduziu uma arquitetura de três camadas conhecida como MTS¹¹³, no entanto o nome foi atribuído de forma não totalmente correta, porque gere mais do que transações, possui uma terceira camada intermédia (*middleware*) que gere a criação / remoção / pedido de objetos de servidor. De fato, MTS é um componente COM que:

- Gere recursos de sistema (e.g., processos, *threads* e ligações).
- Gere criação / execução e remoção de objetos de servidor.
- Cria automaticamente e controla as transações.
- Gere a segurança, de forma a controlar o acesso por parte dos utilizadores.
- Fornece ferramentas para configuração, gestão e publicação de componentes de aplicação.

O benefício chave é que permite que os programadores se concentrem na lógica de negócio, em vez de desenvolver a camada intermédia. Num sistema típico são utilizados ODBC¹¹⁴ ou OLEDB¹¹⁵ para ligar a um servidor de base de dados. Funcionalidade muito importante fornecida pelo MTS é a componente *Just in Time Activation*, que reduz drasticamente o número de componentes e ligações à base de dados, quando utilizado em conjunto com *Object Pooling*. Baseado na ideia de que a maior parte de tempo um utilizador escreve ou pensa, apenas 10% do tempo é

¹¹³Microsoft Transaction Server.

¹¹⁴Open Database Connectivity.

¹¹⁵Object Linking and Embedding, Database.

utilizado verdadeiramente com os objetos instanciados. Isto significa um enorme desperdício de recursos, se cada cliente tem um componente dedicado e uma ligação à base de dados sempre ativos na camada intermédia.

COM+ - COM e MTS surgiram com o Windows NT. Com o aparecimento do Windows 2000, os mesmos serviços eram fornecidos pelo COM+, que é a combinação de COM e MTS e oferece novos serviços que não eram fornecidos pelo COM e MTS. COM+ não substitui o COM e MTS, apenas os estende. O COM+ é compatível e assim migrar pacotes MTS é simples. As melhorias incluem um novo modelo de *threading* e uma melhor gestão de ligações à base de dados. Também melhora o COM, fornecendo implementações de interfaces padrão e automatizando a gestão de código empresarial (*housekeeping*), que ocupava cerca de 30% do tempo dos programadores.

Vantagens da utilização da tecnologia COM/DCOM:

- Enorme base de dados de utilizadores e componentes.
- Integração binária, reutilização de *software* e desenvolvimento em várias linguagens de programação.
- Atualizações de *software* online, permitindo a atualização de componentes de aplicações, sem recompilar, relincar ou reiniciar.
- Múltiplas interfaces por objeto.
- Várias ferramentas disponíveis, com padronização e automatização no desenvolvimento de componentes.

2.2.3 XML-RPC

XML-RPC¹¹⁶ é um protocolo de pedidos de procedimento remoto (*Remote Procedure Calling*) que funciona através da Internet. Uma mensagem XML-RPC é um pedido

¹¹⁶ *eXtensible Markup Language - Remote Procedure Call protocol.*

HTTP POST. O corpo do pedido é em XML. Um procedimento é executado num servidor e é retornado um resultado formatado em XML. Os parâmetros do procedimento podem ser escalares, números, *strings*, datas, etc. Também podem ser registos complexos ou listas de estruturas. Existem vários padrões de XML-RPC definidos que especificam: tipos de dados (vetores, booleanos, *string*, etc.), estruturas de pedido / resposta e especificações de transporte.

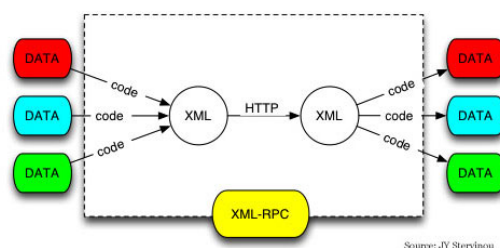


Figura 2.9 – XML-RPC(Stervinou, 2003)

Características do XML-RPC:

- Simples de utilizar, desenvolver e consumir.
- Utiliza XML.
- Muito mais leve do que o SOAP.
- Não necessita / suporta WSDL¹¹⁷.
- Não suporta internacionalização (*i18n*).
- Apenas permite uma forma de serialização.

Exemplo de um pedido XML-RPC:

```

1  POST /RPC2 HTTP/1.0
2  User-Agent: Frontier/5.1.2 (WinNT)
3  Host: betty.userland.com
4  Content-Type: text/xml
5  Content-length: 181

```

¹¹⁷ *Web Services Description Language*.

```
6
7
8     <?xml version="1.0"?>
9     <methodCall>
10    <methodName>examples.getStateName</methodName>
11    <params>
12    <param>
13    <value><i4>41</i4></value>
14    </param>
15    </params>
16    </methodCall>
```

Listagem 2.1: Pedido XML-RPC**Formato do cabeçalho XML-RPC:**

O URI¹¹⁸ na primeira linha do cabeçalho pode não ser especificado. Pode ser vazio, um *slash*(/), se o servidor estiver a responder apenas a pedidos XML-RPC. No entanto se o URI for utilizado, pode ser utilizado para encaminhar (*routing*) os pedidos para o código que responde aos pedidos XML-RPC. O *User-Agent* e o *Host* devem ser especificados. O *Content-Type* é *text/xml* e o *Content-Length* deve ser especificado e deve estar correto.

Formato do corpo de dados XML-RPC:

O corpo (*payload*) é em XML com uma estrutura única *<methodCall>*. A estrutura *<methodCall>* deve conter um sub-item *<methodName>* do tipo *string*(maiúsculas e minúsculas de A-Z, 0-9, *underscore*, ponto, traço, *slash*) que contém o nome do método que vai ser chamado e cujos caracteres do nome serão interpretados pelo servidor. Se o método possuir parâmetros, o *<methodCall>* deverá conter o sub-item *<params>*. O sub-item *<params>* pode possuir qualquer número de elementos *<param>* e cada um contém um elemento *<value>*.

Escalar *<value>*:

<value> pode ser um escalar, o tipo é indicando com uma *tag* dentro do *<value>*.

¹¹⁸ *Uniform Resource Identifier*.

Tipos definidos:

TAG	TIPO	EXEMPLO
<code><i4></code> ou <code><int></code>	inteiro de 4 bytes com sinal	-12
<code><boolean></code>	0 (falso) ou 1 (verdadeiro)	1
<code><string></code>	string	olá mundo
<code><double></code>	real com dupla precisão	-12.214
<code><dateTime.iso8601></code>	data/hora	19980717T14:08:55
<code><base64></code>	binário codificado em base64	eW91IGNhbid0IHJlYWQgdGhpcyE=

Tabela 2.2 – Tipos definidos em XML-RPC.

Se não for especificado o tipo, então é assumido o tipo *string*.

Estrutura `<struct>`:

O `<value>` também pode ser do tipo `<struct>`. O elemento `<struct>` contém elementos `<member>` e cada `<member>` contém um elemento `<name>` e um elemento `<value>`.

Exemplo de estrutura `<struct>` de dois elementos:

```

1    <struct>
2    <member>
3    <name>lowerBound</name>
4    <value><i4>18</i4></value>
5    </member>
6    <member>
7    <name>upperBound</name>
8    <value><i4>139</i4></value>
9    </member>
10   </struct>
```

Listagem 2.2: Estrutura `<struct>` do XML-RPC

As estruturas `<struct>` podem ser recursivas e qualquer `<value>` pode conter um elemento `<struct>` ou qualquer outro tipo, incluindo `<array>`.

Vetor `<array>`:

O `<value>` também pode ser do tipo `<array>`. O elemento `<array>` possui um único elemento `<data>` que pode conter qualquer número de elementos *value*.

```

1    <array>
2    <data>
3    <value><i4>12</i4></value>
4    <value><string>Egypt</string></value>
5    <value><boolean>0</boolean></value>
```

```
6      <value><i4>-31</i4></value>
7      </data>
8      </array>
```

Listagem 2.3: Elemento `<array>` do XML-RPC

Os elementos `<array>` não possuem nome e vários tipos podem ser misturados. Podem ser recursivos e qualquer `<value>` pode conter um `<array>` ou qualquer outro tipo, incluindo o tipo `<struct>`, descrito acima.

Exemplo de uma resposta XML-RPC:

```
1      HTTP/1.1 200 OK
2      Connection: close
3      Content-Length: 158
4      Content-Type: text/xml
5      Date: Fri, 17 Jul 1998 19:55:08 GMT
6      Server: UserLand Frontier/5.1.2-WinNT
7
8
9      <?xml version="1.0"?>
10     <methodResponse>
11     <params>
12     <param>
13     <value><string>South Dakota</string></value>
14     </param>
15     </params>
16     </methodResponse>
```

Listagem 2.4: Resposta do XML-RPC**Formato da resposta XML-RPC:**

A não ser que exista um erro de baixo nível, a resposta será sempre um '200 OK'. O *Content-Type* é *text/xml*. O *Content-Length* deve estar presente e ser correto. O corpo da resposta é XML, o elemento `<methodResponse>`, que contém um elemento `<params>`, que possui um único `<param>`, que por seu lado contém um `<value>`. O elemento `<methodResponse>` também pode possuir um elemento `<fault>`, que contém um elemento `<value>`, que é uma `<struct>` que possui dois elementos: um `<faultCode>` do tipo `<int>` e um `<faultString>` do tipo `<string>`. O elemento `<methodResponse>` não pode ter ao mesmo tempo um elemento `<fault>` e um `<params>`.

Formato da resposta com falhas (`<fault>`) em XML-RPC:

```
1      HTTP/1.1 200 OK
2      Connection: close
3      Content-Length: 426
4      Content-Type: text/xml
5      Date: Fri, 17 Jul 1998 19:55:02 GMT
```

```
6      Server: UserLand Frontier/5.1.2-WinNT
7
8
9      <?xml version="1.0"?>
10     <methodResponse>
11     <fault>
12     <value>
13     <struct>
14     <member>
15     <name>faultCode</name>
16     <value><int>4</int></value>
17     </member>
18     <member>
19     <name>faultString</name>
20     <value><string>Too many parameters.</string></value>
21     </member>
22     </struct>
23     </value>
24     </fault>
25     </methodResponse>
```

Listagem 2.5: Resposta com falhas em XML-RPC

Estratégias e objetivos do XML-RPC:

- Um dos objetivos deste protocolo é manter a compatibilidade com diferentes ambientes. As *firewalls* apenas precisam de vigiar os HTTP POST cujo *Content-Type* seja *text/xml*.
- Formato simples e extensível. Um programador deve facilmente entender um pedido XML-RPC e adapta-lo às suas necessidades.
- Facilidade de implementação de forma a ser executado em diversas plataformas ou diferentes sistemas operativos.

2.2.4 SOAP

O protocolo SOAP é uma versão modificada e mais potente do XML-RPC. É baseado no WSDL (*Web Services Description Language*) e no UDDI¹¹⁹.

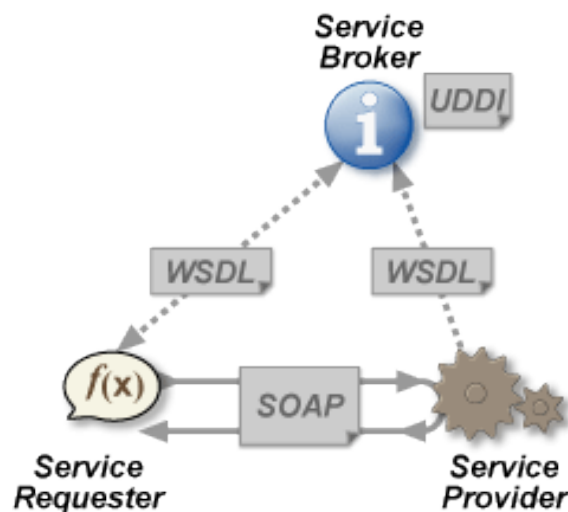


Figura 2.10 – SOAP(Voormann, 2006)

As principais especificações são o SOAP 1.1(W3C, 2000) de maio de 2000 e o SOAP 1.2(W3C, 2000) de abril de 2007. As especificações especificam tipos de dados, estrutura, *namespaces*/atributos.

Exemplo de estrutura de pedido SOAP:

```

1    <?xml version="1.0"?>
2    <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
3    >
4    <soap:Header>
5    </soap:Header>
6    <soap:Body>
7    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
8    <m:StockName>IBM</m:StockName>
9    </m:GetStockPrice>
10   </soap:Body>

```

¹¹⁹ *Universal Description, Discovery and Integration.*

```
10 </soap:Envelope>
```

Listagem 2.6: Exemplo de estrutura de pedido SOAP

```
1 float getPrice(String IBM) {  
2     return stockPrice;  
3 }
```

Listagem 2.7: Exemplo de código de pedido SOAP

Exemplo de estrutura de resposta SOAP:

```
1 <?xml version="1.0"?>  
2 <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"  
3 >  
4 <soap:Header> </soap:Header>  
5 <soap:Body>  
6 <m:GetStockPriceResponse>  
7 <m:Price>34.5</m:Price>  
8 </m:GetStockPriceResponse>  
9 </soap:Body>  
</soap:Envelope>
```

Listagem 2.8: Exemplo de estrutura de resposta SOAP

A combinação entre SOAP, WSDL e UDDI definem o modelo geral da arquitetura do serviço *web*. O SOAP define determinadas funções chave necessárias ao ambiente distribuído de computação, nomeadamente:

- Possibilidade de mecanismo de troca de mensagens *stateless* num único sentido.
- Modelo de processamento de mensagens para os nós SOAP.
- Definição de mensagem estruturada e abstrata, de forma a poder executar diferentes serializações.
- Definição de *bindings*¹²⁰ para protocolos de transporte (HTTP, SMTP).

¹²⁰Ligação de dados é uma técnica geral que une duas fontes de dados/informações e as mantém em sincronia em um processo que estabelece uma ligação entre interface de utilizador da aplicação e a lógica de negócio.

- Mecanismo de extensão através de elementos do cabeçalho, permitindo funcionalidades em diferentes *namespaces* como *WS-addressing*.
- Modelo de tratamento de falhas.

O SOAP define a estrutura da mensagem utilizada para troca de informação entre o fornecedor e consumidor do serviço *web*. Os elementos de uma mensagem genérica SOAP são:

- **SOAP Envelope** - O envelope (elemento raiz) é o contendor do elemento opcional *header* e o elemento obrigatório *body*.
- **SOAP Header** - O SOAP *header* pode conter informação de controlo. A informação é organizada por blocos, cada um com o seu *namespace* individual, definindo o *schema*. O *header* é extensível, *namespaces* arbitrários podem ser conjugados de forma a obter uma determinada forma de processamento.
- **SOAP Body** - Transporta a informação da aplicação codificada em XML. O *schema* do *body* é definido pelo WSDL.

O SOAP *header* possui atributos que definem o comportamento do processamento nos nós SOAP, são eles:

- **env:role** – Define o nó que vai processar o *header*, por exemplo: *next*, *none*, *ultimateReceiver*.
- **env:mustUnderstand** – *Header* processado pelo nó alvo, por exemplo: *true*, *false*.
- **env:relay** – Se *true* e se o **env:mustUnderstand** for inexistente ou *false*, então um nó intermediário deve dar seguimento (*relay*) ao bloco *header* e não o deve processar, por exemplo: *true*, *false*.
- **env:encodingStyle** – Define a codificação ou o esquema de serialização. Por defeito *soap-encoding* (codificação XML), por exemplo: qualquer esquema (*schema*) de codificação definido por um *namespace* URI.

SOAP define o mapeamento entre mensagens da aplicação definidas por um *WSDL schema* e as mensagens físicas transportadas na rede.

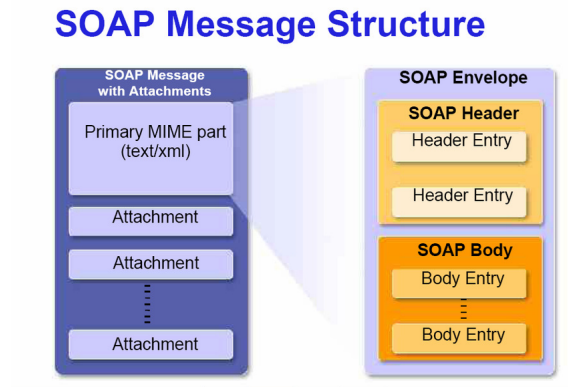


Figura 2.11 – Estrutura da mensagem SOAP (Ghossoon M. Waleed, 2009)

O SOAP também define os papéis (*roles*) do SOAP *sender* e SOAP *receiver*. O modelo de mensagem básica é de apenas um sentido e *stateless*, i.e., o emissor envia a mensagem ao recetor sem guardar o estado da troca de mensagens. O padrão mais comum de troca de mensagens é *request-response* entre o serviço web emissor e recetor. O SOAP também define outra entidade intermediária (*intermediary*) que executa determinadas funções de filtragem de mensagens e de *cache*¹²¹.

O SOAP 1.2 define a modelação de mensagens SOAP, aplicando o modelo RPC, através de:

- *Binding* de transporte – Se o *binding* de transporte do SOAP é HTTP, o SOAP-RPC mapeia o HTTP *request* e *response* em que endereço do HTTP URI é o endereço do processador SOAP.
- Padrão de troca de mensagens – O SOAP-RPC utiliza o *SOAP-Response message exchange pattern* (MEP).
- Identificação de recurso – O SOAP-RPC recomenda a identificação do recurso pelo URI e pelo nome da operação (com argumentos).

¹²¹Componente de *hardware* ou *software* que armazena dados, para que pedidos futuros aos mesmos recursos sejam atendidos mais rapidamente.

- Codificação do pedido (*request*) – Modelado como uma estrutura XML.
- Codificação da resposta (*response*) – Modelado como uma estrutura XML.
- *SOAP fault element* – O SOAP utiliza o elemento XML *fault* como parte do corpo do SOAP para indicar os erros.

Em termos de segurança foi implementada a extensão WSS (*WS-Security*). A extensão *Web Services Security* faz parte da especificação da OASIS¹²². O protocolo especifica como podem ser aplicadas a integridade e a confidencialidade às mensagens e permite a comunicação de vários formatos de *tokens* de segurança, como SAML¹²³, X.509¹²⁴ e Kerberos¹²⁵. O seu principal objetivo é fornecer segurança *end-to-end* através de assinaturas XML e encriptação XML.

As estratégias e objetivos do SOAP passam pela versatilidade, utilização de diversos protocolos (HTTP, SMTP¹²⁶, etc.), utilização de diversas ferramentas de automação, utilização de XML, utilização de WSDL e pela elevada verbosidade.

2.2.5 REST

Representational State Transfer, abreviado como REST¹²⁷, não é protocolo, mas sim uma aproximação arquitetural. Na transmissão pode ser utilizado o tradicional XML ou o JSON¹²⁸.

¹²² *Organization for the Advancement of Structured Information Standards.*

¹²³ *Security Assertion Markup Language.*

¹²⁴ Padrão ITU-T (*Telecommunication Standardization Sector*) para infraestruturas de chaves públicas (ICP).

¹²⁵ Protocolo de rede que permite comunicações individuais seguras e identificadas numa rede insegura.

¹²⁶ *Simple Mail Transfer Protocol.*

¹²⁷ *Representational State Transfer.*

¹²⁸ *JavaScript Object Notation* - Formato compacto, de padrão aberto independente, de troca de dados simples e rápida (*parsing*) entre sistemas, especificado por Douglas Crockford em 2000, que utiliza texto legível por humanos, no formato atributo-valor (natureza auto-descritiva).

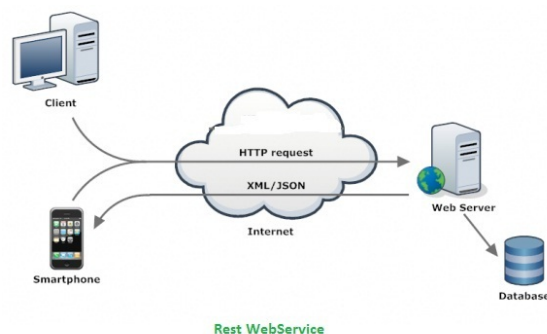


Figura 2.12 – Serviço *web* REST (Parvez, 2017)

Os princípios do REST são: *stateless*¹²⁹, utilização dos métodos HTTP para as operações *CRUD*¹³⁰, estrutura de diretório, utilização adequada dos tipos *MIME*¹³¹. Cada sistema informático utiliza recursos. Os recursos podem ser páginas *web*, imagens, vídeos, informação empresarial ou outro tipo de informação representável num sistema informático. Em suma, um serviço REST deve ter as seguintes características e propriedades que serão descritas mais em pormenor:

- Representação.
- Mensagens.
- URI.
- Interface uniforme.
- *Stateless*.
- Ligações entre recursos.
- *Caching*.

¹²⁹Nenhuma informação é retida pelo remetente ou pelo destinatário, o que significa que são agnósticos do estado um do outro.

¹³⁰*Create, Read, Update e Delete*.

¹³¹*Multipurpose Internet Mail Extensions*.

Representação

Pode ser utilizado qualquer formato para representação de recursos, pois o REST não especifica nenhuma restrição. Por exemplo, pode ser utilizado JSON:

```
1  { "employees": [  
2    { "firstName": "John", "lastName": "Doe" },  
3    { "firstName": "Anna", "lastName": "Smith" },  
4    { "firstName": "Peter", "lastName": "Jones" }  
5  ] }
```

Listagem 2.9: Formato JSON

Ou XML:

```
1  <employees>  
2  <employee>  
3    <firstName>John</firstName> <lastName>Doe</lastName>  
4  </employee>  
5  <employee>  
6    <firstName>Anna</firstName> <lastName>Smith</lastName>  
7  </employee>  
8  <employee>  
9    <firstName>Peter</firstName> <lastName>Jones</lastName>  
10 </employee>  
11 </employees>
```

Listagem 2.10: Formato XML

Algumas características terão de ser respeitadas na representação:

- Formato de representação compatível com cliente e servidor.
- Subdividir grandes recursos em partes mais pequenas de forma a obter maior rapidez nos serviços.
- Representação capaz de estabelecer ligação entre recursos.

Mensagens

Cliente e servidor comunicam entre si através de mensagens HTTP. O pedido é enviado ao servidor por parte do cliente, que responde com uma mensagem que possui metadados¹³² sobre a própria mensagem.



Figura 2.13 – Formato do pedido REST

Pedido REST

<VERB> – Método HTTP (vid. tabela 2.3).

<URI> – URI do recurso sobre o qual se vai realizar a operação.

<VERSÃO HTTP> – Versão HTTP (“HTTP v1.1”).

<CABEÇALHO DO PEDIDO> – Metadados da mensagem de pedido organizados numa coleção de pares chave-valor.

<CORPO DO PEDIDO> – Conteúdo da mensagem de pedido.

¹³²Dados / informações que fornecem informações sobre outros dados.

Método HTTP	Operação
GET	Obter elementos de um recurso.
POST	Inserir elementos num recurso.
PUT	Atualizar elementos num recurso.
DELETE	Apagar elementos num recurso.
PATCH	Atualizar parte de elementos num recurso.
HEAD	Obter metadados.
OPTIONS	Obter detalhes de um recurso.
TRACE	Para depurar <i>proxies</i> .
CONNECT	Reencaminhar determinado protocolo através de um <i>proxy</i> HTTP.

Tabela 2.3 – REST - Métodos HTTP

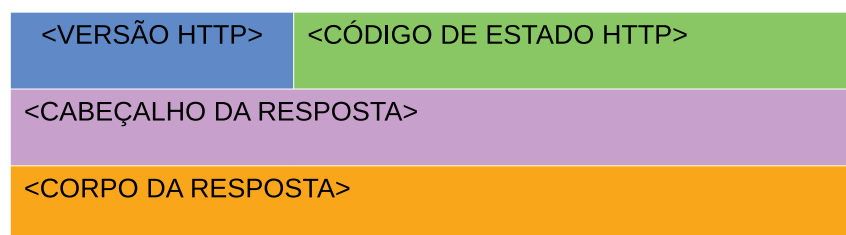


Figura 2.14 – Formato da resposta REST

Resposta REST

<VERSÃO HTTP> – Versão HTTP (“HTTP v1.1”).

<CÓDIGO DE ESTADO HTTP> – O servidor responde com um código de estado HTTP. O código de resposta encontra-se na tabela 2.4.

<CABEÇALHO DA RESPOSTA> – Metadados da mensagem de resposta organizados numa coleção de pares chave-valor.

<CORPO DA RESPOSTA> – Conteúdo da mensagem de resposta.

Código de estado HTTP	Informação
200	OK
201	Recurso criado
204	Sem conteúdo
400	Pedido mal formatado
401	Sem autorização
404	Não encontrado
405	Método não permitido
500	Erro de servidor

Tabela 2.4 – REST - Alguns códigos de estado HTTP

URI

Em REST cada URI deve representar um recurso, o método HTTP deve identificar a operação pretendida:

URI	Método HTTP	Ação
/status/	GET	Obter todos os estados
/status/6	GET	Obter estado com id 6
/status/	POST	Inserir um novo estado
/status/8	PUT	Atualizar estado com id 8
/status/8	DELETE	Remover estado com id 8

Tabela 2.5 – REST - Desenho de URI

Os serviços utilizam uma hierarquia de diretório para gerir os recursos. Também se podem enviar parâmetros no URI (*query parameter*):

GET http://server/service/persons?id=1

GET http://server/service/persons?groupid=10

Interface uniforme

Nos serviços REST deve ser utilizada uma interface uniforme. Para isso, existem métodos HTTP que são classificados (vid. tabela 2.6) como seguros (não afetam o valor original do recurso) e outros como idempotentes (operação que retorna o mesmo valor independentemente das vezes que é executada). Em síntese, a classificação de métodos facilita a previsão dos resultados no ambiente *web*, onde o mesmo pedido pode ser efetuado inúmeras vezes pelo cliente. Contudo, o grau de uniformização do serviço REST deve ser decidido pelo arquiteto do serviço.

