

PEDRO DE ALMEIDA PERDIGÃO

DIÁLOGO SAUDÁVEL

Desenvolvimento da aplicação *Debaqi*

Orientador: Professor Doutor Manuel Arturo Marques Pita

Universidade Lusófona de Humanidades e Tecnologias

Escola de Comunicação, Artes e Tecnologias de Informação (ECATI)

Departamento de Engenharia Informática e Sistemas de Informação

Lisboa

2020

PEDRO DE ALMEIDA PERDIGÃO

DIÁLOGO SAUDÁVEL

Desenvolvimento da aplicação *Debaqi*

Dissertação defendida em provas públicas no dia 17 de Junho de 2020, para a obtenção do Grau de Mestre em Engenharia Informática e Sistemas de Informação no Curso de Mestrado em Engenharia Informática e Sistemas de Informação, conferido pela Universidade Lusófona de Humanidades e Tecnologias, perante o Júri nomeado pelo Despacho de Nomeação de Júri nº 06/2020, de 08 de Janeiro de 2020, com a seguinte composição:

Presidente: Professor Doutor José Luís Azevedo Quintino Rogado (ULHT)

Arguente: Professor Doutor Francesco Costigliola (ULHT)

Orientador: Professor Doutor Manuel Arturo Marques Pita (ULHT)

Universidade Lusófona de Humanidades e Tecnologias

Escola de Comunicação, Artes e Tecnologias de Informação (ECATI)

Departamento de Engenharia Informática e Sistemas de Informação

Lisboa

2020

EPÍGRAFE

*Without requirements or design, programming is
the art of adding bugs to an empty text file*

LOUIS SRYGLEY

Application Architect – UPS

DEDICATÓRIA

Aos meus pais, especialmente, em memória da minha querida Mãe, Maria Manuel Perdigão

À Sara Custóias

Ao meu filho, Manuel Perdigão

À minha avó, Maria José Fragoso

Ao meu irmão, Manuel Perdigão

AGRADECIMENTOS

Ao Professor Doutor Manuel Pita pela maneira cativante com que orientou esta dissertação

À Rede de Bibliotecas Escolares

A todos os Professores Bibliotecários que aceitaram abraçar este projeto nas suas escolas

A todos os alunos e Pais que concordaram em participar nesta investigação

À minha grande amiga Sónia Borges Araújo pelo incansável apoio

Obrigado

RESUMO

Ao longo dos tempos a introdução de novas tecnologias no nosso quotidiano tem vindo a moldar gerações e a revolucionar o modo como trabalhamos e vivemos em sociedade. A tecnologia, para além de ter tornado as nossas vidas bastante mais fáceis e simultaneamente mais eficientes, abalou profundamente os alicerces da necessidade humana de socialização presencial. Há algumas décadas, seria necessário esperar vários dias e até, nalguns casos, meses por uma mensagem, privilegiando-se a comunicação direta. Agora, através de uns meros cliques num *smartphone* podemos enviar instantaneamente um *e-mail* ou uma mensagem a qualquer pessoa no Mundo, desde que tenha em sua posse um dispositivo habilitado a receber este tipo de conteúdo. A crescente disponibilização de *chats* e redes sociais como meios de comunicação indireta entre pessoas através da *Internet* tem vindo a mudar profundamente a maneira como os Seres Humanos tendem a comunicar, afetando negativamente a aptidão de comunicação frontal. Destacam-se neste âmbito os jovens - os «*heavy users*» - enquanto população alvo deste estudo, por serem aqueles que tendem a ter maior fragilidade na «saúde» de um diálogo *online*.

Atentos a esta temática, a Rede de Bibliotecas Escolares e o COPELABS desafiou-nos a desenvolver uma aplicação *Web*, que recorrendo às mais atuais tecnologias «*open-source*» e *Cloud*, permitisse realizar debates *online* em formato de *chat*, onde os utilizadores pudessem partilhar livremente e anonimamente as suas opiniões sobre qualquer tema passível de ser parametrizado no sistema: uma espécie de «prós e contras» digital. Por isso, «batizámos» esta aplicação com o nome de *Debaqi*, um diminutivo de «debate aqui». Esta plataforma, para além de ser um meio de comunicação digital, destaca-se das restantes por incluir algoritmos de inteligência artificial que permitem extrair métricas referentes à «saúde» de um diálogo.

No âmbito desta investigação, convidámos diversos Professores de várias Escolas Nacionais para criarem grupos de 15 a 20 de estudantes. Mediante autorização prévia dos respetivos encarregados de educação, os alunos puderam expressar as suas ideias acerca de temas da atualidade através da aplicação *Debaqi*. Os estudantes tiveram oportunidade de debater assuntos como a alteração climática, a privacidade no namoro e a violência de género. A análise dos dados gerados nas atividades visa apoiar a evolução do papel do Professor Bibliotecário numa sociedade constantemente *online*, facultando ferramentas que permitam identificar tendências, áreas problemáticas e oportunidades de intervenção em tempo real.

Palavras-chave: Diálogo, novas tecnologias, comunicação digital, *Debaqi*, análise dos dados

ABSTRACT

Over the years, the introduction of new technologies in our daily lives has been shaping generations and revolutionizing the way we work and live in society. Technology, in addition to make our lives much easier and at the same time more efficient, has deeply shook the foundations of the Human need for face-to-face interaction. A few decades ago, it would be necessary to wait several days, or even months in some cases, for a message, favouring direct communication. Now, by simply clicking on a smartphone, we can instantly send an email or a message to anyone in the World who has a device enabled to receive such content. The growing availability of chats and social networks as indirect media between people over the Internet has profoundly changed the way Humans tend to communicate, negatively affecting the ability for frontal communication. In this context, the youth - the «heavy users» - stands out as the target population of this study, because they are those who tend to have greater fragility in the «health» of an online dialogue.

Aware of this subject, the Rede de Bibliotecas Escolares and COPELABS has challenged us to develop a web application that by using cutting-edge open-source and Cloud technologies, would enable holding online chat debates, where users could freely and anonymously share their opinions on any topic that could be parameterised in the system: a kind of digital "pros e contras". We therefore named this application *Debaqi*, a diminutive of «debate aqui» (debate here). This platform, besides being for digital communication, stands out from the rest by including artificial intelligence algorithms that allow extracting health-related metrics from a dialogue.

As part of this investigation, we invited several Teachers from various National Schools to create groups of 15 to 20 students. Upon prior authorization from their guardians, students were able to express their ideas on current topics through the *Debaqi* application. Students had the opportunity to discuss topics such as climate change, privacy in dating and gender-based violence. Analysis of data generated from those activities aims to support the evolution of the role of the Professor Bibliotecário (librarian teacher) in a constantly online society, providing tools to identify trends, problem areas and real time intervention opportunities.

Keywords: Dialogue, new technologies, digital communication, *Debaqi*, data analysis

ABREVIATURAS E SÍMBOLOS

0..* – Zero or many

.. – Many to Many

1..* – One or many

1-1 – One to one

*..1 – Many or one

∞ – Infinito

AJAX – Asynchronous JavaScript and XML

API – Application Programming Interface

AWS – Amazon Web Services

BaaS – Back-end as a Service

BD – Base de Dados

BOW – Bag-of-words

Cf. – Confira

CI/CD – Continuous Integration / Continuous Delivery

Consult. – Consultado

CPU – Central Processing Unit

CRUD – Create, read, update, delete

CSS – Cascading Style Sheets

DBMS – Database Management System

DOM – Document Object Model

EE – Enterprise Edition

Etc. – et cetera

FaaS – Function as a Service

FCT – Fundação para a Ciência e a Tecnologia

HMAC – Hash-based Message Authentication Code

HTML – Hypertext Markup Language

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

IA – Inteligência Artificial

IDE – Integrated Development Environment

IoT – Internet of Things

IT – Information technology

IT/Ops – Information technology / Operations
JDBC – Java Database Connectivity
JS – JavaScript
JSON – JavaScript Object Notation
JWT – JSON Web Token
MSG – Mensagem
N/A – Não aplicável
NoSQL – No Structured Query Language
NLP – Natural Language Processing
ORM – Object-relational mapping
p. – Página
PaaS – Platform as a Service
PoC – Proof of Concept
POO – Programação Orientada a Objetos
REST – Representational State Transfer
RGPD – Regulamento Geral sobre a Proteção de Dados
RBE – Rede de Bibliotecas Escolares
SHA – Secure Hash Algorithm
SOA – Service Oriented Architecture
SpEL – Spring Expression Language
SPNS – Sequência Potencialmente Não Saudável
SQL – Structured Query Language
SSL/TLS – Secure Sockets Layer / Transport Layer Security
TCP – Transmission Control Protocol
TF-IDF – Term frequency-inverse document frequency
UI/UX – User Interface / User Experience
URL – Uniform Resource Locator
vs. – Versus
Web – World Wide Web
WSS – Secure Web Socket

ÍNDICE GERAL

EPÍGRAFE	1
DEDICATÓRIA	2
AGRADECIMENTOS	3
RESUMO.....	4
ABSTRACT	5
ABREVIATURAS E SÍMBOLOS	6
ÍNDICE DE QUADROS	10
ÍNDICE DE TABELAS	11
ÍNDICE DE FIGURAS	12
INTRODUÇÃO	15
Organização.....	20
CAPÍTULO I	21
I – A EVOLUÇÃO DA RELAÇÃO HUMANA NA <i>INTERNET</i>	22
1.1. Dos <i>chats</i> às redes sociais.....	22
1.2 Desconexão social	23
CAPÍTULO II.....	25
II – ARQUITETURA DE APLICAÇÕES <i>WEB</i>	26
2.1. Funcionamento de uma aplicação <i>Web</i>	27
2.2 Componentes de aplicações <i>Web</i>	29
2.3 Arquitetura de um servidor <i>Web</i>	30
2.4 Modelos de componentes de aplicações <i>Web</i>	31
2.5 Arquiteturas de aplicações <i>Web</i>	32
2.6 Dicas para o desenvolvimento de aplicações <i>Web</i>	43
2.7 <i>WebSockets</i>	44
2.8 <i>Docker containers</i>	46
CAPÍTULO III	49
III – <i>DEBAQI</i>	50
3.1. Objetivo geral	50
3.2 Objetivo específico	50
3.3 Tecnologias e bibliotecas.....	51
3.4 Arquitetura da solução	52
3.5 Controlo de acesso na aplicação	54
3.6 Funcionalidades vs. Perfil	57
3.7 Funcionamento da aplicação <i>Debaqi</i>	58
3.8 Estrutura e manipulação de dados na aplicação <i>Debaqi</i>	80
3.9 <i>DevOps</i> – <i>Debaqi</i>	87

CAPÍTULO IV.....	91
IV – ANÁLISE DOS DADOS.....	92
4.1. – O conhecimento que pretendemos obter	92
4.2 – Os objetivos pretendidos com os novos conhecimentos	95
4.3 – Definição operacional de um «diálogo saudável»	96
4.4 – Alinhamento	99
4.5 – Coerência.....	100
4.5 – Coesão	101
4.6 – Resultados.....	102
CONCLUSÃO.....	106
Trabalhos futuros	107
BIBLIOGRAFIA	109
GLOSSÁRIO	111
APÊNDICES	I
Apêndice I – Estrutura relacional de dados desenvolvida para a aplicação <i>Debaqi</i>	II
Apêndice II – Diagrama de «classes» que demonstram as entidades desenvolvidas para a aplicação <i>Debaqi</i>	III
Apêndice III – Diagrama de Entidade-Relação desenvolvido para a aplicação <i>Debaqi</i>	IV

ÍNDICE DE QUADROS

Quadro 1: Diferenças técnicas entre um pedido HTTP e uma ligação por WebSocket	46
Quadro 2: Tecnologias relevantes no desenvolvimento da aplicação Debaqi, agrupadas por camadas tecnológicas	51
Quadro 3: «Check-list» de perfis que têm acesso ou não às diversas funcionalidades presentes na aplicação	57
Quadro 4: As diferentes funções disponíveis em cada ORM e a sintaxe SQL que estas «geram»	81
Quadro 5: Exemplo de código desenvolvido para cada ORM, de forma a implementar um modelo relacional de base de dados	82

ÍNDICE DE TABELAS

Tabela 1: Número de participantes por atividade, agrupados por escola	102
Tabela 2: Número de intervenções realizadas em cada atividade, agrupadas por escola	102

ÍNDICE DE FIGURAS

Figura 1: Perspetivas estratégicas de interação dialogal	18
Figura 2: Modelo conceptual back-end e front-end.....	28
Figura 3: «Workflow» de um pedido HTTP	30
Figura 4: Modelo conceptual de uma arquitetura monolítica.....	35
Figura 5: Modelo conceptual de uma arquitetura em micro serviços.....	39
Figura 6: Modelo conceptual de uma arquitetura Serverless	42
Figura 7: Comparação do «workflow» entre um pedido HTTP e uma ligação por WebSocket.....	45
Figura 8: Diferenças estruturais inerentes às arquiteturas tradicionais em máquinas virtuais face às recentes arquiteturas em containers	48
Figura 9: Diagrama conceptual da arquitetura implementada para a aplicação Debaqi	53
Figura 10: «Print screen» da página inicial	58
Figura 11: «Print screen» do «modal window» para autenticação dos utilizadores.....	59
Figura 12: «Print screen» da página que possibilita o registo do aluno	59
Figura 13: «Print screen» da página que possibilita a gestão de perguntas para serem disponibilizadas no ecrã de registo	61
Figura 14: «Print screen» do «modal window» visualizado após um registo efetuado com sucesso	62
Figura 15: «Print screen» da página inicial da aplicação, para um utilizador que possua perfil de «aluno»	63
Figura 16: «Print screen» da página que possibilita visualizar a informação genérica do aluno	63
Figura 17: «Print screen» da página que possibilita o utilizador entrar nos debates a decorrer.....	64
Figura 18: «Print screen» do «modal window» apresentado antes do aluno entrar no debate a decorrer.....	65
Figura 19: «Print screen» da página que apresenta a lista de todos os tópicos existentes	66

Figura 20: «Print screen» da página que permite criar ou editar um tópico.....	66
Figura 21: «Print screen» da página que apresenta a lista de todas as salas de debates existentes	67
Figura 22: «Print screen» da página da página que permite criar ou editar uma sala de debate.....	67
Figura 23: «Print screen» da funcionalidade de chat.....	69
Figura 24: «Print screen» da funcionalidade de emojis Figura 25: «Print screen» da funcionalidade de envio de imagens	69
Figura 26: «Print screen» da funcionalidade que permite o envio de menções e reações a mensagens	69
Figura 27: «Print screen» do «modal window» pós-debate. Momento de reflexão....	71
Figura 28: «Print screen» da página inicial da aplicação, para um utilizador que possua perfil de «administrador».....	72
Figura 29: «Print screen» da página que possibilita efetuar a gestão dos utilizadores presentes na aplicação	73
Figura 30: «Print screen» da página que possibilita a visualização das métricas aplicacionais	74
Figura 31: «Print screen» da página que possibilita visualizar as auditorias realizadas à aplicação	75
Figura 32: «Print screen» da página que apresenta as auditorias realizadas aos dados transferidos por pedidos HTTP	76
Figura 33: «Print screen» da página que possibilita efetuar a gestão do nível de «logging» na aplicação	77
Figura 34: «Print screen» da ferramenta de UI - Swagger	78
Figura 35: «Print screen» das notificações de informação presentes na aplicação	79
Figura 36: Diagrama de «classes» alusivo às entidades que suportam efetuar o registo do aluno	83
Figura 37: Diagrama de «classes» das entidades pertencentes aos questionários pré-debate, pós-debate e questões do chatbot	85
Figura 38: Diagrama de «classes» referente às entidades que suportam as interações no chat	86

Figura 39: Código desenvolvido para a pipeline de integração automática com a Cloud Heroku	89
Figura 40: «Print screen» da funcionalidade «Pipeline» do Bitbucket, onde é apresentada a informação sobre o sucesso e insucesso das integrações automáticas do servidor Flask (Python)	89
Figura 41 – Processo de análise comparativa	93
Figura 42 – Processo de análise correlacional	94
Figura 43 – Processo de análise descritiva	95
Figura 44: Ilustração do método utilizado para criar a representação vetorial de um texto fonte $\mathbf{x}$	98
Figura 45: Ilustração do conceito de alinhamento entre vetores correspondentes a intervenções diferentes sobre a crise climática, usando como medida de similaridade o cosseno do ângulo entre os mesmos	100
Figura 46: Resultados da análise das intervenções durante a primeira sessão de debate sobre o tema privacidade no namoro	104
Figura 47: Resultados do questionário por inquérito no final da sessão de debate A1 escola E1	105

INTRODUÇÃO

De acordo com a definição proposta por A. Tavares Louro, a palavra diálogo “provém do vocábulo grego *diálogos* por intermédio do vocábulo latino *dialogu-*. A palavra original grega foi formada pelo elemento *dia-*, que significa «por intermédio de», e por *logos*, que significa «palavra». No conjunto, a palavra grega *diálogos* significa «conversa» e «conversação»” (Louro, 2007)¹. Literalmente, então, diálogo traduz qualquer comunicação entre dois ou mais intervenientes utilizando palavras para transmitir significado. Embora se desenvolva a partir de pontos de vista diferentes, o verdadeiro diálogo supõe um desenvolvimento de uma atmosfera de confiança, respeito mútuo, abertura e honestidade, através de um processo participativo que é particularmente adequado para atender as necessidades da sociedade.

Num diálogo saudável, os intervenientes reúnem-se para construir uma relação de confiança e entendimento comum, reconhecendo as suas diferenças, de forma a criar resultados positivos através da conversação. A avaliação contínua do progresso alcançado em conjunto aprofunda as relações entre as pessoas e fortalece a sua capacidade de fazer correções e enfrentar novos problemas ou oportunidades à medida que estas surgem.

A qualidade crítica do diálogo reside no facto de os participantes se reunirem num «espaço seguro» para entender o ponto de vista do outro, a fim de desenvolver uma nova compreensão dos problemas reconhecidos. O diálogo faculta um espaço no qual os participantes aprendem mais sobre si mesmos, sobre os outros e sobre os seus relacionamentos; que os seus interesses não podem ser levados com sucesso sem a colaboração de outros - até mesmo dos seus inimigos; que a cooperação pode gerar poder; e a trabalhar juntos para o interesse comum ou público, preservando as suas próprias identidades. O diálogo elabora-se através de um «pensamento em conjunto», que não inclui apenas os produtos do nosso intelecto consciente, mas também os nossos sentimentos, emoções, intenções e desejos. São esses aspetos, em grande parte não declarados e invisíveis das interações Humanas no diálogo, que levam as pessoas a aprender e a mudar. Os relacionamentos começam a mudar quando as pessoas envolvidas aprendem a ser abertas sobre as suas próprias preocupações, expectativas e incertezas, e quando se tornam prontas e capazes de ouvir os outros com a intenção de compreendê-los. As transformações individuais que ocorrem nos processos de diálogo tornam-se a fonte de

¹ [Consult. 01-12-2019]. Disponível em WWW:<URL:<https://ciberduvidas.iscte-iul.pt/consultorio/perguntas/a-origem-da-palavra-dialogo/22161>>

mudança nos grupos sociais, nas instituições e na sociedade em geral. Mudanças significativas nos relacionamentos levam tempo e implicam comprometimento por parte dos envolvidos.

Definimos frequentemente um diálogo como algo que não é – por exemplo, negociação, debate ou deliberação. O diálogo é diferente do debate, na medida em que incentiva a diversidade de pensamento e opiniões, em vez de suprimir essas noções. Na prática do diálogo há um acordo mediante o qual os conceitos ou crenças de uma pessoa não devem ter precedência sobre os dos outros. Isso significa que, no diálogo, a intenção não é advogar, mas inquirir; não discutir, mas explorar, não para convencer, mas para descobrir. Os participantes de um debate tendem a assumir a sua visão como a «mais correta» e investem para a pressionar e defender; o diálogo, por outro lado, visa a possibilidade de se encontrar uma solução melhor para um problema comum. O diálogo também não é uma discussão. Esta assemelha-se a um jogo de pingue-pongue, no qual as pessoas vão lançando as suas ideias de um lado para o outro, sendo que o objetivo é ganhar. Possivelmente, podemos reter algumas dessas ideias para nosso próprio «*backup*», concordando com algumas e discordando de outras – mas o ponto fundamental é ganhar o jogo. Num diálogo saudável, no entanto, ninguém está a tentar vencer.

O diálogo difere de uma intermediação ou negociação de diversas maneiras:

- ❖ O objetivo do produto de intermediação ou negociação é um acordo concreto. O objetivo do diálogo é um relacionamento;
- ❖ A moeda da intermediação é definir e satisfazer os interesses materiais por meio de acordos específicos e conjuntos. O resultado do diálogo é criar novas capacidades Humanas e políticas para resolver problemas;
- ❖ As negociações exigem que as partes estejam dispostas a tentar chegar a um acordo. O diálogo pode ser frutífero envolvendo partes que ainda não estão prontas para negociações, mas não querem um contínuo relacionamento destrutivo;
- ❖ O diálogo não substitui a negociação e a intermediação em situações de conflito. É uma parte essencial nos processos de resolução e prevenção de conflitos, nos quais o objetivo é construir uma paz sustentável. Ao traçar distinções claras, o diálogo e os restantes processos fazem parte de uma iniciativa maior de paz;
- ❖ Existem também diálogos «falsos». Estes podem ser processos que aproximam as pessoas principalmente para demonstrar que as partes opostas podem sentar-se juntas «à mesa», mas evitando totalmente as questões difíceis que as mantêm divididas.

O diálogo deve estar assente sobre 5 princípios fundamentais:

1. **Inclusão** – Os processos de diálogo que promovem a democracia devem ser inclusivos, pois todos os cidadãos têm o direito de ser ouvidos, incluindo organismos políticos, económicos, sociais e militares, bem como todos aqueles que foram continuamente omissos no passado. Se a inclusão não for abrangente, esta situação pode comprometer a sustentabilidade no exercício do diálogo;
2. **Propriedade coletiva** – Este critério exige, no mínimo, que o processo de diálogo não seja um instrumento de apenas um ator, por exemplo, o Governo. Pelo contrário, o diálogo é uma «troca», mesmo quando convocado por instituições poderosas. O diálogo incorpora a noção democrática de que todos estão envolvidos e comprometidos igualmente. É uma via onde nenhum dos lados está a ditar o outro;
3. **Aprendizagem** – O diálogo cria uma oportunidade para aprender através da autorreflexão. Os intervenientes começam a perceber que cada um só tem um pouco de verdade. Numa escala maior, isso pode levar ao desenvolvimento de «conhecimento público» que se pode tornar numa mudança muito positiva e mais sustentável. O diálogo não é sobre o ato físico de falar, mas sim, sobre a mente se expandir.

Distingue um diálogo saudável de um diálogo «falso», onde a comunicação é unidirecional e, a partir de um debate ou negociação, os participantes concentram-se apenas em ganhar o máximo possível para o seu próprio partido. Os principais pontos que criam uma interação saudável são os seguintes:

- a. **Escutar** – sem resistência ou imposição;
 - b. **Respeitar** – consciência da integridade da posição de outra pessoa e a impossibilidade de compreendê-la plenamente;
 - c. **Suspensão** – suspensão de premissas, julgamento e certeza.
4. **Humanidade** – Esta característica tem muito que ver com o modo como as pessoas se comportam quando se comprometem totalmente no diálogo. Requer empatia – a capacidade de se colocar no lugar do outro. Quando as pessoas começam a esforçar-se para entender o outro, a «semente» do diálogo é plantada;

5. **Perspetiva a longo prazo** – Diálogo é saber usar o tempo de uma maneira diferente, no sentido de perceber que não há soluções rápidas. Mudanças significativas nos relacionamentos levam tempo e comprometimento dos envolvidos.

A capacidade de dialogar é sem dúvida uma das características mais importantes dos Seres Humanos. É uma vantagem não só para si, mas também para a realização dos seus propósitos. Posto isto, e considerando o «poder» do diálogo e as perspetivas estratégicas de interação dialogal (Figura 1) que a plataforma *Debaqi*, sobre a qual nos iremos debruçar neste tema se posiciona. A inclusão de qualquer participante num espaço seguro onde se possa expressar livremente de forma a entender a realidade dos seus sentimentos sobre os quais a temática se enquadra e com o objetivo que os mesmos cheguem a um entendimento comum. Caso isso não seja possível, espera-se que pelo menos um dos princípios fundamentais do diálogo seja alcançado, a «aprendizagem».

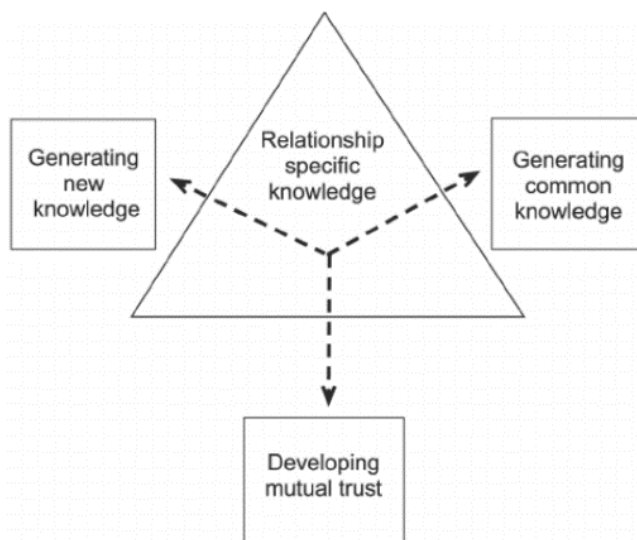


Figura 1: Perspetivas estratégicas de interação dialogal

Fonte: (Ballantyne, 2009)²

Resumo: A avaliação contínua do progresso alcançado em conjunto aprofunda as relações entre as pessoas e fortalece a sua capacidade de fazer correções e enfrentar novos problemas ou oportunidades à medida que estas surgem. A qualidade crítica do diálogo reside no facto de os participantes se reunirem num «espaço seguro» para entender o ponto de vista do outro, a fim de desenvolver uma nova compreensão dos problemas reconhecidos. O diálogo faculta um espaço no qual os participantes aprendem mais sobre si mesmos, sobre os outros e sobre os seus relacionamentos. A construção de redes circulares de diálogo e interações é fundamental para o nosso desenvolvimento social e afetivo, de maneira a desenvolver novos conhecimentos; estes numa escala maior, podem levar ao desenvolvimento de «conhecimento público» que se pode tornar numa mudança muito positiva e mais sustentável.

² [Consult. 01-09-2019]. Disponível em WWW:<URL:
https://www.researchgate.net/publication/215915361_Dialogue_and_its_role_in_the_development_of_relationship_specific_knowledge>

Nesta investigação, pretendemos realizar uma atividade educativa no âmbito da Biblioteca Escolar com objetivo de proceder a uma série de debates em sala virtual com grupos de 15 a 20 estudantes anónimos pertencentes a Escolas distintas, utilizando uma rede social segura e privada (*Debaqi*), desenvolvida para o estudo, semelhante a um grupo de Facebook.

Nos diversos debates serão introduzidos temas específicos da atualidade que foram previamente definidos pela equipa científica, bem como pela Rede de Bibliotecas Escolares e restantes Professores Bibliotecários. A realização de cada debate será sempre principiada por um teste de escolha múltipla sobre os conhecimentos e atitudes individuais de cada aluno sendo apenas desbloqueado o acesso ao *chat* com o seu devido preenchimento. Todos os debates serão moderados por um moderador que tem por objetivo manter a «ordem» nas salas virtuais, alertar os participantes que estão em formato de monólogo e não deixam espaço para os restantes membros participarem, como também propiciar uma ótima experiência a todos intervenientes. Para além destes objetivos o moderador também é responsável por encerrar a sessão, promovendo a introspeção e a realização de um questionário pós-debate.

De maneira a tornar-se uma atividade imersiva e semelhante às aplicações em voga, esta deverá permitir comentários, respostas, menções, reações e outros elementos multimédia de forma a enriquecer as intervenções dos alunos. Por fim, os dados de interação recolhidos são guardados de forma anónima para análise posterior de acordo com o Regulamento Geral de Proteção de Dados.

Organização

A presente dissertação está organizada da forma que a seguir se descreve. No capítulo 1 iremos elucidar sobre a evolução da relação Humana na *Internet* de forma a compreender o impacte que esta tem vindo a exercer na sociedade atual, como também futuras suposições alusivas ao relacionamento Humano. De seguida, no capítulo II irão ser abordados os conceitos que compreendem o domínio dos mais atuais modelos de arquiteturas de *software Web*, a sua história, vantagens, desvantagens e as suas falhas. Qualquer modelo apresentado no capítulo II é uma opção válida para desenvolver uma arquitetura para a aplicação *Debaqi*, mas de acordo com os objetivos traçados para esta aplicação existe apenas uma arquitetura que mais se adequa à necessidade. Motivado para estudar a temática exposta na introdução e capítulo I desta dissertação e suportado por uma vasta gama de tecnologias descritas no capítulo II, o capítulo III irá abordar toda a arquitetura desenvolvida para a aplicação *Debaqi*, o seu objetivo, funcionalidades, tecnologias e estrutura de dados. No capítulo IV irão ser apresentados os resultados obtidos no âmbito desta investigação e por fim na conclusão será ainda efetuada uma introspeção sobre a nossa experiência na concretização desta dissertação, bem como futuras propostas para reflexão.

CAPÍTULO I

A evolução da relação Humana na Internet

I – A EVOLUÇÃO DA RELAÇÃO HUMANA NA *INTERNET*

Há 20 anos atrás, muitos dos avanços tecnológicos que hoje ligam as pessoas e permitem a rápida circulação da informação ainda não estavam consolidados. Os computadores eram meramente utilizados como ferramentas de trabalho. O acesso à *Internet* ainda era restrito a instituições de ensino e pesquisa. Telemóveis eram artigos de luxo. E a possibilidade de fazer navegar na *Internet* aparelhos tão comuns atualmente, como os *smartphones*, era algo imaginado apenas por algumas mentes visionárias.

Até aos dias de hoje, ocorreram evoluções e revoluções que transformaram a comunicação e, como não poderia deixar de ser, o mundo. A democratização do uso da *Internet* e dos telemóveis, a digitalização da informação e a convergência dos media fizeram mais do que facilitar a comunicação e o fluxo de informações. Essas inovações criaram formas verdadeiramente novas de interação e de criação de conhecimento, como também provocaram mudanças profundas na relação entre produtores de media e os consumidores. A *Internet*, embora seja uma plataforma virtual, influencia diretamente o mundo físico e a própria humanidade, dando às pessoas a possibilidade de terem um papel ativo na comunicação global, através da interação entre o emissor e a mensagem em tempo real, reelaborando conteúdos e criando novos significados. Podemos mesmo afirmar que a *Internet* funciona como uma extensão do nosso mundo. Por meio dela inclusive é mais fácil exercer alguns aspetos da cidadania democrática:

- ❖ Permite-nos ter liberdade de escolha;
- ❖ Permite-nos expressar pensamentos livremente, talvez de uma maneira mais audível do que no mundo físico.

1.1. Dos *chats* às redes sociais

Desde o surgimento da *Internet* foram criadas diversas formas de interação virtual; isso demonstra o seu enorme potencial exploratório. No entanto, precisam de ser realizadas diversas melhorias e necessitam de ser ajustadas de forma a dinamizar a transmissão do conhecimento e transformá-la num ambiente ideal para a interação Humana. É ainda necessária mais segurança, mais acessibilidade para as pessoas com menor poder financeiro, mais fiscalização, mais legislação e maior controlo na contante disseminação automatizada de

propagandas de desinformação em larga escala: “We cannot accept this as a new normal.” (Jourová/King/Gabriel, 2019).

A interação coletiva digital mais atraente nos anos 90 eram os *chats* – conversas em salas virtuais segmentadas por cidades, idade, religião, namoro e outros interesses particulares. Também faziam sucesso os grupos virtuais do Yahoo e, posteriormente, os da Google. Estes grupos que ainda persistem vão-se extinguindo aos poucos. Os grupos do Yahoo e da Google já permitiam a criação de um perfil, mas, nesses grupos, as pessoas eram obrigadas a associar-se sempre a um tópico específico. Além disso, existia um moderador que filtrava as mensagens enviadas pelos membros nos fóruns de discussão. Só era possível comunicar livremente utilizando *email* ou programas como o ICQ – acrónimo proveniente das iniciais da expressão em inglês «*I Seek You*» – Mirc, MSN Messenger, etc.

Graças às redes sociais foi colocado um ponto final no interesse das pessoas pelos *chats*. As redes sociais ao oferecerem a cada indivíduo a possibilidade de personalização de um perfil pessoal acabaram por digitalizar a personalidade individual de cada cidadão físico no mundo virtual. Dessa forma, as pessoas passaram a assumir a sua individualidade e, de alguma forma, comprometeram-se mais com o que dizem e partilham. As redes sociais, especialmente o Facebook, foram integrando ao longo do tempo diversas funcionalidades de comunicação que antes eram utilizadas separadamente: *chats*, voz e vídeo; e ainda os grupos, comunidades, «*fan pages*», etc.; a tudo isto acresce a possibilidade de configuração de um perfil pessoal, o que fez com que as pessoas se sentissem mais conectadas num ambiente que espelha a própria sociedade, onde cada qual se relaciona e se expressa à sua maneira, revelando a própria personalidade publicamente.

