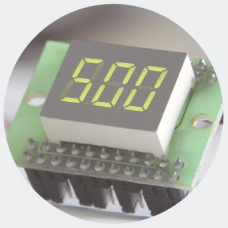
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Desenvolvimento de Sistema de Apoio à Operação das Máquinas de Pré Programação da Bosch

JOÃO RITO BRANDÃO
Outubro de 2017

POLITÉCNICO
DO PORTO

DESENVOLVIMENTO DE SISTEMA DE APOIO À OPERAÇÃO DAS MÁQUINAS DE PRÉ PROGRAMAÇÃO DA BOSCH

João Rito Brandão

Departamento de Engenharia Eletrotécnica

Instituto Superior de Engenharia do Porto

2017

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Eletrotécnica e de Computadores especialização em Telecomunicações

Candidato: João Rito Brandão, Nº 1120484, 1120484@isep.ipp.pt
Orientação científica: Jorge Botelho da Costa Mamede, jbm@isep.ipp.pt
Supervisão: Vítor Manuel Lopes Serrano, vitor.serrano@pt.bosch.com

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
25 de outubro de 2017

Agradecimentos

A todas as pessoas que me ajudaram e contribuíram para o desenvolvimento deste projeto, expresso aqui os meus agradecimentos.

Gostaria de começar por agradecer a toda a minha Família por todo o apoio incansável. Aos meus pais e irmão, os principais pilares da pessoa que sou hoje, estou eternamente grato pela confiança e força constante para atingir esta nossa meta. Um especial obrigado aos meus avós e madrinha por estarem sempre disponíveis para me apoiar nesta jornada na minha cidade natal. À Sara, pelo amor, paciência e constante apoio para alcançar os meus objetivos. Ao "Pessoal que quer ser Mestre" por todos os momentos de brincadeira e trabalho partilhados ao longo destes dois anos de Mestrado.

Agradeço à entidade patronal, a Bosch *Car Multimedia*, pela oportunidade e pelo apoio para a conclusão deste Mestrado. Uma empresa com valores bem definidos e com uma organização de excelência permitindo uma boa integração. Em particular à equipa TS1 e aos elementos da minha ilha de trabalho, por todos os bons momentos. Em especial, ao Daniel pelo apoio e acompanhamento diário durante todo o projeto. Ao Vítor e ao Ricardo pela sua capacidade de liderança exemplar e por acreditarem em mim para alcançar o objetivo deste projeto.

Por fim, gostaria de agradecer ao meu orientador, o engenheiro Jorge Mamede não só pela sua orientação e toda a dedicação ao longo do desenvolvimento deste projeto, mas também por todas as dicas, ensinamentos e conselhos que ao longo dos últimos cinco anos tive o prazer de presenciar e ouvir nas diversas unidades curriculares que lecionou.

Resumo

Este projeto tem como objetivo o desenvolvimento de um sistema para apoiar todas as atividades realizadas nas máquinas de pré programação da Bosch *Car Multimedia* em Braga. O projeto surgiu de uma necessidade de melhorar e facilitar o trabalho das diversas equipas que interagem com estas máquinas.

O desenvolvimento de qualquer sistema impõe que seja criada uma rede de comunicação entre os diversos componentes que dele fazem parte, permitindo facilitar a interoperabilidade entre todos. O desenvolvimento deste sistema começa com um programa que assume a responsabilidade de recolher os dados necessários e culmina numa aplicação *Web* que serve como interface gráfica para os utilizadores.

O programa responsável por recolher os dados necessários faz também o envio dos mesmos para o servidor da secção de *Manufacturing Engineering* 3 (MFE3), recolhendo um conjunto de informação de processos e configurações de programas das máquinas de pré programação. Do lado do servidor, desenvolveu-se um conjunto de serviços *Web* e de uma base de dados de forma a receber e guardar a informação, respetivamente. A aplicação *Web* foi desenvolvida com PHP recorrendo a uma *framework* de forma a garantir uma maior robustez, flexibilidade e estabilidade da aplicação. O desenvolvimento desta aplicação no formato *Web* fez com que fosse possível aceder a partir de qualquer lugar, sem haver uma prévia instalação de software no dispositivo a utilizar.

Assim, este projeto apresenta todo o sistema desenvolvido como uma solução para apoiar as atividades que as diferentes equipas desenvolvem através da sua interação com as máquinas de pré programação de circuitos integrados.

Palavras-Chave

Bosch, *Car Multimedia*, Pré Programação, circuitos integrados, indústria, produção, serviços *Web*, base de dados, Laravel.

Abstract

This project aims to develop a system to support all the activities carried out on Bosch Car Multimedia pre-programming machines in Braga. The project arose from a need to improve and facilitate the work of the various teams that interact with these machines.

The development of any system requires the creation of a communication network between the various components that make part of it, making it possible to facilitate interoperability among all. The development of this system starts with a program that assumes the responsibility of collecting the necessary data and culminates in a Web application serving as graphical user interface.

The program responsible for collecting the necessary data also sends them to the Manufacturing Engineering 3 (MFE3) section server, collecting a set of process information and program configurations from the pre-programming machines. On the server side, a set of Web services and a database were developed in order to receive and save the information, respectively. The Web application was developed with PHP using a framework to ensure greater robustness, flexibility and stability of the application. The development of this application in Web format made it possible to access from anywhere, without having a previous software installation in the device to be used.

Thus, this project presents the whole system developed as a solution to support all the activities that the different teams develop through their interaction with the pre-programming machines of integrated circuits.

Keywords

Bosch, Car Multimedia, pre-programming, integrated circuits, industry, production, Web services, server, data bases, Web applications, PHP, Laravel.

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

API	-	*Application Programming Interface*
BD	-	Base de Dados
CRUD	-	*Create Read Update Delete*
CSS	-	*Cascading Style Sheets*
CSUQ	-	*Computer System Usability Questionnaire*
DNS	-	*Domain Name System*
FK	-	*Foreign Key*
FTP	-	*File Transfer Protocol*
HTML	-	*HyperText Markup Language*
HTTP	-	*HyperText Transfer Protocol*
I/O	-	*Input/Output*
IC	-	*Integrated Circuits*
IoT	-	*Internet of Things*
IP	-	*Internet Protocol*
JS	-	*JavaScript*
JSON	-	*JavaScript Object Notation*
MAC	-	*Media Access Controll*
MFE3	-	*Manufacturing Engineering 3*

MVC - *Model View Controller*

ORB - *Object Request Broker*

ORM - *Object-Relational Mapping*

OSI - *Open System Interconnection*

OTP - *One Time Programmable*

PCB - *Printed Circuit Board*

PHP - *Hypertext Preprocessor*

PK - *Primary Key*

REST - *Representational State Transfer*

SMTP - *Simple Mail Transfer Protocol*

SO - Sistema Operativo

SQL - *Structured Query Language*

TCP - *Transmission Control Protocol*

TP - Taxa de Produção

TR - Taxa de Rejeição

UCC - Unidades de Controlo de Conectividade

UDP - *User Datagram Protocol*

URI - *Uniform Resource Identifier*

URL - *Uniform Resource Locator*

WWW - *World Wide Web*

XML - *eXtensible Markup Language*

1. INTRODUÇÃO

1.1. MOTIVAÇÃO

Nos dias que correm, a evolução tecnológica desempenha cada vez mais um papel decisivo no sucesso das organizações. As empresas vivem um ambiente competitivo muito exigente, encontrando na tecnologia uma forma de as ajudar a evoluir de modo a garantir a diferença através da oferta de serviços mais eficazes e eficientes e de produtos com um padrão de qualidade mais elevado, aos seus clientes.

A utilização de sistemas de computadores em rede apresenta um papel cada vez mais comum no ambiente empresarial para a realização das suas atividades diárias. A passagem do papel para a máquina é uma evolução que ainda hoje muitas empresas têm a necessidade de fazer, sempre com o objetivo de melhoria contínua. A utilização destes sistemas permite às empresas um acesso permanente a um conjunto de dados permitindo-lhes trabalhar de forma sistemática na resolução de problemas e melhoria de processos tanto a nível interno como externo, quando é proporcionada uma melhoria nas relações com os seus clientes.

Este projeto foi proposto pela empresa, a Bosch, com o intuito de desenvolver uma aplicação *Web* para apoiar as atividades que são desenvolvidas nas máquinas de pré programação de circuitos integrados. A aplicação estará disponível a diversas equipas internas da Bosch, desde que tenham necessidade de interagir com as máquinas de pré programação.

1.2. BOSCH

A Bosch foi fundada em Estugarda no ano de 1886 por Robert Bosch como apenas uma oficina ligada à mecânica e à eletricidade. Hoje em dia a empresa está presente em aproximadamente 60 países seja por subsidiárias, empresas regionais ou pela Robert Bosch Gmb que em conjunto originam o Grupo Bosch. A empresa é líder mundial no fornecimento de tecnologia e serviços e encontra-se dividida em quatro áreas de negócio: Tecnologia Industrial, Soluções de Mobilidade, Bens de Consumo e Tecnologia de Energia e Edifícios [1].

A Bosch *Car Multimedia* é a divisão responsável pelo desenvolvimento de soluções integradas de entretenimento, navegação, telemática e ajuda à condução. Esta divisão foca-se nas necessidades do condutor e procura desenvolver as tecnologias que permitem melhorar a segurança, o conforto da condução e a redução do consumo de energia. A evolução diária do desenvolvimento tecnológico tem permitido que o setor automóvel se torne cada vez mais dependente destas novas tecnologias. Esta divisão tem como uma das suas principais competências a produção de Unidades de Controlo de Conectividade (UCC) para veículos comerciais, carros de passageiros e veículos de duas rodas. A função das UCC é conectar o veículo à *Internet*, permitindo utilizar funcionalidades de apoio em caso de acidente ou avaria, por exemplo [1].

Em Portugal, a Bosch está presente desde 1911 como um fornecedor líder no fornecimento de tecnologia e serviços, contando atualmente com cerca de 3800 colaboradores que fazem da empresa um dos maiores empregadores industriais a nível nacional. Esta é representada pela Bosch *Car Multimedia* em Braga, pela Bosch Termo Tecnologia em Aveiro e pela Bosch *Security Systems* em Ovar. Dividida pelas diversas áreas a Bosch desenvolve e fabrica nas representações mencionadas multimédia automóvel, soluções de água quente e sistemas de comunicação e segurança, respetivamente. Em conjunto, as várias divisões da empresa representadas em Portugal somaram um total de 1.3 biliões de Euros em 2016 [1].

Em Braga, a divisão da empresa é a *Car Multimedia* sendo a principal fábrica desta divisão e a maior empresa do Grupo Bosch em Portugal, sendo responsável por 68 % do total das vendas do Grupo. Esta divisão começou por desenvolver autorrádios, no entanto, com a evolução da tecnologia e com a necessidade de melhorar a segurança e o conforto da condução, começou a desenvolver soluções integradas para entretenimento, navegação e

funções de auxílio à condução. Assim, a Bosch *Car Multimedia* desenvolve diversas tecnologias de ponta adaptadas aos requisitos e necessidades da mobilidade moderna com o principal objetivo de reduzir a distração do condutor através da implementação de novas ideias nas quais a interface é centrada no utilizador [2].

1.2.1. PRÉ PROGRAMAÇÃO DE CIRCUITOS INTEGRADOS

Na Bosch, a primeira fase da montagem de um produto é a Pré Programação. Nesta fase é feita a programação dos Circuitos Integrados (*Integrated Circuits* – IC) para que, na fase seguinte, possam ser inseridos nas *Printed Circuit Boards* (PCB) dos diferentes produtos.

One Time Programmable (OTP) era a designação dada ao conjunto das máquinas que fazem a pré programação dos IC. Estas máquinas de diferentes fornecedores e, por isso, com diferentes características, são responsáveis por fazer a programação dos IC antes de estes serem levados para as linhas de inserção automática, onde são inseridos nas PCB. Isto permite tornar o processo mais eficiente, uma vez que não há a necessidade de programar os IC quando já se encontram inseridos nas PCB, havendo uma passagem ao nível seguinte da linha de montagem mais rapidamente.

As OTP são memórias que apenas podem ser programadas uma vez e, por isso, não são as únicas memórias utilizadas pela Bosch neste processo de pré programação. No entanto, era o nome dado ao conjunto das máquinas que são responsáveis por fazer a pré programação dos IC. Com o desenvolvimento do projeto apresentado neste documento, percebeu-se que o termo teria de ser abandonado uma vez que não é corretamente aplicado à atual denominada Pré Programação.

1.3. OBJETIVOS

O objetivo principal deste projeto consiste no desenvolvimento de uma solução de interface *Web* para apoiar as atividades das máquinas de pré programação. Até ter sido desenvolvido este projeto, a Bosch não apresentava quaisquer bases para que fosse apenas desenvolvida a aplicação em si. Desta forma e dado que a criação de uma aplicação *Web* exige uma prévia criação de ferramentas para o seu correto funcionamento, dividiu-se este objetivo em múltiplas tarefas, como as seguintes:

- Estudar o sistema existente;
- Fazer o levantamento de requisitos das máquinas de pré programação;
- Fazer o levantamento de requisitos de *software*;
- Analisar quais as ferramentas e *frameworks* que mais se adequam ao sistema;
- Definir especificações para o sistema;
- Desenvolver um programa para recolher informação das máquinas;
- Desenvolver uma base de dados utilizando *Structured Query Language* (SQL);
- Desenvolver um conjunto de serviços *Web* de forma guardar, ler e alterar os dados da base de dados;
- Desenvolver uma aplicação *Web*.

1.4. ORGANIZAÇÃO DO DOCUMENTO

Este relatório encontra-se dividido em oito capítulos. No primeiro capítulo é feita uma introdução ao tema do projeto e uma abordagem à empresa onde o mesmo foi elaborado. No segundo capítulo é feito o estudo de sistemas e aplicações distribuídas como introdução ao funcionamento do sistema existente e quais os benefícios do novo. No terceiro capítulo são apresentadas as especificações para todos os elementos do sistema, de forma a garantir que tudo o que é necessário desenvolver é desenvolvido da forma correta levando ao seu funcionamento. Neste capítulo é ainda feito um levantamento das tecnologias utilizadas para o desenvolvimento deste projeto assim como um estudo sobre *frameworks*. Os capítulos quatro, cinco e seis são aqueles que estão diretamente relacionados com o desenvolvimento deste projeto. Assim, o quarto capítulo diz respeito a todo o desenvolvimento que foi efetuado a nível da aplicação responsável por fazer a recolha dos dados das máquinas de pré programação. No quinto capítulo é apresentada a arquitetura da base de dados assim como alguns dos campos mais relevantes das tabelas criadas. O sexto capítulo apresenta os serviços que recebem os dados provenientes das máquinas e que proporcionam uma forma de acesso à informação da base de dados. Ainda neste capítulo é apresentada a aplicação

desenvolvida com o objetivo disponibilizar toda a informação aos seus utilizadores de forma a apoiá-los nas suas atividades diárias quando interagem com as máquinas de pré programação. No sétimo capítulo são apresentados alguns testes feitos ao sistema como um todo e a alguns dos seus elementos individualmente, de forma a avaliar e a comprovar o seu funcionamento. No oitavo e último capítulo são apresentadas as conclusões obtidas a partir da realização deste projeto.

Por questões de confidencialidade, ao longo deste relatório serão utilizados nomes fictícios para ficheiros, tabelas e seus conteúdos, programas e serviços.

2. APLICAÇÕES DISTRIBUÍDAS BASEADAS EM *WEB*

Neste capítulo são estudadas redes de computadores e aplicações distribuídas baseadas em *World Wide Web (WWW)*, salientando a comunicação entre os seus diversos elementos. Começa-se por fazer uma contextualização, abordando um pouco a história dos sistemas computacionais e introduzindo os conceitos de sistemas e aplicações distribuídas. Em seguida, são apresentadas as principais arquiteturas de sistemas e aplicações distribuídas baseadas em *Web*, seguidas da comunicação utilizada e os seus principais modelos. Por fim é introduzido o estudo de caso da Bosch, integrando os conceitos teóricos apresentados no sistema que foi desenvolvido. Neste capítulo é apresentado um tema mais generalizado, como uma apresentação teórica geral, e que vai afunilando até se chegar ao problema que levou ao desenvolvimento deste projeto.

2.1. CONTEXTUALIZAÇÃO

A meados dos anos 80 do século XX, com o aparecimento dos primeiros microprocessadores de 8 bit, fez-se sentir uma evolução exponencial nos sistemas computacionais. Desde então, começaram a surgir os microprocessadores de 16, 32 e 64 bit, tornando os sistemas computacionais mais poderosos, mais comuns e mais baratos. Com a disponibilização dos computadores em empresas sentiu-se a necessidade de criar uma forma de comunicação entre eles, dando origem às primeiras redes de computadores. A constante

evolução da tecnologia levou à expansão das redes locais a redes cada vez mais abrangentes e distanciadas. A invenção das redes de computadores de alta velocidade permitiu criar um sistema de comunicação entre centenas de máquinas, servindo como ponto de partida para as redes de computadores que hoje em dia existem [3].

Hoje em dia os computadores apresentam-se como estruturas com elevada complexidade interligados através da principal rede de dados que existe, a *Internet*. A *Internet* é um sistema distribuído constituído por múltiplas plataformas interligadas e que comunicam de acordo com um *standard* especificado, o protocolo *Internet Protocol* (IP) [4]. A utilização de um protocolo comum permite que, independentemente do sistema computacional, se possa realizar a comunicação entre os diversos sistemas [3].

De acordo com *Andrew S. Tanenbaum* e *Maarten Van Steen*, um sistema distribuído é uma coleção de computadores independentes que surge aos seus utilizadores como um único sistema coerente [3]. O conceito de aplicação distribuída tem como base o conceito de sistema distribuído. No entanto, uma aplicação distribuída, incide na existência de um *software* desenvolvido sobre uma rede de computadores [3] [4].

O principal objetivo das aplicações distribuídas é proporcionar aos diversos tipos de utilizadores uma forma de aceder a recursos remotos e controlá-los de uma forma eficiente [4]. Os recursos vão desde ficheiros individuais até mesmo outras máquinas. Isto permite tornar a colaboração entre os diversos utilizadores mais fácil e ao mesmo tempo permite que diversos utilizadores possam aceder a recursos mais dispendiosos sem haver uma necessidade de duplicação dos mesmos.

2.2. COMUNICAÇÃO EM REDES DE COMPUTADORES

A comunicação é feita através de redes que são responsáveis por transportar a informação sobre o formato de mensagens entre os componentes. Se se pensar na *Internet* como ela é hoje em dia, é possível perceber que, por si só, é um sistema distribuído, composto por milhares de elementos criando uma rede que permite a troca de informação entre todos. Em seguida são apresentados dois dos principais conceitos que fazem das redes IP o que são hoje em dia.

2.2.1. MODELO DE REFERÊNCIA OSI

O modelo *Open System Interconnection* (OSI) é uma referência fundamental para tudo o que se relaciona com comunicações por computadores. Este modelo define uma pilha de sete camadas que permitem efetuar a comunicação entre computadores, desde o nível mais baixo de transmissão de sinais através das ligações físicas até ao nível mais elevado onde se encontram as aplicações que utilizam a rede. A Tabela 1 apresenta um esquema da apresentação deste modelo de referência.

Tabela 1 Estrutura do Modelo OSI.

Nº	Camadas
7	Aplicação
6	Apresentação
5	Sessão
4	Transporte
3	Rede
2	Ligação de Dados
1	Física

A camada da aplicação é responsável por servir de interface para a apresentação da informação ao utilizador. É nesta camada que são efetuados os pedidos para obtenção de novos dados. Isto leva à passagem para a camada de apresentação. Esta camada preocupa-se com a sintaxe da informação transmitida e, por isso, é responsável por formatar os dados de forma a apresentar a informação [4] [5].

A camada de sessão permite estabelecer uma sessão entre o cliente e o servidor. Esta camada apresenta mecanismos de gestão de sessão, permitindo melhorar a sincronização entre clientes e servidores. Estabelecida a sessão entre os dois extremos da comunicação, é necessário transportar a informação em ambos os sentidos, sendo esta responsabilidade atribuída à camada de transporte. O principal objetivo desta camada é garantir que a informação é entregue na outra ponta da conexão. Esta camada apresenta mecanismos de

controlo de fluxo, capaz de criar distintas ligações para cada conexão estabelecida de forma a melhorar o fluxo da informação entre os diferentes pontos. Apesar de o transporte da informação ser da responsabilidade da camada de transporte, a correta identificação do cliente e do servidor que garante a comunicação entre os dois, é da responsabilidade da camada de rede. Esta camada é responsável pelo controlo das operações na sub-rede das extremidades das conexões [4] [5].

A camada de ligação de dados é a camada que serve de ponte entre as interfaces físicas e a rede. Esta camada é responsável por atribuir os endereços *Internet Protocol* (IP) a endereços *Media Access Control* (MAC) da placa de rede das máquinas. A camada de ligação física diz respeito ao conjunto de ligações compostas por cabos de cobre, fibra ótica, placas de rede, etc [4] [5].

Este modelo, como o próprio nome diz, serviu de referência para a definição de conceitos que posteriormente vieram a ser postos em prática nas atuais redes de computadores [5]. A seguir é apresentada a arquitetura *Transmission Control Protocol* / *Internet Protocol* (TCP/IP) como sendo uma arquitetura aberta e como sendo a base de funcionamento das redes de computadores atuais.

2.2.2. ARQUITETURA TCP/IP

A arquitetura TCP/IP é considerada a avó das redes de computadores atuais. A necessidade de se desenvolver um *standard* para tornar possível a comunicação entre os diferentes computadores, levou ao desenvolvimento desta arquitetura, que também já é conhecida como Modelo de Referência TCP/IP [4]. À semelhança do modelo OSI, também se encontra organizada em camadas. Na Tabela 2, pode ver-se a estrutura desta arquitetura e a sua comparação com a estrutura do modelo OSI.

Tabela 2 Arquitetura TCP/IP e comparação com o Modelo OSI [4].

Nº	**OSI**	**TCP/IP**
7	Aplicação	Aplicação
6	Apresentação	Não implementadas nesta arquitetura.
5	Sessão	

4	Transporte	Transporte
3	Rede	Internet
2	Ligação de Dados	Interface de Rede
1	Física	

Como é possível reparar a partir da tabela anterior, esta arquitetura apenas apresenta quatro camadas não existindo as camadas de apresentação e de sessão uma vez que se percebe que não havia uma necessidade de as incluir no modelo. A camada de aplicação passa assim a conter todos os protocolos de mais alto nível e de aplicação como o *HyperText Transfer Protocol* (HTTP) para a utilização das páginas na *Web*, *Domain Name System* (DNS), *File Transfer Protocol* (FTP), *Simple Mail Transfer Protocol* (SMTP), entre outros. A camada de transporte apresenta-se desenvolvida da mesma forma que é apresentada para o modelo OSI. No entanto, neste caso foram definidos dois protocolos responsáveis por fazer o transporte dos dados: o *Tansmission Control Protocol* (TCP) e o *User Datagram Protocol* (UDP). O primeiro orientado à conexão permitindo a troca de dados entre duas máquinas através de uma conexão estabelecida e com a entrega dos pacotes sem erros. O segundo protocolo é orientado ao pacote e, por isso, não implementa uma conexão prévia para garantir a entrega dos dados. A camada da *Internet* apresenta a solução que permite o transporte de dados na rede de forma independente e através de diferentes redes. Para esta camada foi definido um protocolo que define um *standard* para o formato dos pacotes que são nela transportados: o protocolo *Internet Protocol* (IP). Por fim, a camada de ligação de rede, apresenta-se como uma camada que representa a conexão entre as máquinas e a rede e que são capazes de enviar os pacotes IP para a rede [4].

2.3. ARQUITETURAS DAS APLICAÇÕES DISTRIBUÍDAS BASEADAS EM *WEB*

Apresentado o modelo de referência e a atual arquitetura utilizada para a comunicação entre computadores, passa-se ao estudo das diversas arquiteturas das aplicações distribuídas.

2.3.1. ARQUITETURAS CLIENTE-SERVIDOR

As arquiteturas cliente-servidor abordam três principais modelos distintos: modelo de dois níveis, modelo de três níveis e modelo de níveis múltiplos. Estes três modelos representam muitos dos sistemas e aplicações que são desenvolvidas com base em arquiteturas cliente-servidor [6] [7].