Método HTTP	Idempotente	Seguro
GET	Sim	Sim
POST	Não	Não
PUT	Sim	Não
DELETE	Sim	Não
PATCH	Não	Não
HEAD	Sim	Sim
OPTIONS	Sim	Sim
TRACE	Sim	Sim
CONNECT	Não	Não

Tabela 2.6 – REST - Classificação de métodos HTTP

Stateless

Serviço RESTful é *stateless* pois não mantém o estado da aplicação para cada cliente. Um pedido não pode ser dependente de um pedido anterior, cada pedido é tratado pelo serviço de forma independente.

Desenho *Stateless*:

Pedido 1: GET http://server/service/persons/1 HTTP/1.1

Pedido 2: GET http://server/service/persons/2 HTTP/1.1

Desenho *Stateful*:

Pedido 1: GET http://server/service/persons/1 HTTP/1.1

Pedido 2: GET http://server/service/nextperson HTTP/1.1

O serviço deve ser desenhado de forma a que cada pedido nunca se possa referir a um pedido anterior.

Ligações entre recursos

O serviço deve ser desenhado de forma a que seja simples a ligação entre recursos (representação de recursos com ligações para outros recursos). Analogamente, como uma página *web* com referências para outras páginas, o serviço REST deve permitir uma navegação simples e intuitiva.

Caching

Caching permite que resultados sejam guardados em vez de serem gerados repetidamente e utilizados se um mesmo pedido surgir num futuro próximo. O processo de *caching* pode ser desempenhado pelo cliente, servidor ou componente intermédio como um servidor *proxy*. O *caching* tem de ser bem gerido, senão, o resultado pode ser indesejado. *Caching* pode ser controlado por cabeçalhos HTTP:

Cabeçalho HTTP	Aplicação
<i>Date</i>	Data/instante em que a representação foi gerada.
<i>Last modified</i>	Data/instante em que foi alterada a representação pelo servidor.
<i>Cache-control</i>	Cabeçalho HTTP 1.1 utilizado para controlo de <i>caching</i> .
<i>Expires</i>	Data/instante em que expira a representação.
<i>Age</i>	Duração em segundos desde o pedido ao servidor. Pode ser inserido por um componente intermediário.

Tabela 2.7 – REST - Cabeçalhos HTTP para controlo de *caching*

Valores dos cabeçalhos podem ser combinados com diretivas *Cache-control*, para verificar se os resultados guardados ainda são válidos. As diretivas mais comuns são:

Diretiva	Aplicação
<i>Public</i>	Por omissão. Cada componente pode fazer <i>cache</i> da representação.
<i>Private</i>	Componentes intermediários não podem fazer <i>cache</i> da representação, apenas o cliente e servidor.
<i>no-cache/no-store</i>	<i>Caching</i> desativo.
<i>max-age</i>	Duração em segundos desde a Data/instante do campo <i>date</i> em que a representação é válida.
<i>s-maxage</i>	Similar ao <i>max-age</i> mas válido para <i>caching</i> intermédio.
<i>must-revalidate</i>	Indica que a representação deve ser revalidada pelo servidor se o <i>max-age</i> ultrapassado.
<i>proxy-validate</i>	Similar a <i>max-validade</i> mas válido para <i>caching</i> intermédio.

Tabela 2.8 – REST - Diretivas de *Cache-control*

Características do REST

- Linhas de desenvolvimento mais abertas (código aberto).
- Utilização de XML ou JSON.
- Fácil de desenvolver e manter.
- Depende de outros esquemas de segurança como *OAuth*¹³³ (Google, Facebook, Microsoft, Twitter¹³⁴, etc.).
- Utiliza apenas HTTP.

¹³³Padrão aberto para autenticação, comumente utilizado para permitir que os utilizadores da Internet se possam autenticar em sites de terceiros utilizando as suas contas da Google, Facebook, Microsoft, Twitter, etc.

¹³⁴Serviço de notícias e serviço de redes sociais norte-americano nos quais os utilizadores postam e interagem com mensagens conhecidas como “*tweets*”.

2.2.6 REST vs. SOAP

Em termos de protocolo de transporte o serviço REST utiliza unicamente o protocolo HTTP/HTTPS, enquanto o serviço SOAP (independente do protocolo de transporte) pode utilizar vários protocolos: TCP, HTTP/HTTPS, SMTP, etc. Em termos de formato o SOAP está limitado ao formato especificado em XML, por outro lado no serviço REST pode ser definido o formato, sendo os mais utilizados o XML e o JSON, constituindo desvantagem em termos de interoperabilidade e vantagens em termos da variedade de formatos disponíveis para a codificação das mensagens (Wagh and Thool, 2012). No serviço SOAP existe especificamente a descrição do serviço em WSDL, enquanto no serviço REST existem formas de descrever os serviços como as linguagens de desenho / descrição da API, (Swagger, RAML, API blueprint, etc.) que serão abordadas na secção seguinte. Embora tanto o SOAP como o REST possam utilizar o HTTPS que aumenta o nível de segurança, o serviço SOAP permite utilizar determinados mecanismos (*WS-Security*) para aumentar a segurança independentemente do protocolo de transporte utilizado. Outra diferença entre estas duas tecnologias é que para o REST todas as operações / recursos estão disponíveis na *web*, enquanto que para o SOAP a *web* é apenas uma das hipóteses existentes para a publicação de um serviço. O REST suporta todo o tipo de dados diretamente, enquanto que o SOAP utiliza anexos por exemplo no envio de informação em binário. A largura de banda consumida pela arquitetura SOAP é bastante superior quando comparada com a largura que é necessária na tecnologia REST (Wagh and Thool, 2012). O SOAP sobre HTTP utiliza POST com XML bastante complexo, enquanto o REST pode utilizar simplesmente o GET, o que também é vantajoso em termos de utilização de servidores *proxy* e *reverse-proxy* (facilidade de utilização de mecanismos de *cache*).

2.2.7 Conclusão

Serviços *web* são bastantes versáteis e são parte integrante da *web* programável. Em termos de interoperabilidade desempenham uma importante função nas transações

de serviços empresariais (B2B¹³⁵) e na exposição de serviços ao utilizador final, facilitando a ligação de serviços heterogéneos. A grande generalidade dos serviços *web* baseiam-se em protocolos de Internet abertos e padronizados, o que é positivo, mas ao mesmo tempo negativo, pois a verdadeira interoperabilidade é alcançada através do sacrifício da largura de banda de dados utilizada.

¹³⁵ *Business-to-business.*

2.3 Desenho / documentação automática de API

2.3.1 Introdução

API¹³⁶ consiste num conjunto de padrões e procedimentos que os programas de *software* podem seguir para comunicar entre si. Serve de interface entre vários programas de *software* e facilita a sua interação, de forma semelhante a uma interface de utilizador que facilita a interação entre humanos e computadores.

Ao longo dos anos foram vários os desenvolvimentos registados na evolução das API. Com o aparecimento dos sistemas distribuídos nos anos 70, as API tiveram uma grande evolução. Um dos principais avanços foi o *Message Oriented Middleware*, respondendo às necessidades específicas da EAI¹³⁷, oferecendo ligação entre sistemas distribuídos utilizando integração API. O exemplo mais notório foi o IBM¹³⁸ *MQSeries*¹³⁹ que introduziu tecnologias de troca de mensagens entre sistemas distribuídos.

No final da década de 80 as API tiveram outra grande evolução com o aparecimento da programação orientada a objetos e as aplicações complexas puderam organizar-se como objetos.

Durante a década de 90 os sistemas distribuídos tornaram-se ainda mais populares com o surgimento da *www* comercial e evolução das tipologias cliente-servidor. Também novas técnicas de programação orientada a objetos emergiram o que facilitou o acesso remoto a instâncias de objetos. Tim Berners-Lee no início da década surgiu também com o prototipo do primeiro navegador e com ele a primeira página HTML. Em 1995 surgiu o JavaScript, contribuindo significativamente para a evolução das API, nomeadamente no desenvolvimento de aplicações do lado do cliente. Finalmente a história e integração de API teve no final da década o

¹³⁶ *Application Programming Interface*.

¹³⁷ *Enterprise Application Integration*.

¹³⁸ Empresa multinacional americana de tecnologia da informação sediada em Armonk, Nova York, Estados Unidos, com operações em mais de 170 países.

¹³⁹ Família de *software* cujos componentes são utilizados para unir outras aplicações de *software* para que possam trabalhar juntas.

contributo do SOA¹⁴⁰, que é uma abordagem arquitetural que permite a criação e reutilização de serviços de negócio interoperáveis e partilhados entre aplicações e empresas.

Nesta década os programadores aperceberam-se que o modelo *www* existente e a sua infraestrutura técnica poderia ser utilizada para melhorar a camada intermédia (*middleware*) utilizada até este ponto.

Atualmente vivemos os tempos da *Web* API, que começou por volta de 2005. A integração de API fez com que surgissem plataformas como o *Ebay*¹⁴¹ ou a *Amazon*¹⁴² que hoje em dia gozam de enorme popularidade(EBizMBA, 2018).

Na implementação de serviços *web* na tecnologia REST não existem padrões definidos para a documentação e descrição de um determinado serviço. Por conseguinte existem alguns métodos (ferramentas) utilizados para a descrição de uma API REST, que permitem de forma automática manter a documentação atualizada e sincronizada com o desenvolvimento da API. Posteriormente serão revistas algumas linguagens de especificação como o RAML, Slate, API blueprint, I/O Docs e o Swagger / OpenAPI.

¹⁴⁰ *Service-Oriented Architecture*.

¹⁴¹ Empresa de comércio eletrónico fundada nos Estados Unidos, em Setembro de 1995, por Pierre Omidyar.

¹⁴² Empresa americana de comércio eletrónico e computação na nuvem sediada em Seattle, Washington, fundada por Jeff Bezos em 5 de julho de 1994.

2.3.2 RAML

RAML ([RAML Workgroup, 2013](#)) é a *RESTful API Modeling Language* que facilita a gestão de toda API desde a concepção à partilha. É baseada em YAML¹⁴³ e fornece toda a informação necessária à descrição de API REST. Incentiva a reutilização, possibilita a descoberta e a partilha de padrões e um dos seus principais objetivos é unificar critérios e incentivar as melhores práticas na descrição de API([Stowe, 2015](#)).

2.3.3 Slate

Slate ([Lord, 2013](#)) é uma linguagem que permite criar documentação estática e responsiva¹⁴⁴ de uma API. Tem como principais características: desenho limpo e intuitivo, toda a documentação gerada numa única página, utiliza *Markdown*¹⁴⁵, escrita de código em várias linguagens, realce de sintaxe de código (*syntax highlighting*¹⁴⁶), deslocamento automático de conteúdos, permitidas contribuições publicas de edição de documentação, suporte para linguagens RTL¹⁴⁷, etc.

2.3.4 API blueprint

API blueprint ([API Blueprint, 2016](#)) é uma conjunto de ferramentas de desenvolvimento de documentação de todo o ciclo de vida de uma API, disponível ao desenvolvimento individual ou coletivo. A sintaxe é concisa e expressiva. Tem como principais características: colaboração / desenvolvimento de API /

¹⁴³Formato de serialização (codificação de dados) de dados legíveis por humanos e inspirado em linguagens como XML, C, Python, Perl, assim como o formato de correio eletrónico especificado pela RFC 2822.

¹⁴⁴Abordagem ao design da *web* que torna as páginas da *web* renderizadas em uma variedade de dispositivos e tamanhos de janelas ou telas.

¹⁴⁵Linguagem simples de marcação originalmente criada por John Gruber e Aaron Swartz. *Markdown* converte texto em HTML válido.

¹⁴⁶Funcionalidade disponível em alguns editores de texto que apresenta texto - em especial código fonte - numa formatação específica para cada categoria de termos.

¹⁴⁷*Right-to-left*.

governance (Drake and Force, 2005) e entrega, código aberto (Licença MIT¹⁴⁸), em repositório no *Github*¹⁴⁹, construído para permitir melhores desenhos de API através de abstração, diversas ferramentas que garantem o desenvolvimento / *governance* e entrega, etc.

2.3.5 I/O Docs

I/O Docs (Mashery, 2011) é um sistema de documentação interativo de *web API RESTful*. A API é definida a nível de recursos, métodos e parâmetros através de um *schema* JSON gerando uma interface cliente em JavaScript. Os pedidos da API podem ser executados através da interface cliente, que são reencaminhados pelo servidor I/O Docs para gerar dados formatados em JSON ou XML. *Tags* de texto básicas de HTML são habilitadas no *shcema* JSON.

2.3.6 Swagger / OpenAPI

Swagger / OpenAPI (Open API Initiative, 2016) é um conjunto de ferramentas utilizadas para desenho / documentação / testes / distribuição de uma API. Constitui a mistura de ferramentas livres e comerciais desenvolvidas por engenheiros e gestores. O Swagger / OpenAPI é construído pela SmartBear Software¹⁵⁰. Assim este sistema suporta o desenvolvimento de API nas várias fases do seu ciclo de vida: desenho, construção, documentação, teste, padronização.

¹⁴⁸Universidade privada de pesquisa localizada em Cambridge, Massachusetts, Estados Unidos. Fundada em 1861, em resposta à crescente industrialização dos Estados Unidos, o MIT adotou um modelo europeu de universidade politécnica e salientou a instrução laboratorial em ciência aplicada e engenharia.

¹⁴⁹Plataforma de hospedagem de código-fonte com controlo de versão utilizando o Git.

¹⁵⁰Empresa privada de tecnologia da informação que fornece ferramentas para monitorização de desempenho de aplicações, desenvolvimento de *software*, testes de *software* e gestão de API.

2.3.7 SERIN

SERIN (Lira et al., 2015) é uma *IDL* (*Interface definition language*) utilizada para definir modelos abstratos e conceituais que descrevem serviços *web* RESTful de forma sintática e semântica. O servidor implementa a interface dos serviços RESTful para acessar a recursos que são instâncias da classes especificadas na interface abstrata. O SERIN permite anotações que determinam as operações (*GET*, *POST*, *PUT* e *DELETE*) que os serviços RESTful podem desempenhar, bem como especificam restrições de integridade.

2.3.8 Conclusão

A utilização de linguagens de especificação de API é vantajosa em termos de poupança de tempo na escrita de código, pois existem ferramentas que permitem gerar uma grande parte de código. Em termos colaborativos também é importante, pois o processo criativo / especificativo pode envolver todas as partes ligadas à utilização da API. A qualidade da API implementada também pode ser assegurada pois existem mecanismos de validação da API com a especificação desenvolvida. No caso de uma API REST, como é o caso da implementação prática desta dissertação, é notável e necessária a utilização deste tipo de linguagens, pois não existem mecanismos próprios de especificação. Em fase final de implementação da API, também é agilizado o processo de publicação e geração de documentação interativa.

2.4 *Frameworks* de desenvolvimento (REST)

2.4.1 Introdução

Selecionou-se um leque variado de *frameworks*¹⁵¹ consoante a sua popularidade(Slant, 2018) e consoante as diferentes linguagens de programação em que foram desenvolvidas, dando uma especial atenção às *frameworks* desenvolvidas em Node.js.

As *frameworks* permitem um desenvolvimento *web* mais rápido e simplificam o trabalho de manutenção, o que aumenta significativamente o tempo de vida útil das aplicações. As *frameworks* de código aberto trazem muito menos restrições em termos de licenciamento. Quando estas são utilizadas por um enorme número de programadores, a sua documentação e suporte tendem a ser maiores e melhores. De notar o aumento na eficiência quando é utilizada uma *framework* de desenvolvimento, pois diminui drasticamente a necessidade de reescrever código. Também existe uma grande vantagem em termos de segurança, pois os problemas costumam ser identificados e resolvidos de forma célere com o recurso às comunidades que suportam o *software*.

2.4.2 Express

Desenvolvida por:	TJ Holowaychuk, StrongLoop e outros
Desenhada por:	TJ Holowaychuk
Desenvolvida em:	JavaScript; Node.js
Criada em:	Novembro de 2010
Última versão estável:	4.16.3 / 12 de março, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	MIT License
Repositório:	https://github.com/expressjs/express
Página oficial:	expressjs.com

¹⁵¹Conjunto de classes implementadas em uma linguagem de programação específica, utilizadas para auxiliar o desenvolvimento de software.

Notas:

Express (Holowaychuk, 2010) é uma *framework* de desenvolvimento *web* desenvolvida em Node.js. É gratuita e de código aberto e utiliza uma licença MIT. Segundo o seu autor é baseada na *framework* Sinatra, com um funcionamento simples que pode ser estendido através da utilização de *plugins*. Express é o *backend* da *stack* MEAN¹⁵², juntamente com o sistema de base de dados MongoDB¹⁵³ e *framework* de *frontend*¹⁵⁴ AngularJS¹⁵⁵.

Outras características:-**2.4.3 Flask-RESTful**

Desenvolvida por:	-
Desenhada por:	Kevin Burke, Kyle Conroy, Ryan Horn, Frank Stratton, Guillaume Binet
Desenvolvida em:	Python
Criada em:	Julho de 2000
Última versão estável:	0.2.12 / 4 de março, 2014
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	BSD License
Repositório:	https://github.com/flask-restful/flask-restful
Página oficial:	https://flask-restful.readthedocs.io/

Notas:

Flask-RESTful (Kevin Burke, 2017) é uma extensão da *framework* Flask¹⁵⁶ que

¹⁵²*Stack* de *software* JavaScript gratuita e de código aberto para criar sites dinâmicos e aplicativos da *web*. A *stack* MEAN é MongoDB, Express.js, AngularJS e Node.js.

¹⁵³*Software* de base de dados orientado a documentos de multi-plataforma gratuito e de código aberto. Classificado como um programa de base de dados NoSQL, o MongoDB utiliza um esquema de documentos JSON.

¹⁵⁴Camada de apresentação de um *software*.

¹⁵⁵*Framework* JavaScript código aberto, mantida pela Google, utilizada no desenvolvimento de aplicações *single-page*.

¹⁵⁶*Framework web* escrita em Python e baseada na biblioteca WSGI Werkzeug e na biblioteca de Jinja2.

acrescenta suporte para construir API REST de forma rápida. É uma camada de abstração que trabalha em conjunto com a biblioteca / ORM¹⁵⁷.

Outras características:-

2.4.4 Phoenix

Desenvolvida por:	-
Desenhada por:	Chris McCord
Desenvolvida em:	Elixir
Criada em:	Maior de 2014
Última versão estável:	1.3.4 / 6 de agosto, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	MIT License
Repositório:	https://github.com/phoenixframework/phoenix
Página oficial:	https://phoenixframework.org/

Notas:

Phoenix (McCord, 2014) é uma *framework* de desenvolvimento *web* escrita na linguagem de programação funcional Elixir¹⁵⁸. A Phoenix utiliza um modelo *server-side* MVC¹⁵⁹ e é baseada na biblioteca Plug¹⁶⁰ e na *framework* Erlang Cowboy¹⁶¹.

Outras características: alta performance e aplicações *web* escaláveis.

¹⁵⁷ *Object-relational mapping*.

¹⁵⁸ Linguagem de programação funcional, concorrente e de utilização genérica que é executada na *Erlang virtual machine* (BEAM).

¹⁵⁹ *Model-view-controller*.

¹⁶⁰ Biblioteca de desenvolvimento.

¹⁶¹ Servidor HTTP para Erlang.

2.4.5 Spring boot

Desenvolvida por:	-
Desenhada por:	Rod Johnson
Desenvolvida em:	Java; Groovy
Criada em:	Agosto de 2013
Última versão estável:	2.0.6 / outubro, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	Apache License 2.0
Repositório:	https://github.com/spring-projects/spring-boot
Página oficial:	https://spring.io/projects/spring-boot

Notas:

Spring boot ([Software, 2013](#)) é uma *framework* desenvolvida em Groovy¹⁶².

Outras características:

Criação de aplicações - Criação de aplicações *standalone* Spring¹⁶³.

Servidor web direto embebido - Tomcat¹⁶⁴, Jetty¹⁶⁵, Undertow¹⁶⁶.

Configuração automática - Configuração automática do Spring e outras bibliotecas.

Caraterísticas *production ready* - métricas, *health check*¹⁶⁷ e configuração exteriorizada¹⁶⁸.

Sem geração automática (?) - Sem geração automática de código e sem necessidade de configuração de XML.

¹⁶²Linguagem de programação orientada a objetos desenvolvida para a plataforma Java como alternativa à linguagem de programação Java.

¹⁶³*Framework* de desenvolvimento de aplicações em Java.

¹⁶⁴Servidor *web* Java, mais especificamente, um *container* de *servlets*.

¹⁶⁵Servidor HTTP e *Servlet Container* 100% escrito em Java. É o grande concorrente do Tomcat que ficou famoso por ter sido utilizado como o *servlet container* do JBoss.

¹⁶⁶Servidor *web* de alto desempenho escrito em Java.

¹⁶⁷Cada aplicação *web* precisa de verificação de integridade e requer uma lista de verificação. As verificações de integridade garantem que uma aplicação esteja a funcionar como esperado.

¹⁶⁸Exteriorização de configuração para permitir utilização do mesmo código de aplicação em diferentes ambientes.

2.4.6 Django REST Framework

Desenvolvida por:	Tom Christie
Desenhada por:	Tom Christie
Desenvolvida em:	Python
Criada em:	Fevereiro de 2011
Última versão estável:	3.8.2 / 6 de abril, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	Encode OSS Ltd
Repositório:	https://github.com/encode/django-rest-framework
Página oficial:	https://www.django-rest-framework.org/

Notas:

Django REST Framework ([Christie, 2011](#)) é uma *framework* para desenvolver *web* API.

Outras características: API navegável na *web*, suporta OAuth1a e OAuth2, serialização que suporta modelos ORM e não ORM, personalizável, documentação extensível, etc.

2.4.7 Laravel

Desenvolvida por:	Taylor Otwell
Desenhada por:	Taylor Otwell
Desenvolvida em:	PHP
Criada em:	Junho de 2011
Última versão estável:	5.7.2 / 6 de setembro, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	MIT License
Repositório:	https://github.com/laravel/framework
Página oficial:	http://laravel.com

Notas:

Laravel (Otwell, 2011) é uma *framework* PHP que possui como principais características: nova estrutura de diretórios, sistema de *caching* no encaminhamento (*route caching*), sistema de autenticação embebido, suporte a múltiplos sistemas de ficheiros, sistema de injeção melhorado e contratos¹⁶⁹.

Outras características:-

2.4.8 Zend Framework

Desenvolvida por:	Zend Technologies
Desenhada por:	-
Desenvolvida em:	PHP
Criada em:	Março de 2006
Última versão estável:	3.0.0 / 8 de junho, 2018 ⁶
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	New BSD license
Repositório:	https://github.com/zendframework/zendframework
Página oficial:	http://framework.zend.com/

Notas:

Zend Framework (Technologies, 2006) é uma *framework* desenvolvida em PHP e possui como características principais: sistema de encaminhamento (*routing*), contentor *PSR-11*¹⁷⁰, sistema de aplicação de modelos (*templating*) e sistema de tratamento de erros (*error handling*).

Outras características:-

¹⁶⁹Conjunto de interfaces que definem os principais serviços fornecidos pela *framework*.

¹⁷⁰Interface para aumentar a interoperabilidade da aplicação.

2.4.9 CakePHP

Desenvolvida por:	Cake Software Foundation, Inc.
Desenhada por:	-
Desenvolvida em:	PHP
Criada em:	Abril de 2005
Última versão estável:	3.2.8 / 24 de abril, 2016
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	MIT License
Repositório:	https://github.com/cakephp
Página oficial:	https://cakephp.org/

Notas:

CakePHP ([Cake Software Foundation, 2005](#)) é uma *framework* desenvolvida em PHP que possui as seguintes características: modelo de desenvolvimento MVC, ORM (*Object-relational mapping*), herança de classes¹⁷¹, facilmente extensível através da utilização de *plugins*¹⁷², sem configuração, validação embebida e suporte direto para operações CRUD.

Outras características:-

¹⁷¹Herança, ou *Inheritance* em inglês - Permite que classes compartilhem atributos e métodos, através de “heranças”.

¹⁷²Componente de *software* que adiciona um recurso específico a um programa de computador existente.

2.4.10 Restlet

Desenvolvida por:	-
Desenhada por:	Jerome Louvel
Desenvolvida em:	Java
Criada em:	2005
Última versão estável:	2.3.12 / 2 de outubro, 2017
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	Apache ver 2.0; CDDL ver 1.0; LGPL ver 2.1; LGPL ver 3.0; EPL ver 1.0
Repositório:	https://github.com/restlet/restlet-framework
Página oficial:	https://restlet.com/projects/restlet-framework/

Notas:

Restlet ([Restlet](#), 2005) é uma *framework* desenvolvida em Java com as seguintes características: flexibilidade na configuração, suporte nativo para REST, segurança e escalabilidade, suporte para multi-plataforma, solução completa para serviços *web*, conjunto extenso de conectores e compatibilidade com os padrões da *web* atuais.

Outras características:-

2.4.11 Spark

Desenvolvida por:	-
Desenhada por:	Per Wendel
Desenvolvida em:	Java
Criada em:	Julho de 2000
Última versão estável:	2.6.0 / 25 de abril, 2017
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	Apache License 2.0
Repositório:	https://github.com/perwendel/spark
Página oficial:	http://sparkjava.com/

Notas:

Spark (Wendel, 2011) é uma *framework* desenvolvida para Java e Kotlin¹⁷³. É executada num servidor embebido *web* Jetty, mas pode ser configurada para ser executada noutros servidores *web*. Não é baseada no modelo MVC, mas sim num modelo de criação rápida de aplicações através do menor esforço. Também suporta vários sistemas de modelos (*templates*).