1.2 Desconexão social

Para se conectarem as pessoas já não necessitam de estar obrigatoriamente num local fixo, o que se deve à integração geral da *Internet* nos dispositivos eletrónicos e *smartphones*. Por isso, vivemos simultaneamente num mundo físico e virtual, hoje duas realidades totalmente indissociáveis. Este «*blending*» apesar de dar a sensação que as pessoas estão mais conectadas também acabou por abalar os alicerces da necessidade humana de socialização presencial. Toda a cultura primitiva de mitos e rituais, sem dúvida, requereu a necessidade humana de socialização. Até ao século passado a vinculação entre o homem e o meio ambiente tem sido coincidente. No entanto, com o progresso, a tecnologia e a evolução das nossas necessidades

atuais têm prejudicado a estabilidade familiar e emocional das pessoas em quase todas as sociedades do planeta.

Ao observarmos com discernimento o comportamento dos nossos antecessores, concluímos que a construção de redes circulares de diálogo e interações foi fundamental para o nosso desenvolvimento social e afetivo. Não obstante, a crise atual de desconexão social que prolifera a nosso redor é um indicador que temos perdido, ou nos temos desviado, do legado cultural que nos incentivava a unir e interagir em comunidade para benefícios mútuos (Wiessner, 2014)³.

Por esta razão, se unirmos esforços e retomarmos esse tipo de conexão circular através de ferramentas de diálogo participativo que visam melhorar a conversa em grupos (*Debaqi*) e atividades que nos permitam experimentar essa sensação de unidade que vivenciaram as gerações passadas, poderemos falar de novas formas de nos relacionar e superar os problemas e desafios de um mundo que necessita de mudanças; fundamentalmente uma mudança na maneira de nos relacionarmos como Seres Humanos e com tudo o que nos rodeia.

³ [Consult. 12-10-2019]. Disponível em
WWW:<URL:<https://www.pnas.org/content/early/2014/09/16/1404212111.abstract>>

CAPÍTULO II

Arquitetura de aplicações Web

II – ARQUITETURA DE APLICAÇÕES WEB

Contrariamente a um passado recente, hoje em dia a *Internet* não são mais páginas estáticas e de tempos de carregamento demorados. Com o tempo, a *Internet* permitiu uma participação mais ativa por parte do utilizador, bem como o desenvolvimento de funcionalidades visualmente atrativas e poderosas por meio de aplicações *Web*. Uma aplicação *Web* assemelha-se a uma aplicação normal de computador, exceto pelo fato de que funciona na *Internet*, mais especificamente num navegador. Nos dias de hoje a maior parte das pessoas navegam diariamente na *Internet*, pelo que a maioria dos desenvolvimentos nesta área procuram beneficiar deste tipo de aplicações para atrair o maior número possível de utilizadores por meio de ofertas oportunas.

Antes de nos aventurarmos num projeto de desenvolvimento de uma aplicação *Web* devemos ter a consciência de que é extremamente importante escolher a arquitetura certa para o conseguir fazer, de modo a que esta possa atender a todas as necessidades científicas e de negócio, como também garantir uma ótima experiência ao utilizador. Existem principalmente três arquiteturas populares usadas na indústria, sendo elas o *Monolith*⁴, os *Microservices*⁵ e o *Serverless*⁶. Antes de passar ao seu desenvolvimento, a equipa deve em primeiro lugar escolher a arquitetura certa, bem como definir o modelo de componentes *Web* que a aplicação irá utilizar. Esta decisão é essencial e extremamente importante para o sucesso da aplicação *Web*.

Nos últimos anos, a popularidade da arquitetura *Serverless* tem vindo a aumentar. Existem vários serviços de computação *Serverless* disponíveis, tais como o *Azure Functions* e o *AWS Lambda*. No entanto, o *Monolith* é o mais forte do grupo, como já o era desde os seus primórdios. No entanto, assistimos a um interesse crescente pelos *Microservices*, no qual toda a aplicação está dividida em vários micro serviços, os quais comunicam por meio de API's. A arquitetura de uma aplicação *Web* deve então descrever as interações entre aplicações, base de dados e sistemas de «*middleware*». Esta visa garantir que várias aplicações funcionem simultaneamente e corretamente.

⁴ O conceito de *software* monolítico está em diferentes componentes de uma aplicação, combinados num único programa e plataforma. Esta arquitetura é descrita no ponto 2.5 Arquiteturas de aplicações *Web* a partir da página 31.

⁵ Os micro serviços são um tipo de arquitetura de *software* orientada a serviços que se concentra na construção de uma série de componentes autónomos que constituem uma determinada aplicação. Esta arquitetura é descrita no ponto 2.5 Arquiteturas de aplicações *Web* a partir da página 34.

⁶ A arquitetura *Serverless* é uma abordagem de computação em *Cloud* para criar e executar aplicações e serviços sem necessidade de gestão de infraestruturas. Esta arquitetura é descrita no ponto 2.5 Arquiteturas de aplicações *Web* a partir da página 38.

Para entender este conceito podemos utilizar um exemplo simples como a abertura de uma página *Web*:

- ❖ Assim que um determinado utilizador prime a tecla «*Enter*» após digitar um URL na barra de endereços do navegador, ele solicita aceder a esse endereço específico. O servidor envia então os ficheiros para o navegador como resposta ao pedido. O navegador, por sua vez, processa então esses ficheiros, o que resulta na visualização da página solicitada;
- ❖ Por fim, o utilizador pode interagir com a página *Web*. O mais importante a reter neste exemplo é o código processado pelo navegador;
- ❖ Esse código pode ou não ter instruções específicas que informam o navegador como responder face a diferentes tipos de utilizadores;
- ❖ Portanto, a arquitetura de uma aplicação *Web* não precisa lidar apenas com eficiência, mas também com confiabilidade, escalabilidade, segurança e robustez. Esta deve incluir todos os componentes, subcomponentes e trocas de informação com aplicações externas para toda a aplicação.

A definição de uma boa arquitetura de aplicações *Web* é completamente indispensável no mundo moderno, devido ao facto que grande parte do tráfego de rede mundial, assim como a maioria das aplicações e dispositivos, promovem a sua comunicação através da *Internet*.

2.1. Funcionamento de uma aplicação *Web*

O desenvolvimento de qualquer aplicação *Web* compreende dois conjuntos diferentes de programas que são executados separadamente e simultaneamente com o objetivo comum de trabalharem uniformemente para a entrega de soluções.

Normalmente, estes dois conjuntos de programas incluem o código do lado do cliente que funciona conforme as interações do utilizador e o código do servidor que funciona através de pedidos por protocolos HTTP(S). Os programadores têm a responsabilidade de decidir como as funções implementadas no código do lado do cliente e no lado do servidor irão funcionar em relação umas às outras.

- ❖ **Código do lado do cliente / *Front-end*** – Código interpretado pelo navegador e que responde a pedidos do utilizador;
 - Para escrever código do lado do cliente, normalmente são utilizadas tecnologias como HTML, CSS, JavaScript, TypeScript, etc.;
 - Este código é interpretado pelo navegador e de acordo com as instruções programadas e com as possíveis diferentes respostas vindas do servidor irá comporta-se de acordo com o instruído;
 - O código do lado do cliente comunica apenas por meio de pedidos HTTP, e está impossibilitado de aceder diretamente a dados ou ficheiros do servidor.

- ❖ **Código do lado do servidor / *Back-end*** – Código alojado num servidor que responde a pedidos HTTP.
 - Para escrever o código do lado do servidor, normalmente são utilizadas tecnologias como .NET, Java, JavaScript, Python, PHP, Ruby, etc.;
 - Este código normalmente vai ser responsável pela criação da página solicitada pelo utilizador, pelo transporte da informação a ser visualizada numa interface ou pela persistência de diferentes tipos de dados. Todo este processo é totalmente impercetível ao utilizador final.

A separação lógica destes dois conjuntos de programas supramencionados torna-se passível de representação esquemática através de um modelo conceptual ilustrado na Figura 2.

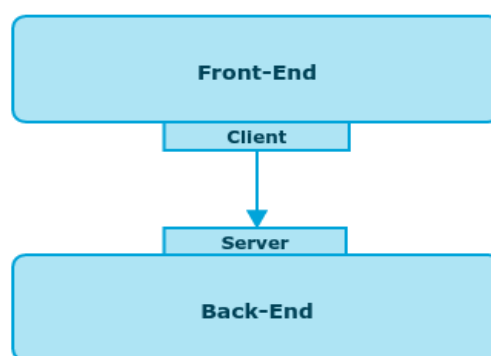


Figura 2: Modelo conceptual *back-end* e *front-end*

Fonte: Elaboração própria

Resumo: O desenvolvimento de qualquer aplicação *Web* inclui por norma dois conjuntos diferentes de *softwares* que são executados separadamente e simultaneamente com o objetivo comum de trabalharem uniformemente para a entrega de soluções. O *front-end* é a interface de interação com o utilizador, a forma como o conteúdo é apresentado, a estrutura da informação e a aplicação do *design*. Esta é desenvolvida em tecnologias como HTML, CSS, JavaScript, TypeScript, etc. O *back-end* por outro lado, é o *software* que está alojado no servidor e que contém as regras de negócio, faz a interação com as bases de dados, gere API's e outros recursos que o sistema pode disponibilizar. Para desenvolver o código do lado do servidor, podem ser utilizadas tecnologias como .NET, Java, JavaScript, Python, PHP, Ruby, etc.

2.2 Componentes de aplicações *Web*

As arquiteturas de aplicações *Web* compreendem vários componentes que são segregados em duas categorias: os componentes de UI/UX e os componentes estruturais. Os componentes de UI/UX fazem referência às páginas *Web* que têm uma função relacionada com a «apresentação». Estes componentes estão relacionados ao plano de estruturação de «*layouts*» de interfaces e à experiência de utilizador, não tendo qualquer papel na definição de uma arquitetura ou desenvolvimento. Lidam com «*logs* de atividades», painéis, notificações, configurações, estatísticas, etc.

Os componentes estruturais de uma aplicação *Web* referem-se à funcionalidade da aplicação com a qual o utilizador interage, ao controlo/lógica e ao armazenamento na base de dados. Por outras palavras, relaciona-se mais com os aspetos estruturais da arquitetura. Em suma, inclui primeiramente o navegador ou cliente *Web*, secundamente o servidor *Web* e por fim servidor de base de dados.

1. **Componente cliente** – O componente cliente é executado sobre um navegador e não requer qualquer necessidade de ajustes para determinados sistemas operativos ou dispositivos. Este componente resume-se à visualização de funcionalidades de uma aplicação *Web* com a qual o utilizador final interage;
2. **Componente servidor** – O componente do servidor pode ser desenvolvido com uma ou mais linguagens de programação e opera como «*hub* central». suportando toda a lógica de negócio da aplicação;
3. **Componente servidor de base de dados** – O componente de servidor de base de dados detém a lógica de negócio e os dados relevantes que são armazenados e geridos pelo servidor *Web*. É responsável pelo armazenamento, recuperação e fornecimento da informação.

2.3 Arquitetura de um servidor *Web*

A principal função de um servidor *Web* é aceder a pedidos de clientes, como navegadores, e aplicações móveis por meio de protocolos seguros HTTP. Os pedidos podem pertencer a recursos pretendidos para uma determinada página ou podem também estar relacionados a uma chamada de uma API REST.

O processo inicia-se com a ligação do computador cliente ao computador onde está alojado o servidor *Web*. Como estamos no domínio da *Internet* não é possível prever a que hora se dará essa mesma ligação, portanto os servidores *Web* necessitam de estar continuamente disponíveis. A partir deste momento é processado o pedido do cliente e conforme as restrições de segurança e a existência da informação solicitada o servidor «devolve» os dados.

Um exemplo muito básico pode ser descrito da seguinte forma: quando um navegador faz um pedido utilizando o protocolo HTTP sempre que necessitar de um ficheiro armazenado no servidor *Web*, no momento em que este pedido for intercetado pelo o servidor (*hardware*), o servidor HTTP (*software*) enviará o documento solitado, também via HTTP. Genericamente tudo o que se enquadre no conceito de ficheiro pode ser enviado como resultado de um pedido HTTP. O «*workflow*» descrito neste exemplo é ilustrado na Figura 3.

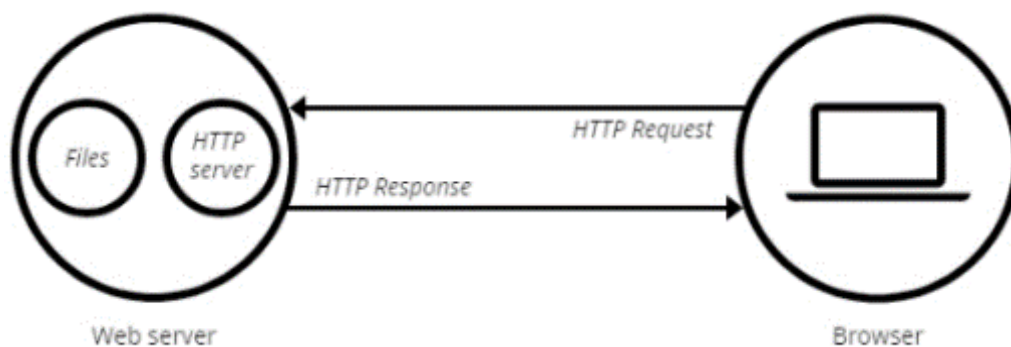


Figura 3: «*Workflow*» de um pedido HTTP

Fonte: https://developer.mozilla.org/pt-BR/docs/Learn/Common_questions/o_que_e_um_web_server

Resumo: Quando navegamos na *Internet* estamos constantemente a enviar e a receber pedidos (mensagens) de e para servidores que alojam as páginas a que tentamos aceder. Quando o servidor recebe um pedido por parte do navegador do cliente, posteriormente envia um «estado» e uma mensagem que pode conter a informação que foi pedida ou simplesmente uma mensagem de erro de que não é possível aceder ao recurso solicitado. O *Hypertext Transfer Protocol* (HTTP) é um protocolo utilizado para transmitir e receber informação através da *Internet*. É utilizado para que a informação possa ser transmitida entre servidores de forma rápida e fácil. De uma forma genérica tudo o que se enquadre no conceito de ficheiro é passível de ser enviado como resultado de um pedido HTTP.

Os servidores *Web* são intrínsecos ao funcionamento das aplicações *Web*, o que implica a necessidade de dar maior ênfase à arquitetura do servidor e à na sua capacidade física – armazenamento, memória, poder de computação e desempenho. Isto significa que a capacidade física é diretamente proporcional à complexidade da aplicação e à tecnologia utilizada, quanto maior mais recursos serão necessários.

A origem do conteúdo enviado pelo servidor *Web* numa resposta a um pedido HTTP pode ser estática se vier diretamente de um ficheiro já existente no servidor ou dinâmica se for criada dinamicamente por outro programa, «*script*» ou API chamado pelo servidor *Web*.

2.4 Modelos de componentes de aplicações *Web*

Para que uma aplicação *Web* seja estável e à prova de falhas é importante escolher um modelo que atenda às necessidades exigidas. Após esta reflexão o modelo de uma aplicação *Web* vai depender do número total de servidores e de bases de dados que serão necessários para cumprir com os requisitos. Os modelos que podem ser implementados pelas equipas de desenvolvimento são os seguintes:

- ❖ **Um servidor e uma base de dados** – É o modelo mais simples, arriscado e o menos fiável, no qual uma única base de dados faz parte de um único servidor *Web*. Se o servidor ficar indisponível, a aplicação *Web* também ficará. Este modelo pouco refinado pode adequar-se a alguns projetos piloto, «*Proof of Concept* – PoC» ou práticas particulares. Ainda assim, se existe a necessidade de uma aplicação fiável, esta certamente não será a melhor opção;
- ❖ **Vários servidores *Web* e uma base de dados** – Para dimensionar servidores *Web* horizontalmente é necessário que a base de dados seja executada a partir de uma máquina fisicamente separada do servidor *Web*. A ideia por detrás deste tipo de modelo é que o servidor *Web* não armazene qualquer dado. Quando o servidor *Web* obtém informação de um cliente, este processa essa mesma informação, guarda-a numa base de dados gerida fora do servidor e «esquece-a»; também conhecido como arquitetura sem estado.
Neste modelo são necessários pelo menos 2 servidores *Web* com o intuito de evitar quaisquer falhas na disponibilização do serviço, ou seja, quando um dos servidores *Web* fica indisponível por alguma razão, o outro assume o controlo.

Todas os novos pedidos serão redirecionados automaticamente para o novo servidor por um *proxy* e a aplicação *Web* continuará a ser executada na normalidade. Portanto, a fiabilidade é bastante superior em comparação com o modelo de servidor único de base de dados inerente. No entanto, se a base de dados «parar» a aplicação *Web* também acabará por o fazer;

- ❖ **Vários servidores *Web* e várias bases de dados** – É o modelo mais eficiente, porque nem os servidores *Web*, nem as bases de dados têm um único ponto de falha. Existem duas opções para esse tipo de modelo: armazenar os mesmos dados em todas as bases de dados ou distribuí-los igualmente entre elas.

No primeiro caso, geralmente, não são necessárias mais de duas bases de dados; quando uma está inativa a outra pode substituí-la, sem qualquer perda. Como os dados não são replicados na segunda opção, alguns dados podem ficar temporariamente inacessíveis se uma das muitas bases de dados ficar indisponível. A normalização do DBMS é utilizada, no entanto, em ambos os cenários.

Quando a escala é grande, ou seja, mais de 5 servidores *Web*, bases de dados ou ambos é aconselhada a instalação de balanceadores de carga.

2.5 Arquiteturas de aplicações *Web*

Uma arquitetura para uma aplicação *Web* é o padrão de interação entre os vários componentes constituintes da aplicação. O tipo de arquitetura vai depender de como a lógica da aplicação é distribuída entre os lados do cliente (*front-end*), do servidor *Web* (*back-end*) e do servidor de base de dados. Portanto, quando pretendemos desenvolver um novo produto *Web* devemos optar pela arquitetura mais apropriada à sua execução, de forma a mitigar o risco inerente a uma aplicação mal estruturada. Esta reflexão é um passo crucial para o sucesso da mesma.

Como referido anteriormente existem três tipos principais de arquiteturas para aplicações *Web*: a *Monolith*, os *Microservices* e o *Serverless*. De acordo com o *Cambridge Dictionary*, *Monolith* significa: “a large block of stone standing by itself that was put up by people in ancient times”. Embora este termo possa ter diversas utilizações, hoje em dia a imagem permanece a mesma. Na engenharia de *software*, um padrão monolítico refere-se a uma única unidade não divisível. O conceito de *software* monolítico está em diferentes

componentes de uma aplicação, combinados num único programa e plataforma, como ilustra a Figura 4. Geralmente, uma aplicação monolítica tem por base a implementação de código massivo, constituído por uma base de dados, uma interface de utilizador (*front-end*) e pela lógica do lado do servidor (*back-end*), onde todas estas partes do *software* estão unidas e todas as suas funções são geridas num único sítio. Isto significa que se os programadores fizerem alterações ou atualizações ao código vão necessitar de fazer «*build*» e «*deploy*» da aplicação inteira sempre que tal acontecer.

Uma arquitetura monolítica é mais confortável para equipas pequenas trabalharem, proporcionando-lhes menos carga operacional, motivo pelo qual muitas «*startups*» escolhem este padrão para criarem as suas primeiras aplicações. Os componentes de um *software* monolítico estão intrinsecamente ligados e dependentes entre si, o que ajuda o *software* a ser independente. Esta arquitetura é uma solução tradicional para a criação de aplicações *Web*, no entanto começa a ser considerada desatualizada aos olhos de muitos programadores. É igualmente interessante reparar que a maioria das aplicações corporativas foram desenvolvidas totalmente baseadas em estruturas monolíticas devido a serem muito previsíveis e restritas em termos de interações externas.

“Even though we had had these positive experiences of using microservices at Google, we [at Scaylr] went for a monolith route because having one monolithic server means less work for us as two engineers.” (Lumetta, 2018)⁷

De acordo com (Lumetta, 2018)⁸ existirá sempre uma série de vantagens e desvantagens com as quais se terá de lidar ao implementar qualquer uma das três arquiteturas supramencionadas. Abaixo serão descritos os pontos mais e menos positivos que cada uma das arquiteturas.

Vantagens de uma arquitetura monolítica

- ❖ **Desenvolvimento e integração simples** – Existem inúmeras ferramentas que podem ser integradas para facilitar o desenvolvimento. Além disso, todas as ações são executadas num único repositório, o que facilita a sua implementação;
- ❖ **Menos preocupações transversais** – A maioria das aplicações depende de uma grande quantidade de configurações transversais, como auditoria, «*logging*», etc. As aplicações monolíticas incorporam essas configurações com maior facilidade

⁷ [Consult. 13-09-2019]. Disponível em WWW:<URL:<https://blogs.mulesoft.com/dev/tech-ramblings/why-the-monolith-isnt-dead/>>

⁸ Informação igualmente disponível na hiperligação presente na nota de rodapé número 7.

por terem uma base de código comum. É mais fácil interligar estes componentes a estas configurações quando tudo é executado na mesma aplicação;

- ❖ **Melhor performance** – Se criadas corretamente, as aplicações monolíticas geralmente têm melhor performance que as aplicações baseadas em micro serviços. Uma aplicação com uma arquitetura de micro serviços pode necessitar de executar 30 chamadas a API's para 30 micro serviços diferentes para carregar um ecrã, o que obviamente resultará num desempenho mais lento. As aplicações monolíticas, por sua vez, permitem uma comunicação mais rápida entre os componentes do *software* devido ao código e memória partilhada.

Desvantagens de uma arquitetura monolítica

- ❖ **O código começa a ficar mais complicado de gerir ao longo do tempo** – Com o decorrer do tempo, a maioria dos produtos tendem a aumentar o seu desenvolvimento e âmbito tornando consequentemente a sua estrutura mais difusa. O código começa a parecer grande e fica difícil de entender e modificar, especialmente para novos programadores que sejam alocados ao projeto. Também fica cada vez mais difícil encontrar impactos e dependências. Com um número crescente de código a ser inserido e alteado no projeto, a qualidade diminui e o ambiente de desenvolvimento fica sobrecarregado;
- ❖ **Dificuldade em adotar novas tecnologias** – Se houver necessidade de adicionar uma nova tecnologia à solução, os programadores podem enfrentar algumas barreiras à entrada. Adicionar uma nova tecnologia significa reescrever toda a aplicação, o que pode ser extremamente dispendioso e demorado;
- ❖ **Agilidade limitada** – Em aplicações monolíticas, qualquer pequena atualização ao código requer um completo «*redployment*». Assim, toda a equipa necessita de aguardar até que o «*deploy*» esteja concluído. Quando existem várias equipas a trabalhar no mesmo projeto, a agilidade pode tornar-se bastante reduzida. Este «*redployment*» também faz com que este tipo de arquitetura não seja ideal para «*continuous delivery*», ou seja, sempre que exista uma alteração no código que esteja em produção, o negócio não pode parar por completo aguardando que o novo código seja integrado no servidor. Se estivermos a falar em aplicações muito grandes pode significar «*downtimes*» significativos.

Em suma, o modelo monolítico não está de todo desatualizado e ainda funciona perfeitamente em alguns cenários. Grandes empresas como a Etsy permanecem monolíticas, apesar da crescente popularidade de arquiteturas em micro serviços. A arquitetura monolítica pode ser benéfica em equipas que se encontrem num estado inicial/júnior na criação de um produto experimental e que sejam inexperientes no desenvolvimento em micro serviços. O *Monolith* é perfeito para «startups» que precisam de desenvolver e lançar um produto no mercado o mais rapidamente possível. No entanto, não é um padrão perfeito devido a alguns dos problemas supramencionados.

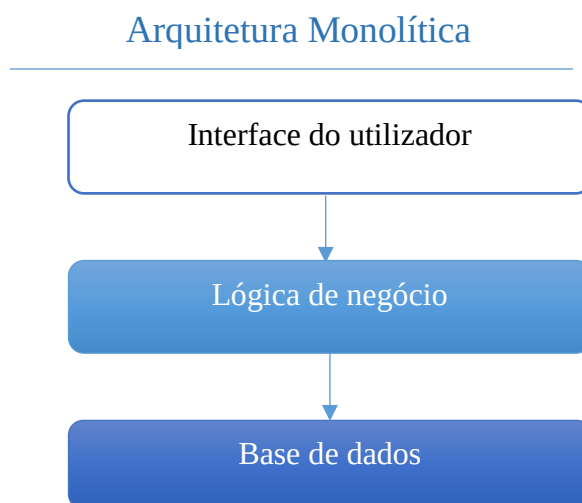


Figura 4: Modelo conceptual de uma arquitetura monolítica

Fonte: Adaptado de (Barashkov, 2018)

Resumo: Em engenharia de *software*, um padrão monolítico refere-se a uma única unidade não divisível. O conceito de *software* monolítico está em diferentes componentes de uma *aplicação*, combinados num único programa e plataforma. Nos sistemas com arquiteturas monolíticas, a aplicação é concentrada numa única base de código, que se torna extensa à medida que a aplicação cresce e onde os seus componentes estão intrinsecamente ligados e dependentes entre si, o que ajuda o *software* a ser independente. Esta arquitetura não está desatualizada e pode ser utilizada em várias aplicações. As equipas que se encontrem num estado inicial/júnior na criação de um produto experimental e que sejam inexperientes no desenvolvimento em micro serviços podem beneficiar deste tipo de arquitetura.

Imaginemos «cortar» uma aplicação aos pedaços e executá-la como um conjunto de partes mais pequenas em vez de um todo conjunto monolítico. Esta é de uma forma básica o conceito de uma arquitetura em micro serviços. Apesar do nome, os micro serviços não têm de ser necessariamente pequenos. O que os torna «micro» e o facto de apenas gerirem serviços referentes a um único conceito de negócio de uma aplicação maior.

Os micro serviços são um tipo de arquitetura de *software* orientada a serviços que se concentra na construção de uma série de componentes autónomos que compõem uma determinada aplicação, como ilustra a Figura 5. Ao contrário das aplicações monolíticas,

criadas como uma única unidade não divisível, as aplicações em micro serviços consistem em vários componentes independentes que se encontram disponíveis por meio de API's (Krause, 2015).

Uma abordagem em micro serviços concentra-se principalmente nas prioridades e requisitos de negócio, enquanto a abordagem monolítica é organizada em torno de camadas tecnológicas, como a interface do utilizador e a base de dados.

Na perspetiva do utilizador, uma aplicação criada em micro serviços possui uma única interface e deve funcionar da mesma forma que uma aplicação desenhada em «*stack*». No entanto, nos bastidores, cada micro serviço possui a sua própria base de dados e é executado separadamente e individualmente da restante aplicação. Além disso, os micro serviços podem ser desenvolvidos para a mesma aplicação em linguagem de programação e bases de dados diferentes, como pode também usar bibliotecas totalmente distintas.

Os micro serviços tornaram-se uma tendência nos últimos anos à medida que cada vez mais as empresas se tornam ágeis e avançam para o *Dev(Sec)Ops*. Existem diversos exemplos de empresas que evoluíram de uma abordagem monolítica para micro serviços. Entre os mais destacados estão a Netflix, a Amazon, o Twitter, o eBay e a PayPal.

“Microservices are important simply because they add unique value in a way of simplification of complexity in systems. By breaking apart your system or application into many smaller parts, you show ways of reducing duplication, increasing cohesion and lowering your coupling between parts, thus making your overall system parts easier to understand, more scalable, and easier to change.” (Krause, 2015)

Quando avançamos para a definição da arquitetura de um projeto devemos sempre determinar se os micro serviços são de facto um padrão adequado para a execução do mesmo.

Prós de uma arquitetura em micro serviços

- ❖ **Fácil de desenvolver, testar e integrar** – A maior vantagem dos micro serviços em relação às restantes arquiteturas reside no facto de estes poderem ser desenvolvidos, testados e integrados de forma totalmente independente. Como a quantidade de código para ser integrado é pequena facilita e acelera o desenvolvimento e a «*release*». Além disso, a «*release*» de um serviço não é limitada pelo lançamento de outro serviço que ainda não esteja concluído. Outra grande vantagem é diminuição dos riscos de integração, pois os programadores integram apenas frações do *software* e não a aplicação inteira;

- ❖ **Maior agilidade** – Com os micro serviços, várias equipas podem trabalhar nos seus serviços de forma independente e rápida. Cada parte individual de uma aplicação pode ser construída de forma independente devido à dissociação dos componentes do micro serviço. Conseguimos demonstrar um aumento de agilidade com o seguinte exemplo: uma equipa de 100 pessoas a trabalhar em toda a aplicação (como na abordagem monolítica) ou 10 equipas de 10 pessoas a trabalharem no desenvolvimento de serviços diferentes para a mesma aplicação.

A maior agilidade provem da capacidade que os programadores têm de atualizar os componentes do sistema sem interromper a execução da aplicação. Além disso, a agilidade fornece um processo de integração mais seguro e um menor tempo de inatividade. Novos recursos podem ser integrados conforme o necessário, sem aguardar a integração de toda a aplicação;

- ❖ **Possibilidade de escalar horizontalmente** – Escalar verticalmente (executar o mesmo *software*, mas em máquinas mais potentes) pode ser limitado pela capacidade de cada serviço. Escalar horizontalmente (criação de mais serviços na mesma «*pool*») não é limitado e pode ser executado dinamicamente com micro serviços. Além disso, escalar horizontalmente pode ser realizado de forma totalmente automatizada;
- ❖ **Melhor organização** – As arquiteturas em micro serviços encontram-se geralmente melhor organizadas e estruturadas. Cada micro serviço tem uma «responsabilidade» muito específica e não tem que se preocupar com o propósito dos restantes componentes;
- ❖ **Performance** – Sob circunstâncias ideais, os micro serviços podem também fornecer vantagens de desempenho, dependendo do modo como estão organizados. É possível isolar serviços que são usados com maior regularidade e escalá-los independentemente da restante aplicação;
- ❖ **Separação/Integração** – Os serviços ao serem independentes entre si também são mais fáceis de se redefinirem e reconfigurarem para atenderem propósitos diferentes da aplicação. Também permitem uma integração rápida e independente de «peças» individuais de *software* dentro de um sistema maior e integrado.

Contras de uma arquitetura em micro serviços

- ❖ **Complexidade** – A maior desvantagem dos micro serviços reside na sua complexidade. Dividir uma aplicação em micro serviços independentes implica mais artefactos para gerir. Este tipo de arquitetura requer um planeamento cuidadoso, um enorme esforço, recursos e bastante conhecimento por parte da equipa de desenvolvimento. Os motivos que levam à sua alta complexidade são os seguintes:
 - Maior necessidade de automatização, pois todos os serviços devem ser testados e monitorizados;
 - As ferramentas disponíveis não funcionam com dependências entre serviços;
 - A consistência dos dados e a gestão de transações são mais difíceis, pois cada serviço possui a sua própria base de dados.
- ❖ **Preocupações com segurança** – Numa aplicação desenvolvida em micro serviços, cada funcionalidade comunica externamente por meio de uma API, isto aumenta a probabilidade de ocorrência de ataques. Esses ataques têm maior probabilidade de ocorrer se não forem implementadas as medidas de segurança adequadas durante o desenvolvimento da aplicação;
- ❖ **Diferentes linguagens de programação** – A capacidade de escolher diferentes linguagens de programação é simultaneamente boa e má. O uso de diferentes linguagens dificulta a integração, além disso, é mais difícil efetuar alterações nos programadores entre as diferentes fases de desenvolvimento do produto quando cada serviço é desenvolvido em tecnologias diferentes;
- ❖ **Maior sobrecarga operacional** – Os micro serviços são frequentemente integrados nas suas próprias máquinas virtuais, causando consequentemente um aumento no trabalho de manipulação de ambientes virtuais. Estas tarefas por norma são automatizadas com ferramentas de gestão de *containers*.

Em suma, os micro serviços são bons, mas não para todos os tipos de aplicações. Este padrão funciona muito bem para aplicações em constante desenvolvimento e sistemas complexos. Devemos considerar escolher uma arquitetura em micro serviços quando tivermos várias equipas experientes e quando a aplicação for suficientemente complexa para transformá-

lo em serviços. Quando uma aplicação é grande, e precisa de ser flexível e escalável, os micro serviços são sem dúvida benéficos.

É essencial entender se esta arquitetura é a melhor opção para o projeto. O seu contexto, avaliado com base nas considerações supramencionadas, é a chave para decidir se devemos usar uma arquitetura monolítica ou de micro serviços.

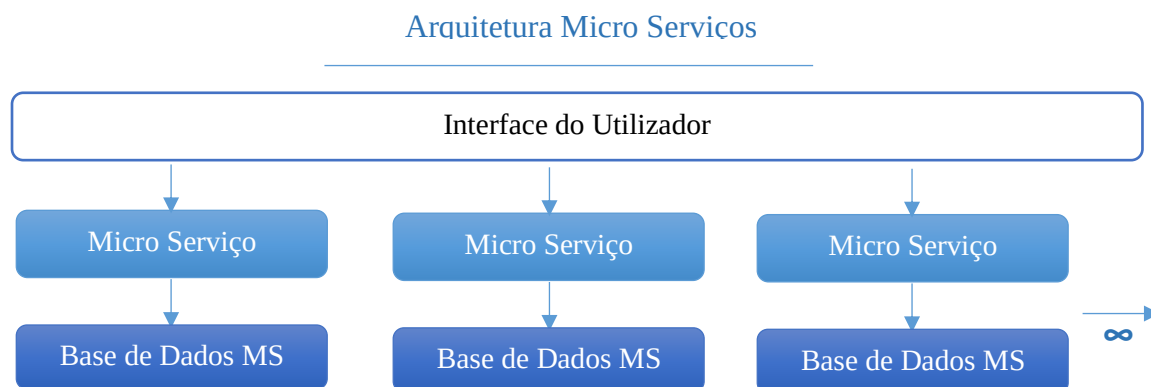


Figura 5: Modelo conceptual de uma arquitetura em micro serviços

Fonte: Adaptado de (Barashkov, 2018)

Resumo: Uma arquitetura em micro serviços visa uma abordagem para desenvolver uma única aplicação através de um conjunto de serviços, cada um desenvolvido de forma independente e que comunica por meio da rede local ou da *Internet*, através de uma REST API. Estes serviços estão divididos de acordo com as necessidades de negócio e podem ser «*deployed*» de forma independente uns dos outros. Uma arquitetura de micro serviços tem alguns benefícios em relação às arquiteturas monolíticas: 1) cada micro serviço é relativamente pequeno, e o código é de fácil compreensão por parte dos programadores. 2) o código pode ser implementado em separado, pois cada micro serviço tem a sua própria base de dados e é independente dos restantes. 3) cada micro serviço pode ser escalado horizontalmente (aumentando o número de instâncias) 4) a arquitetura elimina um compromisso de longo prazo com uma tecnologia ou «*stack*» de *software* específica.