O modelo de dois níveis, apresentado na Figura 1, é composto no primeiro nível pelas aplicações clientes. Estas aplicações são responsáveis por fazer pedidos ao segundo nível, recorrendo ao HTTP. No segundo nível, o servidor, os pedidos são recebidos por uma aplicação servidora que será responsável por comunicar com a base de dados, se for necessário, e retornar informação requisitada pelas aplicações clientes. A presença da base de dados no mesmo nível que a aplicação servidora, apresenta algumas vantagens no que diz respeito à velocidade de processamento dos dados da base de dados. Por outro lado, a divisão do trabalho não é tão assente, podendo limitar a aplicação distribuída no que diz respeito a escalabilidade [6] [7].

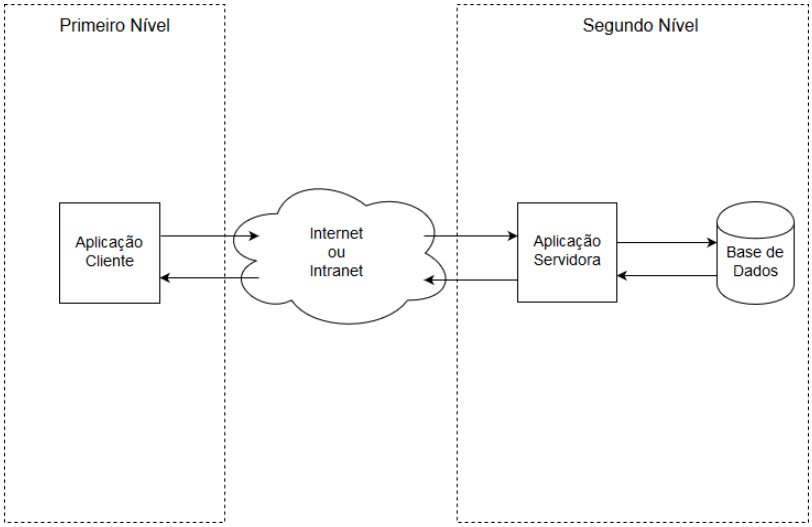


Figura 1 Modelo da arquitetura Cliente-Servidor a dois níveis.

O modelo de três níveis, apresentado na Figura 2, faz a separação da base de dados para um servidor para ela dedicado. O primeiro nível continua a ser composta pelas aplicações clientes. O segundo nível passa a ser composto pelo servidor que suporta apenas a aplicação servidora, ou seja, o nível que implementa as funcionalidades de atendimento de pedidos dos clientes passa a precisar de comunicar com um nível diferente antes de retornar as respostas aos clientes [6] [7].

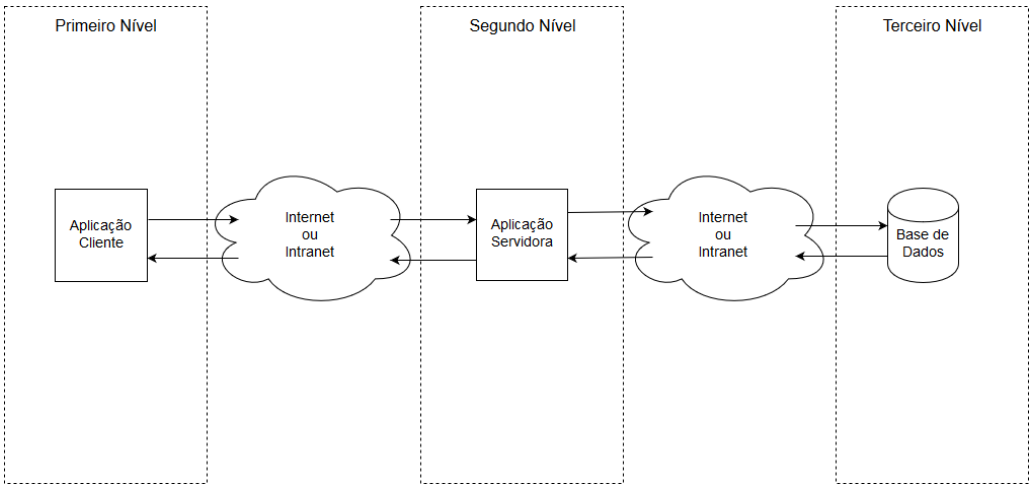


Figura 2 Modelo da arquitetura Cliente-Servidor a três níveis.

O modelo de níveis múltiplos, apresentado na Figura 3, é maioritariamente utilizado em aplicações distribuídas cujas arquiteturas se baseiam em conjuntos de objetos que funcionam como clientes ou servidores [7]. Neste modelo há um objeto designado por *Object Request Broker* (ORB) responsável por receber as mensagens do cliente e invocar o respetivo serviço. Assim, invocam-se os serviços que se encontram num servidor para ele dedicados, podendo ainda interagir com um outro nível, também ela dedicada para as bases de dados. Este modelo é muito utilizado no desenvolvimento de *Application Programming Interfaces* (API), proporcionando soluções bastante complexas. A principal desvantagem deste modelo é a sua baixa eficiência proporcionada pela necessidade de interpretar os objetos [6].

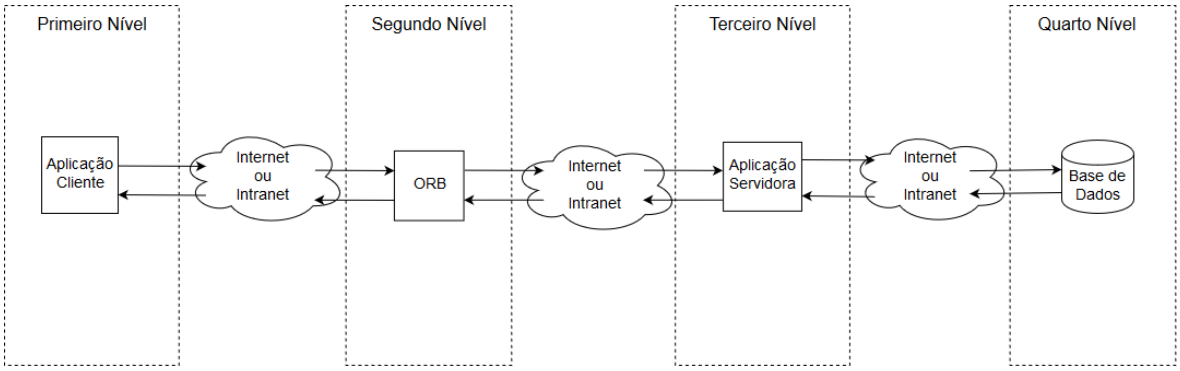


Figura 3 Modelo da arquitetura Client-Servidor a múltiplos níveis.

2.3.2. ARQUITETURAS PEER-TO-PEER

Nestas arquiteturas, as aplicações distribuídas não apresentam funcionalidades específicas para cada elemento. Neste caso, todos os constituintes destas arquiteturas funcionam como cliente e como servidor em simultâneo, como é possível ver na Figura 4. São arquiteturas muito utilizadas hoje em dia em jogos em rede no mundo dos *smartphones* ou em sistemas de *Internet of Things* (IoT), permitindo uma comunicação mais rápida e direta entre os diversos componentes do sistema [6].

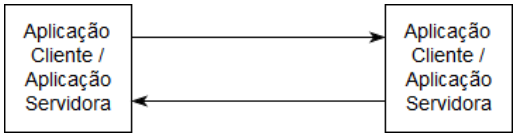


Figura 4 Modelo da arquitetura Peer-to-Peer.

É possível perceber que, se os elementos da aplicação servem como cliente e servidor, muitas vezes pertencem a arquiteturas maiores, onde são responsáveis por desempenhar funções mais complexas [6].

2.4. ESTUDO DE CASO – BOSCH

Em seguida apresenta-se a consolidação de todo o capítulo até aqui desenvolvido, culminando na integração dos conceitos teóricos abordados no caso que levou ao desenvolvimento deste projeto.

As máquinas de pré programação de IC da Bosch são máquinas responsáveis por fazer a programação daquilo que será o coração dos produtos finais apresentados aos clientes. A necessidade de análise da produção destas máquinas, levou à proposta que desencadeou o desenvolvimento deste projeto. A seguir são apresentadas as limitações do sistema utilizado antes do desenvolvimento deste projeto na empresa, assim como os benefícios que o projeto traz.

2.4.1. LIMITAÇÕES DO SISTEMA ANTIGO

A Bosch precisa de aceder a uma série de indicadores presentes nas máquinas de pré programação de uma forma fácil e intuitiva. Aceder a este conjunto de indicadores é do total interesse da empresa a nível de melhoria contínua de processos internos e melhoria de

processos de qualidade. De seguida apresentam-se as limitações que o sistema anterior ao desenvolvimento efetuado apresentava:

- A recolha dos dados presentes nas máquinas de pré programação era efetuada manualmente por colaboradores habilitados para tal, uma vez que há uma necessidade de perceber o funcionamento das mesmas e do seu próprio sistema;

- Os dados recolhidos eram armazenados, ou não, de uma forma apenas temporária em documentos, levando a que os colaboradores tivessem uma necessidade de ir manualmente recolhendo informação sempre que necessitassem da mesma;

- A obtenção dos indicadores necessários era calculada manualmente pelos colaboradores e apenas para o conjunto de dados que teriam recolhido para tal;

- A descentralização da informação leva a uma constante necessidade deslocações às máquinas, uma sistemática recolha e manipulação da informação de forma manual e, por isso, uma dispensa de tempo de trabalho muito grande.

2.4.2. BENEFÍCIOS DO NOVO SISTEMA

Dados os problemas citados na subsecção anterior, a Bosch idealizou uma plataforma que deverá ser capaz de contornar estas limitações, trazendo novos benefícios aos seus utilizadores, facilitando o trabalho, reduzindo o tempo dispensado para a realização de certas tarefas e ainda aumentando a capacidade de melhoria dos processos.

O sistema desenvolvido e apresentado neste documento representa uma total mudança na forma como a Bosch interage com as máquinas para a recolha de informação, representando uma passagem de um processo manual a automático. Isto é possível através do desenvolvimento tecnológico e a integração de aplicações distribuídas que permitem a atualização constante da informação. De seguida apresentam-se os avanços que este novo sistema traz para a Bosch:

- Pretende-se que seja feita uma recolha de informação das máquinas de pré programação de forma automática e recorrente de forma a obterem-se os dados sempre atualizados e dispensando tempo aos colaboradores de fazer tal tarefa;

- Pretende-se manter um registo da informação de forma centralizada para que possa ser acedida por qualquer utilizador autorizado e em qualquer parte, sem haver a necessidade de deslocação até às máquinas;

- Com a informação guardada, pretende-se determinar uma série de indicadores que permitam aos colaboradores trabalhar em métodos de melhoria contínua no que diz respeito aos processos de trabalho das máquinas de pré programação e ainda na melhoria da relação com o cliente;

- Pretende-se ter uma plataforma permanentemente disponível e que possibilite diferentes acessos por parte de diferentes equipas da Bosch.

2.5. SÍNTESE

A evolução dos computadores e das redes de computadores tem vindo a facilitar a disponibilização de dados aos utilizadores. O desenvolvimento de sistemas distribuídos cada vez mais complexos levou à criação daquela que hoje é a maior rede de dados que existe, a *Internet*.

À medida que os sistemas evoluíam, houve a necessidade de criar modelos que hoje em dia servem como base para toda a comunicação entre sistemas computacionais. O desenvolvimento do Modelo OSI, apesar de não ter sido implementado na prática, permitiu que se desenvolvesse um segundo modelo capaz de definir um *standard* na comunicação em *Web*, entre os diversos sistemas existentes no mercado: o modelo TCP/IP. Este modelo, dividido por quatro camadas define as diferentes interfaces necessárias para tornar a comunicação entre os sistemas possível, desde a camada de Aplicação onde se dá a interação com o utilizador e o pedido de dados através do protocolo HTTP até à camada de Interface de Rede onde é feita a conexão entre as máquinas e a rede.

O desenvolvimento de *standards* de comunicação entre os diversos sistemas permitiu o desenvolvimento de aplicações capazes de se apresentarem divididas em diversos componentes, podendo estes estar em máquinas diferentes. As aplicações distribuídas apresentam-se como uma grande fatia das soluções de *software* presentes no mercado devido à sua robustez, escalabilidade e versatilidade. A existência de diferentes modelos

arquitetónicos, permitem o desenvolvimento das aplicações para as mais díspares situações desde sistemas com informação armazenada de forma centralizada até à utilização dos vários componentes a funcionar como cliente e servidor em simultâneo.

Na Bosch pretende-se implementar um sistema composto por uma aplicação de recolha de dados e uma aplicação distribuída com uma arquitetura cliente/servidor, com o número de níveis a definir de acordo com as especificações e capaz de armazenar os dados, disponibilizar uma interface de comunicação com outras aplicações e disponibilizar uma interface gráfica para interação com os seus utilizadores.

3. ESPECIFICAÇÕES

Este capítulo apresenta as especificações da solução. Para isso é necessário fazer uma divisão em duas partes: objetivos e requisitos, para cada componente desenvolvida.

Antes de introduzir as especificações para este sistema, é necessário compreender o processo de pré programação dos IC, ilustrado na Figura 5.

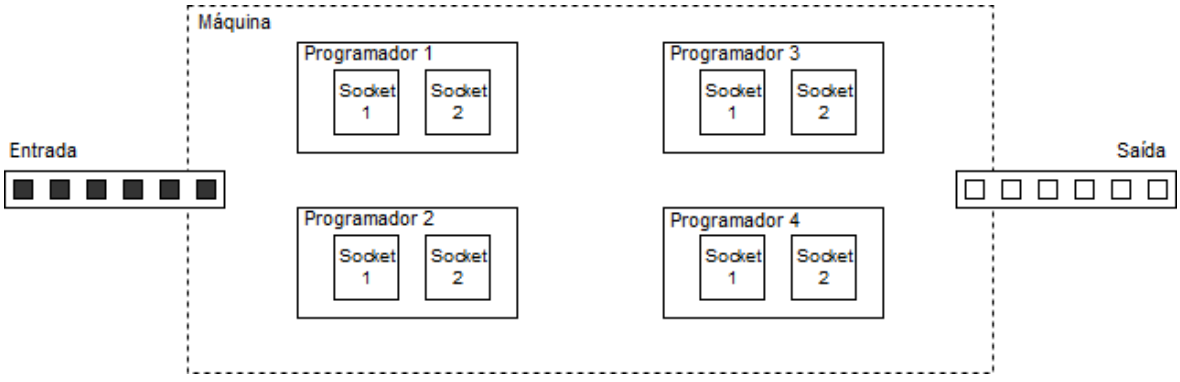


Figura 5 Componentes das Máquinas de Pré Programação.

O operador, como pessoa responsável por colocar as máquinas de pré programação a funcionar, insere uma fita que contém ICs não programados (vazios) na entrada da máquina e uma fita sem ICs na saída da máquina, de forma a recolher os ICs programados. De seguida e de acordo com o tipo de IC inserido na entrada da máquina, escolhe o programa que deve

ser programado nos IC, dando início ao processo automático da máquina. Quando a máquina dá início ao seu processo automático de programação dos IC, um braço robótico recolhe os IC na entrada da máquina e coloca-os em *sockets* presentes nos diversos programadores da máquina. No final da programação de um IC, o braço robótico recolhe o IC para a fita de saída da máquina. Quando a quantidade de IC que o operador necessita estiverem na fita de saída da máquina, o operador dá o processo de programação por terminado. Antes de enviar a fita para o passo seguinte da produção, o operador tem de identificar a fita com uma etiqueta, que contém informação do programa programado nos IC.

Como é possível de perceber, os componentes denominados de programadores são os elementos das máquinas responsáveis por fazer a programação dos IC. No entanto, cada programador, necessita de uma estrutura onde seja possível encaixar os ICs, permitindo o acesso aos diversos contactos nele presentes para que possa ser feita a programação. Estas estruturas são os *sockets*.

3.1. ARQUITETURA DO SISTEMA

Pretende-se desenvolver um sistema capaz de recolher dados, armazená-los e disponibilizá-los a diversos utilizadores de forma a apoiar o seu trabalho. Desta forma, especificou-se que o sistema deve ser desenvolvido sobre o formato de uma plataforma *Web*, com base num modelo de funcionamento do tipo Cliente-Servidor. Assim, é possível manter a informação organizada e centralizada e o acesso à mesma, sendo feito através de uma aplicação *Web*, pode ser efetuado a partir de qualquer lado sem haver uma necessidade de instalação prévia de *software*.

O desenvolvimento deste sistema apoia-se num dos vários modelos existentes para descrever a arquitetura de aplicações distribuídas *Web*. A arquitetura utilizada baseia-se no modelo cliente/servidor sendo composta por duas camadas, como apresentado na Figura 6.

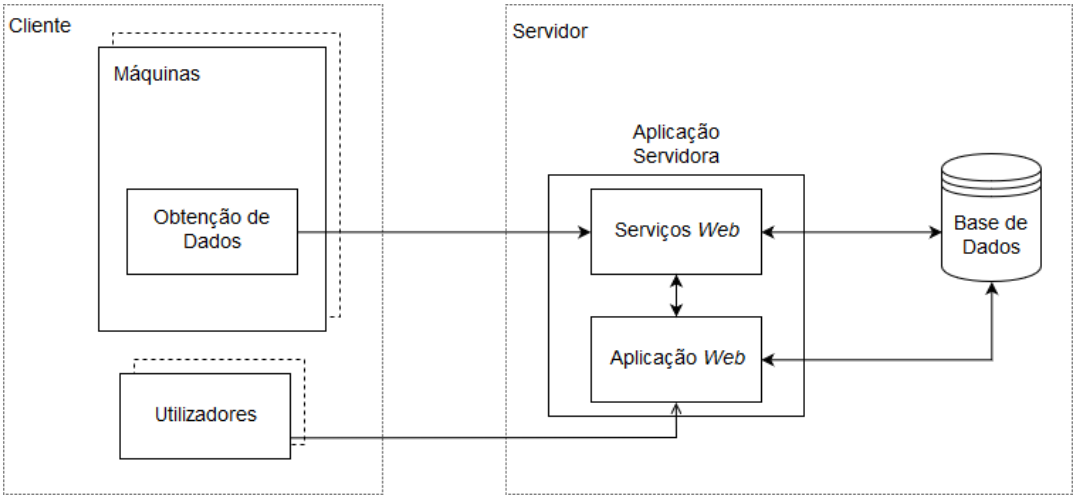


Figura 6 Esquema Geral do Sistema.

A primeira camada é composta pela aplicação cliente, que neste caso poderá ser um navegador *Web* ou outra aplicação. É nesta camada que são também executados todos os *scripts* que permitem tornar as diversas páginas *Web* dinâmicas. A segunda camada diz respeito à aplicação servidora, aquela que implementa as funcionalidades de atendimento dos pedidos dos clientes gerando respostas dinâmicas e o acesso aos dados numa Base de Dados. A existência de outras aplicações *Web* já desenvolvidas na Bosch leva a não haver uma necessidade de desenvolver todos os componentes da arquitetura de duas camadas. No entanto, de forma a atingir os objetivos definidos para este projeto, foi necessário desenvolver de raiz os elementos Cliente, Aplicação Servidora e Base de Dados.

De acordo com as figuras anteriormente apresentadas, têm-se dois clientes *Web* distintos: as máquinas e os utilizadores. Isto deve-se ao facto de os dados necessários para o funcionamento da aplicação *Web* desenvolvida se encontrarem nas diversas máquinas de pré programação. Desta forma, é necessário desenvolver um programa capaz de obter os dados das máquinas e enviá-los para o servidor, de modo a disponibilizá-los mais tarde à aplicação. Por outro lado, têm-se os utilizadores como outro cliente *Web*. Estes últimos interagem com a aplicação através de navegadores *Web*.

A Aplicação Servidora encontra-se dividida em dois elementos: um conjunto de serviços *Web* e a Aplicação *Web*. Os serviços *Web* são a interface entre as diversas aplicações

e base de dados. Estes permitem a manipulação da informação para e da base de dados através de operações conhecidas como *Create Read Update Delete* (CRUD), que permitem criar, ler, atualizar e apagar a informação da base de dados, respetivamente. A aplicação *Web* é a interface entre os utilizadores e a informação. É a partir dela que os utilizadores terão acesso a toda a informação das máquinas de pré programação: os seus processos, as suas configurações, etc.

Em seguida são apresentadas as especificações separadamente pelos diversos elementos do sistema desenvolvido: o programa de obtenção de dados (*Data Collection*), o servidor e a base de dados e por fim os serviços *Web* e a aplicação.

3.2. *DATA COLLECTION*

3.2.1. OBJETIVOS

O principal objetivo desta componente é recolher dados das máquinas que fazem a pré programação dos IC e enviá-la para o servidor. Para isso, terá de recolher dois tipos de dados: resultados de processos das máquinas e configurações dos programas.

Os resultados de processos das máquinas são um conjunto de dados registados por uma máquina, em ficheiros de *log*, sempre que esta termina a programação de uma determinada quantidade de IC. Destes ficheiros é possível obter dados como a identificação da máquina e de diversos componentes que a constituem, tempos de processo, quantidades de IC inseridos, programados, rejeitados, etc. Dentro destes dados de processo geral, é possível ainda obter alguns detalhes, nomeadamente no que diz respeito às quantidades de IC inseridos, programados e rejeitados. Para além desta informação do processo geral da máquina, é possível ainda obter estes dados relativos a cada programador utilizado num determinado processo da máquina e ainda dos diferentes *sockets* presentes em cada programador.

As configurações dos programas são conjuntos de dados guardados quando se regista um novo programa para posteriormente ser instalado num IC. Depois de ser registado o novo programa, é gerado um ficheiro com informação desse programa. Estes ficheiros incluem informação como os códigos de identificação do programa, dos IC antes e depois de serem

programados, etc. De uma forma geral, a Figura 7 apresenta o diagrama de atividade geral do que se pretende fazer com este componente.

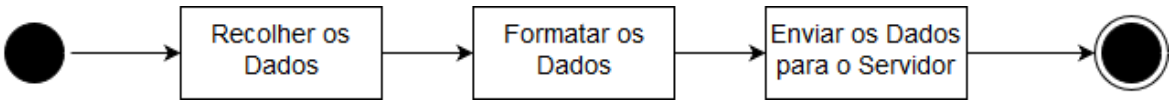


Figura 7 *Data Collection*: Diagrama de Atividade Geral.

Assim, os objetivos desta componente desenvolvida resumem-se aos seguintes:

- Recolher dados de processos;
- Recolher dados de configurações dos programas;
- Manipular os dados de modo que o servidor seja capaz de os perceber;
- Enviar os dados para o servidor.

3.2.2. REQUISITOS

Como é possível perceber pela Figura 6 o componente responsável pela obtenção de dados encontra-se situado nas máquinas de pré programação. Isto significa que, tendo N máquinas, haverá N instalações do *Data Collection*. Assim, é preciso criar um programa em que, caso seja necessário fazer alguma alteração de configuração, não haja a necessidade de fazer alteração do código fonte, uma vez que implicaria uma recompilação do código e uma nova instalação do mesmo. De forma a melhorar o processo de configuração, é preciso forçar a que todas as configurações necessárias possam ser efetuadas através de um ficheiro de configuração externo. Desta forma, quando o programa é iniciado, são carregadas as configurações através do ficheiro externo e, apenas depois, é despoletada a execução da lógica do programa.

As máquinas de pré programação funcionam através de aplicações do fornecedor instaladas em Sistemas Operativos (SO) *Microsoft Windows*. Por isso, este programa deve ser desenvolvido utilizando uma linguagem de programação robusta e, ao mesmo tempo, utilizar poucos recursos das máquinas de modo a não interromper o funcionamento das mesmas. Desta forma, decidiu-se utilizar a linguagem C de forma a desenvolver este

programa. A utilização desta linguagem permite criar um programa leve, robusto, consumidor de poucos recursos e capaz de comunicar com outras aplicações através de redes que utilizem *Internet Protocol* (IP), facilitando a comunicação com os serviços *Web*.

Algumas das máquinas de pré programação já se encontram em produção há muitos anos o que leva a que haja um elevado número de dados registado individualmente em cada uma. De forma a tornar o processo de obtenção de dados mais eficiente é necessário manter um registo de quais os dados que já foram recolhidos e enviados para o servidor de forma a evitar abrir pastas e ficheiros cuja informação é desnecessária, evitando a perda de tempo com esses dados.