Outras características:-**2.4.12 Sinatra**

Desenvolvida por:	Konstantin Haase
Desenhada por:	Blake Mizerany
Desenvolvida em:	Ruby
Criada em:	Setembro de 2007
Última versão estável:	2.0.3 / 8 de junho, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	MIT License
Repositório:	https://github.com/sinatra/sinatra/
Página oficial:	http://www.sinatrarb.com/

Notas:

Sinatra (Mizerany, 2007) é uma *framework* desenvolvida em Ruby on Rails¹⁷⁴ e o seu nome é em homenagem a Frank Sinatra¹⁷⁵. É desenvolvida em código aberto, suporta multi-plataforma e é executada sobre a interface do servidor *web* Rack¹⁷⁶. Tal como a *framework* Spark não utiliza o modelo MVC (ao contrário do Ruby on

¹⁷³Linguagem de programação de tipagem estática que é executada na máquina virtual Java (JVM) e também pode ser compilada em código-fonte JavaScript ou usar a infraestrutura do compilador LLVM.

¹⁷⁴*Framework* livre que promete aumentar velocidade e facilidade no desenvolvimento de sites orientados a base de dados (*database-driven web sites*), uma vez que é possível criar aplicações com base em estruturas pré-definidas.

¹⁷⁵Cantor, ator e produtor americano, sendo considerado um dos mais populares e influentes artistas musicais do século 20.

¹⁷⁶Interface modular e adaptável para o desenvolvimento de aplicações *web* em Ruby.

Rails), mas sim um modelo de desenvolvimento rápido.

Outras características:-

2.4.13 Restify

Desenvolvida por:	Restify team
Desenhada por:	-
Desenvolvida em:	Javascript; Node.js
Criada em:	Maior de 2011
Última versão estável:	7.2.1 / 7 de junho, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	MIT License
Repositório:	https://github.com/restify/node-restify
Página oficial:	http://restify.com/

Notas:

Restify (Team, 2011) é uma *framework* desenvolvida em Node.js. Está otimizada para desenvolver serviços *web RESTful* de alta performance. Assenta em características como: facilidade de produção de aplicações, mesmo modelo de *middleware* que a *framework* Express, facilidade de depuração e semanticamente correta (apoiada em RFC¹⁷⁷).

Outras características:-

¹⁷⁷*Request for Comments* - documentos técnicos desenvolvidos e mantidos pelo IETF (*Internet Engineering Task Force*).

2.4.14 SailsJS

Desenvolvida por:	Mike McNeil e outros
Desenhada por:	-
Desenvolvida em:	Javascript; Node.js
Criada em:	Julho de 2012
Última versão estável:	1.1.0-3 / 24 de agosto, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	MIT License
Repositório:	https://github.com/balderdashy/sails
Página oficial:	http://sailsjs.com/

Notas:

SailsJS (McNeil, 2012) é uma *framework* desenvolvida em Node.js.

Outras características: desenvolvida completamente em Javascript, suporta vários sistemas de base de dados, associações flexíveis na modelação de dados baseadas no modelo relacional (*one-to-many*, *many-to-many*)¹⁷⁸, auto geração de API REST, suporte nativo de *websockets*¹⁷⁹, declarativa / reutilizável / políticas de segurança implementadas, agnóstica no *frontend*, *pipeline*¹⁸⁰ de desenvolvimento de *assets* de *frontend*, estrutura de base sólida (baseada em Express, *socket.io*¹⁸¹), enorme comunidade de desenvolvimento, etc.

¹⁷⁸Um para muitos, muitos para muitos.

¹⁷⁹Tecnologia que permite a comunicação bidirecional por canais *full-duplex* sobre um único *socket Transmission Control Protocol* (TCP).

¹⁸⁰Cadeia de elementos de processamento (processos, *threads*, corrotinas, funções, etc.), organizados de modo que a saída de cada elemento seja a entrada do próximo.

¹⁸¹Biblioteca JavaScript para desenvolvimento de aplicações *web* em tempo real.

2.4.15 LoopBack

Desenvolvida por:	StrongLoop (pertence à IBM)
Desenhada por:	-
Desenvolvida em:	Javascript; Node.js
Criada em:	Junho de 2013
Última versão estável:	3.23.2 / outubro, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	MIT License
Repositório:	https://github.com/strongloop/loopback
Página oficial:	https://loopback.io/

Notas:

LoopBack (IBM, 2013) é desenvolvida em Node.js. Desenvolvida pela StronLoop¹⁸².

Outras características: desenvolvimento da *stack* completa, rápido desenvolvimento (assente em desenvolvimento modular), suportado pela IBM, multi-ferramentas fornecidas pela StronLoop, ORM com múltiplos *connectors* disponibilizados pela comunidade, controlo de acessos por utilizador baseado em *roles*, desenvolvido sobre Swagger, etc.

2.4.16 Gugamarket

Desenvolvida por:	-
Desenhada por:	Pliik
Desenvolvida em:	Javascript; Node.js
Criada em:	-
Última versão estável:	-
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	-
Repositório:	https://github.com/pliik/gugamarket
Página oficial:	https://www.gugamarket.com/

¹⁸²Empresa americana que pertence à IBM e trabalha com o Node.js para criar e oferecer suporte à *suite* StrongLoop e desenvolvimento de API para dispositivos móveis.

Notas:

Gugamarket (Pliik, 2015) é uma *framework* desenvolvida em Node.js.

Outras características: baseado em várias tecnologias base como: Express, Swagger, Mongoose¹⁸³, Jade¹⁸⁴, Passwordless¹⁸⁵ e Mocha¹⁸⁶, desenvolvimento da *stack* completa, etc.

2.4.17 Grails

Desenvolvida por:	Graeme Rocher
Desenhada por:	-
Desenvolvida em:	Java; Groovy
Criada em:	Outubro de 2005
Última versão estável:	3.3.8 / 10 de agosto, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	Apache License 2.0
Repositório:	https://github.com/grails/grails-core
Página oficial:	http://grails.org/

Notas:

Grails (Rocher, 2011) é uma *framework* desenvolvida em Groovy e baseada em Spring boot.

Outras características: baseada em Spring boot, curva de aprendizagem rápida, fácil integração com Java, integrada com *GORM*¹⁸⁷ / API REST / React¹⁸⁸ / Angular¹⁸⁹, extensível através da utilização de *plugins*, tecnologias de apresentação

¹⁸³Servidor *web* embebido multi-plataforma e biblioteca de rede com funções TCP, cliente e servidor HTTP, cliente e servidor *websocket*, cliente MQTT, etc.

¹⁸⁴Linguagem de programação orientada a objetos que possui uma grande integração com um sistema de gestão de base de dados orientado a objetos.

¹⁸⁵Autenticação por *token* via correio eletrónico.

¹⁸⁶*Framework* de testes Javascript para programas desenvolvidos em Node.js.

¹⁸⁷ORM do Grails.

¹⁸⁸Biblioteca JavaScript de código aberto para criar interfaces de utilizador.

¹⁸⁹Plataforma de desenvolvimento de aplicações *web* de código aberto e *frontend* baseado em TypeScript liderado pela equipa Angular da Google e por uma comunidade de indivíduos e corporações.

View¹⁹⁰, código aberto, capacidades de desenvolvimento assíncrono, linguagens *Domain-Specific*¹⁹¹ e suporte para diferentes IDE¹⁹².

2.4.18 Conclusão

Framework	Linguagem de programação	Específica para REST
Django REST Framework	Python	Sim
Flask-RESTful	Python	Sim
Laravel	PHP	Não
Zend Framework	PHP	Não
CakePHP	PHP	Não
Restlet	Java	Sim
Spark	Java	Não
Sinatra	Ruby on Rails	Não
Express	Node.js	Não
Restify	Node.js	Sim
SailsJS	Node.js	Não
LoopBack	Node.js	Sim
Gugamarket	Node.js	Sim
Spring boot	Groovy	Não
Grails	Groovy	Não
Phoenix	Erlang	Não

Tabela 2.9 – *Frameworks* de desenvolvimento (REST)

Na resolução de problemas e implementação de novas soluções, os programadores podem enveredar por várias vias: desenvolver uma solução própria ou utilizar bibliotecas cuidadosamente selecionadas ou ainda utilizar uma *framework*. Como o problema a resolver tem grandes probabilidades de ser um problema com solução já existente, torna-se cara e demorada a implementação de uma solução de raiz, só sendo válida essa implementação se for um problema único e sem solução existente. Normalmente o código fonte das *frameworks* não pode ser alterado, no entanto costumam ser fornecidos mecanismos de extensão que permitem o desenvolvimento de novos componentes. De entre as *frameworks* revistas algumas são específicas para

¹⁹⁰Tecnologia que permite gerar HTML, JSON e XML a partir de GSP, JSON Views e Markup Views.

¹⁹¹Linguagem de computador especializada para um domínio de aplicação específico, em contraste a uma linguagem de propósito geral.

¹⁹²*Integrated Development Environment*.

REST e outras podem facilmente ser estendidas para a utilização em API REST (vid. tabela 2.9). A utilização de *frameworks* também trás facilidade em termos de controlo do fluxo de desenvolvimento.

2.5 Microserviços

2.5.1 Introdução

A implementação de serviços *web* pode utilizar duas abordagens: arquitetura monolítica ou microserviços. Nesta secção serão apresentados os conceitos de arquitectura monolítica e de microserviços e será exposta a sua relevância no desenvolvimento de serviços *web*.

2.5.2 Arquitetura monolítica e microserviços

Em 2005 o Dr. Peter Rodgers falou pela primeira vez no termo *micro web services* (Rogers, 2005). Existe assim uma evolução de sistemas complexos e de arquitetura monolítica para uma arquitetura constituída por microserviços (vid. figura 2.15). Um microserviço é um pequeno serviço, granular, colaborativo e com interface perfeitamente definida (Dmitry Namiot, 2014). Pretende-se assim passar da conceção de serviços grandes e complexos para serviços pequenos e unificados numa estrutura maior e com capacidade evolutiva independente (Richardson, 2014b). As principais características de uma arquitetura monolítica são:

- Estrutura mais simples.
- Facilidade de desenvolvimento, testes e publicação.
- Bons resultados para aplicações pequenas.

No entanto, alguns problemas surgem da implementação de uma arquitetura monolítica, nomeadamente:

- Publicação contínua mais difícil, devido a indisponibilidade de todo o sistema durante a publicação de serviços.
- Menos flexibilidade em termos de escolha de diferentes tecnologias.
- Dificuldade de manutenção com aumento de complexidade.
- Dificuldade de coordenação de equipas de desenvolvimento.
- Degradação de qualidade de código fonte com o decorrer do tempo.
- Maior consumo de recursos (servidor de aplicação, IDE, etc.).
- Aumento de problemas de escalabilidade.

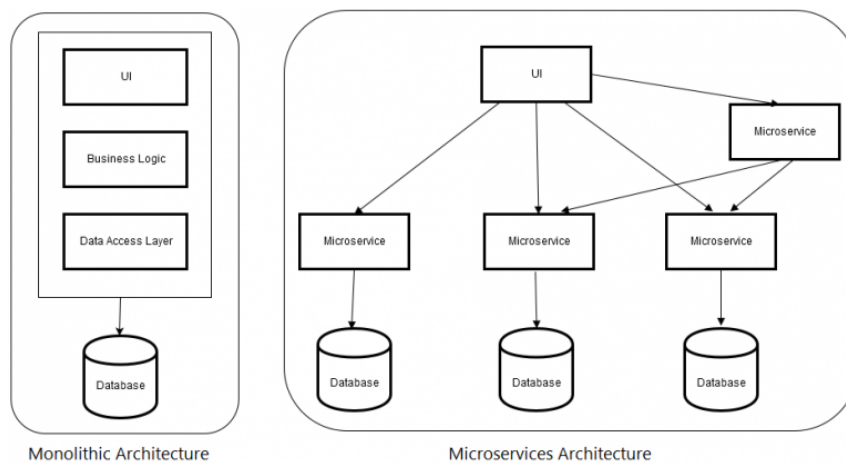


Figura 2.15 – Arquitetura monolítica e microserviços(Mishra, 2017)

Assim, microserviços é uma técnica de desenvolvimento de *software*, variante da arquitetura orientada a serviços (SOA), que estrutura uma aplicação como um conjunto solto de serviços. A grande vantagem da utilização de uma arquitetura de microserviços é que uma aplicação pode ser facilmente decomposta em serviços mais pequenos, o que melhora a modularidade e torna mais simples de perceber, desenvolver e testar(Chen, 2018). As principais propriedades de microserviços são:

- Baseado em componentes, através da utilização de serviços.
- Pequenos serviços, princípio da responsabilidade e coesão.

- Desenvolvimento e publicação de serviços de forma independente.
- Desenvolvimento focado em produtos em vez de projetos.
- Organização em torno de capacidade de negócio.
- Comunicação baseada no modelo REST (HTTP + JSON).
- Dados descentralizados.

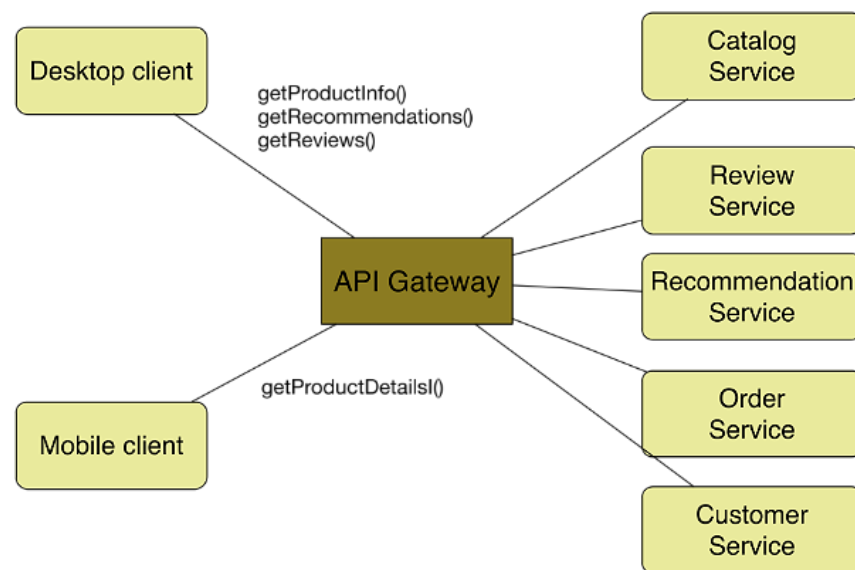


Figura 2.16 – Microserviços(Richardson, 2014a)

Assim cada microserviço pode ser desenvolvido de forma independente, sendo de importância crítica a forma como se interligam todos os microserviços (vid. figura 2.16), que terão de obedecer a regras pré-estabelecidas da interface de ligação das várias componentes. Algumas vantagens da utilização de microserviços são:

- Código mais legível (cada serviço corresponde a um aspeto específico de negócio).
- Facilidade de adoção de novas tecnologias, sem necessidade de recriação de todo o sistema.
- Modelo mais vantajoso em termos de publicação contínua.

A implementação de microserviços pode utilizar diferentes tipos de infraestrutura, a saber:

- Instância única ou múltiplas instâncias de um serviço num servidor físico ou virtual.
- Instância única por *container* (e.g., Docker¹⁹³).

2.5.3 Conclusão

A arquitetura modular dos microserviços permite que grandes projetos de *software* sejam divididos em componentes mais pequenos (módulos) independentes que comunicam entre si através de uma API. Grandes aplicações compostas por microserviços não serão significativamente afetadas em caso de falha de um dos módulos. Microserviços também permitem flexibilidade na escolha de novas tecnologias, pois não existem tantas dependências e torna-se simples reverter o desenvolvimento de cada um dos módulos, acrescentando-se que, com o aumento de simplicidade no desenvolvimento modular, também se discrimina melhor o funcionamento de cada serviço. Todavia alguns problemas podem surgir, pois o desenvolvimento de sistemas distribuídos pode ser complexo, cada módulo atua como um serviço independente e a intercomunicação entre todos os módulos tem de ser bem gerida. A gestão de transações também é crítica quando existem múltiplos acessos a múltiplas fontes de dados (base de dados). Em termos de testes de um sistema composto por microserviços a tarefa pode ser pesada, pois ao contrário de um sistema monolítico, cada microserviço tem de ser testado independentemente. Igualmente a publicação de um sistema modular deverá ser mais complicada, pois depende da coordenação dos vários microserviços, em oposição à publicação de um sistema monolítico.

¹⁹³Programa de computador que realiza virtualização a nível do sistema operativo, também conhecido como “containerização”.

2.6 Sistemas de gestão de base de dados

2.6.1 Introdução

Bases de dados são arquivos que armazenam diferentes tipos de dados e são configurados e geridos através de várias linguagens de programação. As bases de dados revestem-se de grande importância no contexto dos sistemas de informática (Writer, 2017), pois armazenam alguns dos bens mais importantes das instituições, que é a informação. De maneira a acompanhar o mercado cada vez mais competitivo, as empresas procuram inovar, de forma a ter a informação sempre disponível e com a maior rapidez possível, recorrendo assim à utilização de sistemas de gestão de base de dados, consequentemente ajudando a administração e equipas estratégicas no processo de decisão.

Em meados dos anos 60 surgiram as primeiras definições de base de dados. O desenvolvimento de tecnologia de base de dados pode ser dividida em três eras (conforme estrutura ou modelo): Navegacional, relacional/SQL e pós-relacional (Bachman, 1973).

2.6.2 DBMS Navegacional

Os DBMS¹⁹⁴ navegacionais surgiram no início dos anos 60 com os primeiros conceitos de bases de dados. Um dos primeiros sistemas que surgiram foi o CODASYL, que assentava o seu funcionamento em navegação manual de dados ligados em grandes redes. Mais tarde surgiram as *B-trees*¹⁹⁵ fornecendo diferentes caminhos para aceder aos dados. Em 1966 a IBM também lançou o seu próprio DBMS com o IMS¹⁹⁶ (Corporation, 2013).

¹⁹⁴ *Database management system.*

¹⁹⁵ Árvore B é uma estrutura de dados em árvore, auto-balanceada, que armazena dados classificados e permite pesquisas, acesso sequencial, inserções e remoções em tempo logarítmico.

¹⁹⁶ *Information Management System.*

2.6.3 DBMS Relacional

Em 1970 Edgar Codd da IBM descontente com as capacidades de pesquisa dos sistemas navegacionais lançou os conceitos dos primeiros DBMS relacionais(Codd, 1983). Assim surge o modelo relacional que dividia os dados numa série de tabelas normalizadas, com determinados elementos a serem movidos da tabela principal, para tabelas onde realmente sejam necessários. Os dados podem ser geridos nas tabelas e o DBMS trata do processamento necessário de forma a disponibilizar as *table views* ao utilizador/aplicação quando solicitado. Enquanto que no modelo navegacional era requerido que um programa tivesse de iterar inúmeras vezes para recolher registos, no modelo relacional, segundo sugestão de Edgar Codd, surge uma linguagem orientada para gestão de dados que mais tarde dá origem ao SQL. Eugene Wong e Michael Stonebraker, apoiados num artigo de Edgar Codd, surgem com o projeto INGRES¹⁹⁷. Com o passar dos anos o INGRES evolui para o padrão SQL. Durante a década de 70 e 80 existiram esforços para desenvolver sistemas integrados (*software* e *hardware*¹⁹⁸) de forma a aumentar a eficiência, reduzindo os custos, alguns desses esforços foram desenvolvidos por projetos como *IBM System/38* e a máquina de base de dados da *Britton Lee, Inc*¹⁹⁹, no entanto estes esforços não foram recompensados devido ao desenvolvimento dos computadores de utilização genérica. No início da década de 70 a IBM começou a trabalhar num prototipo chamado *System R* baseado nos conceitos de Edgar Codd. Mais tarde surgem as versões de produção SQL/DS e *Database 2 (DB2)*. Larry Ellison surgiu com a *Oracle Database* ou simplesmente *Oracle* baseado nos artigos da IBM do *System R*. Stonebraker a partir do INGRES também surge com o *PostgreSQL*²⁰⁰. Na Suécia, também baseados nos artigos de Edgar Codd, aparece o *Mimer SQL*. Em 1976 surgiu o modelo relação entidade (*ER model*) e ganhou popularidade devido a uma descrição mais agradável do modelo relacional. Durante a década de 80 e inícios de 90 e com a potenciação do computador pessoal aparecem produtos como a folha de cálculo *Lotus 1-2-3*²⁰¹

¹⁹⁷Sistema de gestão de base de dados.

¹⁹⁸Componentes físicos, tangíveis de um computador.

¹⁹⁹Empresa pioneira de sistemas de base de dados relacional.

²⁰⁰Sistema de base de dados objeto relacional (ORDBMS).

²⁰¹Programa de folha de cálculo da *Lotus Software*.

e o *dBASE*²⁰² tornando muito mais simples o processo de manipulação de dados e alheando o utilizador de operações de manipulação de ficheiros e gestão de recursos.

2.6.4 Orientada a objetos

Nos anos 90 e com o desenvolvimento da programação orientada a objetos, os programadores começaram a tratar os dados como objetos e as relações dos dados numa base de dados começaram a tornar-se relações entre objetos e os seus atributos. Em termos de programação surgiram as ORM (*Object-relational mappings*) de forma a estender o SQL e abraçar este novo paradigma nos modelos de base de dados.

2.6.5 NoSQL e NewSQL

Nos anos 2000 apareceu o NoSQL, que é muito rápido, não necessita de esquemas de tabela (*table schemas*) fixos, evitando operações de junção sobre dados não normalizados e também são sistemas desenhados para crescer horizontalmente (Cattell, 2011) (aumento do número de computadores na rede de processamento em vez do aumento dos recursos de cada computador). Os sistemas mais populares de NoSQL incluem MongoDB, Memcached²⁰³, Couchbase²⁰⁴, Riak²⁰⁵, Redis²⁰⁶, CouchDB²⁰⁷, Hazelcast²⁰⁸, HBase²⁰⁹ e *Apache Cassandra*²¹⁰. O

²⁰²Um dos primeiros sistemas de gestão de base de dados para microcomputadores.

²⁰³Sistema distribuído de *cache* em memória de propósitos gerais.

²⁰⁴*Software* distribuído de código aberto de base de dados orientado a documentos multi-modelo NoSQL e otimizado para aplicações interativas.

²⁰⁵*Software* de base de dados distribuído com armazenamento de dados em *key-value* NoSQL, com alta disponibilidade, tolerância a falhas, simplicidade operacional e escalabilidade.

²⁰⁶*Software* de base de dados de *key-values* distribuído de código aberto e com durabilidade opcional.

²⁰⁷*Software* de base de dados de código aberto que se concentra na facilidade de utilização e possui uma arquitetura escalável.

²⁰⁸Estrutura de dados na memória em código aberto baseada em Java.

²⁰⁹*Software* de base de dados distribuído em código aberto orientado para a coluna, modelado a partir do Google BigTable e escrito em Java.

²¹⁰*Software* de base de dados distribuído altamente escalável de segunda geração, que reúne a arquitetura do DynamoDB, da Amazon Web Services (AWS) e modelo de dados baseado no BigTable da Google.

NewSQL é um desenvolvimento recente do NoSQL que permite sistemas eficientes de grande carga de transações online (operações leitura-escrita), permitindo a utilização do SQL e mantendo as garantias das operações ACID²¹¹ dos sistemas tradicionais de base de dados. Os sistemas NewSQL incluem o Citus²¹², Google F1/Spanner²¹³, TiDB²¹⁴ (HTAP²¹⁵ distribuída e compatível com MySQL²¹⁶), ScaleBase²¹⁷, MemSQL²¹⁸, NuoDB²¹⁹, VoltDB²²⁰ e o CockroachDB²²¹.

2.6.6 Conclusão

Em síntese, os sistemas de gestão de base de dados são compostos por cinco componentes principais: dados, *software*, *hardware*, utilizadores e procedimentos. A utilização destes sistemas trazem algumas vantagens, em resumo: controlo de redundância de dados²²², inconsistência de dados pode ser evitada (alterações propagadas por toda a base de dados), dados são facilmente partilhados por múltiplas aplicações, integridade de dados pode ser reforçada (através de imposição de restrições), restrição e controlo de acessos e controlo concorrencial, simplicidade na aplicação de padrões (sistema centralizado mais simples de aplicar um padrão, do que a um conjunto de sistemas separados), sistemas automáticos de *backup* e *data recovery*, baixo custo de desenvolvimento e manutenção por parte do DBA²²³ (em comparação com informação guardada em sistemas de ficheiros), acrescentando-se que existe menos complexidade na aplicação de um modelo de

²¹¹ *Atomicity, Consistency, Isolation, Durability.*

²¹² Sistema de base de dados distribuído, instalado em servidores comuns e com *shard* (partição horizontal de dados numa base de dados ou motor de busca) e replicação transparentes.

²¹³ Base de dados NewSQL distribuída globalmente.

²¹⁴ Base de dados HTAP distribuída e compatível com MySQL.

²¹⁵ *Hybrid transaction/analytical processing.*

²¹⁶ Sistema de gestão de base de dados que utiliza a linguagem SQL como interface.

²¹⁷ Empresa que vendia *software* para implementar bases de dados MySQL distribuídas para computação na nuvem.

²¹⁸ Sistema distribuído em memória de gestão de bases de dados SQL.

²¹⁹ Empresa de bases de dados fundada em 2008 e com sede em Cambridge, Massachusetts.

²²⁰ Base de dados em memória projetada por Michael Stonebraker, Sam Madden e Daniel Abadi.

²²¹ Empresa de *software* que desenvolve sistemas de gestão de bases de dados para empresas.