A arquitetura *Serverless* visa uma abordagem de computação em *Cloud* que permite desenvolver e executar aplicações e serviços sem necessidade de gestão de infraestruturas. Para compreendermos o conceito inerente a esta arquitetura é necessário entender como surgiu a aceitação por este padrão de desenvolvimento. Como tal, nas próximas páginas iremos elucidar sobre os seus paradigmas e constituintes, tal como ilustrados na Figura 6..

Em aplicações *Serverless*, a execução do código é gerida por um servidor, permitindo aos programadores desenvolverem o código sem se preocuparem com a manutenção e disponibilidade do servidor. Por se chamar *Serverless* não significa de facto «sem servidor». As aplicações continuam a ser executadas em servidores, mas num serviço *Cloud* de terceiros como a AWS ou Azure, e estes assumem total responsabilidade pelos seus servidores.

A arquitetura *Serverless* elimina a necessidade de recursos extras como a escalabilidade de aplicações e a manutenção de servidores e sistemas de base de dados. Esta total abstração permite desenvolver aplicações em qualquer linguagem de programação e executá-las. Todas as restantes preocupações estarão ao cuidado do «*cloud provider*» isto regidas pelos limites definidos nos pacotes comerciais que foram adquiridos para a execução da aplicação. Isto aumenta a possibilidade de escalabilidade da aplicação ao máximo possível, sem que seja detetado algum problema. Portanto, qualquer novo requisito pode ser adicionado imediatamente e qualquer tecnologia pode ser utilizada para o desenvolvimento.

O uso de uma infraestrutura híbrida também pode ser uma opção a ponderar. Uma arquitetura híbrida consiste essencialmente num projeto estruturado em arquiteturas de micro serviços e *Serverless*, pois ambas são extremamente flexíveis. No entanto, este modelo não é muito recomendado, pois aumenta a complexidade e o esforço de manter a aplicação.

Uma arquitetura *Serverless* inclui dois conceitos principais:

- ❖ **BaaS (*Back-end as a Service*)** – Com a evolução dos «*cloud providers*», surgiu uma oferta de serviços de uma infraestrutura de *back-end* já construída e distribuída, conhecido como BaaS. Desta forma, os programadores podem concentrar-se em implementar as regras de negócio definidas nos requisitos, abstraindo-se da necessidade de pensar e implementar uma infraestrutura;
- ❖ **FaaS (*Function as a Service*)** – Modelo de computação *Cloud* que permite que os programadores enviem «pedaços» de *software* para a *Cloud* e que essas peças/funções sejam executadas independentemente.

Com todo este trabalho periférico mitigado, os programadores passam a ter mais tempo para se concentrarem no produto e se dedicarem às suas funções primárias (programar), entregando muito mais *software* no mesmo período de tempo.

Prós de uma arquitetura *Serverless*

- ❖ **Integração fácil** – Em aplicações *Serverless*, os programadores não necessitam de se preocupar com a infraestrutura, o que permite que se concentrem no próprio código. Esta arquitetura possibilita gerir uma aplicação com extrema rapidez, pois a integração leva apenas horas ou dias, comparativamente com o número de dias ou semanas implicado numa abordagem tradicional;

- ❖ **Redução de custos** – Cada vez que uma função é executada é cobrado um valor monetário apenas pela memória e processamento consumido pelo CPU. Sendo assim, não é cobrado tempo de inatividade, um problema recorrente hoje em dia nas máquinas virtuais;
- ❖ **Escalabilidade** – Muitos empresários querem que as suas aplicações se tornem influentes e escaláveis, como o Google ou o Facebook. A computação *Cloud* torna a escalabilidade automática e contínua. A aplicação será escalada automaticamente conforme a sua carga ou número de utilizadores, sem afetar o desempenho. Aplicações *Serverless* podem lidar com um grande número de pedidos, enquanto uma aplicação tradicional ficará sobrecarregada pelo aumento repentino de pedidos.

Contras de uma arquitetura *Serverless*

- ❖ **Dependência de fornecedor** – Situação em que um determinado fornecedor tem o controlo total das operações da organização ou projeto. Consequentemente, as alterações na lógica do negócio são limitadas e a migração de um fornecedor para outro pode tornar-se um desafio;
- ❖ **Inadequado para operações de longo prazo** – Aplicações desta natureza são boas para processos curtos em tempo real, mas se uma tarefa levar mais de cinco minutos, uma aplicação *Serverless* irá requer o desenvolvimento de uma função FaaS adicional.

Em suma uma arquitetura de *software Serverless* é benéfica para a realização de tarefas únicas e processos auxiliares. Funciona muito bem para aplicações que têm um número massivo de clientes diários, aplicações que estão a crescer rapidamente e precisam de ter escalabilidade infinita ou empresas que não desejam gerir ou dar suporte a servidores, bem como ao *hardware* para o qual desenvolveram a aplicação.

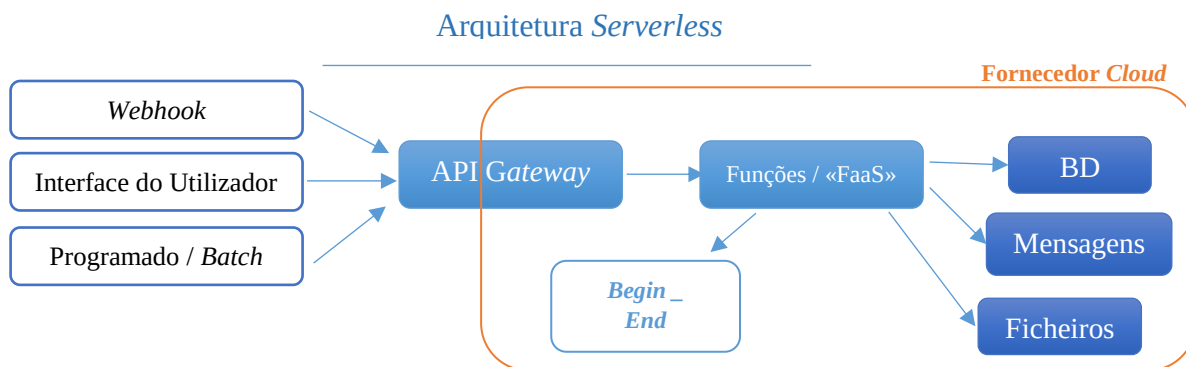


Figura 6: Modelo conceptual de uma arquitetura *Serverless*

Fonte: Adaptado de (Roberts, 2018)

Resumo: A arquitetura *Serverless* visa uma abordagem de computação em *Cloud* que permite desenvolver e executar aplicações e serviços sem necessidade de gestão de infraestruturas. Em aplicações *Serverless*, a execução do código é gerida por um servidor, permitindo aos programadores desenvolverem o código sem se preocuparem com a manutenção e disponibilidade do servidor. Por se chamar *Serverless* não significa de facto «sem servidor». As aplicações continuam a ser executadas em servidores, mas num serviço *Cloud* de terceiros como a AWS ou Azure, e estes assumem total responsabilidade pelos seus servidores eliminando a necessidade de recursos extras como a escalabilidade de aplicações e manutenção de sistemas de base de dados. A lógica da aplicação é dividida em funções, onde cada uma delas é desenvolvida individualmente através de uma plataforma de *function as a service* (FaaS) que se encontra continuamente disponível, aguardando o evento que a faça despoletar. Assim, a plataforma de FaaS é configurada para responder a um pedido, executando o código necessário e, por fim, encerrar a função solicitada. A plataforma BaaS por outro lado visa uma oferta de serviços de uma infraestrutura de *back-end* já construída e distribuída. Em nenhum desses dois conceitos da arquitetura *Serverless* (BaaS e FaaS), o cliente gere a infraestrutura necessária para manter a aplicação.

Os tipos de arquitetura para aplicações *Web* e os modelos de componentes têm vindo a evoluir juntamente com a própria *Web*. Enquanto as antigas estruturas «*legacy*» e modelos de componentes básicos apareceram nos tempos da *Web 1.0*, os padrões modernos de arquiteturas e os modelos de componentes escaláveis foram revolucionários nas eras da *Web 2.0* e *3.0*.

Das três principais arquiteturas, tudo vai depender dos requisitos inerentes aos projetos. Estas arquiteturas são completamente distintas na sua abordagem, portanto, decidir a mesma não é um problema. Se a aplicação for simples, o monolítico é melhor. Se a aplicação for divisível em várias partes independentes, existindo a necessidade de disponibilizar serviços para o exterior por meio de API's, os micro serviços são melhores.

Se estivermos prontos para criar uma «*startup*» e quisermos lançar um produto para o mercado o mais rapidamente possível seria recomendável optar pelo monolítico. Isto porque o desenvolvimento da arquitetura monolítica é simples e consome menos tempo. Não há complexidade nos sistemas e o tempo de desenvolvimento é mais reduzido. Mas se existir a necessidade de adicionar novos desenvolvimentos de *software* de uma forma continua ao longo do tempo, o monolítico não é uma boa opção. Pode existir a necessidade de migrar a aplicação completa para micro serviços de modo a evitar situações de «*bottleneck*».

No entanto, quando decidimos optar por uma arquitetura *Serverless* essa decisão é mais baseada pela integração do que no desenvolvimento e esse é o motivo pelo qual se tem tornado tão popular ultimamente. A arquitetura *Serverless* elimina a dor de cabeça da disponibilidade, escalabilidade e manutenção dos servidores, pois estes são geridos automaticamente pelo fornecedor de *Cloud*. Assim, não existe a necessidade de criar uma equipa de *DevOps*. De fato, todos os serviços alojados em *Cloud* são essencialmente arquitetura *Serverless*. Na arquitetura *Serverless* somente é possível escalar horizontalmente, o que pode levar algumas empresas a optar pelo monolítico.

A escolha de um modelo e da arquitetura pode determinar o quão ágil, robusta, segura e rápida será a aplicação *Web*. Portanto, antes de iniciarmos qualquer projeto de desenvolvimento, devemos sempre analisar de perto os requisitos do projeto e avaliar todas as opções disponíveis.

2.6 Dicas para o desenvolvimento de aplicações *Web*

Qualquer aplicação *Web* que se encontra em produção não pode ser rotulada como «a melhor». Os parâmetros de avaliação da excelência de uma aplicação *Web* vão muito além da sua capacidade de computação.

Para garantir que uma aplicação *Web* cumpre o seu desempenho máximo esta deve obedecer aos seguintes requisitos:

- ❖ Evitar *crashes* frequentes;
- ❖ Facilidade em escalar consoante a necessidade momentânea;
- ❖ Ser fácil de utilizar;
- ❖ Ter uma boa UX/UI;
- ❖ Ter um tempo de resposta rápido;
- ❖ Ter integrações automáticas;
- ❖ Existência de configurações de «*logging*» para erros;
- ❖ Não ter um único ponto de falha;
- ❖ Resolver uma pesquisa de maneira consistente e uniforme;
- ❖ Implementar e suportar os mais recentes padrões e tecnologias;
- ❖ Utilizar medidas de segurança reforçadas para diminuir a possibilidade de ataques;

Por seu turno, a equipa desenvolvimento deve seguir um conjunto de boas práticas, de forma a contribuir igualmente para a operacionalidade dessa mesma aplicação:

- ❖ Ter uma boa documentação sobre a aplicação;
- ❖ Rever o progresso do projeto de uma forma regular;
- ❖ Requisitos claros;
- ❖ Foco e partilhar o conhecimento entre as equipas;
- ❖ Comunicação ativa entre todos os intervenientes;
- ❖ Realizar ações de formação e pesquisa;

2.7 WebSockets

Os *WebSockets* representam uma evolução há muito esperada no Universo *Web*. Esta tecnologia permite estabelecer uma ligação «*socket*» TCP de longa duração entre um cliente e um servidor, que possibilita assim a troca instantânea de mensagens bidirecionais. A ligação *WebSocket* é estabelecida por um processo conhecido como «*handshake*».

Os *WebSockets* ao manterem continuamente uma ligação aberta entre o cliente e o servidor facilitam o envio de informações ao cliente mesmo na ausência de um pedido explícito do mesmo, tal como ilustra a Figura 7. O cliente continua a poder fazer pedidos HTTP normalmente; no entanto, como existe uma comunicação constante, no caso da existência de novos dados, estes requerem processamento em tempo real pelo lado do cliente; é deste modo que serviços como o *e-mail*, *chats*, *IoT* e jogos *online* funcionam. O «*feed*» é atualizado instantaneamente, sem que o utilizador tenha a necessidade de atualizar a página manualmente. Este tipo de comunicação entre o servidor e o cliente é chamado de «*full-duplex*».

Atualmente, esta tecnologia para além de ser adotada por muitos programadores que necessitam desenvolver aplicações com interação em tempo real, também foi considerada como uma «*Web next big thing*» (Lengstorf, 2013). Os *WebSockets* não tornam o *AJAX* obsoleto, mas substituem o HTTP «*long polling*» e HTTP «*streaming*» representando a solução para uma verdadeira funcionalidade em tempo real. Devemos continuar a usar o *AJAX* para fazer pedidos *Web* de curta duração, mas sempre que exista a necessidade de uma ligação de baixa latência e em tempo real entre o cliente e o servidor optármos por uma tecnologia de *Websockets* será a decisão mais acertada. Resumidamente no Quadro 1 encontram-se enunciadas as principais diferenças técnicas entre pedidos HTTP e *Websockets*.

Temos de ter em mente que os *Websockets* podem exigir repensar a forma como criamos as aplicações do lado do servidor e mudar o foco para tecnologias que lidam com filas de eventos. A única razão pela qual podemos considerar a não utilização de *WebSockets* para o desenvolvimento de uma aplicação *Web* é devido à sua compatibilidade atual; 95%⁹ dos possíveis utilizadores têm um navegador compatível com esta tecnologia.

Concluindo, é importante referir que esta tecnologia desempenhou um papel crucial, de modo a alcançar uma das principais etapas do desenvolvimento da aplicação. Tornou viável a criação de um espaço virtual que possibilita a realização de inúmeras conversas em simultâneo, cada uma das quais igualmente habilitadas a suportar um elevado número de participantes.

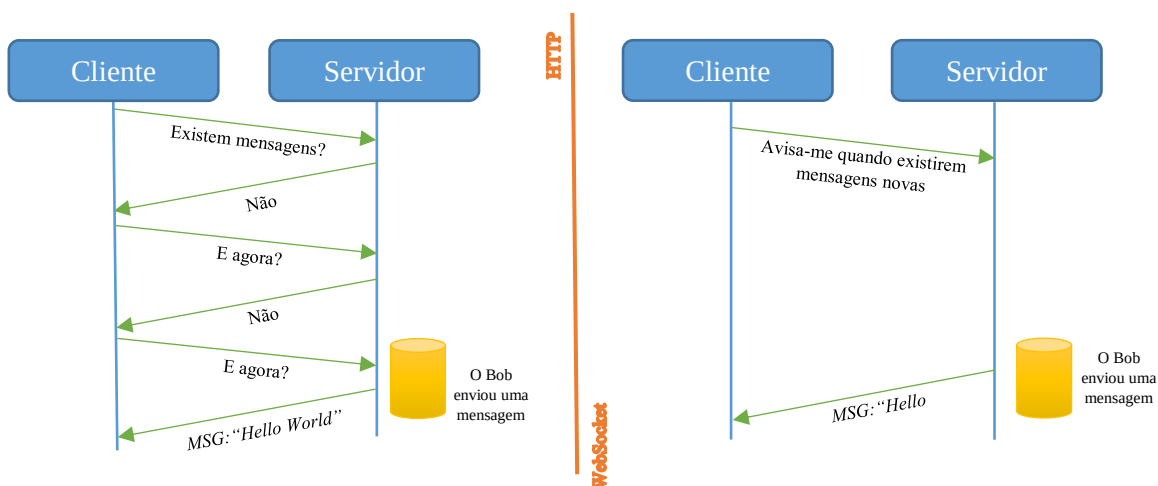


Figura 7: Comparação do «workflow» entre um pedido HTTP e uma ligação por *WebSocket*

Fonte: Elaboração própria

Resumo: Os *WebSockets* são uma tecnologia que permite estabelecer uma ligação «socket» TCP de longa duração entre um cliente e um servidor, que possibilita a troca instantânea de mensagens bidirecionais. Esta ligação é estabelecida por um processo conhecido como «handshake». O cliente, por sua vez, continua a poder realizar pedidos HTTP normalmente. No entanto, como existe uma comunicação constante, no caso da existência de novos dados, estes requerem processamento em tempo real pelo lado do cliente, ou seja, o «feed» é atualizado instantaneamente, sem que o utilizador tenha a necessidade de atualizar a página manualmente. Este tipo de comunicação entre o servidor e o cliente é chamado de «full-duplex» e não torna o AJAX obsoleto, mas substitui o HTTP «long polling» e o HTTP «streaming».

⁹ [Consult. 05-12-2019]. Disponível em WWW:<URL:<https://caniuse.com/#feat=websockets>>

	HTTP	WebSocket
«Duplex»	<i>Half</i>	<i>Full</i>
Direção	Cliente / Servidor + <i>Push</i>	Bidirecional
Serviço de «push»	Não suportado nativamente. São usadas técnicas como « <i>client polling</i> » ou « <i>streaming</i> ».	Característica principal
Carga	Carga moderada por cada pedido e ligação	Carga moderada para estabelecer e manter a ligação, depois mínima por mensagem.
Intermediação / « <i>Edge caching</i> »	Característica principal	Não é possível
Clientes suportados	Amplo suporte	Linguagens e clientes modernos
Headers	Comprimido (HPACK)	Nenhum
Binário	Binário	Binário e textual
Priorização	Possível	Não é possível
Compressão	Possível	Possível
« <i>Multiplexing</i> »	Possível	Possível

Quadro 1: Diferenças técnicas entre um pedido HTTP e uma ligação por *WebSocket*

Fonte: <https://developerinsider.co/difference-between-http-and-http-2-0-websocket/>

2.8 Docker *containers*

O Docker é uma «*Platform as a Service – PaaS*» «*open-source*» desenvolvido na linguagem de programação Go pela empresa Docker Inc. O *software* visa garantir uma maior facilidade na criação e administração de ambientes isolados, assegurando assim a rápida disponibilização de aplicações ao utilizador final.

Esta tecnologia usa o *kernel* do Linux e os recursos do *kernel* como *Cgroups* e *namespaces* para segregar processos. Assim, estes podem ser executados de maneira independente. O objetivo dos *containers* é criar essa independência – a habilidade de executar diversos processos e aplicações separadamente para utilizar melhor a infraestrutura e, simultaneamente, manter a mesma segurança. Traçado esse objetivo a Docker Inc. criou esta

plataforma que permite a criação, implementação e execução de aplicações num ambiente separado da máquina original com recurso a *containers*.

Os *containers* são uma unidade padrão de *software* que permite aglomerar toda a aplicação e todas as suas dependências num recipiente virtual que possa ser executado em qualquer ambiente Linux de forma rápida e segura. Isto permite uma maior flexibilidade e portabilidade no local no qual a aplicação será executada; «on-prem», *Cloud* pública, *Cloud* privada, etc. A maior vantagem de utilização desta tecnologia é a rapidez com que o *software* pode ser disponibilizado e a possibilidade de configurar diferentes ambientes de forma rápida sem que haja preocupação com incompatibilidades entre sistemas operativos.

Para além das vantagens supramencionadas também existem outros benefícios que são disponibilizados pela tecnologia:

- ❖ **Modularidade** – A abordagem do Docker para a «*containerization*» concentra-se na possibilidade de «desativar» uma parte de uma aplicação, seja para reparação ou atualização, sem a interromper totalmente. Além desta abordagem baseada em micro serviços é possível partilhar processos entre várias aplicações tal como na arquitetura orientada a serviços (SOA);
- ❖ **«Layers» e controlo de versão de imagens** – Cada ficheiro de imagem Docker é composto por uma série de «*layers*» combinadas numa única imagem. O Docker reutiliza estas «*layers*» para a construção de novos *containers*, o que torna o processo de criação extremamente rápido e eficiente. O controlo de versões é inerente ao uso de «*layers*». Sempre que é realizada uma nova alteração é gerado um «*changelog*», fornecendo assim o controlo total sobre as imagens do *container*;
- ❖ **Reversão** – A melhor vantagem na criação de «*layers*» é a possibilidade de as reverter sempre que seja necessário. Se a versão atual da imagem não agradar, basta apenas reverte-la para a versão anterior. Este processo é compatível com uma abordagem de desenvolvimento ágil e possibilita as práticas de integração contínua (CI/CD);
- ❖ **Rápido «deploy»** – Grandes empresas de *IT* sabem que é extremamente importante fazer o «*deploy*» de uma aplicação o mais rapidamente possível. Por esse mesmo motivo, o Docker surge como uma excelente opção. Ao criarmos um *container* para cada desenvolvimento é possível partilhar rapidamente processos semelhantes com novas aplicações. Como não é necessário iniciar um

sistema operativo para adicionar ou mover um *container*, o tempo de «*deploy*» é substancialmente menor. A plataforma também fornece aos programadores a possibilidade de rapidamente retrocederem a aplicação para uma versão anterior caso seja detetado algum problema. Esta agilidade traz assim uma maior produtividade e segurança à equipa de desenvolvimento.

Por si só, o Docker é excelente a gerir *containers* únicos. No entanto, ao utilizarmos cada vez mais *containers* nos quais segregamos aplicações em dezenas ou centenas de *containers*, a gestão e a orquestração podem tornar-se um enorme desafio. Eventualmente, será necessário recuar e agrupar os *containers* para que sejam estabelecidos serviços como rede, segurança, telemetria etc. Para superar este desafio o Kubernetes é a tecnologia mais indicada a ser utilizada.

Na Figura 8 conseguimos verificar, através de uma representação esquemática, as diferenças estruturais inerentes às arquiteturas tradicionais em máquinas virtuais face às recentes arquiteturas em *containers*.

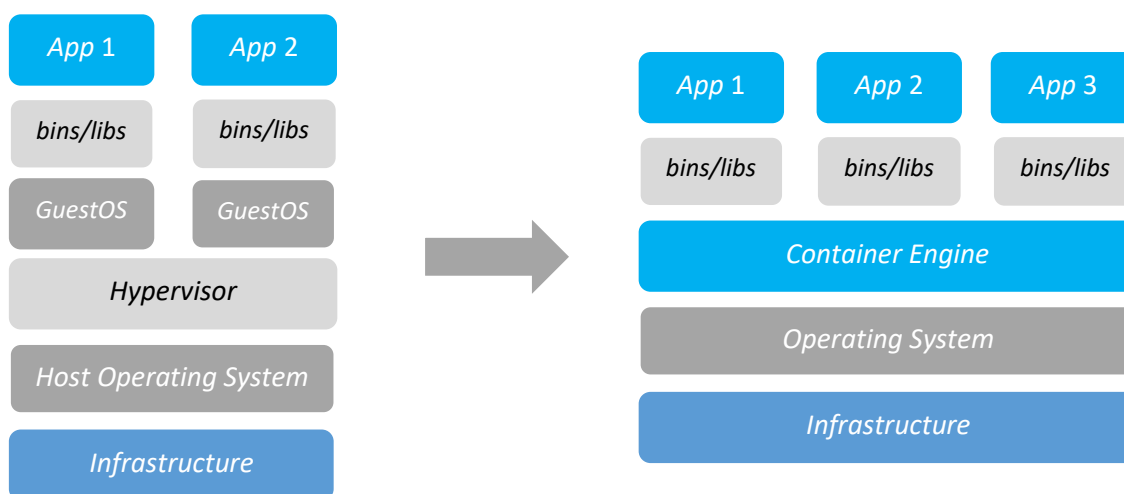


Figura 8: Diferenças estruturais inerentes às arquiteturas tradicionais em máquinas virtuais face às recentes arquiteturas em *containers*

Fonte: <https://www.docker.com/resources/what-container>

Resumo: A virtualização de recursos já se mostrou uma tendência que definitivamente veio para ficar. A criação de ambientes virtuais já provaram ser uma opção mais barata, flexível e escalável para as organizações e, por esse motivo, muitas delas estão empenhadas em apostar neste tipo de tecnologia. Gerar uma máquina virtual significa que um disco rígido virtual foi criado dentro de um ambiente físico por meio de um *software* específico, para que este seja operado de forma independente. Tanto em arquiteturas de *containers* como em máquinas virtuais ambas utilizam a virtualização. O que as distingue é que a virtualização é realizada ao nível do sistema operativo e os *containers* não utilizam *hypervisor*, mas sim os recursos do sistema e processos de *kernel* para criar os diversos ambientes. Este paradigma impede o *container* de ter uma visão geral do ambiente físico fora de seu espaço, contrariamente à máquina virtual.

CAPÍTULO III

Debaqi

III – DEBAQI

3.1. Objetivo geral

Pretende-se com este projeto realizar um estudo piloto sobre a interação e o diálogo dos jovens *online*. Para alcançar este objetivo foi desenvolvida uma aplicação *Web* que possibilita, através métodos computacionais, identificar o diálogo menos saudável, aquele que potencia a crispação, a polarização, a discriminação, o *bullying*, os discursos de ódio, a disseminação de notícias falsas e outras atitudes análogas.

3.2 Objetivo específico

O problema

Hoje em dia as plataformas *online* têm um efeito multiplicador na comunicação, o que se pode traduzir tanto em riscos, como em oportunidades para os seus utilizadores e estudiosos. Existe uma desmedida «riqueza» no conteúdo partilhado por estes e, por esse motivo, este estudo centra-se, sobretudo, na deteção dos «produtos» do *online*, recorrendo à ciência dos dados. Este é um enorme desafio para um problema difícil, de natureza complexa e multifatorial.

Nesta investigação, a comunicação será realizada via rede social fechada, de forma a entender que temas, comportamentos ou atitudes podem desencadear um diálogo menos saudável, por vezes violento ou mesmo discursos de ódio. Procuraremos ainda identificar áreas onde existe uma cultura positiva de diálogo construtivo. A abordagem centra-se particularmente na promoção do «diálogo saudável» entre os jovens.

Sinergias

Com o desenvolvimento desta aplicação queremos criar uma solução que permita realizar estudos em larga escala e em qualquer área, de modo a identificar fatores não óbvios à experiência Humana, bem como as suas interações e dinâmicas: elementos que cada vez são mais críticos para a tomada de decisões estratégicas. Para além destes intentos, queremos apoiar a evolução do papel do Professor Bibliotecário numa sociedade que se encontra permanentemente *online*, fornecendo ferramentas que permitam identificar tendências, áreas problemáticas e oportunidades de intervenção em tempo real.

3.3 Tecnologias e bibliotecas

“Lidar com tecnologias emergentes é sempre um desafio. É difícil descobrir se determinado fenómeno tecnológico que está a surgir é exagero, tendência ou *tsunami*” (Taurion, 2016). Por outro lado, o mercado está cada vez mais complexo e competitivo forçando empresas e pessoas a estarem em constante inovação.

As tecnologias emergentes têm o potencial de criar e reestruturar indústrias a um ritmo cada vez mais acelerado, tornando obsoletas práticas tradicionais e incentivando a disrupção digital que tem vindo a ganhar força nos últimos anos.

O papel da aplicação *Debaqi* não passa por ser mais uma força de resistência à mudança, mas sim o de incentivar e correr ao lado dela. Para o desenvolvimento desta aplicação recorreremos às mais «trendy» e atuais tecnologias «open-source», aquelas que estão alinhadas com os objetivos traçados pelos «stakeholders». Outra grande preocupação, presente na escolha do quadro tecnológico, foi a de garantir que a tecnologia permitisse um rápido desenvolvimento e possibilitasse a implementação de uma arquitetura moderna em *Cloud Microservices* altamente escalável.

O Quadro 2 enuncia as tecnologias mais relevantes para o desenvolvimento da aplicação *Debaqi*.

<i>Front-end</i>	<i>Back-end & AI</i>	<i>Base de Dados</i>	<i>DevOps & Cloud</i>	<i>Servidores</i>
Node JS	Spring Boot v2.1.6	PostgreSQL	Docker	Node
Angular 8	Spring Security	Liquibase	Bitbucket	Apache Tomcat
JavaScript e TypeScript	Hibernate & Spring Data	H2	Swagger	Gunicorn Nginx
SocketIO Client	Python 3		«Bitbucket Pipelines»	PostgreSQL
NGX-Translate	Flask		«Integration Tests»	
HTML 5	SQLAlchemy		«JUnits Tests»	
SASS e SCSS	Flask SocketIO		Heroku Cloud	
NPM	Maven		Heroku Telemetry	
Angular Bootstrap	Numpy & Nltk			
Angular-toastr	Scipy & Pandas			

Quadro 2: Tecnologias relevantes no desenvolvimento da aplicação *Debaqi*, agrupadas por camadas tecnológicas
Fonte: Elaboração própria

3.4 Arquitetura da solução

A aplicação *Debaqi* foi desenvolvida seguindo um modelo de componentes de múltiplos servidores e um servidor de base de dados com o objetivo primordial de ser altamente escalável e executável em qualquer ambiente *Cloud*, como ilustra a Figura 9.

Cada «*cloud node*» representa um ambiente Heroku¹⁰ independente, onde está a ser executado um Docker *container* para cada servidor distinto. Considerando a utilização de *containers*, o *software* pode ser disponibilizado rapidamente e tem ainda a possibilidade de poderem ser configurados diferentes ambientes de forma rápida sem a preocupação de ter incompatibilidades entre sistemas operativos.

Como podemos verificar pelo esquema representado na figura 9, apresentamos um *container* para um servidor Node JS, na qual vai ser executada toda a camada de *front-end* com recurso à *framework* Angular 8. De seguida, após um «*drill down*» encontramos mais dois *containers* que incorporam dois servidores de *back-end* distintos; um em Spring Boot (Java EE 8) e outro em Flask (Python 3).

A opção de dois servidores de *back-end* com tecnologias distintas foi assim deliberada, pois cada uma destas tecnologias é extremamente eficiente e robusta para alcançar o objetivo principal da aplicação.

O servidor desenvolvido em Spring Boot é responsável pela maior parte das funcionalidades do sistema, ou seja, pela autenticação, registo, gestão de tópicos, gestão de salas de debate, gestão de perguntas, auditorias, gestão do sistema, etc. Por outro lado, o servidor Flask foi desenvolvido em Python; uma linguagem de programação de baixo nível, que ao contrario do Java não requer um compilador, pois é uma linguagem interpretada.

Esta separação de «responsabilidades» traz atuais e trará futuros benefícios à aplicação: o processamento mais eficiente das mensagens vindas do *WebSocket*; sendo o Python uma linguagem de primeira opção no campo da inteligência artificial, ao contrário do Java, a introdução de mais agentes deste tipo é facilitada, pois a tecnologia já se encontra implementada no «*workflow*» da aplicação. Por final, o Python por não requerer compilação, tem a vantagem de o «*restart*» do servidor ser realizado de forma quase instantânea, diminuindo assim o «*downtime*».

Durante o período de «*handshake*», o servidor Flask recorrendo a um «*Object Relational Mapping – ORM*» (SQL-Alchemy) vai guardar na base de dados todas as interações

¹⁰ O Heroku é uma «*Platform as a Service*» (PaaS) que permite aos programadores criar, executar e operar aplicações desenvolvidas nas mais diversas linguagens de programação num ambiente totalmente *Cloud*.

realizadas durante a atividade. Estas interações são exatamente todo o conteúdo escrito nas mensagens, reações de cada mensagem («likes», «deslikes», «angry», etc.), menções, imagens partilhadas e *emojis*. À posteriori todo este conjunto de dados será extraído e analisado de forma a obter métricas relevantes para o estudo pretendido.

Na terceira camada está implementado mais um *container* que executa um servidor de base de dados PostgreSQL. Escolheu-se esta tecnologia por esta ser uma de base de dados «open-source» altamente robusta e que incorpora tecnologias NoSQL. Esta tecnologia pode vir a ser extremamente útil no futuro da aplicação. De momento, a aplicação está apenas a guardar os dados de forma «relacional», mas caso haja a necessidade no futuro de alterar para «não relacional», a tecnologia já suporta a necessidade e o tempo necessário para esta alteração será mais reduzido.

Concluindo e mais uma vez salientando a preocupação com a segurança, toda a transferência de dados na aplicação é realizada de forma segura através dos protocolos HTTPS e WSS. O HTTPS é uma implementação do protocolo HTTP sobre uma camada adicional de segurança que utiliza o protocolo SSL/TLS. Essa camada complementar permite que os dados sejam transmitidos por meio de uma ligação criptografada e que se verifique a autenticidade do servidor e do cliente por meio de certificados digitais. A porta TCP usada por norma para o protocolo HTTPS é a 443.