Após recolhidos os dados, é necessário formatá-los de forma a ser compreendidos do lado do servidor. Assim, recorre-se à tecnologia *JavaScript Object Notation* (JSON) para transportar os dados entre todas as componentes desenvolvidas neste projeto. JSON trata-se de um formato leve de troca de dados. A sua referência ao *JavaScript* deriva do facto de utilizar sintaxe de *JavaScript* no entanto, o formato JSON é unicamente texto tal como o *eXtensible Markup Language* (XML). Desta forma, pode ser lido e utilizado como um formato de dados por qualquer linguagem de programação e não exclusivamente por *JavaScript* sendo, por isso, o principal formato de dados utilizado na comunicação assíncrona entre navegador e servidor começando a impor-se face ao XML [8] [9].

Esta tecnologia foi desenvolvida de forma a ser capaz de ser não só interpretada pela máquina, mas também pelo homem. À semelhança do XML, encontra-se organizada por uma hierarquia, no entanto não utiliza uma etiqueta final, tornando-se mais rápido de ler e escrever, podendo conter na sua estrutura tipos de dados como por exemplo vetores e objetos. O seguinte exemplo de código demonstra uma estrutura de JSON que envolve todos os tipos de pares nome/valor: texto, números, objetos, vetores, valores booleanos e nulos.

```
{
  “primeiro_nome”: ”John”,
  “vivo”: true,
  “idade”: 22,
  “morada”: {
       “rua”: “Rua de Santa Catarina”,
       “porta”: 123,
       “cidade”: “Porto”
  },
  “telefone”:[
       {
            “tipo”: “casa”,
            “numero”: “229999999”
```

```
        },
        {
              "tipo": "pessoal",
              "numero": "901123456"
        }
  ],
  "filhos": [],
  "conjuge": null
}
```

Assim, os requisitos desta componente desenvolvida resumem-se aos seguintes:

- Recolher dados de processos e configurações de programas;
- Ser configurável a partir de um ficheiro externo;
- Utilizar a linguagem C;
- Comunicar com o servidor através de pedidos HTTP;
- Utilizar JSON como formato de dados para enviar a informação para o servidor;
- Consumir poucos recursos das máquinas;
- Não interromper o normal funcionamento das máquinas;
- Manter um registo temporal atualizado;
- Extra: recolher informação dos programadores.

3.3. SERVIDOR E BASE DE DADOS

3.3.1. OBJETIVOS

O principal objetivo destas componentes é servir de base para todo o sistema em si uma vez que é no servidor que se vai encontrar a BD, os serviços e a aplicação *Web*. Pretende-se utilizar um servidor que seja compatível com a *framework* escolhida para o desenvolvimento do projeto assim como um servidor que seja capaz de suportar a tecnologia utilizada para desenvolver a BD. Pretende-se ainda desenvolver uma base de dados capaz de guardar a informação recolhida pelo *Data Collection*, descrito na subsecção 3.2 e dados

provenientes da aplicação *Web*. A arquitetura da base de dados deve ser desenvolvida com o intuito de manter a maior organização possível dos dados. A tecnologia a utilizar para o desenvolvimento da BD é o SQLite permitindo um elevado desempenho num sistema de base de dados.

Assim, os objetivos deste conjunto de elementos resumem-se aos seguintes:

- Utilizar um servidor compatível com as tecnologias e *frameworks* utilizadas neste projeto;
- Proporcionar uma base estável para suportar outros elementos desenvolvidos;
- Armazenar os dados numa BD organizada;
- Utilizar uma tecnologia da BD que proporcione um elevado desempenho.

3.3.2. REQUISITOS

Limitações já existentes na Bosch, fazem com que seja preciso ter algum cuidado com a escolha do servidor que será responsável por suportar o sistema.

O servidor deve ser *Apache* uma vez que é o utilizado na secção de MFE3 e gerido por uma equipa especializada na área de *software*. Isto faz com que não haja um acesso direto ao servidor e às suas configurações. Desta forma, a aplicação necessita de ser desenvolvida num ambiente semelhante para se perceber as suas funcionalidades e principalmente as suas limitações. Assim, utiliza-se uma aplicação que permite instalar e utilizar um servidor *Web Apache* de forma fácil: o XAMPP.

XAMPP é um servidor *Web Apache open source* que contém diversas ferramentas que permitem um desenvolvimento facilitado, permitindo aos programadores focarem-se no desenvolvimento das suas aplicações. De forma a garantir a compatibilidade com o departamento da secção de MFE3 deverá utilizar-se a versão 5 de PHP e garantir a existência de uma versão atualizada do *Apache*, compatível com a versão do PHP.

A tecnologia usada para a base de dados é o SQLite. Esta tecnologia é um sistema de Bases de Dados (BD) que proporciona armazenamento local para aplicações e dispositivos individuais. Este sistema não é comparado a outros sistemas de BD como MySQL, Oracle ou SQL Server, uma vez que não se tratam de bases de dados do tipo cliente/servidor. SQLite

foi pensado para apresentar um elevado desempenho em situações onde há uma grande necessidade de eficiência, independência ou simplicidade [9].

No caso deste projeto, a escolha de SQLite baseou-se na quantidade de dados que é necessário carregar para a BD. Uma vez que essa quantidade não é significativamente elevada para que seja necessário um servidor de base de dados dedicado, decidiu-se recorrer a esta ferramenta de modo a garantir a simplicidade no acesso à BD. Em termos de estrutura, foi necessário recorrer à equipa de *software* da secção de MFE3, de forma a definir uma série de especificações como os nomes das tabelas, campos e interligações para manter um *standard* nas diferentes bases de dados da secção. Por questões de confidencialidade, a arquitetura da base de dados apresentada neste capítulo e no capítulo 5 não é a completa e está representada por nomes fictícios, não correspondendo aos nomes utilizados na BD desenvolvida.

Assim, os requisitos deste conjunto de elementos resumem-se aos seguintes:

- Utilizar um servidor *Web Apache* [10];
- Garantir a compatibilidade entre *Apache*, *OpenSSL* e PHP [10];
- Utilizar SQLite para o desenvolvimento da BD;
- Desenvolver a BD num formato organizado e escalável.

3.4. SERVIÇOS *WEB* E APLICAÇÃO *WEB*

3.4.1. OBJETIVOS

O principal objetivo destes dois elementos é garantir uma forma de acesso à BD quer através da aplicação que obtém os dados das máquinas quer dos utilizadores da aplicação *Web*.

Especificamente, pretende-se desenvolver serviços *Web* de forma a proporcionar uma interface CRUD para manipular os dados da BD. No caso da aplicação *Web*, pretende-se desenvolver uma aplicação que seja estável, escalável, robusta e intuitiva. A aplicação *Web* deve cobrir todos os requisitos especificados a seguir.

Assim, os objetivos deste conjunto de elementos resumem-se aos seguintes:

- Garantir a comunicação com a BD;
- Desenvolver um conjunto de serviços *Web* que permitam manipular a informação necessária;
- Desenvolver uma aplicação robusta, escalável e intuitiva.

3.4.2. REQUISITOS

Os serviços *Web* devem ser desenvolvidos sobre uma arquitetura *Representational State Transfer* (REST). REST é uma arquitetura *Web* standard que utiliza o protocolo HTTP para a comunicação de dados. Esta arquitetura baseia-se num conjunto de componentes designados recursos capazes de serem acedidos através dos conhecidos métodos *standard* HTTP: *DELETE*, *GET*, *OPTIONS*, *POST* e *PUT* [11]. Isto permite criar uma interface de manipulação dos dados da BD através da técnica CRUD [12]. A aplicação deve proporcionar uma ferramenta mais completa do que apenas a consulta de dados obtidos das máquinas. Desta forma, pretende-se criar um sistema que permita fazer não só a consulta de dados e indicadores, mas também que permita apoiar as restantes atividades das máquinas de pré programação.

No que diz respeito às máquinas de pré programação, o desenvolvimento da aplicação deve incluir um subsistema de Gestão de Máquinas, forma de administrar as mesmas. A partir da aplicação deve ser possível registar uma nova máquina que tenha sido comprada e instalada na Bosch e em seguida validar se o processo de instalação da máquina foi corretamente feito através da validação de uma *checklist*.

No que diz respeito aos processos das máquinas de pré programação deve ser desenvolvido um subsistema de Gestão de Processos. Como é referido nas especificações das componentes de obtenção de dados, do servidor e da base de dados, é guardado um conjunto de informação dos processos das máquinas. No entanto, é necessário trabalhá-lo de forma a obter-se o conjunto de indicadores que se pretende consultar na interface gráfica da aplicação. Desta forma é necessário apresentar os dados em três diferentes indicadores: Produção, Taxa de Produção (TP) e Taxa de Rejeição (TR). O indicador Produção diz respeito ao número efetivo de IC inseridos e retirados da máquina, ou seja, este indicador deverá ser apresentado de forma a comparar o *Output* de unidades face ao *Input* de unidades,

tanto a nível do processo geral da máquina como de forma detalhada nos programadores e sucessivamente nos seus *sockets*. O indicador da Taxa de Produção representa a percentagem de IC que são colocados na saída da máquina e, por isso, que foram programados com sucesso. A fórmula apresentada em seguida representa o cálculo da Taxa de Produção (TP) onde IC_{out} representa o número de ICs que sai da máquina e IC_{in} representa o número de IC que entra na máquina.

$$TP = \frac{IC_{out}}{IC_{in}} \times 100$$

O indicador da Taxa de Rejeição representa a percentagem de IC que não são colocados na saída da máquina e, por isso, que por algum motivo não foram programados com sucesso, levando a sua rejeição. A fórmula apresentada em seguida representa o cálculo da Taxa de Rejeição seguida da demonstração para cálculo da mesma através da utilização da já calculada Taxa de Produção. Na fórmula, IC_{rej} indica o número de IC que a máquina rejeitou.

$$TR = \frac{IC_{rej}}{IC_{in}} \times 100 \Leftrightarrow$$

$$\Leftrightarrow TR = \frac{IC_{in} - IC_{out}}{IC_{in}} \times 100 \Leftrightarrow$$

$$\Leftrightarrow TR = \left(\frac{IC_{in}}{IC_{in}} - \frac{IC_{out}}{IC_{in}}\right) \times 100 \Leftrightarrow$$

$$\Leftrightarrow TR = \frac{IC_{in}}{IC_{in}} \times 100 - \frac{IC_{out}}{IC_{in}} \times 100 \Leftrightarrow$$

$$\Leftrightarrow TR = 100 - TP$$

No que diz respeito às configurações dos programas, quando um programa é instalado pela primeira vez numa máquina é necessário seguir um procedimento para garantir que fica corretamente instalado de acordo com as suas configurações. Durante o processo de instalação do programa é necessário identificá-lo e configurar o conteúdo da etiqueta impressa pelo operador no final da programação dos IC. Este processo é complexo e efetuado por humanos o que pode levar a erros na configuração do programa e, desta forma, à entrega de IC errados à produção para serem inseridos nos produtos e, consequentemente, à perda de dinheiro pela empresa. De forma a minimizar este erro, pretende-se que a

aplicação apresente um modo de validação da configuração feita através de uma *checklist*. Desta forma, a seguir à instalação feita pelo utilizador da secção pretende-se que o mesmo execute o *Data Collection* para atualizar os dados da base de dados e, em seguida, aceda à aplicação para efetuar a validação do novo programa que instalou e configurou.

É comum fazerem-se atualizações de *software* e *firmware* ou surgirem avarias nas máquinas. Para estes e outros eventos, pretende-se desenvolver um subsistema de Gestão de Eventos na aplicação *Web* para registo e consulta de eventos. O objetivo deste desenvolvimento é manter um registo de acontecimentos das máquinas de pré programação para se conseguir perceber melhor quais as possíveis razões para eventuais problemas que possam surgir na programação dos IC.

Associados às máquinas de programação encontra-se uma série de documentos, entre eles os seus manuais e diversos procedimentos. Assim, pretende-se que seja desenvolvido um subsistema de Gestão de Documentos na aplicação *Web* de forma a ser possível registar e consultar documentos, associando-os às diferentes máquinas. Isto permitirá ter acesso aos diversos manuais e procedimentos a partir da mesma plataforma que se utiliza para consultar indicadores, fazer a validação de máquinas ou configurações de processo. Desta forma, não há necessidade de recorrer a aplicações ou localizações na rede diferentes.

Por fim, pretende-se que a aplicação tenha autonomia para gerar notificações via e-mail em duas situações: quando o estado de uma máquina é alterado para ser utilizada em produção sem estar devidamente validada e quando é validada uma configuração de um programa cuja informação não se encontra corretamente configurada. Estes e-mails devem ser enviados para o administrador, mas também para o utilizador que poderá eventualmente fazer alguma das ações mencionadas anteriormente. Desta forma, é possível notificar o utilizador de que fez uma ação que não está correta podendo retificar a ação.

O desenvolvimento da aplicação *Web* deve recorrer às tecnologias *HyperText Markup Language* (HTML), *Cascade Style Sheets* (CSS) e *JavaScript* (JS) para lidar com a lógica da apresentação e *Hypertext Preprocessor* (PHP) para trabalhar a lógica da programação.

Desde a primeira versão disponibilizada em 1993 que HTML é o formato de ficheiros capaz de ser interpretado pelos navegadores *Web* de forma a apresentar informação. Esta linguagem é composta por diferentes elementos, sendo cada um deles responsável por

diferentes funcionalidades. Com a evolução das tecnologias, há também uma necessidade de proporcionar ao utilizador novas funcionalidades e uma interação mais dinâmica. Desta forma, foram surgindo diversas versões desta linguagem, apresentando-se agora na quinta versão [13].

O HTML5 é identificado através da utilização do chamado *doctype*, incluído no início do ficheiro de uma forma simples:

```
<!DOCTYPE html>
```

Esta versão introduz uma série de novos elementos como <*svg*> e <*canvas*> para manipulação gráfica, elementos de multimédia como <*audio*> e <*video*> que permitem disponibilizar conteúdos de multimédia de forma mais simples ou até novos atributos possíveis de ser utilizados em diversos elementos presentes em formulários como datas, tempos, calendários, etc [14]. Juntamente com *Cascade Style Sheets* (CSS) e *JavaScript* torna a experiência de utilização de uma aplicação *Web* interativa, moderna e única.

Cascade Style Sheets (CSS) é uma linguagem pensada para integrar estilos em documentos *Web* e, por isso, descreve como é que os elementos utilizados em HTML devem ser apresentados. Uma das grandes vantagens na utilização de CSS é e divisão da componente de apresentação da componente de conteúdo, permitindo uma melhor divisão do trabalho e, por isso, da flexibilidade das aplicações [15] [16].

A sintaxe utilizada é composta por blocos, onde cada bloco diz respeito a um ou mais elementos para os quais se estabelecem as regras de estilo. Cada bloco é composto por pares nome-valor que permite manipular as diversas características do elemento. O excerto a baixo é um exemplo de como trocar a cor de fundo de uma página *Web* [16].

```
body{ background-color: blue; }
```

O nome *Cascade Style Sheet* deriva do facto de ser possível alterar propriedades de um elemento em cascata, ou seja, um elemento específico que se encontra dentro de outros. No excerto a baixo é apresentado um exemplo de alteração da cor de fundo de um botão que se encontra dentro do elemento *body* [16].

```
body button{ background-color: white; }
```

JavaScript é uma linguagem de programação que permite proporcionar conteúdo *Web* dinâmico. Desde a sua primeira integração em aplicações *Web*, que esta linguagem é processada pelos navegadores *Web* e, por isso, do lado do cliente. No entanto, hoje em dia

já é possível utilizar JS do lado do servidor recorrendo a novas tecnologias como Node.js [17] [18].

Trata-se de uma linguagem orientada a objetos, mas apesar de incluir sintaxe que permite definir classes, os objetos desta linguagem não são baseados em classes como em outras linguagens como *Java* ou PHP. Em *JavaScript* os objetos podem ser criados de diversas formas através da escrita literal do mesmo ou através de construtores criando-os e executando uma determinada ação em seguida. Um construtor é uma função composta pela propriedade *"prototype"* que é utilizada para implementar a herança da base de um protótipo e que partilha das suas propriedades. Desta forma, a criação dos objetos para o programador pode ser feita da mesma forma que em outras linguagens de programação através da expressão "*new*" [17].

À semelhança do CSS, esta linguagem também permite alterar as características dos elementos utilizados em HTML. Mas de forma comum a outras linguagens de programação, também permite fazer essas alterações com base em prévias verificações do estado anterior. Por exemplo, quando um utilizador clica numa *checkbox* de modo a alterar o seu estado é necessário verificar antes qual o estado atual da mesma. Este tipo de operações é possível de ser executada através de JS. Veja-se o exemplo a seguir:

```
var box = document.getElementById('checkbox');

box.on('click, function(){
   if(box.checked){
        box.checked = false;
   }else{
        box.checked = true;
   }
});
```

PHP é uma linguagem de *scripting* desenvolvida principalmente para utilização em ambientes *Web*, podendo também ser executada através de uma linha de comandos e, assim, ser utilizada para desenvolver aplicações *standalone*. Quando utilizada para desenvolvimento *Web*, o código desenvolvido é embebido no HTML e interpretado ainda do lado do servidor sendo gerada uma página *Web* com os dados processados [19].

De forma a utilizar-se PHP, é necessário indicar onde começa e onde termina o código. Dessa forma são utilizados os elementos identificativos da linguagem de PHP, da seguinte forma:

```
<?php
   // Código PHP aqui
?>
```

À semelhança de *JavaScript*, PHP é uma linguagem orientada a objetos. No entanto, neste caso é possível criar objetos através de classes desenvolvidas pelo programador. Isto permite ao programador reutilizar código sem ter de o reescrever no seu código. Sendo uma linguagem processada do lado do servidor, permite estabelecer a ponte entre a componente visual de uma aplicação e os dados que a compõem. Dito de outra forma, esta linguagem permite aceder a sistemas de bases de dados, de forma a compor a aplicação antes de ser enviada para o cliente. Isto faz com que PHP se torne uma linguagem poderosa proporcionando maior rapidez e fluidez ao utilizador final, quando interage com a aplicação.

Pretende-se que o desenvolvimento dos serviços e da aplicação *Web* sejam efetuados recorrendo a uma *framework*, com o objetivo de melhor organizar as diferentes componentes. Assim, os requisitos deste conjunto de elementos resumem-se aos seguintes:

- Desenvolver um conjunto de serviços *Web* REST;
- Desenvolver uma aplicação *Web* com uma interface gráfica intuitiva para os seus utilizadores;
- Utilizar a linguagem PHP para desenvolver os serviços e a aplicação *Web*;
- Desenvolver os serviços e a aplicação *Web* recorrendo a uma *framework* de PHP;
- Desenvolver todos os subsistemas especificados e respetivas funcionalidades;
- Desenvolver os indicadores Produção, Taxa de Produção e Taxa de Rejeição;
- Garantir que os indicadores dos processos das máquinas são acessíveis por qualquer tipo de utilizador;
- Extra: desenvolver funcionalidade gráfica para apresentar indicadores ao nível dos programadores e *sockets*.

3.4.3. *USE CASES* (UC)

Os *Use Cases*, ou em português Casos de Uso, são uma ferramenta que descreve como diferentes tipos de utilizadores interagem com um determinado sistema para

solucionar um problema. Com esta ferramenta é possível obter informações como os propósitos dos utilizadores, as interações entre os utilizadores e o sistema e o comportamento necessário do sistema para satisfazer os objetivos. O diagrama de UC desenvolvido apresenta-se na Figura 8.

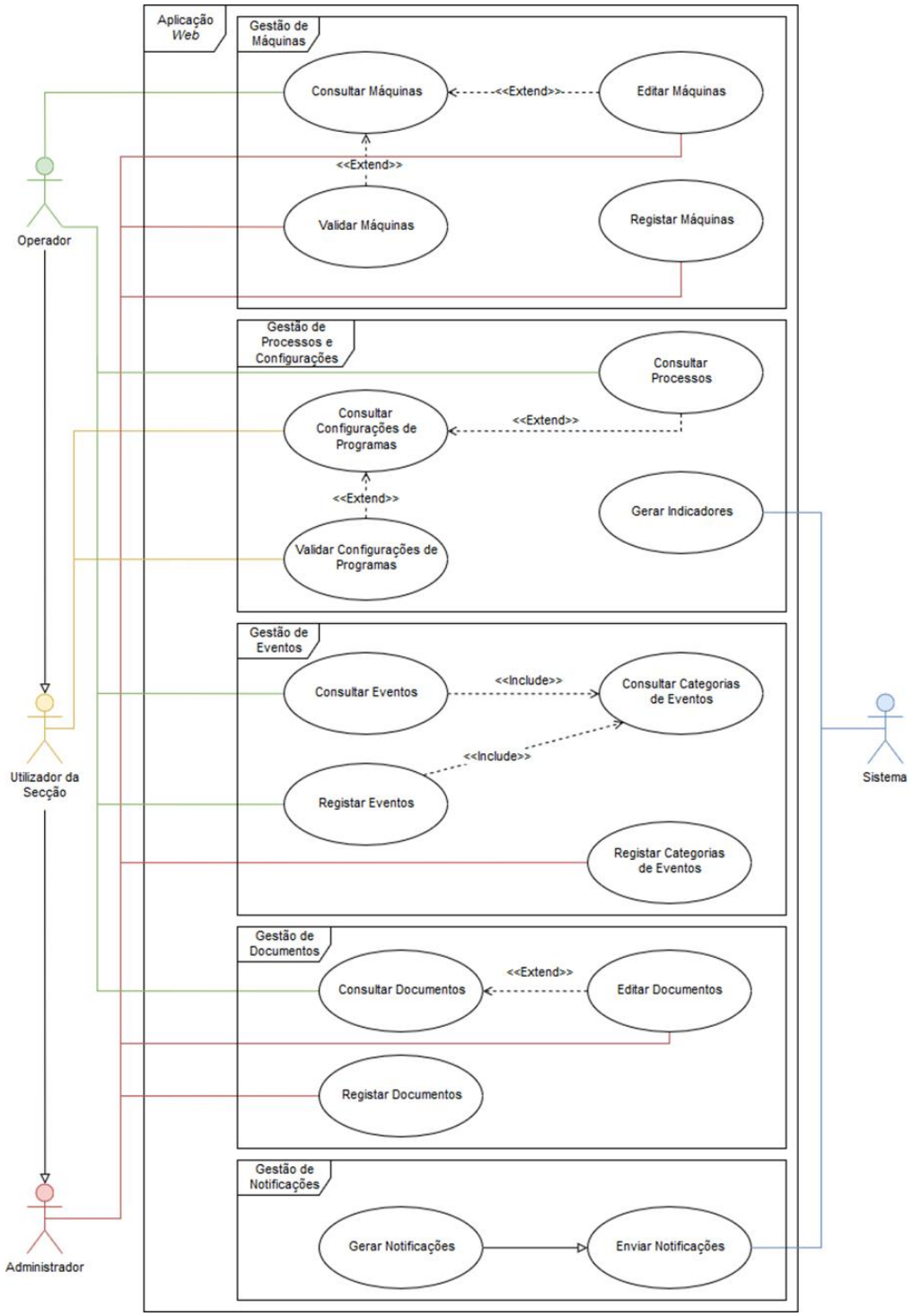


Figura 8 Diagrama de Casos de Uso

A aplicação desenvolvida apresenta quatro atores diferentes: o Operador, o Utilizador da Secção, o Administrador e o Sistema. Como é possível de perceber através da interpretação da Figura 8, existem diversos casos de uso definidos e nem todos os atores interagem com todos eles. Esta interação está representada pelas cores verde, amarelo, vermelho e azul apenas por questões de estética e de melhor compreensão do diagrama de forma a representar a interação dos atores Operador, Utilizador da Secção, Administrador e Sistema, respetivamente. É ainda possível perceber a existência de diferentes subsistemas: Gestão de Máquinas, Gestão de Processos, Gestão de Eventos, Gestão de Documentos e Gestão de Notificações.

O Operador é o ator cujo nível de autorização de acesso aos dados é menor, podendo apenas efetuar consultas. O Utilizador da Secção é um ator que herda as propriedades do Operador com o acréscimo de poder efetuar a validação das configurações dos programas. O Administrador herda as propriedades do Utilizador da Secção com o acréscimo de poder administrar todos os casos de uso definidos. O ator Sistema não representa um tipo de ator humano à semelhança dos restantes, apenas é responsável pelo cálculo e a geração automática dos indicadores necessários assim como o envio das notificações quando há essa necessidade.