²²² Repetição de dados.

²²³ *Database administrator.*

dados (modelo relacional). No entanto, também existem algumas desvantagens na utilização de sistemas de gestão de base de dados, em síntese: maior complexidade comparativamente a dados armazenados em sistemas de ficheiros, aumento de tamanho que surge com o aumento da complexidade e funcionalidades, grande impacto de falhas (falha de um componente afeta múltiplas aplicações devido à sua natureza centralizada), custos adicionais de *hardware* (mais discos, computadores mais potentes, etc.), diminuição de performance devido ao carácter genérico (agrega várias aplicações) comparativamente a sistemas de ficheiros dedicados a uma única aplicação, além disso, os custos de conversão de sistema de gestão de base de dados também tem tendência a ser elevados (técnicos especializados na conversão de sistemas de gestão de base de dados, formação de técnicos para trabalhar no novo sistema). O SIDE utiliza como sistema de gestão de base dados o MySQL, sendo um dos sistemas mais utilizados do mundo juntamente com o Oracle, Microsoft SQL Server, PostgreSQL e MongoDB(IT, 2018).

2.7 Métodos de autenticação

2.7.1 Introdução

Autenticação digital é o processo de identificar um indivíduo ou dispositivo através de um “*nome de utilizador*” e uma “*palavra-chave*”. Em sistemas de segurança a autenticação é diferente de autorização, que é o processo de permitir o acesso a um indivíduo / dispositivo a objetos de sistema baseado na sua identidade. A autenticação apenas assegura a identidade digital do indivíduo / dispositivo, não estabelece nenhuma regra relativa a direitos de acesso.

As formas de autenticação enquadram-se em quatro tipos de fatores(Stallings and Brown, 2014, p. 75):

- Alguma informação que utilizador sabe: palavra-passe, PIN²²⁴, etc.

²²⁴ *Persona Identification Number*.

- Alguma informação que utilizador possui: cartão de segurança, *token* de segurança, dispositivo implantado, etc.
- Alguma informação do que o utilizador “é”: sequência de ADN²²⁵, impressão digital, retina, etc.
- Alguma informação do que o utilizador “faz”: padrão de voz, padrão de escrita, ritmo de escrita, etc.

Os métodos de autenticação variam com nível de segurança fornecido pelo número e complexidade de fatores utilizados que se inserem em várias categorias:

- Autenticação de fator simples(Turner, 2016): mais fraco das categorias de autenticação, apenas um fator é utilizado na autenticação.
- Autenticação de fator duplo (2FA)(Turner, 2016): utilizados dois fatores na autenticação (e.g., combinação de cartão bancário e PIN).
- Autenticação de múltiplo fator(Turner, 2016): utilizados mais do que dois fatores na autenticação.
- Autenticação forte(Turner, 2016): utilizados dois ou mais fatores, mas os fatores utilizados tem de ser mutuamente independentes e pelo menos um fator tem de ser não reutilizável.
- Autenticação contínua(Brocardo et al., 2017): sistemas convencionais autenticam os utilizadores apenas quando entram no sistema, enquanto que este tipo de autenticação é executada continuamente (e.g., biometria comportamental baseada na forma de escrever de um utilizador).

Seguidamente serão abordados alguns métodos de autenticação (shamim Hassan, 2017) utilizados no desenvolvimento *web*.

²²⁵Ácido desoxirribonucleico.

2.7.2 Autenticação baseada em sessões

Como o protocolo HTTP é *stateless*, num pedido seguinte a aplicação não sabe se é a mesma pessoa ou dispositivo, para evitar este problema apareceu o sistema baseado em sessões / *cookies*²²⁶, o que torna o processo de autenticação *stateful*²²⁷. A sessão tem de ser guardada no servidor e no cliente. O servidor guarda o registo das sessões ativas (base de dados, memória, etc.) e o cliente utiliza uma *cookie* de autenticação.

2.7.3 Autenticação baseada em *tokens*

A autenticação baseada em *tokens*²²⁸ surgiu em grande força nos últimos anos devido ao aparecimento das *web API*, aplicações de página única e IOT²²⁹. Os *tokens* mais utilizados são os JWT²³⁰. Existem diversas implementações de sistemas de autenticação baseados em *tokens*, mas o JWT tornou-se quase um padrão.

2.7.4 Autenticação sem palavra-passe

Em vez de um utilizador enviar a tradicional combinação *email* / nome de utilizador e palavra-passe, envia apenas o *email* e seguidamente é enviado para o *email* um *link* que lhe permite aceder a uma zona da aplicação protegida. Em vez de um *link* também pode ser enviada uma OTP²³¹ para o *email* ou por SMS²³².

²²⁶Arquivo de computador ou pacote de dados enviados por um sítio de Internet para o navegador do utilizador, quando o utilizador visita o sítio de Internet.

²²⁷Suporta estados diferentes, reagindo à mesma entrada de maneira diferente, dependendo do estado atual.

²²⁸Dados que são utilizados em comunicações de rede (HTTP/HTTPS) para identificar uma sessão, uma série de trocas de mensagens relacionadas.

²²⁹*Internet Of Things*.

²³⁰*JSON Web Tokens*.

²³¹*One Time Passord*.

²³²*Short Message Service*.

2.7.5 *Single Sign On* (SSO)

O SSO²³³ possui um serviço central de autenticação, em que um utilizador pode navegar entre vários serviços apenas com uma autenticação única, exemplo deste tipo de autenticação é o serviço da Google. O utilizador autentica-se uma vez, o serviço central de autenticação cria uma *cookie* que persiste enquanto se navega nos serviços pertencentes aos mesmo grupo. Existem três entidades de confiança bem definidas neste processo de autenticação: o utilizador, o provedor de identidades (IDP²³⁴) que é o serviço central de autenticação e o provedor de serviços (SP²³⁵). O utilizador confia no IDP, o SP confia no IDP, por seu lado o SP pode confiar no utilizador.

2.7.6 *Sign-in Social*

Este método de registo / *Sign-in* é semelhante em alguns aspetos ao SSO, embora tecnicamente seja uma implementação diferente. Neste método a aplicação faz um registo inicial num provedor de uma rede social, então é fornecida uma *app_id* e *keys*(chaves) necessárias ao processo de configuração para a comunicação com o provedor da rede social. Algumas redes sociais utilizam o *OAuth1*²³⁶, *OAuth2*²³⁷. Existem várias bibliotecas que simplificam a implementação deste método de autenticação como o Passport²³⁸, etc.

2.7.7 Autenticação de dois fatores (2FA)

O método de autenticação 2FA aumenta a segurança do processo de autenticação, pois necessita de dois fatores para verificar a identidade do utilizador. Exemplos

²³³*Single Sign On.*

²³⁴*Identity provider.*

²³⁵*Service Provider.*

²³⁶*OAuth version 1.0.*

²³⁷*OAuth version 2.0.*

²³⁸*Middleware* de autenticação para Node.js.

de 2FA é a verificação “2 Step” do Facebook e da Google. No caso do Facebook (Facebook, 2018), pode ativar-se a autenticação de dois fatores e utilizar um fator adicional de autenticação:

- Códigos de mensagens de texto (SMS) a partir telemóvel.
- Códigos de acesso a partir de uma aplicação de autenticação de terceiros.

2.7.8 Conclusão

Portanto, a autenticação é importante para as empresas / organizações manterem as suas redes seguras, permitindo só acesso aos seus recursos protegidos por parte de utilizadores autenticados, além disso, as organizações também controlam quem tem acesso às suas redes, bem como que servidores e máquinas são acedidos. Em grandes organizações, a autenticação pode ser centralizada num sistema (*Single Sign On*), o que permite que vários sistemas utilizem o mesmo sistema de autenticação centralizado, o que exhibe vantagens em termos de custos e facilidade de manutenção, mas é um enorme ponto de vulnerabilidade em caso de comprometimento. As organizações também utilizam autenticação para permitir o acesso seguro de colaboradores a aplicações e redes privadas (VPN²³⁹). De referir que no SIDE e em grande parte dos sistemas de informação da UTAD são utilizados métodos de autenticação baseados em sessões e *Single Sign On*.

2.8 Controlo de acesso / Auditoria

2.8.1 Introdução

O processo de restringir o acesso de um grupo de utilizadores / utilizador a um recurso protegido é denominado controlo de acesso. Depois de autenticado, o utilizador poderá ser submetido a um processo de controlo de acesso, para

²³⁹ *Virtual Private Network*.

determinar se é permitido o acesso a determinado recurso protegido ou sistema, com efeito, o fato de um utilizador estar autenticado não significa que pode aceder a um recurso protegido, pois poderá ter de passar por um controlo de acesso (autorização). O processo de autenticação costuma decorrer antes do processo de autorização. Os termos autenticação e autorização são muitas vezes confundidos. Enquanto a autenticação é o processo de validar a identidade de um utilizador, a autorização é o processo de validar se um determinado utilizador pode ter acesso a um recurso protegido.

Também o registo de dados de eventos em ficheiro (*logs*) é um processo relevante numa aplicação, pois permite que o estado original de uma aplicação seja restabelecido ou se perceba um comportamento passado da aplicação, ainda mais, permite processos de auditoria ou diagnóstico a problemas funcionais ou de acessos indevidos a uma aplicação.

2.8.2 Controlo de acesso

Em sistemas informáticos normalmente as políticas de controlo de acesso enquadram-se nas seguintes categorias(Stallings and Brown, 2014, p. 116-133):

- Controlo de acesso discricionário (DAC²⁴⁰): neste modelo de controlo de acesso o proprietário de um objeto estabelece quem pode ter acesso e privilégios. Várias propriedades são importantes neste modelo: propriedade e data do objeto, privilégios e permissões.
- Controlo de acesso mandatário (MAC²⁴¹): neste modelo o sistema é que aplica as políticas de acesso, respeitando as configurações de privilégios (determinados pelo administrador de sistema) e os rótulos de informação (consoante classificação efetuada pelo gestor de informação).
- Controlo de acesso baseado em *roles* (RBAC²⁴²): controlo de acesso baseado em

²⁴⁰*Discretionary Access Control.*

²⁴¹*Mandatory Access Control.*

²⁴²*Role-Based Access Control.*

roles, a política de acesso é determinada pelo sistema e não pelo proprietário do objeto. Este modelo é não discricionário e é definido por três regras: atribuição de *roles*, autorização de *roles* e transação de *roles*.

- Controlo de acesso baseado em atributos (ABAC²⁴³): controlo de acesso não baseado nos direitos de acesso de um utilizador, mas sim nos atributos do utilizador. O motor que determina o acesso a um objeto tem de analisar os atributos necessários para permitir o acesso.

Em suma, autorização é o processo de permitir / rejeitar o acesso de um indivíduo / dispositivo a objetos de um sistema baseado em determinados critérios como a sua identidade (i.e., perfil / grupo a que pertence), localização, tempo (i.e., hora do dia, dia da semana, etc.), tipo de transação.

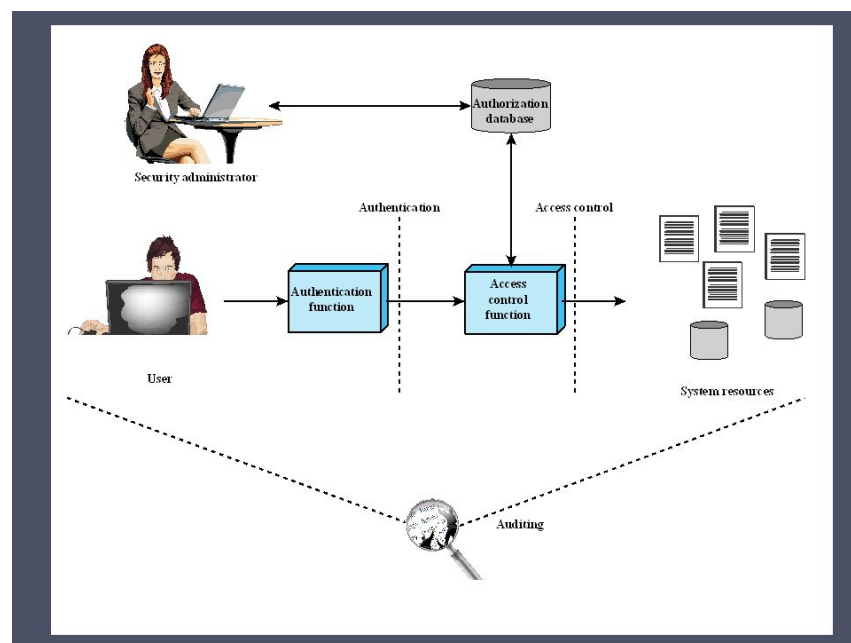


Figura 2.17 – Relação entre o controlo de acesso e as outras funções de segurança (Stallings and Brown, 2014, p. 115)

Na prática, determinado número de componentes podem cooperar e partilhar funções no controlo de acesso (vid. figura 2.17). Os sistemas operativos possuem

²⁴³ *Attribute-based Access Control*.

componentes de controlo de acesso simples ou complexos. Determinadas aplicações (e.g., sistemas de gestão de base de dados, *firewall*, etc.) também possuem funções de controlo de acesso. As *frameworks* de desenvolvimento revistas também podem ter implementado o controlo de acesso, como se pode ver na figura seguinte:

Framework	Controlo de acesso
Django REST Framework	Sim
Flask-RESTful	Sim
Laravel	Sim
Zend Framework	Sim (ACL)
CakePHP	Sim (ACL)
Restlet	Sim
Spark	Não
Sinatra	Sim (Rack middleware)
Express	Não
Restify	Não
SailsJS	Sim
LoopBack	Sim (ACL)
Gugamarket	Não
Spring boot	Sim (Spring security)
Grails	Sim (Spring security, Apache Shiro)
Phoenix	Não

Tabela 2.10 – Controlo de acesso em *frameworks* de desenvolvimento (REST)

2.8.3 Auditoria

A função de auditoria monitoriza e mantém registos de ocorrências de controlo de acesso aos recursos do sistema. O registo de eventos (*logs*) durante o controlo de acesso é o processo que permite associar um sujeito (indivíduo / dispositivo) com a execução de determinadas operações. Auditoria assume enorme relevância para detetar violações de segurança, recrear incidentes de segurança ou analisar comportamentos de uma aplicação. Frequentemente sistemas informáticos possuem métodos automáticos de auditoria que despoletam o registo de eventos (*logs*) sob determinados critérios ou ativação de *triggers*²⁴⁴ específicos, além disso, podem determinar algum tipo de resposta automática (e.g., restrição de operações, envio de aviso ao administrador do sistema, etc.).

Existem determinados requerimentos para efetuar uma auditoria de

²⁴⁴Recurso de programação executado sempre que ocorre o evento associado.

segurança(Stallings and Brown, 2014, p. 581): definição de eventos, deteção de eventos, registo de eventos, ferramentas / interfaces de análise de registo de eventos, garantia de não comprometimento das ferramentas de análise e menor efeito possível das ferramentas de análise (não alteração de dados).

Também deve existir uma efetiva proteção / armazenamento dos dados de registos para auditoria(Stallings and Brown, 2014, p. 588):

- Leitura / escrita de ficheiro num servidor.
- Gravação em meios não regraváveis (e.g., cd-rom, dvd-rom).
- Registo em dispositivos de escrita única (e.g., impressora).

Assim, segundo Brown e Stallings(Stallings and Brown, 2014, p. 595), fazer auditoria a nível de sistema pode não ser suficiente para detetar problemas de funcionamento e problemas de lógica aplicacional. Pode ser necessário detetar em pormenor o comportamento da aplicação, para além da sua interação com o sistema operativo. A informação necessária para detetar ataques a nível de aplicação pode não existir ou ser difícil de extrair dos registos do sistema operativo.

2.8.4 Conclusão

O processo de controlo de acesso a sistemas informáticos é baseado na definição de políticas de acesso aos recursos protegidos. Frequentemente é necessário alterar ou remover a autorização de um utilizador a um determinado recurso (i.e., alteração ou remoção da regra de acesso do sistema informático), o que deverá ser simplificado e célere se existirem interfaces próprias para o efeito. Por conseguinte, a segurança de uma aplicação é reforçada se forem tomadas medidas que permitam a devida implementação do processo de autenticação, autorização e auditoria.

2.9 Ferramentas de testes de API

2.9.1 Introdução

As ferramentas de testes são *softwares* que permitem perceber quantos utilizadores concorrenciais a aplicação suporta sem grande degradação de uma boa experiência de utilização, permitem detetar potenciais *bottlenecks*²⁴⁵ da aplicação e detetar pontos de quebra e ajustes necessários da *stack tecnológica* utilizada no desenvolvimento, possibilitam a identificação do comportamento da aplicação em cenários próximos da realidade, permitem determinar tempos de resposta em situações de carga, possibilitam a identificação problemas de *hardware* ou *VM*²⁴⁶ e permitem perceber qual a perceção dos *end users*²⁴⁷ aos erros e problemas da aplicação sob carga. Assim num sistema em análise existem vários tipos de testes chave(Masood, 2015):

- Testes de performance (*performance test*) – Qualquer teste que se utilize para medir a performance, capacidade, fiabilidade, escalabilidade e / ou tempo de resposta / taxa de transferência de uma aplicação.
- Testes de carga (*load test*) – Quando se testa o comportamento de um sistema com um número grande de utilizadores durante uma determinada janela temporal.
- Testes de *stress* (*stress test*) – O objetivo destes testes é descobrir como um sistema se comporta sob condições extremas de utilização. Coloca-se propositadamente o sistema alvo sob condições extremas para provocar o seu colapso (e.g., sistema com menos memória, aumento de utilizadores, sistema com processadores menos potentes, etc.).
- Testes de pico (*spike test*) – Trata-se de um sub-teste do teste de *stress*, em que se submete o sistema a um aumento de carga acima de níveis espetáveis

²⁴⁵Ponto de estrangulamento, gargalo ou restrição é uma designação do componente que limita o desempenho ou a capacidade de todo um sistema, que se diz ter um estrangulamento.

²⁴⁶*Virtual Machine*.

²⁴⁷Utilizadores finais.

em períodos de tempo reduzido.

- Testes de absorção (*soak test*, *endurance test*) – Sistema é colocado em carga durante longos períodos de tempo, a duração do teste pode durar de um par de horas até um dia. Este tipo de teste pretende descobrir o comportamento de um sistema com condições semelhantes e correlacionadas com um cenário real. Assim, as condições de teste serão aproximadas da realidade.
- Testes de capacidade / escalabilidade (*capacity test*) – Ajuda a determinar qual a capacidade máxima de utilizadores que podem utilizar uma aplicação, não ultrapassando o tempo máximo de resposta especificado.

A tabela seguinte enumera algumas ferramentas de testes, evidenciando características como a licença de *software* e o tipo de plano de utilização:

FERRAMENTA	LICENÇA	PLANO
Apache JMeter	Apache License 2.0	Totalmente gratuito
BlazeMeter	Proprietária	Plano gratuito disponível
CloudTest	Proprietária	Plano gratuito disponível
Gatling	Apache License 2.0	Totalmente gratuito
Loader.io	Proprietária	Plano gratuito disponível
NeoLoad	Proprietária	Plano gratuito disponível
OpenSTA	GNU GPL 2.0	Totalmente gratuito
Siege	GPLv3 or later	Totalmente gratuito
WebLOAD	Proprietária	Plano gratuito disponível

Tabela 2.11 – Ferramentas de testes de API

Imediatamente a revisão irá incidir sobre as ferramentas de testes com licenças de código aberto e que sejam de utilização totalmente gratuita.

2.9.2 Apache JMeter

Desenvolvida por:	Apache Software Foundation
Desenvolvida em:	Java
Criada em:	Julho de 2000
Última versão estável:	5.0 / 18 de setembro, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	Apache License 2.0
Página oficial:	https://jmeter.apache.org/

Notas:

Apache JMeter (Halili, 2008) é uma aplicação de código aberto, feita em Java, construída para testar a eficiência de uma aplicação em testes de carga. Surgiu inicialmente para testar aplicações *web*, mas expandiu a sua utilização a outro tipo de aplicações.

Outras características: IDE com possibilidade de construção de planos, depuração e registo, suporta múltiplos protocolos / serviços / aplicações: *Web*(HTTP, HTTPS) / SOAP / REST / etc., modo de linha de comandos, geração de relatórios HTML, simplificação na extração de dados dos formatos mais conhecidos (HTML, JSON, XML, etc.), extensível através da utilização de *plugins*, capacidades de funcionamento *offline* com *caching*, suporta *multi-threading*, etc.

2.9.3 Gatling

Desenvolvida por:	Stéphane Landelle e Gatling Corp
Desenvolvida em:	Scala
Criada em:	Janeiro de 2012
Última versão estável:	3.0.0 / 23 de outubro, 2018
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	Apache License 2.0
Página oficial:	http://gatling.io/

Notas:

Gatling (Landelle, 2012) é uma ferramenta de testes escrita predominantemente em Scala.

Outras características: possibilidade de desenvolvimento de cenários de testes em Scala(*scripts*), HTTP *proxy recorder* em modo *standalone*, motor assíncrono não bloqueante para maximização de eficiência, suporta múltiplos protocolos (HTTPS, JDBC²⁴⁸, JMS²⁴⁹, etc.), desenvolvimento de testes em DSL²⁵⁰, múltiplas validações e *assertation*²⁵¹, geração de relatórios em HTML, etc.

2.9.4 OpenSTA

Desenvolvida por:	Cyrano
Desenvolvida em:	C++
Criada em:	-
Última versão estável:	1.4.4 / 19 de outubro, 2007
Sistema operativo / Plataforma:	Microsoft Windows
Licença:	GNU General Public License 2.0
Página oficial:	http://opensta.org/

Notas:

OpenSTA (CYRANO, 2001) é uma ferramenta de testes com interface gráfica originalmente desenvolvida em C++ por Cyrano.

Outras características: suporta HTTP/HTTPS, baseado na arquitetura CORBA, interface gráfica intuitiva, etc.

²⁴⁸ *Java Database Connectivity.*

²⁴⁹ *Java Message Service.*

²⁵⁰ *Domain-Specific Language.*

²⁵¹ Predicado que é inserido no programa para verificar uma condição que o programador supõe que seja verdadeira em determinado ponto.

2.9.5 Siege

Desenvolvida por:	Jeffrey Fulmer e outros
Desenvolvida em:	C
Criada em:	-
Última versão estável:	4.0.4 / 11 de setembro, 2017
Sistema operativo / Plataforma:	Multi-plataforma
Licença:	GPLv3 or later
Página oficial:	http://www.joedog.org/siege-home

Notas:

Siege (Fulmer, 2012) é uma ferramenta de testes multi-plataforma, que suporta Linux²⁵², BSD²⁵³, Solaris²⁵⁴, etc.

Outras características: suporta múltiplos protocolos como HTTP, HTTPS, FTP²⁵⁵ e suporta autenticação básica ou *cookies*, configuração de múltiplos clientes que colocam uma aplicação sob teste, etc.

2.9.6 Conclusão

Testes a uma aplicação denotam-se de grande importância, pois aproximam o comportamento da aplicação a um cenário real, além disso a aplicação pode ter comportamentos divergentes do esperado, pois a própria implementação / extensão da aplicação podem mudar de forma inesperada o seu comportamento. A contínua integração e desenvolvimento estão incompletos sem os devidos testes, por conseguinte o impacto a nível financeiro poderá ser acentuado. Os testes ao *software* de uma corporação revestem-se de demasiada importância para serem negligenciados. Paralelamente existem grandes vantagens em utilizar processos automatizados de testes, pois são revestidos de rapidez e eficiência.

²⁵²Sistemas operativos que utilizam o Kernel Linux.

²⁵³Sistema Operativo UNIX-Like.

²⁵⁴Sistema Operativo UNIX desenvolvido pela antiga Sun Microsystems, hoje subsidiária da Oracle.

²⁵⁵*File Transfer Protocol*.

Foram obtidos dados para entender qual o grau de utilização do SIDE. Em termos de acessos, a plataforma teve em média, segundo dados da Google Analytics²⁵⁶, 3663 utilizadores no ano letivo 2016/2017 e 3415 utilizadores em 2017/2018. Normalmente existem picos máximos de acesso que coincidem com o início e fim de cada período de aulas, assim, foi atingido o máximo do ano letivo 2016/2017 com 5215 utilizadores (na primeira semana de início do período de aulas do segundo semestre). O máximo do ano letivo 2017/2018 foi atingido também durante a primeira semana do início do período de aulas do segundo semestre com 5769 utilizadores.

²⁵⁶Serviço gratuito de registo e análise estatística da *web* oferecido pela Google, atualmente é uma plataforma dentro da marca *Google Marketing Platform*. A Google lançou o serviço em 2005 depois de adquirir a Urchin.



Trabalho desenvolvido

3.1 Introdução

O objetivo do trabalho da dissertação é desenvolver uma camada que permita interligar outros sistemas de informação ao SIDE e possibilite também a implementação de novas aplicações que utilizem a base de dados do SIDE, com possibilidade de agregação de dados provenientes de outras fontes de dados (outros sistemas de informação da UTAD).

A API REST implementada divide-se em vários módulos principais. A figura [3.1](#) exemplifica de um modo geral o sistema implementado, que é constituído por três módulos principais. O primeiro módulo trata do processo de autenticação. O segundo módulo encarrega-se do controlo e gestão de acesso aos recursos. Finalmente, o terceiro módulo processa o pedido e os dados são filtrados, agregados e encapsulados numa estrutura de resposta.