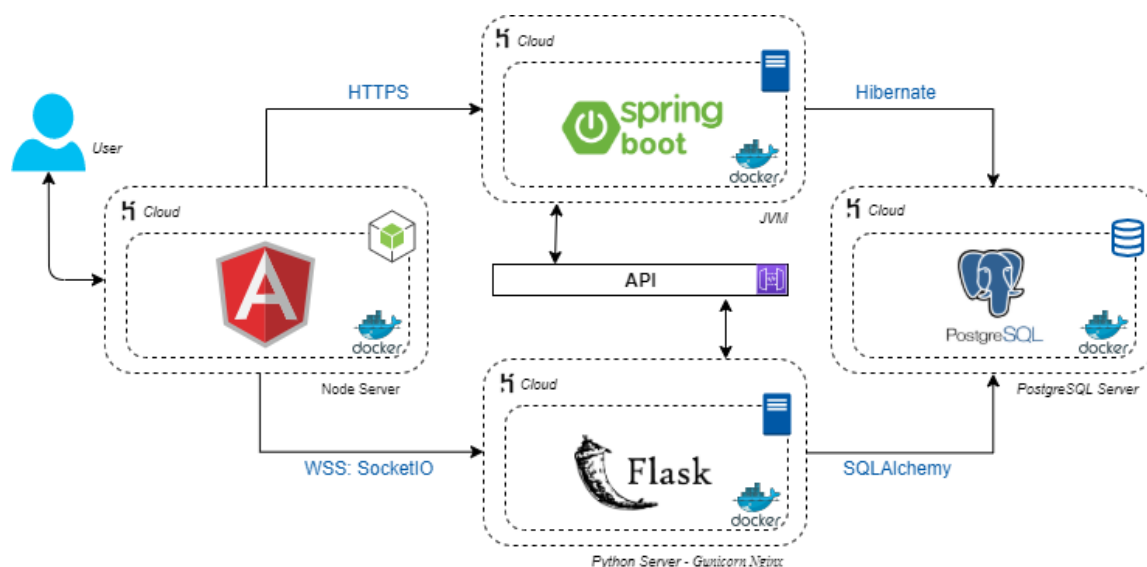


Figura 9: Diagrama conceptual da arquitetura implementada para a aplicação *Debaqi*

Fonte: Elaboração própria

Resumo: A aplicação *Debaqi* foi desenvolvida seguindo um modelo de componentes de múltiplos servidores e um servidor de base de dados. Cada «cloud node» representa um ambiente Heroku independente onde está a ser executado um Docker container para cada servidor distinto. O front-end foi desenvolvido recorrendo à *framework* Angular 8. O back-end, por outro lado, foi desenvolvido em tecnologias diferentes, um em Spring Boot (Java EE 8) e outro em Flask (Python 3). Para o servidor de base dados optámos por utilizar uma base de dados relacional PostgreSQL.

3.5 Controlo de acesso na aplicação

A segurança em aplicações *Web* é atualmente considerada um tópico de extrema criticidade, principalmente devido ao aparecimento de vírus e de outras formas de exploração de *software* cada vez mais sofisticadas. A segurança deve ser considerada um elemento chave em qualquer projeto.

A autorização é um conceito crucial na implementação e entendimento sobre segurança em aplicações. A autorização utiliza a informação que foi validada durante o processo de autenticação para determinar se deverá ser concedido o acesso a um recurso em particular. Assim, as aplicações definem perfis aos utilizadores e com base nesses perfis os utilizadores têm ou não acesso a determinados recursos. Também devemos ter em consideração que para um utilizador ter um determinado perfil este deve estar autenticado, e como tal deve passar pelo sistema de autenticação.

Na aplicação *Debaqi* o acesso à maior parte das funcionalidades é controlado com base no perfil que o utilizador tem durante o período em que está autenticado no sistema. Este controlo de acessos é garantido com recurso a tecnologias como o «*Jason Web Token – JWT*», *Spring Security* e *Angular 8 Route Guards*. Isto significa que todas as funcionalidades da aplicação estão devidamente protegidas nas suas «*layers*» constituintes (*front-end* e *back-end*).

Para aceder a uma determinada funcionalidade na «*layer*» visual (*front-end*), o sistema não só cria dinamicamente o HTML no DOM, o que irá permitir a navegação, como também faz com que o «*route guard*» esteja em constante validação se o utilizador tem as permissões necessárias para aceder a uma determinada funcionalidade. Caso ocorra a invulgar situação na qual um utilizador acesse a uma página que não tenha permissões para visualizar, o Angular vai redirecioná-lo automaticamente para uma página de erro parametrizada no código da aplicação.

Os «*route guards*» da *framework* Angular são interfaces que informam o «*router*» se deve ou não permitir a navegação para uma página solicitada pelo utilizador. Os «*guards*» validam o pedido através de um valor «booleano» (verdadeiro ou falso) de uma «classe» que implemente a interface «*guard*» conferida.

Existem cinco tipos de «*route guards*» diferentes e cada um deles é ativado por uma sequência específica. Cada sequência fará com que o «*router*» se comporte de maneira diferente para o tipo de «*guard*» utilizado.

- ❖ ***CanActivate*** – Mediar a navegação para uma «*route*»;
- ❖ ***CanActivateChild*** – Mediar a navegação para uma «*route child*» ou «*sub-route*»;
- ❖ ***CanDeactivate*** – Mediar a navegação fora da «*route*» atual;
- ❖ ***Resolve*** – Recuperar dados da «*route*» antes da ativação da mesma;
- ❖ ***CanLoad*** – Mediar a navegação para um determinado módulo carregado de forma assíncrona.

A proteção de acesso à informação não se encontra apenas implementada na «*layer*» visual. No *back-end* também está implementada uma camada adicional de segurança com recurso à tecnologia Spring Security, que irá validar se o utilizador tem o perfil necessário para aceder a um determinado serviço.

O Spring Security é um dos muitos projetos da Spring by Pivotal, que tem por objetivo adicionar uma camada de segurança robusta a aplicações Java EE. A partir de uma classe de configuração podem ser definidas regras de acesso a partes específicas da aplicação conforme o perfil de cada utilizador.

Na aplicação *Debaqi* a segurança foi implementada com recurso à anotação «*@PreAuthorize*» nos métodos do controlador. Esta anotação contém código desenvolvido em linguagem SpEL, que é despoletado sempre que for necessário avaliar se determinado pedido tem a autenticação necessária. Se o acesso não for concedido, o método não será executado e por sua vez será retornado um erro HTTP 401 – Unauthorized. Na prática, o uso da anotação «*@PreAuthorize*» num método do controlador é muito semelhante ao uso de «*HttpSecurity pattern matchers*» num «*endpoint*» específico.

Por final, a autenticação no *Debaqi* é realizada através de um «*token*» JWT. Este «*token*» vai conter a informação que possibilita certificar a identidade de um utilizador e as suas permissões atuais inerentes ao seu perfil.

O JSON Web Token (JWT) é um padrão «*open-source*» (RFC-7519) que define como transmitir e armazenar objetos JSON de forma compacta e segura entre diferentes aplicações. Os dados nele contidos podem ser validados a qualquer momento, uma vez que o «*token*» é assinado digitalmente. Podemos assinar um JWT usando um segredo gerado por um algoritmo

«*Hash-based Message Authentication Code* – HMAC» ou utilizando um par de chaves públicas e privadas usando RSA. Por ser compacto este padrão torna-se rápido e pode ser enviado através do URL por parâmetro ou dentro de um *header* HTTP.

O JWT encaixa-se perfeitamente em dois cenários: nas autenticações ou na troca de informação. O uso mais comum hoje em dia é realizado em autenticações para «*single page applications*», podendo também a vir carregar dados que poderão ser úteis para a aplicação.

O funcionamento do JWT no *Debaqi* é realizado no momento do «*login*» em que a aplicação recebe um «*token*» JWT que será armazenado no «*local storage*» como um «*cookie*». A cada novo pedido feito ao *back-end* será passado um «*Authorization: Bearer <token>*» no cabeçalho HTTP que foi armazenado anteriormente no «*local storage*». Quando o *back-end* receber o «*token*» fará a validação do mesmo conferindo a sua autenticidade. Caso seja válido, o recurso é devolvido para o *front-end*, caso seja inválido retorna o erro HTTP 403 – Forbidden.

3.6 Funcionalidades vs. Perfil

O Quadro 3 enuncia as funcionalidades e perfis existentes na aplicação *Debaqi*. Para cada perfil foram delineadas as funcionalidades às quais faria sentido terem acesso. As funcionalidades marcadas pelo ícone (✓) indicam que o perfil tem permissões para aceder ao recurso, as marcadas pelo ícone (✗) indicam o oposto.

A criação de novos perfis não está limitada, somente não foram identificados mais perfis necessários nos requisitos definidos para a aplicação.

Funcionalidade \ Perfil	Aluno	Moderador	Administrador
Página Inicial	✓	✓	✓
Mudar de idioma	✓	✓	✓
Criar conta	✓	✗	✗
Autenticação	✓	✓	✓
Sair / «Logout»	✓	✓	✓
Configuração de conta	✓	✓	✓
Mudar palavra-chave	✓	✓	✓
Lista de debates abertos	✓	✓	✓
Aceder ao debate	✓	✓	✓
Gestão de questões do utilizador	✗	✓	✓
Gestão de tópicos	✗	✓	✓
Gestão de salas de debate	✗	✓	✓
Gestão de utilizadores	✗	✗	✓
Métricas	✗	✗	✓
Estado do sistema	✗	✗	✓
Configuração do sistema	✗	✗	✓
Auditorias	✗	✗	✓
Logs	✗	✗	✓
API - Swagger	✗	✗	✓
Auditoria de entidades	✗	✗	✓
Gestão de entidades	✗	✗	✓

Quadro 3: «Check-list» de perfis que têm acesso ou não às diversas funcionalidades presentes na aplicação
Fonte: Elaboração própria

3.7 Funcionamento da aplicação *Debaqi*

De seguida, iremos proceder a uma descrição, por vezes exaustiva, do funcionamento da aplicação *Debaqi* por funcionalidades/páginas. Todas as imagens presentes neste ponto são de elaboração própria e obtidas recorrendo a «*print screens*» à aplicação.

3.7.1 Página Inicial

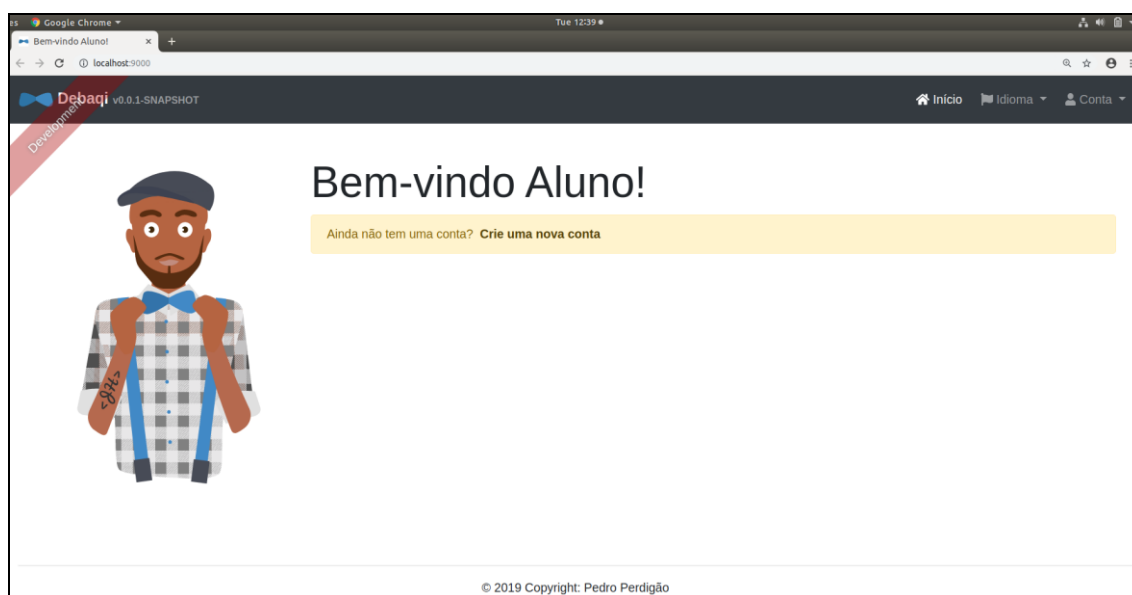


Figura 10: «*Print screen*» da página inicial

O ecrã representado na Figura 10 demonstra a página inicial da aplicação. O ecrã dá as boas vindas aos alunos e a possibilidade de os mesmos selecionarem uma de três opções. Efetuar o registo de um aluno novo, iniciar a sessão de um aluno previamente registado ou mudar o idioma da aplicação.

Neste momento o servidor não trata qualquer pedido de «*handshake*» com o *WebSocket*. Estes pedidos são realizados apenas nas páginas que envolvem comunicação em tempo-real: *chat*.

3.7.2 Autenticação

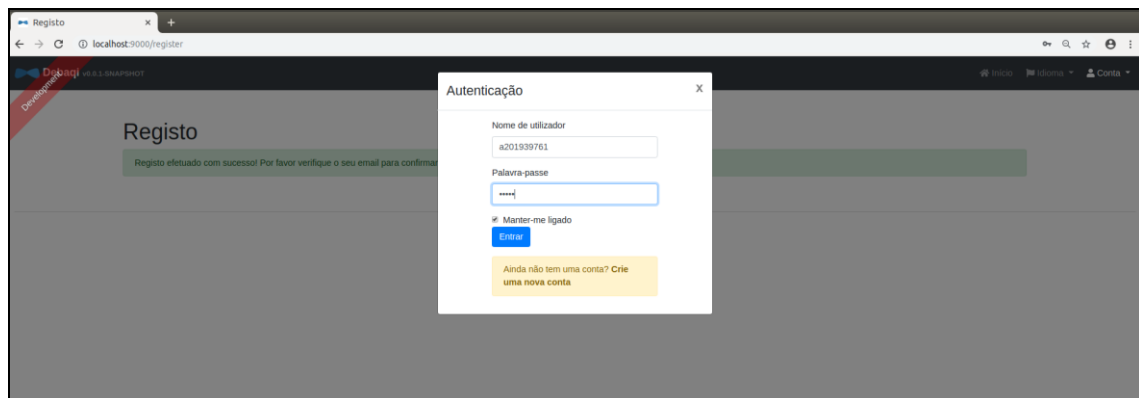


Figura 11: «Print screen» do «modal window» para autenticação dos utilizadores

O ecrã representado na Figura 11 demonstra a página que possibilita a autenticação de um determinado utilizador na aplicação, através de um protocolo seguro HTTPS e a encriptação da palavra-chave por meio de uma função *hash* criptográfica SHA256. Se os dados introduzidos estiverem incorretos o utilizador será notificado para o sucedido.

3.7.3 Registo do aluno

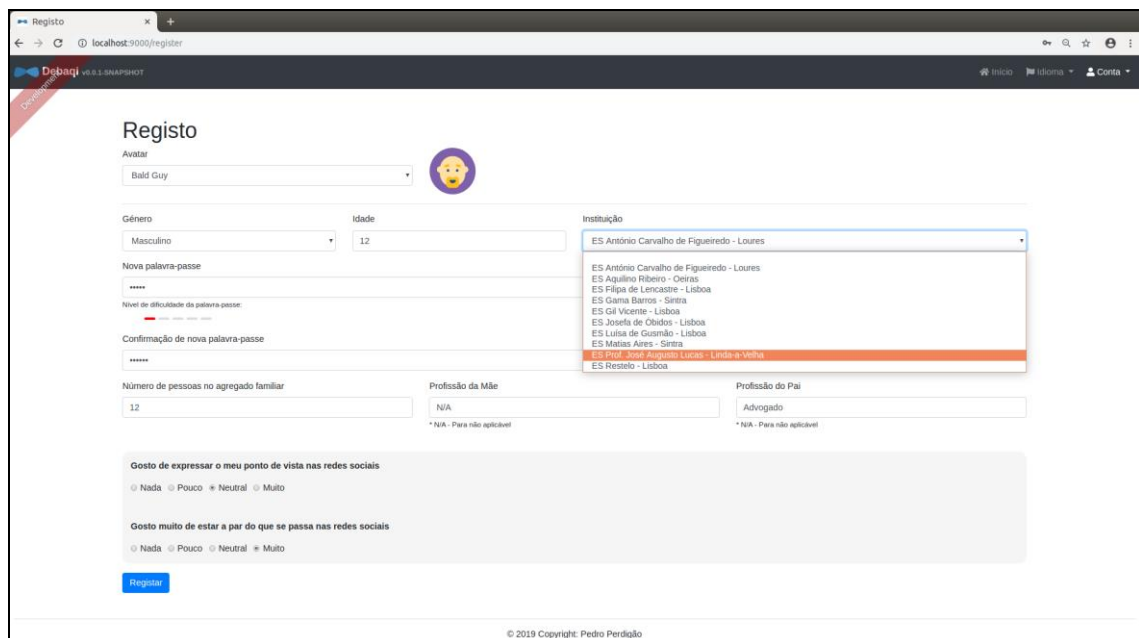


Figura 12: «Print screen» da página que possibilita o registo do aluno

O ecrã representado na Figura 12 demonstra a página que possibilita o registo de um aluno na aplicação.

A privacidade de cada aluno e a sua informação é um dos principais focos de atenção nesta aplicação, como tal, não é guardada qualquer informação que possa levar à identificação direta do aluno que está a participar nas sessões de trabalho. O registo é efetuado de forma totalmente anónima e desta forma criamos a liberdade necessária para que cada aluno sinta que se pode exprimir sem medo de ser identificado pelos restantes colegas. Este anonimato aumenta consequentemente o grau de franqueza e o sentimento depositado nas afirmações que cada aluno expõe durante as sessões em que participa, o que para a componente analítica é um ponto fundamental.

Um dos principais objetivos da aplicação *Debaqi* passa por analisar os dados gerados no debate após cada atividade com recurso a inteligência artificial. Para conseguirmos distinguir e entender determinadas polarizações nos dados recolhidos, no momento em que os alunos se registam na aplicação, temos de recolher alguma informação genérica sobre os mesmos, mas tendo sempre presente a preocupação de que não seja possível a sua identificação direta.

Os dados que a aplicação recolhe no momento em que o aluno se regista são os seguintes:

- ❖ *Avatar*;
- ❖ Género – masculino; feminino; outro; não quero dizer;
- ❖ Idade;
- ❖ Instituição onde é estudante;
- ❖ Palavra-chave e a sua confirmação;
- ❖ Número de pessoas no seu agregado familiar;
- ❖ Profissão do Pai e da Mãe (N/A para não aplicável);

Depois de ser inserida a informação supramencionada, existe um formulário visível com fundo cinzento (presente na Figura 12), que contém perguntas e respostas totalmente parametrizáveis pelo grupo científico responsável pela gestão da aplicação *Debaqi*. As perguntas vão possibilitar compreender qual o tipo de personalidade do aluno que se registou no sistema.

Durante as sessões foram realizadas as seguintes questões:

- ❖ Gosto de expressar o meu ponto de vista nas redes sociais;
- ❖ Considero importante tomar em conta as opiniões dos outros;
- ❖ Gosto de discutir *online* mesmo que não concorde com algumas opiniões.

3.7.4 Parametrização de perguntas e respostas para o formulário de registo

The screenshot displays the 'Questões do Utilizador' interface. At the top, there's a header with navigation links: 'Início', 'Entidades', 'Chat', 'Administração', 'Idioma', and 'Conta'. Below the header, the main content area is titled 'Questões do Utilizador' and includes a '+ Crie uma nova questão de utilizador' button. The interface shows a list of existing questions, each with a 'Código' (1 and 2) and a 'Questão' (text). Each question has a 'Eliminar' button. Below the list, there's a form to add a new question, consisting of a 'Código' field, a 'Questão' text area, and a 'Eliminar' button. At the bottom of the form are 'Cancelar' and 'Guardar' buttons. The footer of the page indicates '© 2019 Convidado: Porto Portugal'.

Figura 13: «Print screen» da página que possibilita a gestão de perguntas para serem disponibilizadas no ecrã de registo

O ecrã representado na Figura 13 demonstra a página que possibilita a parametrização do formulário exposto no momento em que um aluno acede à página de registo. Apenas utilizadores com perfil de «administrador» ou «moderador» têm acesso a esta funcionalidade. Esta limitação não está apenas implementada na camada visual. A chamada ao serviço também impõe que o «token» contenha os perfis mencionados; caso contrário o utilizador não terá permissões de acesso ao serviço.

As questões podem ser introduzidas através de texto livre, mas as respostas são opções estanques, para que exista coerência na componente analítica. O utilizador pode adicionar, modificar e apagar as perguntas e respostas a qualquer altura.

3.7.5 Registo efetuado com sucesso

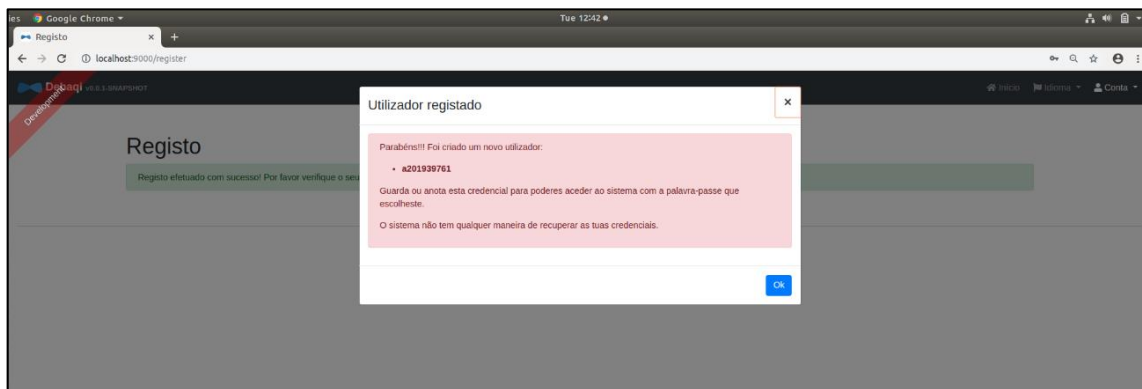


Figura 14: «Print screen» do «modal window» visualizado após um registo efetuado com sucesso

O ecrã representado na Figura 14 é apresentado sempre que um aluno efetue um registo com sucesso na aplicação.

Mantendo a premissa do anonimato, os alunos não têm a possibilidade de escolher o nome de utilizador que querem utilizar na aplicação. Eventualmente acabariam por escolher nomes que usam em plataformas *online*, o que poderia levar à sua identificação por colegas mais próximos. Para que essa situação não se suceda, a aplicação gera automaticamente um nome de utilizador aleatório que os alunos terão de guardar da maneira que acharem mais oportuna. O nome de utilizador gerado de forma automática é apresentado logo após o momento do registo, como podemos verificar no exemplo da Figura 14: Foi criado um novo utilizador **a201939761**.

A aplicação não tem qualquer informação direta sobre os alunos que estão registados e as suas palavras-chave encontram-se totalmente encriptadas. Caso algum aluno perca as suas credenciais a aplicação não conseguirá recuperar a conta perdida. O cenário anteriormente descrito foi previsto e, por esse motivo, implementado na aplicação fazendo com que esta se encontre em «*compliance*». Deste modo, fica assegurado o cumprimento dos requisitos legais de privacidade e proteção de dados regulamentados pelas leis europeias e portuguesa, bem como as exigências prescritas pelo Ministério da Educação.

Qualquer registo efetuado na aplicação através do seu «*workflow*» normal irá atribuir automaticamente ao utilizador um perfil de «aluno».

3.7.6 Autenticação com sucesso – Perfil de aluno

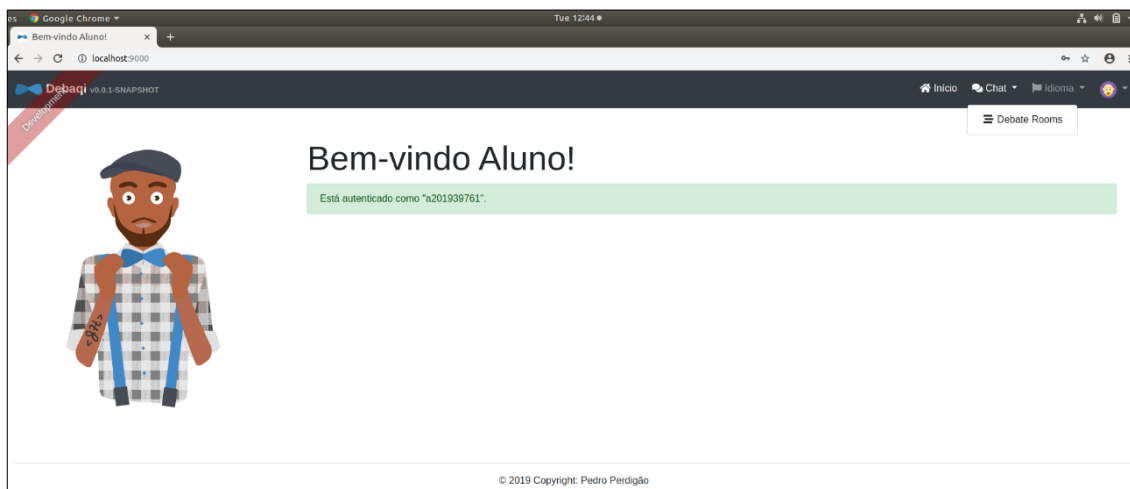


Figura 15: «Print screen» da página inicial da aplicação, para um utilizador que possua perfil de «aluno»

O ecrã representado na Figura 15 demonstra a página inicial após uma autenticação bem-sucedida feita por um utilizador que tem o perfil de «aluno».

Com este tipo de perfil, o aluno continua não só a poder aceder às opções anteriormente disponíveis no ecrã inicial, como passa também a ter acesso a funcionalidades tais como aceder à lista de debates que estão a decorrer, alterar a sua palavra-chave, ver o seu *avatar* e aceder à sua informação genérica.

3.7.7 Configurações do utilizador

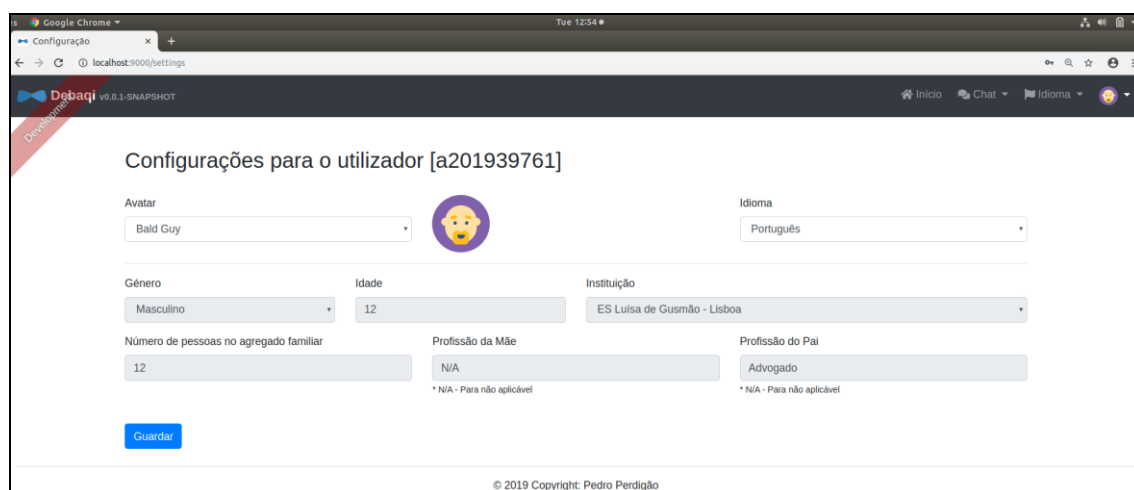


Figura 16: «Print screen» da página que possibilita visualizar a informação genérica do aluno

O ecrã representado na Figura 16 demonstra a página responsável por apresentar ao aluno toda a sua informação genérica que foi introduzida no momento do registo.

Nesta página grande parte dos dados não são passíveis de alteração, considerando, uma vez mais, a manutenção da coerência na componente analítica. Os alunos têm apenas a possibilidade de alterarem o seu *avatar* e de escolherem o idioma no qual desejam que a aplicação seja visualizada no momento de autenticação.

3.7.8 Lista de debates a decorrer

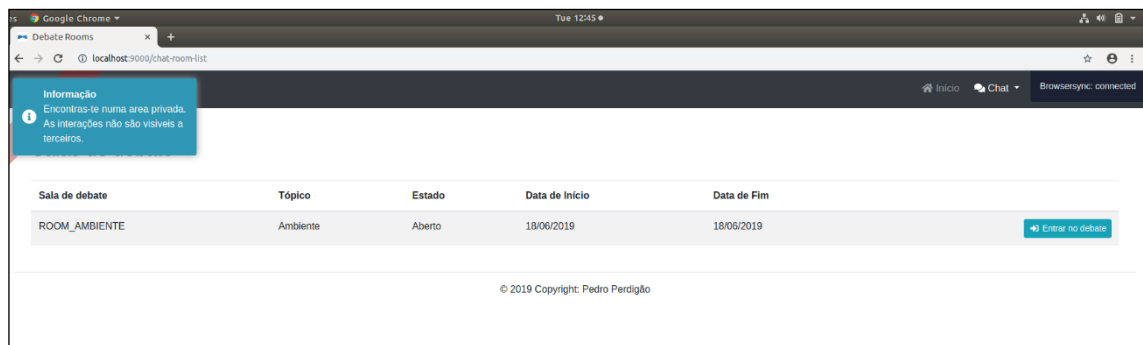


Figura 17: «Print screen» da página que possibilita o utilizador entrar nos debates a decorrer

O ecrã representado na Figura 17 demonstra a página responsável por apresentar ao aluno os debates em curso. Aqui o aluno tem a possibilidade de aceder a qualquer debate que esteja a decorrer durante o tempo previamente definido pelo administrador do sistema ou grupo responsável pela atividade, bem como saber o tópico/tema de cada debate, a sua data de início e fim e o estado da sala de debate.

3.7.9 Entrada no canal de debate

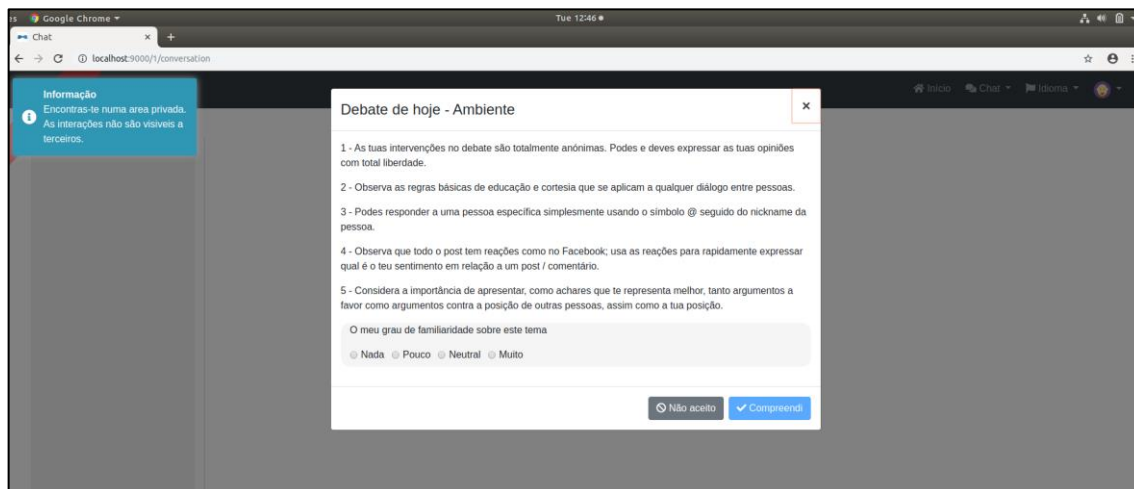


Figura 18: «Print screen» do «modal window» apresentado antes do aluno entrar no debate a decorrer

O ecrã representado na Figura 18 demonstra o ecrã que é apresentado ao aluno no momento em que o mesmo solicita aceder ao um determinado debate.

Neste ecrã serão apresentados os termos e as condições do debate, os quais terão de ser aceites pelo aluno. Na eventualidade destes não serem aceites pelo mesmo, este ficará impossibilitado de aceder ao debate e o «router» reencaminhará automaticamente o navegador para a página da lista de debates a decorrer.

Neste ecrã também pode ser parametrizado um vídeo, que poderá ser visualizado pelos alunos, para que estes fiquem a par do enquadramento da temática a ser abordada e terão também de responder a um conjunto de questões igualmente parametrizáveis para que a componente analítica consiga recolher dados relevantes relativamente ao grau de conhecimento que cada aluno tem antes das atividades.

Após o aluno ter aceite os termos e condições e respondido a todas as questões exigidas, este ecrã não irá aparecer mais para o tópico/tema do debate em questão, ou seja, se o aluno sair e voltar a entrar na sala será reencaminhado diretamente para o *chat*. Se o tópico for diferente então o aluno será obrigado a repetir novamente o processo e a responder às respetivas questões sobre a temática e a aceitar os termos e as condições. A questão colocada para todas as atividades foi: Qual o grau de familiaridade sobre o tema.