3.5. *FRAMEWORKS*

O desenvolvimento de *Web* pode tornar-se complicado quando se desenvolvem aplicações de maiores dimensões ou complexidade. Uma *framework* é uma ferramenta que permite tornar este processo mais simples. Trata-se de um *software* que proporciona um ambiente base reutilizável e algumas funcionalidades que funcionarão como parte integrante da aplicação a desenvolver. O mercado do *software* está em constante desenvolvimento e, por isso, existem imensas *frameworks* diferentes e que podem ser aplicadas e diversos tipos de aplicações. Estas *frameworks* costumam incluir ferramentas como suporte para arquiteturas de *software*, compiladores, bibliotecas, *Application Programming Interface* (APIs), entre outras [20].

Nesta secção são apresentadas e comparadas quatro *frameworks* de PHP que, com base num primeiro estudo realizado pelo *website SitePoint* em 2015 e num segundo realizado pelo *website Coder Eye* em 2017, foram consideradas relevantes: *Laravel*, *Symfony*, *CakePHP* e *CodeIgniter*. O primeiro estudo contou com aproximadamente 7800 respostas e foi dividido em várias vertentes: utilização da *framework* para fins de trabalho, utilização da *framework* para projetos pessoais, utilização da *framework* em diferentes países, etc. A Figura 10 e Figura 9 apresentam os resultados desse estudo para as categorias da utilização da *framework* para projetos profissionais e pessoais filtrados pelas quatro *frameworks* apresentadas anteriormente [20].

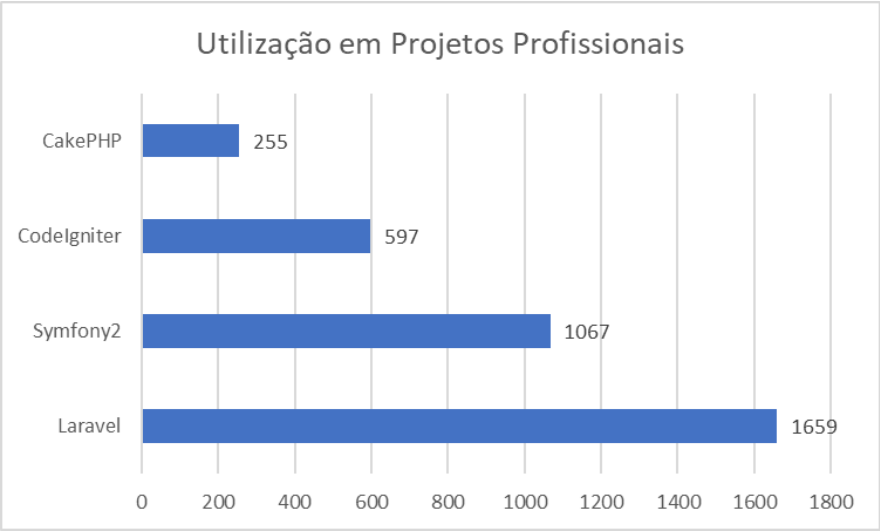


Figura 9 Comparação entre frameworks em projetos profissionais.

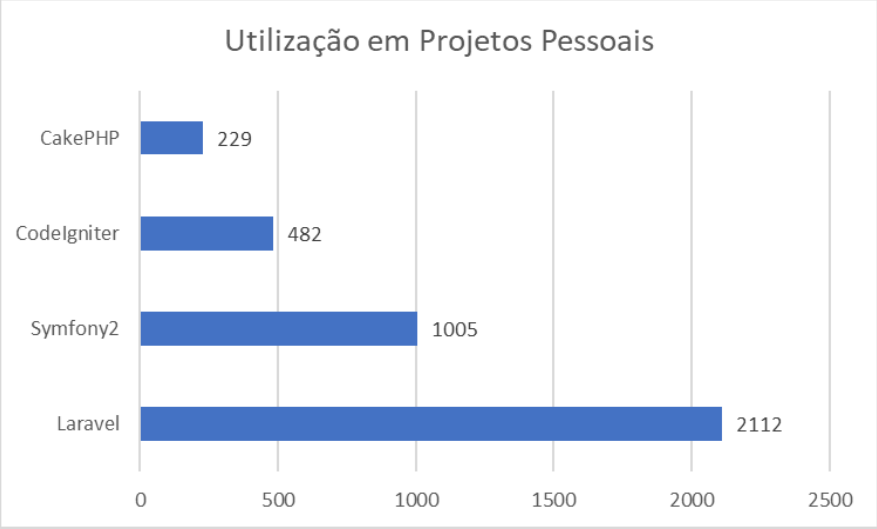


Figura 10 Comparação entre frameworks em projetos pessoais.

O segundo estudo contou com aproximadamente 7500 respostas e, ao contrário do primeiro, não foi dividido em duas vertentes sendo analisado de uma forma geral para a utilização das diversas *frameworks* em projetos. Figura 11 apresenta os resultados deste estudo [21].

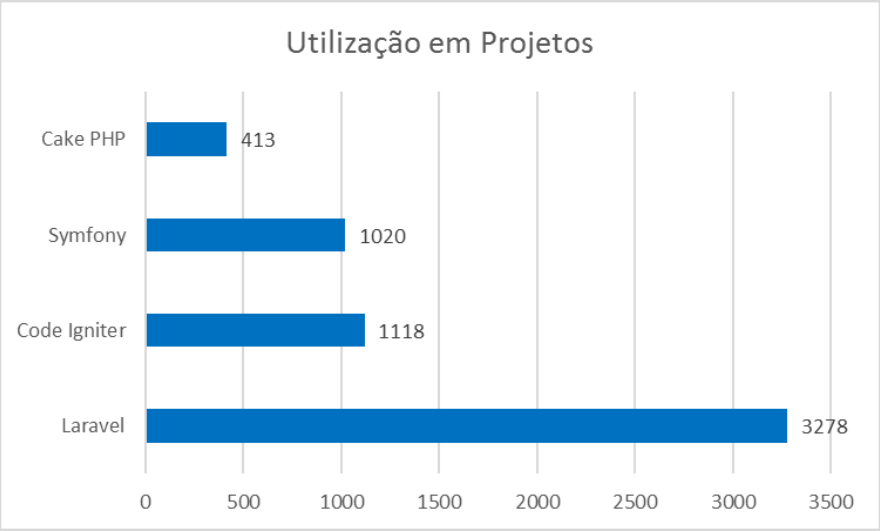


Figura 11 Comparação entre frameworks (segundo estudo).

3.5.1. LARAVEL

Laravel é uma das *frameworks* de PHP mais utilizadas por programadores em todo o mundo. É uma *framework* de fácil instalação e compatível com os sistemas operativos (SO) *Windows*, *Linux* e *Mac*. *Laravel* tem um servidor integrado, o que permite correr a aplicação sem recorrer a servidores externos. No entanto, é possível integrar as aplicações criadas com esta *framework* em servidores externos como o *Apache* ou Nginx. Para trabalhar com esta *framework*, é necessário recorrer a um sistema de gestão de dependências. No caso da *Laravel* é o *Composer*. Esta ferramenta não só permite tornar a instalação da *framework* num processo simples, mas também gerir dependências do projeto [10].

Como grande parte das *frameworks* utilizadas para o desenvolvimento *Web*, está organizada numa arquitetura *Model View Controller* (MVC). Esta arquitetura permite dividir o desenvolvimento em três partes: o *Model* responsável por mapear e interagir com a base de dados; a *View* que representa a parte da apresentação dos dados ao utilizador; o *Controller* que é a parte responsável por controlar os dados de toda a aplicação e, por isso, interagir tanto com o *Model* para aceder à base de dados como com a *View* para apresentar os dados ao utilizador. Com base nesta arquitetura, as aplicações *Web* apresentam o conceito de

Object-Relational Mapping (ORM). Isto permite que a aplicação crie uma camada virtual de mapeamento da base de dados o que faz com que seja possível aceder ao conteúdo dela a partir de estruturas de dados como objetos. Desta forma, é importante também saber quais as tecnologias de base de dados com as quais esta *framework* funciona, sendo elas MySQL, PostgreSQL, SQLite e SQL *Server* [10].

Laravel também contém um motor de *Views*, ou seja, uma tecnologia que é responsável por criar ficheiros HTML a partir das *Views* que são criadas pelo programador. No entanto, sendo *Laravel* uma *framework* de PHP, este motor de *Views* cria ficheiros PHP. O conceito assenta na criação de *templates* e reutilização de código. No caso da *Laravel*, este motor é o *Blade*. Tendo o acesso à informação da base de dados e as *Views* criadas para apresentar a informação ao utilizador final, é importante para o programador saber qual o tipo de dados predefinido que é utilizado pela aplicação. Aqui é utilizado *JavaScript Object Notation* (JSON) encapsulado em vetores, ou seja, o programador pode passar a informação para as *Views* utilizado vetores que no seu conteúdo apresentam objetos de JSON [10].

Sendo uma das *frameworks* de PHP mais utilizadas em todo o mundo, *Laravel* tem um suporte técnico bastante ativo, não só a partir da constante atualização de documentação, mas também no fórum do seu website apresentando uma página dedicada à explicação e discussão de diversos conceitos e demonstrações de como utilizar a *framework*. Isto permite tornar o processo de aprendizagem mais simples, o que leva a uma melhor utilização da própria *framework* [10].

3.5.2. SYMFONY

Symfony é uma das mais conhecidas *frameworks* de PHP. Para utilizar esta *framework* é possível instalá-la de duas formas: uma primeira através da descarga do instalador no website ou de uma segunda através do *Composer*, já apresentado na *framework Laravel*. Isto significa que esta *framework* também pode utilizar este gestor de dependências e, desta forma, incorporar outras ferramentas no projeto. *Symfony* é compatível com os sistemas operativos *Windows*, *Linux* e *Mac* e pode ser configurada para ser utilizada com servidores como o *Apache* ou Nginx. No entanto, é possível utilizar também o servidor que é acompanhado da instalação do PHP, não sendo aconselhado utilizá-lo quando a aplicação se encontra numa fase de produção [22].

Esta *framework* também utiliza uma arquitetura MVC o que permite uma melhor estruturação do código para futuras integrações. *Symfony* utiliza a biblioteca *Doctrine* para implementar o seu sistema de ORM e, desta forma, conseguir dar suporte a bases de dados como MySQL, PostgreSQL e SQL *Server* [22].

À semelhança de *Laravel*, esta *framework* também inclui um motor de *views*: o *Twig*. A partir dele é possível passar variáveis dos *Controllers* para apresentar a informação sob o formato de HTML para posteriormente ser interpretado por um navegador *Web*. Ainda à semelhança de *Laravel* são utilizados vetores para passar a informação para as *Views* [22].

No que diz respeito ao suporte ao desenvolvimento, esta *framework* é acompanhada pela sua documentação presente no website e de uma comunidade onde é possível esclarecer dúvidas.

3.5.3. CAKEPHP

É uma *framework* de fácil instalação, embora mais complexa do que *Laravel*, e compatível com os sistemas operativos *Windows*, *Linux* e *Mac*. *CakePHP* e, por isso, para utilizar esta *framework* é necessário configurar um servidor externo como por exemplo *Apache*, Nginx, *LightHTTPD* ou *Microsoft* IIS e em seguida copiar a base de desenvolvimento da *framework* utilizando a ferramenta de gestão de dependências *Composer*, como *Laravel* [23].

À semelhança de outras *frameworks* utilizadas para desenvolvimento *Web*, *CakePHP* também utiliza uma arquitetura MVC. Esta *framework* utiliza um sistema ORM próprio com base em *Active Record*, um padrão de mapeamento de bases de dados baseados em objetos. Desta forma, *CakePHP* permite trabalhar com bases de dados como MySQL, PostgreSQL, SQL *Server* e SQLite [23].

No que diz respeito às *Views*, *CakePHP* não tem um motor de *Views*. Esta *framework* apresenta as *Views* através dos *Controllers*, recebendo as variáveis através da implementação de código PHP num ficheiro de HTML. Assim, o programador pode utilizar os diversos tipos de variáveis para disponibilizar a informação que são dispostas pela linguagem PHP [23].

Esta *framework* conta com um bom suporte técnico através da documentação no seu *website*, uma série de vídeos que, organizados sob o modelo de *workshops*, ajudam à melhor perceção do funcionamento da *framework* e uma comunidade de utilizadores da *framework*.

3.5.4. CODEIGNITER

CodeIgniter é uma *framework* muito simples de instalar. Para isso, basta descarregar um ficheiro do seu *website*, descompactá-lo e copiar os ficheiros para o servidor. Em seguida aceder ao ficheiro de configuração e escrever o *Uniform Resource Locator* (URL) base da aplicação. Assim, é possível compatibilizar esta *framework* com sistemas operativos como *Windows*, *Linux* ou *Mac*. É possível de perceber que esta *framework* não inclui um servidor integrado pelo que é necessário recorrer a um servidor externo. *CodeIgniter* é compatível com vários servidores, desde que obedeçam aos requerimentos da *framework*: drivers de conexões à base de dados que se utilizará. Desta forma é possível utilizar esta *framework* com a maioria dos servidores já apresentados nomeadamente *Apache* e *Microsoft* [24].

À semelhança de outras *frameworks*, *CodeIgniter* também utiliza uma arquitetura MVC e um sistema ORM *Active Record*. No entanto, esta *framework* não tem um motor de V*iews*, pelo que estes componentes são desenvolvidos como páginas de PHP *standard*. O sistema de ORM desta *framework*, permite mapear bases de dados como MySQL, *Oracle*, PostgreSQL, SQL *Server*, SQLite, CUBRID ou ODBC [24].

Semelhante à *framework Symfony*, no que diz respeito ao suporte ao desenvolvimento, esta *framework* é acompanhada pela sua documentação presente no *website* e de uma comunidade onde é possível esclarecer dúvidas.

3.5.5. ESCOLHA DA *FRAMEWORK*

É possível perceber que as *frameworks* de desenvolvimento *Web*, tendo como finalidade o mesmo objetivo, acabam por se diferenciar em alguns pormenores, que deverão ser tidos em conta de acordo com os requisitos da aplicação que se pretende desenvolver. Na Bosch *Car Multimedia* em Braga, utiliza-se PHP para o desenvolvimento *Web* e servidores *Apache* para alojar as diversas aplicações. Desta forma, qualquer uma das *frameworks* apresentadas anteriormente pode ser utilizada. No entanto, é importante ter em conta não só os aspetos técnicos, mas também aspetos como o suporte e a facilidade de obtenção de informação para o desenvolvimento. Conclui-se que *Laravel* e *CakePHP* são *frameworks* com um grande suporte técnico, apesar de a taxa de utilização da *framework* ser muito mais elevada no caso de *Laravel*. Por sua vez, *CodeIgniter* é uma *framework* muito leve o que melhora o desempenho da aplicação no servidor, ao contrário de *Laravel* que, ao aumentar

a facilidade do utilizador aumenta também a sua própria complexidade levando a tornar-se mais pesada.

Durante os primeiros dois meses do desenvolvimento deste projeto a Bosch proporcionou uma formação em *Laravel*, pelo que no final, pediu que a *framework* fosse utilizada para o desenvolvimento da aplicação.

3.6. *Model View Controller* (MVC)

Model View Controller é um padrão de arquitetura de *software* desenvolvido com o objetivo de separar a camada de interação com o utilizador e a camada de representação de dados. A diferenciação das responsabilidades contribui, desta forma, para que o código desenvolvido seja completamente isolado na construção das interfaces gráficas, modularizando e simplificando o desenvolvimento e, futuramente, facilitando a manutenção da aplicação [25] [26]. Neste padrão há três objetos chave: o Modelo (*Model*), o Controlador (*Controller*) e a Vista (*View*) A Figura 12 mostra a relação entre os três objetos do MVC.

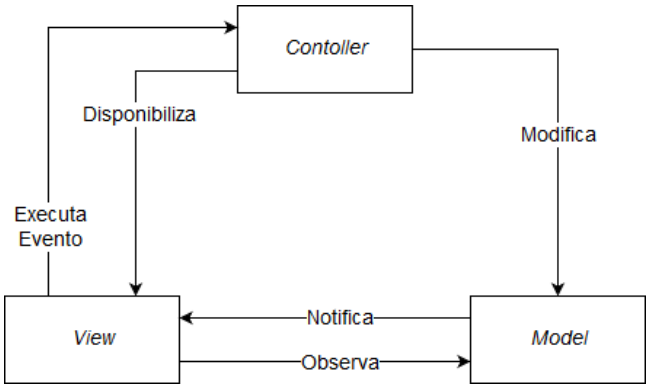


Figura 12 Arquitetura MVC.

De uma forma simplificada, o modelo trata-se de um conjunto de classes responsáveis por guardar os dados e as subsequentes regras a ele aplicadas, a vista é responsável por disponibilizar a interface gráfica ao utilizador e o controlador faz a gestão da interação entre os outros dois objetos, o modelo e a vista. De seguida são detalhados os diversos elementos [27].

3.6.1. CONTROLLER

Em qualquer aplicação *Web*, os controladores apresentam um papel de grande relevância, nomeadamente nas aplicações *Laravel*. Estes são os responsáveis por interpretar os dados enviados através de pedidos HTTP e preparar o modelo utilizado pelas vistas na geração do HTML. É nestes elementos que é feita a implementação dos métodos responsáveis por toda a lógica da programação [25].

3.6.2. VIEW

As vistas são o componente com os quais o utilizador interage. Estes elementos são responsáveis pela construção das interfaces gráficas através dos dados fornecidos pelo modelo e que lhe são transmitidos pelo controlador. Numa aplicação *Web*, as vistas são os elementos responsáveis por gerar o HTML que, posteriormente, será apresentado num navegador *Web*. Assim, não é permitido aceder diretamente a uma vista uma vez que esta é sempre gerada a partir de um controlador [25].

3.6.3. MODEL

Qualquer aplicação *Web* necessita de dados para poder ter conteúdo a disponibilizar e por isso se recorrem a Bases de Dados (BD). O modelo é o elemento do MVC que permite o acesso aos dados guardados na BD. Neste elemento são disponibilizados os métodos que permitem fazer a manipulação dos dados tanto no sentido *Controller*-BD como no sentido inverso [25].

A implementação de sistemas ORM, permite que seja criada uma camada transparente de mapeamento da BD e, desta forma a partir do modelo, seja possível aceder ao conteúdo da BD recorrendo a estruturas de dados como objetos.

3.6.4. APLICAÇÃO DO MVC NA *FRAMEWORK* LARAVEL

Apesar de a estrutura organizacional de *Laravel* ser baseada em MVC, esta *framework* apresenta algumas diferenças do padrão tradicional. Como é possível ver a partir da Figura 13, esta *framework* apresenta mais um elemento na sua arquitetura, o *Route*. Este elemento é responsável por receber os pedidos HTTP e encaminhá-los para o respetivo *Controller*.

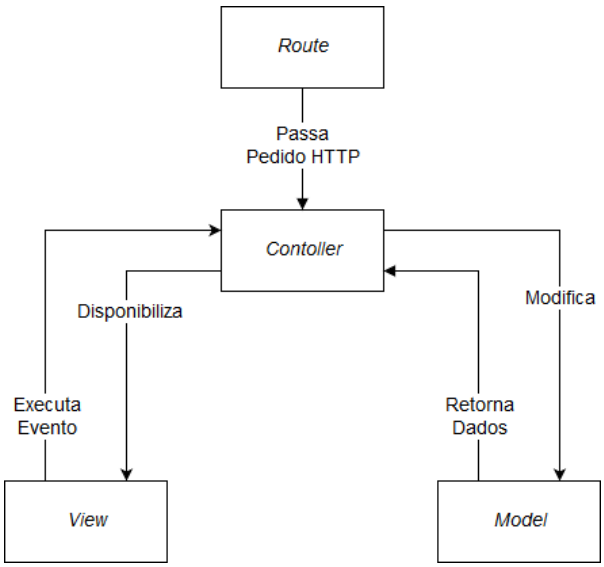


Figura 13 Arquitetura MVC na *framework Laravel*.

Como é apresentado na Figura 13, no padrão utilizado por esta *framework*, não há uma interação direta entre o *Model* e a *View*. Essa interação é sempre efetuada recorrendo ao *Controller* e, por isso, este será responsável pela atualização dos dados da *View*.

3.7. SÍNTESE

É fundamental fazer uma escolha acertada das tecnologias a utilizar no desenvolvimento de *software*, de forma a tornar o projeto num sucesso. O sistema deve ser desenvolvido com base no modelo cliente/servidor a duas camadas onde do lado do cliente existirão as diversas máquinas de pré programação a recolher e enviar os dados através de um programa desenvolvido para o efeito e ainda os utilizadores que, através de navegadores *Web*, poderão aceder à aplicação *Web* desenvolvida. Do lado do servidor deverá ser desenvolvido um conjunto de serviços *Web* que serão responsáveis por realizar operações CRUD e, por isso, capazes de receber os dados das máquinas e guardá-los numa base de

dados. Ainda do lado do servidor, deverá ser desenvolvida uma base de dados responsável por armazenar os dados, mantendo-os de forma centralizada. Por fim do lado do servidor, deverá ser desenvolvida uma aplicação *Web* a qual pode aceder à base de dados diretamente ou através dos serviços *Web* criados de forma a evitar repetição de código e à qual poderá ser acedida através de navegadores *Web* por diversos utilizadores.

O *Data Collection* tem como objetivo recolher os dados dos processos e respetivas configurações das diversas máquinas de pré programação e enviá-las para o servidor. Este programa deve ser desenvolvido recorrendo à linguagem C de forma a gastar poucos recursos das máquinas.

O servidor *Web* deve ser *Apache* uma vez que é o que suporta todas as restantes aplicações da secção de MFE3. Recorrendo à utilização do servidor da secção tem algumas limitações, levando a que não seja possível utilizar a versão mais recente de PHP, havendo a obrigatoriedade de todo o conteúdo que seja necessário desenvolver nessa linguagem seja assente na versão 5. A base de dados deve ser desenvolvida recorrendo ao SQLite, devendo apresentar-se organizada de tal forma que garanta a sua escalabilidade.

Os serviços têm como objetivo proporcionar uma interface de acesso e manipulação de dados da BD e, por isso, devem ser desenvolvidos de forma a que seja possível realizar operações CRUD. É importante desenvolver os serviços de forma independente entre eles de forma a garantir que futuras alterações individuais não afetem os restantes. Os serviços *Web*, deverão ser desenvolvidos recorrendo à definição de uma rota "/api" a partir da *framework Laravel* que se utilizará também para fazer o desenvolvimento da aplicação *Web*. No caso da aplicação *Web*, esta funcionará como a interface gráfica de interação dos diversos utilizadores com os dados guardados na BD. Deverá ser desenvolvida de forma a garantir a cobertura de todos os casos de uso especificados neste capítulo e apresentar uma interface gráfica intuitiva para os seus utilizadores. O seu desenvolvimento deverá ser feito recorrendo à *framework Laravel*.

As comunicações entre os diversos componentes do sistema devem ser efetuadas recorrendo ao protocolo HTTP e os dados devem ser sempre formatados para o formato de dados JSON, de forma a garantir um *standard* na comunicação utilizada em todo o sistema.

4. DATA COLLECTION

Este capítulo apresenta os desenvolvimentos efetuados a nível do programa responsável por recolher os dados das máquinas de pré programação, o *Data Collection*, de forma a proporcionar a informação necessária para o correto funcionamento do sistema.

O *Data Collection* é a base do sistema desenvolvido neste projeto. É este *software* que, após instalado nas diversas máquinas de pré programação, recolhe a informação necessária para que o resto do sistema possa funcionar de acordo com as especificações definidas. Este programa foi desenvolvido na íntegra, uma vez que a Bosch ainda não tinha nada que efetuasse a recolha dos dados que precisava. Após ser recolhida a informação necessária, recorreu-se à biblioteca *Parson*, deforma a integrar a tecnologia JSON na linguagem C [28]. Através desta biblioteca é possível formatar os dados para posteriormente enviá-los para o servidor através de *sockets*.

O *Data Collection* foi desenvolvido de forma a executar a sequência de acontecimentos apresentada na Figura 14.