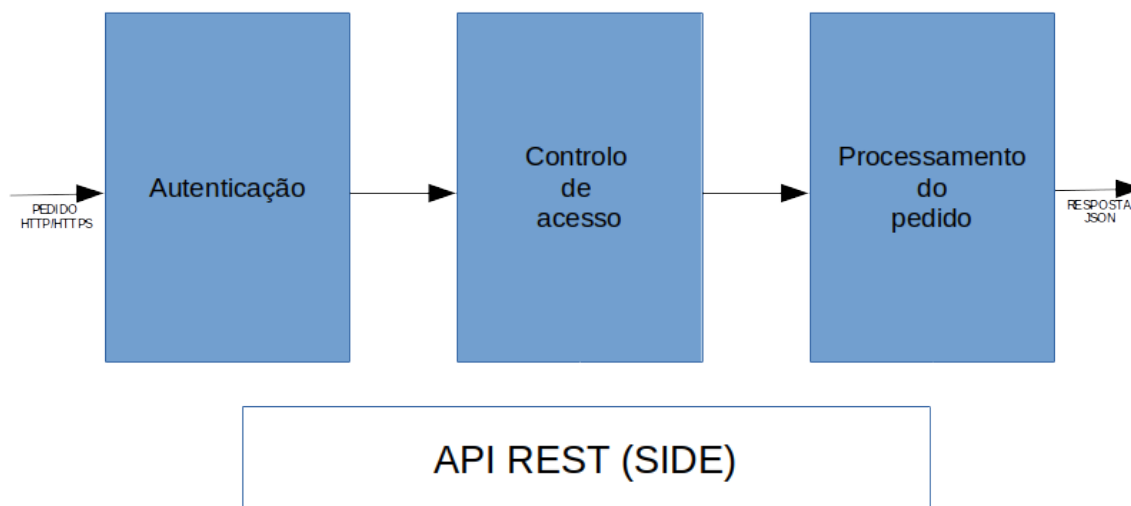


Figura 3.1 – Modelo implementado

A API REST comporta um módulo de autenticação, controlo de acesso, tratamento de pedidos e filtragem de dados. Foi utilizada a linguagem de programação Node.js e em termos de configurações existe um ficheiro que permite controlar determinadas configurações de ambiente como:

- Configurações de acesso a servidores de base de dados.
- Porta de aplicação HTTP/HTTPS.
- Número máximo de registos devolvidos.
- Versão de produção / desenvolvimento.
- Ativação de depuração.
- Ativação de auditoria e nível de auditoria.
- Ativação de autenticação / utilizador por defeito e *callback* de autenticação.

- Configurações de sessão.
- Modo de funcionamento em termos de *clusters*¹ e número de processos.

A listagem 3.1 é o ficheiro de configurações global *config.json*.

```
1 {
2   "db_sqlserver_User": "johndoe",
3   "db_sqlserver_Password": "*****",
4   "db_sqlserver_Host": "192.168.xxx.xxx",
5   "db_sqlserver_Name": "sigacaddatabase",
6   "db_Host": "192.168.xxx.xxx",
7   "db_Name": "sidedatabase",
8   "db_User": "api_user",
9   "db_Password": "*****",
10  "api_BeginPath": "/side_api",
11  "api_HTTP_Port": "8080",
12  "api_HTTPS_Port": "8443",
13  "api_MAXROWS_Length": 1000,
14  "api_PRODUCTION": 0,
15  "api_DEBUG": 1,
16  "api_AUDIT": 0,
17  "api_AUDITLEVEL": 10,
18  "api_USEAUTH": 0,
19  "api_USERNAME_AUTH_DUMMY": "johndoe",
20  "api_SESSION_EXPIRATION_TIME": 720000,
21  "api_SESSION_CHECK_CLEAN_TIME": 900000,
22  "api_CLUSTER_MANUAL": 0,
23  "api_CLUSTER_MANUAL_NUMPROCESSES": 2,
24  "api_AUTH_CALLBACK_URL": "http://192.168.xxx.xxx:6060/_authenticate"
25 }
```

Listagem 3.1: Ficheiro de configuração global

Para o trabalho desenvolvido nesta dissertação instalaram-se várias máquinas virtuais em VirtualBox², com as seguintes configurações e com os seguintes objetivos:

¹Computadores fracamente ou fortemente ligados que trabalham em conjunto.

²*Software* de virtualização inicialmente desenvolvido pela empresa Innotek depois comprado pela Sun Microsystems (pertence à Oracle) que, possibilita a instalação de sistema operativo como sistema operativo convidado (*Guest OS*) e permite a sua execução em ambiente virtual.

Nº	Sistema Operativo	Objetivo
1	Linux debian	MySQL Server
2	Linux Ubuntu	SQL Server
3	Linux debian	Application Server 1
4	Linux debian	Application Server 2

Tabela 3.1 – Máquinas virtuais instaladas

As máquinas virtuais instaladas possuem um processador Intel Core i7-7740X (4.3GHz) e possuem 2 GB de memória RAM. Na máquina virtual N^o 1 (vid. tabela 3.1) está instalado um servidor MySQL com uma cópia da base de dados do SIDE. A máquina virtual N^o 2 (vid. tabela 3.1) tem instalado um servidor de SQL Server com algumas tabelas de informação adicional de algumas entidades existentes na estrutura da base de dados do SIDE. O objetivo da máquina virtual N^o 3 (vid. tabela 3.1) é ser utilizada para servidor aplicativo da API REST e a máquina virtual N^o 4 (vid. tabela 3.1) é ser utilizada como servidor aplicativo de desenvolvimento de um caso de utilização da API REST. A base da implementação desta dissertação é a *framework* Express.

No desenvolvimento da API REST foram utilizados módulos externos que foram instalados com a ferramenta de gestão de pacotes de Node.js (npm). Os módulos externos utilizados foram:

- **express** - *Framework web* rápida e minimalista para Node.js.
- **http** - Módulo HTTP para Node.js que permite a transferência de dados através do protocolo HTTP.
- **promise** - Implementação simples de *Promises*. *Super set* dos *Promises* implementados no ES6³ e desenhados para fornecer as extensões necessárias para a utilização de *promises*.
- **bluebird** - Bluebird é uma biblioteca de *promises* com incidência na implementação de novas funcionalidades e a melhoria da performance.

³ECMAScript 6.

- **Sequelize** - Sequelize é um módulo ORM (*Object-relational mapping*) para PostgreSQL, MySQL, SQLite e Microsoft SQL Server baseado em *promises*. Suporte sólido de transações, relações, replicação de leitura, etc.
- **promise-mysql** - Promise-mysql é um *wrapper* mysqljs/mysql que engloba chamada a funções com *promises* bluebird.
- **body-parser** - *Middleware* para Node.js que analisa o corpo (*body*) de um pedido HTTP. Analisa o corpo antes dos *handlers*, disponível na propriedade *req.body*.
- **squel** - Construtor flexível de *querys* SQL para Node.js.
- **debug** - Módulo de depuração para Node.js.
- **connect-flash** - O *flash* é uma área especial das sessões utilizada para guardar mensagens. As mensagens são colocadas no *flash* e limpas quando são apresentadas ao utilizador.
- **path** - Módulo para Node.js que fornece utilitários para trabalhar com caminhos de ficheiros e diretórios.
- **fs** - Módulo para Node.js que fornece uma API para manipulação de ficheiros através de funções POSIX⁴.
- **bunyan** - Bunyan é uma biblioteca simples e rápida para produção de *logs* em JSON para os serviços Node.js.
- **async** - Async é um modulo que fornece funções para trabalhar com JavaScript assíncrono.
- **cluster** - Gestor de multi processador para Node.js.
- **cors** - Módulo para fornecer *middleware* Connect/Express para Node.js que permite ativar o CORS (*Cross-origin resource sharing*) com várias opções.
- **Passport** - Passport é uma *middleware* de autenticação para Node.js compatível com a *framework* Express. O principal objetivo do Passport é

⁴Interface Portável entre Sistemas Operativos - Família de normas definidas pelo IEEE para a manutenção de compatibilidade entre sistemas operativos.

autenticar pedidos, o que concretiza mediante a extensão com vários *plugins* referidos como estratégias.

- **mysql-password** - Implementação da função *password()* de MySQL para Node.js.
- **sqlite3** - Ligações assíncronas e não bloqueantes de SQLite3 para Node.js.
- **role-acl** - Controlo de acesso para Node.js baseado em perfis (*roles*), condições e atributos.
- **node-random-name** - Gerador de nomes aleatórios para Node.js.
- **merge** - Junta múltiplos objetos num objeto, criando opcionalmente uma cópia do objeto (clone).
- **mssql** - Cliente Microsoft SQL Server para Node.js.

Os métodos da API REST devolvem uma estrutura JSON *return_data* com o seguinte formato:

```
1 var return_data =  
2 {  
3   "error":0,  
4   "error_message":"","  
5   "pagination":false,  
6   "page":"","  
7   "limit":"","  
8   "data":""  
9 };
```

Listagem 3.2: Estrutura *return_data*

Os elementos da estrutura *return_data* possuem as seguintes funções:

- **error** - Se for erro a variável tem o valor 1 ou devolve 0 nos casos de não existir erro.
- **error_message** - A mensagem de erro quando existir, senão devolve vazio.
- **pagination** - Paginação ativa (*true*) ou não (*false*).

- **page** - Número de página quando a paginação está ativa.
- **limit** - Número de elementos por página.
- **data** - Os dados numa estrutura JSON ou vazio.

Quando é chamado qualquer um dos métodos da API REST, a estrutura `return_data` é colocada em modo *clean* (limpo) através do método interno *clean_return_data*:

```
1 function clean_return_data( )
2 {
3     return_data =
4     {
5         "error":0,
6         "error_message":"",
7         "pagination":false,
8         "page":"",
9         "limit":"",
10        "data":""
11    };
12 }
```

Listagem 3.3: Método *clean_return_data*

A API REST permite a utilização de métodos personalizados, para isso é necessário desenvolver um módulo que será colocado no diretório *custom* da API REST. O procedimento principal terá de ter o mesmo nome do módulo (vid. linha 2 da listagem 3.4), pois será esse o nome exportado pelo módulo. Por exemplo para o método *getinfousers*, terá que ser colocado o ficheiro *getinfousers.js* (módulo) no diretório *custom*. A estrutura do módulo é a seguinte:

```
1 (... )
2 var getinfousers =
3 function(params)
4 {
5     (... )
6 }
7
8 var info = "DESCRIPTION:";
```

```
10 var getdescription = function ()
11 {
12     return new Promise(function (resolve, reject)
13     {
14         if (typeof info === 'undefined')
15         {
16             var err = new Error("No info available.");
17             reject(err);
18         }
19         resolve(info);
20     });
21 }
22 exports.getdescription = eval( getdescription );
23 exports.getinfousers = getinfousers;
```

Listagem 3.4: Estrutura do módulo para método personalizado

Na variável *info* tem de ser colocada a descrição do método implementado de forma a estar disponível na consulta da documentação. No desenvolvimento de métodos personalizados e com o intuito de testar a funcionalidade, foi desenvolvido o método *getinfousers*, que permite a agregação de dados da base de dados do SIDE com os dados da base de dados do SIGACAD. Os dados são agregados utilizando a chave primária *userID*. Desta forma obtém-se dados de forma mais completa. Na secção seguinte serão apresentados os métodos que constituem a API REST desenvolvida no âmbito desta dissertação.

3.2 Descrição dos métodos da API REST

Foram desenvolvidos vários métodos que se enquadram em várias categorias, consoante o objetivo de cada método. As categorias de classificação dos métodos são:

- **Autenticação** – Métodos necessários ao processo de autenticação.
- **Esquema** – Métodos para obtenção de informação de estrutura da base de dados ou de uma tabela específica.

- **Tabela** – Métodos para obtenção de elementos (operação CRUD), permitindo filtragem / paginação de dados.
- **Inserção** – Métodos para inserção de elementos (operação CRUD).
- **Atualização** – Métodos para atualização de elementos (operação CRUD).
- **Remoção** – Métodos para remoção de elementos (operação CRUD).
- **Personalização** – Métodos personalizados.
- **Documentação** – Métodos para consulta documentação da API.

Os vários métodos serão descritos nas subsecções seguintes.

3.2.1 Autenticação (_authentication)

Método 1 - _authentication/listmethods	
Content-type	facultativo
Body	facultativo
Método HTTP	GET
Descrição	Método retorna os métodos de autenticação disponíveis

Método 2 - _authentication/listusermethods	
Content-type	application/json
Body	{"username":"johndoe"}
Método HTTP	POST
Descrição	Método retorna a listagem de métodos de autenticação disponíveis para um determinado utilizador / aplicação

Método 3 - <code>_authentication/login/{METHOD}</code>	
Content-type	application/json
Body	<code>{"username":"johndoe", "password":"*****"}</code>
Método HTTP	POST
Descrição	Método permite a utilização de um determinado método (<code>{METHOD}</code>) de autenticação

Método 4 - <code>_authentication/logout</code>	
Content-type	application/json
Body	facultativo
Método HTTP	POST
Descrição	Método permite terminar sessão de autenticação

3.2.2 Esquema (`_schema`)

Método 5 - <code>_schema/showtables</code>	
Content-type	facultativo
Body	facultativo
Método HTTP	GET
Descrição	Método devolve a lista das tabelas da base de dados do SIDE

Método 6 - <code>_schema/showtables/{TABLENAME}</code>	
Content-type	facultativo
Body	facultativo
Método HTTP	GET
Descrição	Método devolve a estrutura da tabela <code>{TABLENAME}</code> da base de dados do SIDE

3.2.3 Tabela (_table)

Método 7 - _table/{TABLENAME}/count	
Content-type	facultativo
Body	facultativo
Método HTTP	GET
Descrição	Método devolve a contagem de elementos da tabela {TABLENAME} da base de dados do SIDE

Método 8 - _table/{TABLENAME}	
Content-type	application/json
Body	{"columns": "*"OU "columns": "a,b,c", "condition": "a>10", "orderby": "a,b,c", OU "orderby": "a ASC,b DESC,c ASC"OU "groupby": "a,b,c"}
Método HTTP	POST
Descrição	Método retorna elementos da tabela {TABLENAME} sem paginação. Possibilidade de seleção de colunas (<i>columns</i>), filtragem condicional (<i>condition</i>), ordenação (<i>orderby</i>), agrupamento (<i>groupby</i>)

Método 9 - <code>_table/{TABLENAME}/page/{PAGE}/limit/{LIMIT}</code>	
Content-type	application/json
Body	<code>{"columns": "*"OU "columns": "a,b,c", "condition": "a>10", "orderby": "a,b,c", OU "orderby": "a ASC,b DESC,c ASC"OU "groupby": "a,b,c"}</code>
Método HTTP	POST
Descrição	Método retorna elementos da tabela <code>{TABLENAME}</code> com paginação. Possibilidade de seleção de colunas (<i>columns</i>), filtragem condicional (<i>condition</i>), ordenação (<i>orderby</i>), agrupamento (<i>groupby</i>)

3.2.4 Inserção (`_insert`)

Método 10 - <code>_insert/{TABLENAME}</code>	
Content-type	application/json
Body	<code>{"userID":"johndoe"}</code>
Método HTTP	POST
Descrição	Método permite inserir elementos na tabela <code>{TABLENAME}</code>

3.2.5 Atualização (`_update`)

Método 11 - <code>_update/{TABLENAME}</code>	
Content-type	application/json
Body	<code>{"userID":"johndoe1", "condition":"userID='johndoe2'"}</code>
Método HTTP	PUT
Descrição	Método permite atualizar elementos na tabela <code>{TABLENAME}</code> . Possibilidade de filtragem condicional (<i>condition</i>)

3.2.6 Remoção (_delete)

Método 12 - _delete/{TABLENAME}	
Content-type	application/json
Body	{"condition":"userID='johndoe'"}
Método HTTP	DELETE
Descrição	Método permite remover elementos da tabela {TABLENAME}. Possibilidade de filtragem condicional (<i>condition</i>)

3.2.7 Personalização (_custom)

Método 13 - _custom/list	
Content-type	facultativo
Body	facultativo
Método HTTP	GET
Descrição	Método retorna listagem dos métodos personalizados

Método 14 - _custom/showfuncorproc/{PROCFUNCNAME}	
Content-type	facultativo
Body	facultativo
Método HTTP	GET
Descrição	Método devolve descrição do método personalizado {PROCFUNCNAME}

Método 15 - <code>_custom/call/{PROCFUNCNAME}</code>	
Content-type	application/json
Body	<code>{"page": "1", "limit": "3000", "condition": "users.userID='alXXXXX'"}</code>
Método HTTP	POST
Descrição	Método executa método personalizado <code>{PROCFUNCNAME}</code> . Possibilidade de filtragem condicional (<i>condition</i>), paginação (<i>page</i> , <i>limit</i>)

3.2.8 Documentação (`_documentation`)

Método 16 - <code>_documentation</code>	
Content-type	facultativo
Body	facultativo
Método HTTP	GET
Descrição	Método devolve documentação da API REST

3.3 Documentação

A documentação da API REST foi implementada em OpenApi 3.0 e pode ser acessada através do método 16(vid. método 3.2.8):

```

1  "/_table/{table_name}/count": {
2          "get": {
3                  "summary": "Get table element count",
4                  "description": "Get table element count",
5                  "parameters": [
6                          {
7                                  "name": "table_name",
8                                  "in": "path",
9                                  "required": true,
10                                 "schema": {
11                                         "type": "string"
12                                 }
                    ]
        }
}

```

```
13         }
14     ],
15     "operationId": "",
16     "responses": {
17         "200": {
18             "description": "successful operation",
19             "content": {
20                 "application/json": {
21                     "schema": {
22                         "$ref": "#/components/schemas/
23                             Response"
24                     }
25                 }
26             }
27         },
28         "tags": [
29             "_table"
30         ]
31     }
32 },
```

Listagem 3.5: Segmento da documentação implementada em OpenApi

Os ficheiros da documentação da API REST *side-api-generator-specification.json* e *side-api-generator-specification.yml* em formatos JSON e YAML⁵ respetivamente são colocados na pasta *documentation* de forma a que seja criada uma interface *Web* automática para consulta da documentação, tal como se pode ver na figura 3.2:

⁵YAML Ain't Markup Language - formato de serialização (codificação de dados) de dados legíveis por humanos inspirado em linguagens como XML, C, Python, Perl, assim como o formato de correio eletrónico especificado pela RFC 2822.

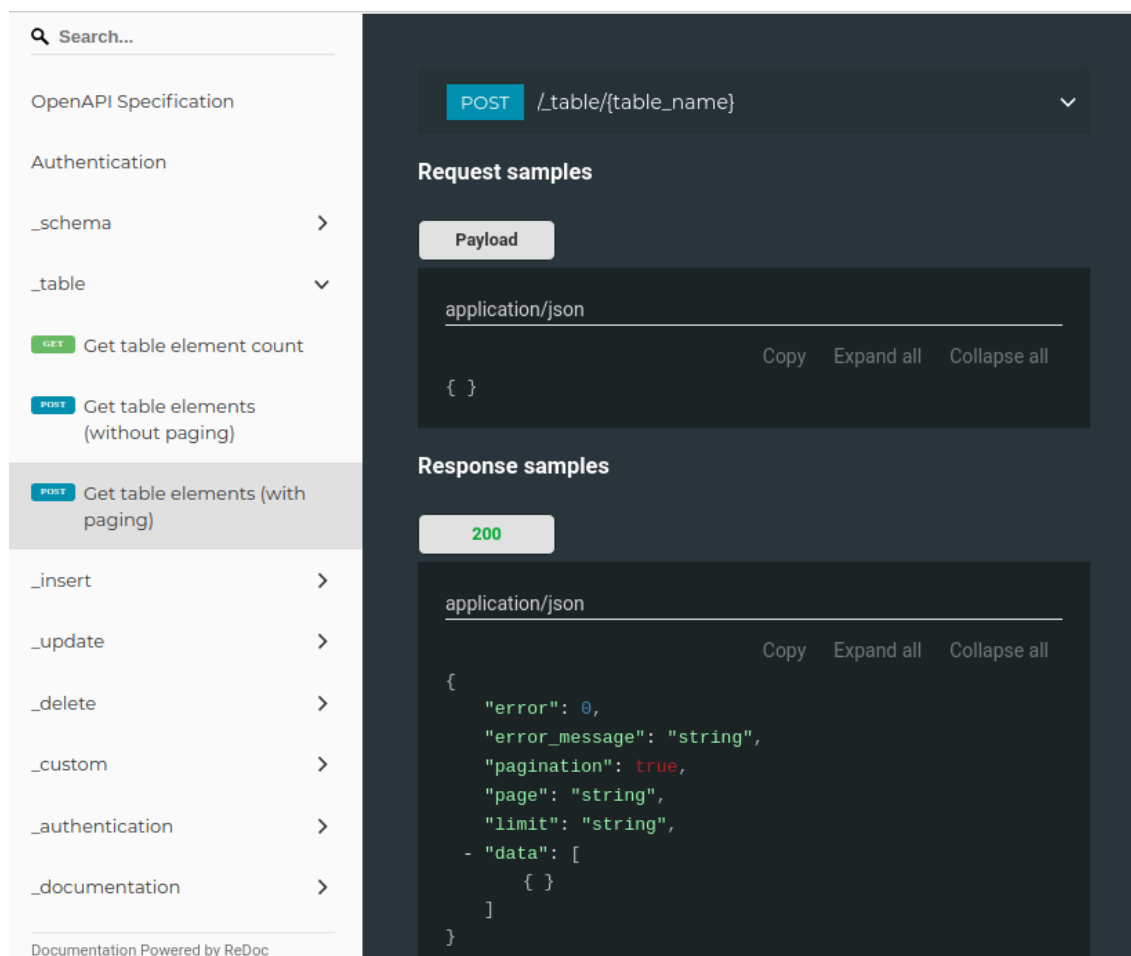


Figura 3.2 – Interface web para consulta de documentação em OpenApi

3.4 Autenticação

O módulo de autenticação utiliza o Passport. Alguns métodos de autenticação foram implementados porque são métodos que são utilizados nos diversos serviços informáticos da UTAD (vid. figura 1.1) e os restantes foram implementados devido à sua simplicidade de utilização e devido aos mecanismos de autenticação disponibilizados por algumas das redes sociais mais populares (Moreau, 2016). Assim estão disponíveis os seguintes métodos de autenticação:

- Autenticação base de dados local - Autenticação em base de dados local.
- Autenticação LDAP - Autenticação em servidor LDAP⁶.
- Autenticação *shibboleth* - Autenticação em *shibboleth*⁷.
- Autenticação HTTP Básica - Autenticação básica HTTP⁸.
- Autenticação HTTP Digest - Autenticação digest HTTP⁹.
- Autenticação OAuth Facebook - Autenticação com OAuth Facebook.
- Autenticação OAuth Google - Autenticação com OAuth Google.
- Autenticação OAuth Twitter - Autenticação com OAuth Twitter.

O utilizador / aplicação pode requerer os vários métodos de autenticação disponíveis em geral ou os métodos disponíveis para o utilizador / aplicação em concreto através dos métodos 1 e 2 (vid. métodos 3.2.1) respetivamente. Ademais dos métodos de autenticação genéricos, também podem ser definidos métodos de autenticação por utilizador / aplicação. Os métodos de autenticação são definidos numa base de dados em SQL lite *authentication_side_api*. Para definição de métodos de autenticação específicos existe uma tabela *authentication* com a seguinte estrutura:

#	cid	name	type	notnull	dflt_value	pk
0	0	authenticationid	INTEGER	1	NULL	1
1	1	type	VARCHAR	1	NULL	0
2	2	target	VARCHAR	1	"	0
3	3	method	VARCHAR	1	NULL	0
4	4	lastmodified	DATETIME	1	CURRENT_TIMESTAMP	0

Figura 3.3 – Estrutura da tabela de *authentication*

⁶ *Lightweight Directory Access Protocol*.

⁷ Sistema de SSO para redes de computadores e Internet.

⁸ HTTP Basic authentication.

⁹ HTTP Digest authentication.

A estrutura desta tabela é constituída por campos com a seguinte descrição:

- **type:** *generic* ou *user*.
- **target:** se for tipo *user* tem de estar definido.
- **method:** nome do método.
- **lastmodified:** *timestamp* da última alteração.

Para o controlo de autenticação existem dois métodos internos que se revestem de crucial importância: *ensureNotAuthenticated*, *ensureAuthenticated*. O método interno *ensureNotAuthenticated* é chamado sempre que existe o pedido a um método da API que não necessite que o cliente esteja autenticado. Por outro lado o método interno *ensureAuthenticated* é chamado na utilização de métodos da API que necessitem de clientes autenticados.

3.5 Controlo de acesso

Em termos globais, a API desenvolvida utiliza um modelo RBAC (vid. modelo de controlo de acesso 2.8.2). Assim, quando autenticado o utilizador / aplicação, o controlo de acesso é feito numa primeira fase através da verificação de uma *hash* estática, em que o utilizador / aplicação é enquadrado ou não num dos papeis (*roles*) existentes na *hash* estática:

```
1 {
2   "utilizador": {
3     "description": "Utilizadores do SIDE",
4     "parameters": {},
5     "list": {
6       "select": "sql:SELECT userID FROM users WHERE userID=?",
7       "browse": "sql:SELECT userID FROM users"
8     }
9   }
10 },
11 {
12   "docente": {
13     "description": "Docentes",
14     "parameters": {},
```

```
15     "list": {
16         "select": "sql:SELECT userID FROM users WHERE classID=\"docente\"
                    AND userID=?",
17         "browse": "sql:SELECT userID FROM users WHERE classID=\"docente\" "
18     }
19 }
20 },
```

Listagem 3.6: Segmento da *hash* estática de controlo de acesso

Alternativamente, como mecanismo complementar também podem ser definidos papéis e perfis de acesso através de uma base de dados SQL lite *permission_side_api*. Na base de dados existem duas tabelas: *role*, *permission*. Através da tabela *role* é possível definir novos papéis mediante determinadas condições (campos *condition* e *params*). A tabela *permission* permite estabelecer permissões dinâmicas para um utilizador / aplicação concreto (campo *type*) ou para um determinado *role* em que se define as operações (campo *operation*) concretas ou todas (*), especifica-se as tabelas (campo *table*) em que se está a definir as permissões segundo determinadas condições (campos *condition* e *params*).