3.7.10 Gestão de tópicos



Figura 19: «Print screen» da página que apresenta a lista de todos os tópicos existentes

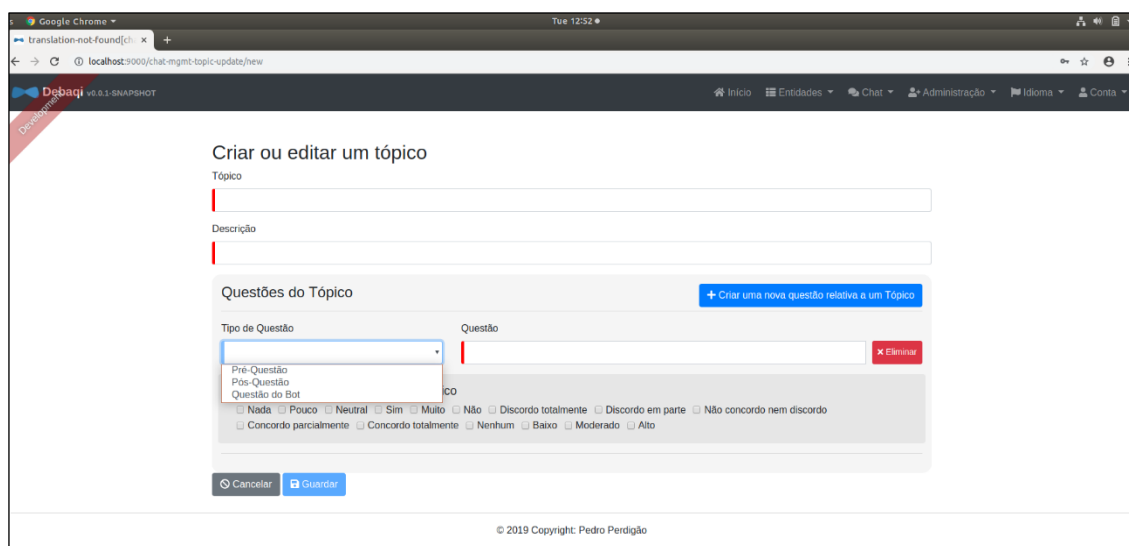


Figura 20: «Print screen» da página que permite criar ou editar um tópico

Apenas os utilizadores com o perfil de «administrador» e «moderador» estão habilitados a aceder a esta funcionalidade. Os ecrãs representados na Figura 19 e na Figura 20 demonstram as páginas que possibilitam efetuar a gestão de tópico/temáticas que serão debatidas nas salas virtuais.

No ecrã representado na Figura 19, o utilizador tem a possibilidade de criar um novo tópico ou uma nova sala de debate virtual. Para além destas opções, o utilizador também tem a possibilidade de visualizar uma tabela que demonstra os tópicos que já foram criados anteriormente e pode visualizar e editar cada um deles, mas nunca apagar.

O ecrã representado na Figura 20 é apresentado sempre que o utilizador deseja editar ou criar um novo tópico. Nesta página o gestor do tópico tem «inputs», nos quais deve inserir por poucas palavras o tema do debate e uma pequena descrição que esclareça os alunos acerca dos pontos que pretendem abordar durante a atividade.

Para além dos «inputs» referidos, esta página também possibilita a parametrização de perguntas e respostas, a serem apresentadas nas seguintes situações:

- ❖ PRE-QUESTION – Questões apresentadas no primeiro acesso ao debate (ecrã de termos e condições);
- ❖ POS-QUESTION – Altura em que o moderador solicita um momento de reflexão sobre a temática e na qual é apresentado um conjunto de questões, às quais os alunos têm de responder após terem participado na atividade;
- ❖ BOT-QUESTION – Questões que podem ser lançadas automaticamente pelo *bot*, para «quebrar o gelo» no decorrer da atividade ou para suscitar mais intervenção por parte dos alunos, quando esta está a ser insuficiente.

3.7.11 Gestão de salas de debate

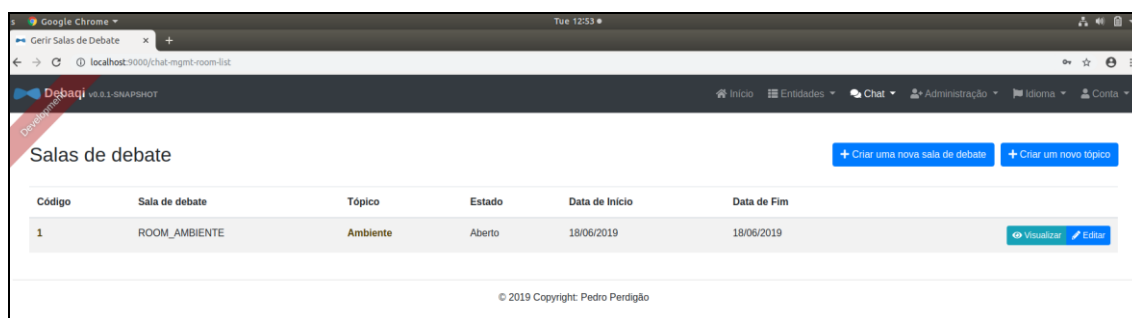


Figura 21: «Print screen» da página que apresenta a lista de todas as salas de debates existentes

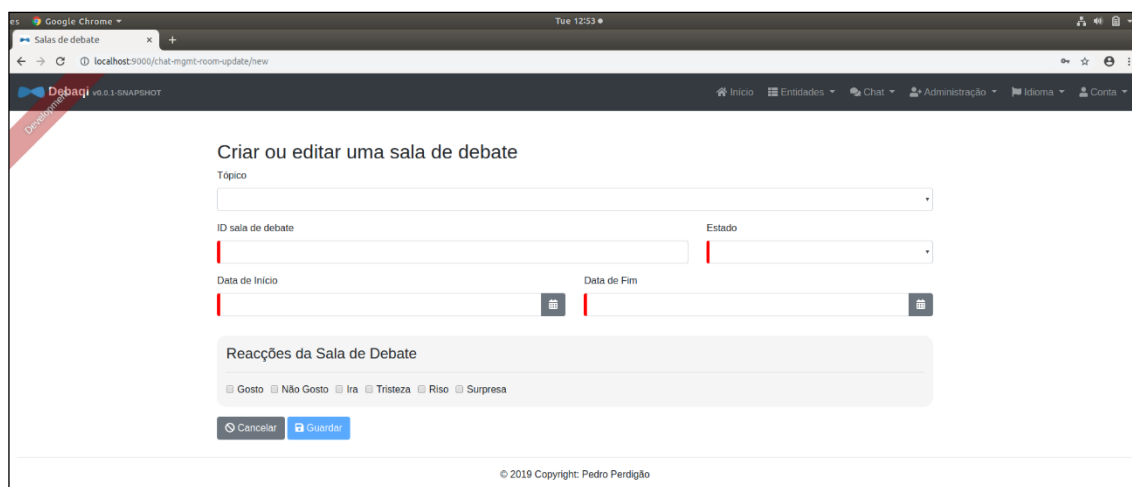


Figura 22: «Print screen» da página da página que permite criar ou editar uma sala de debate

Uma vez mais, estão apenas habilitados a aceder os utilizadores com o perfil de «administrador» e «moderador». Os ecrãs representados na Figura 21 e na Figura 22 demonstram as páginas que possibilitam efetuar a gestão das salas de debate.

No ecrã representado na Figura 21, o utilizador tem igualmente, tal como na Figura 19, a possibilidade de criar um novo tópico ou uma nova sala de debate virtual. Estas funcionalidades encontram-se igualmente disponíveis em ambos os ecrãs, porque ambas «caminham lado a lado», ou seja, um tópico terá sempre «0..*» salas de debate; e uma sala de debate terá sempre um tópico associado.

Para além das opções referidas o utilizador também tem a possibilidade de visualizar uma tabela que demonstra as salas de debate já criadas anteriormente e pode visualizar ou editar cada uma delas, mas nunca apagar.

O ecrã representado na Figura 22 é apresentado sempre que o utilizador deseje editar ou criar uma nova sala de debate. Nesta página, o gestor da sala tem «*inputs*» onde deve referir o tópico inerente à sala de debate, introduzir um identificador da sala, definir o estado em que a sala se encontra (Aberta, Fechada ou Pendente) e o intervalo de datas em que será realizado o debate. O estado da sala e as datas são parâmetros que devem estar bem definidos, pois quando os alunos tentarem aceder aos debates, se a data de fim for anterior à data atual ou o estado for diferente de aberto, o debate não se irá encontrar disponível na lista de debates a decorrer.

Por último, neste mesmo ecrã o utilizador tem ainda a possibilidade de parametrizar as reações que irão estar disponíveis para que os alunos possam reagir a uma determinada mensagem no *chat*. É ainda importante referir que é possível parametrizar reações diferentes nas múltiplas salas relacionadas ao mesmo tópico/tema.

3.7.12 O chat

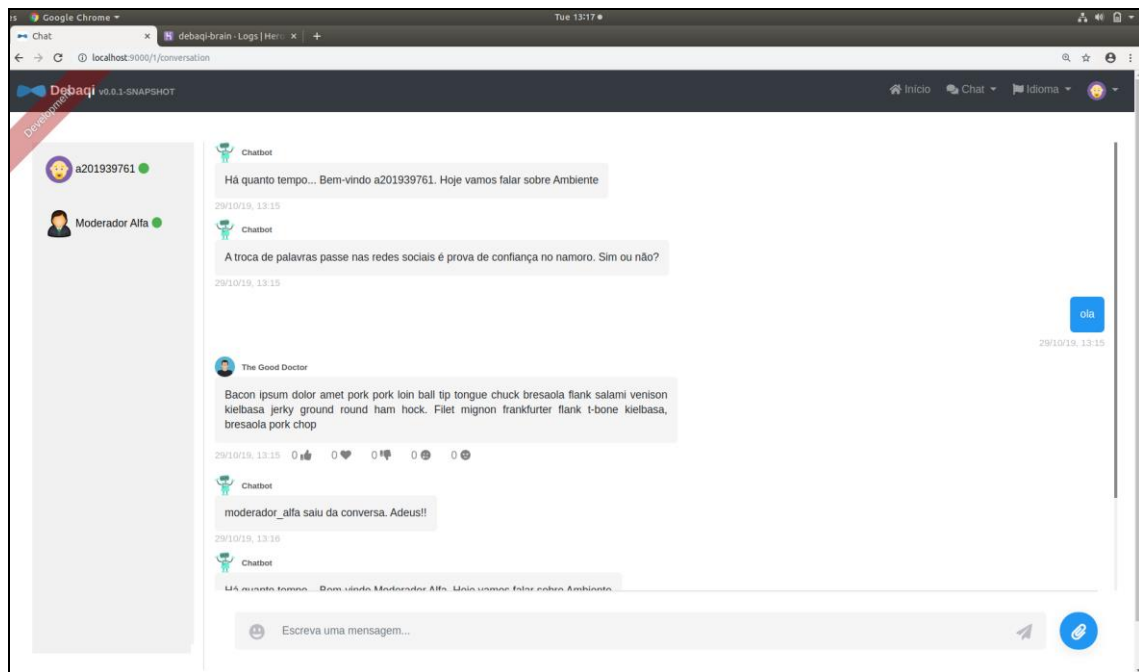


Figura 23: «Print screen» da funcionalidade de chat

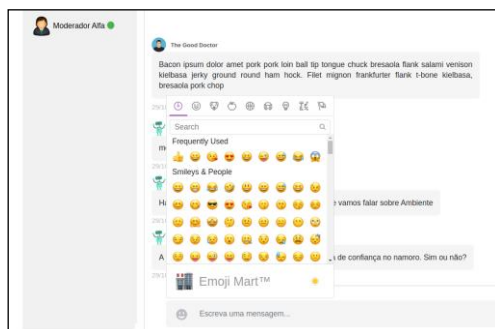


Figura 24: «Print screen» da funcionalidade de emojis

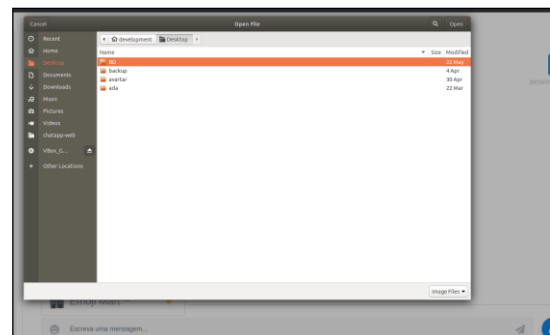


Figura 25: «Print screen» da funcionalidade de envio de imagens

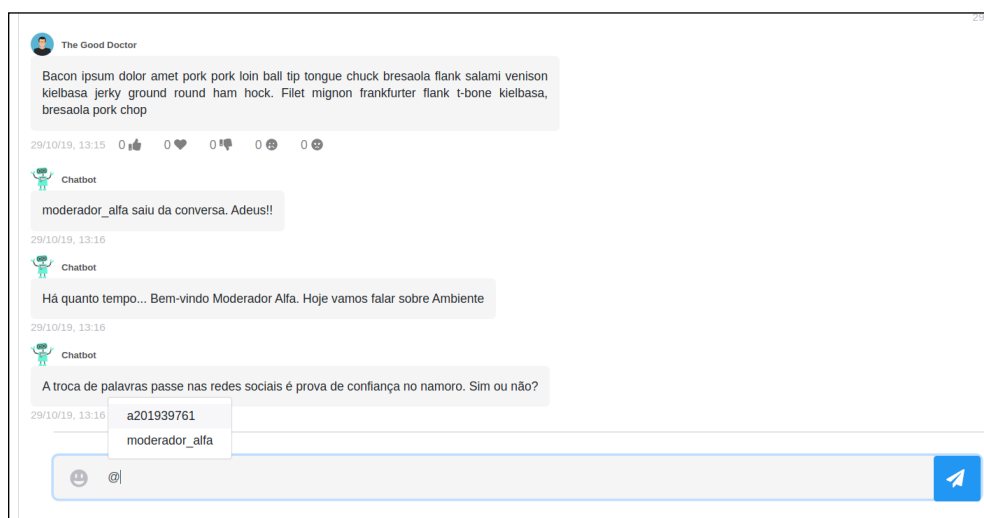


Figura 26: «Print screen» da funcionalidade que permite o envio de menções e reações a mensagens

O ecrã representado na Figura 23 demonstra o ecrã que é apresentado ao aluno no momento em que o mesmo acede a um determinado debate. Quando este entra na sala virtual o *chatbot* dá-lhe as boas-vindas e apresenta o tema que irá ser debatido. Caso um aluno abandone a sessão a meio o *chatbot* avisa os restantes participantes do sucedido.

Como podemos verificar na mesma figura, esta funcionalidade tem um *design* tripartido, típico de qualquer *chat*. Aqui conseguimos visualizar o *avatar* e o «*alias*» dos alunos que se encontram na mesma sala de debate; o «*feed*» de conversação que contém as intervenções de outros é apresentado à esquerda de cor cinza e as próprias à direita de cor azul; por fim na zona inferior do ecrã encontramos a área que permite ao utilizador realizar a sua intervenção.

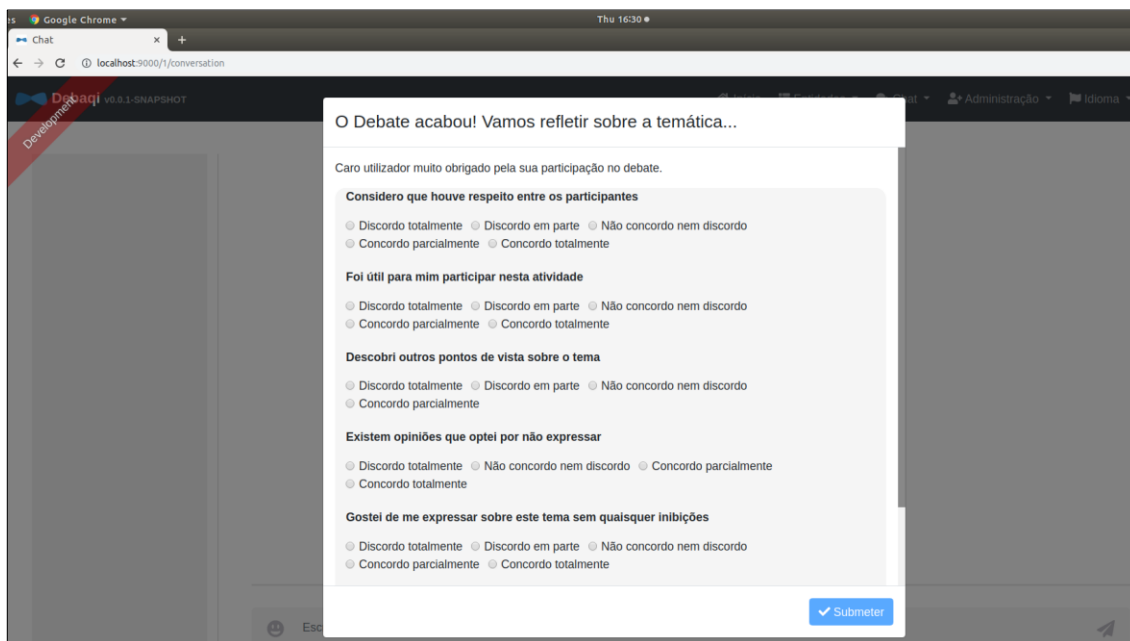
Como podemos verificar nos ecrãs representados nas figuras 24, 25 e 26, as intervenções realizadas pelos alunos podem ser enriquecidas de diversas formas. O *chat*, para além de enviar mensagens de texto, também permite mencionar outros alunos através da tecla @, enviar *emojis* e imagens no corpo da mensagem.

O *chatbot*, como referido anteriormente, pode ter um papel muito importante para «quebrar o gelo» durante uma atividade. Assim, como exposto na Figura 26, para além da menção, podemos verificar o *chatbot* a interagir com os alunos lançando perguntas para o «*feed*». Esta automatização vai deixar os alunos mais confortáveis para intervir sobre a temática.

Para além das intervenções já referidas, todos os participantes têm a possibilidade de reagir a qualquer mensagem que não tenha sido o próprio a redigir. O número de reações presentes em cada mensagem é incrementado em tempo real, possibilitando assim os alunos de visualizar as emoções incitadas ao grupo por uma determinada intervenção.

Por fim, o *chat* garante que toda a troca de mensagens esteja disponível durante o tempo da atividade, isto significa que, caso um utilizador por algum motivo técnico deixar de ter ligação à aplicação a meio da conversação, terá sempre a oportunidade de visualizar tudo o que foi escrito anteriormente, bem como reagir a intervenções que foram efetuadas enquanto se encontrava *offline*.

3.7.13 Fim do debate – Momento de reflexão



The screenshot shows a web browser window with the URL `localhost:9000/1/conversation`. A modal window is displayed in the center with the title "O Debate acabou! Vamos refletir sobre a temática...". The modal contains a thank-you message and a series of reflection questions, each with four radio button options: "Discordo totalmente", "Discordo em parte", "Não concordo nem discordo", and "Concordo parcialmente". The questions are: "Considero que houve respeito entre os participantes", "Foi útil para mim participar nesta atividade", "Descobri outros pontos de vista sobre o tema", "Existem opiniões que optei por não expressar", and "Gostei de me expressar sobre este tema sem quaisquer inibições". A "Submitter" button is at the bottom right of the modal.

Figura 27: «Print screen» do «modal window» pós-debate. Momento de reflexão...

O ecrã representado na Figura 27 demonstra uma das páginas mais importantes da aplicação: o questionário pós-debate. Aqui os alunos terão a possibilidade de refletir individualmente sobre a temática abordada, respondendo às perguntas que foram previamente parametrizadas pelo grupo de trabalho responsável pelo estudo.

Este ecrã surge no momento em que o moderador considera o debate encerrado. Ele é o grande responsável por decidir quando tal deve acontecer e, através de um comando específico, executar a função que vai despoletar o encerrar da sessão. Esta função vai impossibilitar o acesso ao *chat* a todos os participantes e logo de seguida exibir o questionário pós-debate, o qual deverá ser preenchido obrigatoriamente.

As respostas dos alunos a este questionário são muito importantes, por possibilitarem uma análise em diversos âmbitos, procurando, sobretudo, compreender se esta experiência teve ou não um impacto positivo nos mesmos.

As questões parametrizadas para todas as temáticas foram as seguintes:

- ❖ Considero que houve respeito entre os participantes;
- ❖ Foi útil para mim participar nesta atividade;
- ❖ Descobri outros pontos de vista sobre o tema;
- ❖ Existem opiniões que optei por não expressar;
- ❖ Gostei de me expressar sobre este tema sem inibições;
- ❖ Mudei o meu ponto de vista em relação ao tema do debate.

3.7.14 Autenticação com sucesso – Perfil de administrador

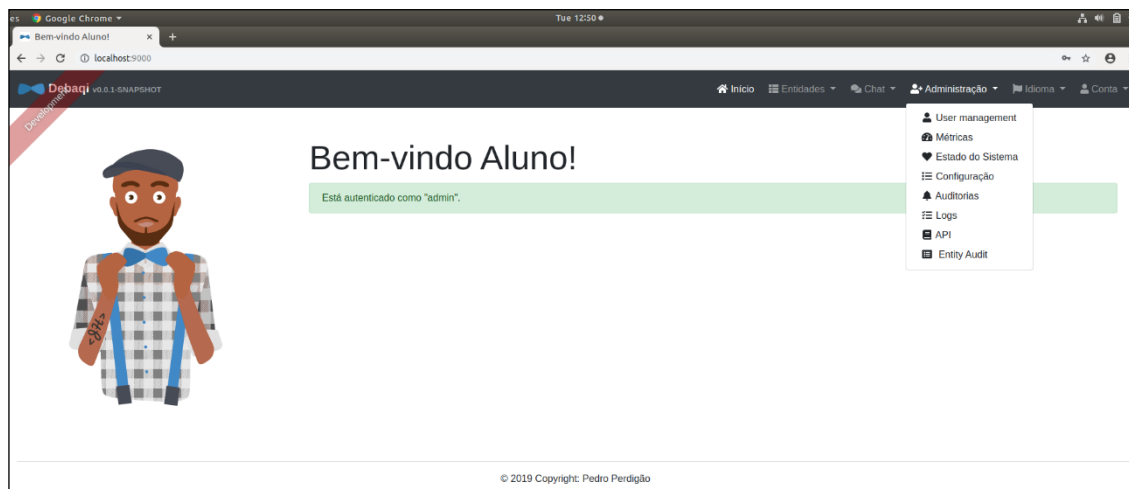


Figura 28: «Print screen» da página inicial da aplicação, para um utilizador que possui perfil de «administrador»

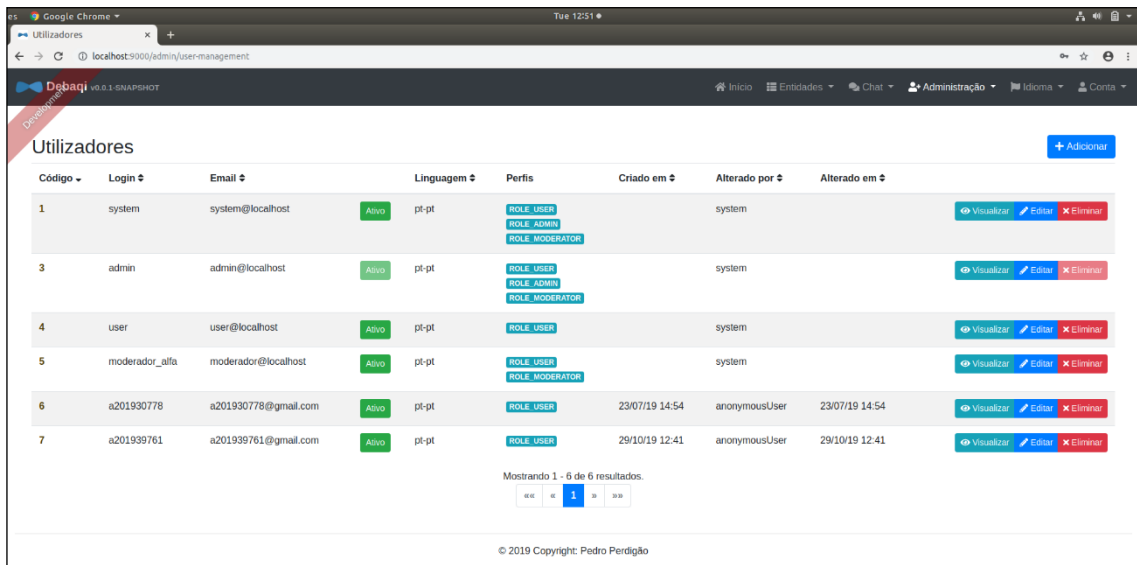
O ecrã representado na Figura 28 demonstra a página inicial após uma autenticação bem-sucedida por um utilizador que tem o perfil de «administrador».

Através deste perfil o utilizador tem acesso a todas as funcionalidades disponíveis na aplicação *Debaqi*. Acrescido às funcionalidades de um perfil de «aluno» e «moderador», é também disponibilizado ao administrador uma série de funcionalidades para efetuar a administração do sistema.

Estas funcionalidades passam pela gestão de utilizadores, métricas, estado do sistema, configurações, auditorias, *logs*, Swagger – API e auditorias à base de dados. Além do menu de administração é igualmente disponibilizado um novo menu que possibilita a gestão da base de dados de uma forma visual, ou seja, o administrador não necessita de escrever «*scripts*» se necessitar de corrigir alguma informação. O facto da gestão ser efetuada visualmente permite que os seus utilizadores não detenham conhecimentos técnicos sobre bases de dados.

Os restantes pontos abaixo vão descrever com mais detalhe cada funcionalidade de administração do sistema.

3.7.15 Gestão de utilizadores



Código	Login	Email	Estado	Lingagem	Perfis	Criado em	Alterado por	Alterado em	
1	system	system@localhost	Ativo	pt-pt	ROLE_USER ROLE_ADMIN ROLE_MODERATOR		system		Visualizar Editar Eliminar
3	admin	admin@localhost	Ativo	pt-pt	ROLE_USER ROLE_ADMIN ROLE_MODERATOR		system		Visualizar Editar Eliminar
4	user	user@localhost	Ativo	pt-pt	ROLE_USER		system		Visualizar Editar Eliminar
5	moderador_alta	moderador@localhost	Ativo	pt-pt	ROLE_USER		system		Visualizar Editar Eliminar
6	a201930778	a201930778@gmail.com	Ativo	pt-pt	ROLE_USER	23/07/19 14:54	anonymousUser	23/07/19 14:54	Visualizar Editar Eliminar
7	a201939761	a201939761@gmail.com	Ativo	pt-pt	ROLE_USER	29/10/19 12:41	anonymousUser	29/10/19 12:41	Visualizar Editar Eliminar

Mostrando 1 - 6 de 6 resultados.

© 2019 Copyright: Pedro Perdigão

Figura 29: «Print screen» da página que possibilita efetuar a gestão dos utilizadores presentes na aplicação

O ecrã representado na Figura 29 demonstra a página que habilita o administrador do sistema efetuar a gestão de todos os utilizadores registados na aplicação.

Neste ecrã o administrador pode visualizar, editar e remover utilizadores que tenham sido registados no sistema, bem como gerir o perfil que cada utilizador tem. Nem todos os dados podem ser editados por esta funcionalidade. Nela o administrador apenas consegue atualizar o *email*, o idioma, os perfis e o estado (ativo ou não) de cada utilizador específico.

Como não existe de forma propositada nenhuma página para registo de um moderador, o administrador nesta funcionalidade tem a possibilidade de criar um novo utilizador e ao fazê-lo pode atribuir-lhe o perfil de «moderador».

3.7.16 Métricas do sistema

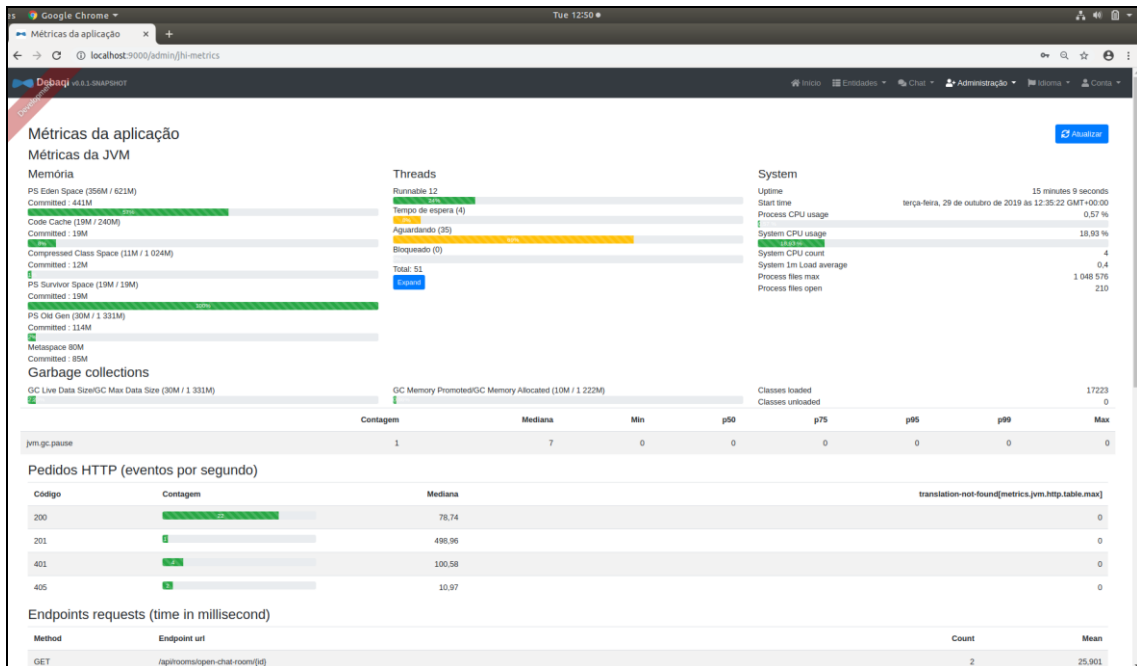


Figura 30: «Print screen» da página que possibilita a visualização das métricas aplicacionais

O ecrã representado na Figura 30 demonstra a página que habilita o administrador do sistema monitorizar métricas importantes do funcionamento da aplicação. Neste ecrã encontramos métricas como:

❖ JVM

- Memória;
- *Threads*;
- Estado do CPU;
- Estatística do «garbage collector».

❖ Pedidos HTTP

- Diferenciação por cada tipo de resposta (Número de pedidos e média em ms de resposta).

❖ Estado da cache

- Estatísticas da Ehcache utilizada pelo Spring.

❖ Estado da base de dados

- «Connection pools» abertas;
- Espaço em disco (livre e ocupado).

Todas estas métricas recolhidas são interessantes para entender se temos um sistema eficiente e inserido dentro das capacidades técnicas exigidas para a sua execução. Por exemplo, se identificarmos que os processos do CPU estão a atingir níveis perto dos 100%, sabemos imediatamente que temos de escalar e adicionar mais recursos ao CPU. Esta situação é válida para qualquer outra métrica; a identificação imediata do problema.

3.7.17 Auditorias

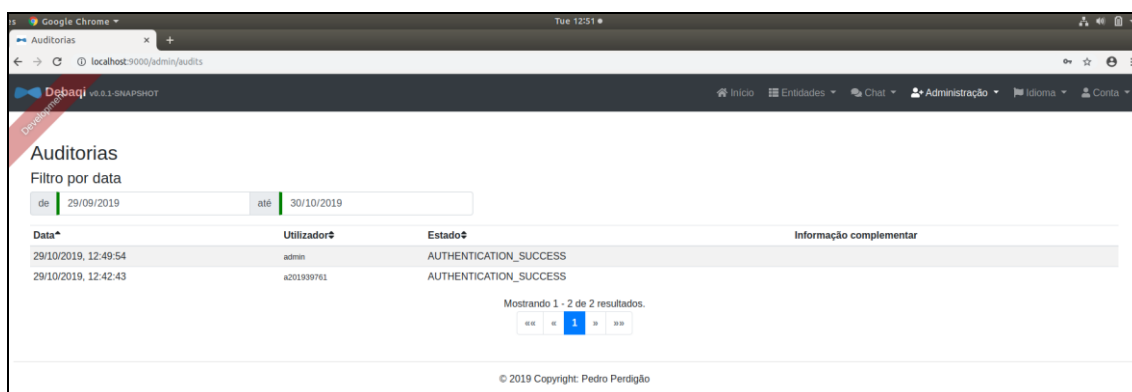


Figura 31: «Print screen» da página que possibilita visualizar as auditorias realizadas à aplicação

O ecrã representado na Figura 31 demonstra a página que habilita o administrador do sistema visualizar os registos de auditoria que foram captados pela aplicação.

A aplicação *Debaqi* por implementar uma interface genérica, permite auditar qualquer evento que possa ocorrer no sistema. De momento só foram parametrizadas auditorias para as autenticações com e sem sucesso, com o intuito de perceber se existem tentativas de acesso a contas alheias através de «*brute force*» ou «*password guessing*».

O sistema de auditoria permite que fique guardado em base de dados a data em que o evento sucedeu, o utilizador que a fez despoletar e a ação que foi auditada.