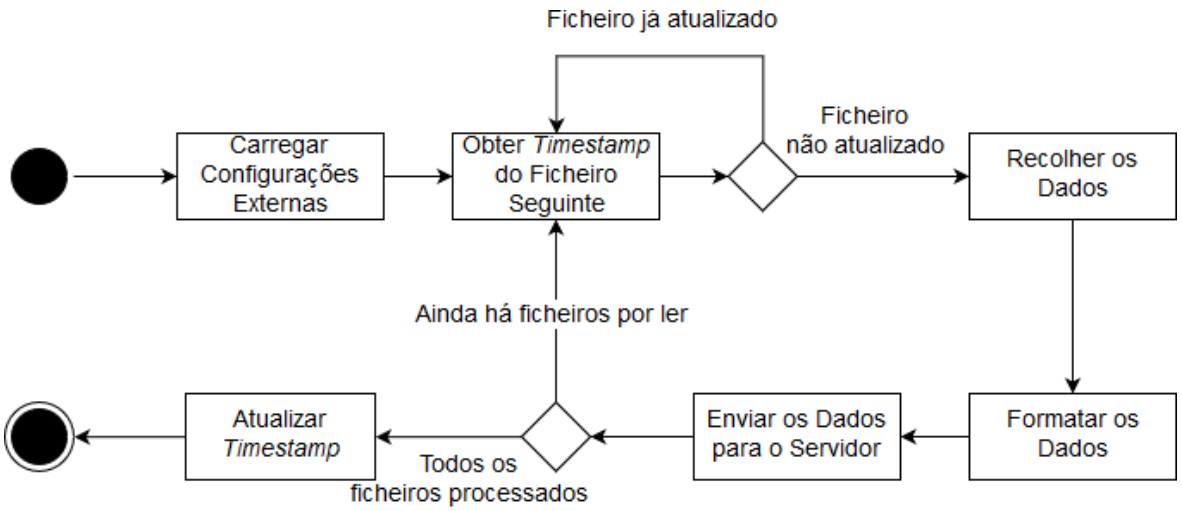


Figura 14 Data Collection: Diagrama de Atividade Pormenorizado.

4.1. MODOS DE EXECUÇÃO

O programa foi desenvolvido de forma a ser executado a partir de uma linha de comandos e dependendo de um argumento. Esse argumento permite executar o *Data Collection* de quatro formas distintas.

Como já foi referido em capítulos anteriores, o *Data Collection* tem como objetivo recolher os dados dos processos e respetivas configurações, presentes nas máquinas de pré programação de IC. A recolha de informação dos processos ou das configurações dos programas envolve ficheiros distintos e que se encontram guardados em localizações diferentes. Isto faz com que o *Data Collection* seja desenvolvido de forma a despoletar fluxos de atividade distintos, dependendo da informação que se pretende recolher. Assim, criaram-se dois modos de recolha dos dados necessários distintos: um para recolher dados de processos e outro para recolher as configurações dos programas. Estes dois modos criados são executados através dos argumentos 101 e 102, respetivamente, permitindo que o *Data Collection* efetue a recolha dos dados necessários.

Os modos criados, foram pensados para não serem verbosos, ou seja, para não apresentarem informação na linha de comandos à medida que correm. Isto permite que o programa não se preocupe em lidar com os *standard outputs*, focando-se em executar as suas tarefas de forma rápida e eficaz. No entanto, foram desenvolvidos mais dois modos de

execução do *Data Collection*. Estes dois novos modos permitem tornar o programa verboso e, por isso, quando está a ser executado apresenta mensagens do estado do programa, qual a informação que está a ser recolhida e qual a informação que está a ser enviada para o servidor. Este processo é normalmente denominado por *debug*, e é um método que permite perceber os diversos acontecimentos de um programa à medida que este está a funcionar. Para utilizar estes dois modos, manteve-se a mesma nomenclatura utilizada para os primeiros e, por isso, é possível fazer a recolha da informação dos processos e das configurações dos programas através dos argumentos 1 e 2, respetivamente. A Tabela 3 apresenta um resumo dos modos existentes no *Data Collection*.

Tabela 3 Data Collection: Modos de Execução.

Modos	**Descrição**
1	Executa o programa em *debug* recolhendo dados de processos. Imprime num terminal informação à medida que o programa corre.
2	Executa o programa em debud recolhendo dados de configurações de programas. Imprime num terminal informação à medida que o programa corre.
101	Executa o programa de recolha de dados de processos.
102	Executa o programa de recolha de dados de configurações de programas.

4.2. CONFIGURAÇÕES E INSTALAÇÃO

Como é descrito no capítulo 3 das Especificações, o *Data Collection* tem de ser configurado através de ficheiros externos de forma a evitar a sua recompilação. Uma vez que se utiliza em todo o sistema o formato de dados JSON para transportar a informação entre as diversas componentes, decidiu-se utilizar o mesmo formato de dados para manter as

configurações do programa. O excerto de código apresentado a seguir, representa uma aproximação da estrutura criada no ficheiro de configuração deste *software*.

```
{
   "TIMESTAMP_FILE":"timestamps.json",
   "MACHINE_CODE":100,
   "MACHINE_NUMBER":5,
   "MAX_PROGRAMMERS":20,
   "PROCESS_DIR":"C:\\Process\\",
   "CONFIG_DIR":"C:\\Process_Configurations\\",
   "SVR_ADDRESS":"192.168.1.22",
   "SVR_PORT":80,
   "SVR_SERVICE":[
      "/api",
      "/api/config"
   ]
}
```

O campo TIMESTAMP_FILE especifica qual o ficheiro que deve ser utilizado para manter um registo atualizado dos *timestamps* do último processo ou configuração de programa que foi recolhido e enviado para o servidor. Os campos MACHINE_CODE, MACHINE_NUMBER e MAX_PROGRAMMERS permitem que o programa identifique a máquina e seus programadores e, dessa forma, despolete diferentes algoritmos para a recolha da informação necessária assim como a identificação da máquina para efeitos de registo dos dados posteriormente numa base de dados, localizada do lado do servidor. Os campos PROCESS_DIR e CONFIG_DIR identificam os diretórios onde se encontram os ficheiros relativos aos processos e às configurações de programas. Nestes últimos campos, é necessário perceber que, uma vez que as máquinas de pré programação recorrem ao SO *Microsoft Windows*, os caminhos para os diretórios são divididos recorrendo ao '\'. Os últimos campos SVR_ADDRESS, SVR_PORT e SVR_SERVICE apresentam as configurações necessárias para o *Data Collection* conseguir efetuar ligações ao servidor e, dessa forma, enviar os dados recolhidos.

Feitas as configurações necessárias, a instalação deste programa é realizada recorrendo à ferramenta *schtasks* dos SO *Microsoft Windows* [29]. Esta ferramenta permite criar tarefas automáticas e cíclicas. Assim, criam-se duas tarefas para a execução do programa *Data Collection*: uma para recolher dados dos processos das máquinas e outra para recolher as configurações dos programas. O excerto de código seguinte permite criar as tarefas mencionadas sendo que a primeira é executada a cada hora certa e a segunda executada aos 30 minutos de cada hora.

```
C:\> schtasks /create /sc hourly /st 00:00:00 /ru
"SYSTEM" /tn "DataCollection_process"
```

```
/tr "C:\DataCollection\DataCollection.exe 1"

C:\> schtasks /create /sc hourly /st 00:30:00 /ru
"SYSTEM" /tn "DataCollection_config"
/tr "C:\DataCollection\DataCollection.exe 2"
```

Como é possível reparar, as tarefas executam o programa *DataCollection.exe* com os argumentos 1 ou 2. Com isto executa-se o programa sem a verbosidade. O argumento "*SYSTEM*" permite que seja o utilizador relativo ao SO fazendo com que a tarefa seja executada em *background*. Desta forma, sempre que a tarefa é executada, a janela da linha de comandos não é aberta, não permitindo que um operador das máquinas de pré programação a feche e interrompa o processo de recolha de dados. Os restantes argumentos são apresentados na Tabela 4.

Tabela 4 Argumentos para configuração de tarefas.

Argumento	**Funcionalidade**
/*create*	Criar tarefa.
/sc	Frequência da execução da tarefa.
/st	Tempo de início de uma tarefa.
/ru	Utilizador sob o qual a tarefa corre.
/tn	Nome da tarefa.
/tr	Caminho para o ficheiro a executar pela tarefa.

4.3. RECOLHA DE DADOS

A recolha de dados do *Data Collection* é o coração do programa. Dependendo se se pretende recolher dados de processos ou dados das configurações de programas, o algoritmo desenvolvido difere. No caso da recolha das configurações dos programas apenas é necessário percorrer os ficheiros relativos a cada programa de forma a fazer a extração da

informação. No entanto, no caso da recolha dos dados de processos, o algoritmo é mais complexo, pelo que será explicado em seguida.

4.3.1. RECOLHA DOS DADOS DE PROCESSOS

Os dados relativos aos diferentes processos das máquinas encontram-se divididos por diferentes ficheiros de texto que se encontram num determinado diretório da máquina de pré programação. Cada ficheiro diz respeito a um dos programas instalados nas máquinas que posteriormente serão inseridos nos IC. Assim, cada ficheiro contém os diferentes dados dos processos relativos a um determinado programa. O algoritmo que será descrito a seguir pode ser resumido pela Figura 15.

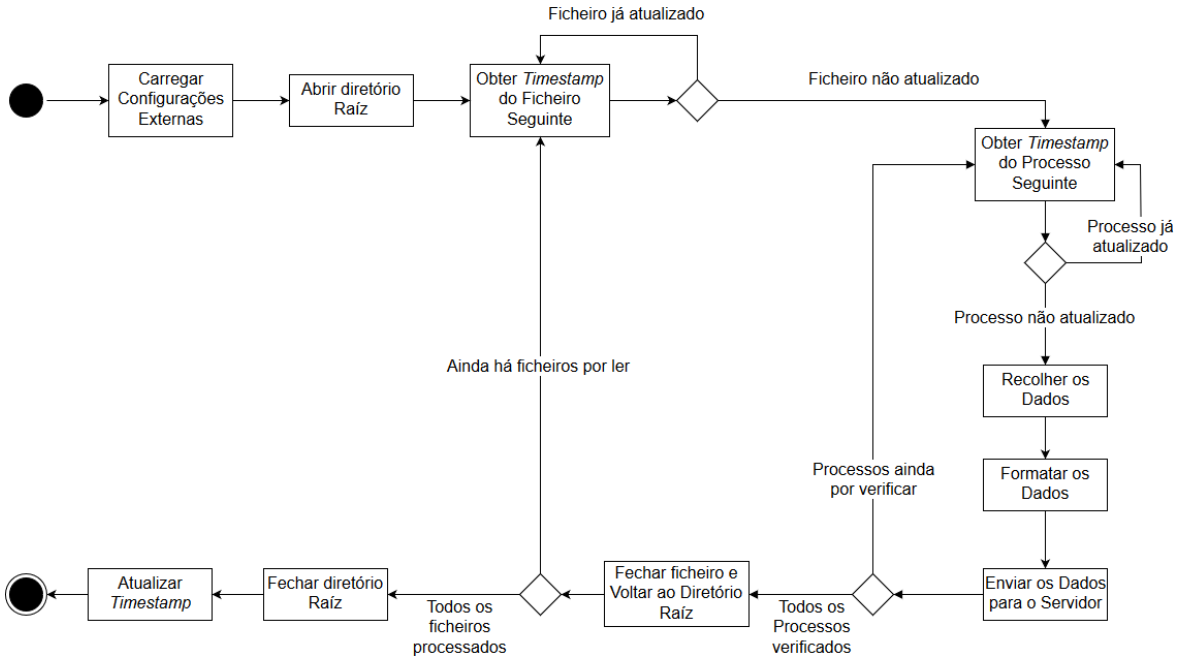


Figura 15 Diagrama de atividade da recolha de dados de processos

Após carregadas as configurações externas, o *Data Collection* começa o processo de recolha dos dados. Nesta altura, o primeiro passo é abrir o diretório raíz onde se encontram os ficheiros com os dados dos processos. De seguida, o programa percorre o diretório, ficheiro a ficheiro, recolhendo o *timestamp* da última vez que o ficheiro foi modificado. É feita uma comparação deste *timestamp* com o que está guardado no ficheiro externo ao programa que contém a data do último processo enviado para o servidor. Caso o *timestamp* do ficheiro seja mais recente do que o guardado, o *Data Collection* abre o ficheiro do programa para ler os registos dos processos no seu interior.

Como já foi referido, cada ficheiro contém vários processos registados no seu interior. Isto faz com que haja necessidade de verificar a data em que os diferentes processos registados no ficheiro também são mais recentes do que o *timestamp* guardado no ficheiro externo. Esta verificação permite otimizar *Data Collection* fazendo com que este não leia ficheiros e até mesmo os processos no seu interior que são antigos e que, por isso, já foram enviados para o servidor. Esta verificação permite ainda evitar que seja enviada informação repetida para o servidor.

Quando o programa termina de recolher todos os dados necessários dos processos relativos a um determinado programa, formata a informação e envia os dados para o servidor. Por fim fecha o ficheiro e volta ao diretório, passando para o ficheiro seguinte.

4.4. FORMATAÇÃO DOS DADOS

Como foi especificado no capítulo 3, definiu-se que as trocas de dados a efetuar entre os diversos componentes do sistema devem ser no formato de JSON, de forma a estabelecer um *standard* entre todas as comunicações. No entanto, este tipo de formato de dados não é nativo da linguagem C e, por isso, recorreu-se à biblioteca *Parson*, desenvolvida para formatar dados em JSON nesta linguagem [28]. Uma vez que o *Data Collection* recolhe dois tipos de dados distintos, é necessário preparar e organizar os dados de formas diferentes para posteriormente serem enviados para o servidor.

No caso das configurações dos programas, apenas é necessário criar um objeto de JSON geral com as configurações e que identifique o programa a que as configurações pertencem e a máquina em que estão.

No caso da recolha de dados dos processos, e como já foi referido, pretende-se recolher informações relativas aos processos no geral, mas também recolher dados com mais detalhe nomeadamente dados de processos relativos aos programadores e aos *sockets*. Desta forma criou-se um objeto de JSON dividido em três partes: um objeto com a informação geral do processo, um vetor composto por vários objetos que contêm os dados dos vários programadores e, por fim, outro vetor composto por vários objetos que contêm os dados dos vários *sockets*. Como já foi explicado no início deste capítulo, existem vários programadores que dependem de uma máquina e os vários *sockets* dependem dos vários programadores.

Desta forma em todas as estruturas criaram-se campos de identificação para fazer essa ligação e tornar o trabalho do servidor mais fácil quando precisar de interpretar os dados. O excerto de código apresentado a seguir representa a estrutura de dados explicada anteriormente.

```
{
  "processo":{
        //...
        "ic_entrada": 100,
        "ic_rejeitados": 2,
        "ic_saida": 98
  },
  "programadores":[
        {
              "id": 1,
              //...
              "ic_entrada": 10,
              "ic_rejeitados": 0,
              "ic_saida": 10
        },
        //...
  ],
  "sockets":[
        {
              "id": 1,
              "programador_id": 1,
              //...
              "ic_entrada": 2,
              "ic_rejeitados": 0,
              "ic_saida": 2
        },
        //...
  ]
}
```

4.5. COMUNICAÇÃO COM O SERVIDOR

A comunicação é a chave de todo o sistema. É o que permite partilhar a informação entre todos os componentes. A decisão de culminar o desenvolvimento do sistema numa aplicação *Web* de forma que os utilizadores sejam capazes de aceder aos dados, leva à pressuposição da existência de um sistema apoiado numa rede *Internet Protocol* (IP). Sabe-se que, após ser desenvolvido este software, o *Data Collection*, será desenvolvida uma aplicação que será responsável por receber os dados e armazená-los numa base de dados.

A linguagem C permite efetuar comunicações através de *sockets*. Uma vez que o *Data Collection* será executado em SO *Microsoft Windows* recorre-se à biblioteca *Winsock2* [30] para a utilização de *sockets*. Apesar de as funções e os seus protótipos apresentarem

algumas diferenças para a biblioteca utilizada em sistemas UNIX por exemplo, a lógica do funcionamento dos *sockets* não difere. Por definição, um *socket* é um ponto de término de uma comunicação de dois sentidos entre dois programas numa rede [31]. A comunicação através de *sockets* desenvolvida apresenta uma sequência lógica entre o cliente e o servidor, como é apresentada no excerto de código seguinte.

```
unsigned char connectToServer(/* … */){

    char option[32];
    option[0] = '\0';
    //...

    SOCKET s;
    struct sockaddr_in server;

    if((s = socket(AF_INET, SOCK_STREAM, 0 )) == INVALID_SOCKET){
        fprintf(stderr,
                "ERROR:\tCould not create socket %d\n",
                WSAGetLastError());
        return 0;
    }

    server.sin_addr.s_addr = inet_addr(SVR_ADDRESS);
    server.sin_family = AF_INET;
    server.sin_port = htons(SVR_PORT);

    if (connect(s,
           (struct sockaddr *)&server,
           sizeof(server)) != 0){

        fprintf(stderr, "ERROR:\tConnecting to server\n");
        closesocket(s);
        return 0;
    }

    char* serialized_string;
    serialized_string = prepareData(/* … */);

    char *message;
    if((message = (char *) calloc(2*strlen(serialized_string),
     sizeof(char))) == NULL)
           perror("ERROR");

    char *data;
    if((data = (char *) calloc(strlen(serialized_string)+1,
     sizeof(char))) == NULL)
           perror("ERROR");

    char host[64];
    host[0] = '\0';

    sprintf(data, serialized_string);
    sprintf(host, SVR_ADDRESS, ":", SVR_PORT);
```

```
    sprintf(message, "\
POST %s HTTP/1.1\r\n\
Host: %s\r\n\
Content-Type: application/json; charset=utf-8;\r\n\
Content-Length: %d\r\n\r\n\
%s\r\n", option, host, strlen(data), data);

    if( send(s, message, strlen(message), 0) < 0){
        fprintf(stderr,
           "ERROR:\tFailed sending message to server\n");

        json_free_serialized_string(serialized_string);
        free(data);
        free(message);
        closesocket(s);
        return 0;
    }
    json_free_serialized_string(serialized_string);


    char server_reply[1024];
    int recv_size;

    if((recv_size = recv(s, server_reply, 1024, 0))
      == SOCKET_ERROR){

        fprintf(stderr,
           "ERROR:\tFailed receiving message from the server\n");
        free(data);
        free(message);
        closesocket(s);
        return 0;
    }

    server_reply[recv_size] = '\0';

    free(data);
    free(message);
    closesocket(s);
    return 1;
}
```

Como é possível de ver no excerto anterior, antes de se efetuar a troca dos dados entre o cliente (*Data Collection*) e o servidor, é estabelecida uma conexão. Isto é feito graças à utilização de *Transmission Control Protocol* (TCP). A escolha do TCP para a comunicação é baseada na necessidade da fiabilidade. O TCP ao contrário do *User Datagram Protocol* (UDP), é um protocolo orientado à conexão e, por isso, é composto por uma série de mecanismos que garantem a entrega dos dados [4].

Para o *Data Collection* apenas se necessitou de criar o lado do cliente da comunicação, uma vez que este é responsável por enviar os dados recolhidos para o servidor. Para isso, inicializou-se a estrutura que permite fazer as configurações dos *sockets* para ser possível depois fazer a comunicação. Inicializada a estrutura, é preciso configurar os campos através da função *socket()*. Assim, criou-se um *socket* com as configurações AF_INET e SOCK_STREAM. A primeira configuração (AF_INET), indica que se pretende utilizar a família de endereços IPv4 uma vez que é nesta família de endereços que toda a rede interna da Bosch se encontra configurada. A segunda configuração (SOCK_STREAM) indica que se pretende utilizar o protocolo TCP para o transporte de dados [30].

De seguida configurou-se uma estrutura responsável por armazenar as configurações do servidor, nomeadamente o endereço e a porta onde este é possível de ser encontrado assim como a família de endereços que utiliza. O endereço e a porta do servidor são parâmetros carregados nas configurações iniciais do *Data Collection* e a família de endereços foi definida para endereços IPv4.

Aceite o pedido de conexão ao servidor, cria-se uma mensagem para enviar os dados. A mensagem é criada com base em HTTP, utilizando POST. Apesar de na linguagem C não haver uma forma nativa de criar mensagens HTTP, este tipo de mensagens não é nada mais do que texto de tal forma organizado que quando é recebido é possível de ser interpretado e identificado como uma mensagem HTTP. Desta forma, cria-se uma *string* com a estrutura de uma mensagem HTTP, como a que é apresentada no excerto de código seguinte.

```
POST %s HTTP/1.1\r\n
Host: %s\r\n
Content-Type: application/json\r\n
Content-Length: %d\r\n\r\n
%s\r\n
```

No excerto anterior é possível identificar os dois principais elementos das mensagens HTTP: o cabeçalho e o corpo. O cabeçalho da mensagem começa no início do excerto e termina na linha que inclui "*Content-Length*", linha inclusive. O corpo é representado pela última linha do código apresentado. O texto apresentado a negrito identifica as zonas onde serão inseridas informações necessárias ao pedido. Um ponto importante é a presença de "\r\n" em todas as linhas que significa a mudança de linha. Isto deve-se à estrutura das mensagens HTTP. Informações distintas devem apresentar-se em linhas distintas. O cabeçalho da mensagem e o corpo encontram-se separados por uma linha em branco e, por

isso, a penúltima linha do excerto encontra-se com uma duplicação dos caracteres de término de linha.

Na primeira linha identifica-se uma mensagem do tipo POST que utiliza a versão 1.1 do HTTP e onde será inserido na zona a negrito o caminho para o recurso do servidor para onde se destina a mensagem. Na segunda linha, indica-se o endereço e porta do servidor. Na terceira linha define-se o tipo de dados que se pretende enviar. Como foi referido, os dados serão formatados em JSON e, dessa forma, é necessário que a mensagem HTTP vá identificada como transportadora desse tipo de dados. Na quarta linha é inserido o tamanho dos dados que se enviarão no corpo da mensagem. Por fim, na última linha é inserida a estrutura JSON anteriormente formatada pelo *Data Collection.*

Preparada a mensagem, é possível enviá-la para o servidor através da função *send()*. Após a receção e o processamento dos dados do lado do servidor, este envia uma mensagem HTTP com a resposta. Recebida a resposta o *Data Collection* fecha o *socket* e prossegue o seu normal funcionamento.

4.6. SÍNTESE

Em síntese, o *Data Collection* é o elemento do sistema responsável por fazer a recolha de dados das máquinas de pré programação. Foi desenvolvido de forma a ser executado de 4 modos distintos, sendo que 2 deles são para ser executados num formato de *debug* permitindo perceber o que acontece no programa à medida que ele corre.

Como foi especificado, havia uma necessidade de se configurar o software através de um ficheiro externo. Dessa forma, utilizou-se um ficheiro para manter as configurações necessárias para o correto funcionamento do programa. Estas configurações são carregadas antes de ser despoletada a lógica de todo o programa.

Para fazer a instalação do programa é necessário mover para a máquina a pasta que contém o executável e os restantes ficheiros do qual este depende e fazer as devidas configurações através do ficheiro construído para o efeito, presente na pasta. Depois basta executar o programa a partir de uma linha de comandos passando em argumento o modo de execução. Para o caso da recolha automática dos dados das máquinas de pré programação,

foram criadas tarefas de forma que o programa seja executado de hora a hora, intercalando a cada meia hora a recolha de dados dos processos e das configurações dos programas.

O JSON é a tecnologia utilizada para formatar os dados para estes depois serem enviados para o servidor. Uma vez que este formato de dados não é nativo da linguagem C, recorreu-se a uma biblioteca externa para o efeito. Devido à facilidade de utilização, de leitura e interpretação, decidiu-se utilizar esse formato também para os ficheiros de configuração do *Data Collection*.

A comunicação é o coração de todo o sistema desenvolvido e, sabendo que os restantes elementos do sistema deveriam ser desenvolvidos sobre redes IP, recorreu-se à utilização de *sockets* para o envio dos dados para o servidor. A utilização dos *sockets* configurados para utilizar TCP de forma a transportar os dados, permitiu garantir a entrega dos dados e a entrega de forma ordenada, graças à sua orientação à conexão e aos seus mecanismos de controlo de fluxo. As mensagens transportadas por este protocolo são mensagens HTTP, configuradas para enviarem os dados através de pedidos POST, encapsulando os dados no corpo da mensagem.

O capítulo seguinte, apresenta os desenvolvimentos efetuados na base de dados do sistema, de forma a estar preparada para armazenar os dados quando o servidor começar a recebê-los.

5. BASE DE DADOS

Este capítulo abordará uma série de desenvolvimentos efetuados do lado do servidor, nomeadamente os aqueles efetuados ao nível das bases de dados. Os desenvolvimentos apresentados fazem parte de um processo de *back end* que permite suportar o funcionamento da aplicação *Web* e, principalmente, permite criar uma estrutura centralizada para guardar a informação. Como já foi mencionado, por questões de confidencialidade as tabelas apresentadas neste projeto não se encontram completas e estão representadas por nomes fictícios, não correspondendo aos nomes utilizados na BD desenvolvida.