A tabela *role*:

#	cid	name	type	notnull	dflt_value	pk
0	0	roleid	INTEGER	1	NULL	1
1	1	role	VARCHAR	1	NULL	0
2	2	condition	TEXT	1	1	0
3	3	params	TEXT	1	"	0
4	4	lastmodified	DATETIME	1	CURRENT_TIMESTAMP	0

Figura 3.4 – Estrutura da tabela de *role*

A estrutura da tabela *role* é constituída por campos com a seguinte descrição:

- **role**: nome do *role* (papel).
- **condition**: condição que tem que satisfazer para pertencer ao *role*.
- **params**: parâmetros que validam a condição *condition*.
- **lastmodified**: *timestamp* da última alteração.

E a tabela *permission*:

#	cld	name	type	notnull	dflt_value	pk
0	0	permissionid	INTEGER	1	NULL	1
1	1	type	VARCHAR	1	NULL	0
2	2	target	VARCHAR	1	NULL	0
3	3	operation	VARCHAR	1	NULL	0
4	4	table	VARCHAR	1	NULL	0
5	5	condition	TEXT	1	1	0
6	6	params	TEXT	1	"	0
7	7	lastmodified	DATETIME	1	CURRENT_TIMESTAMP	0

Figura 3.5 – Estrutura da tabela de *permission*

Os campos da estrutura da tabela *permission* possuem a seguinte descrição:

- **type:** *user* ou *role*.
- **target:** definição de utilizador / papel.
- **operation:** lista de operações ou todas (*).
- **table:** lista de tabelas ou todas (*).
- **condition:** condição que tem que satisfazer para conceder a permissão.
- **params:** parâmetros que validam a condição *condition*.
- **lastmodified:** *timestamp* da última alteração.

No controlo de acesso existem dois métodos internos que desempenham importante papel: *accesscontrolUserRolesInternal*, *accesscontrolUserRolesDynamic*.

Quando existe uma autenticação com sucesso são verificados perfis internos (*accesscontrolUserRolesInternal*). Nos outros métodos da API REST são verificados os perfis dinâmicos, as permissões gerais, as permissões dos perfis dinâmicos e as permissões específicas do cliente. O método interno *accesscontrolUserRolesInternal* preenche um vetor (*roles*) com todos os perfis a que pertence o cliente e o vetor é guardado numa variável de sessão *req.user.roles* (*cache*). A sessão criada é guardada numa tabela da base de dados com a seguinte estrutura:

```

1 CREATE TABLE 'sessionsexpress' (
2   'session_id' VARCHAR(128) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin
      NOT NULL,
3   'expires' INT(11) UNSIGNED NOT NULL,
4   'data' TEXT CHARACTER SET utf8mb4 COLLATE utf8mb4_bin,
5   PRIMARY KEY ('session_id')
6 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Listagem 3.7: Estrutura da tabela de sessão

No exemplo da sessão podemos verificar o vetor *roles* criado no processo de autenticação:

Sessão criada no processo de autenticação	
session_id	9p4dXmKXwAxApkProGmkwaqVJ2B-t_w
expires	1539019432
data	{ "originalMaxAge": null, "expires": null, "httpOnly": true, "path": "/" } , "passport": { "user": { "username": "johndoe", "roles": ["utilizador", "docente", "administrador", "docentecadeira", "docenteturma"] } }

A API REST possui mecanismos para determinar / especificar perfis dinâmicos em *runtime*, o que implica a diminuição do tempo de inatividade (*downtime*) do serviço, i.e., se for necessário acrescentar um novo perfil de utilização ou alterar um existente, apenas é necessário inserir ou alterar um perfil na base de dados e não será necessário recorrer a quebras de serviço.

3.6 Tratamento de pedidos / filtragem de dados

A principal fonte de dados da API REST é a base de dados do SIDE. Tendo sido um dos objetivos na elaboração do trabalho desta dissertação existe a possibilidade de agregar dados de outras fontes, como é demonstrado no método *getinfousers*, em que se agregam dados de uma base de dados MySQL (SIDE) e dados de uma base de dados SQL Server (SIGACAD). Seguidamente temos o exemplo de dados obtidos num consulta normal sem agregação:

```
1 {  
2   "error" : 0,  
3   "error_message" : "",  
4   "pagination" : false,  
5   "page" : "",  
6   "limit" : "",  
7   "data" : [ {  
8     "userID" : "alXXXXX",  
9     "classID" : "aluno",  
10    "password" : null,  
11    "email" : "alXXXXX@utad.eu",  
12    "nome" : "John Doe",  
13    "desactivado" : 0,  
14    "lastmodified" : "2015-11-30T14:30:43.000Z",  
15    "lastmodifiedby" : "alXXXXX",  
16    "serial" : 1,  
17    "email_state" : 2,  
18    "liveedu" : 2  
19  } ]  
20 }
```

Listagem 3.8: Dados obtidos sem agregação

Em baixo, exemplo de dados obtidos com agregação:

```
1 {
2   "error" : 0,
3   "error_message" : "",
4   "pagination" : false,
5   "page" : "",
6   "limit" : "",
7   "data" : [ {
8     "NUMERO" : XXXXX,
9     "NOME" : "JOHN DOE",
10    "NOMEPROPRIO" : "JOHN",
11    "APELLIDO" : "DOE",
12    "SEXO" : "M",
13    "DATANASC" : "1977-04-25T00:00:00.000Z",
14    "PAI" : "JOHN DOE Sr",
15    "MAE" : "MARY DOE",
16    "NACIONALIDADE" : "PRT",
17    "TIPOALUNO" : "1",
18    "PASSWORD" : null,
19    "EMAIL" : "johndoe@somecorp.pt",
20    "OBERV" : "",
21    "REGIME" : "O",
22    "NIB" : "",
23    "COMOUTGOING" : null,
24    "ComValidadeNotificacaoEmail" : true,
25    "userID" : "alXXXXX",
26    "classID" : "aluno",
27    "password" : null,
28    "email" : "alXXXX@utad.eu",
29    "nome" : "John Doe",
30    "desactivado" : 0,
31    "lastmodified" : "2012-01-31T16:21:42.000Z",
32    "lastmodifiedby" : "admin",
33    "serial" : 1,
34    "email_state" : 2,
35    "liveedu" : 2,
36    "finalizado" : null,
37    "numero_aluno" : "XXXXX"
38  } ]
39 }
```

Listagem 3.9: Dados obtidos com agregação

Pode verificar-se na listagem 3.9 que foram agregados vários campos como: NOMEPRPRIO, APELIDO, SEXO, DATANASC, etc. Também nas ligações às bases de dados e de forma a aumentar a disponibilidade de dados foram criadas *pools* de conexões¹⁰ de dados.

3.7 Auditoria

Para o desenvolvimento do módulo de auditoria (*logs*) foi utilizado o módulo *bunyan*. A auditoria pode ser ativada / desativada e o seu nível pode ser configurado:

- **60 - FATAL** - Situação crítica que deve ser revista por um operador.
- **50 - ERROR** - Situação grave que deve ser revista por um operador quando possível.
- **40 - WARN** - Situação deve ser revista eventualmente por um operador.
- **30 - INFO** - Detalhes de operações normais.
- **20 - DEBUG** - Aumento da verbosidade dos detalhes das operações normais.
- **10 - TRACE** - Auditoria detalhada de qualquer operação incluído chamadas a módulos externos à própria API.

Exemplo de *logs* com nível 10 (TRACE) em formato JSON:

```
1 {  
2   "name": "SIDEAPI",  
3   "hostname": "debian",  
4   "pid": 785,  
5   "level": 30,  
6   "response": {  
7     "status_code": 200,  
8     "timestamp": "2018-10-08T11:54:05.193Z",  
9     "timestamp_ms": 1538999645193,  
10    "elapsed": 146,
```

¹⁰ *Cache* de conexões de bases de dados mantido de forma a que as conexões possam ser reutilizadas quando requisições futuras à base de dados forem requeridas.

```

11      "body": "{ \"error\":0, \"error_message\": \"\", \"pagination\": false
12      , \"page\": \"\", \"limit\": \"\", \"data\": [ \"LOCAL_DB\", \"
13      HTTP_BASIC\", \"HTTP_DIGEST\", \"JWT_AUTH\" ] } "
14  },
15  "request": {
16      "method": "GET",
17      "url": "/side_api/_authentication/listmethods",
18      "query": {
19      },
20      "headers": {
21          "host": "192.168.xxx.xxx:8080",
22          "connection": "keep-alive",
23          "upgrade-insecure-requests": "1",
24          "user-agent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit
25          /537.36 (KHTML, like Gecko) snap Chromium/69.0.3497.100
26          Chrome/69.0.3497.100 Safari/537.36",
27          "accept": "text/html,application/xhtml+xml,application/xml;q
28          =0.9,image/webp,image/apng,*/*;q=0.8",
29          "accept-encoding": "gzip, deflate",
30          "accept-language": "pt-BR,pt;q=0.9,en-US;q=0.8,en;q=0.7"
31      },
32      "timestamp": "2018-10-08T11:54:05.047Z",
33      "timestamp_ms": 1538999645047,
34      "body": "{}"
35  },
36  "msg": "",
37  "time": "2018-10-08T11:54:05.224Z",
38  "v": 0
39  }

```

Listagem 3.10: Auditoria a método da API REST em nível 10 (TRACE)

O sistema de auditoria quando ativado está configurado para efetuar a rotação de *logs* diariamente e manter *backups* da última semana. Estes parâmetros podem ser eventualmente alterados para melhor servir as necessidades. O módulo de auditoria também possui mecanismos de eliminação (nos ficheiros de *logs*) de palavras-passe ou qualquer tipo de informação mais sensível.

3.8 Segurança

Os parâmetros dos métodos implementados são analisados de forma a que sejam mitigadas eventuais situações que possam tornar instável o funcionamento da API REST, desde limitação de alguns parâmetros, como análise de alguns parâmetros que possam de alguma forma explorar a obtenção não privilegiada de dados. Também de forma a não colocar a API REST sob elevada carga, são implementadas respostas simples e com o tamanho mais pequeno possível mantendo sempre a resposta padronizada (estrutura *return_data*). Assim, quando existe a tentativa de acesso ao um método da API REST privado por parte de um cliente não autenticado, a resposta será do tipo:

```
1 {  
2   "error" : 1,  
3   "error_message" : "ACCESS DENIED!",  
4   "pagination" : false,  
5   "page" : "",  
6   "limit" : "",  
7   "data" : ""  
8 }
```

Listagem 3.11: Exemplo de *return_data*

3.9 Conclusão

Em conclusão, o desenvolvimento da API REST trouxe alguns desafios, nomeadamente o fato de a linguagem de programação Node.js ser assíncrona, o que fez com o que o planeamento de algumas operações tivesse de ser mais cuidadoso (i.e., encadeamento de operações assíncronas), com efeito, as operações assíncronas (operações I/O em ficheiros, operações de acesso a base de dados, etc.) tinham de registar um *callback*, com várias hipóteses de retorno e trabalhar com objetos devolvidos do tipo *promise* que representam a eventual conclusão (ou falha) de uma operação assíncrona.

Portanto, foram definidos os componentes basilares para a construção de uma API

REST, mais completa e capaz de resolver os problemas de interoperabilidade do SIDE e de agregação de dados provenientes de outros sistemas de informação da UTAD. Assim, através da utilização da API, o desenvolvimento de aplicações fica facilitado e o programador não precisa de ter conhecimento de baixo nível da camada de dados, apenas necessita utilizar a API REST disponibilizada.

4

Casos de utilização

4.1 Introdução

Neste capítulo, foi utilizado o Angular Material¹ para desenvolvimento de um caso de utilização. A aplicação desenvolvida possui um formulário de autenticação, um formulário de listagem de elementos e um formulário de atualização de elementos. Através da utilização de vários métodos da API desenvolvida, implementou-se uma aplicação, exemplificando a simplicidade e flexibilidade de utilização da API REST construída no âmbito desta dissertação.

4.2 Aplicação em Angular Material

A aplicação desenvolvida em *Angular Material* possui três partes principais: autenticação, listagem de elementos e atualização de elementos (utilizadores). Assim os componentes principais da aplicação possuem diferentes roteamentos:

¹ *Framework* de componentes de interface de utilizador e implementação de referência da especificação do Material Design da Google.

- Por defeito, não autenticado (*path*: “/”);
- Listagem de elementos, autenticado (*path*: “/data”);
- Atualização de elementos, autenticado (*path*: “/update/:id”);

Para o desenvolvimento da aplicação, foi instalado o servidor virtual N^o 4 (vid. tabela 3.1), no qual foi instalado todo o ambiente de desenvolvimento da aplicação. O ambiente de desenvolvimento inclui o *Node.js* e o *npm*²:

```
1 user@host:~$ sudo apt-get update
2 user@host:~$ sudo apt-get install nodejs
3 user@host:~$ sudo apt-get install npm
```

Listagem 4.1: Instalação do Node.js e npm

Seguidamente é instalado o Angular CLI³:

```
1 user@host:~$ npm install -g @angular/cli
```

Listagem 4.2: Instalação do Angular CLI

Através da Angular CLI é possível criar / executar aplicações, componentes, roteamentos, serviços e *pipes* com um simples comando. Assim foi criada uma aplicação *sideapp*, na qual foi instalado o Angular Material:

```
1 user@host:~$ ng new sideapp
2 user@host:~$ cd sideapp
3 user@host:~/sideapp$ npm install --save @angular/material @angular/
  cdk @angular/animations
```

Listagem 4.3: Criação da aplicação *sideapp* e instalação do Angular Material

Para a implementação deste caso de utilização e recorrendo à Angular CLI foram criados dois serviços de dados, um para autenticação (*login*) e outro para a obtenção / atualização dos dados (*data*):

```
1 user@host:~/sideapp$ ng generate service login
2 user@host:~/sideapp$ ng generate service data
```

Listagem 4.4: Criação dos serviços de dados através da Angular CLI

²Gestor de pacotes para a linguagem de programação JavaScript.

³Angular Command Line Interface.

4.2.1 Autenticação da aplicação

Na figura 4.1 está o formulário de autenticação:

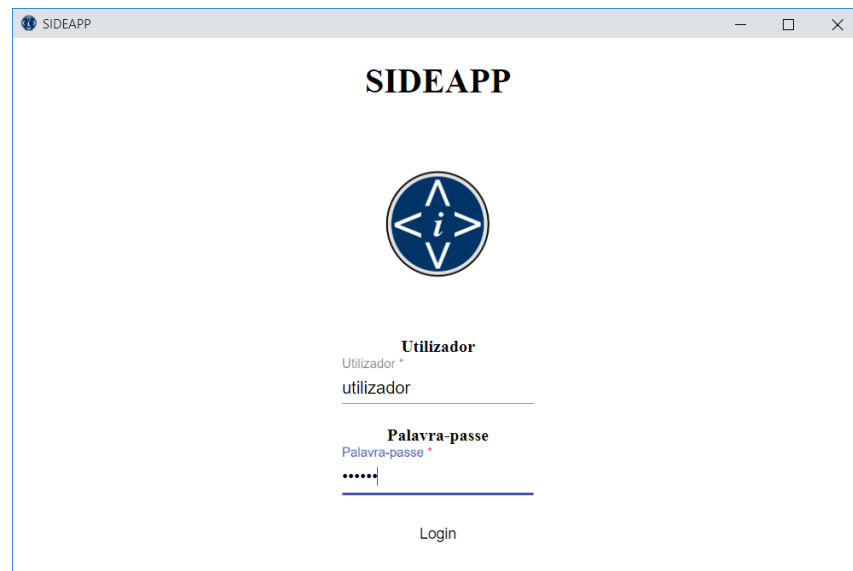
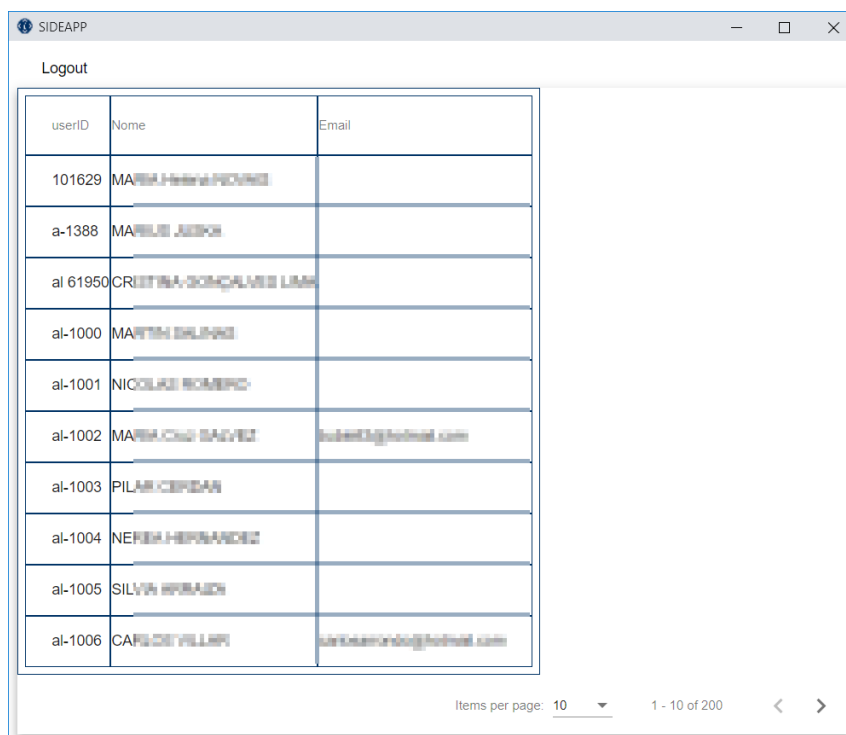
The image shows a web browser window titled "SIDEAPP". The page has a white background with the title "SIDEAPP" at the top center. Below the title is a circular logo with a blue border and a white center containing a stylized 'i' with four arrows pointing outwards. Under the logo, there are two input fields. The first is labeled "Utilizador" in bold, with a small asterisk and the text "Utilizador *" above it. The input field contains the text "utilizador". The second is labeled "Palavra-passe" in bold, with a small asterisk and the text "Palavra-passe *" above it. The input field contains seven dots. Below the input fields is a "Login" button.

Figura 4.1 – Formulário de autenticação da aplicação *sideapp*

Na aplicação foi criado um serviço de dados ligado ao método de autenticação definido para a aplicação *sideapp*. Foram criadas credenciais de acesso para a aplicação *sideapp* através do método de autenticação LOCAL_DB. Numa primeira fase é invocado o método *listusermethods* (vid. método 3.2.1) e depois de devolvidos os métodos de autenticação disponíveis para o utilizador, é utilizado o método *login* (vid. método 3.2.1) com os respetivos parâmetros (*username* e *password*) que são obtidos através do formulário de autenticação da figura 4.1.

4.2.2 Listagem de elementos

Na figura seguinte está a listagem de elementos na aplicação *sideapp*:



userID	Nome	Email
101629	MARIA HELENA FONSECA	
a-1388	MARCELO JESUS	
al-61950	CRISTINA SOUZA LIMA	
al-1000	MARTIN SALAS	
al-1001	NICOLAZI ROMERO	
al-1002	MARIA CLOTE SAUZES	luisa@sideapp.com
al-1003	PILAR CORDAN	
al-1004	NEREA HERNANDEZ	
al-1005	SILVIA HERNANDEZ	
al-1006	CARLOS VILLER	luisa@sideapp.com

Items per page: 10 1 - 10 of 200

Figura 4.2 – Listagem de elementos na aplicação *sideapp*

Depois de autenticado, o utilizador é direcionado para uma *view* onde aparece uma listagem de elementos (utilizadores) (vid. figura 4.2). Neste caso é criado um serviço de dados ligado ao método 9 (vid. método 3.2.3). Através do controlo de acesso, foi atribuído ao perfil da aplicação *sideapp* a permissão de leitura / atualização de dados da tabela *users*, restringindo o acesso a qualquer outra tabela da base de dados do SIDE. Inicialmente é obtida a contagem dos elementos do tipo pretendido através do método 7 (vid. método 3.2.3) da API REST. Seguidamente, através do método 9 (vid. método 3.2.3) e como a tabela de *users* é uma tabela genérica, torna-se necessário filtrar os dados através de determinadas parametrizações:

```

1 getDataElements() {
2     let httpHeaders = new HttpHeaders({
3         'Content-Type' : 'application/json'
4     });
5     const body = {
6         columns: "*",

```



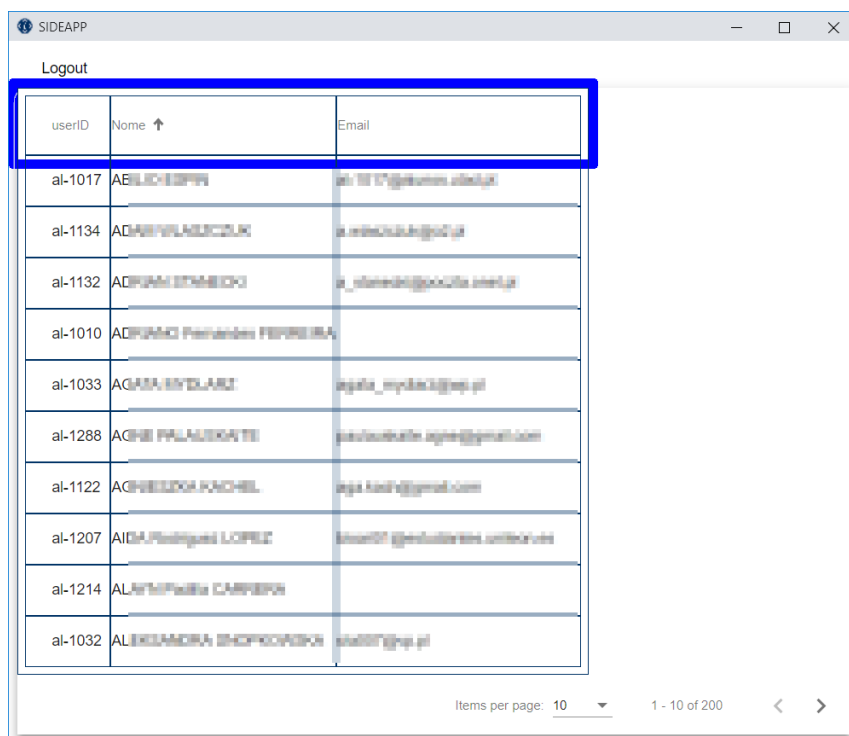
```
7      condition: "classID='aluno' "  
8    };  
9    return this.http.post("http://server:8080/side_api/_table/users/  
      page/1/limit/200", JSON.stringify(body), {  
10      headers: httpHeaders  
11    });  
12  }
```

Listagem 4.5: Serviço de dados para listagem de elementos

Neste caso é determinado que o tipo de elementos é do tipo aluno (`classID='aluno'`) e são requeridas todas as colunas de dados (`columns:'*`). Também é criada uma ligação entre as colunas da listagem de elementos (`userID`, `nome`, `Email`) da *view* e os dados provenientes do serviço de dados.

4.2.3 Ordenação de elementos

Na figura seguinte é exemplificada a ordenação de elementos na aplicação *sideapp*:



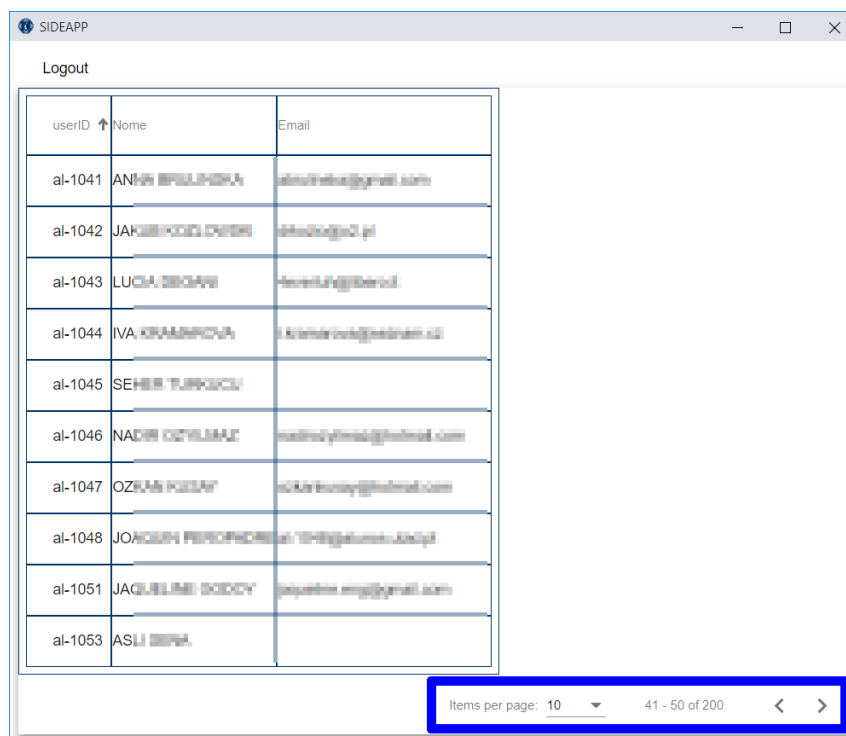
userID	Nome ↑	Email
al-1017	ABELIO ESPIN	al_1017@sideapp.sideapp
al-1134	ADRIANA PALMEIRA	a.palmeira@sideapp
al-1132	ADRIANA STRECH	a.strech@sideapp.sideapp
al-1010	ADRIANO FERNANDES FERREIRA	
al-1033	AGATA MYDLANSKI	agata_mydlanski@sideapp
al-1288	AGNE PALAUSKYTE	agne.palauskyte@sideapp
al-1122	AGNE KADZINSKY	agne.kadzinsky@sideapp
al-1207	AIDA PABLO LOPEZ	aida.pablo@sideapp.sideapp
al-1214	ALAN PABLO CARREIRA	
al-1032	ALEXANDRA BUCHHEIT	alexandra.buchheit@sideapp

Figura 4.3 – Ordenação de elementos na aplicação *sideapp*

A ordenação de elementos (vid. figura 4.3) é implementada de forma automática pela aplicação.

4.2.4 Paginação de elementos

Paginação de elementos na aplicação *sideapp*:



The screenshot shows a web application window titled "SIDEAPP". Inside, there is a "Logout" button and a table of users. The table has three columns: "userID" (with an upward arrow icon), "Nome", and "Email". The table contains 10 rows of user data. Below the table, there is a pagination control bar. It includes a dropdown menu for "Items per page" set to "10", a text display "41 - 50 of 200", and navigation arrows for previous and next pages. The pagination bar is highlighted with a blue rectangle.

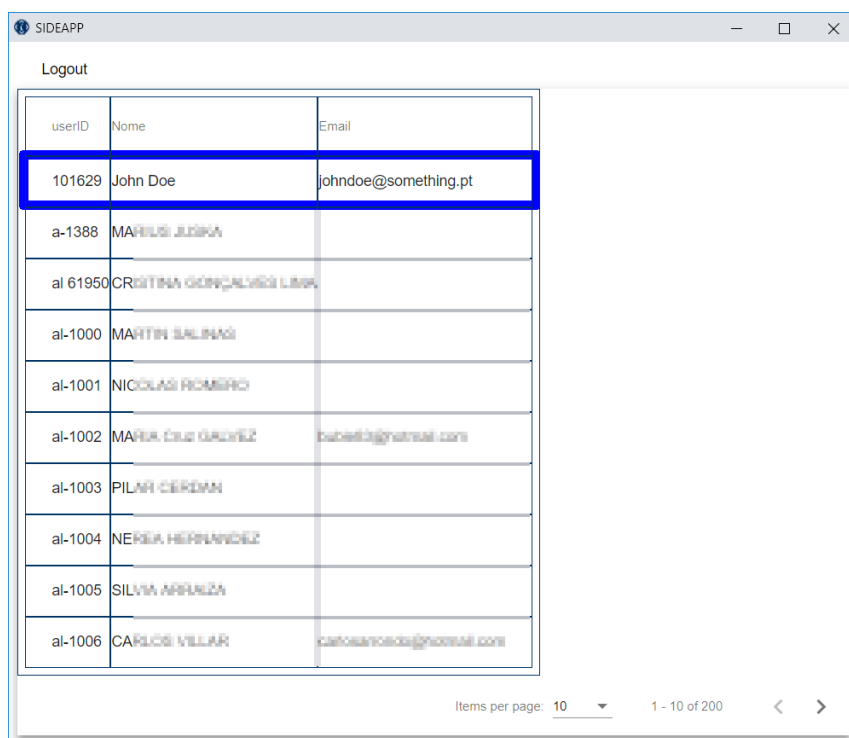
userID ↑	Nome	Email
al-1041	ANNA BRUNDA	anna.brunda@gmail.com
al-1042	JAKUB KOZLOVSKI	jakubkozi@p.pl
al-1043	LUCIA DEGAZI	lucia.dga@tercel.it
al-1044	IVA KRAMKOVA	ikramkova@seznam.cz
al-1045	SEHİR TURKOGLU	
al-1046	NADIR OZULMAZ	nadir.ozulmaz@hotmail.com
al-1047	OZKAN KESER	ozkan.keser@hotmail.com
al-1048	JONATH PERCHONEN	jperchon@tampere.fi
al-1051	JAGIELMI BODOY	jagielmi.bodo@gmail.com
al-1053	ASLI DEPA	

Figura 4.4 – Paginação de elementos na aplicação *sideapp*

A paginação de elementos é implementada passando os parâmetros para o método 9 (vid. método 3.2.3), assim a escolha da página e o número de elementos por página estão ligados aos parâmetros *page* e *limit* do método 9 (vid. método 3.2.3) da API REST.

4.2.5 Atualização de elementos

A listagem de elementos da figura 4.5 permite a seleção de elementos individuais para atualização.



userID	Nome	Email
101629	John Doe	johndoe@something.pt
a-1388	MAURÍCIO JÚDEKA	
al-61950	CRISTINA GONÇALVES LIMA	
al-1000	MARTIN SALINAGI	
al-1001	NICOLAS ROMERO	
al-1002	MAURICIO GARCIAZ	gabriel@netmail.com
al-1003	PILAR CÉRIANI	
al-1004	NEREA HERNANDEZ	
al-1005	SILVIA APPELZA	
al-1006	CARLOS VILLAR	carlosanaco@netmail.com

Items per page: 10 1 - 10 of 200 < >

Figura 4.5 – Seleção de elemento na *sideapp*

Depois de selecionado o elemento, o método *getDataElement* (vid. listagem 4.6) é invocado para a obtenção de dados do elemento e preenchimento do formulário de atualização da figura 4.6.

```
1 getDataElement(userID: string) {  
2   let httpHeaders = new HttpHeaders({  
3     'Content-Type' : 'application/json'  
4   });  
5   const body = {  
6     columns: "*",  
7     condition: "userID='" + userID + "'"  
8   };  
9   return this.http.post("http://server:8080/side_api/_table/users",  
    JSON.stringify(body), {
```

```
10     headers: httpHeaders
11   });
12 }
```

Listagem 4.6: Serviço de dados para obtenção de dados de um elemento

Formulário para atualização de um elemento (*path*: “/update/:id”):

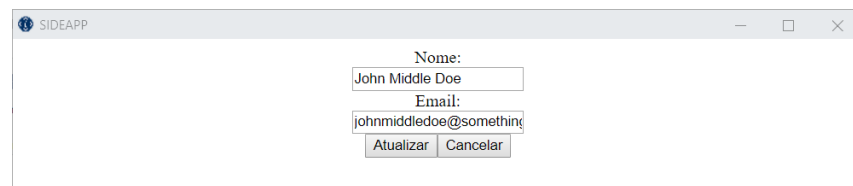
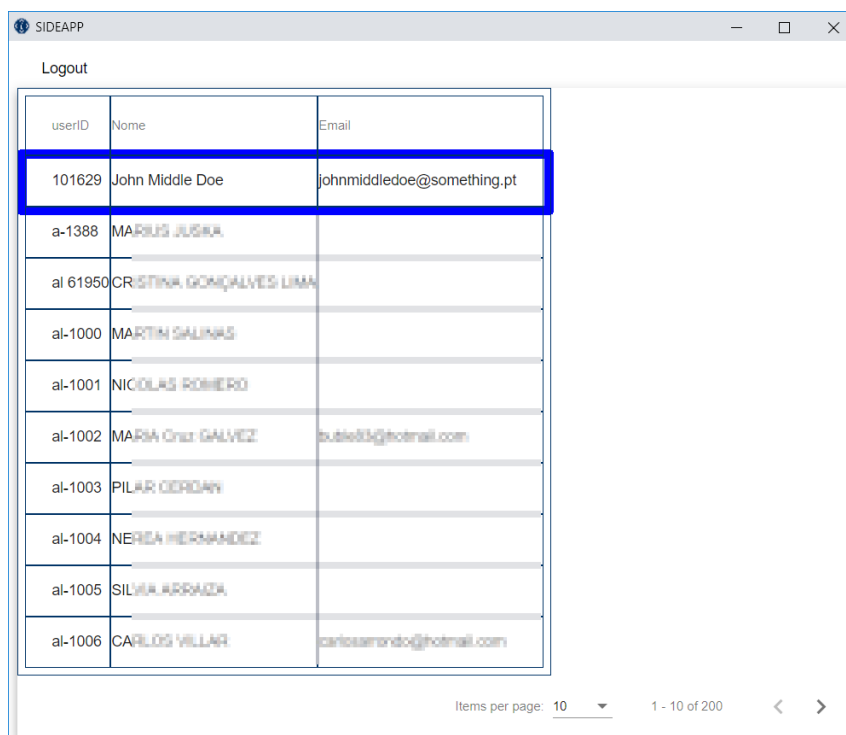


Figura 4.6 – Atualização de elemento na *sideapp*

Quando se carrega no botão “Atualizar” é chamando o método *updateData* (vid. listagem 4.7) que invoca o método da API REST para atualização do elemento. Na *condition* está a condição para atualização do elemento selecionado.

```
1 updateData(updatedElement: Element) {
2   let httpHeaders = new HttpHeaders({
3     'Content-Type' : 'application/json'
4   });
5   const body = {
6     nome: updatedElement.nome,
7     email: updatedElement.email,
8     condition: "userID='"+ updatedElement.userID + "' "
9   };
10  return this.http.put("http://server:8080/side_api/_update/users",
11    JSON.stringify(body),{
12    headers: httpHeaders
13  });
14 }
```

Listagem 4.7: Serviço de dados para atualização de dados de um elemento



The screenshot shows a web application window titled 'SIDEAPP'. Inside, there is a section labeled 'Logout' containing a table with three columns: 'userID', 'Nome', and 'Email'. The first row of the table is highlighted with a blue border. Below the table, there is a pagination control showing 'Items per page: 10' and '1 - 10 of 200'.

userID	Nome	Email
101629	John Middle Doe	johnmiddledoe@something.pt
a-1388	MARIUS JUSKA	
al-61950	CRISTINA GONÇALVES LIMA	
al-1000	MARTIN GALINAS	
al-1001	NICOLAS ROMERO	
al-1002	MARIA Cruz GALVEZ	butiele84@hotmail.com
al-1003	PILAR GORDAN	
al-1004	NEREA HERNANDEZ	
al-1005	SILVIA ARRADA	
al-1006	CARLOS VILLAR	carlosarondo@hotmail.com

Figura 4.7 – Elemento atualizado na *sideapp*

4.3 Conclusão

Em síntese, foi desenvolvida uma aplicação *Single-page application* em Angular Material, em que pela sua natureza responsiva poderá ser utilizada em dispositivos com diferentes resoluções. A *framework* Angular Material é utilizada apenas para desenvolvimento do *frontend* e apenas irá consumir dados de várias fontes. Todo o processo de autenticação, controlo de acesso e filtragem de dados (*backend*) será efetuado pela API REST.



Testes de carga

5.1 Introdução

De maneira a avaliar a API REST desenvolvida em termos de performance, optou-se pela utilização das ferramentas *open-source* mais populares (Abbas et al., 2017): Apache JMeter e Siege. Assim, foram executados vários testes à API REST, estabeleceu-se com esses testes uma comparação do tempo de resposta em milissegundos, para cenários de simulação de 1000 utilizadores por segundo em que se configurou a API para ser executada num processo, com dois processos e com quatro processos. A flexibilidade da API REST permite que se configure o sistema em modo automático de *clustering* ou modo manual (estabelecendo o número de processos a executar), para isso basta alterar as variáveis de ambiente `api_CLUSTER_MANUAL` e `api_CLUSTER_MANUAL_NUMPROCESSES`.

5.2 Configuração Apache JMeter / Siege

Na figura seguinte está o plano de testes criado no Apache JMeter:

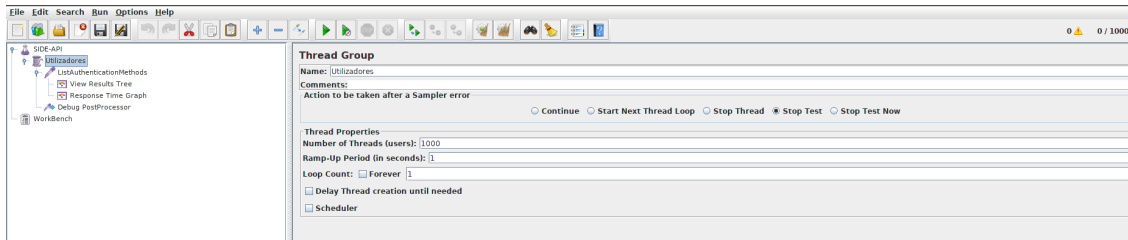


Figura 5.1 – Configuração Apache JMeter

Em termos de configuração, no Apache JMeter foi criado um plano de testes, ao qual se adicionou um *thread group* de 1000 *threads* (utilizadores) com um tempo *ramp-up*¹ de um segundo. Ao *thread group* foi adicionado um *sampler* do tipo HTTP *request* onde se configurou o pedido. Neste caso não foi necessário extrair nenhum *token* de autenticação porque o teste foi executado sobre um método público. Finalmente foram adicionados vários *listeners* ao HTTP *request*:

- *View Results tree* - Árvore das respostas de cada cenário simulado.
- *Graph results* - Gráfico para cada cenário simulado do tempo de resposta.
- *Aggregate graph* - Gráfico agregado para todos os cenários simulados do tempo de resposta.

A ferramenta Siege, em termos de preparação, não possui nenhuma configuração específica, apenas se tem de retirar o limite de 255 *threads* (utilizadores) em simultâneo e executar o teste de carga invocando o comando:

```
1 user@host:~$ siege -c1000 -t1S http://server:8080/side_api/_authentication/listmethods
```

Listagem 5.1: Comando executado para teste de carga com o Siege

¹Tempo que o JMeter deve demorar para iniciar o número total de *threads*.

5.3 Testes - Siege

5.3.1 Resultados

-	Disponibilidade	Nº Transações	Falhas(%)	Taxa de Transações
1 PROC	100%	180	0%	418.6 trans/segundo
2 PROC	100%	824	0%	958.1 trans/segundo
4 PROC	100%	1101	0%	1223.3 trans/segundo

Tabela 5.1 – Resultados de testes de carga com o Siege

5.3.2 Análise de resultados

Como se pode constatar nos resultados do teste com o Siege a disponibilidade é de 100% em qualquer um dos cenários analisado. Também se evidencia uma maior capacidade de resposta da API REST, começando com um valor de 418.6 transações por segundo, subindo para 958.1 e passando finalmente para 1223.3 transações por segundo, conforme se aumenta o número de processos em que se executa a API REST.

5.4 Testes - Apache JMeter

5.4.1 Resultados - Execução num processo

Configuração da API REST para ser executada num processo:

```
1 {  
2     "api_CLUSTER_MANUAL" : 1 ,  
3     "api_CLUSTER_MANUAL_NUMPROCESSES" : 1 ,  
4 }
```

Listagem 5.2: Configurações para sistema a ser executado num processo

Gráfico do tempo de resposta em milissegundos do sistema a ser executado num processo:

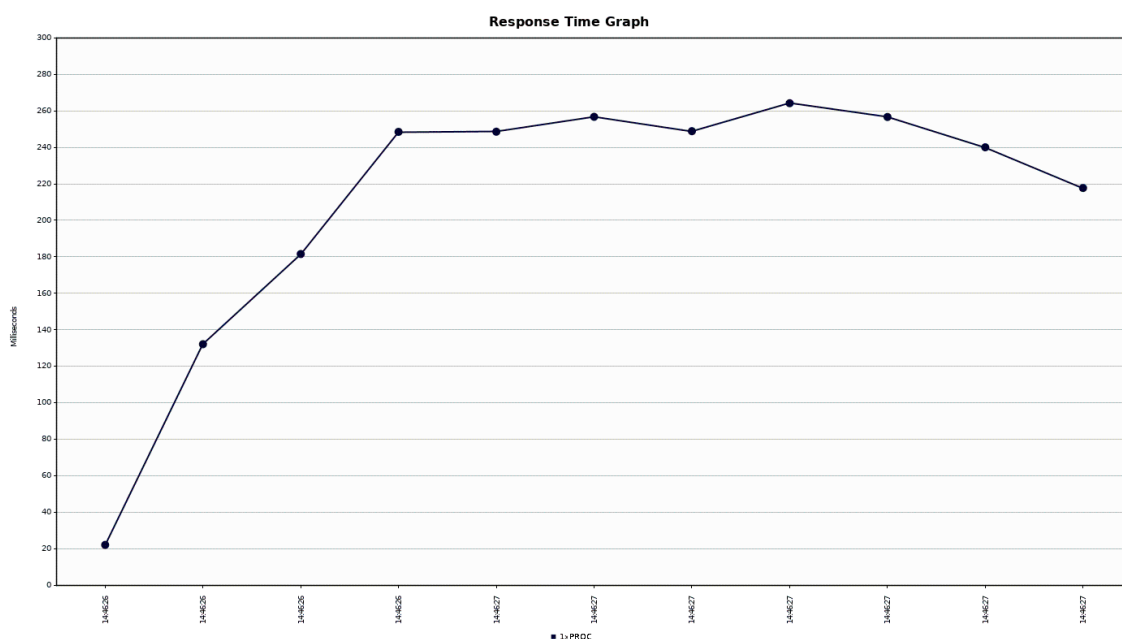


Figura 5.2 – Tempo de resposta para sistema a ser executado num processo

Cada intervalo na escala do eixo horizontal equivale a 100 milissegundos. O tempo mínimo de resposta alcançado no final de um segundo é cerca de 220 milissegundos.

5.4.2 Resultados - Execução em dois processos

Configuração da API REST para ser executado em dois processos:

```
1 {  
2     "api_CLUSTER_MANUAL": 1,  
3     "api_CLUSTER_MANUAL_NUMPROCESSES": 2,  
4 }
```

Listagem 5.3: Configurações para sistema a ser executado em dois processos

Gráfico do tempo de resposta em milissegundos do sistema a ser executado em dois processos:

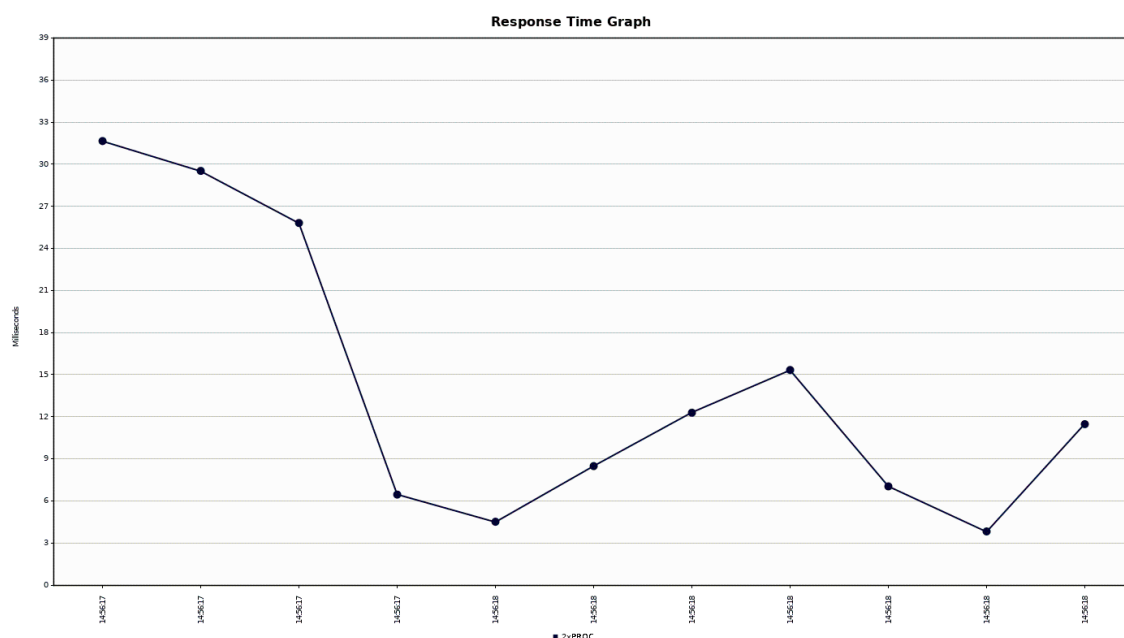


Figura 5.3 – Tempo de resposta para sistema a ser executado em dois processos

Cada intervalo na escala do eixo horizontal equivale a 100 milissegundos. O tempo mínimo de resposta alcançado no final de um segundo é cerca de 11,7 milissegundos.

5.4.3 Resultados - Execução em quatro processos

Configuração da API REST para ser executada em quatro processos:

```
1 {  
2     "api_CLUSTER_MANUAL": 1,  
3     "api_CLUSTER_MANUAL_NUMPROCESSES": 4,  
4 }
```

Listagem 5.4: Configurações para sistema a ser executado em quatro processos

Gráfico do tempo de resposta em milissegundos do sistema a ser executado em quatro processos:

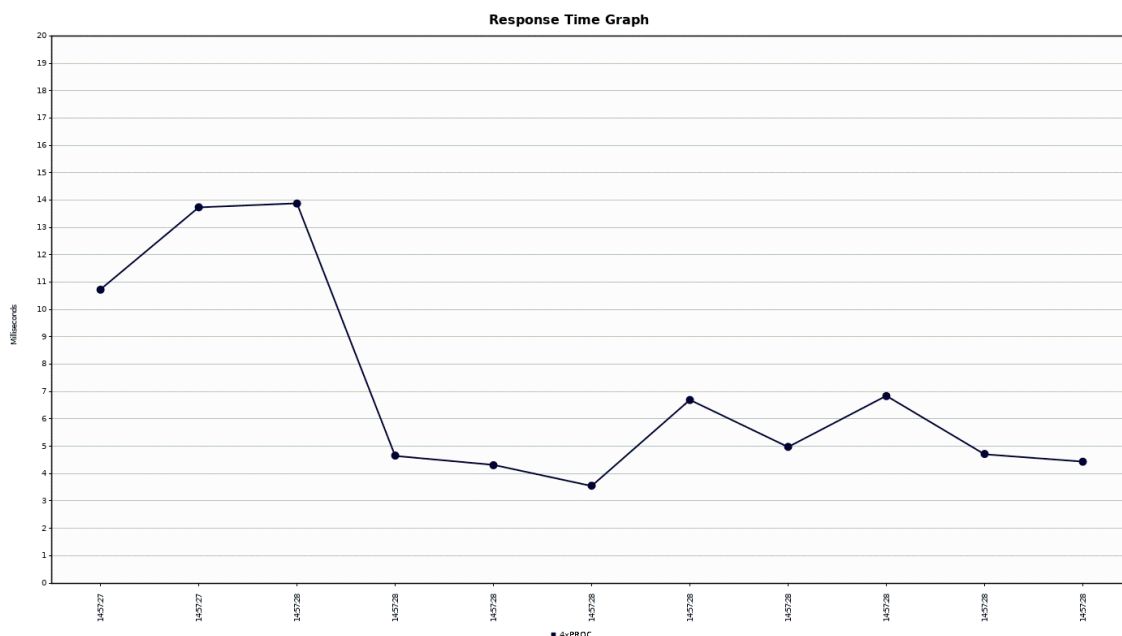


Figura 5.4 – Tempo de resposta para sistema a ser executado em quatro processos

Cada intervalo na escala do eixo horizontal equivale a 100 milissegundos. O tempo mínimo de resposta alcançado no final de um segundo é cerca de 4,5 milissegundos.

5.4.4 Análise de resultados

Gráfico dos tempos de resposta em milissegundos da agregação dos vários cenários testados, com conversão em escala logarítmica de base 10 no Microsoft Excel²:

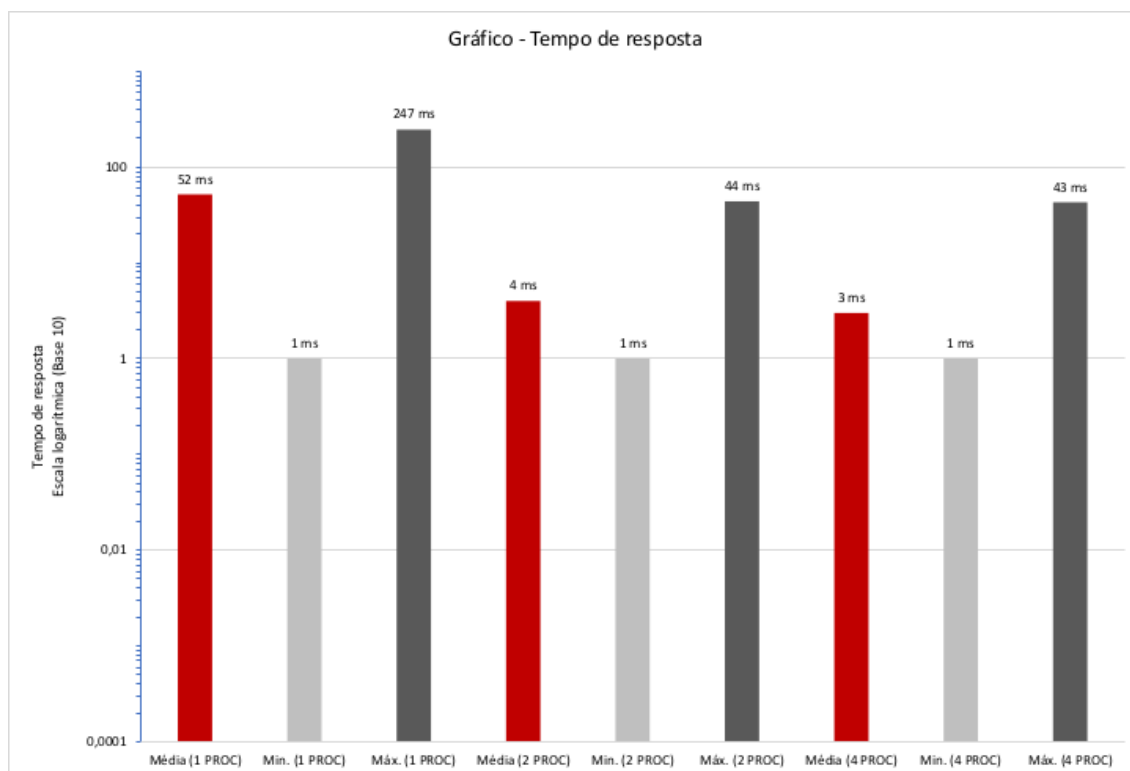


Figura 5.5 – Gráfico - Tempo de resposta

Pelo gráfico de resultados agregados e depois de executar dez testes individuais de 1000 *threads* (utilizadores), ou seja, um acumulado de 10000 *threads* (utilizadores) por cada 10 segundos é possível constatar que as médias de tempo de resposta são 52 milissegundos, 4 e 3 milissegundos por cada um dos cenários de execução (um processo, dois processos e quatro processos).

²Software de folha de cálculo desenvolvido pela Microsoft.

5.5 Conclusão

No capítulo atual, concluímos a importância dos testes no desenvolvimento da API REST, pois permitiram verificar a melhoria no tempo de resposta quando se escalava a API em termos do número de processos utilizados. Também foram executados testes num ambiente próximo do ambiente de produção, em que as características do servidor de testes se aproximavam das características do servidor em que a API será disponibilizada.

Em síntese, os resultados dos testes realizados são muito satisfatórios, obtendo assim bons tempos de resposta. Se necessário, a API possui mecanismos em que facilmente se aumenta o número de processos que servem a API, resolvendo e prevenindo eventuais problemas de congestionamento da API.



Conclusão e trabalho futuro

6.1 Conclusão

O SIDE é um sistema que já possui uma vida útil bastante longa (2001/2002), tendo começado por ser o sistema de apoio ao processo educativo no departamento de Engenharias e tendo alargado o seu funcionamento a toda a academia.

O SIDE está assente sobre algumas tecnologias que não atraem a maior parte do universo de programadores, assim era de necessária importância desenvolver interfaces que permitissem o desenvolvimento de aplicações em que se utilizassem algumas tecnologias mais recentes e populares.

Para alcançar os objetivos (gerais e específicos), teve que se efetuar investigação prévia, assim, no segundo capítulo desta dissertação, foi elaborado um estudo do estado da arte, em que primeiro se abordaram as linguagens de programação para desenvolvimento do lado do servidor (*server-side*), depois foi feita uma retrospectiva às tecnologias de desenvolvimento de serviços *web*. Seguidamente, abordaram-se as tecnologias de desenho e documentação de API. Em seguida, como a API desenvolvida é constituída por microserviços e como foram utilizadas *frameworks* de desenvolvimento (REST), sistemas de gestão de bases de dados, métodos de autenticação / autorização / auditoria, então estes temas também foram abordados.

Em termos mais específicos, foi desenvolvida uma API com um nível de abstração bastante elevado. Assim, quem a utiliza, não necessita de ter conhecimento de baixo nível da camada de acesso aos dados, podendo eventualmente alterar-se a estrutura da base de dados do SIDE ou efetuar uma migração de servidores, ligando para isso a API a uma cópia da base de dados, sendo necessário apenas alterar pequenas configurações, exigindo menores tempos de quebra de serviço.

Também se pretendeu dotar a API de extensibilidade, sendo um objetivo atingido, pois através dos métodos desenvolvidos, nomeadamente a possibilidade de personalização de métodos (vid. métodos 3.2.7), abriu-se o caminho ao desenvolvimento de novas funcionalidades. Foi desenvolvido como exemplo o método *getinfousers*, utilizando a interface do método (vid. método 3.2.7), permitindo a agregação de dados de várias fontes, neste caso específico a agregação de dados da base de dados do SIDE e da base de dados do SIGACAD, testando-se assim novas perspectivas em termos de extensão e agregação aos dados existentes.

Outro ponto assente é que a API REST que foi implementada nesta dissertação é uma API minimalista, pois apenas foram desenvolvidos dezasseis métodos (vid. métodos 3.2), que permitem operações (CRUD) sobre a base de dados do SIDE, bem como capacidade de extensão de novas funcionalidades.

Tendo em conta os objetivos gerais delineados no início do trabalho desta dissertação, especificou-se, implementou-se e desenvolveu-se um caso de utilização da API REST, de forma a testar a solução alcançada. Assim, foi possível testar novas soluções no que toca a desenvolvimento de aplicações, pois foi desenvolvida uma aplicação SPA¹ em Angular Material, salientando-se mais uma vez a abstração no consumo de dados da API.

Outro objetivo específico deste trabalho era também a possibilidade de gestão e criação de novos perfis de acesso aos dados, objetivo alcançado com os mecanismos que permitem adicionar novos perfis, bem como definir as permissões de acesso aos recursos.

Finalmente, com os testes de carga sobre a API, também foi possível constatar a melhoria do tempo de resposta conforme se aumentava o número de processos

¹*Single-page application.*

que executavam o serviço, sublinhando a escalabilidade como uma das principais características da API desenvolvida.

Em síntese, quando for necessário desenvolver / utilizar um novo sistema de apoio ao ensino, a API estará presente, para permitir (camada de alto nível de abstração) a interoperabilidade com o novo sistema.

6.2 Trabalho futuro

Como trabalho futuro surgem alguns pontos, entre os quais, otimização / organização de código fonte e publicação em repositório público (e.g., Github²).

Outro ponto com relevância futura será a implementação de novas estratégias de autenticação e novos mecanismos de armazenamento de sessões, com possibilidade de utilização de *cache*.

Também se pretende no futuro desenvolver novas aplicações (RIA³), que permitiam aos utilizadores conseguir aceder a informação mais centralizada como avisos, horários, avaliações, exames, notas, submissão de trabalhos, etc.

Outro trabalho futuro será a migração para novas infraestruturas como infraestrutura na nuvem (e.g., PAAS, IAAS) ou *applicatin containers* (e.g., Docker), o que será um trabalho facilitado, pois a API foi desenvolvida utilizando módulos que facilitam a migração, podendo assim tirar partido da escalabilidade da API REST desenvolvida.

Também futuramente, se pretende implementar HATEOAS⁴ na API REST desenvolvida, permitindo maior facilidade de navegação na API e também maior capacidade de descoberta da API por parte de aplicações que a utilizem.

Igualmente, também se pretende desenvolver e implementar novos modelos de controlo de acesso, nomeadamente o modelo ABAC (*Attribute-based Access*

²Plataforma para repositório de código-fonte com controle de versão usando o Git (Linus Torvalds, 2005).

³*Rich Internet application*.

⁴*Hypermedia as the Engine of Application State*.

Control), de forma a aumentar a flexibilidade da API REST.

Referências bibliográficas

- Abbas, R., Sultan, Z., and Bhatti, S. N. (2017). Comparative analysis of automated load testing tools: Apache JMeter, Microsoft Visual Studio (TFS), LoadRunner, Siege. In *International Conference on Communication Technologies, ComTech 2017*. 159
- Abbate, J. E. (1994). From ARPANET to Internet: A history of ARPA-sponsored computer networks. <http://repository.upenn.edu/dissertations/AAI9503730>. Acesso em: 20 de Jun de 2017. 7
- Adams, K., Evans, J., Maher, B., Ottoni, G., Paroski, A., Simmers, B., Smith, E., and Yamauchi, O. (2014). The hiphop virtual machine. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications - OOPSLA '14*, pages 777–790, New York, New York, USA. ACM Press. 23
- Aiello, L. C. (2016). The multifaceted impact of ada lovelace in the digital age. *Artif. Intell.*, 235:58–62. 8
- Alksentrs (2008). Diagram illustrates autogeneration of the infrastructure code from

- an interface defined using the CORBA IDL. <https://en.wikipedia.org/wiki/File:Orb.svg>. Acesso em: 18 de Jun de 2018. 43
- API Blueprint (2016). API Blueprint. <https://apiblueprint.org/>. Acesso em: 24 de Nov de 2017. 80
- Armstrong, J., Virding, R., and Williams, M. (1986). Erlang. <http://www.erlang.org/>. Acesso em: 20 de Jun de 2017. 21
- Augustsson, L., Barton, D., Boutel, B., Burton, W., Fasel, J., Hammond, K., Hinze, R., Hudak, P., Hughes, J., Johnsson, T., Jones, M., Jones, S. P., Launchbury, J., Meijer, E., Peterson, J., Reid, A., Runciman, C., and Wadler, P. (2010). Haskell. <https://www.haskell.org/>. Acesso em: 20 de Jun de 2017. 24
- Bachman, C. W. (1973). The programmer as navigator. *Communications of the ACM*, 16(11):653–658. 101
- Barbosa, L., Alves, P., and Barroso, J. (2011). SIDE - Teaching support information system. In *Proceedings of the 6th Iberian Conference on Information Systems and Technologies, CISTI 2011*. 1
- Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. (2004). Web Services Architecture, W3C Working Group Note 11 February 2004. *World Wide Web Consortium*, article available from: <http://www.w3.org/TR/ws-arch>, page 13. 41
- Bright, W. and Digital Mars (2001). D programming language. <https://dlang.org/>. Acesso em: 16 de Jun de 2018. 20
- Brocardo, M. L., Traore, I., Woungang, I., and Obaidat, M. S. (2017). Authorship verification using deep belief network systems. *International Journal of Communication Systems*, 30(12):e3259. 106
- BVBA, B. (2017). WHAT ARE THE MOST INFLUENTIAL PROGRAMMING LANGUAGES OF 2018 AND WHICH ARE ON THE RISE? <https://>

- www.brainbridge.be/news/what-are-the-most-influential-programming-languages-of-2018-and-which-are-on-the-rise. 27
- Cake Software Foundation, I. (2005). CakePHP Framework. <https://cakephp.org/>. Acesso em: 15 de Jun de 2018. 89
- Cardelli, L. (1996). Type systems. *ACM Comput. Surv.*, 28(1):263–264. 12
- Cattell, R. (2011). Scalable sql and nosql data stores. *SIGMOD Rec.*, 39(4):12–27. 103
- Chaniotis, I. K., Kyriakou, K. I. D., and Tselikas, N. D. (2015). Is Node.js a viable option for building modern web applications? A performance evaluation study. *Computing*. 32
- Chen, L. (2018). Microservices: Architecting for continuous delivery and devops. In *2018 IEEE International Conference on Software Architecture (ICSA)*, volume 00, pages 39–397. 98
- Christie, T. (2011). Django REST Framework. <https://www.django-rest-framework.org/>. Acesso em: 24 de Nov de 2017. 87
- Codd, E. F. (1983). A relational model of data for large shared data banks. *Commun. ACM*, 26(1):64–69. 102
- Corporation, I. (2013). IBM Information Management System (IMS) 13 Transaction and Database Servers delivers high performance and low total cost of ownership. <http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?subtype=ca&infotype=an&appname=iSource&supplier=897&letternum=ENUS213-381>. 101
- CYRANO (2001). OpenSTA. <http://opensta.org/>. Acesso em: 20 de Out de 2018. 117
- Dahl, R. (2017). Node.js. <https://nodejs.org/en/>. Acesso em: 24 de Nov de 2017. 31

- Dershem, H. L. and Jipping, M. J. (1990). *Programming Languages: Structures and Models*. Wadsworth Publ. Co., Belmont, CA, USA. 8
- Dmitry Namiot, M. S.-S. (2014). On Micro-services Architecture. *International Journal of Open Information Technologies*, 2(9):24–27. 97
- Drake, W. and Force, U. N. I. T. (2005). *Reforming Internet Governance: Perspectives from the Working Group on Internet Governance (WGIG)*. Department of Economic and Social Affairs - Ict Task Force. United Nations Information and Communication Technologies Task Force. 81
- EBizMBA (2018). Top 15 Most Popular Websites | May 2018. <http://www.ebizmba.com/articles/most-popular-websites>. Acesso em: 20 de Out de 2018. 15, 79
- Eddon, G. and Eddon, H. (1998). *Inside Distributed COM*. Microsoft programming series. Microsoft Press. 55
- Facebook (2018). O que é a autenticação de dois fatores e como funciona? <https://pt-pt.facebook.com/help/148233965247823>. Acesso em: 17 de Jun de 2018. 109
- Fulmer, J. (2012). Siege. <https://www.joedog.org/siege-home/>. Acesso em: 20 de Jul de 2018. 118
- Garrett, J. (2005). Ajax: A new approach to web applications. *Adaptive Path*. 28
- Ghossoon M. Waleed, R. A. (2009). SOAP Message Structure. https://www.researchgate.net/profile/Ghossoon_Waleed/publication/224386981/figure/fig1/AS:393832328908816@1470908352332/Figure-1-SOAP-Message-Structure.jpg. Acesso em: 04 de Jul de 2018. 66
- Gosling, J. (1995). Java. <http://oracle.com/java/>. Acesso em: 17 de Jun de 2017. 26

- Greenhalgh, C. (2008). Java RMI programming by example. <http://www.cs.nott.ac.uk/~pszcmg/G53ACC/java-rmi/rmi-tutorial.html>. Acesso em: 20 de Set de 2018. 52
- Griesemer, R., Pike, R., and Thompson, K. (2009). The Go Programming Language. <https://golang.org>. Acesso em: 17 de Abr de 2018. 22
- Group, T. (2018). ECMAScript® 2018 Language Specification. <https://www.ecma-international.org/ecma-262/9.0/index.html>. Acesso em: 02 de Out de 2018. 30
- Halili, E. H. (2008). Apache JMeter. <https://jmeter.apache.org/>. Acesso em: 20 de Jul de 2018. 116
- Hasegawa, Y. (2000). http://www.cs.cmu.edu/~yhase/tech/iStudy_files/image014.gif. Acesso em: 04 de Set de 2018. 45
- Hogg, S. (2014). Software Containers: Used More Frequently than Most Realize. <https://www.networkworld.com/article/2226996/cisco-subnet/software-containers--used-more-frequently-than-most-realize.html>. Acesso em: 20 de Out de 2018. 4
- Holowaychuk, T. (2010). Express - Node.js web application framework. <https://expressjs.com/>. Acesso em: 15 de Jun de 2017. 84
- IBM (2013). LoopBack Framework. <https://loopback.io/>. Acesso em: 17 de Abr de 2017. 94
- Ilegbodu, B. (2015). History of ECMAScript. <http://www.benmvp.com/learning-es6-history-of-ecmascript/>. Acesso em: 20 de Out de 2017. 27
- IT, S. (2018). DB-Engines Ranking. <https://db-engines.com/en/ranking>. Acesso em: 30 de Out de 2018. 105
- Kambona, K., Boix, E. G., and Meuter, W. D. (2013). An evaluation of reactive programming and promises for structuring collaborative web applications. In

Proceedings of the 7th Workshop on Dynamic Languages and Applications, New York, NY, USA. ACM, ACM. 30

Kevin Burke, Kyle Conroy, R. H. (2017). Flask RESTful Framework. <https://flask-restful.readthedocs.io/en/latest/>. Acesso em: 12 de Mar de 2017. 84

Landelle, S. (2012). Gatling. <https://gatling.io/>. Acesso em: 20 de Jul de 2018. 117

Laverdet, M. (2010). XHP. <https://github.com/hhvm/xhp-lib/releases>. Acesso em: 17 de Jun de 2018. 39

Linus Torvalds, J. H. (2005). Git. <https://git-scm.com/>. Acesso em: 05 de Jun de 2017. 169

Lira, H. A., Dantas, J. R. V., Muniz, B. d. A., Nunes, T. M., and Farias, P. P. M. (2015). An Approach to Support Data Integrity for Web Services Using Semantic RESTful Interfaces. In *Proceedings of the 24th International Conference on World Wide Web Companion*. 82

Loeffler, B. (2011). What is Infrastructure as a Service? <https://social.technet.microsoft.com/wiki/contents/articles/4633.what-is-infrastructure-as-a-service.aspx>. Acesso em: 20 de Out de 2017. 4

Lord, R. (2013). Slate. <https://github.com/lord/slate>. Acesso em: 08 de Mar de 2017. 80

Markus Horstmann, M. K. (1997). DCOM Architecture. http://cs.hadassah.ac.il/staff/martin/Seminar/middleware/DCOM_arch.pdf. Acesso em: 20 de Set de 2018. 54, 55

Mashery (2011). I/O Docs. <http://www.mashery.com/product/io-docs>. Acesso em: 08 de Mar de 2017. 81

- Masood, A. (2015). PERFORMANCE TESTING: KEY CONCEPTS, ISSUES & TESTING TYPES. <http://www.testerlogic.com/performance-testing-types-concepts-issues/>. Acesso em: 20 de Out de 2018. 114
- Matsumoto, Y. (1995). Ruby. <http://www.ruby-lang.org>. Acesso em: 07 de Mar de 2017. 37
- McCord, C. (2014). Phoenix framework. <https://phoenixframework.org/>. Acesso em: 12 de Mar de 2017. 85
- McNeil, M. (2012). Sails.js Framework. <https://sailsjs.com/>. Acesso em: 12 de Mar de 2017. 93
- Mesoderm (2013). Arquitetura Peer-to-peer. https://upload.wikimedia.org/wikipedia/en/f/fa/Unstructured_peer-to-peer_network_diagram.png. Acesso em: 18 de Jun de 2018. 9
- Microsoft (2000). C#. <https://docs.microsoft.com/dotnet/csharp/language-reference/>. Acesso em: 20 de Out de 2018. 16
- Mishra, V. (2017). Monolithic and microservices architecture. <https://www.tatvasoft.com/blog/wp-content/uploads/2016/06/Monolithic-Micro-services-Architecture-768x421.png>. Acesso em: 17 de Jul de 2018. 98
- Mizerany, B. (2007). Sinatra Framework. <http://sinatrarb.com/>. Acesso em: 12 de Mar de 2017. 91
- Moreau, E. (2016). The Top 25 Social Networking Sites People Are Using. <http://webtrends.about.com/od/socialnetworkingreviews/tp/Social-Networking-Sites.htm>. Acesso em: 12 de Out de 2018. 136
- Network, M. D. (2014). Promise. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise. Acesso em: 03 de Jul de 2018. 30

- Odersky, M. (2004). Scala. <http://www.scala-lang.org/>. Acesso em: 24 de Nov de 2017. 38
- Open API Initiative (2016). Swagger RESTful API Documentation Specification. <http://docs.swagger.io/spec.html>. Acesso em: 24 de Nov de 2017. 81
- Oracle (2017). Arquitetura RMI. <https://docs.oracle.com/javase/tutorial/figures/rmi/rmi-2.gif>. Acesso em: 26 de Jun de 2018. 50
- O’Sullivan, B. (2014). Where credit belongs for Hack. <http://www.serpentine.com/blog/2014/03/28/where-credit-belongs-for-hack/>. Acesso em: 24 de Nov de 2017. 23
- Otwell, T. (2011). Laravel Framework . <https://github.com/laravel/framework>. Acesso em: 24 de Nov de 2017. 88
- Parvez (2017). REST Webservice. <https://www.phpflow.com/wp-content/uploads/2012/07/rest-webservices.jpg>. Acesso em: 10 de Jul de 2018. 68
- Patil, A., Rajesh, K., and Sabharwal, K. (2011). Comparison of Middleware Technologies - CORBA , RMI & COM/DCOM. <https://pdfs.semanticscholar.org/f4a2/2d516dbb7482ab01722760d8da586b87c2a0.pdf>. 41, 42, 49, 54
- Pliik (2015). Gugamarket Framework. <http://www.gugamarket.com/>. Acesso em: 17 de Abr de 2018. 95
- Pressman, R. and Maxim, B. (2016). *Engenharia de Software - 8ª Edição*. 26
- Raj, G. S. (1998). A Detailed Comparison of CORBA, DCOM and Java/RMI. <http://gsraj.tripod.com/misc/compare.html>. Acesso em: 20 de Out de 2018. 45, 51
- RAML Workgroup (2013). RAML Version 0.8: RESTful API Modeling Language. <http://raml.org/spec.html>. Acesso em: 24 de Nov de 2017. 80

- Restlet, I. (2005). Restlet Framework. <http://restlet.com/products/restlet-framework/>. Acesso em: 17 de Abr de 2018. 90
- Richardson, C. (2014a). <https://res.infoq.com/articles/microservices-intro/en/resources/3Fig5.png>. Acesso em: 03 de Jul de 2018. 99
- Richardson, C. (2014b). Microservices: Decomposing Applications for Deployability and Scalability. <https://www.infoq.com/articles/microservices-intro>. Acesso em: 20 de Out de 2017. 97
- Ritchie, D. M. (1993). The development of the C language. In *The second ACM SIGPLAN conference on History of programming languages - HOPL-II*, volume 28, pages 201–208, New York, New York, USA. ACM Press. 18
- Rocher, G. (2011). Grails (framework). <https://grails.org/>. Acesso em: 17 de Abr de 2017. 95
- Rogers, P. (2005). Service-Oriented Development on NetKernel- Patterns, Processes & Products to Reduce System Complexity | Cloud Computing Expo. In *CloundExpo 2005*. 97
- Severance, C. (2012a). Inventing PHP: Rasmus Lerdorf. *Computer*, 45(11):6–7. 35
- Severance, C. (2012b). JavaScript: Designing a Language in 10 Days. *Computer*, pages 7–8. 27, 33
- shamim Hassan, A. (2017). How do you authenticate, mate? <https://hackernoon.com/how-do-you-authenticate-mate-f2b70904cc3a>. Acesso em: 10 de Fev de 2018. 106
- Silva, R. F. (2004). A importância da interoperabilidade. <http://www.phpbrasil.com/artigo/-XVEH0RZGzGr/a-importancia-da-interoperabilidade>. Acesso em: 20 de Out de 2018. 3
- Slant (2018). 18 Best web frameworks to create a web REST API as of 2018 - Slant. <https://www.slant.co/topics/1397/~best-web-frameworks-to-create-a-web-rest-api>. Acesso em: 20 de Out de 2018. 83

- Software, P. (2013). Spring Boot. <https://spring.io/projects/spring-boot>. Acesso em: 05 de Jun de 2017. 86
- StackOverflow (2014a). <https://i.stack.imgur.com/4BPbz.jpg>. Acesso em: 03 de Jul de 2018. 32
- StackOverflow (2014b). <https://i.stack.imgur.com/cRq1h.jpg>. Acesso em: 03 de Jul de 2018. 33
- Stallings, W. and Brown, L. (2014). *Computer Security: Principles and Practice*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition. 105, 110, 111, 113
- Stervinou, J.-Y. (2003). XML-RPC. https://docs.typo3.org/typo3cms/extensions/xmlrpc_server/_images/img-1.png. Acesso em: 03 de Jul de 2018. 58
- Stowe, M. (2015). The Power of RAML. <https://www.infoq.com/articles/power-of-raml>. Acesso em: 11 de Out de 2018. 80
- Stroustrup, B. (1994). *The C++ Programming Language, 3rd Edition*. 19
- Team, R. (2011). Restify Framework. <http://restify.com/>. Acesso em: 05 de Jun de 2017. 92
- Technologies, Z. (2006). Zend Framework. <https://framework.zend.com/>. Acesso em: 05 de Jun de 2017. 88
- Tilkov, S. and Vinoski, S. (2010). Node.js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*. 32
- TIOBE (2018). TIOBE Index for January 2018. <https://www.tiobe.com/tiobe-index/>. Acesso em: 02 de Mar de 2018. 28
- Turner, D. M. (2016). Digital Authentication: The Basics. <https://www.cryptomathic.com/news-events/blog/digital-authentication-the-basics>. Acesso em: 20 de Mai de 2018. 106

- van Rossum, G. (1991). Python. <https://www.python.org/>. Acesso em: 05 de Jun de 2017. 36
- Vignoni, D. (2011). Arquitetura cliente-servidor. <https://userscontent2.emaze.com/images/7d7c5b69-807a-4425-be1c-8f692a69856b/66c398359e7857d60d712d30ef63ad29.jpg>. Acesso em: 18 de Jun de 2018. 9
- Voormann, H. (2006). SOAP. <https://upload.wikimedia.org/wikipedia/commons/4/4a/Webservices.png>. Acesso em: 04 de Jul de 2018. 63
- W3C (2000). SOAP Specifications. <https://www.w3.org/TR/soap/>. Acesso em: 09 de Jul de 2018. 63
- Wagh, K. and Thool, R. (2012). A Comparative Study of SOAP Vs REST Web Services Provisioning Techniques for Mobile Host. *Journal of Information Engineering and Applications*. 76
- Wall, L. (1987). Perl. <https://www.perl.org/>. Acesso em: 05 de Jun de 2017. 34
- Wendel, P. (2011). Spark Framework. <http://sparkjava.com/>. Acesso em: 05 de Jun de 2017. 91
- Writer, C. (2017). What Is the Role of the Database Management System in Information Systems? <https://bizfluent.com/about-6546834-role-management-system-information-systems-.html>. Acesso em: 20 de Set de 2018. 101
- Xiao, H. (2005). Arquitetura DCOM. <http://active-undelete.com/pictures/dcomtec03.gif>. Acesso em: 26 de Jun de 2018. 54
- Zimmerman, K. A. (2012). Internet History Timeline: ARPANET to the World Wide Web. <http://www.livescience.com/20727-internet-history.html>. Acesso em: 03 de Jul de 2018. 8

Sobre o Autor

Fernando Manuel Fernandes Rodrigues graduated in Electrical Engineering (electronics, instrumentation and computation) from the University of Trás-os-Montes and Alto Douro (UTAD), Portugal in 2001.

He attended and finished the curricular component of the master's degree in Engineering Technologies from the University of Trás-os-Montes and Alto Douro (UTAD), Portugal in 2003.

Since 2002, he has been working as a informatics technician and informatics specialist in the Informatics and Communications Services from the University of Trás-os-Montes and Alto Douro (UTAD).