3.7.18 Auditorias às entidades (base de dados)

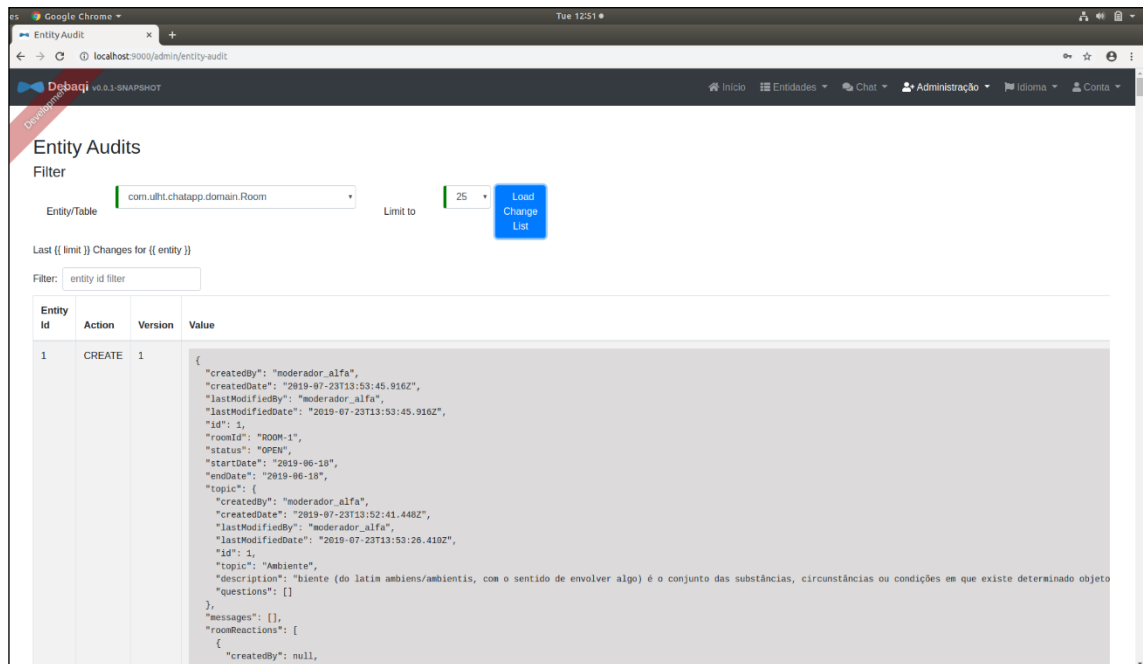


Figura 32: «Print screen» da página que apresenta as auditorias realizadas aos dados transferidos por pedidos HTTP

O ecrã representado na Figura 32 demonstra a página que habilita o administrador do sistema de visualizar através de um formato JSON os dados que foram transferidos para a base de dados em cada pedido HTTP.

Esta auditoria à base de dados vai possibilitar identificar se os dados provenientes de cada serviço são novos (CREATE), atualizados (UPDATE) ou removidos (DELETE). É, de facto, uma funcionalidade muito importante, porque para além de permitir a visualização dos dados que realmente foram transportados pelos serviços, também confere ao administrador do sistema a opção de validar os dados que se encontram na base de dados e se estes foram inseridos corretamente e não «manipulados» via «script».

Para além da auditoria efetuada aos dados provenientes dos pedidos HTTP, todos os registos inseridos na base de dados têm as suas respetivas colunas de auditoria. Este desenvolvimento foi efetuado de forma a ter a informação sobre a propriedade de cada registo.

As colunas de auditoria sobre a autoria do registo são as seguintes:

- ❖ **Created by:** «Username» do utilizador que inseriu o registo;
- ❖ **Created date:** Registo da data e hora (*timestamp*) de quando o registo foi inserido;
- ❖ **Last modified by:** «Username» do último utilizador que atualizou o registo;
- ❖ **Last modified date:** Registo da data e hora (*timestamp*) da última vez que o registo foi atualizado.

3.7.19 Logs

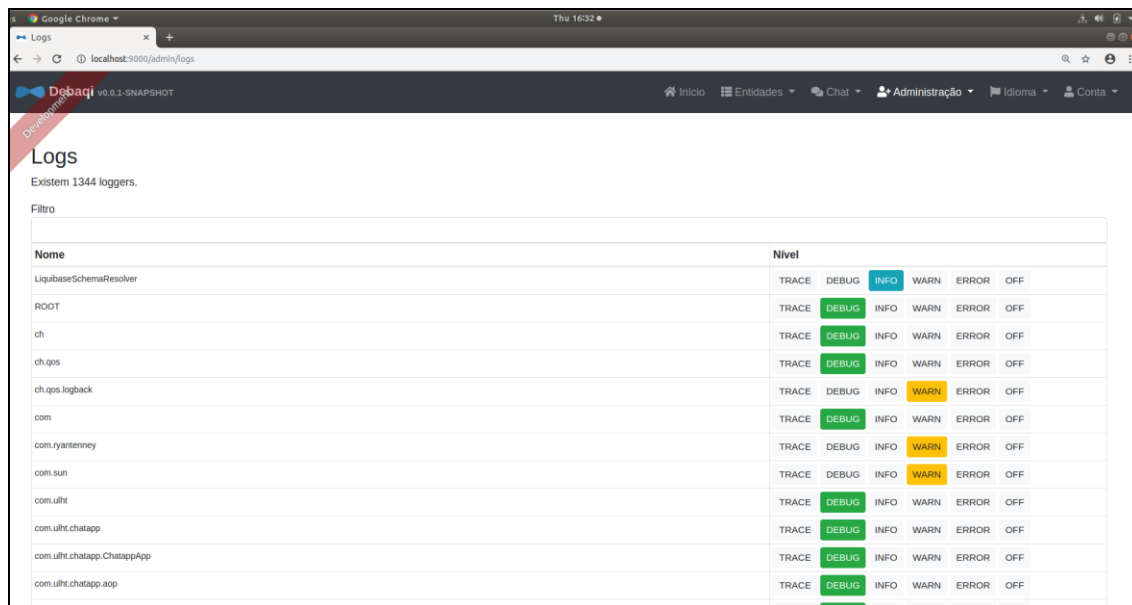


Figura 33: «Print screen» da página que possibilita efetuar a gestão do nível de «logging» na aplicação

A correta utilização de *logs* em aplicações é muito importante, principalmente no cenário em que vivemos atualmente, onde existem inúmeras interligações entre sistemas. Assim, é importante saber quais as informações transmitidas de um sistema para o outro, bem como saber se os dados chegaram corretamente.

Para a correta integração de instruções de «logging» em aplicações é necessário ter bem claro os conceitos de *log*. Quais as informações que deverão ser «logadas»? Qual é o melhor formato para armazenar estes dados? Hoje em dia existem inúmeras soluções de «logging» para Java no mercado. Para este projeto foi utilizado a *framework* log4j da Apache.

Na aplicação *Debaqi* o painel de administração de *logs* permite gerir em tempo de real a configuração do «logback», com podemos verificar no ecrã representado na Figura 33. Aqui o administrador do sistema tem a possibilidade de alterar o nível de «logging» de um determinado pacote Java para *TRACE*, *DEBUG*, *INFO*, *WARN*, *ERROR* ou *OFF* de forma tão simples como um clique num botão. Algo extremamente vantajoso tanto no desenvolvimento como em produção.

3.7.20 API – Swagger

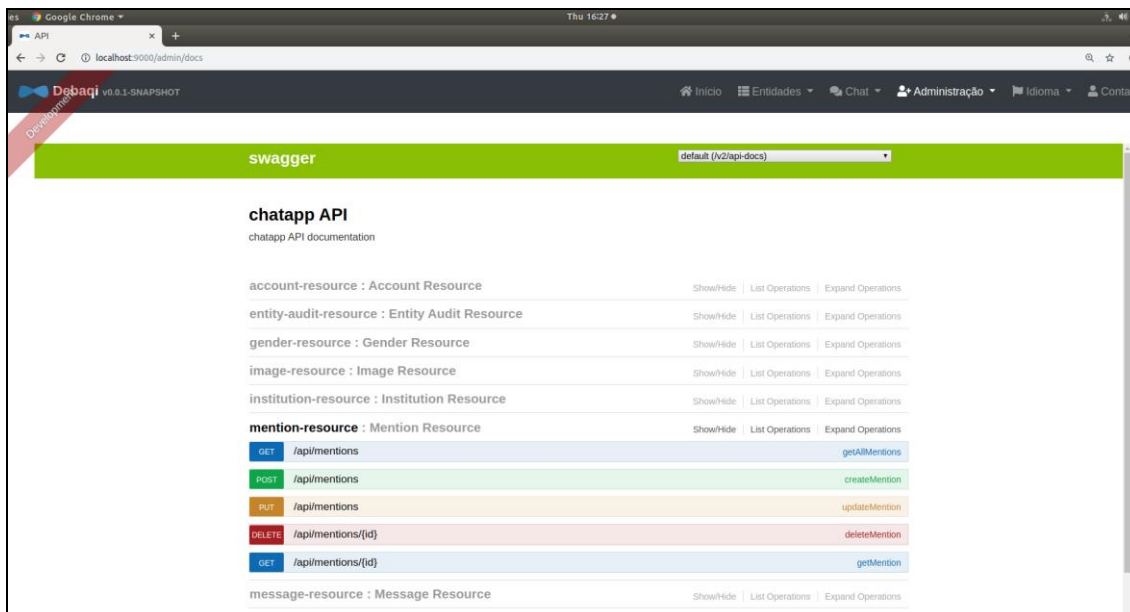


Figura 34: «Print screen» da ferramenta de UI - Swagger

O ecrã representado na Figura 34 demonstra a página que habilita o administrador do sistema aceder ao Swagger. Esta é uma tecnologia «*open-source*» sustentada por uma série de ferramentas que ajudam os programadores a desenhar, criar, documentar e consumir serviços *Web RESTful*.

Apesar da maioria dos utilizadores conhecer o Swagger pela sua ferramenta de UI, esta tecnologia também inclui uma série de funcionalidades de suporte, como a possibilidade de criação de documentação automatizada, geração de código e de casos de teste.

A utilização da tecnologia Swagger no contexto da aplicação *Debaqi* foi efetuada maioritariamente para testar os serviços REST.

3.7.21 Notificações – *Toasters*



Figura 35: «Print screen» das notificações de informação presentes na aplicação

Quando desenvolvemos *software* é sempre boa prática exibir uma mensagem/notificação específica sobre o que o utilizador está a fazer. A aplicação *Debaqi* não foge à regra. Como estamos a trabalhar num âmbito sensível que envolve alunos adolescentes, menores de idade que exprimem o que pensam *online*, foi desenvolvido um motor de notificações que alerta os mesmos conforme se encontrem numa área privada ou numa área pública, tal como ilustra a Figura 35. Quando os alunos se encontram numa área privada as suas interações não são visíveis a terceiros, mas quando estes acedem a áreas públicas as suas interações passam a ser visíveis por todos.

Para o desenvolvimento do motor de notificações recorreremos à tecnologia *angular-toastr*: uma biblioteca que pode ser integrada na *framework* Angular.

3.8 Estrutura e manipulação de dados na aplicação *Debaqi*

Neste ponto considerámos pertinente expor a maneira como definimos toda a estrutura de dados (cf. apêndice I, II e III) para a aplicação, bem como elucidar de uma perspetiva técnica como está a ser realizada a transferência dos mesmos para o servidor de base de dados.

Após uma análise exaustiva realizada ainda numa fase inicial do projeto, tornou-se evidente que a aplicação *Debaqi* iria requerer uma estrutura de dados complexa, «elástica», devidamente «modelada», e mais importante, que todos os atributos identificados teriam de possibilitar alcançar os objetivos pretendidos para este estudo. Perdemos algum tempo nesta fase, mas o resultado final foi bastante satisfatório. Durante as diversas atividades e sessões de teste não foram encontrados quaisquer defeitos na estrutura implementada, acabando por permanecer inalterada ao longo das atividades.

Para além da preocupação em ter uma estrutura de dados íntegra, também ficou delineado que qualquer transação entre a base de dados e o servidor de *back-end* não seria realizada através da codificação de «queries» SQL via ligação JDBC. De forma a responder a este impasse optámos pela utilização de ORM's para os respetivos servidores de *back-end* (Python e Spring Boot).

Um ORM é uma *framework* que permite mapear um modelo orientado a objetos para um modelo relacional de base de dados. A ideia por detrás desta ferramenta passa por dar aos programadores a possibilidade de codificar «queries» na sua linguagem de programação preferencial, utilizando um paradigma orientado a objetos. Isto significa que não é necessário utilizar linguagem SQL para conseguir realizar uma operação CRUD sobre a base de dados.

Como decidimos desenvolver servidores de *back-end* em tecnologias distintas, inevitavelmente a *framework* de ORM também acabaria por ser diferente em cada um deles. Entendemos que o ORM Flask SQLAlchemy¹¹ reunia o mais completo conjunto de funcionalidades para o servidor Python e no caso do servidor Spring optámos pela «famosa» tecnologia Hibernate¹². Para além das vantagens já referidas na utilização de um ORM, acrescenta-se a possibilidade de poder configurar a gestão de transações de forma totalmente automatizada e a abstração do «vendor» de base de dados, ou seja, caso seja necessário realizar a migração de um PostgreSQL para um MySQL esta mudança torna-se extremamente fácil e rápida de executar.

¹¹ [Consult. 29-11-2019]. Disponível em WWW:<URL:<https://flask-sqlalchemy.palletsprojects.com/en/2.x/>>

¹² [Consult. 29-11-2019]. Disponível em WWW:<URL:<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#reference/>>

De forma a exemplificar como as operações CRUD e os objetos relacionais supramencionados são codificados, no Quadro 4 encontram-se alguns exemplos reais desenvolvidos para a aplicação.

Fonte: Elaboração própria

ORM Query	ORM	SQL Query
<code>Message.query.filter_by(message_id=str(data['messageId'])).first_or_404()</code>	Flask SQLAlchemy	<code>SELECT * FROM message WHERE message_id = (parameter)</code>
<code>List<Room> findByStatus(RoomStatus status);</code>	Hibernate	<code>SELECT * FROM room WHERE status = (parameter)</code>
<code>msg = Message(roomDb.id, data['messageId'], data['user']['userId'], data['messageText'], data['time']...) db.session.add(msg) db.session.commit()</code>	Flask SQLAlchemy	<code>INSERT INTO message VALUES ('id', 'messageId', 'userId'...); ou UPDATE message SET id = value, messageId = value, ... WHERE condition;</code>
<code>Optional<Room> saveAndFlush(Room room);</code>	Hibernate	<code>INSERT INTO room VALUES ('id', 'messageId', 'userId'...) ou UPDATE message SET id = value, messageId = value, ... WHERE condition;</code>
<code>db.session.delete(msg) db.session.commit()</code>	Flask SQLAlchemy	<code>DELETE FROM message WHERE id = value;</code>
<code>void deleteById(Room room)</code>	Hibernate	<code>DELETE FROM room WHERE id = value;</code>

Quadro 4: As diferentes funções disponíveis em cada ORM e a sintaxe SQL que estas «geram»

No quadro 4 conseguimos verificar como é realizada a integração de código Java e Python de maneira a «gerarem» diferentes operações SQL. Outro aspeto importante a ser esclarecido é a maneira como o ORM consegue distinguir se deve fazer um INSERT ou um UPDATE utilizando a mesma função para ambas as operações (*saveAndFlush(Entity parameter)*). De facto, isso é realizado de uma forma bastante simples. Caso se pretenda armazenar na base de dados um objeto que contenha o atributo ID preenchido, então o ORM sabe que tem de executar uma operação de UPDATE, caso contrário um INSERT. O DELETE e o SELECT são funções com finalidades exclusivas.

Fonte: Elaboração própria

Modelo relacional de base de dados em ambas <i>frameworks</i> de ORM	
Hibernate	<pre> @Entity @Table(name = "room") public class Room extends AbstractAuditingEntity implements Serializable { private static final long serialVersionUID = 1L; @Id @GeneratedValue(strategy = GenerationType.IDENTITY) private Long id; @NotNull @Column(name = "room_id", nullable = false) private String roomId; @NotNull @Enumerated(EnumType.STRING) @Column(name = "status", nullable = false) private RoomStatus status; @NotNull @Column(name = "start_date", nullable = false) private LocalDate startDate; @NotNull @Column(name = "end_date", nullable = false) private LocalDate endDate; @ManyToOne @JsonIgnoreProperties("rooms") private Topic topic; @OneToMany(mappedBy = "room", cascade = CascadeType.REMOVE) private Set<Message> messages = new HashSet<>(); @OneToMany(mappedBy = "room", fetch = FetchType.EAGER, cascade = CascadeType.ALL, orphanRemoval = true) private Set<RoomReactions> roomReactions = new HashSet<>(); </pre>
Flask SQLAlchemy	<pre> class Message(BaseModel): __tablename__ = 'message' room_id = db.Column(db.BigInteger, nullable=False) message_id = db.Column(db.String(255), nullable=False) user_id = db.Column(db.BigInteger, nullable=False) message_text = db.Column(db.String(255)) time = db.Column(db.DateTime, nullable=False, default=datetime.utcnow) created_by = db.Column(db.String(255)) last_modified_by = db.Column(db.String(255)) image_id = db.Column(db.BigInteger, db.ForeignKey('image.id')) #Defining One to One relationships with the relationship function on the Parent Table image = db.relationship('Image', backref='message') #Defining One to Many relationships with the relationship function on the Parent Table reaction = db.relationship('Reaction', backref='message', cascade="all, delete-orphan", lazy='dynamic') mention = db.relationship('Mention', backref='message', cascade="all, delete-orphan", lazy='dynamic') </pre>

Quadro 5: Exemplo de código desenvolvido para cada ORM, de forma a implementar um modelo relacional de base de dados

No Quadro 5 conseguimos verificar como é possível representar um modelo relacional de base de dados por meio de «classes». Estas são uma abstração de uma tabela de base de dados e que permitem que a sintaxe «nativa» da linguagem de programação escolhida represente atributos, tipologias, chaves primárias e estrangeiras, cardinalidades e «*constraints*».

De forma a concluir a temática dos ORM's gostaríamos de elucidar que para um grande número de pessoas na comunidade de desenvolvimento aplicacional, os ORM's são de facto vistos como ferramentas positivas que efetivamente aceleram todo o processo de desenvolvimento de *software* (Bozhanov, 2012)¹³, mas para os mais reticentes, esta afirmação provoca alguma incerteza (Atwood, 2006)¹⁴.

¹³ [Consult. 04-12-2019]. Disponível em WWW:<URL:<https://techblog.bozho.net/orm-haters-dont-get-it/>>

¹⁴ [Consult. 04-12-2019]. Disponível em WWW:<URL:<https://blog.codinghorror.com/object-relational-mapping-is-the-vietnam-of-computer-science/>>

Tendo agora uma breve noção do que é um ORM, das suas vantagens e de como esta tecnologia foi integrada na aplicação *Debaqi*, podemos então começar a analisar de forma mais pormenorizada as estruturas de dados elaboradas por meio de diagramas de «classes». Nas próximas páginas serão ilustradas diversas figuras que demonstram a forma como o modelo de dados está organizado pelas diversas funcionalidades. Sendo o ORM o principal intermediário de comunicação entre a base de dados e a aplicação, para a seguinte análise, as imagens expostas ilustram as diversas entidades¹⁵ desenvolvidas tanto para o Hibernate, como para o SQLAlchemy. Estas reproduzem uma abstração por meio de «classes» das diversas tabelas existentes na base de dados (Quadro 5). Optámos por esta representação esquemática, devido à possibilidade de serem retratados outros artefactos como as enumerações. Neste caso, as enumerações inseridas no cabeçalho de cada atributo irão impossibilitar que o mesmo obtenha outro valor para além dos existentes na enumeração associada.

Fonte: Elaboração própria

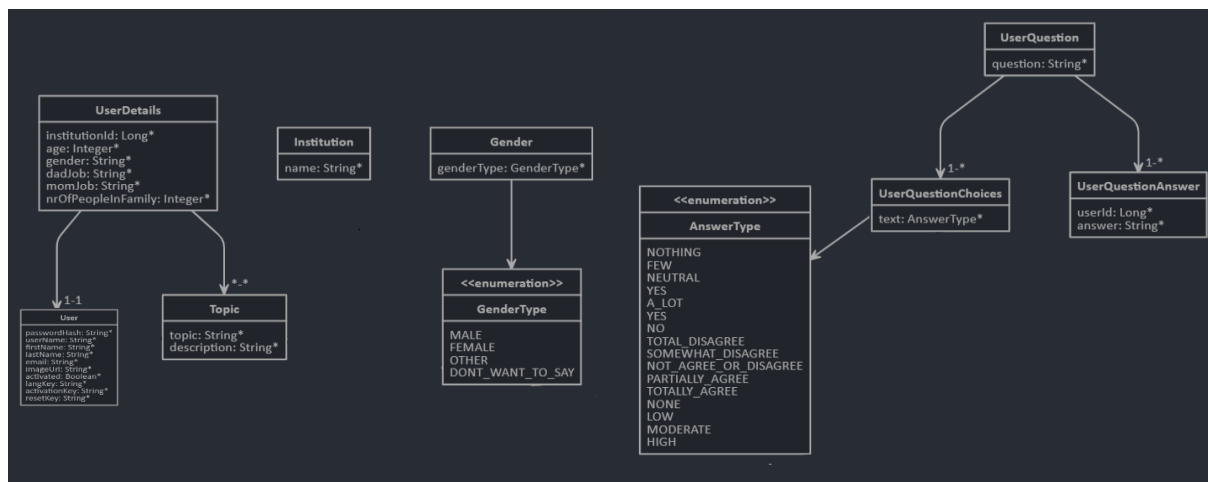


Figura 36: Diagrama de «classes» alusivo às entidades que suportam efetuar o registo do aluno

Na Figura 36 conseguimos verificar o diagrama de «classes» que demonstra as entidades desenvolvidas de forma a suportarem o registo de alunos na aplicação. De seguida, iremos efetuar uma breve análise das entidades representadas nos respetivos diagramas, seguindo o sentido da esquerda para a direita.

Primeiramente conseguimos verificar a existência de uma ligação de 1-1 «lazy»¹⁶ entre a entidade *User* e *UserDetails*. Efetuámos esta diferenciação em duas entidades de modo a

¹⁵ Uma entidade representa uma tabela de uma base de dados relacional, onde cada instância da mesma corresponde a uma linha da tabela. O artefacto de programação primário de uma entidade é uma classe (POO).

¹⁶ «Lazy loading» é um padrão de *design* usado para adiar a inicialização de um objeto. O objeto só é inicializado quando solicitado. [Consult. 06-12-2019]. Disponível em WWW:<URL:<<https://www.baeldung.com/hibernate-lazy-eager-loading>

separar o que são atributos específicos das características físicas e sociológicas do aluno das restantes que são utilizadas para validações técnicas do sistema, como por exemplo, o «*username*», a «*passwordHash*», etc. Desta maneira, sempre que exista a necessidade de aceder à informação de um utilizador, temos a possibilidade de decidir que entidade realmente necessitamos, mitigando o transporte constante de dados desnecessários e «sensíveis». Adicionalmente, existe inclusivamente a possibilidade de adicionar permissões de segurança ao nível do perfil para qualquer entidade.

No centro da Figura 36 encontramos as entidades *Institution* e *Gender*. Estas entidades de momento exercem um papel meramente de apoio, mas dão «elasticidade» futura ao sistema, ou seja, podíamos ter os atributos que as constituem «*hard-coded*» no código da aplicação, mas optámos por armazená-los de forma relacional. Futuramente, caso haja necessidade de acrescentar mais atributos sobre a Escola, como por exemplo, a morada, o código-postal, os contatos, etc., o sistema já está apto para integrar a nova informação sem necessidade de «*rework/reengineering*» do ecrã de registo. Relativamente a entidade *Gender*, gostaríamos também de referir que, devido ao facto de o atributo identificador do género ser um enumerado, não será possível que este obtenha qualquer outro valor para além daqueles que podem ser observados na enumeração *GenderType*.

Por fim, encontramos as entidades que possibilitam disponibilizar ao aluno o questionário de registo. A entidade *UserQuestion* tem por objetivo guardar o texto que constitui a questão. Realizando um «*drill-down*» além desta entidade encontramos mais duas: a *UserQuestionChoices* «1..*, *eager*»¹⁷ e a *UserQuestionAnswer* «1..*, *lazy*». A entidade *UserQuestionChoices* permite através de diversos enumerados parametrizar todas as respostas possíveis para uma questão. A entidade *UserQuestionAnswer*, por outro lado, vai guardar o valor do enumerado selecionado como resposta à questão, bem como o ID do utilizador que a realizou. A cardinalidade desta entidade é de «1..*» de forma a se encontrar apta para formular questões de resposta múltipla.

¹⁷ «*Eager loading*» é um padrão de *design* no qual a inicialização de dados ocorre imediatamente. [Consult. 06-12-2019]. Disponível em WWW:<URL:<https://www.baeldung.com/hibernate-lazy-eager-loading>>

Fonte: Elaboração própria

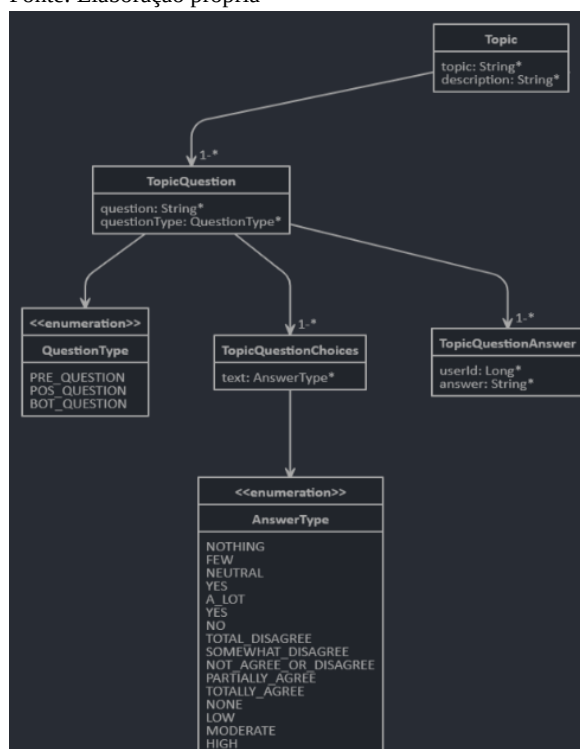


Figura 37: Diagrama de «classes» das entidades pertencentes aos questionários pré-debate, pós-debate e questões do *chatbot*

Na Figura 37 encontramos o diagrama de «classes» que possibilita a parametrização dos questionários pré e pós debate, como também as questões que são lançadas automaticamente pelo o *chatbot*.

Este diagrama é muito similar à estrutura já descrita para o questionário do utilizador diferindo apenas em dois pontos. A sua ligação com a entidade *Topic* e a adição de um atributo enumerado *QuestionType* que permite distinguir qual o tipo de questão que desejamos colocar. Como já referido anteriormente, a partir do momento em que os alunos respondem a todas as perguntas referentes a um tópico, se estes «navegarem» pelas diversas salas de debate relativas à mesma temática as perguntas não serão novamente apresentadas. Isto torna-se possível através da ligação entre as entidades *TopicQuestions* «1..*» e *Topic*, isto é, uma temática terá sempre *N* questões associadas. Podemos exemplificar o «*workflow*» do sistema de questionários da forma que a seguir se descreve. No momento em que um aluno tente aceder a uma sala de debate, o sistema vai verificar na base de dados se para todas as perguntas relativas à temática associada à sala existem as devidas respostas de acordo com o ID do utilizador. Caso existam perguntas por responder a aplicação irá apresentar as questões em falta, caso contrário o sistema navegará diretamente para a página solicitada.

Fonte: Elaboração própria

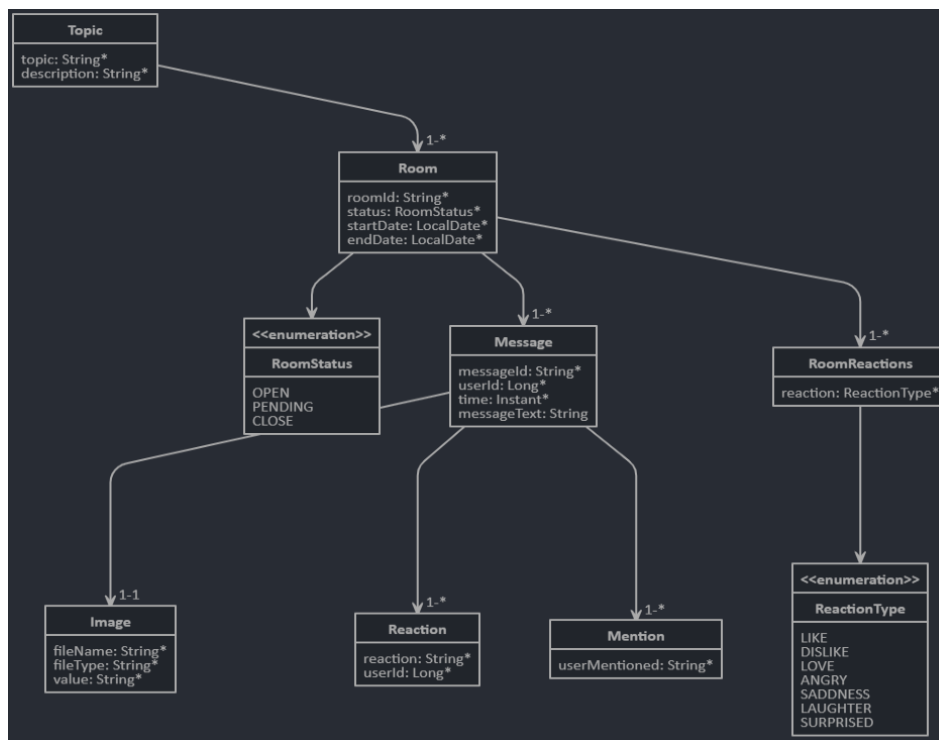


Figura 38: Diagrama de «classes» referente às entidades que suportam as interações no *chat*

Para concluir a análise dos diversos diagramas existentes na aplicação, importa assinalar, como ilustrado na Figura 38, a estrutura que possibilita armazenar todas as interações realizadas no *chat*. Como verificámos anteriormente, um *Topic* tem uma cardinalidade de «1..*» com a entidade *Room*. Relativamente a esta entidade é importante referir que existe a possibilidade de serem parametrizadas diferentes reações para cada sala de debate, o que permite realizar diversos estudos. As reações disponíveis são armazenadas na entidade *RoomReactions* que, por sua vez, estabelece uma cardinalidade de «*..1» com a entidade *Room*. De momento a aplicação apenas permite parametrizar as reações presentes no enumerado *ReactionType*. Adicionalmente às reações, a entidade *Room* contém um atributo enumerado que define o estado em que se encontra a sala de debate. Os diferentes estados visíveis na enumeração *RoomStatus* vão informar se a sala já se encontra disponível para dar início à atividade.

Como em qualquer *chat* a troca de mensagens deve ser realizada em tempo real, de forma a promover uma comunicação efetiva e clara entre todos os utilizadores bem como dar a possibilidade de estes se exprimirem através de diversos meios como textos, imagens, *emojis*, *etc*. O *Debaqi* não foge a regra e como já verificamos no ponto 3.7.12 (p.69) existem múltiplas formas de interação entre os alunos durante as atividades.

A entidade *Message* é a grande responsável por centralizar todas as interações realizadas no *chat*, como aquela na qual foi aplicada um maior número de técnicas de «*data science*». Nela estão contidos atributos como o texto e o «*timestamp*» referentes a uma mensagem enviada para o *chat*. Para que possamos associar todas as reações pertencentes a uma mensagem, a entidade *Reaction*, estabelecendo uma cardinalidade de «**..1*» com a entidade *Message*, vai ser responsável por armazenar os enumerados das diversas reações atribuídas, bem como os utilizadores que as realizaram. Desta forma, conseguimos efetuar uma análise qualitativa referente aos dados recolhidos.

Utilizando a mesma algoritmia desenvolvida para as reações, a entidade *Mention* tem por objetivo guardar todas as menções presentes numa mensagem. Caso a aplicação detete o caractere especial que reconhece a menção (@), esta irá então extrair todos os *UserNames* mencionados para serem armazenados na base de dados.

Por fim, estabelecendo uma cardinalidade de «*1..1*» encontramos a entidade *Image*. Caso a aplicação detete conteúdo multimédia «lançado» no chat, esta entidade será responsável por guardar esse mesmo conteúdo «comprimido» em *base64*¹⁸ – algoritmo criptográfico reversível.

3.9 DevOps – *Debaqi*

A prática de *DevOps* consiste na criação de um conjunto de automatizações processuais entre o desenvolvimento de *software* e as equipas de *IT*, para que estas possam desenvolver, testar e «libertar» *software* de forma mais rápida e segura.

O conceito do *DevOps* baseia-se na construção de uma cultura de colaboração entre equipas, que anteriormente trabalhavam separadas, em sítios e fases de desenvolvimento diferentes. As vantagens evidenciadas pela adoção desta cultura emergente, incluem um aumento na confiança do produto, lançamentos mais rápidos de *software*, maior capacidade de resolver rapidamente problemas críticos e melhorias na gestão do trabalho não planeado.

Todas as ferramentas e automatização existentes hoje em dia são inúteis se não forem acompanhadas pelo desejo genuíno por parte dos profissionais de desenvolvimento e de *IT/Ops* em trabalharem juntos. O *DevOps* não resolve problemas de ferramentas, resolve problemas

¹⁸ Algoritmo criptográfico de chave simétrica. Usa a mesma chave criptográfica para encriptação e descriptação de texto simples.

das pessoas. Se pudéssemos resumir a cultura *DevOps* numa palavra seria «colaboração» e em duas seria «colaboração polivalente».

Cada vez mais as empresas bem-sucedidas estão a adotar uma cultura *DevOps* em todos os seus departamentos e níveis organizacionais. Estas empresas têm canais abertos de comunicação e promovem a conversação regular. Garantem que os objetivos de todos estão alinhados e ajustados conforme a necessidade e, principalmente, assumem que manter os clientes satisfeitos é uma responsabilidade tanto da gestão, como da equipa de desenvolvimento. Assim, entendem que o *DevOps* não é o trabalho de uma só equipa, mas sim o trabalho de todos.

Uma das principais áreas da cultura *DevOps* é a automatização de processos. Investir em automatização elimina o trabalho manual repetitivo, produz processos repetíveis e cria sistemas confiáveis. As recentes equipas de automatização, por norma, começam em primeiro lugar a trabalhar no processo de «*continuous delivery*». Esta é uma prática que visa «entregar» cada alteração realizada no código por meio de uma série de testes automatizados, geralmente executados numa infraestrutura *Cloud*. Aqui, vão ser gerados pacotes de compilações bem-sucedidas para serem instalados em qualquer ambiente, por meio de integrações automatizadas. Como demonstrado, todos os passos necessários para a criação de um processo de «*continuous delivery*» não são rápidos nem fáceis de configurar, mas o tempo investido neste desenvolvimento traz vantagens a longo prazo. Porquê? Porque os computadores executam testes com maior rigor e fidelidade que os Seres Humanos. A execução destes testes ajuda a detetar erros e falhas de segurança mais rapidamente, permitindo assim que os programadores os corrijam com maior facilidade.