5.1. ARQUITETURA

A arquitetura de uma base de dados é um ponto relevante para que todo o sistema funcione corretamente e da forma mais eficiente possível. Assim, com base na especificação feita, implementou-se a arquitetura apresentada na Figura 16 para a base de dados que armazenará os dados do sistema.

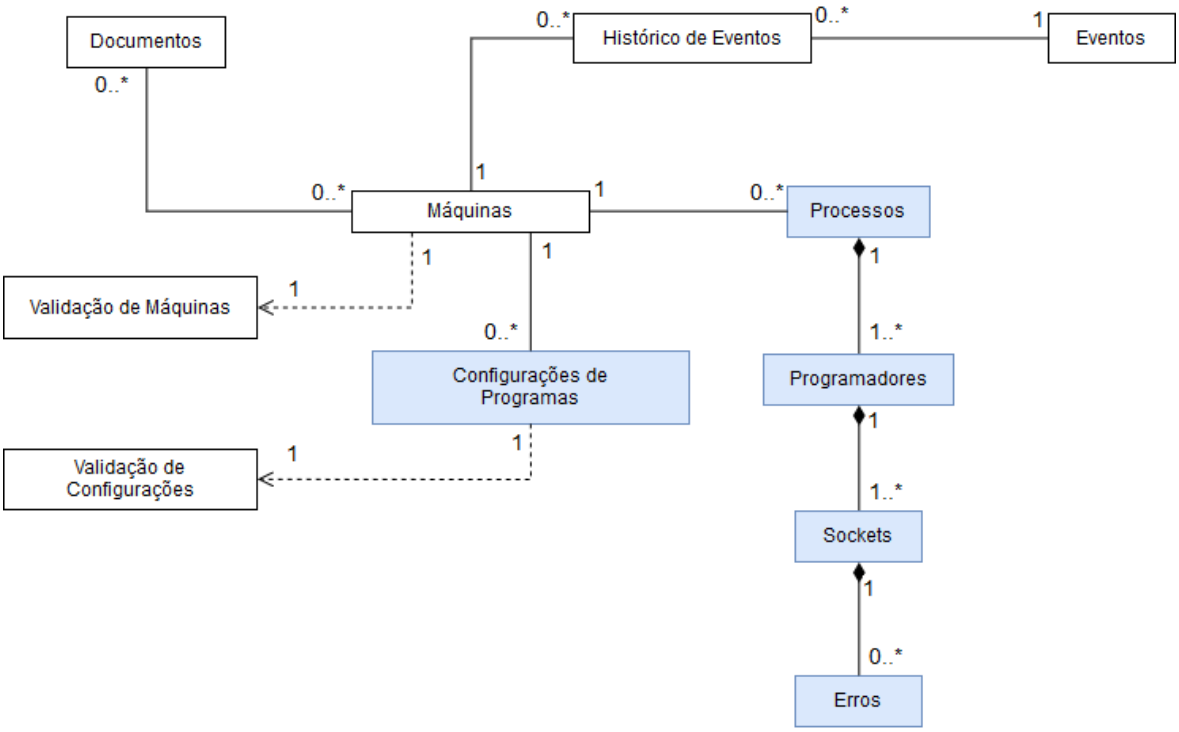


Figura 16 Arquitetura da base de dados.

Como é possível ver a partir da Figura 16, cinco entidades estão apresentadas com a cor azul. Estas entidades são aquelas que são responsáveis por armazenar os dados que são recolhidos pelo *Data Collection*. Assim, para cada máquina é possível encontrar na BD zero ou mais dados de processos e de configurações de programas.

No caso dos processos é possível perceber que a arquitetura é apresentada em escada, ou seja, só podem existir dados na entidade de Erros se existirem dados na entidade *Sockets*, existir dados na entidade *Sockets* se houver dados registados na entidade Programadores e dados destes se houver informação registada na entidade Processos. Para cada processo que é guardado, são associados na entidade Programadores os detalhes dos programadores, associados na entidade *Sockets* os detalhes dos *Sockets* e na entidade dos Erros os detalhes dos erros caso estes últimos existam.

No caso da entidade Configurações de Programas, os dados das configurações provenientes do *Data Collection* são diretamente armazenados nessa entidade. No entanto, como é possível verificar na Figura 16, essa entidade apresenta uma dependência da entidade Validação de Configurações.

As restantes entidades, representadas com o fundo branco, são responsáveis por armazenar dados que são manipulados através da aplicação desenvolvida. À semelhança da relação entre as entidades Configurações de Programas e Validação de Configurações, as entidades Máquinas e Validação de Máquinas também se encontram ligadas com uma dependência de um para um. A entidade Documentos apresenta uma relação de muitos para muitos com a entidade Máquinas uma vez que se podem associar vários documentos a diferentes máquinas. No que diz respeito há criação de um sistema de gestão de eventos, decidiu-se criar a entidade Eventos de forma a criar uma lista *standard* de tipos de eventos e uma entidade Histórico de Eventos para registar as ocorrências associando sempre a um tipo de evento. Assim, cada máquina pode ter registado uma série de ocorrências de um evento ao longo do tempo, como por exemplo a atualização do seu sistema.

5.2. TABELAS

Apresentada a arquitetura e as relações entre todas as tabelas da BD, é necessário analisar os campos e as ligações entre as tabelas. Todas as tabelas têm na sua estrutura configurada uma chave primária (*Primary Key* – PK) designada por ID, que poderá permitir mais tarde utilizá-la para associar a outras tabelas.

5.2.1. MÁQUINAS E VALIDAÇÃO DE MÁQUINAS

A tabela Máquinas é a entidade central de toda a base de dados uma vez que sem o registo das diferentes máquinas não haveria a possibilidade de se associarem os dados a cada uma, não havendo qualquer sentido na estrutura da base de dados. Para além da coluna ID esta tabela apresenta uma série de colunas para registar a informação de cada máquina e uma coluna específica para a marcar como validada ou não.

A tabela Validação de Máquinas é composta por uma coluna MAQUINA_ID configurada como chave estrangeira (*Foreign Key* – FK) ligada à chave primária da tabela Máquinas, por uma série de colunas booleanas e no final por uma coluna designada por PROGRESSO onde será guardada a percentagem de campos que já estão validados, ou seja, a quantidade de colunas cujo valor é *true*. Quando a percentagem de validação for de 100 % o campo da tabela Máquinas que permite marcar uma máquina como validada ou não deve ser também alterado para *true*.

5.2.2. CONFIGURAÇÕES DE PROGRAMAS E VALIDAÇÃO DE PROGRAMAS

A tabela de configurações de programas armazena os dados provenientes do *Data Collection*. Esta é composta por uma chave estrangeira que permite fazer a ligação das diversas configurações à máquina de onde elas vêm, de uma série de campos para guardar as configurações e um campo para marcar a configuração como validadas ou não.

A tabela de Validação de Configurações é composta por uma coluna CONFIG_ID configurada como chave estrangeira ligada à chave primária da tabela Configurações de Programas, por uma série de colunas booleanas e no final por uma coluna PROGRESSO onde será guardada a percentagem de campos que já estão validados. À semelhança das máquinas, quando a percentagem de validação for de 100 % o campo da tabela Configurações de Programas que permite marcar uma máquina como validada ou não, dever ser alterado para *true*.

5.2.3. PROCESSOS, PROGRAMADORES, *SOCKETS* E ERROS

Com o envio dos dados dos processos não só relativos ao processo geral da máquina, mas também do detalhe dos IC processados pelos programadores e ainda pelos *sockets* das máquinas, é possível perceber que há uma necessidade de criar mais duas tabelas adicionais: uma para armazenar os dados dos processos relativos aos programadores e outra para os *sockets*. Ainda associados aos processos existem os erros de programação dos IC. Os erros são associados aos *sockets*, pelo que se criou também uma tabela para armazenar os diversos tipos de erros.

As tabelas Processos, Programadores e *Sockets* são muito semelhantes, apresentando todas os campos relativos às contagens dos ICs inseridos, programados e rejeitados. A diferença está nas colunas configuradas como chave secundária e que faz a ligação à coluna da tabela da qual ela depende. Ou seja, a tabela Programadores tem a chave estrangeira PROCESSO_ID e a tabela *Sockets* tem a chave estrangeira PROGRAMADOR_ID. É de relembrar que é necessário ligar cada processo a uma máquina e, por isso, a tabela Processos tem uma chave estrangeira para ligar à tabela Máquinas.

A tabela Erros é composta por uma chave estrangeira que faz a ligação à tabela *Sockets* e por uma série de campos para armazenar a quantidade de erros ocorridos em cada *socket*.

5.2.4. EVENTOS E HISTÓRICO

Para o subsistema Gestão de Eventos criaram-se duas tabelas. Uma que contém os diversos tipos de eventos existentes como a instalação de um novo programa, introdução de novo *hardware* nas máquinas, etc. A segunda tabela servirá de histórico com o objetivo de manter um registo do que é feito nas máquinas.

Estas duas tabelas em conjunto permitem proporcionar aos utilizadores uma forma de registar acontecimentos relevantes relacionados com as máquinas de pré programação. A tabela Histórico de eventos é composta por uma chave estrangeira que permite fazer a ligação à tabela Máquinas de forma a associar a qual das máquinas se refere o acontecimento. Tem ainda outra chave estrangeira ligada à tabela Eventos de forma a identificar o tipo de acontecimento que se pretende registar.

A tabela Eventos apenas é composta pela sua coluna ID que para além de servir de chave primária serve de código identificativo do tipo de evento e uma coluna que serve para descrever o evento em si.

5.2.5. DOCUMENTOS

No caso do subsistema Gestão de Documentos, criou-se uma tabela que permite manter hiperligações para os diversos ficheiros associados às diferentes máquinas. Desta forma, é possível associar manuais e procedimentos às máquinas.

Esta tabela é a única que apresenta uma relação de muitos para muitos com outra. Ou seja, a arquitetura desta relação implica a criação de uma tabela intermédia para se relacionarem as diferentes máquinas com os diferentes documentos existentes. A tabela Documentos é composta pela sua chave primária e uma série de campos que permitem identificar os documentos.

A tabela intermédia Documentos_Máquinas apenas tem três colunas: a primeira com a sua própria chave primária ID, a segunda com a chave estrangeira MAQUINA_ID para fazer a ligação à tabela Máquinas e a terceira com a chave estrangeira DOCUMENTO_ID para fazer a ligação à tabela Documentos. Isto permite associar N máquinas e N documentos.

5.3. SÍNTESE

A arquitetura da base de dados deve ser bem organizada de forma não só a melhorar a sua consulta, mas também de forma a proporcionar escalabilidade para o caso de haver necessidade de armazenar informações que não estão presentes na arquitetura definida. A utilização de chaves primárias e estrangeiras permite relacionar as diversas tabelas de diversas formas, sendo que neste caso, foram estabelecidas relações de um para um, um para muitos e muitos para muitos. A utilização de relações de muitos para muitos leva à necessidade de recorrer a tabelas intermédias para fazer a ligação entre os diversos registos de cada tabela. De forma a resumir a arquitetura da base de dados e as respetivas relações com a associação das chaves primárias e estrangeiras apresenta-se a Figura 17.

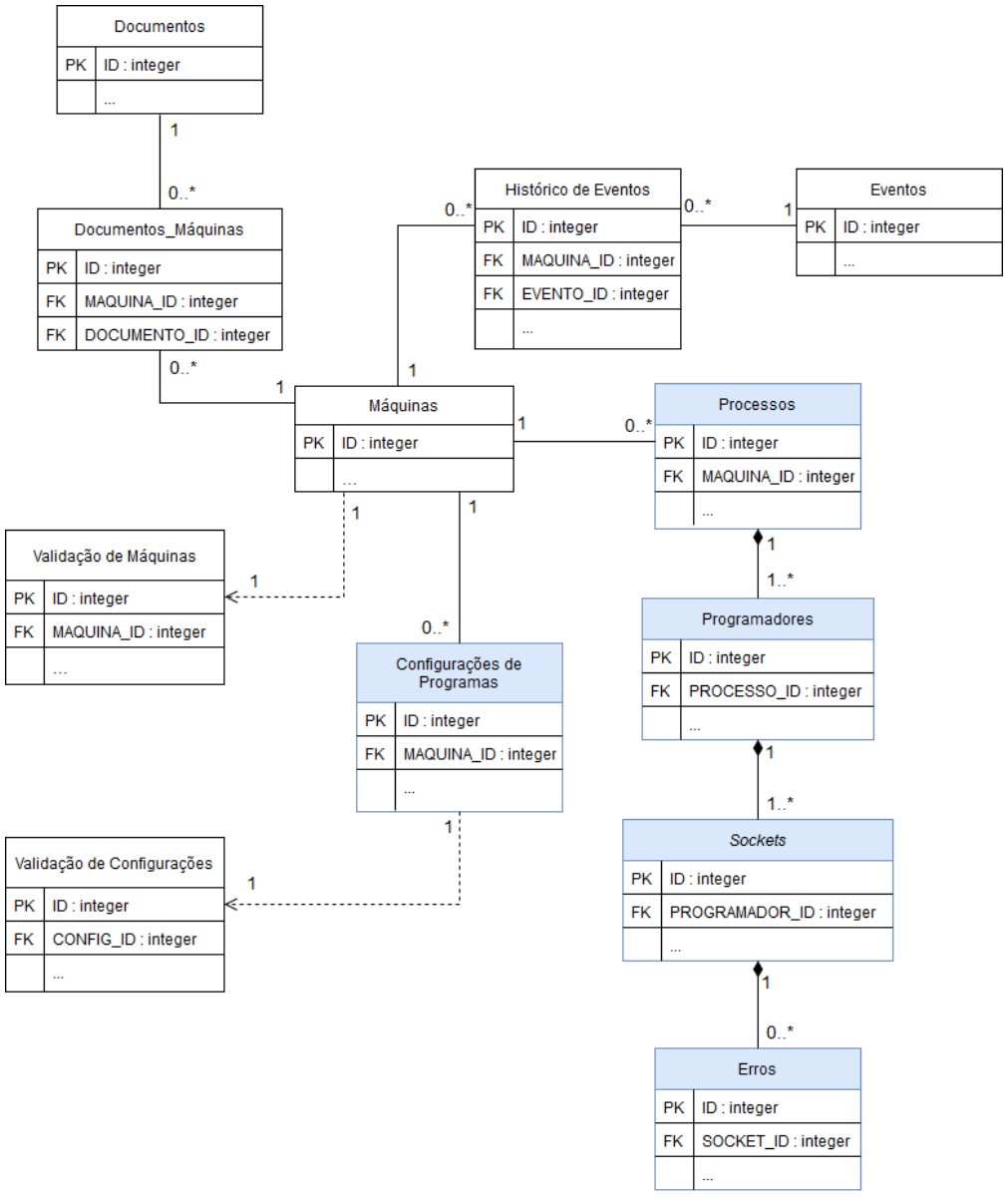


Figura 17 Relações entre as tabelas da base de dados.

6. SERVIÇOS E APLICAÇÃO *WEB*

Neste capítulo são apresentados os serviços e a aplicação *Web* desenvolvidos. Recorrendo à *framework Laravel*, criaram-se os serviços necessários não só para receber os dados do *Data Collection*, mas também os serviços que permitem a manipulação dos dados das diversas tabelas criadas na BD.

Os serviços *Web* são aplicações de cliente e servidor que comunicam sob o *World Wide Web's* (WWW) *HyperText Transfer Protocol* (HTTP). Os serviços *Web* são reconhecidos pela sua grande interoperabilidade e extensibilidade, isto é, programas que executam serviços simples podem interagir uns com os outros de modo a oferecer serviços de valor agregado sofisticado [32]. No âmbito deste projeto, foram desenvolvidos nove serviços *Web*: oito para realizar operações CRUD nas tabelas da BD e um para receber os dados do *Data Collection*.

A aplicação é o último desenvolvimento do sistema desenvolvido e tem como objetivo fornecer uma interface gráfica aos utilizadores que pretendem aceder às informações das máquinas de pré programação. A aplicação apresentada neste capítulo não é a versão final desenvolvida para a Bosch, uma vez que foi necessário preparar uma versão

para a elaboração desta Tese. Assim, a aplicação desenvolvida para a empresa apresenta uma série de funcionalidades consideradas como novas e outras como melhorias.

A aplicação desenvolvida foi dividida em duas grandes partes. Uma das partes diz respeito à organização dos dados por máquina e a outra diz respeito à organização dos dados por IC. Em seguida serão explicadas as funcionalidades de cada página, de modo perceber-se melhor a organização da aplicação.

A utilização de uma *framework* no desenvolvimento desta componente do sistema permite ao programador preocupar-se mais com o seu código e menos com a organização e estruturação do mesmo. Desta forma, recorrendo à *framework Laravel* é possível criar a partir de uma linha de comandos componentes como *controllers*, *models* e *middleware*. Todos estes componentes referidos não passam de classes de PHP que automaticamente se encontram identificadas e que funcionam de acordo com o seu papel na arquitetura MVC.

6.1. OPERAÇÕES CRUD

Como é apresentado no capítulo 3 das especificações do sistema, os serviços foram desenvolvidos sobre uma arquitetura REST [11]. Assim, estes serviços podem ser invocados através da utilização dos diferentes métodos das mensagens HTTP sendo que, para cada combinação método e URL são executadas diferentes operações CRUD [12].

Na Tabela 5 é apresentado o exemplo com base no serviço que permite manipular as máquinas. Quando é feito um pedido HTTP que contém no seu URI o termo “api”, a *framework* automaticamente interpreta como sendo um serviço *Web*, cabendo ao componente *Route* da *framework* reencaminhar o pedido para a classe de PHP responsável por interpretar os pedidos relativos às máquinas. Automaticamente, a *framework* interpreta o método HTTP que se encontra na mensagem e reencaminha o pedido para o método respetivo. É possível perceber na tabela que, os métodos implementados permitem cobrir todas as operações da técnica CRUD.

Tabela 5 Exemplo de acesso a serviço CRUD.

Método HTTP	**URI**	**Classe.Método**	**CRUD**
GET	api/maquinas	Maquinas.*index*	*Read*
POST	api/maquinas	Maquinas.*store*	*Create*
DELETE	api/maquinas/{maquina}	Maquinas.*destroy*	*Delete*
PUT	api/maquinas/{maquina}	Maquinas.*update*	*Update*
GET	api/maquinas/{maquina}	Maquinas.*show*	*Read*

As combinações permitem executar dois tipos de leituras de dados, recorrendo à operação *Read*. No primeiro caso, a execução do método *index*, permite ler da base de dados todos os registos de uma determinada tabela, no caso deste exemplo, permite obter todos os registos de máquinas registadas na BD. No segundo caso, a execução do método *show*, permite ler da base de dados um único registo, uma vez que este método deve receber como argumento o ID do registo da BD que se pretende ler.

No caso das operações *Create* e *Update*, são enviados os dados necessários para se inserir e atualizar a BD, respetivamente. Estes dois métodos retornam o ID do registo que foi criado ou atualizado. A operação *Create* sendo executada a partir de um pedido HTTP POST, os dados são inseridos no corpo do pedido. No entanto, no caso da operação *Update*, tratando-se de um pedido HTTP *PUT*, para além dos dados para atualizar serem apresentados no corpo da mensagem, é necessário incluir no URI o ID do registo que se pretende atualizar. O exemplo apresentado a seguir, representa o *controller* do serviço CRUD para as máquinas.

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\MaquinasModel;

class MaquinasCtrl extends Controller{

     public function index(){
          return MaquinasModel::all();
     }
```

```
    public function store(Request $req){
        $maq = new MaquinasModel;

        $maq->exemplo = $req->input('maq.exemplo);

        //…

        $maq->save();

        $val = ValidacaoMaquinasCtrl::clearOrCreate($maq->id);

        return $maq->id;
    }

    public function show($id){
        return MaquinasModel::where([
            ['id', $id]
        ])->get();
    }

    public function update(Request $req, $id){
        return MaquinasModel::where([
            ['id', $id]
        ])->update([
            'exemplo' => $req->input('maq.exemplo');
        ]);
    }

    public function destroy($id){
        return MaquinasModel::where([
            ['id', $id]
        ])->update([
            'ativa' => 0
        ]);
    }
}
```

Os restantes serviços para manipulação das validações, das configurações dos programas, dos processos, programadores, *sockets* e erros, apresentam uma arquitetura igual ao exemplo das máquinas apresentado anteriormente. É de salientar que, uma vez que há uma dependência entre tabelas da BD, criaram-se algumas dessas dependências também entre os serviços *Web*, de forma a acelerar algumas operações. É exemplo desta manipulação o registo de uma nova máquina que, automaticamente, cria um registo de validação associado à mesma.

6.2. RECEÇÃO DE DADOS DO *DATA COLLECTION*

O serviço responsável por receber os dados do *Data Collection* é o mais complexo. Este serviço tem uma estrutura diferente dos anteriormente apresentados e que se baseavam numa estrutura CRUD. O *Data Collection* foi desenvolvido para enviar dados para apenas dois URI: o "/api" e o "/api/*config*". Desta forma, diferencia apenas processos de configurações de programas, não sendo possível com a estrutura desenvolvida receber separadamente dados dos processos, programadores, *sockets* e erros.

No caso das configurações dos programas, é utilizado o serviço *Web* para o efeito e que foi criado com base na técnica CRUD, uma vez que a informação enviada apenas diz respeito à tabela Configurações de Programas e, por isso, pode ser inserida diretamente.

No caso dos dados dos processos, é necessário dividir a estrutura dos dados enviada pelo *Data Collection* antes de fazer a inserção dos mesmos na BD. Assim, criou-se um serviço *Web* para receber toda a estrutura enviada do *Data Colection* dividindo-a em quatro estruturas distintas: Processo, Programadores, *Sockets* e Erros. Feita a divisão dos dados, passa-se à inserção dos mesmo na BD, recorrendo aos serviços respetivos. Uma vez que há uma dependência entre as tabelas, é preciso registar os dados dos processos antes dos restantes de forma a obter-se o ID do processo registado para ser utilizado na associação aos programadores. O mesmo se sucede para as restantes estruturas de dados Programadores, *Sockets* e Erros.

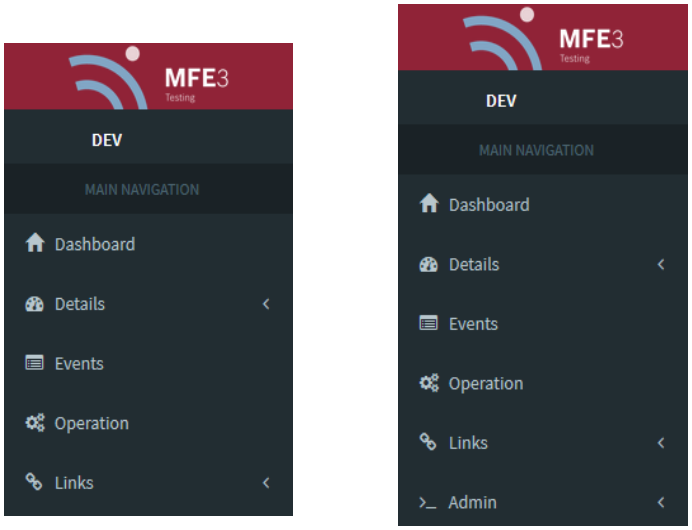
6.3. PRINCIPAIS COMPONENTES DA APLICAÇÃO

6.3.1. CABEÇALHO, MENU E CORPO

Todas as páginas encontram-se divididas em três zonas: o cabeçalho, o menu e o corpo. O cabeçalho apresentado na Figura 19 apresenta apenas dois botões. O botão colocado à esquerda permite abrir ou fechar o menu. Esta funcionalidade tem como principal objetivo aumentar a área do corpo através do fecho do menu, permitindo ter mais espaço para a disponibilização de informação. O botão colocado à direita identifica o utilizador que está a utilizar a aplicação permitindo fazer o *logout* ou redirecionar o convidado para a página de autenticação.

Figura 19 Cabeçalho da aplicação *Web*.

O menu apresentado na Figura 18, permite fazer a navegação pelas diferentes páginas da aplicação. Apresentam-se dois menus diferentes: um para um utilizador com acesso de administração (Figura 18a) e outro sem acesso de administração (Figura 18b). É de salientar que os menus apresentados se encontram abertos.



(a) (b)

Figura 18 Menu da aplicação Web: (a) Menu sem direitos de administração; (b) Menu com direitos de administração.

O corpo representa o que a página terá em si. É onde se encontram as funcionalidades que permitem o utilizador beneficiar da aplicação. Desta forma, esta zona será explicada

mais à frente através da apresentação das páginas *Dashboard*, algumas partes das páginas pertencentes ao grupo *Details* e a página *Operation*, apresentadas no menu da Figura 18. Por questões de confidencialidade há partes da aplicação que não poderão ser ilustradas.

6.3.2. DASHBOARD

O *dashboard* é a página inicial da aplicação e está ilustrado na Figura 20. Nesta página são apresentados gráficos dos três indicadores referidos no capítulo 3: Produção, Taxa de Produção (TP) e Taxa de Rejeição (TR). No entanto, esta página apenas apresenta a informação para as máquinas que estão marcadas como estando em produção, permitindo uma melhor análise aos utilizadores. Por cima dos gráficos encontra-se um conjunto de filtros que permitem alterar os dados apresentados nos gráficos. É possível selecionar uma ou mais máquinas, alterar o intervalo de tempo dos dados e escolher um dos três indicadores que se pretende visualizar a vermelho, azul e verde apresentados na Figura 20, respetivamente.

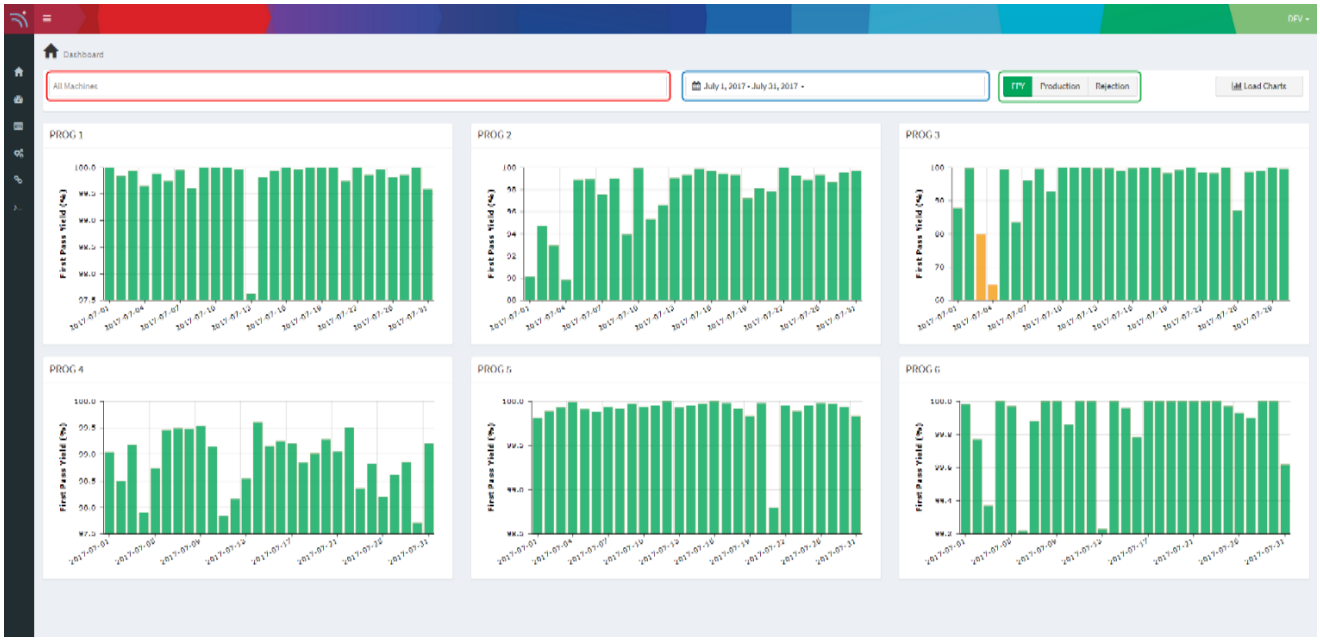


Figura 20 *Dashboard* da aplicação *Web*.

6.3.3. *DETAILS*

Os detalhes são representados por mais do que uma página, sendo feita a divisão em duas categorias: Máquinas e IC. Em ambos os casos a página é dividida em três partes: filtros, gráfico e tabela, como se pode ver na Figura 21.

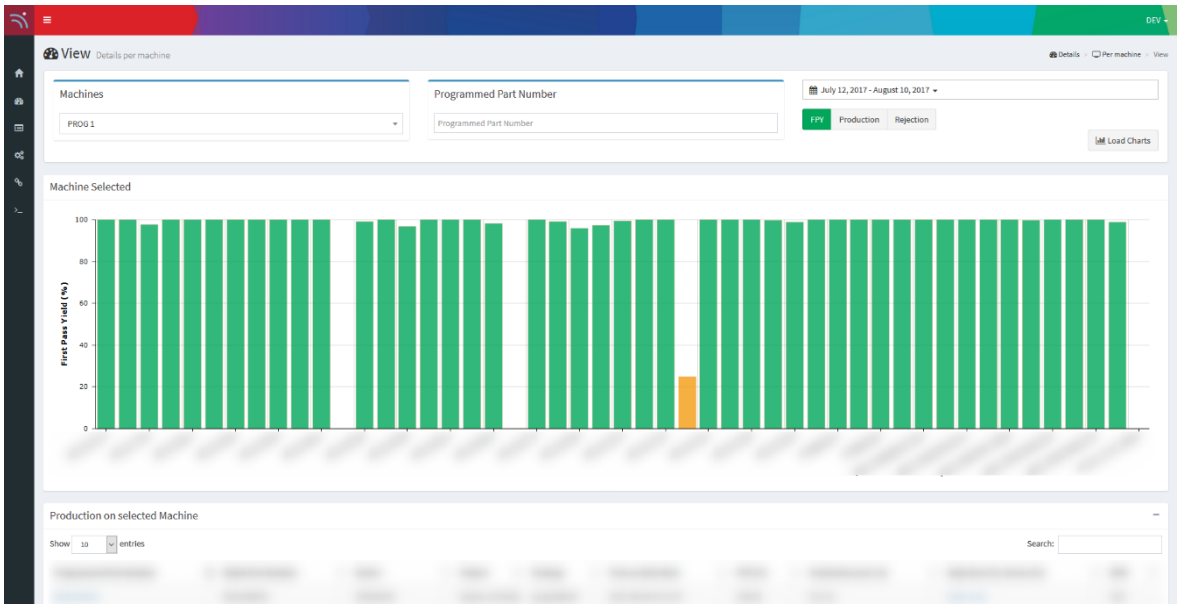


Figura 21 Página de detalhes da aplicação Web.

Nos filtros, no caso das máquinas é possível selecionar a máquina, o programa utilizado para programar um determinado IC, alterar o intervalo de tempo da pesquisa e o indicador que se pretende analisar. No caso dos IC, é possível fazer uma pesquisa por um IC vazio (não programado) ou um IC programado. Caso se selecione um IC vazio, é ainda possível filtrar por um ou mais programas que podem ser utilizados para programar esse IC. Os restantes filtros são iguais às máquinas.

No gráfico, no caso das máquinas apresenta-se o indicador em função dos diversos programas, sendo que cada um acumula os processos para um determinado programa num determinado intervalo de tempo. No caso dos IC, é apresentado um gráfico com o indicador em função das diferentes máquinas, sendo que cada uma acumula os processos para um determinado IC vazio ou IC programado, dependendo da pesquisa, num determinado intervalo de tempo.

Na tabela, são apresentados os dados da produção ilustrados pelos gráficos, mas sobre um formato de tabela, podendo ajudar a interpretação quando há uma grande

quantidade de dados. Nesta tabela é ainda possível aceder a uma outra tabela com informação detalhada dos erros ocorridos por *socket*.

6.3.4. OPERATION

Todas as linhas de produção da Bosch têm um ecrã onde é apresentada informação da produção. No caso das máquinas de pré programação, com o desenvolvimento deste projeto pretende-se utilizar um desses monitores para disponibilizar a aplicação. Dessa forma, esta página apresenta gráficos dos três indicadores pelas diferentes máquinas que vão alterando a cada trinta segundos. A Figura 22 ilustra a página desenvolvida.

Figura 22 Página *Operation* da aplicação *Web*

6.4. PRINCIPAIS FUNCIONALIDADES DA APLICAÇÃO

As principais funcionalidades da aplicação são a visualização de detalhes de Processos, Programadores, *Sockets* e Erros através da interação dinâmica com os gráficos e a utilização de *checklists* para validação de processos da Bosch.

6.4.1. DETALHES DE PROCESSOS, PROGRAMADORES, *SOCKETS* E ERROS

Os detalhes dos processos das máquinas através da divisão em Processos, Programadores e *Sockets*, são algumas das especificações apresentadas no capítulo 3. Isto consiste em tornar todos os gráficos dinâmicos de forma a obter-se essa informação, como apresentado na Figura 23.

Figura 23 Funcionlidade dos Gráficos: (a) Processos; (b)Programadores; (c) *Sockets*

Todos os gráficos apresentam inicialmente dados dos processos, independentemente do indicador ou se a informação no eixo das abcissas é dividida por máquinas ou programas. Quando se clica numa das barras do gráfico, é carregada a informação dos Programadores. Desta forma, a informação no eixo das abcissas passa a ser dividida pelos diversos programadores utilizados nos processos. O mesmo se sucede quando se clica numa das barras que dizem respeito a um determinado programador, passando a apresentar-se a informação dividida por *Sockets*.

No exemplo da Figura 23, recorre-se à página de detalhes das máquinas. Seleciona-se uma máquina e um determinado programa utilizado para programar IC. A programação dos IC levou ao registo de dados de processos e sucessivamente de dados associados aos Programadores e aos *Sockets*. É preciso tentar perceber quais os programadores e *sockets* com pior desempenho que levaram ao aparecimento da barra cuja TP não é igual a 100 %. Desta forma clica-se na barra, resultando o gráfico do Figura 23 (b). é possível perceber que os programadores 1 e 3 têm uma TP entre 20 % a 80 % e, por isso, apresentam-se a amarelo. Para melhor perceber onde falha, clicando no programador 3 obtêm-se os dados da Figura 23 (c), com os dados por *Socket*. Assim, percebe-se que os *sockets* 1, 3 e 4 têm uma TP entre 20 % e 80 % enquanto no *socket* 2 a TP é mesmo 0 %. Logo, o *socket* 2 rejeitou todos os IC que nele foram inseridos sendo preciso analisar os Erros com detalhe na tabela a baixo do gráfico.

6.4.2. *CHECKLISTS* DE VALIDAÇÃO

Foram desenvolvidas uma série de *checklists*, para fazer a validação de diversos processos: instalação de uma nova máquina e instalação de uma nova configuração de programa numa máquina. O objetivo destas *checklist* é tentar evitar futuros eventuais problemas através de um processo de dupla verificação. Após a instalação de uma nova máquina ou de uma nova configuração, o responsável por esse processo deve preencher a *checklist* presente na aplicação de forma a garantir que seguiu todos os passos necessários para o correto funcionamento do que foi instalado.

A *checklist* de validação de uma máquina é um processo confidencial pelo que não é apresentado nesta Tese. No entanto, é a *checklist* que proporciona uma série de verificações que levam à correta instalação de uma máquina após esta ser comprada pela Bosch.

A *checklist* de validação de uma configuração de um programa, permite ao utilizador que faz a instalação do programa, verificar que fez tudo correto. Esta é dividida em duas partes: uma de verificação manual pelo utilizador e outra de verificação automática. Na primeira parte, o utilizador verifica uma série de informação geral do programa, apresentado na Figura 24 como *Task*. Na segunda parte, a aplicação compara os campos *Part Number* com *Programmed* e *Data File* com *Software*. Se estes conjuntos de campos se igualarem, os campos ficam a verde indicando que a informação está correta, podendo-se validar a configuração. Quando uma configuração é validada sem estes campos estarem a verde, é despoletado um e-mail para o utilizador que está a fazer a validação e para a equipa de administração, de forma a avisar do acontecimento, havendo a possibilidade de correção.

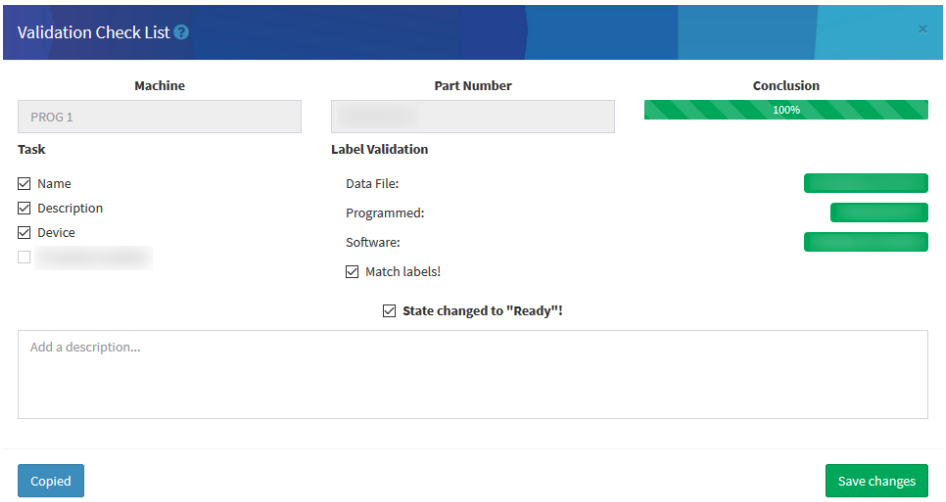


Figura 24 *Checklist* de validação de uma configuração de um programa.

6.5. SÍNTESE

Em síntese, houve a necessidade de se desenvolver uma série de serviços *Web* de forma a aceder aos dados da base de dados. O desenvolvimento destes como uma componente independente das restantes do sistema criado, permite incidir na escalabilidade do mesmo.

Desenvolveram-se nove serviços dos quais oito são responsáveis por realizar operações CRUD com a BD e um tem a responsabilidade de receber os dados proveniente do *Data Collection* e armazená-los na BD. O desenvolvimento dos primeiros oito serviços

permite criar, ler, atualizar e apagar registos das diversas tabelas da BD. O último serviço recebe os dados dos processos recolhidos, faz a sua divisão em estruturas independentes de Processos, Programadores, *Sockets* e Erros e armazena os dados nas respetivas tabelas fazendo a correta associação através das combinações de chaves primárias e chaves estrangeiras. Desta forma, garante-se uma cobertura das necessidades do sistema implementado e, ao mesmo tempo, a possibilidade de se desenvolverem novos serviços de forma independente para novas eventuais funcionalidades do mesmo.

A aplicação está visualmente dividida em três partes: cabeçalho, menu e corpo. O cabeçalho e o menu permitem fazer a navegação pelas diferentes páginas da aplicação enquanto que o corpo apresenta o conteúdo das páginas.

Na página *Dashboard*, são apresentados os diferentes indicadores para as máquinas que se encontram em produção. São apresentados gráficos das máquinas paralelamente permitindo uma análise rápida da produção. Nas páginas *Details*, é possível analisar com detalhe dados dos indicadores TP, TR e Produção por máquina ou por IC, seja este último um IC vazio ou um IC programado. Em todos os casos, é possível aceder a detalhes a nível dos processos, dos programadores e dos *sockets*. Na página de operação é possível ter uma visão geral de cada máquina e de cada indicador, tendo sido esta desenvolvida para ir automaticamente alterando o gráfico.

A utilização de *checklists* para validação de processos é um método que não evita a ocorrência de problemas. Ainda assim, permite baixar a probabilidade da ocorrência dos mesmos através da dupla verificação da informação e à comparação automática de alguns dados. No entanto, a não automatização da verificação de alguns dados, leva a que haja uma maior probabilidade de erro no processo de validação.

7. TESTES

A componente de testes é uma das principais no processo de desenvolvimento de *software*. É nesta altura que se põe à prova o que foi desenvolvido. Para tal existem diversas metodologias e tipos de teste que são aplicados a *software*. De entre os diferentes tipos de testes destacam-se os funcionais, os testes de usabilidade, testes de compatibilidade entre dispositivos, testes de desempenho e testes de segurança. Neste capítulo são efetuados diferentes tipos de teste aos componentes do sistema desenvolvido, nomeadamente, testes de usabilidade, testes de compatibilidade e testes funcionais.

7.1. APLICAÇÃO *WEB*

7.1.1. TESTES DE USABILIDADE

A aplicação *Web* é desenvolvida para servir de interface entre os utilizadores e os dados. Dessa forma, esses intervenientes tornam-se os melhores avaliadores da mesma. Assim, disponibilizou-se a aplicação para ser testada por 6 utilizadores: 2 Operadores, 2 Utilizadores da Secção e 2 Administradores. Ao fim de algum tempo a testar a aplicação, os diferentes utilizadores responderam a um questionário que pretende fazer uma recolha da opinião sobre a sua interação com a aplicação. O questionário desenvolvido tem como base

o *Computer System Usability Questionnaire* (CSUQ) apresentado por *James R. Lewis* no seu relatório técnico "*IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use*" [33]. No Anexo A podem ver-se as questões utilizadas neste questionário.

Analisando os resultados obtidos, é possível perceber a partir da Figura 25 e da Tabela 6 que no geral a aplicação vai de encontro ao que seria esperado. Os Operadores são o grupo de utilizadores que mais valoriza a aplicação uma vez que conseguem aceder a informações necessárias recorrendo a qualquer computador na área da Pré Programação e em qualquer altura. Por outro lado, os utilizadores com acesso de Administração revelaram-se os mais críticos quanto a algumas das funcionalidades da aplicação. No entanto, todos os grupos de utilizadores consideram que a existência da aplicação é uma grande mais valia e que, conforme o principal objetivo do desenvolvimento deste projeto, apoia o trabalho que tem de ser feito nas máquinas de Pré Programação da Bosch.

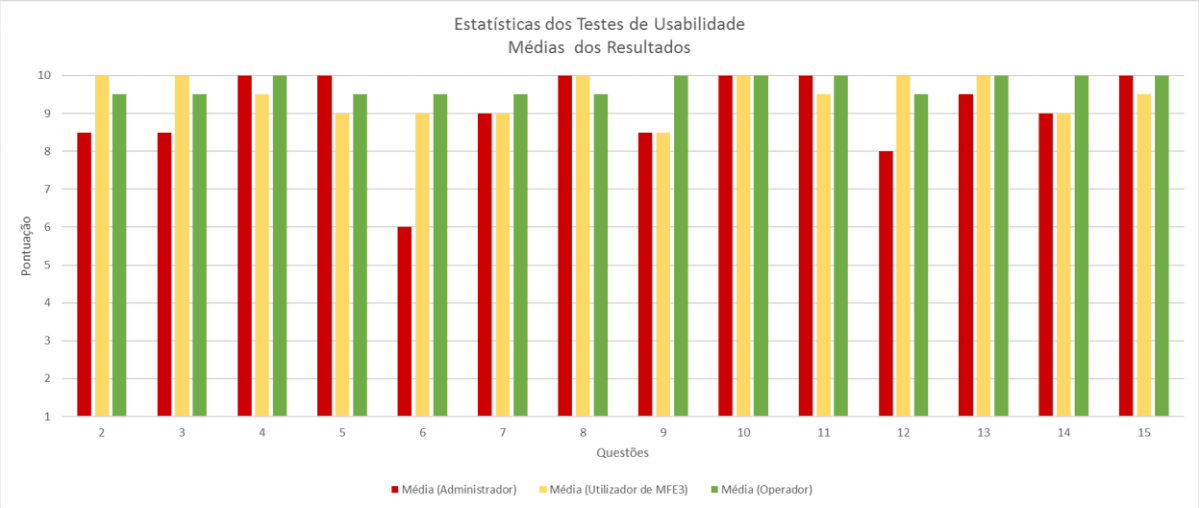


Figura 25 Estatísticas dos Testes de Usabilidade - Médias dos Resultados.

Tabela 6 Médias Finais dos Testes de Usabilidade

	Administrador	**Utilizador de MFE3**	**Operador**
Média Individual	9.07	9.50	9.75
Média Global	**9.44**		

É possível perceber que o Administrador se revela o utilizado mais crítico nas questões mais técnicas como por exemplo nas questões da organização da informação ou na apresentação de mensagens de erro. O Utilizador de MFE apresentam-se com uma opinião menos crítica do que a do Administrador. No entanto, no que diz respeito à facilidade de acesso à informação apresenta-se mais crítico. O Operador é o utilizador que mais valoriza a aplicação, respondendo de forma mais cética quanto à organização da informação.

7.1.2. TESTES DE COMPATIBILIDADE

Os testes de compatibilidade servem para testar se a aplicação é apresentada de forma correta entre diferentes navegadores e diferentes dispositivos. Os testes efetuados e os respetivos resultados são apresentados na Tabela 7. O resultado do teste pode ser PASSA ou FALHA, sendo que a aplicação passa no teste se o seu design e as funcionalidades funcionarem na íntegra. Para falhar no teste basta o design não ser responsivo ou pelo menos uma funcionalidade não trabalhar de acordo com o esperado.

Tabela 7 Resultados dos testes de compatibilidade.

Navegador *Web*	**Dispositivo**	**Resultado**
Firefox	Computador	PASSA
	Tablet	FALHA
	Smartphone	PASSA
Internet Explorer	Computador	PASSA
	Tablet	FALHA

	Smartphone	PASSA
Microsoft Edge	Computador	PASSA
	Tablet	FALHA
	Smartphone	PASSA

7.1.3. TESTES FUNCIONAIS

Os testes funcionais são responsáveis por avaliar as funcionalidades da aplicação. Neste caso, estes foram desenvolvidos para testar a reação da aplicação a pedidos HTTP que simulam a interação do utilizador com a aplicação através da sua interface gráfica. Os testes funcionais da aplicação são divididos em duas partes: testes a pedidos do tipo *GET* e testes a pedidos do tipo *POST*. Como resultado aos pedidos *GET* espera-se uma resposta HTTP com o código 200 que representa a obtenção de uma resposta da aplicação. No caso dos pedidos do tipo *POST* como resposta espera-se obter uma estrutura JSON que contém os campos necessários para a correta apresentação da página.

A divisão dos testes nestes dois tipos de pedidos permite cobrir algumas das funcionalidades da aplicação que são suportadas pelo *Route Web*. As funcionalidades que se cobrem nestes testes são todas aquelas que são necessárias para garantir a correta apresentação de cada página. Por vezes, quando um utilizador acede a uma página através de um pedido HTTP *GET*, em seguida é despoletado um ou mais pedidos HTTP *POST* de forma a carregar informação restante necessária. São esses pedidos que aqui são testados: 11 do tipo *GET* e 3 do tipo *POST*. Para se efetuarem estes testes criou-se uma classe de PHP como a apresentada no exemplo de código seguinte.

```
<?php

namespace Tests\Feature;

use Tests\TestCase;
use Illuminate\Foundation\Testing\WithoutMiddleware;

class WebTest extends TestCase
{
      // Teste a pedidos GET
      public function test_get_dashboard(){
            $this->get('/dashboard')->assertStatus(200);
      }
```

```
    // Teste a pedidos POST
    public function test_post_dashboard(){
        $res = $this->call('POST', '/dashboard', [
            'maquina' => 1,
            'inicio'  => '2017-07-20 00:00:00',
            'fim'     => '2017-08-18 23:59:59'
        ])->assertJsonStructure([
            'processos',
            'maquina'
        ]);
    }
    //…
```

No caso do exemplo apresentado, é feito um pedido do tipo *POST* para a obtenção dos dados para completar o gráfico da máquina 1. Como resultado deve-se obter uma estrutura que contém dados dos processos e informação da máquina.

A Figura 26 apresenta o resultado dos testes criados através da *framework* de testes *PHPUnit*.

```
PHPUnit 5.7.19 by Sebastian Bergmann and contributors.

..............                                            14 / 14 (100%)

Time: 2.85 seconds, Memory: 27.50MB

OK (14 tests, 14 assertions)
```

Figura 26 Resultados testes funcionais da aplicação *Web*.

7.2. SERVIÇOS *WEB* E BASE DE DADOS

Para testar os serviços *Web* e a base de dados, recorreu-se à *framework* de testes *PHPUnit* criando uma classe de testes que inclui funções para realizar operações CRUD dos diversos serviços *Web* desenvolvidos. Desta forma, é possível testar tanto o funcionamento dos serviços *Web* como a manipulação de dados na BD. Como resultado espera-se uma resposta HTTP cujo conteúdo terá um número inteiro maior do que zero que representa o número de identificação do registo manipulado na base de dados. Este valor pode ainda representar o número de manipulações registos alterados, como no caso da operação DELETE.

Foram criados vários serviços *Web* para realizar operações CRUD e um responsável por receber os dados do *Data Collection*. Para testar estes serviços e as respetivas funcionalidades de manipulação dos dados da BD, criou-se uma classe de PHP em que cada função é responsável por efetuar um teste. Foram realizados no total 13 testes. De acordo com este número é possível perceber que não se testam diretamente todas as operações CRUD dos diferentes serviços. Para tal seria necessário haver 32 testes. Isto deve-se ao facto de haver uma dependência entre as tabelas e, dessa forma, uma dependência entre alguns serviços *Web*. Por exemplo, quando é criado um registo de uma máquina é automaticamente criado um registo de validação da mesma.

Como apresentado no excerto de código seguinte, para testar cada serviço de operações CRUD, realizaram-se os testes na ordem do próprio acrónimo, de forma a criar um registo de teste, lê-lo, fazer uma atualização dos dados e por fim apagá-lo de forma a não ficar registado na base de dados. Neste caso são efetuados os testes às tabelas Máquinas, Validação de Máquinas, Configurações de Programas e Validação de Configurações.