A adoção de uma cultura *DevOps* foi uma decisão essencial para agilizar os processos de desenvolvimento e integrações da aplicação *Debaqi*. Através dela obteve-se o tempo necessário para efetuar todos os desenvolvimentos que foram exigidos ao longo do projeto piloto.

Durante o calendário de atividades foram realizados inúmeros tipos de testes e modificações ao código da aplicação. De maneira a integrar essas alterações de forma rápida e segura, foram desenvolvidas três «*pipelines*», uma para cada servidor representado na Figura 9. Cada «*pipeline*» tem por objetivo realizar a integração automática do novo código alojado no repositório *git* para a *Cloud* Heroku. Esta integração é executada sempre que é efetuado um novo «*commit*» no «*master branch*» ou de forma recorrente, todos os dias à meia noite. Assim garante-se um maior foco no desenvolvimento e na qualidade do *software* disponibilizado aos alunos.

Na imagem representada na Figura 39 conseguimos observar o código desenvolvido para a «*pipeline*» que integra automaticamente o código do servidor Flask (Python) na *Cloud*. O executar deste «*pipeline*» vai gerar uma imagem Docker, de seguida instalar todas as dependências do projeto e por fim «entregar» o pacote compilado na *Cloud* Heroku.

```
6 image: python:3.6.8
7
8 clone:
9   depth: full
10 pipelines:
11   default:
12     - step:
13         caches:
14         - pip
15         script: # Modify the commands below to build your repository.
16                 - pip install -r requirements.txt
17                 - git push https://heroku:$HEROKU_API_KEY@git.heroku.com/$HEROKU_APP_NAME.git HEAD:master
18
```

Figura 39: Código desenvolvido para a *pipeline* de integração automática com a *Cloud* Heroku
Fonte: Elaboração própria

No ecrã representado na Figura 40, conseguimos visualizar a página responsável por mostrar o historial de integrações. Aqui conseguimos verificar se estas foram executadas com ou sem sucesso. Caso o «*pipeline*» falhe a meio da integração a aplicação manterá a versão anterior e a equipa de desenvolvimento será alertada para o sucedido, de forma a realizar a correção do defeito que provocou o insucesso da integração.

Para a aplicação *Debaqi* foi escolhida como plataforma de *git* e «*continuous delivery*» o Bitbucket da Atlassian. Esta solução incorpora de forma gratuita e «*out of the box*» todas as funcionalidades necessárias para serem desenvolvidas aplicações de acordo com a cultura *DevOps*.

Fonte: Elaboração própria

thesulht / chatapp / chatapp-brain

Pipelines

What's new

Run pipeline

Schedules

Caches

Usage

All branches

OTHER BRANCHES

master MAIN BRANCH

Status

Trigger type

Only mine

Pipeline	Status	Started	Duration
#74 fix to uri Pedro de Almeida Perdigão · c83910b ↗ master	Successful	4 months ago	47 sec
#73 changes to new model Pedro de Almeida Perdigão · 4552728 ↗ master	Successful	4 months ago	1 min 21 sec
#72 wordlist Pedro de Almeida Perdigão · 1ea2386 ↗ master	Successful	5 months ago	50 sec
#71 wordlist Pedro de Almeida Perdigão · 3058c93 ↗ master	Successful	5 months ago	56 sec
#70 wordlist Pedro de Almeida Perdigão · 3e98eda ↗ master	Successful	5 months ago	48 sec

Figura 40: «*Print screen*» da funcionalidade «*Pipeline*» do Bitbucket, onde é apresentada a informação sobre o sucesso e insucesso das integrações automáticas do servidor Flask (Python)

CAPÍTULO IV

Análise dos dados

IV – ANÁLISE DOS DADOS

4.1. – O conhecimento que pretendemos obter

Os Seres Humanos são uma espécie muito curiosa. Faz parte da natureza Humana questionar e refletir sobre tudo o que nos rodeia. Esta indagação incessante tem vindo a permitir que ao longo de gerações tenhamos conseguido adquirir um vasto conhecimento sobre as mais diversas áreas. É interessante igualmente constatar que, apesar da tecnologia que hoje em dia se encontra presente nas nossas vidas, existem inúmeros fenómenos que não conseguimos ainda compreender e outros tantos que não temos a menor ideia que possam sequer existir. Esta é a motivação: a procura pelo desconhecido. Para que possamos dar início a esta «cruzada» devemos começar por questionar algo que observarmos.

Numa investigação científica, o modo como formulamos uma questão que carece de uma resposta é extremamente importante para delinear o âmbito da própria pesquisa, adquirir novos conhecimentos e resolver um mistério sobre o Mundo Natural. Por esse motivo, devemos ter sempre em mente as principais etapas de um processo científico: 1) a observação de um fenómeno 2) o questionamento 3) a formulação de uma hipótese/teoria. Para isso, deve ser definido o conjunto de dados que pretendemos recolher e, posteriormente, selecionar as técnicas adequadas para os analisar, tal como ilustram as figuras 41, 42 e 43. Por último, com os novos conhecimentos adquiridos devemos desenvolver uma conclusão que possa servir de resposta a uma determinada pergunta. Abaixo, encontram-se algumas das questões às quais pretendemos dar resposta no âmbito desta investigação:

- ❖ Que temas interessam aos jovens?
- ❖ Como é que os jovens expressam os seus pontos de vista?
- ❖ Que tipo de linguagem utilizam?
- ❖ Os debates são heterogéneos ou polarizados? Há uma participação ativa ou limitam-se a uma reação mínima (sim/não, reações, *emojis*, etc.);
- ❖ Reatividade emocional vs. deliberação e raciocínio;
- ❖ Diálogos baseados em ataques e defesas;
- ❖ Consideração das ideias dos outros vs. manter pontos de vista;
- ❖ Alinhamento com o tema do debate.

Encontrar respostas a este tipo de perguntas tem sido o foco de muitas disciplinas como por exemplo, as Ciências Sociais, a Psicologia, a Linguística, a Antropologia e outras áreas científicas. Recentemente, os avanços na Ciência dos Dados e em Inteligência Artificial têm vindo a permitir a criação de novas correntes interdisciplinares, como é o caso da Ciência Social Computacional e da Linguística Computacional, sendo estas o principal enquadramento da investigação desenvolvida. Um dos elementos mais importantes neste contexto é a transformação de grandes volumes de dados em estruturas computacionais que capturam características importantes implícitas nos dados fonte. No caso da linguagem natural, a nossa abordagem consiste na representação vetorial de intervenções individuais em diálogos e debates.

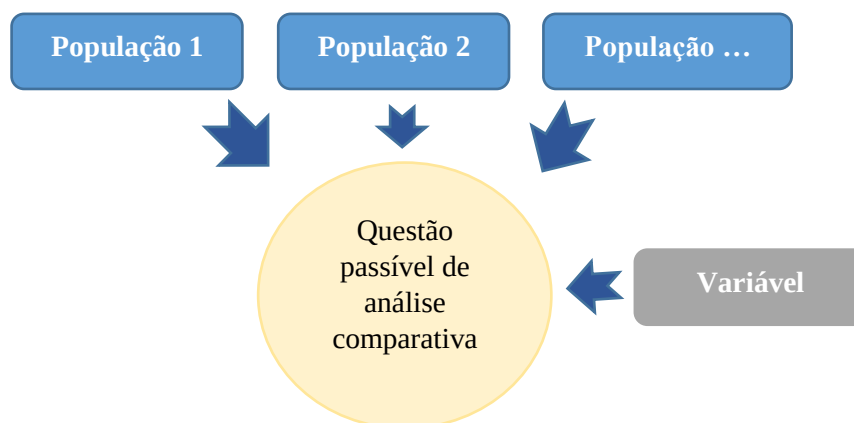


Figura 41 – Processo de análise comparativa

Fonte: Elaboração própria

Resumo: Em processos de análise comparativa, procura-se estabelecer relações causa-efeito e possíveis antecedentes para acontecimentos verificados num passado mais ou menos longínquo. Para isso este tipo de pesquisa, deve-se sempre incluir duas ou mais populações sujeitas a condições distintas e uma variável mensurável.

As Ciências baseadas em grandes coleções de dados permitem realizar investigações comparativas semelhantes às versões clássicas que envolvem a recolha de dados pertencentes a populações que estão sujeitas a condições diferentes. Estas pesquisas devem incluir sempre duas ou mais populações e uma variável medida, como ilustra a Figura 41. Abaixo, podemos verificar alguns exemplos de perguntas passíveis de análise comparativa:

- ❖ Existe uma diferença substancial de altura entre Europeus e Australianos?
- ❖ Existe diferença na diversidade de fungos provenientes de áreas florestadas em comparação com áreas não-florestadas?

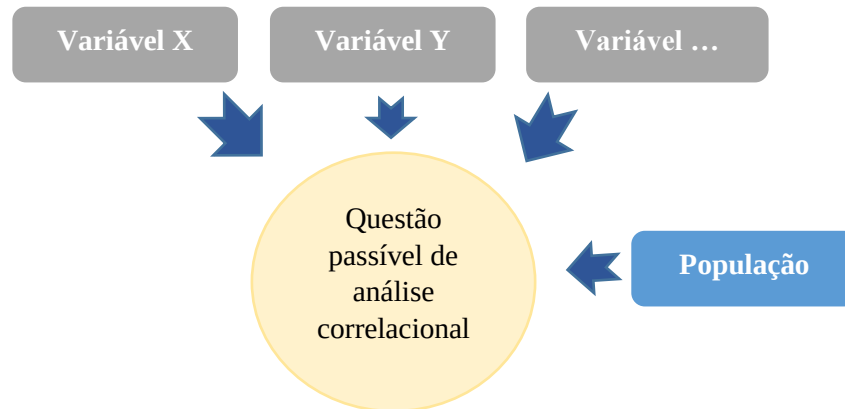


Figura 42 – Processo de análise correlacional

Fonte: Elaboração própria

Resumo: As análises correlacionais permitem estabelecer e quantificar relações entre uma multiplicidade de variáveis simultaneamente. Esta tem por objetivo estabelecer relações entre as variáveis pertencentes ao estudo, como também calcular o comportamento de uma das variáveis a partir do comportamento de outra.

Uma das principais vantagens em recolher grandes volumes de dados é a possibilidade de serem efetuadas investigações correlacionais. Estas envolvem medir ou observar duas ou mais variáveis e procurar um relacionamento entre as mesmas e uma determinada população com um elevado grau de confiança estatística nos resultados obtidos. Esta análise torna-se passível de representação esquemática através da imagem representada pela Figura 42. Abaixo, podemos adicionalmente verificar alguns exemplos de perguntas passíveis de análise correlacional:

- ❖ O grau de satisfação dos alunos que se encontram na Licenciatura em Engenharia Informática e Sistemas de Informação varia de acordo com o semestre?
- ❖ Como «flutua» o valor de uma moeda face a uma eleição?

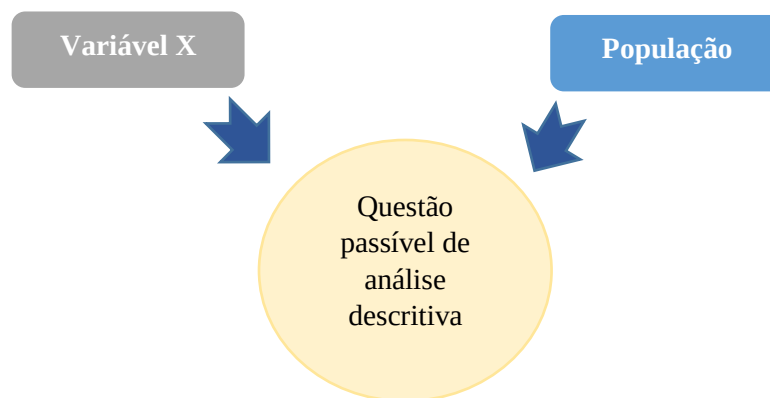


Figura 43 – Processo de análise descritiva

Fonte: Elaboração própria

Resumo: Os estudos descritivos têm por objetivo especificar as propriedades, características e perfis importantes de uma população ou fenómeno. As medidas de estatística descritiva, permitem sintetizar os dados de uma população ou amostra através de um só valor. As medidas descritivas mais utilizadas são: medidas de localização, medidas de dispersão, medidas de assimetria e medidas de concentração.

As investigações descritivas implicam a descrição e/ou quantificação das partes de um determinado ecossistema. Normalmente, inclui uma população e uma variável distinta, como ilustra a Figura 43. De seguida, verificamos alguns exemplos de perguntas passíveis de análise descritiva:

- ❖ Quantos pinheiros existem na área da serra de Sintra?
- ❖ Qual é o alcance de distribuição de uma matilha de lobos?
- ❖ Com que frequência a abelha asiática se reproduz?

4.2– Os objetivos pretendidos com os novos conhecimentos

Em trabalhos de natureza científica os resultados obtidos destinam-se a preencher uma lacuna no conhecimento de uma determinada área, tal como é o propósito central desta dissertação. Através dos diversos resultados alcançados durante as sessões de trabalho temos como objetivo caracterizar a população alvo do estudo em termos dos fatores que podem ter impacto sobre a «saúde» do diálogo *online*. Pretendemos ainda efetuar uma análise dos fatores identificados, assim como as suas interações, de modo a informar o desenho das intervenções educacionais. Por fim, é nossa intenção analisar e identificar intervenções educativas baseadas na inteligência artificial e na ciência dos dados, que podem apoiar efetivamente o Professor Bibliotecário.

4.3 – Definição operacional de um «diálogo saudável»

Uma das linhas de investigação mais produtivas dentro da ciência dos dados é o processamento estatístico de linguagem natural (NLP). Os avanços nesta área têm permitido que organizações como a Google e o Facebook consigam derivar conhecimento semântico válido e útil a partir do processamento estatístico de textos. Um aspeto importante destes métodos é a representação de texto usando vetores que capturam a distribuição de frequência de palavras. O método utilizado neste projeto usa a representação vetorial de textos como base para analisar fatores e variáveis relacionadas com o diálogo saudável. O procedimento para transformar textos em vetores é simples. Numa fase preliminar, define-se um conjunto de textos que existem num contexto temático comum. Tal conjunto é designado como «*corpus*» na disciplina de linguística computacional. Este pode ser, por exemplo, constituído por receitas de cozinha, ou por um arquivo de notícias, ou por transcrições de debates parlamentares, etc. Neste projeto existem três «*corpora*», cada um é o conjunto de intervenções de jovens em atividades de debate sobre cada um dos três temas definidos.

A partir do «*corpus*» que se pretende analisar é gerado um vocabulário comum de referência. Este vocabulário deve incluir um conjunto de palavras informativas e abrangentes de todos os textos incluídos no «*corpus*», de forma equilibrada. Para extrair tal vocabulário, combinam-se métodos estatísticos e análises qualitativas para separar termos que tenham uma presença: 1) estatisticamente significativa nos textos 2) que não sejam termos genéricos, mas sim específicos do tema do «*corpus*». Adicionalmente, considerando que as palavras associadas a um conceito comum são usadas com diversidade gramatical, como por exemplo, plásticos, plástica, plástico, etc., substituímos todas estas palavras pela sua raiz morfológica, neste exemplo, «*plás*». Uma vez ultrapassada esta fase do processamento, cada uma das palavras processadas no vocabulário universal do «*corpus*» designa-se «*Token*». Este conjunto de «*tokens*» definirá, posteriormente, as componentes fixas no vetor de referência do «*corpus*», denotado por $\mathbf{r}$. Os valores numéricos para cada componente deste vetor de referência são calculados através de medidas estatísticas que capturam a importância de cada «*token*» no âmbito do «*corpus*» como um todo.

Neste estudo, cada um dos textos que fazem parte dos diversos «*corpora*» a serem analisados, um «*corpus*» para cada tema de debate, corresponde a uma Intervenção individual nesses debates, denotado por $\mathbf{u}(t)$, em que t representa o momento no qual ocorreu a intervenção. Por exemplo, a primeira intervenção no debate será denotada por $\mathbf{u}(1)$. Para representar qualquer intervenção num «*corpus*», é primeiro criada uma cópia do vetor de

referência, com as suas componentes inicializadas a zeros, e substituem-se esses valores com estatísticas relativas à intervenção específica que se pretende representar, tal como explicado em maior detalhe no resto deste ponto. Por exemplo, na Figura 44 podemos observar que o vetor de referência, $\mathbf{r}$, foi construído a partir de palavras provenientes de todas as intervenções recolhidas $\mathbf{u}(i)$, $\mathbf{u}(j)$ e $\mathbf{u}(k)$. Quando o objetivo é obter o vetor para uma intervenção específica, $\mathbf{u}(t)$, os «tokens» que representam palavras que existem nessa intervenção $\mathbf{u}(t)$ geram estatísticas com valores positivos nas componentes correspondentes – vejam-se as componentes a verde nos vetores $\mathbf{u}(i)$, $\mathbf{u}(j)$ e $\mathbf{u}(k)$ da Figura 44. Ou, no caso contrário, geram zeros – componentes a branco nos mesmos vetores da figura. Portanto, para representar qualquer texto, $\mathbf{x}$, dentro de um «corpus», $\mathbf{c}$, é efetuada a seguinte sequência de operações algorítmicas, para atualizar os valores das diferentes componentes da cópia do vetor de referência criada para representar $\mathbf{x}$:

1. Transformar $\mathbf{x}$ numa lista de palavras – «tokens»;
2. Eliminar «tokens» comuns na língua original de $\mathbf{x}$ (pronomes, artigos, conjunções, etc.);
3. Substituir os «tokens» restantes pela sua raiz morfológica;
4. Calcular uma estatística que descreva a importância de cada «token» tanto em $\mathbf{x}$ como no «corpus»;
5. Atualizar os valores das estatísticas para cada «token» presente no vetor criado para $\mathbf{x}$.

A estatística utilizada para quantificar a importância dum «token» ($\mathbf{i}$) num texto ($\mathbf{x}$), que faz parte dum «corpus» ($\mathbf{c}$), correspondente ao quarto passo do algoritmo supramencionado, é conhecido como TF-IDF (Ramos, 2003). Este valor é o produto da multiplicação da frequência normalizada (TF) do «token» em $\mathbf{x}$ pela frequência inversa do mesmo «token» no corpus $\mathbf{c}$ (IDF). O TF é calculado, simplesmente, através do número de vezes que o «token» $\mathbf{o}$ aparece em $\mathbf{x}$, dividido pelo número total de «tokens» em $\mathbf{x}$ (frequência normalizada). O IDF é o logaritmo do resultado da divisão do número total de intervenções no «corpus» pela proporção destas que contêm o «token» $\mathbf{o}$. O IDF tem um efeito modulador muito importante no valor TF-IDF. Por exemplo, se um «token» $\mathbf{o}$ aparecer muitas vezes numa intervenção $\mathbf{x}$ dum «corpus» $\mathbf{c}$, este «token» terá, claramente, um valor TF elevado. Mas se esse mesmo «token» $\mathbf{o}$ aparecer em muitos documentos do «corpus» $\mathbf{c}$, é evidente que o «token» não terá grande valor informativo para separar intervenções do «corpus» que sejam qualitativamente diferentes entre si. De fato, o valor IDF para um «token» $\mathbf{o}$ na situação descrita estará muito perto de zero. Isto ocorre porque o resultado da divisão do número de documentos em $\mathbf{c}$ pelo número de

documentos que contém o «token» **o** estará próximo de um, e o logaritmo desse número tenderá a ser zero. Nesta situação, o valor IDF baixo modulará (diminuirá) o valor TF elevado. Desta forma, a estatística usada para construir a representação vetorial de intervenções dum «corpus» de debate considera a importância dos «tokens» nas intervenções individuais, mas também a relevância dos «tokens» no contexto do «corpus» como um todo.

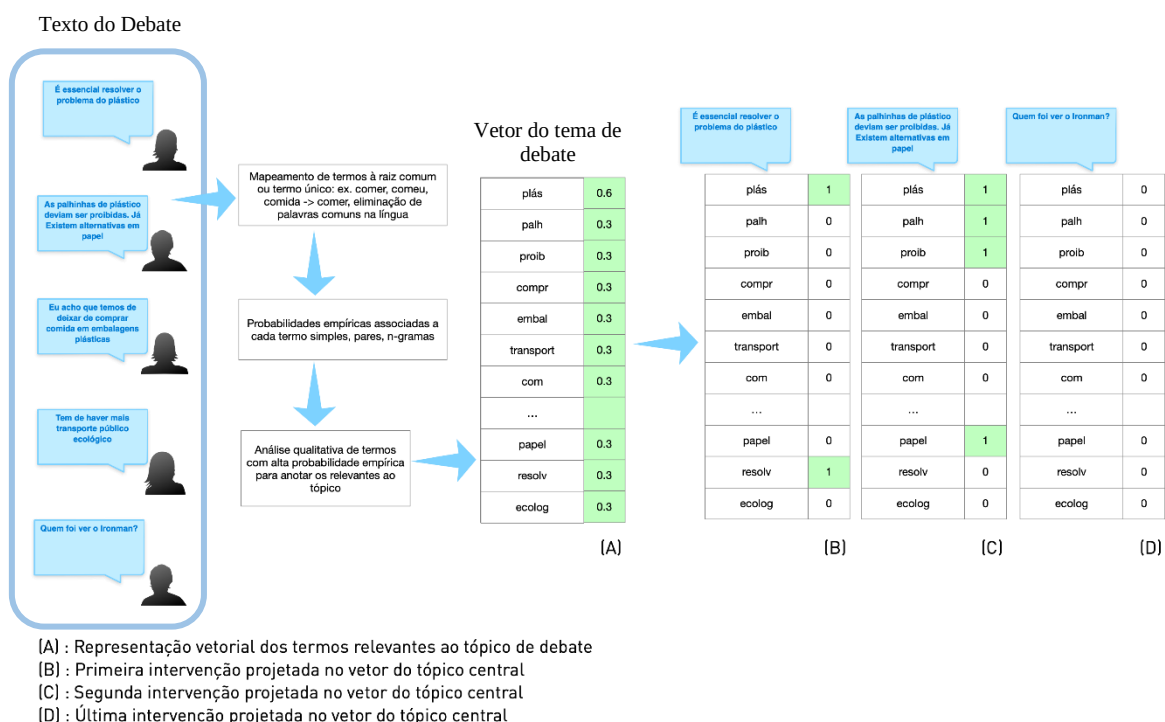


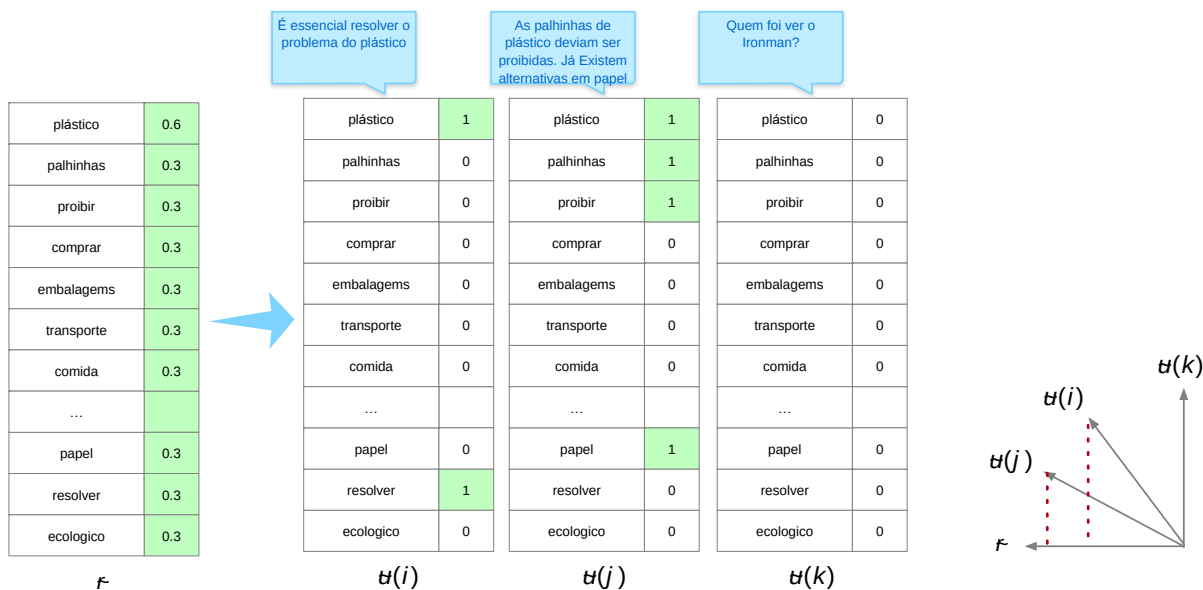
Figura 44: Ilustração do método utilizado para criar a representação vetorial de um texto fonte x

Fonte: Elaboração em coautoria com orientador desta dissertação

Resumo: O método começa por particionar **x** numa lista de palavras (referidas como «tokens»). Da lista resultante, são normalmente eliminados os «tokens» comuns na língua do texto (pronomes, artigos, conjunções, etc.). Os «tokens» restantes são substituídos pela sua raiz morfológica. No passo seguinte, são calculadas as estatísticas relacionadas com a frequência com que cada «token» aparece no texto, produzindo assim o resultado final (vetor). O vetor **r** foi obtido a partir de um conjunto de textos sobre crise climática. Os vetores **u(i)**, **u(j)** e **u(k)** correspondem a intervenções específicas em debates sobre o mesmo tópico. As componentes dos vetores capturam uma estatística relativa à frequência com a qual diferentes palavras aparecem no texto e no «corpus».

4.4 – Alinhamento

Uma das maiores vantagens de representar textos através de vetores é a facilidade com que se pode calcular a distância angular entre pares de vetores. Quando o ângulo entre dois vetores que representam textos – intervenções em debates, no nosso caso – é pequeno, interpretamos estes textos como sendo similares, ou como estando alinhados entre si. Dentro da abordagem metodológica «*bag of words* – BoW» (Zhang, 2010), usada neste projeto, foi utilizado o cosseno do ângulo entre dois vetores, $\mathbf{x}$, $\mathbf{y}$, para definir o seu «alinhamento» (Huang, 2008). Quando os vetores são idênticos, os mesmos formam um ângulo $\beta = 0^\circ$, e, portanto, $\cos(\beta) = 1$ (máximo alinhamento). Por outro lado, quando não existe qualquer similaridade entre os vetores o ângulo será de $\beta = 90^\circ$, e o $\cos(\beta) = 0$ (vetores ortogonais, sem alinhamento). A Figura 45 contém uma ilustração gráfica do conceito de «alinhamento». Este é o fundamento matemático das análises apresentadas na presente dissertação. Através desta medida, é possível determinar se uma intervenção dentro dum debate está alinhada com o tema central do mesmo, assim como a magnitude dessa relação, representada com um valor entre zero e um. É importante salientar que foi definido «alinhamento», no contexto desta análise, como uma medida que relaciona qualquer uma das intervenções num debate com o correspondente tema central. Para calcular o alinhamento numa intervenção com o tema de debate é necessário, portanto, ter construído o vetor de referência para o «*corpus*» do debate previamente, e que todas as intervenções sejam representadas vectorialmente com as mesmas componentes deste vetor, tal como ilustrado na Figura 45.



4.5 – Coerência

A coerência é uma medida que representa, numericamente, o grau em que uma intervenção $u(t)$ está alinhada com as intervenções imediatamente anteriores, até uma intervenção $u(t - w)$. O número de intervenções anteriores a serem extraídas é determinado por um parâmetro (w). Este parâmetro é estimado empiricamente e, portanto, dependente do contexto do estudo. A coerência só pode ser calculada para textos que fazem parte de sequências temporais, como no nosso caso, intervenções em debates e diálogos, e não para outro tipo de «corpora» onde relações temporais carecem de qualquer importância, como, a título de exemplo, um «corpus» de receitas de cozinha. Considere-se, por exemplo, um grupo de pessoas a conversar sobre privacidade no namoro. Após um número de intervenções, sempre alinhadas com esse tema, um dos intervenientes surpreende o grupo, mudando o assunto para algum avanço em tecnologias de inteligência artificial (IA). A medida de coerência para esta última intervenção, no contexto das intervenções passadas que eram todas sobre privacidade, terá, portanto, um valor baixo. É de notar, que se este grupo optar por continuar a conversa na linha de IA nas intervenções sucessivas, a coerência dessas intervenções será mais elevada.

Dada uma intervenção $u(t)$, (para a qual se pretende calcular a coerência), há que extrair primeiro um subconjunto das intervenções imediatamente anteriores, $[u(t-1), u(t-2), \dots, u(t-w)]$. Neste projeto foi utilizado o «interval» $w = [5,6,7]$, e determinado que o valor ótimo é $w = 5$. Uma vez extraída esta sequência de intervenções anteriores, deve calcular-se o alinhamento entre $u(t)$ e cada uma das intervenções anteriores extraídas. O valor final da coerência da intervenção $u(t)$ é o máximo alinhamento encontrado entre $u(t)$ e as w intervenções anteriores.

4.5 – Coesão

A medida de coesão que utilizámos neste projeto para interpretar a «saúde» de um debate combina as medidas de alinhamento das diferentes intervenções com o tema central e sua coerência. Para poder calcular a coesão, há que calcular primeiro o alinhamento e coerência para todas as intervenções no «corpus». Depois será necessário identificar a dimensão do «corpus» sobre o qual se pretende calcular a coesão. Por exemplo, é possível calcular a coesão do debate inteiro, incluindo todas as intervenções. É possível também calcular a coesão das intervenções provenientes de alunos e alunas duma mesma escola ou, ainda, separar por intervenções por escola de origem e dividi-las por género. Uma vez escolhida a dimensão de análise, extraem-se as intervenções correspondentes, que são guardadas num conjunto denotado por U . No próximo passo deve ser acertado o valor de dois parâmetros, a e c , ambos entre zero e um. Os valores destes parâmetros são escolhidos pelo investigador, geralmente com base em observações empíricas. O primeiro parâmetro, a , corresponde ao limiar mínimo do que será considerada uma intervenção alinhada com o tópico do debate. Quando existirem muitos subtemas e palavras chave no tópico, o valor de a deve ser baixo. Pelo contrário, se o tema for muito específico, com poucas palavras relevantes, o valor de a deverá ser maior. O outro parâmetro, c , determina o limiar mínimo do que se considera uma intervenção coerente no contexto do debate. Sempre que existam muitos intervenientes e subtemas relevantes dentro do debate, será útil fixar o valor deste parâmetro num valor baixo. Uma vez identificada a dimensão do «corpus» a ser analisado e, após extraídas as intervenções relevantes num conjunto de intervenções U , o próximo passo é extrair de U todas as intervenções com alinhamento maior ou igual a a , e coerência maior ou igual a c , as quais guardamos num conjunto S . A coesão pode agora ser calculada como a proporção de S em U , nomeadamente $|S|/|U|$.

4.6 – Resultados

Tabela 1: Número de participantes por atividade, agrupados por escola

Participantes	E1	E2	E4	E7	E8	E9
Privacidade no namoro	10	-	-	12	-	13
Crise climática	4	12	11	7	11	9
Violência de género	4	-	19	-	-	8

Fonte: Elaboração em coautoria com orientador desta dissertação

Tabela 2: Número de intervenções realizadas em cada atividade, agrupadas por escola

Intervenções	E1	E2	E4	E7	E8	E9
Privacidade no namoro	118	-	-	146		390
Crise climática	70	725	280	107	456	215
Violência de género	131	-	452	-	-	245

Fonte: Elaboração em coautoria com orientador desta dissertação

As atividades de recolha de dados, através de sessões de debate *online* realizadas na plataforma *Debaqi*, ocorreram em cinco momentos diferentes. Seis escolas, designadas pelos códigos E1, E2, E4, E7, E8 e E9 (tabelas 1 e 2), participaram nestas sessões de debate. O primeiro debate foi sobre a privacidade no namoro, no qual participaram as escolas E1, E7 e E9, juntas, numa única sessão. Para o segundo tema, a crise climática reservou-se três sessões de debate. A primeira contou com a participação, novamente, das escolas E1, E7 e E9; a segunda unicamente com a escola E2; e a última com as escolas E4 e E8. Finalmente, para o debate sobre violência de género foram realizadas duas sessões. As escolas E1 e E9 participaram na primeira e a escola E4 na segunda e última sessão.

No início de cada debate, os alunos e alunas entraram na plataforma para efetuar o seu registo individual no sistema. Este registo foi feito sempre no início de cada sessão de debate, não tendo sido guardadas contas de estudante para assegurar o anonimato. Durante esta fase de registo, os estudantes receberam um nome de utilizador aleatório e responderam a um questionário de caracterização sociodemográfica. Para este estudo foram utilizadas, exclusivamente, as variáveis género e escola de origem de cada estudante, obtidos neste questionário inicial. Uma vez registados, os intervenientes acederam à área publica do fórum,

e receberam uma notificação alertando que todas as suas intervenções poderiam ser lidas pelos restantes participantes a partir desse momento. A seguir, foi apresentado um vídeo introdutório do tema e, no final do mesmo, apresentada pelo *chatbot* a pergunta para debate. As perguntas específicas para cada tema de debate foram as seguintes:

1. **Privacidade no namoro** – A troca de palavras-passe nas redes sociais é prova de confiança no namoro? Sim ou não?
2. **Crise climática** – Esperamos que sejam os políticos a resolvê-la ou começamos, desde já, a modificar o nosso estilo de vida? Sim ou não? Como? De que forma?
3. **Violência de género** – As imagens que acabaste de visualizar enquadram-se na discriminação das mulheres? Sim ou não? Porquê?

Todas as sessões foram moderadas pelo autor desta dissertação e seu orientador. O objetivo da moderação foi garantir a cordialidade e respeito entre os participantes, introduzir intervenções para recuperar a linha central do debate em caso de divergência para outros assuntos, controlar o tempo da atividade e resolver, rapidamente, quaisquer falhas técnicas que pudessem surgir. Cada sessão de debate teve uma duração média de cinquenta (50) minutos. Todos os dados recolhidos foram guardados, via canal encriptado, numa base de dados relacional na qual todos os participantes estão anonimizados desde o início, mas onde as escolas de origem de cada interveniente estão identificadas.

Neste estudo apresentamos os resultados obtidos na primeira atividade de debate, centrada no tema da privacidade no namoro, para a Escola E1. Não é possível a inclusão dos restantes resultados preliminares do estudo, visto que estes não foram ainda publicados pela RBE, entidade detentora da propriedade dos dados recolhidos e analisados. O orientador desta dissertação, quem é simultaneamente o líder da equipa científica do projeto *Debaqi*, conseguiu a devida autorização para a inclusão dos resultados parciais mostrados neste volume. A participação da escola E1 contou com dez (10) estudantes, dois (2) rapazes e oito (8) raparigas. Os rapazes produziram trinta e duas (32) intervenções enquanto as raparigas oitenta e seis (86). Identificámos uma diferença significativa na tendência central do alinhamento por género, 0.1 para os rapazes (M), 0.24 para as raparigas (F), (Mann–Whitney - $U = 1071$, $n_M = 32$, $n_F = 86$; $p = 0.023$, $\text{sig} = 0.05$, bicaudal), o que demonstra uma clara tendência das raparigas (F) para manter um maior alinhamento com o tema do debate, em comparação com os rapazes (M). A escola E1 não participou de forma constante durante esta sessão de debate. A participação

esteve mais concentrada em dois momentos. O segmento de maior envolvimento aconteceu entre $u(100)$ e $u(200)$ e observámos, posteriormente, um segundo bloco de intervenções a partir de $u(400)$. A coesão destes dois grupos (rapazes e raparigas da escola E1) coincidiu em grande medida com a tendência central dos seus alinhamentos. A Figura 46 permite comparar os resultados da escola E1 com as outras escolas que participaram neste tema de debate. A Figura 47, por outro lado, permite analisar o resultado dos inquéritos pós-debate, o qual demonstra que nesta primeira atividade de debate os estudantes foram neutrais em relação à utilidade da atividade de debate, possivelmente pela ocorrência de um elevado grau de «sabotagem» do debate por parte de alguns alunos.

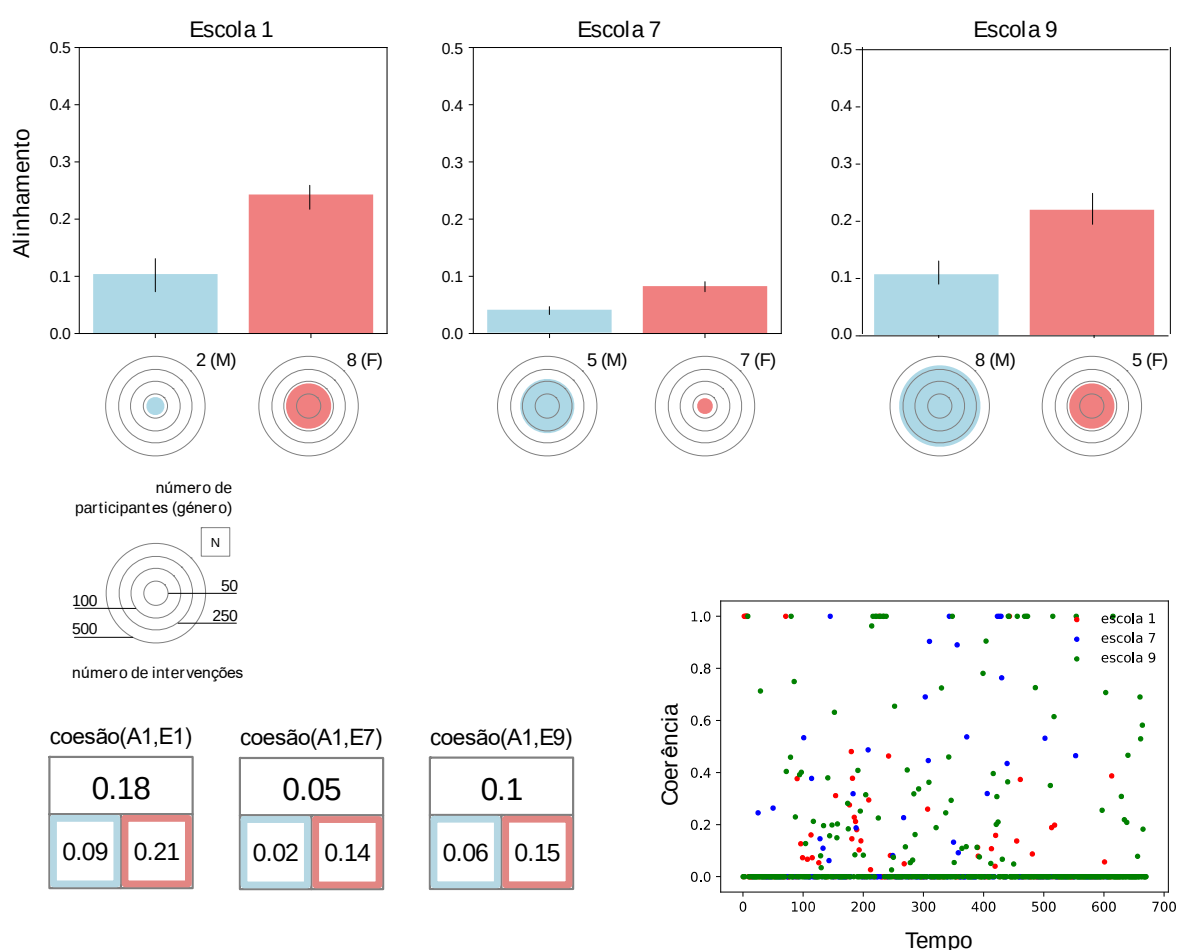


Figura 46: Resultados da análise das intervenções durante a primeira sessão de debate sobre o tema privacidade no namoro
Fonte: Elaboração em coautoria com orientador desta dissertação

Resumo: Este tema foi debatido numa única sessão, na qual participaram três escolas, E1, E7 e E9. Para simplificar a leitura dos gráficos, substituímos a notação matemática com termos explícitos para o indicador de coesão. A cor azul é utilizada para representar intervenções dos rapazes, e o laranja para intervenções das raparigas. Na parte superior da figura representamos o alinhamento, com gráficos da tendência central por escola e género, incluindo o correspondente erro padrão. Estes gráficos incluem uma escala circular adicional para representar o número de intervenções, assim como o número de participantes por género, em cada escola. Incluímos também uma legenda para entender a escala circular. O gráfico que permite observar a coerência a longo do debate está localizado no canto inferior direito e, à sua esquerda, os valores obtidos para o indicador de coesão.



Figura 47: Resultados do questionário por inquérito no final da sessão de debate A1 escola E1

Fonte: Elaboração em coautoria com orientador desta dissertação

Resumo: Os estudantes foram neutrais em relação à utilidade da atividade de debate, possivelmente por ter ocorrido um elevado grau de «sabotagem» do debate por parte de alguns alunos, ou por outros motivos por nós desconhecidos, mas que poderão ser passíveis de interpretações sociológicas por parte de especialistas na matéria. Um dos aspetos mais importantes a ser notado é o facto de os estudantes terem sido expostos a novos pontos de vista, embora não tenham alterado a sua opinião primária, o que podemos considerar como um indicador hipotético de baixa «saúde» do diálogo.

CONCLUSÃO

O principal objetivo da presente dissertação consiste no desenvolvimento de uma aplicação *Web* que permita a todos os estudantes pertencentes às diversas escolas nacionais, participar de forma segura em atividades de debate *online* cuja finalidade passa por ampliar as suas capacidades de pensamento crítico, assim como a participação numa sociedade plural. Neste âmbito, desenvolvemos a plataforma *Debaqi* de forma a integrar, não só as mais atuais e preformantes tecnologias, segurança e proteção de dados, como também um módulo de inteligência artificial que permitisse efetuar o processamento de linguagem natural em tempo real. Em versões subsequentes desta aplicação, este módulo irá possibilitar a moderação de debates de forma totalmente automatizada.

Adicionalmente, utilizámos técnicas de «*data science*» para analisar os diversos dados recolhidos durante as três (3) atividades realizadas no decorrer do projeto piloto. Esta análise permitiu conferir a esta plataforma resultados muito interessantes, os quais vão ser desde já utilizados pelo Ministério da Educação para a delineação de intervenções educativas que permitam ir ao encontro dos desafios relacionados com os potenciais perigos da «era digital», com um especial foco na disseminação de notícias falsas, discursos de ódio e polarização.

No decorrer desta investigação fomos confrontados com algumas limitações logísticas, assim como com a necessidade de integrar num curto espaço de tempo um elevado número de requisitos e restrições estabelecidas pela RBE. Nas primeiras sessões de trabalho ficou muito claro que não poderiam haver quaisquer falhas que pudessem de alguma forma prejudicar e pôr em causa em primeiro lugar os alunos, a equipa científica e a RBE, bem como as escolas participantes. Outra limitação foi o facto de existirem prazos extremamente apertados para finalizar as várias etapas desta aplicação. Houve um especial cuidado relativo a qualquer «*input*» a ser introduzido em novas versões do *software*, ainda que a necessidade de aprovação prévia por parte de todos os «*stakeholders*» implicasse, naturalmente, algumas pausas e reflexões durante o seu desenvolvimento. Esta espera coincidiu em alguns momentos com o dia previsto para realizar a atividade, o que acarretou um esforço adicional para que o novo desenvolvimento estivesse disponível.

Tendo em conta os prazos estabelecidos, o volume das especificações e os objetivos alcançados, consideramos os resultados obtidos nesta investigação absolutamente satisfatórios e acreditamos que a plataforma *Debaqi* poderá contribuir para que sejam realizados novos estudos com um elevado potencial de impacte social no sistema educativo nacional. Enquanto programador sénior será relevante acrescentar que, ao longo deste percurso, foi bastante

benéfico o conhecimento adquirido sobre tecnologias com as quais o contacto até então não tinha sido regular, como Python e *WebSockets*. Estas serão certamente úteis no futuro enquanto profissional na indústria do *IT*.

Ademais, o sucesso deste projeto pode ser comprovado através do financiamento concedido por parte da Fundação para a Ciência e Tecnologia¹⁹, cujo investigador responsável é o orientador desta mesma dissertação de Mestrado, o Prof. Doutor Manuel Pita.

Trabalhos futuros

Sugerem-se como trabalhos futuros a análise estatística dos restantes inquéritos pós-debate, bem como completar a validação qualitativa dos cálculos de coerência e a análise computacional do indicador de coesão para todos os debates.

A Rede de Bibliotecas Escolares deverá disponibilizar os resultados obtidos neste estudo às escolas participantes, para que estes sejam revistos pelos Professores Bibliotecários e assim efetivar o fecho do projeto piloto.

A obtenção de uma bolsa de investigação financiada pela Fundação para Ciência e Tecnologia habilitará a continuação deste estudo nos próximos anos pelos alunos que se encontram a fazer Doutoramento no COPELABS. O meu papel no futuro desta investigação passará por ser consultor independente, auxiliando como «mentor» da aplicação, prestando apoio em qualquer dúvida que possa surgir sobre o funcionamento da mesma, bem como aconselhar sobre a melhor forma de realizar certas integrações.

O planeamento e mobilização para início da segunda fase deverá acontecer no próximo ano letivo 2020/2021.

¹⁹ Este projeto é financiado através de fundos nacionais pela Fundação para a Ciência e Tecnologia (FCT, I.P) – Projeto de Investigação Científica e Desenvolvimento Tecnológico em Ciência dos dados e inteligência artificial na Administração Pública – através do projeto *Fatores para a promoção do diálogo e comportamentos saudáveis em comunidades escolares online*. Ref. DSAIPA/DS/0102/2019 [Consult. 08-12-2019]. Disponível em WWW:<URL: https://www.fct.pt/apoios/projectos/consulta/vglobal_projecto?idProjecto=155483&idElemConcurso=14115>

BIBLIOGRAFIA

David Ballantyne (2004). Dialogue and its role in the development of relationship specific knowledge. *Journal of Business & Industrial Marketing* 19, 114-123.

Huang, A. (2008). Similarity measures for text document clustering. Holland, J., Nicholas, A. and Brignoli, D. (eds), *Proc New Zealand Computer Science Research Student Conference* (NZCSRSC 2008), Christchurch, New Zealand, 49-56.

Jason Lengstorf. (2012). Realtime Web Apps: HTML5 WebSocket, Pusher, and the Webs Next Big Thing. New York: Apress, eBook.

Lucas Krause (2014). Microservices: Patterns and Applications: Designing fine-grained services by applying patterns. [s.l.] : Lucas Krause, eBook.

Polly W. Wiessner (2014). Embers of society: Firelight talk among the Ju/'hoansi Bushmen. *Proceedings National Academy of Science* (PNAS), Michigan, USA : National Academy of Science. [Consult. 15-07-2019]. Disponível em WWW:<URL:<https://www.pnas.org/content/early/2014/09/16/1404212111.abstract>>

Ramos, Juan. (2003). Using TF-IDF to determine word relevance in document queries. New Jersey, USA : Department of Computer Science, Rutgers University 23515 BPO Way, Piscataway, NJ, 08855.

Zhang, Y, Jin, R. & Zhou, Z (2010). Understanding bag-of-words model: a statistical framework. *Int. J. Mach. Learn. & Cyber.* 1, 43–52

Documentos Eletrónicos

Alex Barashkov – Microservices vs. Monolith Architecture. [Consult. 19-08-2019]. Disponível em WWW:<URL:https://dev.to/alex_barashkov/microservices-vs-monolith-architecture-4l1m>

Altexsoft – Web Application Architecture: How the Web Works. [Consult. 20-08-2019]. Disponível em WWW:<URL:<https://www.altexsoft.com/blog/engineering/web-application-architecture-how-the-web-works/>>

Cambridge Dictionary – Cambridge University Press. Disponível em WWW:<URL:<https://dictionary.cambridge.org/pt>>

Cezar Taurion – Como saber se uma tecnologia disruptiva é exagero, tendência ou tsunami? [Consult. 02-09-2019]. Disponível em
WWW:<URL:<https://www.linkedin.com/pulse/como-saber-se-uma-tecnologia-disruptiva-%C3%A9-exagero-ou-tsunami-taurion/>>

Ciberdúvidas da Língua Portuguesa – A origem da palavra diálogo. [Em linha]. Lisboa : ISCTE-Instituto Universitário de Lisboa, 2007 [Consult. 08-07-2019]. Disponível em
WWW:<URL:<https://ciberduvidas.iscte-iul.pt/consultorio/perguntas/a-origem-da-palavra-dialogo/22161>>

Dick Weisinger – Software Development: Are Microservices Always the Right Choice? [Consult. 20-08-2019]. Disponível em
WWW:<URL:<https://formtek.com/blog/software-development-are-microservices-always-the-right-choice/>>

Docker – What is a Container? [Consult. 21-10-2019]. Disponível em
WWW:<URL:<https://www.docker.com/resources/what-container>>

Harshal Patil – Contemporary Front-end Architectures. [Consult. 25-08-2019]. Disponível em WWW:<URL:<https://blog.webf.zone/contemporary-front-end-architectures-fb5b500b0231>>

Jake Lumetta – Why the monolith isn't dead. [Consult. 01-12-2019]. Disponível em
WWW:<URL:<https://blogs.mulesoft.com/dev/tech-ramblings/why-the-monolith-isnt-dead/>>

Liam Kirkaldy – ‘We cannot accept this as the new normal’ – European lawmakers urge action on disinformation. [Consult. 04-11-2019]. Disponível em
WWW:<URL:<https://www.publictechnology.net/articles/news/%E2%80%98we-cannot-accept-new-normal%E2%80%99-european-lawmakers-urge-action-disinformation>>

Mike Roberts – Serverless Architectures. [Consult. 13-08-2019]. Disponível em
WWW:<URL:<https://martinfowler.com/articles/serverless.html>>

WebSockets. [Consult. 13-09-2019]. Disponível em
WWW:<URL:<https://javascript.info/websocket>>

GLOSSÁRIO

Algoritmo – Método para resolver um problema através de uma série de etapas precisas, definidas e finitas. É um conjunto de instruções específicas e detalhadas pelas quais uma ação específica é alcançada.

Amazon Web Services (AWS) – Conjunto de serviços de computação em *Cloud* pertencentes a Amazon.com que possibilitam o desenvolvimento de plataformas altamente escaláveis. A AWS disponibiliza dezenas de serviços *Cloud* para os diversos modelos IaaS, PaaS e SaaS. É o fornecedor de serviços mais popular, com quase 30% de participação no mercado global em IaaS desde 2015.

Angular – *Framework* para desenvolvimento de aplicações recorrendo a tecnologias como HTML, CSS e, principalmente, TypeScript, concebida pela Google.

API – Interface que agrega um conjunto de padrões, rotinas e funções de programação definidos por intermédio de um *software* que possibilita o uso das funcionalidades das aplicações para uso de serviços, sem necessidade do seu envolvimento em processos demasiado complexos de implementação.

Aplicação – Programas de computador desenvolvidos para permitir que um utilizador execute um conjunto de funções ou tarefas.

Aplicação Cloud – Aplicação que é executada na *Cloud*.

Avatar – Representa um utilizador através de uma personagem virtual. Através deste o utilizador pode navegar no ambiente virtual e interagir com as diversas funcionalidades.

Balanceador de carga – *Hardware* ou *software* que distribui a carga de computação em vários recursos.

Base de dados – Coleção organizada de dados.

Cache – Armazenamento de dados ou processos frequentemente utilizados na memória do navegador para um acesso futuro mais rápido, poupando assim tempo e uso desnecessário de pedidos HTTP.

Cardinalidade – Princípio fundamental sobre relações em base de dados relacionais, onde é descrito o número de entidades com as quais outra entidade está associada.

Certificado Digital – Assinatura com validade jurídica que garante a proteção das transações eletrónicas e outros serviços na *Internet*. Esta permite que pessoas e empresas se identifiquem e assinem digitalmente em qualquer lugar do mundo, com mais segurança e flexibilidade.

Chat – Meio de comunicação maioritariamente textual praticado em tempo real. Através deste, duas ou mais pessoas podem partilhar mensagens *online* utilizando as salas disponíveis para o efeito.

Chave criptográfica – Algoritmo que permite descriptar através de um par de chaves pública e privada um ficheiro encriptado.

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 – Em informática, é um processo ou programa que solicita serviços a um servidor.

Cloud – Expressão utilizada para «batizar» serviços de armazenamento na *Internet*, como o Dropbox, Google Drive e OneDrive. A *Cloud* pode ser pública, privada ou híbrida.

Commit – No contexto de sistemas de controlo de versão refere-se a submeter as últimas alterações do código fonte a um determinado repositório, de modo a estas se tornem parte integrante da versão principal («*Head*») do mesmo.

Computação em Cloud – Utilização da memória, capacidade de armazenamento e processamento de servidores interligados através da *Internet* – princípio de computação distribuída.

Cookie – Pequenos ficheiros que ficam armazenados no computador. Guardam dados específicos de um cliente ou *website* para serem acedidos futuramente pelo servidor *Web* mais rapidamente.

CPU – Responsável pelo controlo e execução das tarefas do computador, ou seja, é onde todas as operações do computador são processadas.

Diagrama de classes – Representação da estrutura e relações entre classes que servem de modelo para objetos como entidades.

Docker – Plataforma cujo principal objetivo passa por facilitar a criação e gestão de ambientes isolados recorrendo à tecnologia de *containers*.

Downtime – Tempo em que o sistema não está operacional.

Elasticidade – Termo usado para referenciar a capacidade que um sistema tem para se adaptar às mudanças na carga de processamento.

Entidade – Representa uma tabela de uma base de dados relacional, onde cada instância da mesma corresponde a uma linha da tabela. O artefacto de programação primário de uma entidade é uma classe (POO).

Escalabilidade – Capacidade que um equipamento possui para receber novos componentes/características evitando que se torne obsoleto ou deixe de atender às necessidades do utilizador. Esta capacidade pode incluir, por exemplo, o aumento de quantia de memória, troca de discos ou processador, etc.

Hard-coded – Prática de desenvolvimento de *software* que visa inserir dados diretamente no código fonte de um programa, invés de os obter através de fontes externas, como por exemplo, uma base de dados.

Hardware – Designação genérica de todo tipo de equipamento informático, ou seja, é a parte física do computador, como por exemplo o processador, disco rígido, memória RAM, etc.

Heroku – Plataforma «Platform as a Service» (PaaS) que permite aos programadores criar, executar e operar aplicações desenvolvidas nas mais diversas linguagens de programação num ambiente totalmente *Cloud*.

Hub – Denominação dada ao equipamento ou lugar onde convergem dados que chegam de uma ou várias fontes e perseguem para um ou vários destinos, dependendo do tipo e instrução recebida.

Infrastructure as a Service – Modelo de computação em *Cloud* no qual o fornecedor aloja recursos de computação virtualizados, bem como recursos de rede e armazenamento, e fornece-os ao utilizador em forma de serviço.

Java – Linguagem de programação orientada a objetos. Diferente das linguagens de programação convencionais, que são compiladas para código nativo. Esta é compilada para um *bytecode* e executada em ambiente de máquina virtual (JVM).

Kernel – É o «núcleo» de um sistema operativo, ou seja, a sua unidade básica, responsável por fazer a «ponte» entre os programas e os componentes físicos do computador.

Linux – Sistema operativo «*open-source*», desenvolvido em Unix, usado na maioria dos serviços em *Cloud*.

Login – Identificação de um utilizador para acesso a um determinado computador, sistema ou recurso.

Logs – É uma expressão utilizada para descrever o processo de registo de eventos relevantes numa aplicação.

Máquina Virtual – *Software* que executa programas como um computador «real», também conhecido como processo de virtualização.

Microsoft Azure – Anteriormente conhecido como Windows Azure, é a plataforma de computação em *Cloud* da Microsoft. O Azure foi originalmente lançado como uma solução PaaS, mas agora providência serviços PaaS e IaaS.

Middleware – *Software* que liga componentes de *software* ou aplicações.

Navegador – *Software* que permite visualizar imagens, fotos, sons, vídeos e textos na *Internet*.

Offline – Termo cujo significado literal é «fora da linha», referindo-se a algo que está desligado. É habitualmente usado para designar que um determinado utilizador ou computador não está ligado à rede.

Online – Define que determinado utilizador ou computador está ligado à rede.

On-Prem – *Software* ou infraestrutura executada em computadores nas instalações da pessoa ou organização que a utiliza.

Open-source – Termo criado pela *Open Source Initiative* a partir do texto original da *Debian Free Software Guidelines* e determina que programas de código aberto devem ter em conta certos elementos como a distribuição livre, o código fonte, trabalhos derivados, a não discriminação a pessoas/grupos e áreas de atuação, a distribuição da licença e licença que não restrinja outros programas.

ORM – *Framework* que permite mapear um modelo orientado a objetos para um modelo relacional de base de dados. A ideia por detrás desta ferramenta passa por dar aos programadores a possibilidade de codificar «*queries*» na sua linguagem de programação preferencial, utilizando um paradigma orientado a objetos.

Plataforma – Expressão utilizada para indicar a tecnologia utilizada em determinada infraestrutura, garantindo assim a facilidade de integração dos diversos componentes na mesma.

Platform as a Service – Modelo de computação em *Cloud* no qual um fornecedor faculta as ferramentas de *hardware* e *software* necessárias para criar, desenvolver e gerir aplicações na *Internet*.

Porta – Abstração utilizada pelo protocolo TCP/IP para distinguir entre ligações simultâneas um único destino.

Proof of Concept – Termo utilizado para denominar um modelo prático que possa provar o conceito (teórico) estabelecido por uma investigação ou artigo técnico.

Protocolo – Descrição formal de formatos e normas que dois computadores devem obedecer ao trocar mensagens. Conjunto de regras padrão que especificam o formato, a sincronização, a sequência e a verificação de erros em comunicação de dados.

Proxy – É o termo utilizado para definir os intermediários entre o cliente e o servidor.

Query – Processo de extração de dados de uma base de dados e sua apresentação.

Router – Permite a navegação entre páginas *Web* conforme a interação dos utilizadores na aplicação.

Sala Virtual – Espaço virtual onde os utilizadores podem interagir com os demais que igualmente se encontrem na mesma sala.

Saúde do diálogo – Sequências de intervenções que estão alinhadas consistentemente entre si (são consistentes), aumentando desta forma a coesão do diálogo. A coesão é o nosso principal indicador da saúde de um diálogo.

Servidor – É basicamente, um computador mais potente do que um *desktop* convencional. Desenvolvido especificamente para transportar dados e executar aplicações (*software*), disponibilizando-as a outros computadores que estejam ligados na mesma rede.

Software – Código passível de ser desenvolvido em várias linguagens para executar funções específicas.

SQL – Linguagem de pesquisa padrão para uma base de dados relacional.

Stack – Dois ou mais *switches* com o objetivo de gerir o tráfego de rede através de um único endereço IP.

URL – Endereço de rede no qual se encontra algum recurso informático, como por exemplo um ficheiro ou uma *website*.

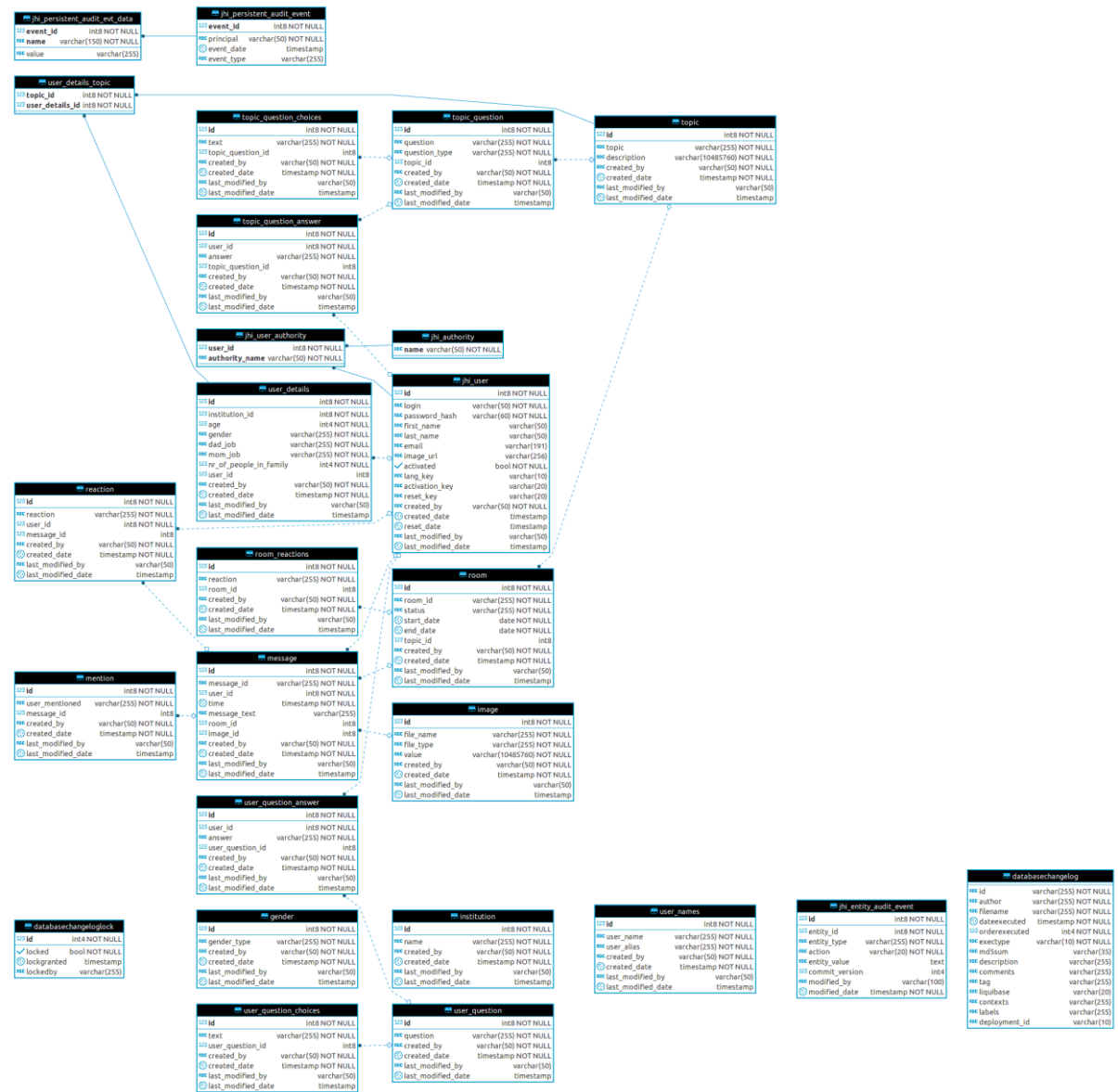
UX – Preocupação das equipas de desenvolvimento em criar aplicações pensadas na visão e utilização dos seus utilizadores.

Vírus – Denominação dada a pequenos programas desenvolvidos para produzir danos a diversos níveis. Este podem afetar a integridade dos dados de um computador em particular ou de toda a rede de uma organização.

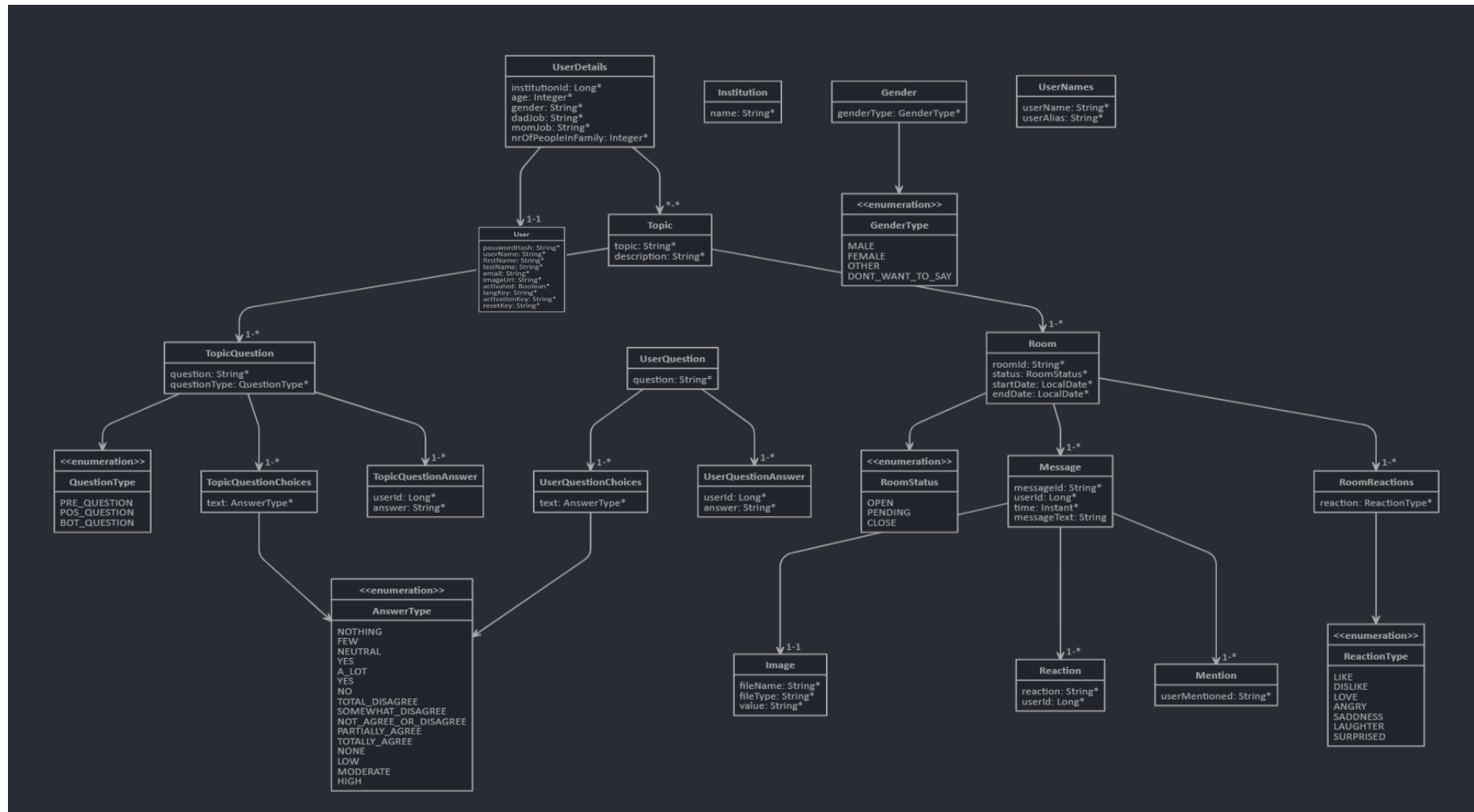
Web – Sigla que designa a rede mundial de computadores ligados em rede.

APÊNDICES

Apêndice I – Estrutura relacional de dados desenvolvida para a aplicação *Debaqi*



Apêndice II – Diagrama de «classes» que demonstram as entidades desenvolvidas para a aplicação *Debaqi*



Apêndice III – Diagrama de Entidade-Relação desenvolvido para a aplicação *Debaqi*