```
<?php

namespace Tests\Feature;

use Tests\TestCase;
use Illuminate\Foundation\Testing\WithoutMiddleware;
use App\MaquinasModel;
use App\ValidacaoMaquinasModel;

class CrudTest extends TestCase
{
      public function test_maquina_create(){
            $res = $this->call('POST', '/api/maquinas, [
                  'maquina' => factory(\App\MaquinasModel::class)
                              ->make()
                  ]);
            if($res->content() == 0)
                  $this->assertTrue(false);
            else
                  $this->assertTrue(true);
      }

      public function test_maquina_read(){
            $id = MaquinasModel::where('exemplo', 'test')->get();

            $this->call('GET', '/api/maquinas/'.$id)
                  ->assertJsonStructure([‘exemplo’]);

            return $id;
      }
```

```
    /** @depends test_maquina_read */
    public function test_maquina_update($id){
        $res = $this->call(
                    'PUT',
                    '/api/maquinas/' . $id,
                [
                'maquinas' => factory(\App\MaquinasModel::class)
                            ->make(['exemplo' => 'test2'])
                ]);

        if($res->content() == 0)
            $this->assertTrue(false);
        else
            $this->assertTrue(true);
    }

    /** @depends test_maquina_read */
    public function test_maquina_delete($id){
        $res = $this->call('DELETE', '/api/maquinas/'. $id);

        if($res->content() == 0)
            $this->assertTrue(false);
        else
            $this->assertTrue(true);
    }

    //…
```

No exemplo de código apresentado, utiliza-se a função *test_maquina_create* para criar um novo registo de uma máquina. Como resultado da criação do registo espera-se que seja devolvido um número inteiro diferente de zero que representa a posição do registo efetuado na base de dados. De seguida na função *test_maquina_read* é feita a leitura do registo efetuado antes. Neste caso espera-se que seja retornada uma estrutura JSON com a informação da máquina e, por isso, verifica-se se na estrutura devolvida se encontra o campo "exemplo". Na função *test_maquina_update* é feita a atualização do campo "exemplo" para passar a conter "test2". No caso da operação *UPDATE* espera-se que o resultado seja um valor inteiro maior do que zero, significando a quantidade de registos alterados. Por fim apaga-se o registo efetuado recorrendo à função *test_maquina_delete*. À semelhança da operação *UPDATE* espera-se que seja retornado um valor cujo significado é a quantidade de registos manipulados que, neste caso, diz respeito à quantidade de registos apagados.

No caso do serviço do *Data Collection*, criou-se um registo de um processo de uma das máquinas para efetuar o teste. Correndo a classe com o PHPUnit obtém-se o resultado apresentado na Figura 27.

Efetuados os testes, resultam 13 confirmações de sucesso, tendo sido executados em 1.73 segundos.

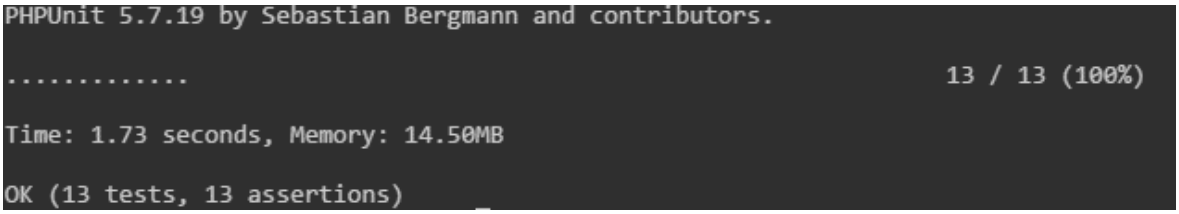


Figura 27 Resultado dos testes dos serviços *Web* e base de dados.

7.3. *DATA COLLECTION*

Os principais objetivos desta componente são recolher os dados das máquinas e enviá-los para o servidor. No capítulo 3 foram apresentadas as especificações e nelas referido que antes de enviar os dados para o servidor, a componente *Data Collection* deveria formatar os dados para o formato JSON. Em seguida descrevem-se os testes realizados onde se pretende avaliar organização da estrutura de dados enviada pelo *Data Collection* para o servidor.

Para efetuar este teste acrescentou-se aos serviços *Web* de recolha das informações enviadas pelo *Data Collection* código de forma a avaliar o conteúdo da mensagem. Esta avaliação permite não só verificar a presença de todos os campos necessários, mas também se estes se encontram preenchidos. Recebidos os dados e verificada a estrutura, o resultado é guardado num ficheiro de log com os possíveis seguintes resultados: PROCESSO_OK, PROCESSO_NOT_OK, CONFIG_OK e CONFIG_NOT_OK. É possível de perceber que os resultados com a palavra "processo" dizem respeito à receção de dados de processos e que os resultados com a palavra "config" dizem respeito à receção de dados de configurações de programas. Quando a estrutura se encontra corretamente formatada em JSON e com os campos todos preenchidos obtém-se o resultado OK, caso contrário o resultado é NOT_OK.

Este teste inclui a recolha de dados durante dois dias consecutivos de trabalho e, por isso, a análise apresentada pela Figura 28 é com base numa amostra de 140 registos.

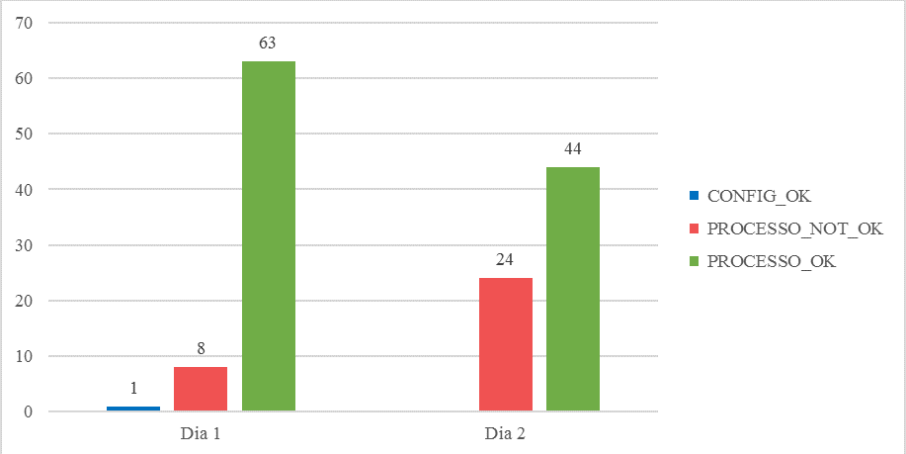


Figura 28 Resultados dos testes do *Data Collection*

Analisando o gráfico com os resultados é possível perceber que apenas uma configuração de programa foi enviada para o servidor tendo passado no teste com sucesso uma vez que não há a presença de resultados CONFIG_NOT_OK. Por outro lado, no que diz respeito aos dados relativos aos processos das máquinas, em 139 registos enviados obtiveram-se 32 em que ou não continham campos ou estes não estavam preenchidos. Isto significa que destes 139 processos recolhidos pelo *Data Collection*, 23.02 % não serão guardados na base de dados devido a falhas desta componente.

Assim, verifica-se que o *Data Collection* recolhe, estrutura e envia os dados para o servidor como é a sua função. No entanto é uma componente que apresenta falhas na recolha e estruturação dos dados, tornando-se vital uma análise aprofundada para se evitar a falta de informação posteriormente na aplicação *Web* desenvolvida.

7.4. SÍNTESE

Neste capítulo apresentaram-se alguns testes efetuados uma vez que há uma infinidade de testes possíveis de se realizarem. Assim, foram feitos alguns testes desde a camada de interação com os utilizadores até ao programa que faz a recolha automática dos dados para suporte do sistema criado. Começou-se por efetuar um teste de usabilidade à aplicação *Web* recorrendo a diversos tipos de utilizadores de forma a analisar-se o seu grau de satisfação face à solução. Ainda ao nível da aplicação *Web* analisou-se a sua capacidade de adaptação a uma série de combinações de navegadores *Web* e dispositivos. A seguir efetuaram-se testes funcionais de forma a avaliar a reação da aplicação a pedidos HTTP que simulam a interação do utilizador com a aplicação *Web*. Ao nível dos serviços Web e bases de dados foram desenvolvidos um conjunto de testes para avaliar diversas operações CRUD. Ao nível do *Data Collection* testou-se de uma forma geral as suas funções através da análise da estrutura de dados que este envia para o servidor.

Apesar da pequena amostra, os testes com os utilizadores permitem perceber se a aplicação vai não só de encontro às especificações, mas também ao que é esperado pelos utilizadores. Assim, recorrendo a um questionário que tem por base o CSUQ, obteve-se uma pontuação média de 9,44 em 10, permitindo concluir que os vários grupos de utilizadores estão satisfeitos com o desempenho da aplicação. Os testes de compatibilidade permitem analisar se a aplicação se comporta bem tanto no formato de um monitor de um computador

como em ecrãs de *tablets* ou *smartphones*. A aplicação foi testada nestes três ambientes diferentes tendo apresentado uma boa adaptação, mas algumas falhas quando os navegadores eram utilizados em formato de *tablet*. Os testes funcionais foram desenvolvidos para testar os pedidos HTTP que são gerados através da interação dos utilizadores com a interface gráfica. Dessa forma, testaram-se os pedidos que são necessários para garantir a correta apresentação das diversas páginas, obtendo-se resultados positivos. Neste caso, os resultados obtidos poderiam ser negativos caso houvesse alguma falha na comunicação entre a componente de cliente da aplicação e a componente servidora ou caso houvesse uma tentativa de acesso a dados não registados na base de dados.

Os serviços *Web* e a BD foram testados recorrendo a testes funcionais, semelhantes aos utilizados na aplicação *Web*. Neste caso o objetivo é fazer operações na base de dados recorrendo aos serviços criados em CRUD e ao serviço responsável por receber os dados do *Data Colletion*. Dessa forma, cobriram-se 13 operações obtendo resultados positivos em todas. À semelhança dos testes feitos para a aplicação *Web*, há sempre a possibilidade de ocorrerem quebras na comunicação entre os dois pontos da conexão levando à falha no registo dos dados.

No caso do *Data Collection*, uma vez que esta componente tem como objetivos recolher, estruturar e enviar os dados das máquinas de pré programação para o servidor, avaliou-se a estrutura de dados enviada. Dessa forma, os testes a este programa pretendem avaliar se a estrutura por ele enviada apresenta não só a presença de todos os campos necessários, mas também se estes campos se encontram preenchidos. Isto permitiu perceber que há uma percentagem de dados que não são corretamente enviados para servidor pelo que há uma necessidade de aprofundar estas falhas de forma a evitar a falta de dados importantes na aplicação *Web*.

8. CONCLUSÃO

Este documento apresenta o sistema desenvolvido como solução ao problema proposto pela Bosch, desde um enquadramento teórico até à especificação do sistema seguido do seu desenvolvimento. Começa-se por abordar de uma forma abrangente o conceito de aplicações distribuídas e nelas teorias, modelos e arquiteturas existentes. Com base nestes conceitos é definida uma especificação que servirá de base para o desenvolvimento da solução pretendida.

A Bosch, como uma referência de desenvolvimento tecnológico a nível mundial, procura constantemente formas de melhorar, facilitar e acelerar os seus processos com o objetivo de se manter na frente do mercado. O desenvolvimento de sistemas e ferramentas que permitam que os seus trabalhadores possam realizar as suas atividades de forma mais eficiente e eficaz é um dos principais fatores que a Bosch considera para que os seus objetivos sejam atingidos. Desta forma, o sistema desenvolvido apresenta-se como uma grande melhoria do processo até agora utilizado. O sistema desenvolvido permitiu especificar e criar uma plataforma capaz de ser acedida via *Web*, dando acesso a toda a informação relevante relacionada com a pré programação de IC da Bosch em Braga.

O desenvolvimento do sistema começou na criação de um programa responsável por fazer a recolha dos dados relevantes presentes nas máquinas de pré programação. O envio

destes dados para uma BD permitiu manter a informação centralizada e acessível a partir de um único local. Assim, a correta análise e implementação das tabelas necessárias da base de dados levou a que não só se garantisse o correto funcionamento do desenvolvimento seguinte, mas também que se tenha disponibilizado uma série de importantes recursos que futuramente poderão ser utilizados pela empresa. No que diz respeito à aplicação *Web*, como é descrito neste documento, a utilização da *framework Laravel* permitiu criar uma plataforma organizada, estável e escalável. No entanto, recorrendo à *framework*, a aplicação apresenta um tamanho de aproximadamente 80 MB, o que a torna pesada.

É possível perceber ao longo deste documento que há um fator muito importante que é tido em consideração e que é a base do correto funcionamento do sistema apresentado: a comunicação. A correta definição das arquiteturas e tecnologias utilizadas para garantir a comunicação entre os diferentes componentes do sistema é aquilo que o torna funcional. Recorrendo à tecnologia SQLite para a implementação da BD, levou à definição de uma arquitetura de duas camadas para o desenvolvimento da aplicação distribuída, permitindo a utilização de apenas um servidor para alojar a aplicação. A utilização de tecnologias baseadas em *Web* permite que os recursos do sistema desenvolvido se tornem acessíveis através do maior sistema distribuído existente, a *Internet*, tornando a sua utilização comum à maioria dos seus utilizadores.

O desenvolvimento deste projeto permitiu à empresa economizar uma quantia significativa não necessitando desta forma de adquirir um sistema semelhante posteriormente apresentado por um dos seus fornecedores.

8.1. TRABALHO EFETUADO

No capítulo 3 foram definidas no total vinte e uma especificações para o desenvolvimento deste sistema. Dois dos pontos definidos foram considerados como implementações extra, tendo sido apontadas já como trabalho futuro que, caso fosse possível, se implementaria juntamente com as restantes especificações.

De acordo com as especificações definidas, apenas a funcionalidade extra de recolha da informação dos programadores não foi desenvolvida. Isto leva a que tenham sido desenvolvidas 95,24 % das especificações iniciais totais, obtendo-se 88,89 % das

especificações para o *Data Collection* e 100 % das especificações definidas para o Servidor, Base de Dados, Serviços *Web* e Aplicação *Web*. Assim, o trabalho efetuado resume-se aos seguintes itens:

- Estudo do sistema existente;
- Levantamento de requisitos das máquinas de pré programação e de *software*;
- Análise das ferramentas e *frameworks* que mais se adequam ao sistema;
- Especificação do sistema;
- Desenvolvimento de um programa para recolher dados dos processos e das configurações dos programas das máquinas;
- Implementação de uma base de dados de forma a manter a informação centralizada;
- Desenvolvimento de um conjunto de serviços *Web* de forma guardar, ler e alterar os dados da BD;
- Desenvolvimento de uma aplicação *Web* de forma a permitir aos utilizadores aceder à informação das máquinas de pré programação.

8.2. TRABALHO FUTURO

Ao longo do desenvolvimento do sistema apresentado neste documento, foram surgindo novas ideias de funcionalidades ou melhorias a implementar no sistema. Entre todas as ideias de trabalhos futuros salientam-se os apresentados em seguida:

- Otimização das *querys* à BD de forma a melhorar o fluxo do sistema;
- Apresentação de um indicador do tempo de funcionamento das máquinas;
- Implementação da aplicação distribuída num servidor central da Bosch;
- Aplicação do sistema desenvolvido na fábrica da Bosch localizada na Malásia;

- Integração do sistema numa equipa de trabalho para suporte, manutenção e desenvolvimento do sistema nas diversas localizações.

Referências Documentais

[1] "Bosch Site," [Online]. Available: http://www.bosch.pt/pt/pt/our_company_10/our-company-lp.html. [Acedido em Fevereiro 2017].

[2] "Bosch Site - Divisões," [Online]. Available: http://www.bosch.pt/pt/pt/our_company_10/business_sectors_and_divisions_10/car_multimedia_7/car-multimedia.html. [Acedido em Fevereiro 2017].

[3] A. S. Tanenbaum e M. Van Steen, Distributed Systems - Principles and Paradigms, 2nd ed., Amesterdão: Pearson Education Inc, 2006.

[4] A. S. Tanenbaum, Computer Networks, 3rd ed., 1996.

[5] "Departamento de Engenharia Informática - Arquiteturas," Instituto Superior de Engenharia do Porto, [Online]. Available: https://www.dei.isep.ipp.pt/~andre/doc/arquitecturas.html. [Acedido em Maio 2017].

[6] M. B. C. N. Malheiro, "Desenvolvimento de Aplicações Web," Instituto Superior de Engenharia do Porto, Porto, 2002.

[7] "MSDN - Physical Tiers and Deployment," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/ee658120.aspx. [Acedido em Junho 2017].

[8] "ECMA - 404," Outubro 2013. [Online]. Available: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf. [Acedido em Maio 2017].

[9] "JSON," JSON ORG, [Online]. Available: https://www.json.org/. [Acedido em Maio 2017].

[10] "Laravel Documentation - 5.4," Laravel, [Online]. Available: https://laravel.com/docs/5.4. [Acedido em Maio 2017].

[11] "Understanding SOAP and REST Basics And Differences," SmartBear Blog, 8 Janeiro 2013. [Online]. Available: https://blog.smartbear.com/apis/understanding-soap-and-rest-basics/. [Acedido em Maio 2017].

[12] "RFC2616 - HTTP/1.1 Methods," Internet Engineering Task Force, Junho 1999. [Online]. Available: https://tools.ietf.org/html/rfc2616#page-36. [Acedido em Maio 2017].

[13] "W3C - HTML5," World Wide Web Consortium, [Online]. Available: https://www.w3.org/TR/html5. [Acedido em Maio 2017].

[14] "Mozilla Developer - HTML5," Mozilla Foundation, [Online]. Available: https://developer.mozilla.org/pt-PT/docs/Web/HTML/HTML5. [Acedido em Maio 2017].

[15] “W3C - CSS,” World Wide Web Consortium, [Online]. Available: https://www.w3.org/Style/CSS/Overview.en.html. [Acedido em Maio 2017].

[16] “Mozilla Developer - CSS3,” Mozilla Foundation, [Online]. Available: https://developer.mozilla.org/pt-PT/docs/Web/CSS. [Acedido em Maio 2017].

[17] “ECMA 262,” ECMA International Organization, [Online]. Available: https://www.ecma-international.org/ecma-262/7.0/index.html. [Acedido em Maio 2017].

[18] “Mozilla Developer - JS,” Mozilla Foundation, [Online]. Available: https://developer.mozilla.org/pt-PT/docs/Web/JavaScript. [Acedido em Maio 2017].

[19] “PHP - What is PHP?,” The PHP Group, [Online]. Available: https://secure.php.net/manual/en/intro-whatis.php. [Acedido em Maio 2017].

[20] “SitePoint - Best PHP Framework Survey Results,” SitePoint, 2015. [Online]. Available: https://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/. [Acedido em Maio 2017].

[21] “11 Best PHP Frameworks for Modern Web Developers in 2017,” Coders Eye, 2017. [Online]. Available: https://coderseye.com/best-php-frameworks-for-web-developers/. [Acedido em 2017].

[22] “Symfony Documentation,” Symfony, [Online]. Available: https://symfony.com/doc/current/index.html. [Acedido em Maio 2017].

[23] “CakePHP Documentation,” Cake Software Foundation, Inc, [Online]. Available: https://api.cakephp.org/3.4/. [Acedido em Maio 2017].

[24] “CodeIgniter Documentation,” EllisLab, [Online]. Available: https://codeigniter.com/user_guide. [Acedido em Maio 2017].

[25] “MSDN - MVC,” Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/ff649643.aspx. [Acedido em Maio 2017].

[26] “Mozilla Developer - MVC,” Mozilla Foundation, 2017. [Online]. Available: https://developer.mozilla.org/en-US/Apps/Fundamentals/Modern_web_app_architecture/MVC_architecture. [Acedido em Maio 2017].

[27] S. Borini, “Understanding Model-View-Controller,” [Online]. Available: https://github.com/stefanoborini/modelviewcontroller-src. [Acedido em Maio 2017].

[28] K. Gabis, “Github - Parson,” Github, [Online]. Available: https://github.com/kgabis/parson. [Acedido em Agosto 2017].

[29] “MSDN - Schtasks,” Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/bb736357(v=vs.85).aspx. [Acedido em Agosto 2017].

[30] “MSDN - Winsock,” Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/ms738545(v=vs.85).aspx. [Acedido em Agosto 2017].

[31] “Oracle Documentation - Sockets,” Oracle, [Online]. Available: https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html. [Acedido em Agosto 2017].

[32] N. Withanage, “Web Services Best Practices,” CodeProject, 22 Março 2005. [Online]. Available: http://www.codeproject.com/Articles/9916/Web-Services-Best-Practices. [Acedido em Agosto 2017].

[33] J. R. Lewis, “IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use,” IBM Corporation, Boca Raton, Florida.

[34] L. Damas, Linguagem C.

[35] “SQLite Site,” [Online]. Available: https://www.sqlite.org/whentouse.html. [Acedido em Maio 2017].

Anexo A. Questionário de Testes com Utilizadores

1. **Que tipo de utilizador é?** *
   Marcar apenas uma oval.

   - ◯ Operador
   - ◯ Utilizador de MFE3
   - ◯ Administrador

2. **No geral, estou satisfeito com a facilidade de utilização da aplicação.** *
   Marcar apenas uma oval.

	1	2	3	4	5	6	7	8	9	10	
Discordo	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Concordo

3. **É simples utilizar a aplicação.** *
   Marcar apenas uma oval.

	1	2	3	4	5	6	7	8	9	10	
Discordo	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Concordo

4. **Consigo completar o meu trabalho de forma eficiente utilizando esta aplicação.** *
   Marcar apenas uma oval.

	1	2	3	4	5	6	7	8	9	10	
Discordo	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Concordo

5. **Consigo completar o meu trabalho rapidamente utilizando esta aplicação.** *
   Marcar apenas uma oval.

	1	2	3	4	5	6	7	8	9	10	
Discordo	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Concordo

6. **A aplicação apresenta mensagens de erro que indicam como resolver problemas de forma clara.** *
   Marcar apenas uma oval.

	1	2	3	4	5	6	7	8	9	10	
Discordo	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Concordo

7. **Quando cometo um erro a utilizar a aplicação, recupero fácil e rapidamente.** *
Marcar apenas uma oval.

	1	2	3	4	5	6	7	8	9	10	
Discordo	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Concordo

8. **A informação apresentada é clara.** *
Marcar apenas uma oval.

	1	2	3	4	5	6	7	8	9	10	
Discordo	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Concordo

9. **É fácil encontrar a informação que necessito.** *
Marcar apenas uma oval.

	1	2	3	4	5	6	7	8	9	10	
Discordo	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Concordo

10. **A informação apresentada pela aplicação é fácil de entender.** *
Marcar apenas uma oval.

	1	2	3	4	5	6	7	8	9	10	
Discordo	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Concordo

11. **A informação é eficiente em ajudar-me a completar o meu trabalho.** *
Marcar apenas uma oval.

	1	2	3	4	5	6	7	8	9	10	
Discordo	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Concordo

12. **A organização da informação é no design da aplicação é clara.** *
Marcar apenas uma oval.

	1	2	3	4	5	6	7	8	9	10	
Discordo	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Concordo

13. **O design da aplicação é apelativo.** *
Marcar apenas uma oval.

	1	2	3	4	5	6	7	8	9	10	
Discordo	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Concordo

14. **A aplicação tem todas as funcionalidades e capacidades que esperava ter.** *
Marcar apenas uma oval.

	1	2	3	4	5	6	7	8	9	10	
Discordo	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Concordo

15. **No geral, estou satisfeito com a aplicação.** *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	8	9	10	
Discordo	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Concordo